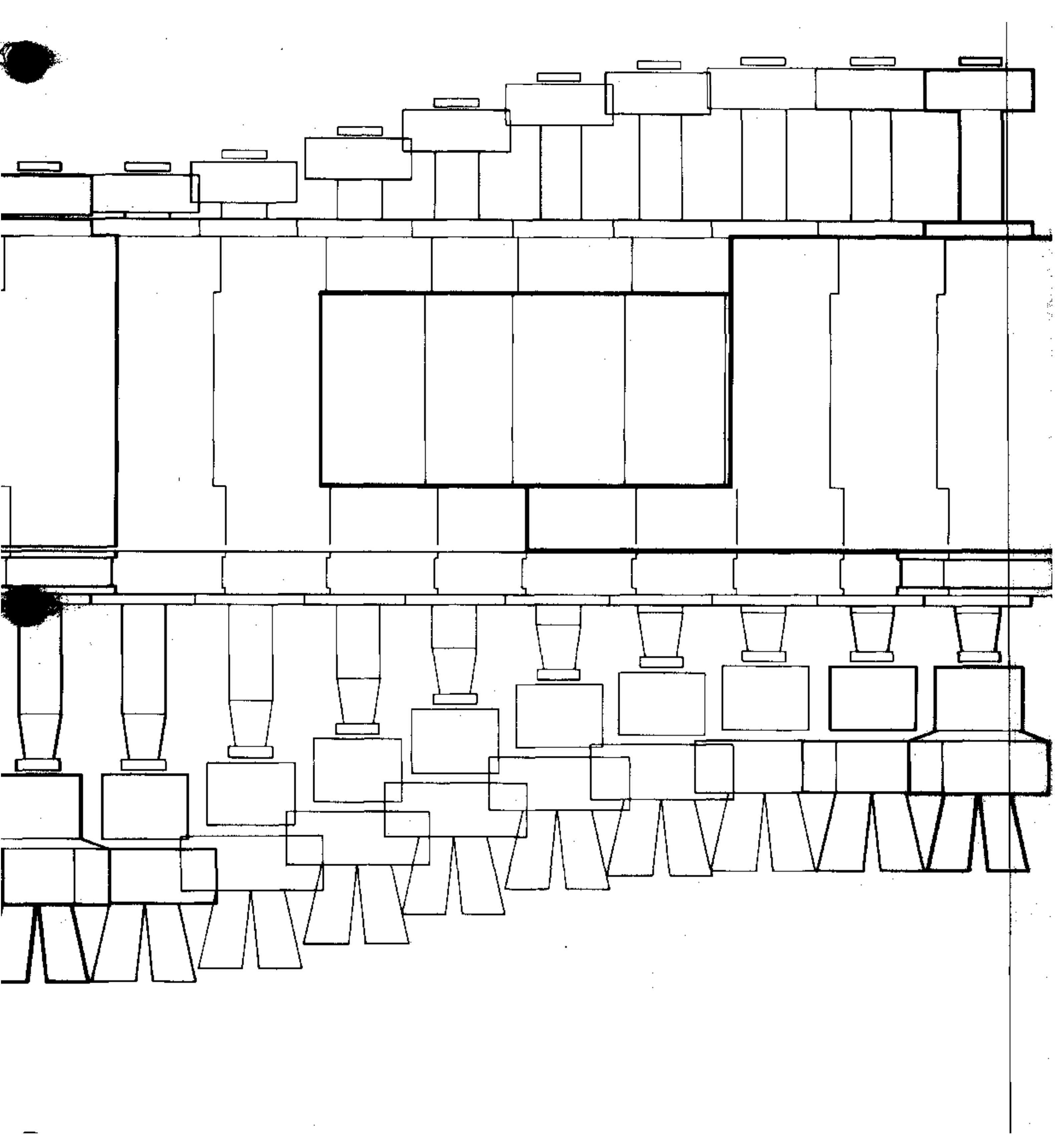
IBM Manufacturing Systr
Software Library



AML/Entry Version 4 User's Guide
(Second Edition)

Warning: This equipment generates, uses, and can radiate radio frequency energy which may cause interference to radio communications. Operation of this equipment in residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Second Edition (August 1985)

Use this publication only for the purpose stated in the Preface.

The following paragraph does not apply to the United Kingdom or any country where provisions are inconsistent with local law:

International Business Machines Corporation provides this publication "as is" without warranty of any kind, either expressed or implied, including, but not limited to the implied warranties of merchantability or fitness for a particular purpose.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein: such changes will be incorporated in subsequent revisions or in Technical News Letters. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below. Requests for copies of this publication should be made to your IBM representative or IBM branch office serving your locality.

The following paragraph applies only to the United States:

A reader's comments form is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, RS Information Development, Department 9C9, P.O. Box 1328, Boca Raton, Florida, 33432. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatsoever.

Copyright International Business Machines Corporation 1985

SAFETY NOT ICES

Safety precautions should always be observed by all personnel operating or servicing the IBM Manufacturing System. As with any electromechanical machine, unpredictable failures can occur while the system is in operation. Since the manipulator arm moves with such force and speed, serious injury could result from failure to observe caution whenever the work area is penetrated. Power to the manipulator must **always** be removed first. Keep this in mind when performing any maintenance or service on the system. Also:

- Ensure compliance to all local and national safety codes for the installation and operation of the system.
- Observe power and grounding instructions.
- The system must not be installed in an explosive atmosphere.
- Observe safe access routes to and from the system.
- Consider installing intrusion devices or safety mats around the manipulator to drop power if the work area is penetrated.
- Utilize signs around the system when servicing it to alert others to potential hazards.
- Consider installing additional emergency-off switches for feeders and other fixtures.
- Stay out of the manipulator work area when power is on. The manipulator arm moves rapidly with a lot of force.
- Always wear safety glasses around the manipulator.
- Remove watches and jewelry when servicing the system.
- Use the **Stop** pushbutton on the control panel to stop the manipulator in emergencies.
- **Always check the work area for adequate clearance before applying power. Be absolutely sure no one is in the manipulator work area.**
- All personnel working with the system must have (as a minimum) instructions on:
  - Safety devices for the system.
  - Use of safety devices. Safety procedures should be practiced to ensure familiarity.
- Fire extinguishers must be located within easy access.

PREFACE

The First Edition of AML/Entry Version 4 User's Guide contained new and updated information to be used in conjunction with the First Edition of AML/Entry Version 3 User's Guide. This publication is the Second Edition of AML/Entry Version 4 User's Guide. It contains all the information needed to use it as a stand-alone document. This publication also discusses the new features of AML/Entry Version 4.1. The new features include:

- Mathematical operations on counters (see "Expressions" on page 4-48).
- Built-in functions (see "Built-in Arithmetic Functions" on page 4-50).
- Support of the 7545 with RPQ R00107 (Extended Reach and Symmetrical Workspace).
- Enhancements to XREF (see "XREF Program" on page 2-34 and "XREF Program" on page 4-89).
- An improved version of COMSAMPL, entitled COMAID (see "COMAID" on page 8-14).
- A new host-end communications tool, entitled AMLECOMM, which facilitates the IBM Personal Computer or IBM Industrial Computer based cell control (see "AMLECOMM" on page 8-30).

Existing AML/Entry Version 4 applications are entirely upward compatible to AML/Entry Version 4.1. Existing AML/Entry Version 4 applications may be compiled and downloaded with the AML/Entry Version 4.1 system without requiring any modifications. Compiled programs now take between 3% and 7% more space than the programs compiled with the Version 4.0 compiler. The execution speed is virtually the same.

RELATED DOCUMENTS

- The IBM Manufacturing System Site Preparation manual for your system. This document contains information about preparing the site for the application.
- The IBM Manufacturing System Maintenance Information manual for your system. This document provides instructions for the setup and repair of the manufacturing systems.
- The IBM Manufacturing System Specifications Guide, 8577126. This document contains a description of the hardware and hardware specifications.

Review the system description and specifications to determine the capability of the system before you plan your application.

CONTENTS

Chapter 1. Introduction 1-1
How To Use This Manual 1-1

Chapter 2. Getting Started on the IBM Personal Computer 2-1
Environmental Consideration for the IBM Personal Computer 2-2
Minimum IBM Personal Computer Requirements 2-2
Configuration of the IBM Personal Computer 2-3
  Display 2-3
  Keyboard 2-3
  System Unit 2-4
  Printer 2-4
  Communications 2-4
The Display 2-5
  Contrast and Brightness Controls 2-5
  Cursor 2-5
The Keyboard 2-6
  Keys 2-6
    Enter Key 2-6
    Cursor Keys 2-6
    Control/Alternate/Delete Keys 2-7
The System Unit 2-8
  On/Off Power Switch 2-8
  In-Use Light 2-8
  Drive Door 2-8
The Printer 2-9
  On/Off Power Switch 2-9
  Control Switches 2-9
  Control Lights 2-9
Creating Self-Booting AML/Entry diskettes 2-10
  How to Insert Diskettes 2-11
  Copying DOS and AML/Entry Shipped Diskettes on to Work Diskettes 2-12
  Copying DOS and AML/Entry Diskettes on to a Fixed Disk 2-15
  Making Backup Copies of the AML/Entry Work Diskettes 2-17
  Some Words About Files 2-17
  Some Words about Diskettes 2-19
  How to Handle/Store Diskettes 2-20
Loading the AML/Entry Self-Booting Work Diskette 2-21
  Method 1 (Power Switch OFF) 2-21
  Method 2 (System RESET) 2-22
Using the AML/Entry Programming System Menu 2-23
  Option 0 (Return to DOS) 2-24
  Option 1 (Edit/Teach a Program) 2-25
  Option 2 (Compile a Program) 2-26
    Load File (.ASC) 2-26
    Compiler Phase Messages 2-26
    Displayed Information 2-26
    Compiler Errors 2-26
    Listing File (.LST) 2-27
    Symbol File (.SYM) 2-27

Option 3 (Load a Program to the Controller) 2-28
Option 4 (Unload a Controller Program) 2-29
Option 5 (Set System Configuration) 2-30
Option 6 (Set Name and Options) 2-32
Option 7 (Communicate with controller) 2-33
Option 8 (Generate Cross Reference Listing) 2-34
XREF Program 2-34
DOS Batch Support 2-36
Invoking the Compiler 2-36
Loading/Unloading a Program to the Controller 2-36
Example Batch Program 2-37
AML/Entry Utility Programs 2-38

Chapter 3. Using the AML/Entry Editor 3-1
Full Screen Editing 3-1
File Specification Line 3-2
Primary Commands 3-2
Date 3-3
Message Display 3-3
Top of File/Bottom of File 3-3
Line Commands 3-3
Line Numbers 3-3
Function Key Settings 3-3
20-Line Program Window 3-3
Keyboard Usage for the Editor 3-4
Function Key Settings 3-4
Help Screens 3-4
Special Keys 3-6
Numeric Keypad 3-7
Control Keys 3-8
Setup for Editor Exercises 3-9
Getting to the Editor from the Main Menu 3-9
Exiting the Editor 3-10
Exiting the Editor for the First Time 3-10
Recalling the Practice Program 3-10
Exiting the Practice Program After You Name It 3-11
Information About Line Commands 3-11
Line Command Conflicts 3-12
Line Commands that Cross Screens 3-12
Using the Line command I (Insert) 3-13
Using the Line Commands D and DD (Delete) 3-15
Using the Line Commands M, MM, with A Or B (Move with After or Before) 3-18
Using Line Commands C, CC, with A or B (Copy with After or Before) 3-22
Using the Line Command R (Repeat) 3-26
Information About Primary Commands 3-30
Using the Primary Command FIND 3-32
Using the Primary Command CHANGE 3-36
Using the Primary Command LOCATE 3-42
Using the Primary Command SAVE 3-44
Using the Primary Command FILES 3-46
Using the Primary Command ? (Recall) 3-48
Using the Primary Command RENAME 3-50
Using the Primary Command GETFILE 3-52
Using the Primary Command PUTFILE 3-53

Using the Primary Command PRINT 3-54
Using the Primary Command DEL (Delete) 3-56
Using the Primary Command CANCEL 3-57
Using the Primary Command CAPS 3-58

Chapter 4. Learning the AML/Entry Language 4-1
A Structural Overview 4-2
Language Structure 4-2
Your Application Program in the Controller 4-2
Comments 4-2
Beginning and Ending Your Program 4-3
Line Number 4-4
Identifier 4-4
Definition Operator 4-5
Keyword/Command 4-5
Statement Delimiter 4-5
AML/Entry Reserved Words and Commands 4-6
AML/Entry Reserved Words 4-6
MOTION Commands 4-8
DELAY Command 4-8
DPMOVE Command 4-8
GETPART Command 4-9
GRASP Command 4-9
LEFT Command (Valid on 7545-800S Only) 4-9 .
LINEAR Command 4-11
PAYLOAD Command 4-11
PMOVE Command 4-12
RELEASE Command 4-12
RIGHT Command (Valid on 7545-800S Only) 4-13
XMOVE Command 4-13
ZMOVE Command 4-13
ZONE Command 4-13
Using Motion Statements 4-14
Using Move, Z-axis, Delay, and Gripper Commands 4-14
Using Linear, Speed, and Precise Motions 4-16
SENSOR Commands 4-17
CSTATUS Command 4-17
GUARDI Command 4-18
MSTATUS Command 4-19
NOGUARD Command 4-20
TESTI Command 4-20
WAITI Command 4-20
WHERE Command 4-21
WRITEO Command 4-21
Using Sensor Statements 4-21
FLOW-OF-CONTROL Commands 4-23
Labels 4-23
BRANCH Command 4-24
BREAKPOINT Command 4-24
ITERATE Statement 4-24
TESTI Command 4-25
Using Flow-of-Control Commands 4-26
Techniques to Simplify Programming 4-28
Multiple Statements on a Line 4-29
Declarations 4-30
Using Declarations 4-30

Constants 4-31
Declaring Constants 4-31
Local Constants 4-32
Global Constants 4-32
Global vs. Local Constants 4-32
Using Constants 4-32
Using Local Constants 4-33
Using Global Constants 4-34
Aggregate Constants 4-35
Using the ITERATE Command with aggregates 4-35
Variables 4-38
Declaring Variables 4-38
Counters 4-39
Round-Off Error 4-39
COUNTER Commands 4-41
DECR Command 4-41
INCR Command 4-41
SETC Command 4-41
COMPC Command 4-42
TESTC Command 4-43
Using Counter Statements 4-43
PT's Defined in Terms of Formals and/or Counters 4-44
Group 4-45
Indexing 4-46
Using Groups with 7545-800S 4-47
Expressions 4-48
Built-in Arithmetic Functions 4-50
Commands That Allow Expressions 4-54
Example Applications Using Expressions 4-55
Circular Motion 4-55
Treating DI As Integers 4-56
Determining the Row and Column of a Part in a Pallet 4-57
Compiler Directives 4-58
Using the Include Compiler Directive --%I 4-58
Using the Page Compiler Directive --%P 4-59
Subroutines 4-60
System Subroutines 4-60
User Subroutines 4-60
User Subroutines in the AML/Entry Program 4-61
Development of User Subroutines 4-61
Formal Parameters in Subroutines 4-63
Parameter Passing 4-63
Example of Subroutine with Formal Parameters 4-63
Restrictions on Parameters 4-64
Formal Parameter Names Restrictions 4-64
Actual Parameter Assignment Restrictions 4-64
Using Subroutines 4-65
Using Subroutines without Parameters 4-65
Rules for Calling Subroutines 4-66
Using Subroutines with Parameters 4-68
Using the ITERATE command to Repeat a Subroutine 4-70
Ownership and Multiple Name Occurrence 4-72
Additional Topics For Program Enhancement 4-74
Pallet 4-74
Pallet Description 4-74
PALLETIZING Commands 4-77

GETPART Command 4-77
NEXTPART Command 4-77
PREVPART Command 4-78
SETPART Command 4-78
TESTP Command 4-78
Using Palletizing Statements 4-79
Palletizing and Formal Parameters 4-79
Region 4-82
REGION Command 4-84
XMOVE Command 4-84
REGION Coordinate Generation 4-84
X and Y Coordinates 4-84
Z Coordinate 4-84
Roll Coordinate 4-85
Using REGIONS 4-85
Host Communications 4-87
GET Command 4-87
PUT Command 4-88
Variable Identification 4-89
XREF Program 4-89
Pallet and Region Listings 4-90

Chapter 5. Writing AML/Entry Programs 5-1
Good Program Structure 5-1
Writing a Simple AML/Entry Program 5-3
Sample Application 5-3
Application Program Comparison 5-4
Writing a Complex AML/Entry Program 5-7
Main Application Task 5-7
Printed Circuit Card 5-7
Manipulator Gripper 5-7
Component Feeders 5-7
Interaction with a Host Computer 5-8
Application Flow 5-9
Define Global Data Types 5-10
Taught Points 5-10
Digital Input and Digital Output 5-11
Constants 5-13
Variables 5-13
Define the Global Subroutines 5-14
Utility Subroutines 5-14
Movement Subroutines 5-17
Gripper Subroutines 5-18
Parts Handling Subroutines 5-19
Initialization Subroutine 5-22
The Main Subroutine 5-24

Chapter 6. Using the AML/Entry Teach Mode 6-1
Teach Mode 6-3
Function Keys 6-5
Special Keys 6-6
Read DI/DO Points 6-8
Initial Setting of Motion Parameters for Safety 6-8
Set Motion Parameters in Teach Mode 6-9
Exiting the Digital Output Control Utility in Teach 6-9
IBM Manufacturing System Teach Responses to Previous Conditions 6-10

Condition 1 (Manipulator Power Off) 6-10
Condition 2 (Manipulator Power On, Return Home Performed) 6-10
Condition 3 (Exit Teach) 6-11
Condition 4 (Exit Teach, Remain in Editor - Control Panel Move) 6-11
Condition 5 (Exit Teach, Remain in Editor after Manipulator Move) 6-13
Teach Mode Exercises 6-14
Power-Up Sequence for Teach Exercises 6-15
Coarse Movement 6-17
Precision Movement 6-18
Entering Known Coordinates 6-19
Return Point Value to Program (Recall) 6-21
Obtaining an Additional Point 6-22
Retrieving a Point from a Program 6-23
Controlling Digital Output (DO) from Teach 6-28
Changing Manipulator Arm Mode 6-29
Removing Power after Teach Exercises 6-30

Chapter 7. Operating the Manufacturing System 7-1

Controller 7-2
Power On/Off Switch/Circuit Breaker 7-2
Power On Light 7-2
Control Panel 7-3
System Power-Up Sequence 7-10
Manual and Automatic Stopping of the Manipulator 7-12
Power-Off SeqUence 7-13
Controller Storage Management 7-14
Compile and Load an Application Program 7-16
Bringing Up the AML/E Menu On a Standard PC 7-16
Bringing Up the AML/E Menu On a PC With a Fixed Disk 7-16
Bringing Up the AML/E Menu On a PC/AT 7-17
Testing Application Programs in Manual Mode 7-22
Manual Operation of the Manipulator 7-23
Manual Mode Control of Axis Motors, Z-Axis, and Gripper 7-24
Automatic Operation 7-24
Starting an Application Program in Automatic Mode 7-25
Resuming an Application Program from a Breakpoint 7-26
Clearing Error Conditions 7-27

Chapter 8. Communications 8-1

Communications Hardware Interface 8-2
Controller Communications Connector 8-3
Communication Startup Sequence 8-3
Communication Capabilities 8-4
Data Drive Mode 8-8
Controller States 8-8
Xon State 8-8
Xoff State 8-8
Waiting for Xon (Wxo) State 8-9
Xoff Time-Out (Xto) State 8-9
Transitions Between States 8-9
Simple State Transitions 8-9
Complex State Transitions 8-10
COMAID 8-14
P - Display the last 51 communication transactions 8-14
L Load a Program to the Controller 8-15

U - Unload Controller Partition 8-15
R - Transmit Read Command 8-15
X - Transmit Execute Command 8-16
D - Controller Initiated Communications 8-17
C - Control Executing Program 8-19
T - Transmit Teach Command 8-20
F - Execute a Command File 8-20
DOS Command Line Processing 8-21
Debugging AML/E Applications 8-23
Using Read Requests 8-23
Reading the Machine Status 8-23
Reading the Current Instruction Address 8-25
Reading Specific Program Variables 8-26
Reading Other Values 8-26
Using Control Requests 8-26
Suspending Program Execution 8-26
Restarting Program Execution 8-27
Executing Until the Next Terminator 8-27
Setting a Debug Breakpoint 8-27
Resetting the Controller 8-28
Changing Variable Values in Controller Memory 8-29
AMLECOMM 8-30
Introduction to the AMLECOMM System 8-30
The AMLECOMM System Files 8-30
Installation Procedure 8-31
Interpreter vs. Compiler 8-32
AMLECOMM Line Numbers 8-32
Using Compiler Basic 8-33
Using BASICA 8-33
Initialization and Configuration Parameters 8-34
Calling AMLECOMM 8-36
The Transaction Buffer - BUFFER.A$ 8-51

Appendix A. Command/Keyword Reference A-1
A A-2
ABS A-3
ATAN A-4
ATAN2 A-5
B A-7
BRANCH A-8
BREAKPOINT A-9
C A-11
CANCEL A-12
CAPS A-13
CC A-14
CHANGE A-15
COMPC A-17
COS A-19
COUNTER A-20
CSTATUS A-22
D A-24
DD A-25
DECR A-26
DEL A-28
DELAY A-29
DPMOVE A-30

END A-32
FILES A-33
FIND A-34
FROMPT A-35
GET A-36
GETFILE A-37
GETPART A-38
GRASP A-40
GROUP A-41
GUARDI A-43
I A-45
INCR A-46
ITERATE A-48
LEFT A-50
LINEAR A-51
LOCATE A-53
M A-54
MM A-55
MSTATUS A-56
NEW A-58
NEXTPART A-59
NOGUARD A-61
PALLET A-62
PAYLOAD A-65
PMOVE A-67
PREVPART A-68
PRINT A-70
PT A-71
PUT A-72
PUTFILE A-73
R A-75
REGION A-76
RELEASE A-79
RENAME A-80
RIGHT A-81
SAVE A-82
SETC A-83
SETPART A-85
SIN A-87
SQRT A-$8
STATIC A-89
SUBR A-90
TAN A-92
TESTC A-93
TESTI A-94
TESTP A-96
TRUNC A-98
WAITI A-99
WHERE A-100
WRITEO A-101
XMOVE A-102
ZMOVE A-104
ZONE A-105
? A-107
--%I A-108
--%P A-109

Appendix B. AML/Entry Messages B-1
Messages Without Numbers B-1
Numbered Messages B-47
AMLECOMM/COMAID Error Messages B-60
MSGCOM.TXT B-66

Appendix C. Values for the LINEAR Command C-1

Appendix D. Values for the PAYLOAD Command D-1
7545 Program Speed Values For PAYLOAD Command D-1
7545-800S Program Speed Values For PAYLOAD Command D-2
7547 Program Speed Values For PAYLOAD Command D-3

Appendix E. Speed/Weight Values Based on Z Position E-1
7545 Speed/Weight Relationship based on Z Position E-2
7545-800S Speed/Weight Relationship based on Z Position E-4
7547 Speed/Weight Relationship based on Z Position E-5

Appendix F. Communications Cable Wiring Diagrams F-1
Local RS-232-C Cable Wiring F-1
Local RS-422 Cable Wiring F-2

Appendix G. Configuration Parameters for AMLECOMM G-1

Appendix H. Advanced Communications H-1
Communications Hardware Interface H-2
  Controller Communications Connector H-3
Communications Protocol H-4
  Transactions H-4
  Identifiers H-5
  Records and Record format H-6
    Identifier H-6
    Byte Count H-6
    Data H-6
    Check Sum H-6
    Record Termination H-7
  Data Rules H-8
  Data Representation H-9
  Floating Point Examples H-10
    Powers Of Two Table H-14
  Data Line Control H-15
    Ack H-16
    Xoff H-16
    Xon H-17
    Eot H-17
    Nak H-17
    Nul H-18
  Communication Startup Sequence H-18
  Record Descriptions H-19
  R - (Read) Record H-19
    R 01 - Read Machine Statue H-20
    R 02 - Read Reject Status H-21
    R 03 - Read Micro-code Level and Machine Type H-22
    R 04 - Read Robot Parameter Table H-22
    R 08 - Read Current Instruction Address H-23

R 10 - Read DI/DO H-24
R 20 - Read All Program Variables H-25
R 40 - Read Current Position in Pulses H-25
R 80 - Read Specific Program Variables H-25
C (Control) - Records H-26
X (Execute) - Records H-28
N (Compiled Program) - Record H-29
E (End) - Record H-30
D (Data) - Record H-30
T (Teach) - Record H-31
Motion Parameters H-32
Motion Control H-32
Changing Digital Outputs H-33
P (Present Configuration) - Record H-33
Controller-Initiated Communications H-34
PUT Transaction H-34
GET Transaction H-35
DEBUG Transaction H-35
Typical Communications Sequences H-36
Manipulator Stop Cycle Sequence H-37
Application Startup Sequence H-38
Reason For Data Error H-40
Program Transmit Sequence H-41
Unload a Partition H-42
Put Transaction H-43
Get Transaction H-44
Read Transaction H-45
Read Instruction Address H-46
Debug Transaction H-47
Write Controller Data Transaction H-48
Example Application Sequence H-49

Index X-1

CHAPTER 1 . INTRODUCTION

A Manufacturing Language/Entry (AML/Entry) is a manufacturing systen programming language. It is used with an IBM Personal Computer or ar IBM Industrial Computer to write, compile, and load programs for the at 7545 and 7547 Manufacturing Systems, including the 7545 with RPQ R0010; (Extended Reach and Symmetrical Workspace).

The structure of this document orients you in A Manufacturing Language/Entry and then provides the information necessary to operate your manufacturing system. It also contains an overall command/keyword reference and the error messages that pertain to the language and to the systems.

Note: Throughout this document, the IBM 7545 Manufacturing System with RPQ R00107 (Extended Reach and Symmetrical Workspace) will be referred to as the "7545-800S". The software, however, refers to this system as the "7545-S". Both of these terms have been adopted as a matter of convenience.

HOW TO USE THIS MANUAL

This manual is designed to be used both as a teaching manual and a learning manual. The first chapters teach you how to use AML/Entry Version 4 to create programs for use by IBM Manufacturing Systems. These chapters are outlined below:

- Chapter 2, "Getting Started on the IBM Personal Computer," teaches you how to use the Personal Computer to create work diskettes and then to load the AML/Entry system.

- Chapter 3, "Using the AML/Entry Editor," is a series of exercises that teach you how to use the AML/Entry editor.

- Chapter 4, "Learning the AML/Entry Language," teaches you how to use AML/Entry Version 4. Chapter 4 begins by teaching you how to use basic AML/Entry commands. After you have learned these basic commands, the chapter proceeds to more complex subjects. This approach allows you to start using AML/Entry right away and then tc proceed to more difficult concepts at your own pace.

- Chapter 5, "Writing AML/Entry Programs," gives examples of AML/Entry Version 4 programs along with hints on how to write application programs that are easy to understand.

- Chapter 6, "Using the AML/Entry Teach Mode," teaches you how to use the Personal Computer to move the manipulator arm to points within the workspace. You are taught how to connect the Personal Computer to the manipulator and how to control the manipulator movement fron the Personal Computer.

- Chapter 7, "Operating the Manufacturing System," teaches you how to operate the manufacturing system and the different functions of the operator control panel.

- Chapter 8, "Communications," describes the communications interface between the IBM PC and the controller. The description is divided into two main sections: the first describes COMAID, the second describes AMLECOMM. Advanced communications topics are discussed in Appendix H, "Advanced Communications."

- Appendix A, "Command/Keyword Reference," is an alphabetical listing of the AML/Entry commands and keywords. A description of each command is given.

- Appendix B, "AML/Entry Messages," contains a description of each of the messages that you can receive when using AML/Entry Version 4.

  Appendices C through E contain manipulator specification information: LINEAR values, PAYLOAD values, and speed/weight values.

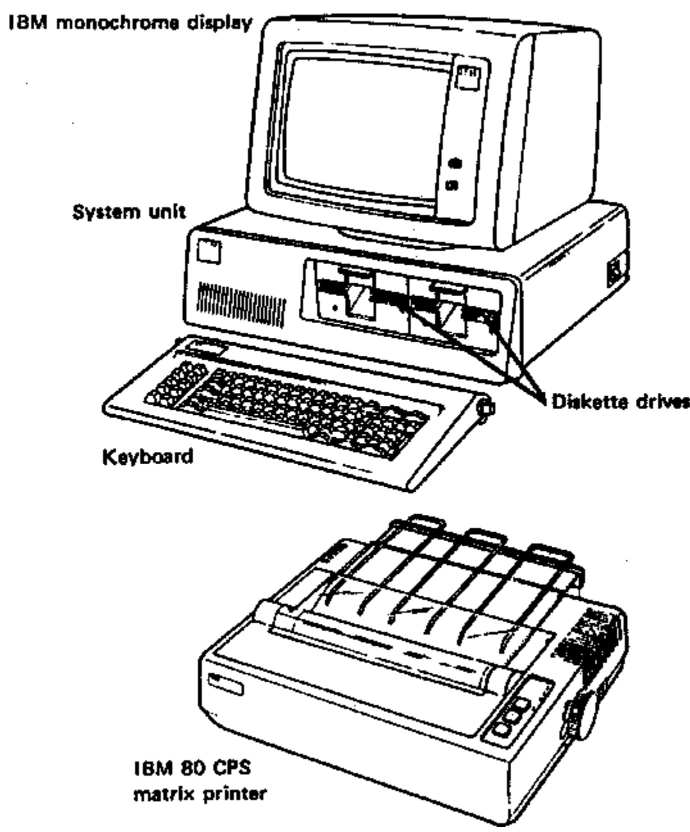
- Appendix F, "Communications Cable Wiring Diagrams," contains the diagrams for the two communications cables that can be used.

- Appendix G, "Configuration Parameters for AMLECOMM," contains a listing and description of the configuration parameters that AMLECOMM uses.

- Appendix H, "Advanced Communications," contains a description of the AML/Entry communications protocol. Example transactions are given to illustrate the protocol.

CHAPTER 2. GETTING STARTED ON THE IBM PERSONAL COMPUTER

This chapter describes how to get started on the IBM Personal Computer as applied to IBM Manufacturing Systems. Before you proceed in this chapter, you should become familiar with the Guide to Operations and the Disk Operating System (DOS) documentation provided with your IBM Personal Computer.



ENVIRONMENTAL CONSIDERATION FOR THE IBM PERSONAL COMPUTER

Refer to the documentation for your Personal Computer.

CAUTION

Operating your IBM Personal Computer in an environment outside the design parameters for extended periods can damage it.

IN-USE CONDITIONS

15.6-32.2 ° Celsius
60-90 ° Fahrenheit
8-80% Relative Humidity

STORAGE CONDITIONS

10-43 ° Celsius
50-110 ° Fahrenheit
8-80% Relative Humidity

MINIMUM IBM PERSONAL COMPUTER REQUIREMENTS

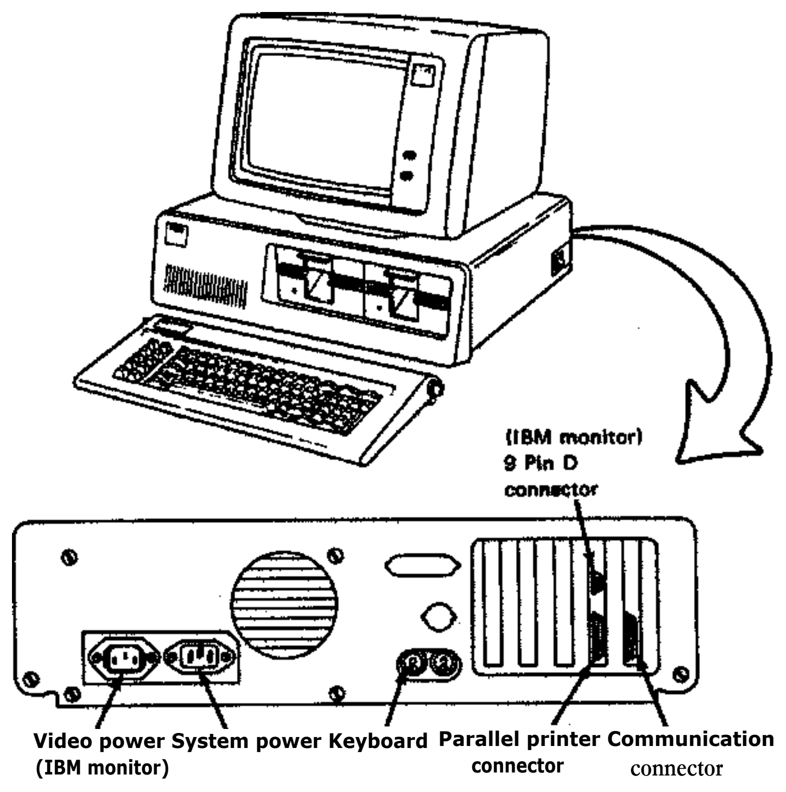
AML/Entry personal computer requirements are outlined below.

- A black and white, monochrome, or color display (80 column) with the correct adapter card.
- A minimum of 192 Kbytes of memory, 256 Kbytes of memory is recommended.
- An asynchronous communications adapter.
- The IBM Disk Operating System (DOS) Version 2.0 or higher and supporting documentation.
- Either two dual-sided diskette drives, one dual-sided diskette drive and one fixed-disk drive, or one high density diskette drive (PC AT).
- Spare diskettes required to save application programs created on the system.

A printer is recommended, but not required. The printer is useful for listing programs, and for printing error messages while debugging programs.

CONFIGURATION OF THE IBM PERSONAL COMPUTER

A typical connector arrangement for the IBM Personal Computer, when it is configured for IBM Manufacturing Systems, is shown below. The exact location of each connector may be different on your system.



The following is a brief explanation of the equipment that makes up an IBM Personal Computer system. Each piece of equipment will be discussed in detail following this section.

Display

The display, or monitor, is similar in structure to a television set. It is used to display information being sent to and stored in the system unit of the computer.

Keyboard

The keyboard is like a typewriter. It is the device used to send information to the system unit.

System Unit

The system unit is the main part of your computer setup. It processes information that is sent to it. The system can be tailored to your needs by the addition of various devices.

Printer

The IBM 80 CPS Matrix, IBM Color, or IBM 5533 Industrial Printers are optional peripherals which may be used to produce hardcopy outputs of your information and programs.

Communications

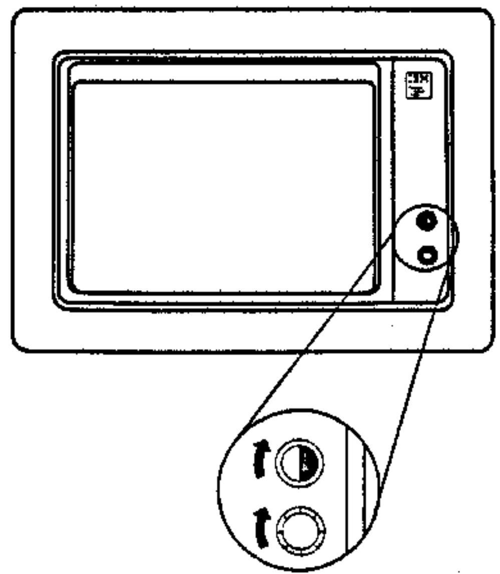
An asynchronous communications adapter is a tool used for communication between your Personal Computer and the manufacturing system. The main use of the communications adapter is for downloading programs and for controlling the manipulator during the teach mode of the AML/Entry language.

THE DISPLAY

The IBM display is similar to a television set. It can be monochrome or color. It is used with the AML/Entry System to display, on its screen, the information you send to the computer and information stored in the computer. Generally, it sits on top of the system unit for easier viewing.

Contrast and Brightness Controls

The IBM display has contrast and brightness controls which are located on the front, right side of the machine. When you first use the display, turn the controls fully clockwise and then adjust to your needs.

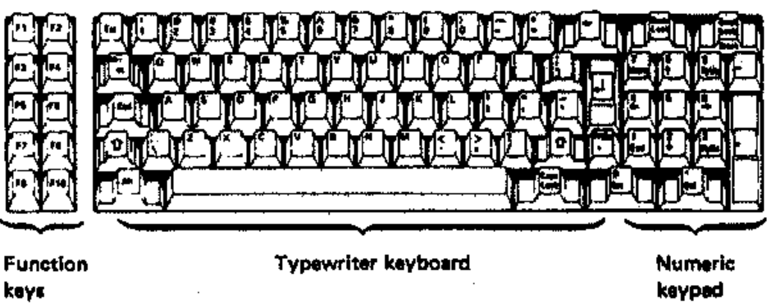


Cursor

The cursor is mentioned in many places of this document. The cursor appears on the screen of the IBM Personal Computer as a flashing underscore (_). The cursor lets you know where the next keyboard character typed is displayed.

THE KEYBOARD

As shown below, the IBM Personal Computer keyboard includes a numeric keypad, a typewriter keypad, and a function keypad



Keys

Keyboard keys are typematic. If you hold down a key, the key repeats as though it was repeatedly pressed.

The computer allows you to enter 15 characters ahead. This is useful during the time the computer is accessing a file on a diskette because it allows you to enter the next instruction without waiting for the screen prompt. The characters you enter are not displayed until the prompt is displayed.

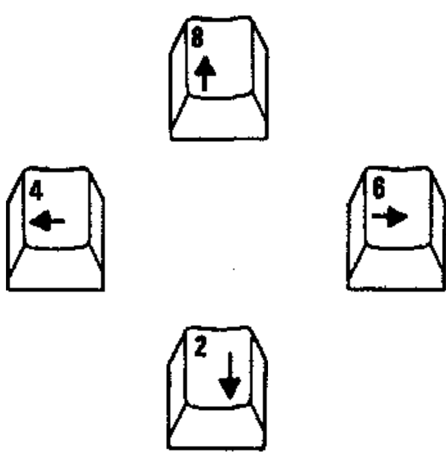
The function keys and the numeric keypad may have preassigned functions. These functions appear at the bottom of the display screen.

Enter Key

The <--J (enter) key, located on the keyboard, is the key most used in the procedures outlined in this document. It signals the computer that you have finished entering the line.

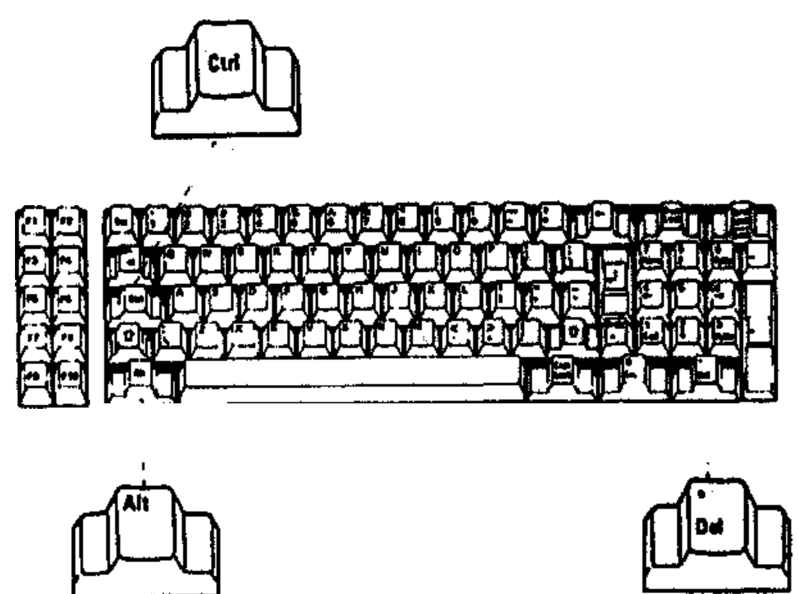
Cursor Keys

A set of four cursor keys, located on the numeric keypad, is used to move the cursor in the direction of the arrow shown on each key. You are able to type over an incorrect character after positioning the cursor underneath it. Cursor keys are shown in the following figure.



Control/Alternate/Delete Keys

The **CTRL, ALT,** and DEL keys, **located on** the keyboard, are used when loading a self-booting diskette with the power switch ON. Simultaneously pressing these three keys cause a system reset. System reset clears computer memory and reloads a self-booting diskette.

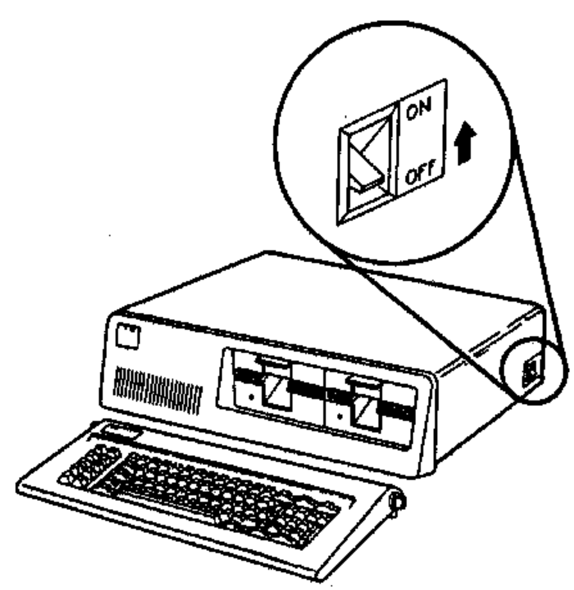


THE SYSTEM UNIT

The System Unit is the main part of your Personal Computer. A choice of optional circuit boards allow you to meet your needs. These circuit boards are plugged into the system unit's expansion slots. Your input to the System Unit is through the keyboard.

On/Off Power Switch

The on/off power switch for the system unit is located on the right side of the machine, towards the back. It is used to power the system unit on and off.



In-Use Light

An in-use light is located on the front panel of the disk or diskette drive in the front of the system unit. The light is on when the computer reads information from the drive.

Drive Door

The drive door is used to open and close the diskette drive. How to insert and remove diskettes is covered later in this chapter in the section called "How to Insert Diskettes".

THE PRINTER

The IBM 80 CPS Matrix, IBM Color, or IBM 5533 Industrial Printers are optional with the AML/Entry System. They are very useful for making hardcopies of program listings or other output.

On/Off Power Switch

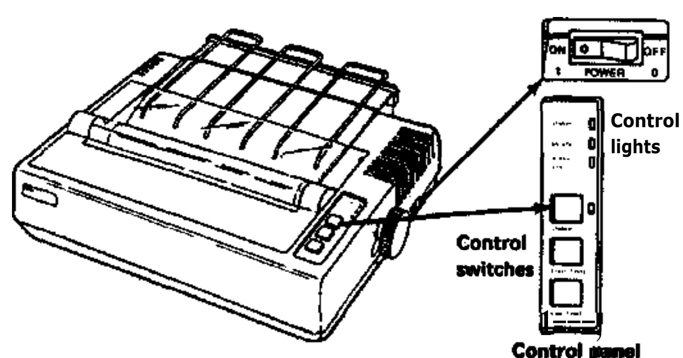
The on/off power switch is located on the bottom, right side of the printer (for the IBM 80 CPS Matrix Printer).

Control Switches

For the IBM 80 CPS Matrix printer, the control switches are located in the control panel on the top of the printer in the front right corner. There are three switches for line feed, form feed and online mode. There are similar switches for the IBM Color and 5533 Industrial printers.

Control Lights

For the IBM 80 CPS Matrix printer, the control lights are also located in the control panel on the top of the printer. These three lights indicate when the power is on, when the printer is ready, and if the paper supply is empty. There are similar lights for the IBM Color and 5533 Industrial printers.



CREATING SELF-BOOTING AML/ENTRY DISKETTES

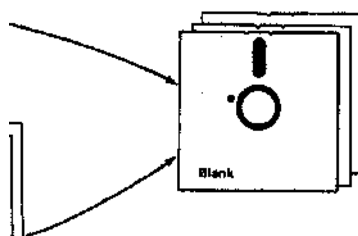
Self-booting AML/Entry diskettes eliminate the requirement that the Disk Operating System (DOS) diskette be used each time you want to work with the AML/Entry programs. The self-booting diskette allows you to insert the diskette and set the power switch to ON, or, with the power ON, reset the system.

> **Note:** If you are using an international DOS, you must create a national diskette, as shown in the DOS manual, before you create self-booting diskettes.

DOS is required to create the self-booting AML/Entry diskettes. DOS is an IBM program product that provides a standard method to store and access data on diskettes. DOS is not included with your shipped AML/Entry diskette.

Self-booting AML/Entry diskettes are created by copying DOS and the AML/Entry shipped diskettes on to work diskettes. (This procedure is discussed in a later section of this document.) Three blank diskettes are required for double-sided drive systems. Your original AML/Entry diskettes should be retained in a safe place in case something happens to your self-booting diskettes or to the files on the fixed-disk drive.

The copying of DOS and AML/Entry shipped diskettes on to work diskettes is accomplished by using AML/Entry's Autoinit program.



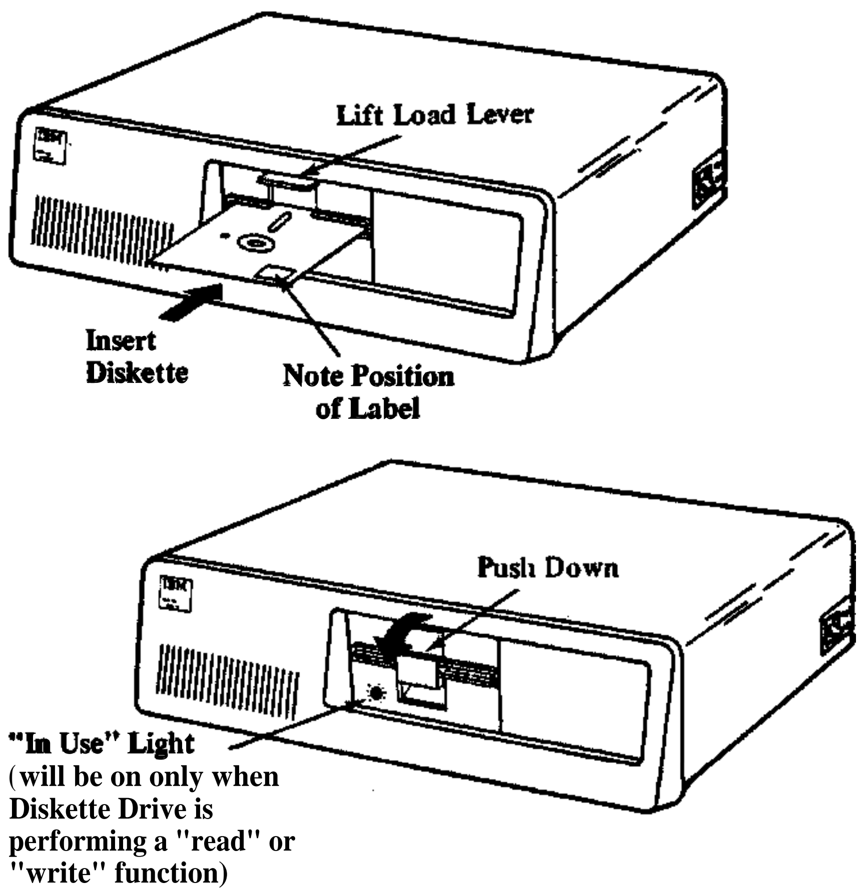
Maancri

How to Insert Diskettes

CAUTION

Damage may result to your diskette drive or diskette if you attempt to open the drive door or remove the diskette while the in-use LED is lit. The diskette should be removed when the in-use LED is off and before switching system power off.

When you install a diskette, the drive door must be open and the label of the diskette must be facing up. The notch along the side of a diskette must be on the left side as you insert the diskette into the drive. Insert the diskette into the drive as shown in the figure. Once the diskette is inserted, the drive door must be closed. The computer cannot read the diskette unless the door is closed.



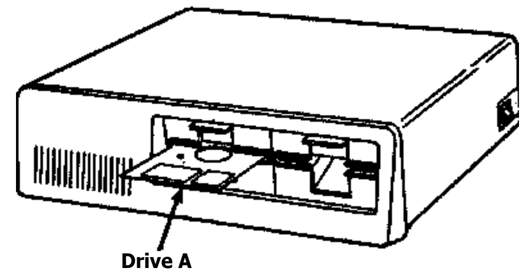
Copying DOS and AML/Entry Shipped Diskettes on to Work Diskettes

When you purchased your AML/Entry package, you received four AML/Entry shipped diskettes. You must merge DOS and the AML/Entry shipped diskettes to produce AML/Entry work diskettes. AML/Entry's Autoinit program formats the diskettes and copies, or merges, DOS and the AML/Entry programs on to the diskettes. The items necessary to create AML/Entry work diskettes are:

Blank diskettes needed as follows:	
PC with dual-sided drive(s)	three (3)
XT with a fixed disk	none
5531 Industrial Computer	
with a fixed disk	none
with dual-sided drive(s)	three (3)
PC AT	
with a fixed disk	none
with high-density drive(s)	two (2)
with dual-sided drive(s)	three (3)
AML/Entry shipped diskettes	four (4)
DOS diskette	

Autoinit is on Volume 1 of the AML/Entry shipped diskettes. Autoinit produces double-sided diskettes when used on a double-sided drive system. If your system has a fixed disk drive, see "Copying DOS and AML/Entry Diskettes on to a Fixed Disk" on page 2-15 for installation instructions.

Perform the following outlined procedure to make self-booting AML/Entry work diskettes (the first steps are required to start DOS).



System: Power off.

You: Open drive A (left drive).
You: Insert your DOS diskette or an exact copy into drive A.
You: Close drive A.
You: Set the System Unit power switch to ON.

System: Drive A in-use LED comes on after the self-check is complete. When the DOS diskette is loaded, the drive A in-use LED goes off and you are asked for the date.

You: Enter the date in the form shown by the prompt.

System: Displays entered date.

You: Press the (enter) key.

System: Screen displays a prompt for time.

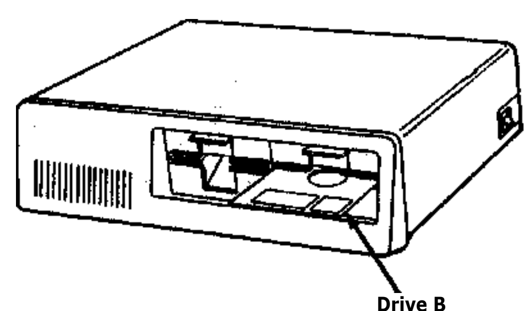
You: Enter time using 24-hour clock times. For example, enter 13:30 for 1:30 pm, or enter 7:24 for 7:24 am.

System: Displays entered time.

You: Press the <--- (enter) key.

System: Screen displays the DOS prompt.

You: Open drive B (right drive).
You: Insert Volume 1 of your AML/Entry shipped diskettes into drive B.
You: Close drive B.



The next instruction starts execution of the program Autoinit, which is on your AML/Entry diskette Volume 1. Follow the screen prompts when asked to press a key or change diskettes.

You: Enter: **b:autoinit**

System: Screen displays B:AUTOINIT

You: Press the <----I (enter) key.

System: Screen displays prompts.

You: Follow the instructions of the system prompts until complete. Place diskettes in a drive, remove diskettes, and replace AML/Entry or DOS diskettes in the appropriate driveas prompted.

Copying DOS and AML/Entry Diskettes on to a Fixed Disk

This procedure loads the AML/Entry programs on a fixed-disk drive.

> **Note:** Make sure that DOS is installed on your fixed-disk drive as the "boot" operating system. If you are using international DOS, make sure that you have installed your national DOS on the fixed-disk drive.

To load the programs, accomplish the following procedure (the first si steps are required whenever you need to start DOS).

You: Check that drive A is open.
You: Set the system unit power switch to ON .

System: Fixed-disk drive in-use LED comes on after the self-check is complete. It goes off when DOS is loaded and you are prompted for the date.

You: Enter the date in the form required by the prompt.

System: Screen displays entered date.

You: Press the (enter) key.

System: Screen displays a prompt for time.

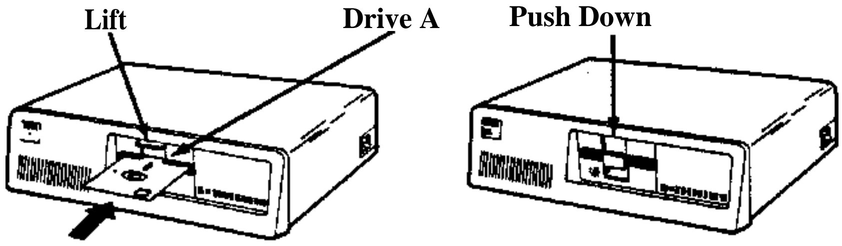
You: Enter time using 24-hour clock times, For example, enter 13:30 for 1:30 pm, or enter 7:24 for 7:24 am.

System: Screen displays entered time.

You: Press the <----I (enter) key.

System: Screen displays the DOS prompt.

You: Insert Volume 1 of the AML/Entry diskettes.
You: Close drive A.



The next instruction starts execution of the program Autoinit, which is stored on your AML/Entry diskette. Follow the screen prompts when asked to press a key or change diskettes.

You: Enter: **a:autoinit**
You: Press the <----I (enter) key.

System: Screen displays prompts until complete.

You have now created the AML/Entry System. The files in the system now contain both AML/Entry language files and system exercisers for maintenance (utility programs).

Note: The Autoinit procedure will add the line FILES=12 to the file CONFIG.SYS on the root directory of the fixed disk. AML/Entry requires this line to run correctly. If the file CONFIG.SYS already contains a FILES= line, then the FILES=12 line will task precedence over the original FILES= line. Thus if you are using other software which requires a FILES parameter greater than 12, make sure to edit CONFIG.SYS and change the FILES=12 line. See the DOS reference for more on CONFIG.SYS and the FILES parameter.

Note: The Autoinit procedure loads the programs to the current directory designated on the fixed disk (see DOS reference).

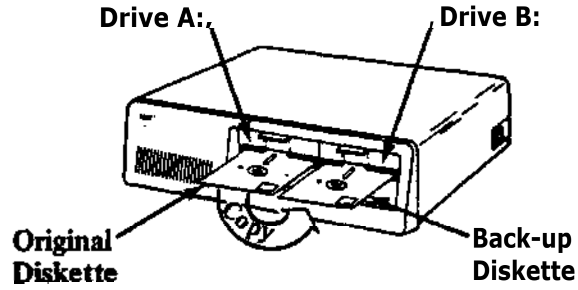
Making Backup Copies of the AML/Entry Work Diskettes

You have created the self-booting AML/Entry work diskette(s). Now, make your backup AML/Entry work diskette copies. The below outlined instructions are for a system with two diskette drives. For a single diskette drive, omit the drive names when entering **diskcopy.**

You: Insert your DOS diskette into drive A (left drive).
You: Close drive A.
You: Enter: **diskcopy a: b:**
You: Press the (enter) key.

System: Prompts instructions on the screen.

You: Enter **Y** each time the prompt is displayed until you have copied each diskette on a formatted blank diskette.



Some Words About Files

AML/Entry programs are saved in files on disk/diskettes. A file is a collection of records. In this case, the records are lines of your program. When you save a program in a file, you assign a name to the file (filename or filespec) so that you are able to reload the program later.

Because DOS is used for naming files, you must use DOS rules for naming your programs. When you name a file to save it or read it back for further use, you need to adhere to the below listed rules.

Device name The device name is A: B:, or C: for drive A (left drive), drive B (right drive), or drive C (fixed disk drive), respectively.

Filename The filename comes directly after the device name, no spaces are allowed between the filename and the device name Filenames are 1 to 8 characters, and certain special characters may be used (see DOS reference). The first character must be a letter. Any DOS reserved words should not be chosen for filenames.

The following filenames have special meaning to the compiler and should not be used for your filename.

AUX	LPT1
COM1	NUL
COM2	PRN
CON	USER
LINE	

File type The file type directly follows the filename and is preceded with a period (.). If you don't use a file type, AML/Entry assumes **.AML,** for editor files. If you want to view other files, such as the files you created during a compile, you must specify the correct file type because each file has the same name as your application program. The files have **.LST** and .SYIN file types.

If you have not changed the file type at any time, the AML/Entry programs create the correct file type for you.

Certain commands, such as **DEL,** erase a file from a diskette so you are required to specify a file type as a safety precaution.

When you specify a file to be edited, the editor reads in the program file from the specified device (disk or diskette). A copy of the program remains in the IBM Personal Computer memory while you are editing. Any changes you make are made only to the copy in memory unless you save the changes. The use of the SAVE command is described in Chapter 3, "Using the AML/Entry Editor" during the editor exercises, and in Chapter 7, "Operating the Manufacturing System."

Some Words about Diskettes

The first time you use a diskette, it must be formatted to get it ready to receive information. Formatting erases any previously stored information and checks the diskette for bad spots. It also builds a directory to hold information about the files that will eventually be written on it.

Backing up your programs by making copies of them is a good habit to get into. This protects you if your original copies are misplaced or damaged.

The diskette is coated with a magnetic substance and enclosed in a permanent protective jacket. Information is written on or read from the magnetic surface of the diskette. The computer can read the information as often as needed or it can write new information on the diskette in any unused space. The computer can also replace, or erase, old information with new information by writing over the old information.

How to Handle/Store Diskettes

Diskettes must be handled with care. Do not touch the exposed recording surfaces. To make sure your diskettes do not get dirty or scratched,. store them in their envelopes and storage boxes. It is also important to keep the diskettes away from heat and magnetic field sources.

To keep the computer from writing over information already on the diskette, you can cover the write-protect notch with a piece of tape. This allows the computer to only read the information from the diskette.

Do's and Don'ts for Handling Diskettes

DO's		
Felt Tip Only	Grasp by Corner	When Not In Use
DON'Ts		
Don't Bend	No Pencils, No Clips, No Ballpoints	Don't Touch Disk
No Sun Not Too Hot	No Magnets	No Cleaning Fluid

LOADING THE AML/ENTRY SELF-BOOTING WORK DISKETTE

The following two procedures describe how to use your self-booting AML/Entry work diskette or your self-booting AML/Entry system on a fixed-disk drive. The first method describes a system with the power switch in the OFF position. The second method describes your system with the power switch ON. It is called System Reset.

Method 1 (Power Switch OFF)

Ignore the instructions pertaining to loading diskettes if your system is already loaded on a fixed disk drive.

You: Open drive A (left drive).
You: Insert an AML/Entry system diskette into drive A (left drive).
You: Close the drive door.
You: Set the system unit on/off power switch to ()N.

System: Cursor appears on the screen in approximately four seconds. After the self-test is completed, the system reads the AML/Entry system files. (The in-use LED comes on).

System: Screen prompts you to enter a date when the in-use LED goes off

You: Enter in the date in the format shown on the screen.

System: Screen displays entered date.

You: Press the (enter) key.
You: Enter in the time in the format shown on the screen.

System: Screen displays entered time.

You: Press the (enter) key.

System: Menu is loading while the in-use LED is on. When the in-use LED goes out, the menu is displayed on the screen.

Note: If you installed the AML/E system onto a subdirectory of the fixed disk, then the AUTOEXEC.BAT file will be unable to start the menu. To bring up AML/E, it is necessary to use the **chdir** DOS command, followed by **menu.**

Your IBM Personal Computer is ready for AML/Entry applications.

Method 2 (System RESET)

If your computer power is On and you want the AML/Entry work diskette loaded, use the below outlined procedure.

Skip the next few instructions pertaining to loading diskettes if your system is already loaded on a fixed disk drive.

You: Open drive A (left drive).
You: insert an AML/Entry system diskette into drive A (left drive).
You: Close drive A.
You: Simultaneously press the **CTRL, ALT,** and **DEL** keys.

System: Cursor appears on the screen and the date request prompt is displayed after the in-use LED goes off.

The remaining procedure is the same as the previous power-up procedure.

USING THE AML/ENTRY PROGRAMMING SYSTEM MENU

The AML/Entry programming system menu shown below allows you to select one of the options necessary for branching into AML/Entry programs or to return to DOS.

You: Enter the selected function number and press the <---┘ (enter) key to select any of the below listed options.

```
        AML/Entry Version 4.1 Programming System

 7505-AAN (C) Copyright IBM Corporation 1983, 1984, 1985

                  Select a function:

               0. Return to DOS.
               1. Edit/Teach a program.
               2. Compile a program.
               3. Load a program to the controller.
               4. Unload a controller program.
               5. Set system configuration.
               6. Set program name and options.
               7. Communicate with controller.
               8. Generate Cross Reference Listing.

                  Enter Option ===>
```

Option 0 (Return to DOS)

You may want to exit to DOS from the AML/Entry system. To return to DOS, do the below outlined procedures.

You: Enter a 0 and press the <—i (enter) key to return to the DOS environment.

System: Displays the DOS prompt.

If you use option 0 and want to return to AML/Entry, follow the below outlined' procedure.

You: Enter **menu** and press the <—i (enter) key to return to the programming system menu.

System: Displays the AML/Entry programming system menu.

Option 1 (Edit/Teach a Program)

Teach mode can only be used from the editor. Accessing and using teach mode is described in Chapter 6, "Using the AML/Entry Teach Mode." The editor is described in Chapter 3, "Using the AML/Entry Editor."

WARNING

Files with the following file types should never be edited with the AML/Entry editor or any other editor:

.TXT
.ASC
.EXE
.COM

Unpredictable results occur if this is attempted

To create or modify application programs and use the teach facilities, do the below outlined procedures.

You: Enter a 1 and press the <---J (enter) key to invoke the editor.

System: Editor screen displays the message outlined below.

ENTER FILESPEC --->

Here, the computer is requesting the name of the file to be read from a diskette. See "Some Words About Files" on page 2-17 regarding names. If you do not enter a filename, the computer assumes that you want to create a new program. If you have previously created a file (program) on the installed diskette, you can enter its name and the computer can then display that program. The AML/Entry system prompts you if you attempt to access a file that is not on that diskette.

Option 2 (Compile a Program)

To compile an AML/Entry program, do the below outlined procedures.

You: Enter a **2** and press the (enter) key to invoke the compiler.

System: Prompts you to enter the filename that is to be processed by the AML/Entry compiler program. It must be an **.AML** type file.

If you elect not to process a file at this time you can exit the compiler by pressing the (enter) key

Load File (.ASC)

The compiler checks the application program for certain errors and then creates a special file to be loaded into the manipulator controller. The file created by the compiler during the compile has the same name as your application program with a .**ASC** file type.

Compiler Phase Messages

When the compiler is running, it displays three messages to indicate the different phases of the compiler program. The messages, listed below, are used only to show the progress of the compiler.

Reading Input File
Converting AML/E Program
Writing .ASC File.

Displayed Information

When an error is encountered, the compiler displays the line number and text of the line that contains the error. An indicator is used to show where the error was detected.

The compiler accepts input from files with line lengths of up to 255 characters. However, the screen displays only the first 80 characters of any line. If the error is in a column off the screen, the error display indicator moves to the right-most column position on the screen.

Compiler Errors

If you have any errors in your application program, the compiler reports them during the Converting AML/Entry program. Errors terminate the compiler execution and eliminate the creation of the .ASC file.

Note: If you modify your application program after it is compiled, you must go through the compile process again to update the .ASC file.

Listing File (.LST)

The compiler can create a file that contains the program listing and any error messages created during the compilation of your program. This file has the same name as your application program, but the file type is **.LST**. This file can be viewed or printed in the editor by specifying the file name and then using the file type .LST. The AML/E editor only handles 72 characters per line, and since the compiler prepends each source file line with the machine address of the first instruction on that line, lines may grow to longer than 72 characters. Thus on entry to the editor, lines may be truncated to 72 characters. Another way to view the file is to use the DOS **type** command and another way to print the file is to use the DOS **print** command.

The listing file provides some very useful information on successfully compiled programs. Along the left margin, the compiler includes the starting address of the instructions on that line. This information helps determine if more efficient code can be written. Because AML/Entry supports Read records that report the address of the failing instruction, you are able to check program listings to find the AML/Entry statement that failed. This is especially useful when determining the cause of a data error. The addresses from the listing are also used when you set a debug address stop using the host communications interface. Refer to the C "10" option of "COMAID" on page 8-14.

Symbol File (.SYM)

The compiler can also create a file that contains information about symbols and variables in your program. This file has the same name as your application program, but the file type is .SYM . This file can be viewed or printed in the editor by specifying the filename and using the file type .SYM. The .SYM file is not intended to be used directly. The XREF program (option 8) is used to produce readable information.

> **Note:** If there is not enough space available on your diskette, the compiler does not create a valid .ASC file. When you list the directory of the diskette, an invalid file appears with a size of 0 following the filename and file type. Occasionally, the compiler may indicate that the file compiled correctly, even though there was insufficient space. Always check the amount of available space on the diskette before compiling your program.

Option 3 (Load a Program to the Controller)

The below listed conditions must exist before you load a program to the controller.

1. The controller must have power-on.
2. The control panel's Manip Power LED must be on.
3. The control panel's On Line LED must be on.
4. The communication cable between the IBM Personal Computer and the controller must be attached.
5. A compiled copy (.ASC type) of your program must be on the diskette or fixed disk.

You can select 1 of 5 partitions. If a program exists in the selected partition, the controller erases that program before the load begins.

To load your application program into the controller, accomplish the below outlined procedures.

You: Enter a **3** and press the (enter) key to load your application program into the controller.

System: Prompts you to enter such information as the filename and the destination partition in the controller.

Option 4 (Unload a Controller Program)

1111

To unload a program from a partition and allow a larger program to be transmitted, the below listed conditions must exist.

1. The controller must have power-on.
2. The control panel's Manip Power LED must be on.
3. The control panel's On Line LED must be on.
4. The communication cable between IBM Personal Computer and the controller must be connected.

Note: Unloading a program does not return the compiled program to the computer, but actually erases it from controller memory.

To unload controller programs from the partitions, accomplish the below outlined procedures.

You: Enter a **4** and press the (enter) key to clear partition(s).

System: Prompts you to enter the partition(s) to be cleared.

You: Select 1, 2, 3, 4, 5, or type "all" as the partition(s) to unload.

System: Program unloader clears the specified storage partitions at the controller.

Option 5 (Set System Configuration)

The configuration utility is a menu-driven program. Prompts are available to aid you to access, exit, or update your AML/Entry program diskettes. If you change the configuration on your self-booting diskette from the configuration that was shipped, you must change each new diskette you create from the master diskette. The configuration information is used by all AML/Entry options.

Note: Make sure that your diskettes are not write-protected before using this utility. If the diskettes are write-protected, an error will occur.

The below listed selections determine your AML/Entry system configuration.

- Which robot model you are using.
- If you want to use inches or millimeters when defining points in the work space.
- Which communication port is selected for communications to the controller.

To execute the system configuration utility, accomplish the below outlined procedure.

You: Enter a **5** and press the <—a (enter) key to gain access to the configuration utility menu.

System: Screen displays configuration utility menu. The following screen is a sample; the one you receive will vary based on the configuration of your system.

```
AML/Entry Version 4.1 Configuration Utility

1. Exit utility, updating the configuration as current
   selections.
2. Quit utility, making no changes to configuration.
3. Specify Robot Type. {Currently 7545)
4. Specify Units. {Currently MM)
5. Specify Communications Port. {Currently COM1:}

Enter Option ===>
```

When using your AML/Entry program configuration, do not mix inches an millimeters in an application program. This causes errors in th controller or yields unpredictable results.

Note: Home position is a special reference point given in millimeters only. Requesting a move to Home position in inches may result in locating a point other than the precise home. If Home position is used in a program, it is recommended that the program operate in millimeters.

Check the editor/teach prompts as you enter the filename to determine the editor is configured for the right model of manipulator. incorrect configuration does not allow you to communicate with th controller in the teach mode.

Note: Do not reconfigure the communication port option unless you have a second asynchronous communication card.

Option 6 (Set Name and Options)

To specify a default filename, compiler options, and a partition number, do the below outlined procedure.

Note: If a default is specified, it automatically applies to each of the menu options. If a file extension is not specified in the filename, it defaults to the function being used. The Editor and Compiler use .AML, Load uses .ASC, and Xref uses .SYM.

You: Enter a **6** and press the <---┘ (enter) key to gain access to the Programming System Options menu.

System: Screen displays the Programming System Options menu.

```
AML/Entry Programming System Options

        Filename: testl
Compiler Options: /s/h
Partition Number: 3

   Select a function:

0. Return to DOS.
1. Edit/Teach a program.
2. Compile a program.
3. Load a program to the controller.
4. Unload a controller program.
5. Set system configuration.
6. Set program name and options.
7. Communicate with controller.
8. Generate Cross Reference Listing.

   Enter Option ===>6
```

You are able to enter the default filename, compiler options, and partition number on this screen. This data is used by all the other options (except 0), until it is changed by using option 6 to enter new default data. This returns the cursor to 'Filename:' and allows you to enter new default data. You must press (enter) after each default has been entered. After 'Partition Number:' is entered, you are able to select options 0 through 8. The programming system options menu replaces the programming system menu until you select option 0 and return to DOS.

Note: The cursor keys should not be used when entering data into the filename, compiler options, or partition number field. The system simply overlays the new information on top of the old information. The space bar should be used to erase an entry within a field.

Option 7 (Communicate with controller)

The below listed conditions must exist before you communicate with the controller.

1. The controller must have power-on.

2. The control panel's On Line LED must be on.

3. The communication cable between the IBM Personal Computer and the controller must be attached.

The program that communicates with the controller is named COMAID. To execute COMAID, follow the below outlined procedure.

You: Enter a **7** and press the <—a (enter) key to gain access to the COMAID program.

System: First greets you with an information screen, followed by the COMAID main menu (see chapter 8).

You: Enter communication requests, giving additional information according to system prompts.

You: Eventually select option "e", which will return control to the AML/Entry Menu.

Option 8 (Generate Cross Reference Listing)

A cross reference listing is a list of all the AML/E variables that consume space in controller memory. The list contains such information as the name of the AML/E variable, its type point, pallet, counter, etc.), its starting variable number, its size, etc. The cross reference listing is generated by a program named XREF.

XREF Program

The XREF.EXE program may be invoked by selecting option 8 from the AML/E menu. It produces a formatted listing of the variable names along with the associated controller variable numbers. When reading or writing variables from or to the controller, you must reference the variables by their numbers produced by the XREF program. See "XREF Program" on page 4-89 for a discussion of the listing produced by XREF.

> **Note:** To use the XREF program you must have generated a .SYM file when you compiled your program.

After the compiler finishes, and control is returned to the AML/E menu, select option 8. The XREF program will begin execution, and will prompt you for the filename. You do not have to include the file extension (.SYM), because the XREF program uses this as the default.

To print the output of the XREF program on the system printer, strike the Ctrl and PrtSc keys simultaneously before selecting option 8. The XREF program will then be started, and all lines that are printed on the screen will also be printed on the printer. When the program terminates, strike the Ctrl and PrtSc keys simultaneously to disable the printing to the printer.

It is also possible to redirect the output of the XREF program to a file using the redirection feature of DOS. See your DOS manual for a description of redirecting the output of a program. To do this requires you to first leave the AML/E menu. After returning to DOS, then a variety of DOS command lines can be used to invoke XREF and route the output to different places. Consider some of the following DOS command lines.

DOS Command Entered	Action
XREF	Starts the XREF program. Will prompt for the file to be used.
XREF TEST	Runs the XREF program on the file TEST.SYM.
XREF TEST >TEST.XRF	Runs the XREF program on the file TEST.SYM. The output is placed in the file TEST.XRF instead of on the screen (see DOS reference).
XREF TEST >PRN	Same as above except the output is printed on the printer (see DOS reference).

Because XREF expects a key to be struck at the end of its execution, th user must remember to do this. If redirection is being used the "Strik any key to continue" message will not appear on screen, and will ti redirected to the file or device specified on the DOS command line. It is best to use the enter key (<—I), because if the file could not b found, then XREF is awaiting a new file name. An enter will cause XRE to end.

IP AML/Entry, you are allowed to compile and load AML/Entry programs using DOS batch programs.

Note: If you have a color screen, the MODE (DOS command) must be set to 80 before running the compiler.

Invoking the Compiler

The compiler is invoked from DOS with an optional command string that specifies filename and option, as listed below. It recognizes both lower and upper case characters.

/L - produces a listing file (.LST) on the diskette or fixed disk

/H - produces a hard copy error report

- /S - produces a symbol file (.SYM) on the diskette or fixed disk
- /E - aborts compile after five errors are encountered
- /B - batch mode operation (does not require user input), compiler does not display "press any key..." message

Options must be entered only <u>after</u> the filename is specified. An example using the above options is outlined below.

```
A> compiler task/L/S/B
```

This invokes the compiler to compile the file TASK.AML and to create both a listing and a symbol file without user input. If the file is not on the default drive, the name of the device that contains the file must be included in the file specification.

ERROR LEVEL: A DOS error level is returned by the compiler. It is accessed in .BAT files by the DOS batch ERRORLEVEL command. If no errors are encountered, the compiler returns a 0 (zero). If an error is encountered, the compiler returns a 4 (four). If a serious compiler error is encountered, (i.e. "drive not ready"), the compiler returns a 10 (ten).

Loading/Unloading a Program to the Controller

Loading (or unloading) a compiled program is done by invoking COMAID. COMAID accepts parameters passed on the DOS command line. The parameters following the COMAID command are optional. Optional parameters must be entered sequentially. Once an optional parameter is omitted, the remaining parameters can not be specified. If any parameters are omitted, you are prompted for the information. Loading a program only accepts an .ASC extension. If any other extension is used, COMAID returns an error. If an extension is not specified in the filespec, .ASC is assumed.

To load a partition:

You: Enter this command:
COMAID L [filespec] [partition]

For example,

```
A> COMAID L MAIN 2
```

will load the compiled program MAIN.ASC to partition 2.

To unload a partition:

you: Enter this command:
COMAID U [partition]

Example Batch Program

An example batch program is outlined below. This batch file compiles the PALLET.AML file, and generates a listing (.LST) and a symbol table (.SYM) file. If errors are detected, an error message is printed. If no errors are detected, then the compiled program is downloaded to partition 3.

```
PAUSE INSERT PROGRAM DISKETTE INTO DRIVE A
COMPILER A:PALLET/L/S/B
REM THIS COMPILES AND GENERATES A LISTING FILE AND A SYMBOL FILE
IF ERRORLEVEL 4 GOTO LAB1
REM ERRORLEVEL NUMBER WAS 0
GOTO LAB2
:LAB1
REM THERE WAS AN ERROR
ECHO ERRORS DETECTED
GOTO END
:LAB2
ECHO SUCCESSFUL COMPILATION -- NOW DOWNLOADING
COMAID L A:PALLET 3
:END
```

AML/ENTRY UTILITY PROGRAMS

AML/Entry diskettes contain the below listed utility programs to check the operation of the controller.

- 75XXexX.AML files for exercising the manufacturing system with a 7545 or 7547 manipulator attached

- 800S-exX.AML files for exercising the manufacturing system with a 7545-800S manipulator attached

- An OFFSET.EXE program used to make corrections to any servoed axis. Refer to the IBM Manufacturing System Maintenance Information Manual, of your system for usage of this program.

- The AMLECOMM modules which are used to create a working copy of AMLECOMM. The following files comprise the AMLECOMM system: AMLECOM*.BAS, COMPILE.BAT, BCVTFLT.COM, BFLTCVT.COM, CCVTFLT.OBJ, CFLTCVT.OBJ, MSGCOM.TXT, CONFIG.BAS, and MENUCOMM.BAS.

If you require one of the exerciser programs, you must do the below outlined procedure.

You: Compile the program.

You: Load the program in a controller partition.

The OFFSET.EXE program is started by entering the name while in the DOS environment and pressing the enter key. The program provides prompts for you to follow.

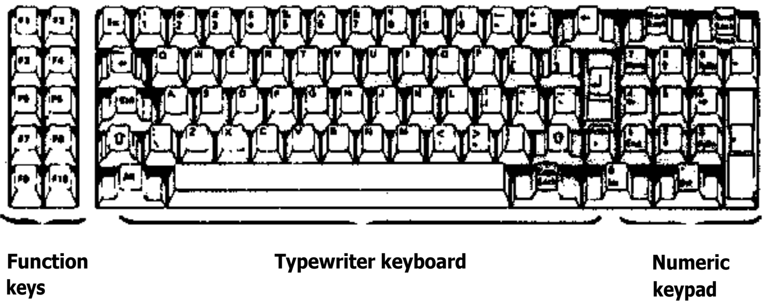
A description of the AMLECOMM system is in Chapter 8, "Communications."

CHAPTER 3. USING THE AML/ENTRY EDITOR

This chapter contains exercises for manipulating text, using the features of the editor. A reference of Editor and AML/Entry commands is provided in Appendix A.

The maximum file size for the Editor is 500 lines for personal computers with 192-K of memory and 800 lines for those with 256-K.

When using the editor, the function keys on the keyboard have special meaning. These keys are described before the description of the editor commands.



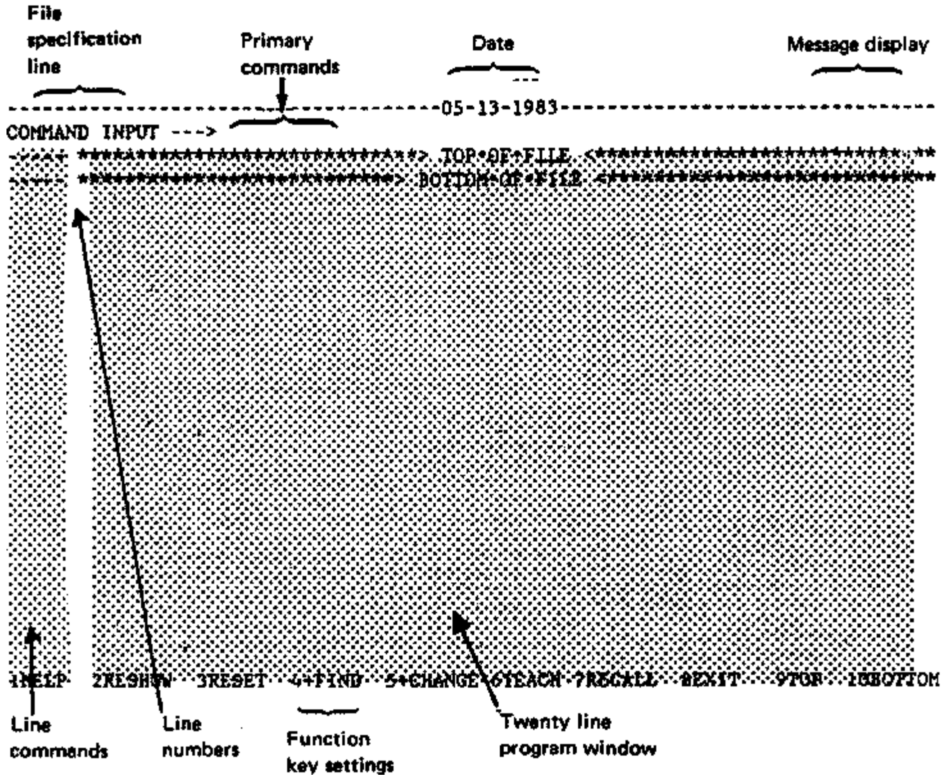
FULL SCREEN EDITING

Any line on the display screen can be changed or added to by positioning the cursor where you want to make the change and then making that change. The AML/Entry editor is a full-screen editor.

With a full screen editor, changes can be made anywhere in the 20-line window by moving the cursor around and entering in the characters. The editor does not really see the changes until the <-┘ (enter) key, a function key, or one of the scrolling keys (PgUp, PgDn) is pressed. If you make some changes and then change your mind, press function key F2 (reshow) and the screen returns to the state displayed when you last pressed the enter key. Once the <-┘ (enter) key is pressed, the editor makes the changes to its copy of the program.

The copy of the program that the editor changes is the storage (memory) copy. It is not a permanent storage area. If the power fails, the copy in this storage area is lost. In this chapter, there is an exercise that saves the storage copy on a diskette. To prevent its loss by any type of interrupt, such as a power failure, the storage copy should be saved periodically on a diskette.

Features of the editor shown below are described in the following paragraphs.



File Specification Line

This is the top line viewed on the screen. It contains the device name (if specified), the file name, the file type, and the date. New programs do not have a file name or file type.

Primary Commands

These commands are entered on the command input line. Primary commands manipulate text and manage disk operations.

Date

The date you enter, when booting the system, is displayed at the top of the screen.

> **Note:** In the example screens used in this chapter, a date of 7/08/85 is used.

Message Display

Both error messages and successful execution messages are displayed in the upper right-hand corner of the editor display area.

Top of File/Bottom of File

Top of File and Bottom of File are used to designate the beginning and ending of the file. All data is inserted between these two lines.

Line Commands

These commands are entered to the left of the line numbers on the screen, starting in the first position. Line commands insert a blank line(s), delete a line(s), copy a line(s), or move a line(s). You can not use the Ins (insert) key in this area of the screen.

Line Numbers

The line numbers run vertically down the screen and are assigned by the editor as you insert, delete, copy, or move lines. The editor numbers all the lines of the program sequentially. The numbers can not be changed, but you can rearrange the lines of the program.

Function Key Settings

The bottom line of the screen displays the settings of the function keys. Function keys allow frequently used commands to be executed by pressing a single key.

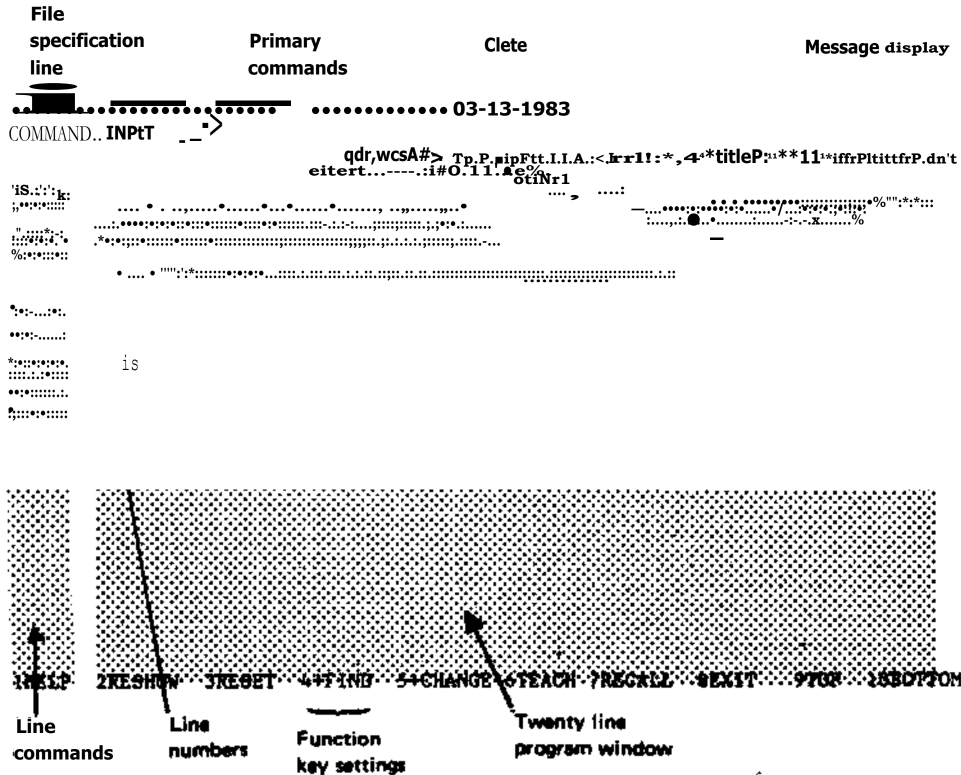
20-Line Program Window

Twenty lines of the program can be viewed or prepared without moving the window. Moving the window is sometimes referred to as scrolling.

KEYBOARD USAGE FOR THE EDITOR

Function Key Settings

Function key settings as shown on the bottom of the screen have numbers that correspond to the numbers on the function keypad of the keyboard. These functions are available only when you are using the editor and are displayed only when available for use. The various uses of the function keys are listed below.



Help Screens

If you want information about any of the commands while you are working in the editor, press the F1 key. The help screen contains several pages of information about commands and keys for the editor. Prompts are provided to help you page through the screens and return to the editor.

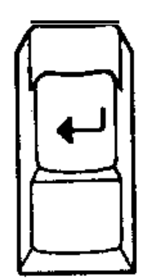
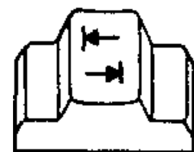
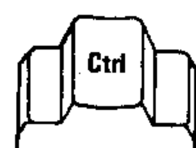
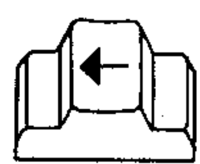
Key	Screen Display	Description
Fl	1HELP	Displays the editor commands.
F2	2RESHOW	Clears the screen of changes that have not been entered.
F3	3RESET	Clears pending line commands.
F4	4+FIND	Searches for the occurrence of a character string defined by a previous FIND command.
F5	5+CHANGE	Searches for and changes the next occurrence of a character string that was previously defined by a CHANGE command.

Note: See CAUTION, WARNING, and DANGER notices in chapter 6 before using TEACH.

Key	Screen Display	Description
F6	6TEACH	Invokes teach mode to control movement of the manipulator by the Personal Computer.
F7	7RECALL	Recalls the last point from teach mode to be used in an application program.
F8	8EXIT	Exits the editing session and saves the file.
F9	9TOP	Moves the window to the beginning of the file.
F10	10BOTTOM	Moves the window to the end of the file.

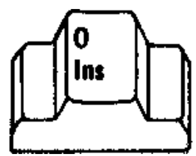
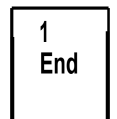
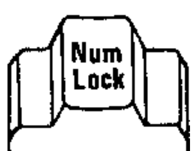
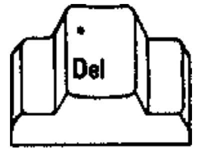
Special Keys

When in the editor, you may be using some of the keys described below.

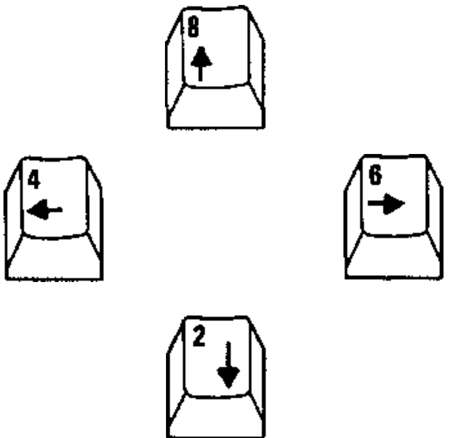
SPECIAL KEY	DESCRIPTION
	The **enter key** moves the cursor down one line to the first position on the next line. Pressing the enter key causes the editor to update changed fields or characters. Primary and line commands are executed when the enter key is pressed.
	The **tab key** moves the cursor six character spaces right. If the shift key is held down while you press the tab key, the cursor moves six characters to the left.
	The **shift key** changes characters entered **from** the keyboard to the uppercase characters shown on the keys. Alphabetical characters are automatically in uppercase in the editor.
	Pressing the **control key** simultaneously with another key provides special meaning to that key. For example, to scroll to the beginning of the program, press the CTRL and PgUp keys at the same time.
	The **backspace key** moves the cursor one position to the left and deletes the character at that position.
	Pressing the **shift key** and the **PrtSc key** simultaneously causes the contents of the screen to be printed on the optional printer.
	Typing this character and pressing enter on the editor primary line causes the previous primary command to be displayed.

Numeric Keypad

The following descriptions are for keys on the numeric keypad:

KEY	DESCRIPTION
	The **Ins key** puts you into insert mode, allowing characters to be inserted before the cursor position. Characters to the right of the cursor are shifted to the right one position for each character entered. To exit the insert mode, press the Ins key again or press the enter key.
	The **End key** advances the display window to the last line of the program if the program is longer than 20 lines.
	The **Home key** moves the cursor to the command input line.
	The Num **Lock key** is a shift key for the numeric keypad only. The key works like a toggle switch; you must press the key a second time to return the numeric keypad to lowercase characters. In AML/Entry, this key should not be used to create uppercase numeric keypad characters. In teach mode, the key is ignored by the AML/Entry programs for safety reasons.
	The **Del key** deletes a character at the cursor position, and characters to the right of the cursor are shifted left one position.
	The **Page Up key** scrolls the window up a half page.
	The **Page Down key** scrolls the window down a half page.

The **cursor** positioning keys move the cursor in the direction shown by the arrow on the key.



Control Keys

The control key provides special meaning to other keys when used simultaneously with those keys. The following are Ctrl key applications when using the editor:

KEY	DESCRIPTION
Ctrl_PgUp	Scrolls the display to the beginning of the program (same as F9).
Ctrl_End	Deletes all text characters from the cursor position to the end of the line.
Ctrl_	Moves the cursor to the first position of the program window on the next line. The screen does not blank and refresh when you use these keys, and commands are not processed.

SETUP FOR EDITOR EXERCISES

Your computer does not have to be connected to the controller during any of these exercises. Refer to Chapter 2, "Getting Started on the IBM Personal Computer" for the power-up instructions for the Personal Computer.

GETTING TO THE EDITOR FROM THE MAIN MENU

With the main menu displayed, perform the following steps to use the editor when you do not have a previously-named file to edit.

You: Enter: 1.
You: Press the <--┘ (enter) key.

System: Screen prompts for filename as outlined below.

ENTER FILESPEC --->:

You: Press the <---┘ (enter) key.

System: Screen displays the new file as outlined below.

```
— — — — — — — — — — — —07-08-1985— — — — — — — — — — — —
COMMAND INPUT --->
 - -  ***********************> TOP•OF•FILE <**************************
 -    ***********************> BOTTOM•OF•FILE <***********************







 1HELP 2RESHOW 3RESET 4+FIND 5+CHANGE 6TEACH 7RECALL 8EXIT 9TOP 10BOTTOM
```

The next instruction moves the cursor down to the line-command area of the editor.

You: Press the (enter) key.

System: Cursor moves to the TOP•OF•FILE line.

EXITING THE EDITOR

Most of the exercises in this chapter use an edit screen developed in a previous exercise of the chapter. To avoid starting over each time, you can name the program when you exit the editor, using the instructions from this section. The exercises at the end of the chapter provide more information on saving and deleting files.

Use these instructions to name the program when you exit the editor the first time. Once the program is named, you can exit the editor using a single step.

Exiting the Editor for the First Time

Perform the following steps the first time you exit the editor exercises in this chapter.

You: Press function key **F8**

System: Screen prompts for filename as outlined below.

ENTER FILESPEC --->:

You: Name the file by entering: **practice**
You: Press the <----I (enter) key.

System: The file is saved on the diskette under the name PRACTICE.AML. The screen refreshes and displays the main menu.

Recalling the Practice Program

If you used the procedure in "Exiting the Editor the First Time", return to the program by performing the below outlined steps.

You: Enter 1 with the main menu displayed on the screen.
You: Press the <---ˡ (enter) key.

System: Screen prompts for filename as outlined below.

ENTER FILESPEC --->:

You: Enter the device name (b: if drive B), and then enter **practice**
You: Press the (enter) key.

System: In-use LED comes on and the screen blanks. After the in-use LED goes out, the program appears on the screen.

Exiting the Practice Program After You Name It

Once you name the file, you can exit the editor by pressing function key **F8** once. The screen displays the main menu after the editor has written the program on the diskette.

INFORMATION ABOUT LINE COMMANDS

There are 10 line commands available for your use in the edit mode. Line commands are typed in the area to the left of the program line numbers. Line commands are not processed until you press the enter key.

Commands not started in the first character position are ignored. The COMMAND PENDING message appears if parts of a command are still required to complete the processing. The message appears when commands that are used in pairs are not all entered prior to processing. This may occur if you enter part of the paired command on a line of the present screen and then press a paging key to display another screen so that the next command can be entered. The COMMAND PENDING message is not an error message.

Line Command	Description
A	Copies/moves after this line
B	Copies/moves before this line
C	Copies this line
CC	Copies this block of lines
D	Deletes this line
DD	Deletes this block of lines
I	Inserts a line or a number (1-9) of lines
	Moves this line
MM	Moves this block of lines
R	Repeats a line a number (1-9) of times

You can clear commands and pending commands before processing by pressing the **F3** key.

Line Command Conflicts

In general, it is not a good idea to use too many line commands on the same screen; it's easy to get confused.

If you have entered multiple line commands on the screen and the editor is not sure what you mean, the message "line command conflict" is displayed. When this occurs, the editor does not execute any of the line commands you entered at that time.

Line Commands that Cross Screens

Sometimes it is necessary to use line commands that cross screens. That is, the screen is scrolled between the first and second commands that are used in pairs. For example, you may want to move some lines of text from the front to the back of a 100 line program. You use the MM command on the lines of text on the first screen displayed and you then press the F10 key to go to the end of the program. When the screen displays the end of the program, the message "COMMAND PENDING " flashes in the right top corner of the screen reminding you that a command is needed to complete processing. You type the **A** command at the desired line of the program and then press the enter key. The text involved in the move is transferred to the location following the A command. The "COMMAND PENDING" message on the screen is cleared.

At any time until you press the enter key you can change your mind about the execution of the command(s) and press the function key **F3** . This clears any commands that are pending or any commands displayed on the screen.

Using the Line command I (Insert)

Use the I (insert) line command to provide additional programming lines to the editor. Follow the next instructions for inserting programming lines.

You: Press the enter ( ) key:

System: Cursor moves to the "TOP•OPTILE" line.

You: Enter: I

System: The screen displays I on the "TOP•OF•FILE" line.

```
———————————— 07-08-1985 ————————————
COMMAND INPUT --->
     *********************> Top•u•FILE <**************************
 - - *********************> BOTTOM•OF•FILE <**********************
```

You: Press the enter ( ) key:

System: Displays the screen:

```
———————————————— 07-08-1985 ——————————————
COMMAND INPUT --->
 - - ***********************> Top•u•FILE <************************
      1
 — — ***********************> BoTTom.u.FILE <*********************
```

The insert command (I) can **be** followed by a digit (1 to 9) that instructs the editor to insert that many lines after the line on which you typed the I.

You: Move the cursor to the left of the line numbers by using the shift and tab keys simultaneously.
You: Enter: **19**

System: Displays the screen:

```
 --------------------------07-08-1985------------------------
 COMMAND INPUT -7->
  -    **********************> Top•oF•FILE <************************
 19 1
 -- -- **********************> BOTTom•u•FILE <*********************
```

You: Press the enter ( ) key:

System: Cursor moves to the right of the number 2 and the screen displays:

```
 --------------------------07-08-1985------------------------
 COMMAND INPUT --->
  -    **********************> Top•0F•FILE <***********************
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
 -- -- **********************> BOTTom•0F•FILE <********************
```

Using the Line Commands D and DD (Delete)

The **D** and **DD** line commands are delete commands. In the following example, your cursor should be positioned to the right of the number 2. The screen for this exercise displays:

```
----------------07-08-1985----------
COMMAND INPUT --->
 -   **********************> Tu•u•FILE <***********************
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
-- **********************> BOTTom•u•FILE <********************
```

You: Move the cursor to the left of line number 2, using the shift and tab key simultaneously. The next instruction deletes a single line of text.

You: Enter: **D**

System: Displays the screen:

```
--------------07-08-1985----------
COMMAND INPUT --->
 -   **********************> To•w•FILE <***********************
    1
D   2
    3
    4
    5
    6
    7
    8
    9
   10
-- **********************> Bomm•u•FILE <********************
```

You: Press the enter (   ) key:

System: Screen now contains lines numbered 1 through 9 (line 2 was deleted and the rest of the lines renumbered) with the cursor positioned on line number 2. The screen displays:

```
— — — — — — — — — — — —07-08-1985— — — — — — — — — —
COMMAND INPUT --->
 - - ***********************> Top•u•FILE <************************
    1
    2
    3
    4
    5
    6
    7
    8
    9
— — ***********************> Barrom•u•FILE <********************
```

The next command is used in pairs to delete blocks of text.

You: Enter: CO
You: Use the cursor keys to move the cursor to line number 6, first character position.
You: Enter: DD

System: Displays the screen:

```
— — — — — — — — — — — —07-08-1985— — — — — — — — — — —
COMMAND INPUT --->
 - - ***********************> Top•u•FILE <************************
    1
DD  2
    3
    4
    5
DD  6
    7
    8
    9
— — ***********************> BOTTom•u•FILE <********************
```

You: Press the enter ( ) key:

System: Screen appears with lines 1 through 4. The cursor is positioned to the left of the line number 2.

```
— — — — — — — — — — — — —07-08-1985— — — — — — — — — — — -
COMMAND INPUT --->
 - - ***********************> Top•u•TILE <************************
    1
    2
    3
    4
— —  ***********************> BOTT01.1.0F•FILE <*********************
```

Using the Line Commands M, MM, with A or B (Move with After or Before)

The M and MM line commands are move commands. The A line command stands for "after" and B line command stands for "before". You are moving text in this exercise.

The cursor is located to the left of the line number 2. Your screen starts with the display:

```
-------------------------------07-08-1985-------------------------
COMMAND INPUT --->
 - - ***********************> Top•oF•FILE <************************
     1
     2
     3
     4
-- ***********************> BOTT011•0F•FILE <*********************
```

You: Use the cursor keys to move up to the first position on the TOP•OF•FILE line.
You: Press the control and enter (**Ctrl - <—'**) keys.

System: The cursor moves to line 1 and is located on the program window side of the number. Note the blank space between the number and the cursor. This blank space is required.

You: Enter: **PICK AND PLACE ROBOT**
You: Press: **Ctrl-**
You: Enter: **EASY TO PROGRAM**
You: Press: **Ctrl-**
You: Enter: **THE IBM MANUFACTURING SYSTEM**
You: Press: **Ctrl- <-'**
You: Enter: **IS VERSATILE**

Your screen should look like this:

```
- - - - - - - - - - - - - - - -07-08-1985- - - - - - - - - - - - -
COMMAND INPUT --->
 -   ***********************> Top•0F•FILE <***********************
    1 PICK AND PLACE ROBOT
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 IS VERSATILE
 -   ***********************> BOTT011•0F•FILE <*********************
```

The next command moves single lines of text.

You: Use the cursor positioning keys to move the cursor to the line command area of the screen. Position the cursor at line 3.
You: Enter: **M**

The next command identifies the line that the text is to follow.

You: Use the cursor positioning keys to move the cursor to the line-command area of the TOP•OP•FILE line.
You: Enter: **A**

System: Displays the screen:

```
- - - - - - - - - - - - - - - -07-08-1985- - - - - - - - - - - - - -
COMMAND INPUT --->
      ***********************> Top•u•FILE <***********************
    1 PICK AND PLACE ROBOT
    2 EASY TO PROGRAM
M 3 THE IBM MANUFACTURING SYSTEM
    4 IS VERSATILE
 -   ***********************> BOTTOIT•F•FILE <*********************
```

You: Press the enter (   ) key:

System: Screen refreshes with line 3 moved to line 1. The cursor is positioned on line 1. The line numbers of the screen are in numeric order.

The next command is used in pairs to move blocks of text.

You: Enter: **MM**
You: Use the cursor positioning keys to move the cursor to the line-command area of line 2.
You: Enter: **MM**
You: Use the cursor positioning keys to move the cursor to the line-command area of line 3.
You: Enter: **A**

System: Displays the screen:

```
— — — — — — — — — — — — —07-08-1985— — — — — — — — — — —
COMMAND INPUT --->
  -   **********************> Tu•oF•FILE <*******:or*:c***********
MM 1 THE IBM MANUFACTURING SYSTEM
MM 2 PICK AND PLACE ROBOT
A 3 EASY TO PROGRAM
    4 IS VERSATILE
  -   **********************> BoTTom•u•FILE <*******************
```

You: Press the enter (<—J) key:

System: Screen refreshes with the contents of lines 1 and 2 moved after line 3. The line numbers are in numeric order and the cursor is located on line 4.

```
— — — — — — — — — — — — —07-08-1985— — — — — — — — — — —
COMMAND INPUT --->
  -   **********************> Top•o•TILE <**********************
    1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
    4 IS VERSATILE
  -   **********************> BOTTom•u•FILE <*******************
```

The only difference between line command B and line command A is with line command B the text is moved before the line that is marked. Line command B can be used with M and MM.

The next instructions" move one line before another.

You: Enter: M on line number 4.
You: Use the cursor positioning keys to move the cursor to the line-command area of line 3.
You: Enter: B

System: Displays the screen:

```
 ------------------07-08-1985----------------
COMMAND INPUT --->
 - - **********************> Top•o•TILE <**********************
    1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
B 3 PICK AND PLACE ROBOT
M 4 IS VERSATILE
 - - ************AU*******> BOTTom•u•FILE <*******************
```

You: Press the enter (<---) key:

System: Screen refreshes with line 4 copied on line 3. The cursor is positioned on line number 4. The line numbers of the screen are in numeric order.

```
 ------------------07-08-1985----------------
COMMAND INPUT --->
      *********************> TOP•OP•FILE <**********************
    1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 IS VERSATILE
    4 PICK AND PLACE ROBOT
 - - *********************> BOTT014•01.•FILE <*******************
```

Repeat the last instructions for using line commands M and B to return lines to their previous positions.

Using Line Commands C, CC, with A or B (Copy with After or Before)

The C and CC line commands are copy commands. The A line command stands for "after " and the B line command stands for "before". In this exercise, you are copying text.

The cursor is located on line 4 at the start of the exercise. Your screen displays:

```
---------------07-08-1985---------------
COMMAND INPUT --->
 -   **********************> Tu•u•FILE <***********************
     1 EASY TO PROGRAM
     2 THE IBM MANUFACTURING SYSTEM
     3 PICK AND PLACE ROBOT
     4 IS VERSATILE
 -   **********************> BoTTom•N•FILE <********************
```

The next command copies single lines of text.

You: Use the cursor positioning keys to move the cursor to the line command area of line 1.
You: Enter: C

The next command identifies the line the text is to follow.

You: Use the cursor positioning keys to move the cursor to the line-command area of line 4.
You: Enter: **A**

System: Displays the screen:

```
---------------07-08-1985---------------
COMMAND INPUT --->
 -   **********************> Top•u•FILE <***********************
C 1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
A 4 IS VERSATILE
 -   **********************> BOTTom•u•FILE <*******************
```

You: Press the enter (<   ) key:

System: Screen refreshes with line 1 copied on line 5.
The cursor is positioned on line number 5. The line numbers of the screen are in numeric order. There are 5 lines on the screen.

The next command is used in pairs to copy blocks of text.

You: Move the cursor to the line command area of line 2.
You: Enter: **CC**
You: Use the cursor positioning keys to move the cursor to the line-command area of line 4.
You: Enter: **CC**
You: Use the cursor positioning keys to move the cursor to the line-command area of line 5.
You: Enter: **A**

System: Displays the screen:

```
---------------07-08-1985-------------
COMMAND INPUT --->
 - - ***********************> Top•o•TILE <************************
    1 EASY TO PROGRAM
CC 2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
CC 4 IS VERSATILE
A 5 EASY TO PROGRAM
 - - ***********************> BOTTom•u•FILE <*********************
```

You: Press the enter (   ) key:

System: Screen refreshes with the contents of lines 2 through 4 copied following line 5. The screen displays eight lines that are in numeric order. The cursor is positioned to the left of line number 6.

```
----------------07-08-1985------------
COMMAND INPUT --->
  -   *********************> Top•w•FILE <************************
      1 EASY TO PROGRAM
      2 THE IBM MANUFACTURING SYSTEM
      3 PICK AND PLACE ROBOT
      4 IS VERSATILE
      5 EASY TO PROGRAM
      6 THE IBM MANUFACTURING SYSTEM
      7 PICK AND PLACE ROBOT
      8 IS VERSATILE
  -   **********************> BorroppopITILE <********************
```

The line command **B** is used with the line command C the same way it was used with the line command M. The following instructions show you how to copy one line before another.

You: Enter: C on line number 6.
You: Move the cursor to line number 8 and enter: **B**.

System: Displays the screen:

```
----------------07-08-1985-------------
COMMAND INPUT --->
  -   **********************> TOP•O•TILE <***********************
      1 EASY TO PROGRAM
      2 THE IBM MANUFACTURING SYSTEM
      3 PICK AND PLACE ROBOT
      4 IS VERSATILE
      5 EASY TO PROGRAM
C  6 THE IBM MANUFACTURING SYSTEM
      7 PICK AND PLACE ROBOT
B  8 IS VERSATILE
  -   **********************> BOTTom•u•FILE <********************
```

You: Press the enter ( ) key:

System: Screen refreshes with 9 lines. Line number 6 is copied before line number 9. The numbers will be in numeric order with the cursor on line number 9. The screen displays:

```
— — — — — — — — — — — 07-08-1985 — — — — — — — — —
COMMAND INPUT --->
 - - ***********************> Top•u•FILE <***********************
    1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
    4 IS VERSATILE
    5 EASY TO PROGRAM
    6 THE IBM MANUFACTURING SYSTEM
    7 PICK AND PLACE ROBOT
    8 THE IBM MANUFACTURING SYSTEM
    9 IS VERSATILE
 - - ***********************> BOTTom•0F•FILE <*********************
```

To return the line numbers to their previous positions, delete line number 8.

You: Move the cursor to line number 8 and enter: **D.**
You: Press the enter ( ) key:

System: Screen refreshes with 8 lines and the cursor is on line number 8.

Using the Line Command R (Repeat)

The R line command repeats a line(s). You are repeating text in this exercise.

The cursor is located on line 8. Your screen displays:

```
---------------------------07-08-1985-----------------------
COMMAND INPUT --->
  -   ***********************> ToppopTILE <************************
    1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
    4 IS VERSATILE
    5 EASY TO PROGRAM
    6 THE IBM MANUFACTURING SYSTEM
    7 PICK AND PLACE ROBOT
    8 IS VERSATILE
  -   ***********************> Burrom•u•FILE <*********************
```

The next instruction repeats a single line.

You: Use the cursor positioning keys to move your cursor to the line-command area of line 1.
You: Enter: R

System: Displays the screen:

```
---------------------------07-08-1985-----------------------
COMMAND INPUT --->
  -   ***********************> Top•o•TILE <************************
R 1 EASY TO PROGRAM
    2 THE IBM MANUFACTURING SYSTEM
    3 PICK AND PLACE ROBOT
    4 IS VERSATILE
    5 EASY TO PROGRAM
    6 THE IBM MANUFACTURING SYSTEM
    7 PICK AND PLACE ROBOT
    8 IS VERSATILE
  -   ***********************> BoTTom•u•FILE <*********************
```

You: Press the enter (   ) key:

System: The screen blanks and then refreshes with line 1 repeated on line 2. All other lines move down one line number. The screen displays:

```
------------------07-08-1985-----------------
COMMAND INPUT --->
  -    *********************> TOPeOF•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
  -    *********************> BOTTOM•OF•FILE <********************
```

The **R** command can be followed with a single digit which instructs the editor to repeat the line that many times.

You: Enter: **R9**

System: Displays the screen:

```
------------------07-05-1985-----------------
COMMAND INPUT --->
       *********************> Top•OF.FILE <***********************
       EASY TO PROGRAM
R9 2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
  -    *********************> BOTTOMoOPTILE ********************
```

You: Press the enter (<--J) key:

System: Screen blanks and then refreshes with line 2 repeated 9 times.


```
------------------------07-08-1985--------------------
COMMAND INPUT --->
 -    **********************> Top•oF•FILE <**********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 EASY TO PROGRAM
     4 EASY TO PROGRAM
     5 EASY TO PROGRAM
     6 EASY TO PROGRAM
     7 EASY TO PROGRAM
     8 EASY TO PROGRAM
     9 EASY TO PROGRAM
    10 EASY TO PROGRAM
    11 EASY TO PROGRAM
    12 THE IBM MANUFACTURING SYSTEM
    13 PICK AND PLACE ROBOT
    14 IS VERSATILE
    15 EASY TO PROGRAM
    16 THE IBM MANUFACTURING SYSTEM
    17 PICK AND PLACE ROBOT
    18 IS VERSATILE
 -    **********************> BOTTOM•O•FILE <*******************
```

The next instructions delete a few lines of the text so the exercise is easier to follow.

You: Enter **DD** on line 3.
You: Move the cursor to line number 11 and type a block delete command: **DD**

System: Displays the screen:

```
- - - - - - - - - - - - 07-08-1985 - - - - - - - - - -
COMMAND INPUT --->
 - - **********************> Top•oF•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
DD   3 EASY TO PROGRAM
     4 EASY TO PROGRAM
     5 EASY TO PROGRAM
     6 EASY TO PROGRAM
     7 EASY TO PROGRAM
     8 EASY TO PROGRAM
     9 EASY TO PROGRAM
    10 EASY TO PROGRAM
DD  11 EASY TO PROGRAM
    12 THE IBM MANUFACTURING SYSTEM
    13 PICK AND PLACE ROBOT
    14 IS VERSATILE
    15 EASY TO PROGRAM
    16 THE IBM MANUFACTURING SYSTEM
    17 PICK AND PLACE ROBOT
    18 IS VERSATILE
 - -**********************> Barrom•u•FILE <********************
```

You: Press the enter (   ) key:

System: Displays the screen:

```
- - - - - - - - - - - - 07-08-1985 - - - - - - - - - -
COMMAND INPUT --->
 - - **********************> Top•OF•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 - - **********************> myrnm•op•TILE <*******************
```

INFORMATION ABOUT PRIMARY COMMANDS

All primary commands are entered on the editor screen following **COMMAND INPUT** --->, as shown in the figure at the beginning of this chapter. Primary commands provide two types of functions: text manipulation and disk management.

The two functions have commands that do the following:

- Text manipulation
  - Find occurrences of characters
  - Change occurrences of characters
  - Position a particular line number on the top line of the editor
- Disk management
  - Save the current editor file without ending the session
  - List file names from a diskette or fixed disk
  - Recall last primary command executed
  - Rename a specified file
  - Get and insert a file into a specified file
  - Extract parts of text and place into a specified file
  - Print a specified file
  - Delete a specified file
  - End the editing session without saving the file by cancelling the current edit session

Primary commands are executed after pressing the enter key or any function key. The **Home** key moves your cursor to the primary command line from other areas of the screen.

Primary Command	Description
CANCEL	Cancels current file and ends session
CAPS	Change character from lower to upper case
CHANGE	Changes character strings in current file
DEL	Deletes a file
FILES	Displays names and types of files
FIND	Finds character strings in current file
GETFILE	Gets and inserts files into current file
LOCATE	Locates (positions) a line number at top line
PRINT	Prints contents of an entire file
PUTFILE	Puts all or part of a file into current file
RENAME	Renames a file
SAVE	Saves current file without ending session
	Recalls last primary command executed

Using the Primary Command FIND

This primary command allows you to search for strings that contain embedded blanks. The leading character of the search string is an optional / (slash) character. In that case, the slash represents the delimiter of the string.

Syntax for the delimited **FIND** command is outlined below.

```
F /string/ [col-1] [col-2]
```

- The only parameter required is the string preceded by the delimiter character, either a blank or a / (slash).
- The string is terminated by a second / (slash) or the last non-blank character.
- A blank character must follow the FIND or F command.
- Additional parameters, if any, must be separated by at least one blank.
- The blank space between the last delimiter character and col-1 is optional.
- The columns are counted beginning with the first character to the right of the line numbers.
- If you enter the col-1 parameter without the col-2 parameter, the search begins in the first column specified and continues to the end of the line. Col-2 can not be specified without co•-1 also being specified.
- If you specify additional parameters, the second / (slash) must be specified.

The **FIND** command finds the next occurrence of a character string within your AML/Entry program. The command may be abbreviated as shown in this exercise.

The search always begins on the top line of the display and proceeds towards the end of the file. If you are searching for a string that is on a line before the line at the top of the screen, the string is not found.

The optional col-1 parameter specifies the a beginning column for the search. The optional col-2 parameter specifies an ending column for the search.

Your screen displays:

```
----------------07-08-1985-------------
COMMAND INPUT --->
 -   **********************> T013•0F•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> BOTTom•u•FILE <********************
```

You: Press the **Home** key.

System: Cursor moves to the command input line.

The next instruction finds the string "ROBOT" in the text. The blank space is required between the command and the string.

You: Enter: **F ROBOT**

System: Displays the screen:

```
----------------07-08-1985-------------
COMMAND INPUT ---> F ROBOT
 -   **********************> Tu•0F•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> imiu•F•FILE <********************
```

You: Press the enter (   ) key:

System: Screen blanks and then line 4, which contained the character string ROBOT, is displayed at the top of the screen window. The cursor is located under the first character of the string that was found. A message flashes in the upper right of the screen to indicate that the string was found. The screen displays:

```
——————————————07-08-1985 - - - - STRING FOUND--
COMMAND INPUT --->
 -   **********************> ne•OF•FILE <************************
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> BOTTom•u•FILE <*********************
```

You: Clear the flashing message by pressing the **F2** key.
(It disappears the next time the enter key is pressed, also.)

You: Press the **Home** key.

System: Cursor moves to command input line.

The next instruction finds the string "RO" while restricting the search between columns 10 and 15 of the display.

You: Enter **F RO 10 15**

System: Displays the screen:

```
——————————————07-08-1985 - - - - STRING FOUND---
COMMAND INPUT ---> F RO 10 15
 -   **********************> Top•u•FILE <************************
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> BOTTom•u•FILE <*********************
```

You: Press the enter ( ) key:

System: Computer searches for the string RO located within columns 10 through 15. It skips over the string RO contained in the word ROBOT in line 4 at the top of the screen because this RO is in columns 16 through 17. The columns are counted beginning with the first character to the right of the line numbers. The first occurrence of string RO is found within columns 10 through 15 of line 6 as part of the word "PROGRAM."

Using the function key in the next step brings the editor screen to the top of the program.

You: Press: F9

System: Screen scrolls to display the TOP•0F•FILE line and places the cursor on the primary command line.

The next key repeats the search for the string RO in columns 10 through 15.

You: Press: F4

System: The computer repeats the search for the string RO and locates the string in line 1. The screen displays:

```
...........................07-08-1985  - - - -  STRING FOUND---
COMMAND INPUT --->
 - -  ***********************> TomR.FILE t***********************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 - -  ***********************> BOTTom•a•FILE <********************
```

You: Press: F9

System: Screen scrolls to display the TOP•O•TILE line and places the cursor on the primary command line.

Using the Primary Command CHANGE

This primary command allows you to search for strings that contain embedded blanks, and change the search string to a new string that may contain embedded blanks. The leading character of the search string is an optional / (slash) character. In that case, the optional / (slash) represents the delimiter of the string.

Syntax for the delimited **CHANGE** command is outlined below.

```
C /stringl/string2/ [col-1] [col-2] [ALL]
```

- The required parameters are stringl and string2, preceded and followed by the delimiter character.
- There must be three delimiter characters present. (Stringl and string2 may not contain the slash character.)
- A blank character must follow the CHANGE or C command.
- Additional parameters, if any, must be separated by at least one blank.
- The blank space between the last delimiter character and col-1 is optional.
- The columns are counted beginning with the first character to the right of the line numbers.
- If you enter the col-1 parameter without the col-2 parameter, the search begins in the first column specified and continues to the end of the line. col-2 can not be specified without col-1 also being specified. If both col-1 and col-2 are specified, the search takes place between the specified columns only.
- The ALL parameter may be specified alone or with the column parameters. The ALL parameter specifies that every occurrence of the search string will be changed if it falls between col-1 and col-2.

The **CHANGE** command is used to find and change the next occurrence of a character string within your AML/Entry program. The command may be abbreviated as C, as shown in the following exercise.

This exercise changes characters in this screen:

```
---------------------07-08-1985-----------------------
COMMAND INPUT --->
 -    **********************> Tu•OF•FILE <************************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -    *************AA********> BOTTom•OF•FILE <*********************
```

The next step contains the command, the string to be changed, and the desired new string. Spaces are required between the three items.

You: Enter: **C VERSATILE FAST**

System: Displays the screen:

```
 - - - - - - - - - - - - - 07-08-1985 - - - - - - - - - - -
COMMAND INPUT ---> C VERSATILE FAST
 -    **********************> Tcp•u•FILE <********-.4**************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -    **********************> BOTTOM• OF •FILE <***************     *
```

You: Press the enter (   ) key:

System: The STRING FOUND message is displayed. The first occurrence of the string VERSATILE is changed to the string FAST on line number 5. The cursor is positioned after the last character of the changed string.

The next step repeats the change command.

You: Press: **F5.**

System: Repeats change for the next occurrence of the string, which is on line 9.

You: Press: **F9.**

System: Screen scrolls to display the TOP.OF.FILE line. The cursor is located on the command input line.

The next step changes all occurrences of the string FAST to the string VERSATILE.

You: Enter: **C FAST VERSATILE ALL**

System: Displays the screen:

```
 - - - - - - - - - - - - - - -  07-08-1985  - - - - - - - - - - - -
COMMAND INPUT ---> C FAST VERSATILE ALL
 - - **********************> TOP•OF•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS FAST
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS FAST
 - - **********************> Barrom•u•FILE <********************
```

You: Press the enter (   ) key:

System: The screen blanks and the message **2 CHANGES** is displayed. Line 9 is displayed (last occurrence) and the cursor is located after the last character of the changed string. The screen displays:

```
 - - - - - - - - - - - - - - 07-08-1985 - - - - 2 CHANGES - -
COMMAND INPUT ---> C FAST VERSATILE ALL
.... *********************> Topp0F.FILE <**********************
   9 IS VERSATILE
 - - *********************> BOTT014.0F.FILE <*******************
```

You: Press: F9

System: Screen blanks and refreshes without flashing the message. Screen is scrolled to the top of the program. The cursor is on the primary command line.

The C primary command also accepts column parameters after the new string characters. The format for entering all the options for the C command is:

```
C stringl string2 [coll] [col2] [ALL]
```

The items in brackets are optional.

The next instructions change the string "SY" to "SI", for all occurrences, while restricting the search between columns 3 and 4 of the display. The columns are counted beginning with the first character to the right of the line numbers.

You: Enter **C SY SI 3 4 ALL**

System: Displays the screen:

```
 - - - - - - - - - - - - - - ,07-08-1985 - - - - - - - - - - -
COMMAND INPUT ---> C SY SI 3 4 ALL
 - - *********************> Tu•u•FILE <**********************
   1 EASY TO PROGRAM
   2 EASY TO PROGRAM
   3 THE IBM MANUFACTURING SYSTEM
   4 PICK AND PLACE ROBOT
   5 IS VERSATILE
   6 EASY TO PROGRAM
   7 THE IBM MANUFACTURING SYSTEM
   8 PICK AND PLACE ROBOT
   9 IS VERSATILE
 - - *********************> Burrom•u•FILE <*******************
```

You: Press the **F5** key.

System: Computer searches for all occurrences of the string SY located in columns 3 and 4. It skips over the string SY contained in the word SYSTEM in lines 3 and 7 because the string SY is in columns 23 and 24. The screen blanks and the message **3 CHANGES** is displayed. This message indicates string SY has been changed to to string SI in all occurrences in columns 3 and 4. Line 6 is is displayed (last occurrence) and the cursor is located after the last character of the changed string. The screen displays:

```
- - - - - - - - - - - - - - - - 07-08-1985 - - - - 3 CHANGES - -
COMMAND INPUT --->
 - - ***********************> Tcw•u•FILE <************************
    6 EASI TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 - - ***********************> BOTTom•u•FILE <*********************
```

Using the function key in the next step brings the editor screen to the top of the program.

You: Press: **F9**

System: Screen scrolls to display line number 1. The cursor is on the primary command line and all changes are displayed. The screen displays:

```
------------------------------07-08-1985-------------------------
COMMAND INPUT --->
 -   ***********************> n•o•TILE <************************
    1 EASI TO PROGRAM
    2 EASI TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASI TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -   **************AA*****> BOTT01.1•0F•FILE <*********************
```

Repeat the previous exercise for primary command C with column parameters to return the string SI to the string SY.

Using the Primary Command LOCATE

The **LOCATE** command puts the indicated line at the top of the program window. This command may be abbreviated **L**, as shown in this exercise.

The screen displays:

```
- - - - - - - - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT --->
 -    **********************> Top•u•FILE .c***********************
      1 EASY TO PROGRAM
      2 EASY TO PROGRAM
      3 THE IBM MANUFACTURING SYSTEM
      4 PICK AND PLACE ROBOT
      5 IS VERSATILE
      6 EASY TO PROGRAM
      7 THE IBM MANUFACTURING SYSTEM
      8 PICK AND PLACE ROBOT
      9 IS VERSATILE
...   **********************> BOTTOM•NN•TILE <********************
```

In this exercise, line 8 is to be located.

You: Enter: **L 8**

System: Displays the screen:

```
- - - - - - - - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT ---> L 8
 -    **********************> Top•co•TILE <***********************
      1 EASY TO PROGRAM
      2 EASY TO PROGRAM
      3 THE IBM MANUFACTURING SYSTEM
      4 PICK AND PLACE ROBOT
      5 IS VERSATILE
      6 EASY TO PROGRAM
      7 THE IBM MANUFACTURING SYSTEM
      8 PICK AND PLACE ROBOT
      9 IS VERSATILE
 -    **********************> BOTTom•u•FILE <********************
```

You: Press the enter ( ) key:

System: Screen blanks and refreshes with line 8 at the top of the window. The cursor is located in the line command area of line 8.

```
 - - - - - - - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT --->
      **********************> Top.u.FILE.<***********************
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 - - ******************t***> RoTTom•u•FILE <********************
```

You: Press: **Home**

The next instructions show that the command locates a line toward the top of the program.

You: Enter: **L 1** and press the enter key ( ).

System: Screen displays line 1.

You: Press: **F9**

System: Screen displays the TOP•OF•FILE line and the cursor is on the primary command line.

Using the Primary Command SAVE

The **SAVE** command stores the present editor information onto a diskette. Use this command often while updating or creating a program to protect the information in the program from inadvertent loss caused by power loss.

```
- - - - - - - - - - - - - - - -07-08-1985- - - - - - - - - - - - -
COMMAND INPUT --->
 -    **********************> Tu•o•TILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    **********************> BOTTom•u•FILE <********************
```

The next steps are used for storing unnamed programs.
The program is called stations.

You: Enter: **SAVE STATION1**

System: Displays the screen:

```
 - - - - - - - - - - - - - - 07-08-1985  - - - - - - - - - - -
COMMAND INPUT ---> SAVE STATION1
 -    **********************> ma•u•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    ****  _***************> Barra•o•TILE <********************
```

You: Press the enter (   ) key:

System: Drive A in-use LED comes on (if the file is being saved on drive A). The screen refreshes after the file has been saved and displays the program name in the upper left corner.

```
--STATION1.AML - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT --->
 - - ***********************> Top•o•ILE <************************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 - - ***********************> BOTT014•0F•FILE <*********************
```

If you are saving a previously named program, you would enter the command **SAVE** and press the enter key.

Using the Primary Command FILES

The **FILES** command displays names and types of file(s) that are stored on the current diskette. This exercise shows you how to display files when you are using the editor.

The screen displays:

```
--STATION1.AML - - - - - -  07-08-1985. - - - - - - - - - -
COMMAND INPUT --->
 -    ***********************> Top.•0F•FILE <************************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM •
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -    ***********************> BOTTom•u•FILE <*********************
```

You: Type: **FILES**

System: Displays the screen:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT ---> FILES
 -    ***********************> Tu•u•FILE <************************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -    ***********************> BOTTom•u•FILE <*********************
```

You: Press the enter (   ) key:

System: Screen displays a listing of the files on the current diskette. The list should be similar to the following display:

```
COMMAND .COM PRACTICE .AML MENU  .EXE EDIT  .EXE CFG  .EXE
MSGCFG .TXT MSGZED1 .TXT MSGZED2 .TXT AUTOEXEC.BAT STATION1.AML
COMAID .EXE MSGCMP .TXT MSGCOM2 .TXT MSGMENU .TXT COMPILER.EXE
MSGCOM .TXT CONFIG .SYS
```

You: Press any key on the keyboard:

System: Screen blanks and then refreshes; it displays the editor. The cursor is located on the command input line.

The next instructions show the method used to display only certain types of files on a specific drive.

You: Enter: **FILES *B:*.AML***
You: Press the enter (   ) key.

System: Screen displays AML type files.

You: Press any key on the keyboard:

System: Screen blanks and then refreshes; it displays the editor. The cursor is located on the command input line.

Using. the Primary Command ? (Recall)

The ? primary command is used in the command input area to display or recall the last primary command executed.

The screen displays:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT --->
 -   ***********************> Top•u•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> Bgrag•u•FILE <********************
```

The next steps are used for recalling the primary command FILES, which was just executed, by using the tj primary command ?.

You: Enter: ? character.

System: Displays the screen:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT ---> ?
 -   **********************> Top•u•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   **********************> BoTTom•u•FILE <********************
```

You: Press the enter ( ) key.

System: Screen displays the FILES B:*.AML command on the primary command line.

The ?, as a DOS command, can be used in conjunction with the primary command FILES to locate certain types of files. A ? in a filename or extension means that any character can be in that position. All files that have a name that matches in all except the ? position are selected.

You: Enter: **FILES STATION? .AML**
You: Press the enter ( ) key.

System: Screen displays all STATION files with an extension of .AML.

Using the Primary Command RENAME

The **RENAME** command renames files while you are working in the editor. When renaming files, you must also include the type of file (in this case, **.AML)** if no device type is entered the default device is assumed. In this exercise, the file stationl is renamed station2.

The system displays this screen when you start:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - - -
COMMAND INPUT --->
-     **********************> Top.opTILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    **********************> BoTTom•01•FILE <********************
```

You: Enter: **RENAME STATION1.AML STATION2.AML**

System: Displays the screen:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - - -
COMMAND INPUT ---> RENAME STATION1.AML STATION2.AML
 -    **********************> 1.013•0F•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    **********************> BoTTom•op•FILE <********************
```

You: Press the enter (   ) key:

System: Screen refreshes with the editor displayed. The program name has not changed at the top of the screen.

Note: When you use the key **F8** to exit the program, the program is saved under the name appearing at the top left of the screen. If you load a file into the editor and then rename the file, the program is saved under the old name, and <u>not</u> under the new name when you exit using **F8** .

Using the Primary Command GETFILE

This primary command allows you to insert other files into the file currently being edited. Syntax of the command is outlined below.

GETFILE filename

In addition to filename (if no extension is entered, the Editor assumes .AML), you must specify where the included text is to be placed. This is accomplished by entering an A (after) or a B (before) in the line command area. A or B must be specified before the GETFILE command is entered on the primary command line and the <--J(enter) key is pressed. If either A or B is not specified when GETFILE is entered, an error results and the command must be re-entered (unless, the ? command is used to recall the command). Errors are also issued for an improper filename, file not found, or filename not specified. In all cases, whatever was entered in the line command area remains both visible and active.

If device type is specified the default drive is assumed.

An example GETFILE command is outlined below.

```
--STATION1.AML - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT ---> GETFILE STATION2.AML
 - - **********************> Top•u•FILE <***********************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
A 3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 - - **********************> BoTTom•u•FILE <********************
```

This example copies the contents of the STATION2.AML file into the STATION1.AML file after line 3.

Using the Primary Command PUTFILE

This primary command allows you to extract all or part of the currently edited text and place it into a file on disk. Syntax of the command is outlined below.

PUTFILE filename

In addition to filename (if no extension is entered, the Editor assumes .AML), you must specify which lines of text currently being edited are to be extracted and placed into the new file. This is accomplished by entering the CC or C line command in the line command area. If you want to extract a single line, the C must be placed adjacent to that line before entering a PUTFILE command. If a block of lines are to be extracted, specify the CC command in the appropriate locations.

C or a pair of CC identifiers must be specified before the PUTFILE command is entered on the primary command line and the <-' (enter) key is pressed. If either C or CC is not specified when PUTFILE is entered, an error results and the command must be re-entered (unless, the ? command is used to recall the command). Error messages are also issued for an improper filename or for the name of a file that already exists.

You can not place text into a file that already exists. Appending to an existing file is not allowed.

An example PUTFILE command is outlined below.

```
--STATION1.AML - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT ---> PUTFILE STATION2.AML
 - - **********************> Top•r•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
CC 3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
CC 5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 - - **********************> BOTTom•u•FILE <********************
```

This example copies the marked contents (lines 3 through 5) of the STATION1.AML file into a new file labelled STATION2.AML.

Commands That Allow Expressions

Expressions are allowed in the following AML/E commands:

- COMPC(exp < exp,label);
- SETC(counter,exp);
- TESTC(exp,exp,label);
- ZMOVE(exp);

Using expressions and functions can make AML/E programs very efficient. For example, applications should use the arithmetic function form of CSTATUS and MSTATUS instead of the command form, as shown below:

```
COMPC(MSTATUS()=0,CONTINUE);
```

Using the command form would require the declaration of a counter, the MSTATUS command, and then finally the COMPC command. Three AML/Entry lines have been replaced by one.

Many other AML/Entry commands (i.e. GUARDI, LINEAR, PAYLOAD, ZONE, to name a few) do not allow expressions as arguments. They do, however, allow counters as arguments. An alternate solution is to use the SETC command to set a counter, and then use the counter in these commands.

Expressions are not allowed within the ITERATE statement. Even though most of the above commands may appear in an ITERATE statement, they must **appear** "stand-alone" for expressions to be legal. When these verbs are used inside an ITERATE statement, the arguments that are unpacked from the aggregates and actually used depend on the command. See the discussion of SETC, TESTC, and ZMOVE in Appendix A, "Command/Keyword Reference."

You: Press the enter (   ) key:

System: Screen displays **CURRENTLY PRINTING** and the printer starts the printing operation. Printer output appears as follows:

```
STATION1.AML                                              07-08-85
                                                            PAGE 1

    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
    3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
    5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
```

System: Displays PRINT COMPLETE message when finished.

Note: The Personal Computer must be hooked up to a printer that is online. If the printer is not present and running, then there is a 20-30 second time period where nothing happens, after which the message "UNABLE TO PRINT" appears.

Using the Primary Command DEL (Delete)

The **DEL** command deletes a file from the diskette. THIS COMMAND SHOULD NOT BE USED TO DELETE ANY OF THE AML/E SYSTEM FILES.

In the following exercise, you are deleting the file STATION2.AML from your diskette. The screen displays:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT --->
 -   .**********************> Top•0F•FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    **********************> Burrom•0F•FILE <********************
```

You: Enter: **DEL STATION2.AML**

System: Displays the screen:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT ---> DEL STATION2.AML
 -    *********************> Tu.u.FILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE .IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -    **********************> BOTTom•01-•FILE <********************
```

You: Press the enter (   ) key:

System: Drive in-use LED comes on. After processing is complete, the screen displays the contents of the editor. In this case, STATION1 is the program in the editor.

Using the Primary Command CANCEL

The **CANCEL** command lets you exit the editor without saving a file copy from the edit session.

The screen displays the following at the beginning of this exercise:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT --->
 -   *********************> Top•op•pILE <**********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   *********************> BoTTom•op•pILE <*******************
```

You: Enter: **CANCEL**

System: Displays the screen:

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - -
COMMAND INPUT ---> CANCEL
 -   *********************> Top•op•pILE <**********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
     3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 -   *********************> 'mm4.01..1.111 <*******************
```

Your Press the enter (   ) key:

System: The screen blanks and then displays the main menu. The Personal Computer has erased the file that was in the editor from memory.

Using the Primary Command CAPS

The **CAPS** command converts any lower case alphabetic characters to uppercase alphabetic characters. The AML/Entry editor only allows characters to be entered in upper case. However the user may use other techniques to enter data into a file, which may cause lower case characters to be entered. By using the **CAPS** command to convert all the alphabetic characters to uppercase, the FIND and CHANGE commands can then be used.

CHAPTER 4. LEARNING THE AML/ENTRY LANGUAGE

This chapter introduces you to the AML/Entry language. It is important that you read this chapter before you attempt to build an application program. This is not intended to be a reference chapter. Inst,ead, it is a teaching chapter. You will first be taught simple commands to make the robot move and then be given more difficult instructions to enhance your program application. The information listed below is included in this chapter.

- Structural overview of the AML/Entry application program
- Rules for usage and examples of AML/Entry reserved words and commands
- Techniques to help simplify programming
  - Declarations
  - Constants
  - Variables
  - Expressions
  - Built-in arithmetic functions
  - Compiler directives
- Usage of Subroutines
- Advanced topics for program enhancement
  - Pallets
  - Regions
  - Host Communications

Note: In this chapter, the AML/Entry program examples are, except where marked, written for use on a manipulator with a Home position of (650,0,0,0). If the Home position on your manipulator is different, you may need to change the points in the examples to produce valid results.

A STRUCTURAL OVERVIEW

The following sections deal with basic information essential to all AML/Entry programs.

Language Structure

AML/Entry is a ,subroutine-oriented language. A subroutine is a small unit of a proriantith a 61171073 defined beginning and ending. The statements that form a subroutine usually relate logically to achieve a result, such as moving the arm.

Within your user subroutines are AML/Entry statements, which send ins the • s ML/Entry statements look like English-language commands. Some of these commands may have additional information enclosed in parentheses, such as names, .Opnditions for decisions, or values or expressions for'control. Some AML/Entry statements do not perform any visible action. These statements perform functions such as reserving storage for your program's variables, names, and constants.

Your Application Program in the Controller

The IBM Personal Computer converts AML/Entry statements into a form suitable for the controller. The converted program sent to the controller is stored in one of five memory partitions. These partitions vary in size according to the program size. It is possible for one large program to use all the memory available in the controller.

When the program is selected and run by the controller, the manipulator performs the sequence of actions dictated by the program. After the last statement is executed, the controller starts over at the beginning of the application program. This sequence is ieIietit6d tnrt ti operator intervention or an error condition stops the program.

When you transfer a program to the controller from the Personal Computer, it remains in the assigned memory partition until you send another application program to that same partition or use the Unload function.

Comments

Comments are very helpful in your program. They are useful in trying to understand or update your program. They have no effect on program execution whatsoever, and should be seen only as a programming aid. The comment is preceded by a double hyphen 1--41,,,and is written on the remainder of the line. The double hyphen indicates to the software that the remainder of line contains your comments and should be ignored.

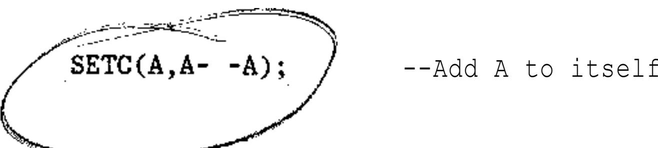
An example of a comment used with the AML/Entry SUBR statement:

```
STATION1:$UBR; --STATION1 ASSEMBLES GEARS
```

Note: AML/Entry Version 4 also allows expressions inside certain commands. Expressions can consist of unary minuses. The reader must not forget that two consecutive dashes (-) indicates the start of a comment. Thus the following will cause an error:

```
SETC(A,A--A); -- Add A to itself
```

This is because the first two dashes are not treated as a minus followed by a unary minus, but instead as the start of the comment. To work correctly, a space must be included between the first two dashes:



--Add A to itself

Beginning and Ending Your Program

Notice the TOP•OF•FILE and BOTTOM•0F•FILE lines in the editor. These lines are not a part of the application program and are not saved in the program file. To start your program in the editor, use the insert line command to open a window between the TOP•OP•TILE and BOTTOM•OP•FILE lines.

Two commands are required for every application program. The first is an id: BR; statement. This is your outer or main subroutine; the ID is a en ifier. The second is an laiw statement for the outer subroutine. In the below outlined example, the END statement is shown on line 2. By using the editor's I line command, lines are inserted between the SUER and END lines. Thus the END line stays at the end of the application program as more lines are inserted.

A description of the characteristics of AML/Entry statements follows the outlined example.

```
 _Line number
 | _Identifier
 |  I       _ Definition operator
 |  |
 +  +      +
1 MAIN   :SUBR; --BEGINNING OF SUBROUTINE

           |  |    |  Comments
              I__ Statement delimiter
           |   Keyword for subroutine


   Line number
 |       Keyword
 |   |
 +   +
2 END;          --END OF SUBROUTINE

                |  Comments
     |   Statement delimiter
```

Line Number

Line numbers are used as reference guides, enabling you to refer to specific lines of the program quickly. The numbers are assigned sequentially by the editor as you insert, delete, copy or move lines.

Identifier

MAIN is an identifier, which is the name for the program. Identifier rules are listed below.

- Up to 72 characters
- First character must be alphabetic
- Remaining characters must be either alphabetic, numeric, or the underscore () character
- An underscore can not be the last character
- Special characters, such as an asterisk (*), are not permitted

Examples of valid identifiers are outlined below.

```
STATION1:SUBR; or STATION 1:SUBR;
```

Examples of invalid use are outlined below.

```
1STATION:SUBR; or STATION*1:SUBR;
```

Definition Operator

The colon (:) is a definition operator. The colon is a required separator between the identifier and the keyword command.

Keyword/Command

SUBR is an AML/Entry keyword that identifies the subroutine. Keywords are reserved words that have special meaning in the language. A list of all keywords is provided in this chapter. A detailed section on subroutines is found in the latter part of this chapter.

END is a keyword, that is always the last line of each subroutine., You must use a semicolon after each END keyword. No other AML/Entry statement is permitted on the END statement line (after the END statement).

Statement Delimiter

The semicolon (;) is a delimiter that indicates the information for the statement is complete. You must use a semicolon at the end of each AML/Entry statement.

AML/ENTRY RESERVED WORDS AND COMMANDS

The following sections are guidelines on using AML/Entry. The first section contains two lists of reserved words with brief definitions. The second section describes, in detail, several command words. Descriptions of some of the command words will be incorporated in later sections.

AML/Entry Reserved Words

Certain words have special meaning in AML/Entry language. These words are called reserved words.

Your program must not attempt to use any of the reserved words for other than the intended use. For example, your program can not create a subroutine named **PT** because this name is already a reserved word.

The reserved words include kezozlsand c22pAgdL. Keywords are used to defirir—rrriTOrThr"the computer. Commands cause an action by the manipulator or controller.

The following reserved word list contains keywords and their descriptions. The next reserved word list describes commands.

Keyword	Use	Description
ABS	Expressions	Returns absolute value of arg
ATAN	Expressions	Returns arc tangent of arg
ATAN2	Expressions	Returns arc tangent of args
COS	Expressions	Returns cosine of arg
COUNTER	Counter	Defines counter name
CSTATUS	Expressions	Returns communications status
END	End	Ends subroutine
FROMPT	Expressions	Returns coordinate of point
GROUP	The Group	Groups related data items
MSTATUS	Expressions	Returns motion status
NEW	Constant	Identifies constants
PALLET	Pallet	Defines pallet name
PT	Point	Defines a point data type
REGION	Region	Defines a region name
SIN	Expressions	Returns sine of arg
SQRT	Expressions	Returns square root of arg
STATIC	Storage	Defines counter, pallet, and group storage
SUBR	Subroutine	Identifies a subroutine
TAN	Expressions	Returns tangent of arg
TESTI	Expressions	Returns state of Digital Input
TESTI'	Expressions	Returns current part of pallet
TRUNC	Expressions	Returns greatest integer <= arg

Command	Use	Description
BRANCH	Flow/Control	Transfers control
BREAKPOINT	Flow/Control	Predetermined stop in program
COMP0.,	Flow/Control	Compares counters
ICSTATUS	Sensor	Controller enabled status
DECR	Counter	Decrements counter by 1
DELAY	Motion	Delays statement execution
DPMOVE	Motion	Moves arm specified distance
GET	Host ommunication	Controller requires data
GETPART	Palletizing	Moves arm to current part
GRASP	Motion	Closes the gripper
GUARDI	Sensor	Guards motion (uses DI port)
INCR	Counter	Increments counter by 1
ITERATE	Flow/Control	Repeats execution
LINEAR	Motion	Changes motion (arc-straight)
4 MSTATUS	Sensor	Move completion status
NEXTPART	Palletizing	Increases current part by 1
ANOGUARD	Sensor	Stops the guard feature
• PAYLOAD	Motion	Controls tool-tip speed
PREVPART	Palletizing	Decrease current part by 1
PMOVE	Motion	Moves arm specified location
PUT	Host/Communication	Controller wants to send data
RELEASE	Motion	Opens gripper
SETC	Counter	Sets counter
SETPART	Palletizing	Sets current part
TESTC	Counter	Tests counter
TESTI	Sensor	Test DI point
TESTP	Palletizing	Test current part counter
WAITI	Sensor	Waits for DI point switch
WHERE	Sensor	Updates a point definition
WRITEO	Sensor	Opens or closes a DO relay
XMOVE	Motion	Regional movement
ZMOVE	Motion	Absolute Z-axis movement
ZONE	Motion	Changes the settle

The following section gives detailed descriptions of several of the command words. will help you to become familiar with manipulator movements. The section begins with Motion Commands.

MOTION Commands

The commands described in this section affect manipulator motion. These motion commands are English-like words that cause an action. Also described in this section is the DELAY command. The DELAY command is not a motion command, but it is described in this section because it is primarily used with motion commands. If a command uses any parameters, the parameters follow the command and are enclosed in parentheses.

Command	Description
DELAY	Delays statement execution
DPMOVE	Moves arm specified distance
GETPART	Movement within a pallet
GRASP	Closes the gripper
LEFT	Changes arm mode (7545-800S only)
LINEAR	Changes motion (arc-straight)
PAYLOAD	Controls tool-tip speed
PMOVE	Moves arm specified location
RELEASE	Opens gripper (DO 2 switch)
RIGHT	Changes arm mode (7545-800S only)
XMOVE	Regional movement
ZMOVE	Absolute Z-axis movement
ZONE	Changes the settle

DELAY Command

```
DELAY(seconds);
```

The DELAY command suspends execution of the program for the time (in seconds) specified as a parameter. Delays are usually needed after every GRASP or RELEASE command to allow the mechanical motion of the gripper to complete. Delays can range from 0 to 25.5 seconds, in increments of 0.1 seconds. The seconds can consist of a simple constant or a counter. Examples of the command's usage are outlined below.

```
DELAY(2);                -- Causes a delay of 2 seconds
DELAY(CTRA);             -- Causes a delay of CTRA seconds
```

DPMOVE Command

```
DPMOVE(vector);
```

The DPMOVE command moves the arm in the direction of the specified vector. It is useful for small moves from a location obtained using one of the other move commands, such as PMOVE, or the pallet move command GETPART.

The vector contains changes for the X, Y, Z, and R-axes coordinates. Commands in AML/Entry appear as:

```
DPMOVE(<AX,AY,AZ,AR>);
```

Note: In this chapter the term "delta" or "A" refers to the difference between two positions.

An example of the changes in the coordinates is explained below.

Example: The arm is located at the coordinates of X=300, Y=450, Z=-150 and R=90°. A part is needed from the coordinates X=310, Y=425, Z=-160, and R=25 . The delta for X is 10, which is the difference between 300 and 310. The delta for Y is -25, which is the difference between 450 and the 425. The delta for Z is -10, which is the difference between -150 and -160. The change in roll is -65 , which is the difference between the 90 and the 25 . The command in the program appears as:

```
DPMOVE(<10,-25,-10,-65>);
```

GETPART Command

```
GETPART(pallet_name);
```

The GETPART command allows movement within a pallet. GETPART will be discussed later in the chapter under the section on pallets.

GRASP Command

```
GRASP;
```

The GRASP command closes the pQ__2relu, which allows air to flow through the Center of the Z-axis shaft. If you have a gripper that is controlled by the Z-axis shaft air supply, the gripper closes. A DELAY command may be necessary to allow the gripper to close.

LEFT Command (Valid on 7545-800S Only)

```
LEFT;
```

The LEFT command instructs the 7545-800S manipulator to switch to the left arm mode. Once this command is executed, all future motions will be performed with the 7545-800S manipulator in left arm mode. This command is only valid for the 7545-800S manipulator because it is the only supported manipulator which has a symmetric elbow (theta two) joint.

The following figure shows the workspace of a 7545-800S (a 7545 with RPQ R00100 installed). Points designated by a "*" or a flop, can be reached in either left or right mode, points designated by an "R" or an "L" can only be reached in right or left mode, respectively.

```
                 *oooooooooooo*
             *oooooooooooooooooooo*
          *oooooooooooooooooooooooooo*
        *oooooooooooooooooooooooooooooo*
      *oooooooooooooooooooooooooooooooooo*
     *oooooooooooooooooooooooooooooooooooo*
    *ooooooooooooo***********oooooooooooooo*
   *oooooooooooo***            **ooooooooooooo*
  *ooooooooooooo*                *ϕoooooooooooo*
   *ooooooRRRRRR       ^          LLLLLLoooooo*
  *oooooRRRRRRR      <-+->         LLLLLLLooooo*
  *oooRRRRRRRRR        v           LLLLLLLLLooo*
   *oRRRRRRRRRRR                  LLLLLLLLLLLo*
   *oRRRRRRRRRRRR                LLLLLLLLLLLLo*
   *oRRRRRRRRRRRRRR            LLLLLLLLLLLLLLo*
    *RRRRRRRRRRRRRRRRRRR**LLLLLLLLLLLLLLLLLL*
     RRRRRRRRRRRRRRRRRRR**LLLLLLLLLLLLLLLLLL
      RRRRRRRRRRRRRRRRR  LLLLLLLLLLLLLLLLL
         RRRRRRRRRRRR      LLLLLLLLLLLL
```

The **LEFT** and **RIGHT** commands affect all subsequent manipulator motion until another **LEFT** or **RIGHT** command is issued.

- If the AML/E program issues a motion command (PMOVE, DPMOVE, XMOVE, ZMOVE, or GETPART) that can be reached in the current mode, the motion will be performed.

  For example, if the manipulator is in left mode and the AML/E program issues a motion command that can be reached in left mode, the manipulator would move to the point in left mode.

- If the AML/E program issues a motion command (PMOVE, DPMOVE, XMOVE, ZMOVE, or GETPART) that rAnnot,be reached in the current mode, the motion yill not be performed, and a data error will occur.

  For example, if the manipulator is in left mode and the AML/E program issues a motion command that can only be reached in right mode, the motion will not be performed, and a data error will occur. In this case, the user should issue a **RIGHT** command before the move.

  If the application uses points that are in the "left-mode-only" lobe or the "right-mode-only" lobe, the application should first issue a **LEFT** or a **RIGHT** command so the arm can change modes, rather than causing a data error to occur.

Linear motion that causes a change in the arm configuration will also give a data error. Thus if the arm is in left mode, and the application issues:

```
RIGHT;
LINEAR(30);
DPMOVE(<-100,0,0,0>);
```

a data error will result. Thus, if linear motion is to be used for a motion command that might cause a data error, a **LINEAR(0);** should be issued.

LINEAR Command

```
LINEAR(quality);
```

The LINEAR command allows the arm to be run)..t_alir).2EL not' suuliode.... In this mode, the path followed by the arm in moving to the target position of a move command is approximately along a straight line. When linear motion mode is on,11/1-70Vr571;7771 e linear; the velocity selecte the user is suspended; the controller selects vagues appropr a e to the !Talon.

The LINEAR command is only effective until the outer END statement of the program. When the END statement of the prog7EFTS;TICountered, the arm returns to the normal type of motion until a new LINEAR statement is encountered.

The " uality" represents the deviation from a straight line. A one (1) for the q ity" gives the straightest achievable move between two points. A fifty (50) for the "quality" gives the greatest deviation from a straight line. A low "quality" value means straighter arm movement, but it also Means slower arm movement. The "quality" numbers and their corresponding LINEAR effects are listed in tabular form in the appendices. To deactivate linear motion mode and return to a normal mode, the value 0 is used for "quality." Examples of the command's usage are outlined below

```
                      = Straight as possible
LINEARK0);            = Exit linear
LINEARK50);           = Fastest with greatest error
```

Note: Linear motion is not valid throughout the work space; the valid linear working area description is in the IBS Manufacturing Systems Specification Guide, 8577126.

Caution must be taken in teaching the end-points of linear moves; the path must remain in the valid linear working area at all times.

PAYLOAD Command

```
PAYLOAD(speed);
```

This command allows you to control tool-tip speed by overriding the speed switches or defaulting to those switches. The value must be in the 0 through 19 range. All values greater than 10 are slower than PAYLOAD value 1. The second digit of the parameter multiplied by 10 gives the perCentage of the PAYLOAD(1) speed. For example, PAYLOAD(13) is 30% of the PAYLOAD(1) speed. PAYLOAD(0) sets the payload to the switch value.

The setting selected is observed by all PMOVEs, DPMOVEs, ZMOVEs, GETPARTs, and XMOVEs, except when the LINEAR command is active. The value in the command is a whole number that reflects the speed attainable. The PAYLOAD command is only effective until the last END statement of the program. When the END statement of the program is encountered, the speed-switch settings take control until a new PAYLOAD statement is encountered.

The actual speed is a function of the value, the particular manipulator, and the weight of the payload carried. The relationship is given in tabular form in the appendixes with the description of this command. Examples of the Payload command are listed below.

```
PAYLOAD(0);        = Default to switches
PAYLOAD(1);        = Slow with heavy load
PAYLOAD(10);       = Fast with light load
PAYLOAD(12);       = 20 percent of PAYLOAD(1);
PAYLOAD(19);       = 90 percent of PAYLOAD(1);
```

PMOVE Command

```
PMOVE(point);
```

In AML/Entry, the PMOVE instruction moves the arm from the present point in the work area to a new location. X, Y, Z, and Roll coordinates must be specified.

The PMOVE command is outlined below.

```
PMOVE(PT(x,y,z,r));
PMOVE(name);
```

The form with the keyword PT uses a cartesian coordinate value for the X, Y, and Z-axes, and a roll value in degrees for the R-axis. The cartesian coordinates are in millimeters unless you have changed the AML/Entry program to work in inches as described in Chapter 2, "Getting Started on the IBM Personal Computer."

The PMOVE with a name uses a declared name as a point. In the declaration, the name is given values for x, y, z, and r. More information about declarations is provided later in this chapter.

RELEASE Command

```
RELEASE; /
```

The RELEASE command opens the DO 2 relay, which stops the air-flow through the center of the Z-axis shaft. If you have a gripper that is controlled by the Z-axis shaft air supply, the gripper opens. A DELAY command may be necessary to allow the gripper to open.

RIGHT Command (Valid on 7545-800S Only)

```
RIGHT;
```

For an explanation of the RIGHT command, refer to "LEFT Command (Valid on 7545-800S Only)" on page 4-9.

XMOVE Command

```
XMOVE(region_name,point_name);
```

The XMOVE command allows movement within a region, using resion4 coordinates. XMOVE will be discussed later in 'Region on page 4-82.

ZMOVE Command

```
ZMOVE(position);
```

This command is used for absolute Z-axis positioning. The position parameter is the exact extension you want the Z-axis to be moved to. If your program works in millimeters the range is from 0 (all the way up) to -250 mm (fully extended). If your program is working in inches the range is from ,0 (all the way up) to -9.8 inches (fully extended). It is also possible to use the ZMOVE command with a name or even a complex expression. The value of name or the expression must be in the Z-axis moveable range. See "Expressions" on page 4-48 for more on expressions.

Examples of the ZMOVE command are outlined below.

```
ZMOVE(-200);    --EXTEND TO 200 mm
ZMOVE(name)     --EXTEND TO POSITION DECLARED IN NAME
ZMOVE(name/2); --EXTEND TO POSITION GIVEN BY NAME/2
```

ZONE Command

```
ZONE(factor);
```

The ZONE command allows Zhe accuracy of finding a particular point to be specified. Position zones d6t1/176 how Physically close the arm must come to the target position of a move for that motion to be considered complete, and execution to continue. The factor used in the ZONE command effects all movement commands executed after the ZONE command. The ZONE command is only effective until the last END statement of the program. When the END statement of the program is encountered, the zone switch setting takes control unless a new ZONE statement is encountered.

Note: Reduction in settle time and target distance varies between machine types.

The effective target-distance is a function of the value. Refer to the Zone switch description in the IBM Manufacturing Systems Specifications Guide, 8577126.

The lower the ZONE factor number (or expression), the longer the time required to stabilize that position. This command can be used to increase throughput by changing the ZONE factor to a maximum practical value for goals that do not require precision. The value should range from 0 to 15, where 0 means use the switch setting. A ZONE(3) is equivalent to having the position zone switches 1 and 2 on. A ZONE(15) is equivalent to having all four switches in the on position. There is no program ZONE factor equivalent to all position switches in the off position, but if you have all switches off and specify a ZONE(0), you get the best possible precision.

```
ZONE(15);      = Not precise but fast
ZONE(1);       = More accurate goal '
ZONE(0);       = Default to switches
```

Using Motion Statements

Motion statements are used to begin manipulator motions. The following two examples show how to use several of the motion commands.

Using Move, Z-axis, Delay, and Gripper Commands

This example shows one method of using motion commands, along with controlling a gripper and the Z-axis. On line 2, a constant "LEVEL" is declared. This will be discussed in "Declarations" on page 4-30.

Some indentation is used in the program. The AML/Entry statements can be anywhere on a line. The indentation shown in the example is a good programming technique that allows the program to be read easier.

```
- - *******************************************************
   1 MAIN:SUBR;                --BEGINNING SUBROUTINE
   2 LEVEL:NEW -50;            --DECLARE A CONSTANT
   3   PMOVE(PT(300,400,- 150,120));
   4 ZMOVE(5*LEVEL);           --MOVE TO "LEVEL" 5 (-250)
   5 GRASP;
   6   DELAY(1.0);
   7   ZMOVE(3*LEVEL);         --MOVE TO "LEVEL" 3 (-150)
   8   DPMOVE(<10,-5,-10, -20>);
   9 ZMOVE(5*LEVEL);           --MOVE BACK TO "LEVEL" 5
  10 RELEASE;
  11   DELAY(1.0);
  12 ZMOVE(0*LEVEL);           --MOVE TO TOP LEVEL (0)
  13 END;                      --END OF PROGRAM
  14
- - * A****************************************************
```

This program moves the arm between the two locations provided as a point in line 3 and a vector move in line 8. At each location following the move, the Z-axis is fully extended (line 4) and a gripper is closed (line 5) or opened (line 10). A delay follows the gripper action to allow the mechanical motion to complete before going to the next statement.

The first move statement (line 3) is a move to a location in the work envelope with an X-coordinate of 300 millimeters, a Y-coordinate of 400 millimeters, 'a Z-coordinate of -150 millimeters, and a roll of 120 . The roll is in a clockwise direction. The second move (line 8) uses a vector instead of coordinates. The vector is plus 10 for X, minus 5 for Y, minus 10 for Z, and minus 20 for R. The vector takes the arm to X=310, Y=395, Z=-160, and R=1/00 .

The ZMOVE command (line 9) is used to extend the Z axis so it is at a point 250 millimeters below its zero point. The other ZMOVE command (line 12) fully retracts the Z-axis. Notice how expressions are used within the ZMOVE commands. These could also have been coded as the actual values of the expressions.

Using Linear, Speed, and Precise Motions

This example shows commands that affect the motion of the arm and the precision of finding a goal.

```
   ***************************************************
 1 MAIN:SUBR;                  --BEGINNING SUBROUTINE
 2 LINEAR(1);                  --SLOW AND LOWEST ERROR LINE
 3   PMOVE(PT(300 ,400,-50,0));
 4   PMOVE(PT(350 ,400,-100,0));
 5 LINEAR(0);                  --EXIT LINEAR MODE
 6   ZONE(15);                 --LEAST PRECISION MOVES
 7 PAYLOAD(1);                 --SLOW SPEED MOVES
 8   PMOVE(PT(400 ,350,-25,0));
 9 END;                        --END OF PROGRAM
10 ***************************************frn********
```

This program moves the arm between three points (lines 3, 4, and 8). The LINEAR statement (line 2) puts the arm in a linear mode at a slow speed. The linear precision is controlled by the number 1, enclosed in the parentheses after the command. A straight-line move is performed by the arm between the points provided in statements number 3 and number 4.

The LINEAR statement in line 5 changes the arm movement back to an arc-type of movement.

The ZONE statement (line 6) changes the amount of settle distance for a point to one that is less precise. The ZONE command then defaults to the switch settings after the END statement is reached.

The PAYLOAD statement (line 7) changes the arm speed to a slow speed to allow moving a heavy object. After the END statement, the speed returns to the setting in the controller.

SENSOR Commands

Sensor commands allow you to control the manipulator and the devices attached to it. The sensor commands are listed below.

ComMand	Description
CSTATUS	Controller enabled status
GUARDI	Guards motion (uses DI port)
MSTATUS	Move completion status
NOGUARD	Stops the guard feature
TESTI	Tests DI point
WAITI	Waits for DI point switch
WHERE	Updates a point definition
WRITE0	Opens or closes a DO relay

CSTATUS Command

```
CSTATUS(counter_name);
```

This command allows you to determine if the controller is able to initiate data transfer in an AML/Entry program. It is also used to monitor communicationsistem status.

The value returned is placed into a counter. Counters are run-time variables which can hold an integer or real value. Counters are discussed in "Variables" on page 4-38.

The value returned in a counter when a CSTATUS command is accomplished represents the state of the communication line, as outlined below.

```
    15 = controller enabled to communicate
Not 15 = controller not enabled to communicate
```

The below listed conditions must be present before a 15 is returned.

- Host Connection (DSR) Active
- Communications Cable Connected
- Controller On-Line
- Controller In Xon'ed State

An example of CSTATUS is outlined below in a program fragment.

```
TOP: CSTATUS(X);                    -- READ STATUS
     TESTC(X,15, GOOD1);            -- IS THE CONTROLLER
     WAITI(OP_OK,1,0);              -- ABLE TO COMMUNICATE?
     GOOD1: GOOD;                   -- CONTINUE PROGRAM
     BRANCH (TOP);
     END;
```

Note: This is the command form of CSTATUS. With the command form of CSTATUS, a counter_name must be given. CSTATUS may also be used as an arithmetic function within expressions. In the latter form, no counter_name is given. The above example could also be coded as:

```
TOP: TESTC(CSTATUS0,15,GOOD1); --IS THE CONTROLLER
     WAITI(OPDX,1,0);          --ABLE TO COMMUNICATE
     GOODI: GOOD;              --CONTINUE PROGRAM
     BRANCH(TOP);
     END;
```

See "Built-in Arithmetic Functions" on page 4-50 for more on arithmetic functions.

GUARD' Command

```
GUARDI(digital_input_point,value);
```

This command allows you to use a DI port to guard motion. It interrupts a motion, based on external input.

As a movement occurs, tie controller monitors the port value. If the DI point attains the speLilledvalue, motion is halted. The manipulator does not stop program execution, but regards the move as completed. Then, unless the GUARDI is ended with a NOGUARD, subseqUent moves will not start.

Both digital_input_point, and value can consist of integers, counters or formal parameters. The value of a counter for the digital_input_point should be an integer ranging fram_i_z:to the maximum number of installed digital inputs. A zero for value is considered "off" or "inactive". A non-zero value isEFITEM41a-4 6e-Z7iiaaiiii".

Only one DI point may be used as the motion guard at any one time. Whenever the GUARD' command is encountered, monitoring is changed to the newl -s e fied o' t. If a previous value was in effect, it is lost. However, the contr ller does not lose the guard condition when it is changed by a subroutine. For example, if GUARDI is set to a particular DI point and value, and a subroutine that changes the guard condition is called., the new condition is used for monitoring while the subroutine remains active. When the subroutine ends, and control is returned to the caller, the controller automatically restores the caller's guard condition.

The distance that the manipulator moves after motion is interrupted is proportional to the manipulator speed. At payloads 11 through 19, a different GUARD' monitoring scheme and deceleration ramp is used (as well as lower speeds), thus allowing shorter stopping distances.

Note: If a move is stopped by a DI point; the system automatically updates the internal location variables so that a DPMOVE executed immediately after a DI guarded motion acts as expected (moves a delta amount from where the arm came to rest).

MSTATUS Command

```
MSTATUS(counter_name);
```

This command allows you to determine the completion status of the last executed move. It specifies the counter into which the status is 17676Aa7--- ;;;Iers are run-time variables which can hold an integer or real value. Counters are discussed in "Variables" on page 4-38.

Status codes are listed below.

```
0 = motion completed normally
1 = motion terminated by a guard
2 = motion not started due to a guard
```

After loading the code into a counter, the program may use either TESTC or COMPC to test the value.

An example of a guarded move is outlined below in a program fragment.

```
  STAT: STATIC COUNTER;
    DI: STATIC COUNTER;
   PT1:NEW PT(0,500,0,0);
   PT2:NEW PT(0,400,0,0);
   PT3: NEW PT(0,0,0,0);
 TESTG: SUBR;
        SETC(DI,3);              -- SET THE POINT GUARD TO 3
        GUARDI(DI,1);            -- GUARD POINT 3 FOR A 1 CONDITION
        PMOVE(PT1);              -- MAKE A MOVE
        PMOVEI(PT2);
        MSTATUS(STAT);           -- DETERMINE THE COMPLETION STATUS
        TESTC(STAT,0,GOOD);      -- MOTION TERMINATED BY GUARD?
        WHERE(PT3);              -- UPDATE POINT
        WRITEO(ERROR,1);         -- INFORM OPERATOR
        WAITI(OP_OK,1,0);        -- WAIT FOR OK
        PMOVE(PT2);              -- FINISH MOVE
  GOOD: ZMOVE(-100);             -- YES, CONTINUE THE PROGRAM
        ZMOVE(0);
        END;
```

Note: This is the command form of MSTATUS. With the command form of MSTATUS, a counter_name must be given. MSTATUS may also be used as an arithmetic function within expressions. In the latter form, no counter_name is given.

In fact, the arithmetic function form is even more efficient then the counter form of MSTATUS. In the above example, the declaration for STAT' on line 1, the MSTATUS command, and the TESTC command could-be replaced by the single line:

```
TESTC(MSTATUS0,0,GOOD);
```

See "Built-in Arithmetic Functions" on page 4-50 for more on arithmetic functions.

NOGUARD Command

```
NOGUARD;
```

This command allows you to stop monitoring the input point currently specified by GUARDI.

TESTI Command

```
TESTI(digital_input_point,value,label);
```

The TESTI command checks the specified DI port for a specified value. If the values are the same a FflrTttPtr—rr"Made to a label in the same This command is described in greater OW-OF-CO TROL Commands" on page 4-23.

WAITI Command

```
WAITI(digital_input_port,value,time_limit [,label]);
```

This command allows you to respond to device time-out, rather than have the controller generate an OT (overtime)' error. It is designed to optimize throughput in an application that employs devices with time-limited behavior (such as grippers and feeders).

If the port does not attain the upcified ulaw,within the time_limit, control b ;; ,es to the s ecified label. If the time limit does not expire, control fa s i rough eiggioarim uction is exec . The label field is optional. If the label field is omitte, an OT will occur if the specified port does not achieve the value within 11e—rtnitr-tiirre7

The digital_input_port, value, and time_limit can be integer constants, counters, or formal parameters. The digital_input_point should be an integer ranging from 1 to the maximum number of installed digital inputs. A value of zero is treated as "off" or "inactive", a non-zero value is treated as "on" or "active". The should range from 0 through 25.5. A time_limit of 0 makes the controller wait "forever" for tETEFUlgitia'l_input_point to attain the specified value.

Examples:

```
WAITI(5,1,10);           --Wait 10 seconds for DI 5 to turn on.
                         --If it does not turn on, then OT error.
WAITI(7,0,5,STILL_ON); --Wait 5 seconds for DI 7 to turn off.
                         --If it does not turn on, branch to STILL_ON.
WAITI(7,0,0);            --Wait forever for DI 7 to turn off.
```

WHERE Command

```
WHERE(point);
```

This command allows you to update a point definition, based on the current position of the arm. The point must not be a fOrmal parameter. It can be used to update the position after a move is stopped by a motion guard. Format of the command is outlined below.

```
G: STATIC GROUP (PT(0,0,0,0));
WHERE(G(1));

H: NEW PT(0,0,0,0);
WHERE(H);
```

After performing the WHERE command, the value of one of the coordinates of the current manipulator location may be extracted using the FROMPT arithmetic function. See "Built-in Arithmetic Functions" on page 4-50 for more on arithmetic functions.

WRITEO Command

```
WRITEO(DO,value);
```

The WRITEO command instructs the controller to open or close a digital output point relay. Both DO and value' can consist of integers, counters, or formal parameters. The DO should be an integer ranging from 1 to the maximum number of installed digital outputs. jyallagsal._ 2m5Lialtens—th4—relay and a non-zero value closes it. The relay stays in the open or closed position until another WRITEO command changes it.

Using Sensor Statements

In the following example, the WAITI and WRITEO are incorporated into one of the earlier programs. The WAITI determines when a part is available for pickup and the WRITEO indicates to the operator that a part has been moved.

```
****************************** ***********************
 1 MAIN:SUBR;                    --BEGINNING SUBROUTINE
 2   PMOVE(PT(300,400, -150,120));
 3   WAITI(6,1,15);              --WAIT 15 SEC. FOR DI 6
 4   ZMOVE(-250);                --TO CLOSE
 5 GRASP;
 6   DELAY(1.0);
 7   ZMOVE(-150);
 8   WRITEO(7,1);                --CLOSE DO 7
 9   DPMOVE(<10,-5,-10,-20>);
10   ZMOVE(-250);
11 RELEASE;
12   DELAY(1.0);
13   ZMOVE(0);
14 WRITEO(7,0);                  --OPEN DO 7
15 END;                          --END OF PROGRAM
16
************************* ***************************
```

In this example, the arm moves to the first point (line 2) and the controller waits (line 3) for the DI 6 to close before going to the next instruction (line 4). If the part, in this example, is not present within the 15-second period, program execution stops and the control panel overtime (OT) LED lights.

Once a part is picked up (line 7), the controller signals through the closing of DO 7 (line 8). The DO point 7 remains on until line 14 of the program, when the controller opens the relay for DO 7.

FLOW-OF-CONTROL Commands

Flow-of-control commands are used to change the order in which program statements are executed. By using them correctly, you can execute different parts of your program for different reasons. Only four flow-of-control commands will be discussed at this point. The remaining commands will be covered in later sections. TESTC and COMPC are covered under Counter Commands because they test counters and expressions. TESTP is covered under Palletizing Commands and ITERATE is covered under Subroutines and Aggregate Constants.

Command	Description
BRANCH	Transfers control
BREAKPOINT	Predetermined stop in program
COMPC	Compares counters
ITERATE	Repeats execution
TESTC	Tests counter
TESTI	Tests DI point
TESTP	Tests current part counter

Labels

In some of the statements for flow-of-control, labels are part of the statement. A label is a name used to locate a line. In a program, a label is used when you want execution to transfer from one part of a subroutine to another line within a subroutine. It is like putting an address in one statement and a matching address at the line desired. (More details on subroutines are in a later section titled "Subroutines" on page 4-60). Labels have the following format and characteristics:

- Up to 72 characters.
- First character must be alphabetic.
- Remaining characters must be alphabetiC, numeric, or underscore (_).
- An underscore can not be the last character.
- Special characters, such as an asterisk (k), are not permitted.

An example of a valid label is outlined below.

```
STATION1:executable statement;
```

An invalid usage is:

```
1STATION:executable statement;
```

is a o•erator that always follows a label. Labels are permitted on lines with or wit ou an ' ;$ e s a emen after the colon (:).

BRANCH Command

```
BRANCH(label);
```

The BR:. comman• . sfers • to the line in the subroutine containing the matching label. The BRANCH instruction allows skipping of lines of the program while staying in the subroutine that contains the BRANCH. The target label of the BRANCH must 'e same subroutine, and can be before or after the ine with the BRANCH mtmtement.

BREAKPOINT Command

```
BREAKPOINT;
```

The BREAKPOINT is a Special flow-of-control type of command that allows the control panel 24.1412L to sto he program at a predetermined location using the Stop em key on the control panel. The controller remembers' the next statement to be executed after the BREAKPOINT command even when power is removed. The operator can power-up the system at a later time and use the **Recall Memory** key to bring the next statement into the processor for execution. The statement is executed when the **Start Cycle** key is pressed. For detailed instructions on host to restart the system, please refer to Chapter 7, "Operating the Manufacturing System" in this manual.

Special things to consider for this command are listed below.

- Down or Grasp conditions are not retained during the power down and future power up. For example, if the gripper is closed before the power down, it is opened at power up.
- DO states are not retained during BREAKPOINT if Manip Power is turned off. All DO's become open.
- If you execute the BREAKPOINT command without removing Manip Power you can restart execution at the place it was stopped. However, if Manip Power is turned off the manipulator must be returned to the Home position befOre execution can resume.

ITERATE Statement

```
ITERATE('command',operand 1,....operand n);
ITERATE('subroutinel,operand 1,....operand n);
```

The ITERATE statement'r a subroutinea command using the values provided in an aggregate. (See e al s in the "Aggregate Constants" on page 4-35 and "Subroutines" on page 4-60).

The command or subroutine names must be enclosed in single quotes (').

TESTI Command

```
TESTI(DI,value,label);
```

The TESTI command checks a digital point for a true condition (specified by the value in t e sta men . The Dr and the value can consist of integers, counters, or formal parameters. The DI should be an integer ranging from 1 to the number of installed digital inputs.

If the condition is true, control is transferred to the line containing the specified label. If the value in the statement is a 0, the transfer occurs if the controller detects an open switch at the DI point. If it has a non-zero value, the transfer occurs if the controller detects a closed switch at the DI point.

The testing for a true condition at a DI point is a momentary test. The controller does not wait for the condition to become true. If slals2-- condition is detected, the program executes the program statement that follows the TEST.I..--,statement and no transfer to a label occurs. The matching label must be in the same subroutine as the TESTI and can be either before or after the TESTI statement.

By using expressions, it is possible to test for any combination of digital inputs. For example, rather than using two TESTI commands to test for inputs 1 and 2 being active, it is possible to use a single COMPC command. See the discussion of the TESTI function in "Built-in Arithmetic Functions" on page 4-50 and "Treating DI As Integers" on page 4-56 for examples of this and a more general solution to this problem.

Note: This is the command form of TESTI. With the command form of TESTI, the three parameters must be given. TESTI may also be used as an arithmetic function within expressions. In the latter form, only the DI is given. See "Built-in Arithmetic Functions" on page 4-50 for more on using TESTI as an arithmetic function.

The next example has incorporated into the previous developed program the following Flow-of+Control Commands:

- BRANCH
- BREAKPOINT
- TESTI

The function of the BRANCH in this program is to avoid program statements unless they are needed. The BREAKPOINT provides a location to stop the program without having to go to the END statement. This might be useful, for example, if the operator must leave the plant floor temporarily. The TESTI provides testing for conditions, such as an empty feeder.

```
-,-,-,..,--,-,-**********************************************
 1 MAIN:SUBR;                        --BEGINNING SUBROUTINE
 2 LEVEL:NEW -50;                    --A LOCAL CONSTANT
 3   PMOVE(PT(300,400,-150,120));
 4 TESTI(6,0,FEEDER2);               --TEST FEEDER 1
 5 BREAKPOINT;
 6 ZMOVE(LEVEL*5);                   --MOVE TO "LEVEL" 5 (-250)
 7   GRASP;
 8   DELAY(1.0);
 9 ZMOVE(3*LEVEL);                   --MOVE TO "LEVEL" 3 (-150)
10 BRANCH(BYPASS);                   --PASS IF FEEDER1 OK
11      FEEDER2:PMOVE(PT( -300,400,-50,120));
12      ZMOVE(5*LEVEL);              --MOVE TO "LEVEL" 5 (-250)
13      GRASP;
14      DELAY(1.0);
15      ZMOVE(LEVEL*0);              --MOVE TO "LEVEL" 0 (0)
16   BYPASS:DPMOVE(<10,-5 ,0,-20>);
17   ZMOVE(4*LEVEL);                 --MOVE TO "LEVEL" 4 (-200)
18 RELEASE;
19   DELAY(1.0);
20 ZMOVE(LEVEL*0);                   --MOVE TO "LEVEL" 0 (0)
21 END;                              --END OF PROGRAM
22
*****************************************************
```

In this example, the arm moves to the first point (line 3); the controller then tests (line 4) for the DI 6 open (no part). If no part is present the program branches to the line with the label FEEDER2 located at line 11 and the application continues. If the test (line 4) shows that a part is present, line 5 of the program is executed.

The BREAKPOINT statement (line 5) allows the operator to stop the program after the arm executes the test for a part at the first part location. This is done by pressing the **Stop and Mem** key on the control panel. The controller can be powered down completely up to 80 days, and the application continues at line 6 of the program when the Recall Memory key is pressed,

The BRANCH statement (line 10) is used when FEEDER2 statements are not needed. When the program executes line 10, a branch to line 16 is executed. Line 16 has the matching label to line 10.

TECHNIQUES TO SIMPLIFY PROGRAMMING

The following sections contain information on certain techniques that can make programs more readable, and therefore easier to understand. Subjects discussed in this section include:

- Use of multiple statements on a line
- Definition of a declaration
- Declaration of Constants
  1. Global Constants
  2. Local Constants
  3. Aggregate Constants
- Declaration of Variables
  - Counters
  - Groups
- Expressions
- Built-in Arithmetic Functions

MULTIPLE STATEMENTS ON A LINE

The next example shows how the previous program can be put in a form where multiple statements are on the same line. This compressed form may be necessary for longer programs. If your Personal Computer has 192K memory, your program must be less than 500 lines long. If your Personal Computer has 256K memory, then your program must be less than 800 lines long.

Note: Each statement ends with a semicolon. This method of coding is a more efficient use of storage space on each diskette for the IBM Personal Computer and allows more program statements in the editor. Multiple statements on a line have no effect on the way the program is stored or used by the controller.

```
--  *****************************************************
 1 MAIN:SUBR;                                 --BEGINNING SUBROUTINE
 2   PMOVE(PT(300,400,-100,120));
 3 TESTI(6,0,FEEDER2);                        --TEST FEEDER 1
 4 BREAKPOINT;
 5   ZMOVE(-250);GRASP;DELAY(1.0);ZMOVE(0);
 6 BRANCH(BYPASS);                            --PASS IF FEEDER1 OK
 7      FEEDER2:PMOVE(PT(-300,400,-50,120));
 8      ZMOVE(-250);GRASP;DELAY(1.0);ZMOVE(0);
 9   BYPASS:DPMOVE(<10,-5,0,-20>);
10   ZMOVE(-200);RELEASE;DELAY(1.0);ZMOVE(0);
11 END;                                       --END OF PROGRAM
12
--  *****************************************************
```

Lines 5, 8, and 10 have multiple statements which have reduced the number of lines in the editor.

DECLARATIONS

When you assign a name as a substitute for the actual value, you are making a declaration. When the program encounters the name, the value for the name is used.

Using Declarations

Using declarations in your program makes programming less complicated because it is easier to remember a name than a series of numbers. Once you define the name to be equal to a number, the compiler uses the correct number each time the name is found in the program..

The AML/Entry keyword you use in your declaration statement informs the compiler what type of term is being declared. The types of terms AML/Entry accepts are:

- onstants, which retain their starting value. They may be one of the ollowing ypes:
  - Numbers, either integers (no decimal points) or real (decimal points
  - 41,111S, which are the coordinates of a location in the arm work space.
  - ,Character strings.
  - Aggregates- which are a collection of like numbers, points, or character strings.
- Variables, which change value when program statements modify them.
- Com•lex structures such as Pallets and Regions. Pallets and Regions descri•e to t e c•ntro er the location o e structures within the work space.

Declarations serve two important purposes:

- Reserve storage for a variable, which can later be used in expressions.
- Provide names for later reference.
- Describe Pallets and Regions.

CONSTANTS

In AML/Entry, constants can be numbers, points, character strings, or aggregates.

Declaring Constants

Constant declaration statements have the following forms:

```
name:NEW n;
name:NEW PT(x,y,z,r);
name:NEW 'string';
name:NEW <aggregate›;
```

The name in the declaration has the following format and characteristics:

- Up to 72 characters
- First character must be alphabetic
- Remaining characters must be alphabetic, numeric, or underscore (_.)
- An underscore can not be the last character
- Special characters, such as an asterisk (k), are not permitted

The term NEW is a keyword used by the AML/Entry program to identify a constantfTgrname and the NEW keyword are separated by the colon (:), a 4417.7711.on operator.

If you are declaring a point, the term PT follows the keyword NEW. **A** blank space must separate the two tellarr^{m}roints contain the X, Y, and Z-axes coordinates in the work space, and rotation of the R-axis in degrees. Commas (,) separate each coordinate and the roll value. The values of the coordinateg and roll are enclosed in parentheses.

The semicolon (;) delimiter indicates that the information for the statement is conFeTW.

Three types of declarations are shown below:

```
TIME:NEW 8;                      -- Declares a number
POINT:NEW PT(650,0,-50,0);       -- Declares a point
LOOP:NEW 'Al';                   -- Declares a string
```

In the first example, the constant is a number with the value 8 and the name TIME. In an application program, the name has meaning when a DELAY statement is used. The statement includes TIME within parentheses after the command to delay.

```
DELAY(TIME); is the same as DELAY(8);
```

The name POINT, in the second example, can be the parameter for a move command. When the program executes, the values 650,0,-50,0 are substituted for the name POINT. The number 650 is a cartesian coordinate value on the X-axis. The number 0 after the comma separator is a cartesian coordinate value on the Y-axis. The number -50 is a cartesian coordinate value on the Z-axis. The cartesian coordinates are in millimeters unless you configure your system to use inches. The last 0 is the angle of the roll axis in degrees. The angle of the roll axis is based on the starting angle when the arm is at its home position.

Note: Expressions are not allowed within declarations. Express e only allowed inside some of the AML/Entry commands. See "Commands That Allow Expressions" on page 4-54 for a discussion of where expressions are allowed.

Local Constants

Local constants are constants declared in a specific subroutine for use TrirtrinTiiirnnTEsubroutine only. To make a local constant declaration, you must lace the constant declaration after the ,ds fallon of the subroutine to which it belongs.

Global Constants

A name declared in a global declaration can be referenced in any statement in your program. To make a name a global declaration, you must place the constant declaration statement before the outmpst subroutine statement.

Global vs. Local Constants

AML/Entry uses both local and global constants. The difference between the two is where they can be used in the program. Global constants can be used anywhere in the program; local constants can only be used within the subroutine where they are declared. It is possible to write a program with only global constants. However, using local constants correctly in your program can make it much easier to read and understand. The important thing to remember is that a constant declared globally can be used anywhere in the program, while a constant declared in a subroutine can only be used in that subroutine.

Using Constants

The following programs show usage of global and local constants declarations.

Using Local Constants

This example performs the same movements for two different ZONE command values. Since the movement of the arm is repeated a subroutine has been created with local point declarations. The program uses nested subroutines which are explained later in the chapter, however they are used in the example to help you understand the concept of local variables. With the point constants declared in the subroutine responsible for movement there is an added organization to the program making it easier to understand.

```
****************************************************************
 1 ZONEVALLNEW 4;                                   --GLOBAL DECLARATIONS
 2 ZONEVAL2:NEW 6;
 3 PAY VAL:NEW 7;
 4 MAIRS:  UBR;                                     --MAIN SUBROUTINE
 5
 6 TRANSPORT:SUBR;                                  --MOVEMENT SUBROUTINE
 7        MOVE1:NEW PT(300,-300,-100,0);            --LOCAL DECLARATION
 8        MOVE2:NEW PT(300,300,0,180);              --LOCAL DECLARATION
 9        PMOVE(MOVE1);
10        ZMOVE(-250);
11        GRASP;
12        DELAY(2.0);
13        ZMOVE(0);
14        PMOVE(MOVE2);
15        ZMOVE(-150);
16        RELEASE;
17 END;
18
19 ZONE(ZONEVAL1);
20 PAYLOAD(PAY_VAL);          --ANY ADDITIONAL MOVEMENTS MUST BE MADE
21 TRANSPORT;                 --SEPARATELY BECAUSE TRANSPORT CALLS ITS
22 ZONE(ZONEVAL2);            --OWN VALUES
23 TRANSPORT;
24 END;
********************************************************
```

The declared names in this program are:

- ZONEVAL1 - A global constant with value 4
- ZONEVAL2 - A global constant with value 6
- PAY VAL - A global constant with value 7
- MOVE1 - A local constant of the subroutine TRANSPORT. It is a point with values of X=300, Y=-300, Z=-100, and R=0. It can only be used by the subroutine TRANSPORT.
- MOVE2 - A local constant of the subroutine TRANSPORT. It is a point with values of X=300, Y=300, Z=0, and R=180. It can only be used by the subroutine TRANSPORT.

Using Global Constants

The following program is an earlier example that has been updated to use global declarations.

```
-- ******************************************************
  1 SLOW:NEW 1;                              --LINES 1-7 ARE ALL
  2 POINT1:NEW PT(300,400,-100,0);           --GLOBAL DECLARATIONS
  3 POINT2:NEW PT(350,400,-50,0);
  4 POINT3:NEW PT(400,350,0,0);
  5 OFF:NEW 0;
  6 LOOSE:NEW 15;
  7 STRAIGHT:NEW 1;
  8
  9 MAIN:SUBR;                               --BEGINNING SUBROUTINE
 10 PMOVE(POINT1);
 11 LINEAR(STRA1GHT);
 12 PMOVE(POINT);
 13 LINEAR(OFF);                             --EXIT STRAIGHT
 14   ZONE(LOOSE);
 15 PAYLOAD(SLOW);
 16   PMOVE(POINT5);
 17 PAYLOAD(OFF);                            --DEFAULT TO SWITCHES
 18 ZONE(OFF);                               --DEFAULT TO SWITCHES
 19 END;                                     --END OF PROGRAM
 20
---".*****************************************************
```

The declared names for global constants in the program are:

- SLOW - This name has a value of 1.
- POINT' —This is the point with coordinates of X=300, Y=400, Z=-100, and roll of 0.
- POINT2 - This is the point with coordinates of X=350, Y=400, Z=-50, and roll of 0.
- POINT3 - This is the point with coordinates of X=400, Y=350, Z=0, and roll of 0.
- OFF - This is used any time a 0 is wanted to end a mode or to default to the controller switches.
- LOOSE - This name, changes the ZONE value.
- STRAIGHT - This name sets the LINEAR command for slow straight mode when used in that type of statement.

These constants are global because they are declared before the outer subroutine in the program (line 9). They can be accessed anywhere in the program.

Aggregate Constants

Another technique for declaring constants is the aggregate declaration statement. This method allows you to declare a number of parameters with one declaration statement. The statement has the following form:

```
name:NEW <aggregate>•
```

Aggregates are enclosed in gie hEackets. The parameters that make up the aggregate are separated by commas, and must be of the same data ty e (numbers, points, strings). Some examp es e ec aration are:

```
PORTS:NEW <3,4,5>;

POINTS:NEW <POINT1,POINT2>;
```

Note that the aggregate PORTS contains only numbers, and the aggregate POINTS contains two declared points. Each point must be declared prior to the POINTS declaration statement. <—

Example:

```
POINT1:NEW PT(300,400,-100,180);
POINT2:NEW PT(400,300,-50,0);
```

These declarations must precede the aggregate declaration.

Aggregates are not allowed to contain expressions. Expressions are only allowed wift1M-1MrrMrbr-*Mr7E7try commands. See "Commands That Allow Expressions" on page 4-54 for a list of where expressions may be used.

Using the ITERATE Command with aggregates

The ITERATE command is a Flow-of-Control command. It repeats a subroutine or a command using the values provided in an aggregate. An aggregate is a listing of like parameters. The parameters in the aggregate are used for values in the subroutine or command, starting with the parameter in the leftmost position, moving towards the parameter at the rightmost pose

Note: When executing an ITERATE command it is not possible to exit from the command until all elements in the aggregate have been used.

The format of the ITERATE command is outlined below.

```
ITERATE('command',<aggregate>,...);
ITERATE('subroutine',<aggregate>,...);
```

Use of ITERATE commands for subroutines is explained under the section on subroutines.

Note: The symbol used for the single quote on the Personal Computer appears as ' on the keyboard.

An example of repeating a command is outlined below.

```
PT1:NEW PT(500,400,0,180);
PT2:NEW PT(450,450,-100,90);
POINTS:NEW <PT1,PT2>;
     OUTER:SUBR;
        ITERATE('PMOVE',POINTS);
       END;
```

In the example, the PMOVE command is exec ted twice. Each time the PMOVE is executed, it uses a new point. The locations for the moves are PT1 and PT2. Using the ITERATE command will perform exactly as if the following two command were used instead:

```
PMOVE(PT1);
PMOVE(PT2);
```

The next example uses the declaration for a WRITEO to DO points 3, 4, and 5, which are named as ports. The ITERATE statement eliminates the need to write separate statements to repeat the action.

```
PORTS:NEW <3,4,5>;
  OUTER:SUBR;
       ITERATE('WRITE0',PORTS,1);
```

During the execution of the program it appears as if a WRITEO statement has been written for every value in the aggregate. Each WRITEO statement uses a value from the aggregate for the PORT number, and the number one as the other parameter. Each value in the aggregate is used in the order it is written, and each value is only used once.

In the compiler the command that is used in the ITERATE statement is repeated for every element of the aggregate. This takes up additional space within the controller. To get the most efficient use of controller space, you should perform the desired action using a subroutine with formal parameters, which is discussed in the section titled "Using Subroutines with Parameters" on page 4-68.

In the example it would have been possible to use another aggregate to represent the "value" parameter at the same time. If you have an ITERATE statement executing a command or subroutine with more.than one aggregate as a parameter, you must make sure that all aggregates have rthe same number of values in them, and since all values are only used once, your values must-6e aligned in the aggregates in the order you want them used as parameters.

Note: The ITERATE command does not allow expressions as arguments. The DELAY, SETC, -",TSTC7—2ffel—ZrOVE—L llow expressions as arguments (see "Commands That Allow Expressions" on page 4-54), but expressions are not allowed to appear as arguments when the command appears within an ITERATE command.

VARIABLES

Variables change values during your program as AML/Entry statements modify the present values. The controller only retains the new value.

The variable structures discussed in this section are:

- Counters
- Groups

The special uses in AML/Entry that variables have for pallets will be discussed briefly in this section. More details on pallets will be covered in the section titled "Pallet" on page 4-74.

Declaring Variables

The following keywords are used when declaring variables:

Keyword	Description
STATIC	Identifies variable name for counter, pallet, region, and group
COUNTER	Defines counter name

You declare variables using the STATIC keyword in the following form:

```
name:STATIC COUNTER;
```

The name has the following format and characteristics:

- Up to 72 characters
- First character must be alphabetic
- Remaining characters must be alphabetic, numeric, or underscore (_)
- An underscore can not be the last character

  Special characters, such as an asterisk (*), are not permitted.

The term STATIC identifies that the name is intended for a variable. It also implies that a variable has no initial value, but a variable keeps any value assigned to it even across power down conditions.

Counters

Counters may be used to hold either an integer or real value. Their use allows the program to maintain an irifFrirErdesCiiiirrMtrgrthe actual external process. For example, rather than performing an operation a fixed number of times, it is possible to produce a specified number of parts by counting non-reject assemblies. These counters static entities. This means that their value is not ost situation or from run to run of the application. Thus, a counter can be us • re .:1 jor weekly pr sob ion. Refer to "COUNTER" on page A-20 to find out how to reset counters.

Counters may be used to specify parameters for other commands. For example, the below outlined command is legal.

```
WRITEO(counter_name,1);    counter_name specifies the port
```

In fact, counters may be used within expressions (see "Expressions" on page 4-48), inside PT declarations, or passed as actual parameters to a subroutine. In general, a counter can be used wherever a formal parameter can be used (See "Formal Parameters") or within an expression.

Round-Off Error

Counters hold either an integer or real value. Whether they hold an integer or real value depends on how they are assigned, but all integers are stored in real number format in the controller. As long as counters are assigned integer values or integer expressions, the value will be stored in memory accurately. As soon as a real value or real expression is stored in a counter, the counter is subject to small round-off errors. The amount of the round-off error depends on the real expression. Generally it is less than 0.00001. This round-off error is common to all binary computers, and cannot be avoided. In most cases, the round-off error may be ignored entirely.

However the round-off error may play a role when used with the COMPC or TESTC command. For example, suppose the counter A contains the value 10.5, and the value of the counter B contains the result of the expression (10.5/1234.56)*(12.3456/0.01). Because B contains a value that is the result of a real expression, it may be subject to a round-off error. It would be fallacious for a program to use COMPC or TESTC to compare A and B for equality. Because B may be off slightly, this would affect the result of a comparison to **A**. Rather than taking the equal course of action (as it should because the result of the expression that determines B reduces to 10.5), the controller would take the not equal course of action. Instead, one should test to see if A and B are sufficiently close to each other that they can be considered equal. For example, rather than:

```
COMPC(A=B,EQUAL);
```

One would use:

```
COMPC(ABS(A-B) <= 0.001,EQUAL);
```

The latter example will branch to the label EQUAL if A and B are within 0.001 of each other.

This is not a problem when integer counters and expressions are used. As long as division, the SQRT, and trigonometric functions (see "Built-in Arithmetic Functions" on page 4-50) are not used, the result will remain an integer.

COUNTER Commands

A counter is declared using the keyword STATIC. There are five verbs exclusively available for use with counters.

Command	Description
COMPC	Compares Counters
DECR	Decrements counter by 1
INCR	Increments Counter By 1
SETC	Sets Counter
TESTC	Tests Counter

DECR Command

```
DECR(name)
```

The DECR statement decreases the counter by 1 each time the statement is executed.

INCR Command

```
INCR(name)
```

The INCR statement increases the counter by 1 each time the statement is executed.

SETC Command

```
SETC(name,value)
```

The SETC statement sets the "named" counter to the "value" each time the statement is executed. The value can consist of azjagconstant or a Will112.1L-2,2(41.LEAS1011. By using the SETC statement to assign an expression to a counter, and by then using the counter in a PT or an aggregate, it is possible to solve many difficult applications. See "Expressions" on page 4-48 for more on expressions.

COMPC Command

```
COMPC(expl condition exp2,label);
```

To allow a greater degree of control over the execution path, this command provides the ability to compare two expressions. The expressions can consist of_simat_gazstants sin le c or even complex expressions. If the condition is met, t e statement branches to the specified label.

The COMPC command supports two sets of relational operators. Allowable conditions are listed below. Enter exactly as shown.

LT or < less than

LE or <= less than or equal to

GT or > greater than

GE or >= greater than or equal to

EQ or = equal to

NE or <> not equal to

Notes:

1. Do not enter '=<' for '<=', '= ' for >=', or >< for <> as the order is important.

2. For the compiler to be able to discern the alphabetic operators from their surrounding expressions, those operators must be surrounded by at least one blank on each side. The following statements are identical:

```
COMPC(CTR EQ O,CONTINUE);
COMPC(CTR=0,CONTINUE);
```

As shown below in a program fragment, the subroutine DO_SOMETHING is executed twice. The third time through the loop C is greater than 2 and the program branches to the end of the program.

```
   C:STATIC COUNTER;
MAIN:SUBR;
     SETC(C,1);
NEXT:COMPC(C>2,EN);
     DO_SOMETHING;
     SETC(C,C+1);
     BRANCH(NEXT);
  EN:
     END;
```

Note: Be careful when using COMPC to compare two expressions that contain real values. Binary computers generally have a small round-off error associated with storing real numbers, which may cause comparisons to fail in unexpected places. See "Round-Off Error" on page 4-39 for more on this.

TESTC Command

```
TESTC(name,value,label);
```

The TESTC command compares name with value in the statement. If a true condition exists, program con axis rs to the statement that has the same label as in the TESTC statement. If the condition is not true, the statement that follows **the** 1.6ba; statement is executed. The matching label must be in the same subroutine as the TESTC, and can be either beifts^757 after in—TESTC–Initement. Both name and value can consist of an expression. See "Expressions" on page 4-48 for more on expressions.

> **Note:** Be careful when using TESTC to compare two expressions that contain real values. Binary computers generally have a small round-off error associated with storing real numbers, which may cause comparisons to fail in unexpected places. See "Round-Off Error" on page 4-39 for more on this.

Using Counter Statements

CTR1 in the following example is a counter to track parts that are built. Another counter, CTR2, keeps track of the decreasing inventory used in the part.

```
COMMAND INPUT -->
     *****************************************************
    1 CTR1:STATIC COUNTER;              --PARTS BUILT
    2 CTR2:STATIC COUNTER;              --INVENTORY MATERIALS
    3 START:SUBR;
    4 SET:SUBR;
    5     TESTI(16,0,NOCHANGE);         -- IS DI 16 ON
    6     SETC(CTR2,200);               --SET COUNTER TO 200
    7 NOCHANGE:                         --BYPASS RESET
    8     END;
    9     SET;                      --CALL SET SUBROUTINE FOR COUNTER
   10     PMOVE(PT(300,400,-100,0));
   11     ZMOVE(-250);GRASP;ZMOVE(0);
   12     INCR(CTR1);                   --ADD 1 TO THE COUNTER
   13     DECR(CTR2);                   --REDUCE INVENTORY
   14     PMOVE(PT(400,300,-50,0));
   15     ZMOVE(-175);RELEASE;ZMOVE(0);
   16     TESTC(CTR2,0,STOP);           --TESTS INVENTORY
   17     BRANCH(PARTSAVAIL);
   18 STOP:WRITEO(10,1);
   19 PARTS AVAIL:
   20        .
   21 END;
```

Line 16 tests the inventory to determine if all the parts are gone. If no parts exist, the program branches to STOP to signal an operator using DO 10. Each time a starting inventory counter value is desired, DI 16 receives an input to set the counter to 200. The program skips line 6 of the program if line 5 does not receive a DI 16 signal with a value of 1.

PT's Defined in Terms of Formals and/or Counters

PT is a keyword that defines position and rotation of a point. As shown below, a PT can be defined in terms of either a formal parameter or a counter.

```
C: STATIC COUNTER;

S: SUBR(F);
P: NEW PT(F,C,0,0);       -- X coordinate is a formal parameter
       PMOVE(P);          -- and Y coordinate is a counter
END;
```

Point P in the above example has an x coordinate that is defined by the value of the formal parameter F and a y coordinate that is a counter. The NEW PT is considered an executable statement in that it does generate code to initialize the point when the subroutine is called.

Note: If this programming technique is used on a PT that is global, the values are only updated when the cycle is started. If the application program uses a branch loop, rather than allowing the end-of-cycle to occur, the PT is not re-evaluated.

Note: If data drive is used to change the value of P via communications, then the values of F and C remain unchanged. P is a point whose fist two values come from F and C, but the values of F and C do not depend on the vales of P.

To improve the efficiency of host data transfers and improve program productivity, ire7-051561:51777Z7FEFULTral77.7ted data items to be 3Tgaa–Mrtribously in the controller's memor . This allows the host to read or write m t is are single or group items.

GROUP is a keyword which must be defined as STATIC. A group consists of either points or ng e-va ued variables (counters). A group must consist of at least one element. If the group contains more than one element, the elements must all be of the same type (a group of points or a group o . . . .) m x •ata types, e compiler reports an error. For example, a group of points is defined as outlined below.

```
FIXTURES: STATIC GROUP(PT1,PT2,...PTn);
```

Here, PTn is either the name of a defined PT or is a PT.

```
    PLACE: NEW PT(200,300,-10,0);
FIXTURES: STATIC GROUP(PLACE,PT(0,0,0,0));
```

This statement allows all of the points to be considered as a group. The group, however, is stored in a separate are of memory. Thus if data drive is used to change the group, the point PLACE is not modified. In order to change the value of PLACE in both the point and the group, then data drive must be used on both the point and the group.

AML/Entry llows •roups of counters to be used. In this case, counters are not assigned in•ivi•ua names wi he PT's) but must be assigned an initial value as outlined below.

```
H: STATIC GROUP(2,2,23,0,5); -- five counters.
```

This statement defines a group of five counters with the group name H. Initial values of the individual counters are listed below.

```
H(1) = 2
H(2) = 2
H(3) = 23
H(4) = 0
H(5) = 5
```

Examples of groups of counters are shown below.

```
SETC(H(3),H(3)/2);          --Divides third element of H by 2

SETC(INDEX_LH(INDEX_2));    --Use one index to set the other

ZMOVE(H(CTR));              --Uses counter named CTR
                            --to index the group
```

Note: A group or a particular group item can not be used in an aggregate. A formal parameter can not be treated as a group.

Indexing

The elements of a group are accessed b indexing. The index can be a number, a parameter, or a counter. xamp es of indexing are shown below.

```
    *******************************************************
  1 POINT1:NEW PT(0,500,0,0);          -- LOCATION OF FIXTURE 1
  2 POINT2:NEW PT(0,600,0,0);          -- LOCATION OF FIXTURE 2
  3 POINT3:NEW PT(50,500,0,0);         -- LOCATION OF FIXTURE 3
  4 FIXTURES:STATIC GROUP (POINT1,POINT2,POINT3);
  5 INDEX:STATIC COUNTER;              -- THE INDEX INTO FIXTURES
  6
  7 MAIN:SUBR;
  8 SETC(INDEX,1);                     -- START AT THE FIRST FIXTURE
  9 TRY:PMOVE(FIXTURES(INDEX));        -- MOVE TO FIXTURE
 10      INCR(INDEX);                  -- GO TO NEXT FIXTURE
 11      TESTC(INDEX,4,EN);            -- MOVED TO ALL 3 FIXTURES
 12      BRANCH(TRY);                  -- IF NOT MOVE AGAIN
 13 EN:                                -- THE REST OF THE PROGRAM
 14      END;
    ***************************A-*************************
```

When an individual point or counter of a group is referenced, an index must be given. The only time a group may ***be*** referenced without an index is in the GET or PUT commands. For example, suppose a global group of 4 counters is used to declare a point, PT1. The following would have to be used:

```
GR:STATIC GROUP(650,0,0,0);
PT1:NEW PT(GR(1),GR(2),GR(3),GR(4));
```

It is tempting to want to use GR without indices, but this will cause a compiler error. Each time the program cycles back to the beginning of the program, PT1 will get reassigned new values based on the current values of GR.

As a second example of *indexing* into a group, suppose an application using a 7545-800S manipulator defines many points. Rather than *having* to place a LEFT or RIGHT command before every PMOVE, it is possible to encode this information into two groups. Then one subroutine **may** decode this information and perform the actual ?MOVE. Essentially, points will contain an implicit LEFT or RIGHT command before motion to the point is performed. This can be easily accomplished by using groups.

```
     ******************************************************
 1 PT1:NEW PT(500,500,0,0);
 2 PT2:NEW PT(-500,500,0,0);
 3 PT3:NEW PT(.500,-500,0,0);
 4 PT4:NEW PT(-500,-500,0,0);
 5 POINTS:STATIC GROUP(PT1,172,P73,12T4);
 6 INFO:STATIC GROUP(0,0,1,2); -- 0-CURRENT MODE, 1-LEFT
 7                                 2-RIGHT
 8 MAIN:SUBR;
 9 F.'■O    ISUBR(N);
10      TESTC(INFO(N),0,DOIT);      7 TEST FOR DON'T C
11      TESTC(INFO(N)„1,LE       )--- TEST FOR LEFT
12      RIGHT; BRANCH(DOIT);       -- RIGHT nom THEN MOVE
13      LEFTMODE: LEFT;            -- LEFT MODE THEN MOVE
14      DOIT: PMOVE(POINTD(N));
15      END;
16  PPMOVE(3);                     -- GO TO PT3 (IN LEFT MODS)
17  PPMOVE(1);                     -- GO TO Il1 (IN LEFT MOOS)
18  PPMOVE(4);                        GO TO PT4 (IN RIGHT MODE)
19  PPMOVE(2);                     -- GO TO PT2 (1N RIGHT MOCS)
20  END;
     ***********************k*****************************
```

By using the group INFO, it is possible to encode the configuration, The MOVE subroutine takes one parameter, which indicates to which point the manipulator will move. If INFO(N) contains a 0, then the current configuration is used. If INFO(N) contains a 1 **then the** 7545-800S is switched to LEFT mode before the move. Any **other value in** INFO(N) means that the 7545-800S is switched to RIGHT mode before the move. Thus only the declaration section of the program 'needs to **be** properly entered. The user does not have to be concerned with **arm** configurations throughout the program.

EXPRESSIONS

AML/Entry Version 4 allows arithmetic ex•ressions to appear in several commands. An expres s a mathematical equation es to a result. The result of the expression is then used in the command in which it appears. In this chapter, expressions in AML/Entry are discussed.

Expressions can consist of constants, counters, operators, formal parameters (see section on "Subroutines" on page 4-60) and built-in functions. If a formal parameter is used for the first time in an expression, then the compiler assumes that the actual parameter piiggige 1!asounter

Expressions in AML/E are identical to how they appear in other computer languages (i.e. IBM Basic). To add two counters, A and B, and store the result in a third counter C, one would use:

2‡11:SCANBIL_

The following could also be used:

```
SETC(C,((((A+B)))) );
SETC(C,B+A);
SETC(C,A+(+B));
```

Subtracting, multiplying, or dividing two counters is equally easy, as indicated below:

```
SETC(C,A-B);
SETC(C,A*B);
SETC(C,A/B);
```

Terms within parentheses are always evaluated before being used in the rest of the expression. Addition and subtraction have the same precedence, as do multiplication and division. But as in other languages, multiplication and division take precedence over addition and subtraction. Thus

```
SETC(C,10+5*2);
```

sets C to the value 20. Parentheses can be used to override the default precedences. To perform the addition first, the following would be used:

```
SETC(C,(10+5)*2);
```

causing C to be set to 30.

As an example, suppose one wanted to write a subroutine which performed the factorial function. The factorial function is a function which multiplies all the integers up to a given integer.

For example,

```
3 factorial = 1 * 2 * 3
5 factorial = 1 * 2 * 3 * 4 * 5
1 factorial = 1
```

The following AML/E program performs this

```
RESULT: STATIC COUNTER; MAIN: SUBR;

  FACTORIAL: SUBR(N);          --Performs N factorial, answer in RESULT
    SETC(RESULT,1);            --Initialize RESULT

   LOOP:
     COMPC(N < 2,DONE);        --Loop until N<2
     SETC(RESULT,RESULT*N);    --Multiply RESULT by N, N-1, N-2,...
     SETC(N,N.4);              -- until N=1
     BRANCH(LOOP);
   DONE:
     END;                      --End of FACTORIAL subroutine

  FACTORIAL(5);                --Computes 5 factorial
  FACTORIAL(14);               --Computes 10 factorial
  END;                         --End of Program
```

Note that to decrement N in the above program, the expression N-1 is used. The DECR function could also have been used. As another example, suppose a group of counters is used to keep production counts of an application, and one wanted to average these. The subroutine AVERAGE performs this. The rest of the main program increments and decrements the counters as needed, and when the average is needed, subroutine AVERAGE is called.

```
PRODSTATS: STATIC GROUP(0,0,0,0,0,0,0,0,0,0);--Holds the prod. counts

AVG: STATIC COUNTER;                   --Holds the average of above
MAIN: SUBR;

   AVERAGE: SUBR;                      --Computes the average
     I: STATIC COUNTER;                --Points into PRODSTATS
     SETC(I,1);                        --Start with the first value
     SETC(AVG0);                       --Clear average
    LOOP:                              --Loop until all 10 values
      SETC(AVG,AVG+PRODSTATS(I));      -- of PRODSTATS added
      SETC(I,I+1);                     --Increment pointer
      COMPC(I<11,LOOP);
    SETC(AVG,AVG/10);                  --Compute average from sum
    END;                               --End of AVERAGE subr
```

The compiler makes no attempt to evaluate an expression at compile time. Thus the user should attempt to reduce the expression as much as possible beforehand. For example,

```
SETC(C,3*(A+10)-1);
```

should be reduced to

```
SETC(C,3*A+29);
```

Similarly, division by 0 is not caught at compiler time. Thus

```
SETC(C,C/(10-2*5));
```

will cause a data error (Arithmetic Error) in the controller at run time.

Built-in Arithmetic Functions

Many built-in arithmetic functions are provided. These functions are allowed to appear in expressions wherever a counter can appear. The following functions are provided.

1. ABS(exp) - Will return the absolute value of the given expression. The absolute value of a number is the number itself if the number is positive, or -1 times the number if the number is negative (making it positive). For example,

   ```
   SETC(C,ABS(5.5));        --Sets C to 5.5
   SETC(C,ABS(0));          --Sets C to 0
   SETC(C,ABS(-5.2));       --Sets C to 5.2
   ```

2. ATAN(exp) - Will return the arctangent of exp in degrees. The arctangent will be the principal value (i.e. from -90 to +90 degrees). For example,

   ```
   SETC(C,ATAN(1));         --Sets C to 45 (i.e. TAN(45)=1)
   SETC(C,ATAN(5/3));       --Sets C to the angle made by the X-Axis and
                            --the line segment from (0,0) through (3,5)
   ```

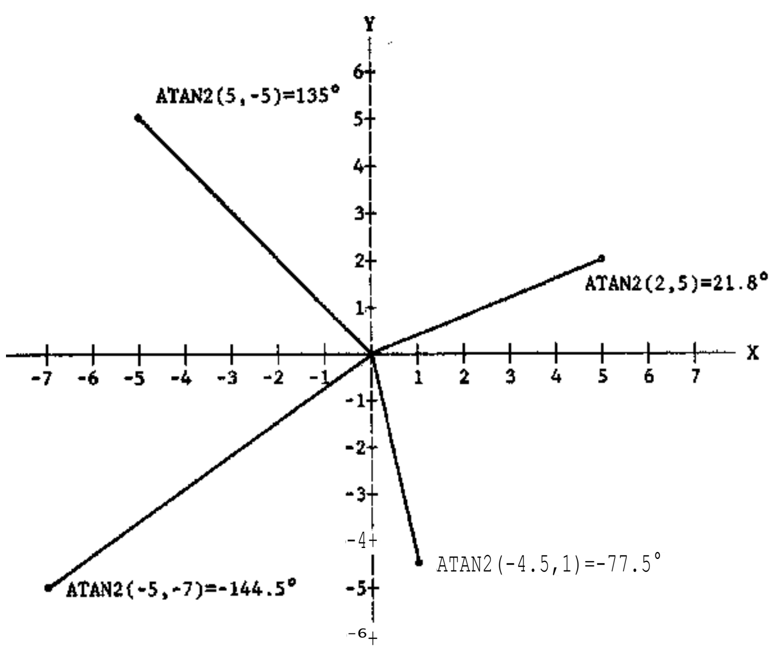
   The arcsine and arccosine functions are not provided by AML/E. However, one may use the following formulas to achieve the same result:

   ```
   ASIN(x) = ATAN(x/5ORT(1-x*x))
   ACOS(x) = 90 - ATAN(x/SORT(1-x*x))
   ```

3 ATAN2(expl,exp2) - Will return the arctangent of expl/exp2 in degrees. This has the advantage over ATAN(expl/exp2) in that the actual angle will not get replaced with the principal angle. The resultant value will be from -180 to 180 degrees. If both expl and exp2 are 0, then a DE (52) will occur. The ATAN2 function returns the angle made by the X-axis and the point whose Y coordinate is expl and whose X coordinate is exp2. For example,

```
SETC(C,ATAN2(5,3));        --Gives same result as ATAN(5/3)
SETC(C,ATAN20.5,5.5));     --Result is same as ATAN(1), which is 45.
SETC(C,ATAN2(-4.1,-4.1)); --Result is 180 degrees off from ATAN(1).
                           --Instead of returning 45, -135 is returned
                           --because the point (-4.1,-4.1) is in the
                           --third quadrant.
SETC(C,ATAN2(-10,0));      --Result is -90. This is because the point
                           --(0,-10) makes an angle of -90 degrees with
                           --the positive X-Axis.
SETC(C,ATAN2(10,0));       --Result is 90. This is because the point
                           --(0,10) makes an angle of 90 degrees with the
                           --positive X-Axis.
SETC(C,ATAN2(0,10));       --Result is 0 because the point (10,0) is on
                           --the positive X-Axis.
SETC(C,ATAN2(0,-10));      --Result is 180 because the point (-10,0) is
                           --on the negative X-Axis.
SETC(C,ATAN2(0,0));        --Will cause a run-time data error.
                           --The error number will be 52.
```

This can be made conceptually clearer by considering the result of the ATAN2 function graphically, as shown in the following figure.



Graphic Representation of the ATAN2 Function

4. COS(exp) - Will return the cosine of the angle given by the expression. Expression must be the angle in degrees. For example,

```
SETC(C,COS(0));          --Sets C to 1 (i.e. COS(0)=1)
SETC(C,COS(90));         --Sets C to 0 (i.e. COS(90)=0)
SETC(C,CO5(-120));       --Sets C to -.5 (i.e. COS(-120)=-.5)
```

5. CSTATUSO - Will return the communications status. The parentheses must be present. The following two AML/E Version 4 statements are equivalent:

```
CSTATUS(C);
SETC(C,CSTATUS());
```

By using CSTATUS as a function, a counter will no longer have to be used to store the result for a later comparison in a COMPC or TESTC command. See "Commands That Allow Expressions" on page 4-54 and "CSTATUS Command" on page 4-17 for more on this. Both forms are provided, the command and the function form. The command form must have a single counter parameter, and the functional form must have no parameters.

6. FROMPTIpt,expl_ - Will return one of the four coordinates of the given point depending on the expression. For example,

```
SETC(C,FROMPT(PTA,3));    --Sets C to the Z coordinate of the point
PTA.
```

The real expression will be truncated to an integer, and if the resulting integer is not 1,2,3 or 4 (X,Y,Z or Roll), then a data error (code = 50) will result.

7. JISTOGS.(4—...-Will return the motion status. The parentheses must be present. The following two AML/E Version 4 statements are equivalent:

```
MSTATUS(C);
SETC(C,MSTATUS());
```

By using MSTATUS as a function, a counter will no longer have to be used to store the result for a later comparison in a COMPC or TESTC command. See "Commands That Allow Expressions" on page 4-54 and "MSTATUS Command" on page 4-19 for more on this. Both forms are provided, the command and the function form. The command form must have a single counter parameter, and the functional form must have no parameters.

8. SIN(exp) - Will return the sine of the angle given by the expression. Expression must be the angle in degrees. For example,

```
SETC(C,SIN(0));           --Sets C to 0 (i.e. SIN(0)=0)
SETC(C,SIN(-90));         --Sets C to -1 (i.e. SIN(-90)=-1)
SETC(C,SIN(150));         --Sets C to .5 (i.e. SIN(150)=.5)
```

9. SQRT(exp) - Will return the square root of the expression. If the expression is negative then a data error (code 51) will occur. For example,

```
SETC(C,SQRT(4));          --Sets C to 2
SETC(C,SQRT(100));        --Sets C to 10
SETC(C,SQRT(0));          --Sets C to 0
SETC(C,SQRT(-ABS(C)));    --Will cause a data error unless C is 0
```

10. TAN(exp) - Will return the tangent of the angle given by the expression. Expression must be the angle in degrees. If the angle is 90 degrees (or a multiple thereof) then a data error (arithmetic error) will occur.

```
SETC(C,TAN(0));           --Sets C to 0 (i.e. TAN(0)=0)
SETC(C,TAN(135));         --Sets C to -1 (i.e. TAN(135)=-1)
SETC(C,TAN(-90));         --Causes a run time error
```

11. TESTI(port number) - Will return either 0 or 1, depending on whether the specified DI port is active or not. For example,

```
SETC(C,TESTI(5));       --Sets C to either 0 or 1, depending on DI 5
```

This function may be used to see if any or all of a group of digital inputs are active. For example, to test that at least one of DI(1) DI(4) are active, one would use:

```
COMPC(TESTI(1)+TESTI(2)+TESTI(3)+TESTI(4) > 0 , OK);
```

If at least one of these DI is on, then control branches to the label OK. To test that all of DI(1) - DI(4) are active, one would use:

```
COMPC(TESTI(1)+TESTI(2)+TESTI(3)+TESTI(4) = 4, OK);
```

If all of these DI are on, then control branches to the label OK.

12. TESTP(pallet) - Will return the current part number of the --____n=riti   nfthat. For example,

```
SETC(C,TESTP(PAL));     --Sets C to the current part number of PAL
```

By making TESTP a function call, users may now use the more powerful COMPC verb for testing the current part number of a pallet. See "Commands That Allow Expressions" on page 4-54 and "TESTP Command" on page 4-78 for more on this.

13. TRUNC(exp) - Will truncate the real expression to an integer. The numbers are "rounded downwards." For example,

```
SETC(C,TRUNC(12.1));     --Sets C to 12
SETC(C,TRUNC(.99));      --Sets C to 0
SETC(C,TRUNC(-.0001)); --Sets C to -1 (note that negative values are
                         --rounded downwards).
```

AML/E does not provide the MOD function, which returns the remainder of an interger division. This may be performed by using the TRUNC function. For example, A MOD N (the remainder after dividing A by N) can be attained from:

```
A-N*TRUNC(A/N)
```

Comands That Allow Expressions

Expressions are allowed in the following /E commands:

```
COMPC(exp < exp,label);

SETC(counter,exp);

TESTC(exp,exp,label);

ZMOVE(exp);
```

Using expressions and functions can make AML/E programs very efficient. For example, applications should use the arithmetic function form of CSTATUS and MSTATUS instead of the command form, as shown below:

```
COMPC(MSTATUS()=0,CONTINUE);
```

Using the command form would require the declaration of a counter, the MSTATUS command, and then finally the COMPC command. Three AML/Entry lines have been replaced by one.

Many other AML/Entry commands (i.e. GUARDI, LINEAR, PAYLOAD, ZONE, to name *a* few) do not allow expressions as arguments. They do, however, allow counters as arguments. An alternate solution is to use the SETC command to set a counter, and then use the counter in these commands.

Extsressions are not *allowed* within the ITERATE statement. Even though most of t e above commands may appear in an ITERATE statement, they must appear "stand-alone" for expressions to be legal. When these verbs are used inside an ITERATE statement, the arguments that are unpacked from the aggregates and actually used depend on the command. See the discussion of SETC, TESTC, and ZMOVE in Appendix A, "Command/Keyword Reference."

EXAMPLE APPLICATIONS USING EXPRESSIONS

Circular Motion

Suppose an application required a manipulator to move in a circle centered at the point (X0,Y0) with a radius of R. This can best be performed if one uses polar coordinates. The circle in the X-Y plane can be represented in polar coordinates as follows:

```
X = XO + R*COS(THETA)
Y = YO + R*SIN(THETA)
```

As THETA is stepped from 0 to 360 degrees, a circle is traced. In order to make the robot move in a circle, these expressions are used to calculate intermediate points on the circle. The following program performs this.

```
X: STATIC COUNTER;
Y: STATIC COUNTER;           --The temporary locations
XO: NEW 0;                   --Center of circle at (0,500)
Y0: NEW 500;
R: NEW 150;                  --150 mm radius
MAIN: SUBR;
    DEGREE: STATIC COUNTER;  --Steps from 0 to 360 degs, 3 per step
    SETC(DEGREE,0);          --Start from right side of circle
    ZONE(15);                --Minimum settle time
    PAYLOAD(11);             --Slow speed

   LOOP:                     --Loop from 0 to 360 degrees
    SETC(X,X0+R*COS(DEGREE));--X location of next point
    SETC(Y,Y0+R*SIN(DEGREE));--Y location of next point
    PMOVE(PT(X,Y,0,0));      --Perform move
    SETC(DEGREE,DEGREE+3);   --Increment for next point
    COMPC(DEGREE<=360,LOOP); --End of Loop

    END;                     --End of program
```

Treating DI As Integers

AML/E programs wishing to take a course of action based on a digital input use the TESTI verb to read in the value of the input. However if action needs to be taken based on a combination of inputs, then using math is the most general solution. The solution is to treat a consecutive string of digital inputs as an unsigned binary number. For example, an application can treat digital inputs 1 through 4 as an unsigned binary number. A binary number consists of just 0's and 1's. The rightmost digit is worth 1, the second from the right is worth 2, the third from the right is worth 4, the fourth from the right is worth 8, etc. This may be better understood by considering some examples.

```
Binary Number      Decimal Value
0  0  0  0  0            0
0  0  0  0  1            1   (1*1)
0  0  1  0  1            5   (1*4 + 1*1)
1  1  0  0  0            24  (1*16 + 1*8)
1  1  1  1  1            31  (1*16 + 1*8 + 1*4 + 1*2 + 1*1)
|  |  |  |
|  |  |  |  └ Worth 1
|  |  Worth 2
|  |  |           Worth 4
      Worth 8
|__|                   Worth 16
```

In order to write a Subroutine that reads in the digital inputs and returns the decimal value, we must first decide how to map the digital inputs to a binary number. The solution given uses two parameters which may be treated as counters (parameters are discussed in "Using Subroutines with Parameters" on page 4-68). The first one, called FIRST, designates the digital input that will be the left digit of the binary number. The second one, called LAST, designates the digital input that will be the right digit of the binary number. FIRST must be <= LAST. For example, if FIRST=1 and LAST=4, then the resulting decimal number will range from 0 (all DI off) to 15 (all DI on).

The algorithm that converts the 1's and 0's of the digital inputs to the actual decimal number is quite simple. First note that each digit except the right most is multiplied by some power of 2. Second note that the power of 2 to which each digit is multiplied is one higher the further left in the binary number you go. Thus in order to convert the binary number to a decimal number, a loop can be used. Each step through the loop, multiply the attained decimal number by 2 and add the new digital input, starting from the left most digit, working toward the right most digit. The subroutine that does this is named TESTDIS, and is shown below inside a program named MAIN. Try tracing execution with several values for FIRST and LAST to convince yourself the algorithm works properly. The decimal number is placed in the global counter RESULT.

```
RESULT: STATIC COUNTER;       --Will contain result of DI string
MAIN: SUBR;
  TESTDIS: SUBR(FIRST,LAST); --FIRST is the lowest DI (most
                              --significant), LAST is the highest
                              --DI (least significant).
  SETC(RESULT,0);             --Initialize result

  LOOP:                       --Loop until FIRST>LAST
   COMPC(FIRST>LAST,DONE);
   SETC(RESULT,2*RESULT+TESTI (FIRST)); --Each time thru, add the next
                              --most significant bit and multiply
                              --all the previous bits by 2.
   SETC(FIRST,FIRST+1);       --Go to next most significant bit
   BRANCH(LOO?);              --Continue
DONE:
  END;                        --End of TESTDIS subroutine



  TESTDIS(1,8);               --Look at first 8 DI
  TESTC(RESULT,255,ALLON);    --If all the DI are on then branch
                              --to label ALLON
```

Determining the Row and Column of a Part in a Pallet

Suppose that an application needs to determine the row or column of the current part of a pallet. By using the TESTP function, this is possible. The following code places the row number in ROW and the column number in COL. Assume PPR contains the number of parts per row of the pallet being used. T is a temporary counter.

```
SETC(T,TRUNC((TESTP(PAL)-1)/PPR)); --Holds the number of skipped rows
SETC(ROW,T+1);                      --Adding 1 gives the actual row
SETC(COL,TESTP(PAL)-T*PPR);         --Subtracting the number of parts
                                    --in the skipped rows gives the
                                    --number of columns
```

Compiler Directives

AML/E Version 4 supports two compiler directives:

Directive	Description
	Includes a file within a file Causes a page break in the listing file

Using the Include Compiler Directive --%I

AML/Entryallocszyti toh-icluUbl_es This means that you are able to include commands in one file that cause additional files to be read by the compiler. The lines read in by the compiler from Included files are included in the listing file and the output (.ASC) file. Format of the command is outlined below.

```
--%I'filespec'
```

The percent sign immediately following a comment marker (the double dash) indicates a Compiler directive. The capital I indicates the desired function is an Include (a lower case i is also recognized). After the I, the file name specification follows in single quote marks. Embedded blanks are not allowed. The compiler does not a Include files (a file being included can not itse f contain an 'Include command). The filespec of the file to be Included must include the file's extension. If not specified, the compiler does not assume .AML. If the drive is not specified in the filename, the compiler assumes the file is on the same 'drive as the AML/Entry program. The filespec may not include a path; only a drive, filename, and file extension may *appear.*

The compiler issues an "Including file" message whenever a file is included. An "Including file" message is issued for both the "Reading Input File" phase and "Converting AML/E Program" phase. If an error occurs in an Include file, the error is in the file named in the last "Including File" message

In the .LST file, the lines of an Included file are identified by a % (percent) sign that follows the line address indicator. If an err&?—rr found on a line within an Include file, the line number reported is numbered with respect to that file and is followed by a percent sign. In the example outlined below, two lines of code are included in the subroutine MAIN by the command --%I'A:DATA.AML'. The example below shows the program (MAIN.AML), the included data (DATA.AML), and the listing (MAIN.LST).

```
*** MAIN.AML File ***        *** DATA.AML File ***

MAIN: SUBR;                   PT1:NEW PT(650,0,0,0);
--WA:DATA.AML'                PT2:NEW PT(0,550,0,0);
PMOVE PT1;
PMOVE PT2;
END;
```

```
 *** MAIN,LST File ***

 MAIN:SUBR;
 --%I'A:DATA.AML'
%PT1:NEW PT(650,0,0,0);
%PT2:NEW PT(0,550,0,0);
 PMOVE PT1;
 PMOVE PT2;
 END;
```

Using the Page Compiler Directive 7-%P

This compiler directive causes a page break in the listing (.LST) file. It allows you to control the paging within the listing file.

Format of the command is outlined below.

SUBROUTINES

A subroutine is a small unit of the program with a clearl fined eginning •n• ' a• s a su•rou usually work .ogether to 1777PCific job, such as moving the arm. This section includes information on the various aspects of subroutines including:

- System subroutines
- User subroutines
- Subroutines and formal parameters
- Name restrictions on parameters
- Using Subroutines
- Ownership

System Subroutines

System subroutines are part of the AML/Entry program. These subroutines rved names. For example, **PMOVE** is a system subroutine that moves the arm to a point in the work space when the statement executes in your program. You cannot alter system subroutines, nor can you use a name that is reserved for a system subroutine for other purposes.

User Subroutines

You have already seen that an outer subroutine is required for every program. The outer subroutine, like all other subroutines, uses a SUBR statement and an END statement. Your program can have user subroutines within this outer subroutine.

User Subroutines in the AML/Entry Program

An example:

```
      ****************************************************
    1 MAIN:SUBR;                     --OUTERMOST SUBROUTINE
    2     STATION1:SUBR;             --LEVEL 1
    3
    4     END;                       --END STATION1
    5
    6     STATION2:SUBR;             --LEVEL1
    7         STATION3:SUBR;         --LEVEL 2
    8
    9         END;                   --END STATION3
   10
   12     END;                       --END STATION2
   13
   14
   15
   16
   17 END;                           --END OF PROGRAM
   18
      ****************************************************
```

The outer subroutine which is your application program, has a SUBR (line 1) and an END (line 17). STATION1 and STATION2 are level 1 subroutines within the outer subroutine. STATION1 and STATION2 are level 1 subroutines because they are contained in the outer subroutine. Each subroutine has its own END statement (lines 4 and 12 respectively).

STATION3 is a level 2 subroutine because it is contained in a level 1 subroutine. There are no limits on the number of levels of subroutines in the program.

Development of User Subroutines

User subroutines are an important part of the development of your programs. It is to your advantage to write a subroutine and keep calling it wherever some action is repeated in different parts of the program. It not only gives you the convenience of less typing, but helps save program space in the controller. Saving space in the controller allows you to write longer programs. By creating user subroutines, you also make your programming easier to follow and update. To call a user subroutine you call its name as a command in your main program, or in any other subroutine after it.

User subroutines take one of two forms.

```
id:SUBR(parameter1,parameter2,....);
     statement)
     statement2


     statementn
END;
```

or

```
id:SUBR;
   statementl
   statement2

   statementn
END;
```

The ID is an identifier that you use like a name. It has the below listed format and characteristics.

- Up to 72 characters
- First character must be alphabetic
- Remaining characters must be alphabetic, numeric, or underscore (._)
- An underscore can not be the last character
- Special characters, such as an asterisk (*), are not permitted.

The colonSilis a definition operator required between the identifier and the SIORfilm.

The SU BR term is a keyword that identifies the start of a subroutine.

TheElm12 1, are placed around the parameters. If the subroutine has no external parameters, the parentheses are not required.

The semicolon (;) is a delimiter that identifies the end of an AML/Entry program statednT7

The statements in the subroutine define the tasks you want performed when the **subroutine** is executed.

The END keyword identifies the last line of the subroutine. The END IEZS"MMmd-ret.arrit the program execution to the next command to be executed after the line that called the subroutine.

Formal Parameters in Subroutines

Parameters are variable or constant values appearing in a statement. Parameters restrict or determine the specific form of the statement.

When using subroutines in your programs you can use formal and actual parameters. Formal parameters are variables that are enclosed in the parentheses after the SUBR keyword. Every parameter in the parentheses of the SUBR statement must be used somewhere in that subroutine. Formal parameters do not have a value associated with them until the subroutine is used and a value is provided. The statement that calls the subroutine provides the parameter enclosed in parentheses after the calling statement. The parameter in the calling statement is an actual parameter because the value is known.

Subroutines with formal parameters are very useful when you want to repeat the same series of commands with a large number of constants of the same kind of data (numbers, points, strings, characters). Since the parameters are passed to the subroutine, you can change different aspects of the program every time the subroutine is called.

When you have a subroutine with formal parameters, the formal parameters do not have to be declared. The way they are used in the digammutiao.- determines the kind of data (naMberb •°ants stin s c ; that m orma parameter.

Parameter Passing

All parameters are passed by value. This means that all formal PIYMMuters (values) are spies of the caller's parameter (actual value). If the called subroutine changes a formal parameter's value, it does not change the value used by the ciaTins-anImadjsul,--135;;;;ions can not be passed as a parameter to g sunroutine. Inste'ad, ass on to a counter and pass the counter.

Example of Subroutine with Formal Parameters

The following is a fragment of a program that uses subroutines with formal parameters. The subroutine PICKUP has the parameters PLACE and TIME which are formal parameters. There are no initial values associated with either parameter.

The subroutine PICKUP does not execute in this program until it is called by its name.

The formal parameters PLACE and TIME have their data types determined by their usage within the subroutine. PLACE is used with the command PMOVE. Since the command PMOVE requires data to be of the type PT, any data passed to parameter PLACE must be of this type. Parameter TIME is used with the DELAY command. Any data passed to the subroutine to be used by parameter TIME must be in the valid working range for the DELAY command.

```
*******************************************************
1 NEWPROG:SUBR;                    --BEGINNING SUBROUTINE
2
3   PICKUP:SUBR(PLACE,TIME);
4        PMOVE(PLACE);
5        ZMOVE(-250);
6        GRASP;
7        DELAY(TIME);
8        ZMOVE(0);
9      END;


*******************************************************
```

Restrictions on Parameters

When using subroutines, certain restrictions exist concerning parameter names and calls.

Formal Parameter Names Restrictions

The name of a formal parameter can not be equivalent to

- Thieenan of alob
- The name of a subroutine

Actual Parameter Assignment Restrictions

An actual parameter in the calling statement can not be

-
- Aggregates
- Subroutine names
- Labels
- A reserved word

Using Subroutines without Parameters

- Except for the outermost subroutine, a subroutine is not executed until it is called by name. Put the identifier (name), followed by a semicolon (;), at the location you want the subroutine to execute.

  Example:

```
STATION:SUBR;
   PMOVE(PT(300,400,-100,0);
   ZMOVE(-250);
   ZMOVE(0);
   END;

STATION; --CALLED LATER IN THE PROGRAM
```

When this program is executing, the subroutine STATION is not activated until the command line :STATION;" The program then executes the entire STATION subroutine as if'it was riated there.

- The subroutine must be declared before you can call it. In other words, the contents of the -gUbitttine must be listed within the program before it can be called for execution.

- When subroutines are contained within other subroutines, the inner or nested subroutines must be declared before an executed within the outer su

  Example:

```
OUTER:SUBR;
     STATION:SUBR;
           PMOVE(PT(300,400,-100,0);
           ZMOVE(-250);
           ZMOVE(0);
           END;

   STATION;
   WAITI(6,1,10);
END;
```

Within the outer subroutine, the WAITI statement is an executable statement. The WAITI statement must follow subroutines listed within the outer subroutine, in other words the WAITI command cannot precede the STATION subroutine declaration. If the subroutine STATION contained a nested subroutine it would have to be declared before the PMOVE command.

Rules______Calling Subroutines

The following example is used to illustrate the rules for calling subroutines. The rules for calling subroutines are as listed below.

1. A subroutine must be previously declared, and

2. The called subroutine must either:

   a. Be at the same level as the calling subroutine, oz,

   b. Be owned by the calling subroutine.

Example:

```
 1 MAIN:SUBR;               --OUTER LEVEL
 2 A;SUBR;                  --LEVEL 1
 3       B:SUBR;            --LEVEL 2
 4             C:SUBR;      --LEVEL 3
 5             END;         --END C LEVEL 3
 6       C;                 --B CALLS C
 7       END;               --END B LEVEL 2
 8  8;                      --A CALLS B
 9  END;                    --END A LEVEL 1
10
11 D:SUBR;                  --LEVEL 1
12       E:SUBR;            --LEVEL 2
13             F:SUBR       --LEVEL 3
14             END;         --END F LEVEL 3
15       F;                 --E CALLS F
16       END;               --END E LEVEL 2
17       G:SUBR;            --LEVEL 2
18       END;               --END G LEVEL 2
19    A;                    --D CALLS A;
20    E;                    --D CALLS
21    G;                    --D CALLS G
22    END;                  --END D LEVEL 2
23 A;                       --MAIN CALLS A, A CALLS B, B CALLS C
24 D; --MAIN CALLS D; D CALLS A, A CALLS B, B CALLS p,
25      --D CALLS E, E CALLS F, D CALLS G
26 END;                       --MAIN END STATEMENT
```

In this example of calling subroutines, all the possible allowed calls are shown. The structure of the program is as follows:

- The program has the required outer subroutine, named MAIN.
- There are two level 1 subroutines, named A and D. Subroutine A precedes subroutine D in the listing.
- Subroutine A has a level 2 subroutine (B) and a level 3 subroutine (C). The level 3 subroutine is owned by the level 2 subroutine.
- Subroutine D has two level 2 subroutines (E and G). Subroutine E owns a level 3 subroutine (F).

In the example, subroutine A can call subroutine B because B has already been declared (rule 1) and it is owned by A (rule 2b). Notice that subroutine A can't call subroutine C because A does not own C (rule 2b). Therefore,MfbY3tITTirn.i'—i-sf ocaSUroutine C.

Subroutine D can call subroutine A because A has already been declared (rule 1), and it is at the same level as D (rule 2a). Subroutine D can also call subroutines E and G since they have already been declared (rule 1) and owned by D (rule 2b).

Subroutine E cannot call subroutine G because it violates rule 1. That is, G has not been declared yet.

Using Subroutines with Parameters

When you send multiple parameters to a subroutine, the order the parameters are listed determines the order in which they are sent. The next example has multiple formal parameters in the SUBR statement. When the subroutine is called (line 13), the actual parameters POINT1 and 1.0 are provided.

The global constant POINT1 is a location declared on line 1. When you place parameters in the SUBR statement and then call the SUBR by name withsaxametere-r-the order of the list in the SUBR statement must match the order of atactuslm4arnetera in the calling statement.

Example

```
.... ******************************************4f*********
    1 POINT1:NEW PT(300,400,-100,0);
    2
    3 MAIN:SUBR;                         --BEGINNING SUBROUTINE
    4
    5   PICKUP:SUBR(PLACE,TIME);
    6        PMOVE(PLACE);
    7        ZMOVE(-250);
    8        GRASP;
    9        DELAY(TIME);
   10        ZMOVE(0);
   11      END;
   12
   13   PICKUP(POINT1,1.0);
   14 END;
....******************************************************
```

Once you have declared a subroutine formal parameter as a particular type of data, all later calls to the subroutine must use the parameter with a similar type of variable. PLACE is used with the command PMOVE. Since the command PMOVE requires data to be of the type PT, any data passed to parameter PLACE must be of this type. Parameter TIME is used with the DELAY command. Any data passed to the subroutine to be used by parameter TIME must be a number in the valid working range for the DELAY command.

Example:

Valid call to the SUBR PICKUP   PICKUP(POINT1,20,0);

Invalid call to the SUBR PICKUP   -   PICKUP(50,20.0);

The next example covers all facets of subroutine usage discussed in this section so far. It is similar to previous examples, but it is written to perform repetitive tasks by using subroutines.

The subroutines are called by placing their names on the lines in the program where you want them to execute. In this example the calls to the subroutines occur on lines 21, 22, 23, and 24. The subroutine PICKUP requires two parameters to execute every time it is called.

Example:

```
   ******************************************************
 1 POINT1:NEW PT(300,400,-100,0);
 2 POINT2:NEW PT(400,300,-50,0);
 3 MAIN:SUBR;                              --BEGINNING SUBROUTINE
 4
 5   PICKUP:SUBR(PLACE,TIME);
 6        PMOVE(PLACE);
 7        ZMOVE(-250);
 8        GRASP;
 9        DELAY(TIME);
10        ZMOVE(0);
11      END;
12
13 DROPOFF:SUBR;
14        DPMOVE(<10,-5,-10,-20>);
15        ZMOVE(-200);
16        RELEASE;
17        DELAY(1.0);
18        ZMOVE(0);
19      END;
20
21   PICKUP(POINT1,1.0);
22 DROPOFF;
23   PICKUP(POINT2,1.0);
24 DROPOFF;
25 END;             --END OF PROGRAM
   ******************************************************
```

It is important to remember that no statements are executed until all the subroutines have been fully declared (line 19). When executed, the statements must parameters in their proper order.

In the above example, POINT1 and POINT2 are global declarations for points. When the subroutine PICKUP is called in line 21, the actual parameter POINT1 is used and 1.0 is used for TIME. When the subroutine PICKUP is called in line 23, the actual parameter POINT2 is used and 1.0 is used again for TIME.

Using the ITERATE command to Repeat a Subroutine

ITERATE statements can be used to shorten re•etitive ro rams ev or In the following examp e, t e ITERATE statement is used to repeat a user subroutine called WORK.

Note: Using ITERATE to repeat a function does not make a program operate any faster or save any controller space. ITERATE is useful for saving lines in the actual written program, and in 'increasing readability.

```
*************************************************************
 1 POINT1:NEW PT(300,400,-100,0);     --GLOBAL POINT DECLARATION S
 2 POINT2:NEW PT(400,300,-50,0);
 3 POINTS:NEW <POINT1,POINT2>;        --AGGREGATE OF DECLARED POINTS
 4 MAIN:SUBR;                         --BEGINNING SUBROUTINE
 5    WORK: SUBR(LOCATION);           --INNER SUBR OWNED BY MAIN
 6       PICKUP: SUBR(PLACE,TIME);    --INNER SUBROUTINE OWNED BY
 7         PMOVE(PLACE);              --WORK
 8         ZMOVE(-250);
 9         GRASP;
10         DELAY(TIME);
11         ZMOVE(0);
12        END;
13
14       DROPOFF:SUBR;                 --ANOTHER INNER SUBROUTINE
15         PMOVE(PT(100,500,0,-20));   --OWNED BY WORK
16         ZMOVE(-200);
17         RELEASE;
18         DELAY(1.0);
19         ZMOVE(0);
20       END;
21
22     PICKUP(LOCATION,1.0);          --THESE COMMANDS ARE EXECUTED
23     DROPOFF;                       --WHEN SUBROUTINE WORK IS USED
24     END;                           --END SUBROUTINE WORK
25   ITERATE('WORK',POINTS);
26
27 END;            --END OF PROGRAM
****************************************,C*** A**************
```

This program has the required outer subroutine and two more levels of user subroutines. The subroutine WORK is a level 1; PICKUP and DROPOFF are level 2.

The aggregate is POINTS, which has two point locations POINT1 and POINT2. The ITERATE statement calls WORK the first time and passes the parameter POINT1 to the subroutine. Within WORK is the subroutine PICKUP, which is called by its name. A formal parameter LOCATION has data type PT and takes POINT1 for its value the first time. It is passed along with 1.0 to subroutine PICKUP. The subroutine DROPOFF is executed after PICKUP but no actual parameters are passed, since none are required.

A simple flow chart for the program would be:

1. Statement 25 is first executed and POINT1 is passed to subroutine WORK.

2. WORK passes POINT1 and 1.0 as formal parameters to subroutine PICKUP

3. PICKUP executes, and then DROPOFF executes

4. Program returns to the ITERATE statement on line 25

5. Statement 25 is executed again, POINT2 is passed to the subroutine work, and the process continues as before.

If more values were present in the aggregate, the program would continue in the same manner until all the values in the aggregate were used.

Ownership and Multiple Name Occurrence

These two concepts are very important to the use of subroutines. They are essential to subroutine structure.

If a subroutine declares a variable, it is the owner of that variable. The variable cannot be used anywhere else in the program. If a subroutine defines another subroutine, the outer subroutine owns the inner subroutine, and the inner subroutine may only be accessed by the outer subroutine.

Example:

```
MAIN:SUBR;
 ANEW 3;
    INNER:SUBR(B);
    C:NEW 5;


    END;



 END;
```

In the example, MAIN owns both the constant A and the subroutine INNER. Subroutine INNER owns formal parameter B and constant C.

Since the subroutine MAIN owns the constant A the subroutine INNER can't use it. The same is true for the constant C, it is owned by the subroutine INNER and cannot be accessed by MAIN. If any other subroutines are declared outside of MAIN they cannot access the subroutine INNER. In the case of the formal parameter, you can only send the parameter B to the program when you call the subroutine INNER.

Names in your program may look alike. If they have different ownership and different definitions, they are not alike.

Example:

```
MAIN:SUBR;
 ,OBJE9ILNEW 3;
    INNER:SUBR;
      OBJECTLNEW PT(400,300,0,0);
      PMOVE(OBJECT);            --OBJECT IS A POINT HERE

    END;

  WAITI(OBJECT,1);              --OBJECT IS AN INTEGER

 END;
```

In this example OBJECT is two different types of data. When used by the subroutine INNER it is a point; when used by the subroutine MAIN it is an integer. This is possible because each subroutine owns a local constant named OBJECT, and local constants can only be used in the subroutine that declares them.

Global declarations are accessible to all levels of a program. Attempting to redefine a global declaration is not permitted.

Note: Even though it is permitted to have identical names for different variables, it can be very confusing to anyone trying to use the program. Therefore, identical name usage should be avoided whenever possible.

ADDITIONAL TOPICS FOR PROGRAM ENHANCEMENT

So far in this chapter you have been introduced to the AML/Entry language, its structure and basic concepts. This final section of the chapter covers three additional topics that will further enhance your program. The topics are:

- Pallet
- Region
- Host Communications

Pallet

A major application of robotic systems is sorting and packing of items. These tasks require the manipulator to move to a large number of distinct locations that are related in a simple way. Rather than requiring you to calculate all these points explicitly, the palletizing aids provided by AML/Entry enable such tasks to be defined by the palletizing commands.

The palletizing extensions aid in two ways.

1. The location and orientatio •allet is inde•endent of th tas •eing described.

2. The task can be described using names, following their declarations.

Pallet Description

A pallet is aregularg nento' emssan a work surface. A regular arrangement is a pattern where the relationship between one item and the next is the same for all items.

A pallet is organized in rows; each row contains 7fixed number of items, with constant spacing between items. The spacing r Mr"1"1: also constant.

The number of items in a pallet is the product of the number of items per row and the number of rows. Each item is given a number. The first item is at the lower left corner_of the2111at. and is item one. The 'Tat part iirtrie row is item 2, and so on to the end of the row. Numbering continues to the last item, which is at the upper right C.0 rnar of •tjaa_pailt,,The following diagram illustrates the numbering of a pallet containing 12 parts in three rows of four items each.

```
 ________________________________
|                      UR        |
|     9     10    11   12        |
|                                |   ppr = 4
|     5     6     7     8        |     n = 12
|--------------------------------|
|     1     2     3     4        |
|     LL               LR        |
```

Pallet is a keyword and its declaration requires parameters as part of the statement. This information provides orientation data, based on the coordinates of parts located in three of the four pallet corners, identifies the number of rows in the pallet, and the total number of parts in the pallet.

A pallet is declared with the STATIC keyword:

```
name : STATIC PALLET(LL,LR,UR,PPR,N);
```

The pallet is defined by the arguments listed below.

1. Defining the location (as a PT) of the lower left part (LL of the pallet (part 1), the IcTarnaOf the lower right par) of the pallet (part 4), the location of the upper right part (UR) of the pallet (part 12).

2. Specifying the r-row (4) (PPR) and the total number o parts (12) (N).

Orientation of the pallet is automatically assumed to be perpendicular to the Z-axis. In the coordinates that declare the lower-left, lower Fignr7-Nun-uNizi- right corners of the pallet, the coordinate for the Z-axis is ignored. Even though the Z-coordinate maintains its last position, regardless of what is entered here, it must be included when working with servoed Z arms. Any palletizing application does not work correctly if the declared pallet is non-perpendicular to the Z-axis of the manipulator.

A one-dimensional pallet can either consist of one row or one column.

EXAMPLE:

```
LL: NEW PT(0,500,0,0);
LR: NEW PT(-300,500,0,0);
UR: NEW PT(-300,200,0,0);
N: NEW 5;
P:  STATIC PALLET(LL,LR,LR,N,N);
Q:  STATIC PALLET(LR,LR,UR,1,N);
```

P is a one dimensional pallet with one row, Q is a one dimensional pallet with one column. Note that for a one dimensional pallet with one row, where an upper right point would normally be specified, the lower right point is repeated. For a one dimensional pallet with one column, where a lower left point would normally be specified, the lower right point is repeated. Also note that for a one dimensional row, the number of parts per row equals the number of parts, and for a one dimensional column the number of parts per row equals 1. In either case, the first part is the part at LL and the last part is the part at UR.

PALLETIZING Commands

Once a pallet is defined, an internal representation of the pallet is maintained to indicate the s•eci is ()cation o 6 11 When first de :re•, t e current part is un•e ine . ou must use the SETPART compaR4to change the current part number to a valid part number. this is done, ,NEXTPART and PREVPART may be used to automatically increment or decrement the current part number for the pallet. These commands are circular., When the last part has already been fetched via ETPART, a subsequent NEXTPART will set the current part back to the first part. Likewise, a PREVPART when the current part is the first part wraps the current part back to the last part. There are five commands available for use with pallets.

Command	Description
GETPART	Moves Arm to Current Part
NEXTPART	Increases Current Part By 1
PREVPART	Decreases Current Part By 1
SETPART	Sets Current Part Counter
TESTP	Tests Current Part Counter

GETPART Command

```
GETPART(pallet_name);
```

Instructs the arm to move to the location of the current part. The arm is moved to the appropriate X-Y location of the pallet. The Z-axis is unchanged from its current position. The roll Roll axis is set to tre" 'icirr-ur-ftie lower left point.

NEXTPART Command

```
NEXTPART(pallet name ;
```

The current part indicator can be advanced to the next part on the pallet.

If the current part indicator is at the last part, then NEXTPART changes the current part indicator to be the first part.

PREVPART Command

```
PREVPART(pallet_name);
```

To move backward through a pallet.

Similarly, if the current part indicator is the first part, then PREVPART changes the current part indicator to the last part.

SETPART Command

```
SETPART(pallet_name,value);
```

The current part can be set to a particular part number.

Where pallet_name is the name of a pallet, and value is a part number. The value can consist of an integer constant, counter, or formal parameter. Since parameters are passed by value, if the SETPART is used in a subroutine to set the part of a pallet that is passed as a formal parameter, then the part number of the actual pallet passed is not set. The part number of the pallet in the subroutine will be set, however.

TESTP Command

```
TESTP(pallet_name,testvalue,label);
```

The TESTP command compares the present pallet art number to If they are the same, a,branch is ma e o a label in the same subroutine. The matching label must be in the same subroutine as the TESTP. The testvalue can consist of an integer constant, counter, or formal parameter.

> **Note:** This is the command form of TESTP. With the command form of TESTP, the testvalue and label must be given. TESTP may also be used as an arithmetic function within expressions. In the latter form, testvalue and label are not given.

Using Palletizing Statements

In the following example, the 12 parts of the sample pallet are individually picked up, and put in an ejector. The ejector indicates by DI/DO when the part has been removed.

```
LL:     NEW PT(200,200,0,30);          --PALLET LOCATION DECLARATIONS
LR:     NEW PT(300,200,0,30);
UR:     NEW PT(300,300,0,30);
N:      NEW 12;
PPR:    NEW 4;

EJECTOR:    NEW PT(300,400,-100,0); --POINT WHERE PARTS ARE PLACED
EJECTOROK: NEW 5;
SAMPLE:     STATIC PALLET(LL,LR,UR,PPR,N);
MAIN: SUBR;
             SETPART(SAMPLE,1);
LOOP:       GETPART(SAMPLE) ; ZMOVE(-250); GRASP; DELAY(1); ZMOVE(0);
             PMOVE(EJECTOR);
FEEDCHK:    WAITI(EJECTOROK,1,10.5);
             ZMOVE(-250); RELEASE; DELAY(1 ); ZMOVE(0);
             NEXTPART(SAMPLE);               --GET THE NEXT PART, IF
             COMPC(TESTP(SAMPLE)<>1,LOOP); --STILL MORE LEFT
DONE:       BREAKPOINT;
END;
```

The above example uses the arithmetic function form of TESTP. The command form of TESTP could be used, but an additional BRANCH command is needed.

Palletizing and Formal Parameters

The palletizing function allows pallets to be defined in terms of formal parameters, as well as to be formal parameters. Passi ng .to a subroutine as a formal parameter is shown below in a program fragment.

```
PALL: STATIC PALLET(UL,LR,UR,PPR,PARTS);
                                               -- passing the name of
  S: SUBR(PAL);                                -- the pallet and
     GETPART(PAL);                             -- move to pickup point
     END;
     S(PAL1);                                  -- define the pallet
```

Note: The parameter is passed by value, therefore if a called subroutine changes the current part (executes SETPART, NEXTPART, or PREVPART), the value is changed only in that subroutine. The caller's copy Zr--1-114-pa-1.1 is not dErFa.

Defining a pallet using formal parameters is shown below in a program fragment.

```
MAIN: SUBR;
   DEPAL: SUBR(UL,LR,UR,PPR,PARTS);
    PAM: STATIC PALLET(UL,LR,UR,PPR,PARTS);



    END;                    -- end of DEPAL

-- Start of Main Subroutine
   DEPAL(PT1,PT2,PT3,4,4);



   END;                     --end of MAIN subroutine
```

An example palletizing program is outlined below.

```
LLPT1: NEW PT(70,400,0,0);
LRPT1: NEW PT(510,400,0,0);
URPT1: NEW PT(510,800,0,0);
LLPT2: NEW PT(-730,-480,0,0);
LRPT2: NEW PT(-410,-480,0,0);
URPT2: NEW PT(-410,-80,0,0);
- - - - - - - - - - - - - - - - - - - - - - - -

  MAIN: SUBR;
DEFPAL: SUBR(A, B, C, D, E, F, G, H, I, J); -- DEFINE PALLETS
  PALL: STATIC PALLET(A, B, C, D, E);        -- USING FORMAL
  PAL2: STATIC PALLET(F, G, H, I, J);        -- PARAMETERS
        SETPART(PAL1,1);
        SETPART(PAL2,J);
  LOOP: GETPART(PAL1);
        ZMOVE(-250);
        GRASP;
        ZMOVE(0);
        NEXTPART(PAL1);
        GETPART(PAL2);
        ZMOVE(-250);
        RELEASE;
        ZMOVE(0);
        TESTP(PAL2, 1, FINI);                -- END OF PAL2?
        PREVPART(PAL2);
        BRANCH(LOOP);
  FINI:
 END;
 - *** BEGINNING OF EXECUTABLE CODE ***  - -
 DEFPAL(LLPT1,LRPT1,URPT1,3,12,LLPT2,LRPT2,URPT2,3,12);
 END;
```

Region

AML/Entry allows you to define a frame of reference (region) in the manipulator workspace that corresponds to some Porternal coordinate system. Frame of reference allows you to describe motions (in this area) that are relative to the region itself, not to the manipulator coordinate syst2115, Primary use of this feature is to allow appTIZgricTir programs ta be constructed to accept host data that refers to some engineering abstract of the assembly object (such as computer-aided design data), rather than requiring taught point data. Because of this, the program may be replicated across multiple machines without changes in the host data base (all applications use the same set of host data).

REGION is a keyword and is defined by the statement listed below.

```
R: STATIC REGION (LL,UL,LR,UR,LS_LEN,RS_LEN,TOP_LEN,BOTLEN);
```

Listed below are the arguments required to describe the region.

LL LL is a point that is taught in manipulator coordinates to describe the lower left-hand corner of the REGION. The LL corner is alwei7F the origin (0,0,0,0) of the REGION coordinate system.

LR LR is a point that is taught 4212511aulAtor coordinates to describe the lower right-hand corner of the REGION. The X axis of the REURIN ±s tr ine connecting the nc11$ points.

UL UL is a point that is taught in manipulator coordinates to describe the upper left-hand corner of the RE of the REGION is always a line connecting the LL and UL points.

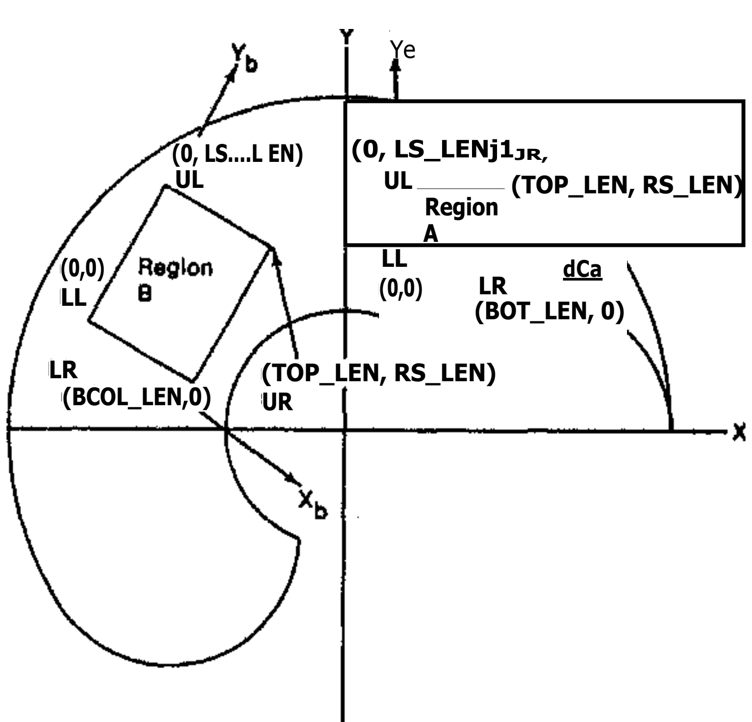
UR UR is a point that is taught in manipulator coordinates to describe the upterright-hand corner of the region.

LS_LEN LS_LEN is the lensal„of the left side of the region as determined by the user. The y coordinate for the upper left caper of the region coordinate system is always LS_LEN. The units for LS_LEN are user-defined, and are not always the same cc=•mate sy

RS_LEN RS_LEN is the length of the right side of the region as determined by the user. The y coordinate for the upper right-hand corner of the region coordinate system is always RS_LEN. The units for RS LEN are user-defined, and are not always th;"same as the manipulat5T E55Ydinate system.

TOP LEN TOP LEN is the length of the top side of the region as determined by the user. The x coordinate for the upper right-hand corner of the region coordinate system is always TOP_LEN. The units for TOP LEN are user-defihed, andiirt_agt always the s as t e mani ulat oor

BOT_LEN BOT_LEN is the length of the bottom side of the region as determined by the user. The x coordinate for the lower right-hand corner of the region coordinate system is always BOT_LEN. The are user-defined, and Are—riet- always the same as the manipulator cooraliat% glitem.



REGION Command

There is one motion command that is used to designate a move within a region. The following motion command is:

Command	Description
XMOVE	Regional movement

XMOVE Command

The REGION definition accepts formal parameters. A REGION need not be rectangular but, in cases where the REGION is rectangular, its use is riaiFirlani57744-rstood. Parallelograms function in a logical manner as an extension of rectangular behavior.

```
XMOVE( region_name,point_name )
```

Here, region_name is the name used in the region definition and point_name is a point in region coordinates.

REGION Coordinate Generation

An explanation of how the REGION coordinates are generated is presented to help you understand the basic concept of a REGION.

X and Y Coordinates

The x and y coordinates are generated in the following manner. The x coordinate is interpolated along a line that connects the LL and the UL points and along a line that connects the LR and UR points. An imaginary line is drawn between these two points. The y coordinate is interpolated along a line that connects the LL and LR points and along the line that connects the UL and UR points. An imaginary line is then drawn connecting these two points. Where this line intersects the line from the x coordinate is the new REGION X,Y coordinate.

Interpolation of X and Y points is done by dividing the XMOVE coordinate by the appropriate LEN parameter. For example, if the X coordinate from an XMOVE is 7 and the LS_LEN is 10, the interpolated point is .7 (7/10) of the way from LL to UL.

Z Coordinate

The REGION does not have to be perpendicular to the Z axis, it can be tilted. The REGION Z coordinate assumes the region is a planar surface that touches the LL, LR, and UL points (the UR Z coordinate is ignored). The Z coordinate is interpolated along this planar surface so that a Z coordinate of 0 in an XMOVE is on the surface of the REGION plane. A Z coordinate of 10 within an XMOVE always moves to a point 10 mm above the

REGION plane. Conversely, a Z coordinate of -10 within an XMOVE always moves to a point 10 mm below the REGION plane.

> **Note:** Nearly vertical regions should be avoided as the mathematical error associated with the region may cause data errors.

Roll Coordinate

The roll coordinate is defined by a line that connects the LL and LR points. This line is defined to be the 0 roll point. **The roll coordinate of the LL, LR, UL, and UR points is ignored.** All roll coordinates within an XMOVE are considered to be positive displacements from the base line. A roll coordinate of 10 in an XMOVE rotates the roll axis to a point 10 counterclockwise from the base line.

Using REGIONS

You are able to define points relative to a REGION. This is especially useful in moving to coordinates that have been generated externally. For example, consider a rectangular REGION somewhere in the workspace. A circuit board is placed in this REGION so that the card is aligned with the X and Y directions of the REGION (not the manipulator). If the card is to be populated, the external system contains the locations at which the components are to be inserted in that coordinate system. If a point is down-loaded from the host, you are able to move the manipulator to the correct location in the workspace by using the XMOVE command.

Frame of reference commands allow a region of space, shown below, to be defined as an independently existing entity.

> **Note:** Because each XMOVE requires 1K of controller memory, an XMOVE should be contained in a subroutine and called whenever required by the program.

An example program using REGION is outlined below.

```
       LLPT: NEW PT(0,350,0,0);
       LRPT: NEW PT(100,350,0,0);     LLPT,LRPT,ULPT,URPT
       ULPT: NEW PT(0,550,0,0,0); -- ARE POINTS TAUGHT
       URPT: NEW PT(100,550,0,0); -- IN MANIPULATOR COORDINATES

 _   ********-_-******   **-        -***-L***    **    ***-_-_**      *****  _
       REG1: STATIC REGION(LLPT,ULPT,LRPT,URPT,4,4,5,5);--4x5 REGION
       CTR1:STATIC COUNTER;
       CTR2:STATIC COUNTER;
       MAIN: SUBR;

       MOVE: SUBR(X,Y);           -- MOVE TO A POINT WITHIN A REGION
       XMPT: NEW PT(X,Y,0,0);
    XMPT_DN: NEW PT(X,Y,-100,0);
             XMOVE(REG1,XMPT);
             XMOVE(REG1,XMPT_DN);
             END;                 -- END MOVE SUBR

  — — *** BEGINNING OF PROGRAM *** — -
             SETC(CTR1,0);         - - INITIALIZE COUNTERS
             SETC(CTR2,0);         - - THAT ARE THE REGION MOVE POINTS
      LOOP1: TESTC(CTR1,6,NEXT1); - - END OF ROW ?
             MOVE(CTR1,CTR2);      - - MOVE TO POINT IN REGION
             INCR(CTR1);
             BRANCH(LOOP1);
      NEXT1: TESTC(CTR2,4,NEXT2); - - LAST COLUMN ?
             SETC(CTR1,0);         - - RESET ROW COUNTER
             INCR(CTR2);           - - INCREMENT COLUMN COUNTER
             BRANCH(LOOP1);
      NEXT2:
      END;
  — — *****************************************************
```

The main logic of the program consists of two loops, one nested within the other. Another approach would be to use expressions. Instead of declaring CTR1 and CTR2 as counters, three counters CTR, ROW and COL would be required. The main loop of the progr'am could then be replaced with:

```
  = = *** BEGINNING OF PROGRAM *** ==
             SETC(CTR,O);              -- INITIALIZE COUNTERS
      LOOP: SETC(ROW,TRUNC(CTR/6));    -- FETCH ROW OF REGION
             SETC(COL,CTR-6*ROW);      -- FETCH COL OF REGION
             MOVE(ROW,COL);            -- MOVE TO POINT IN REGION
             INCR(CTR);
             COMPC(CTR LT 30,LOOP);    -- LOOP UNTIL DONE
      END;
```

Host Communications

AML/Entry supports controllers equipped with enhanced communications. A few concepts added to AML/Entry allow host communications and data reporting to become more effective.

Host data drive is a concept where the robot controller is programmed to perform a generic task. The details of that task are determined by a host computer. The information is transferred from the host to the robot controller. For example, the controller is programmed to perform a series of part insertions on a card. The host supplies the coordinates of the insertion locations on a per card basis. In this manner, one robot program is used to produce a large number of different assemblies.

By the use of controller-initiated communications, you are able to control program execution. Controller-initiated communication is used to change counters or points, or groups of data. By changing counter variables, it is possible to change points, movement control statements, sensor commands, and any other structure that uses integer or real variables.

Command	Description
GET	Controller Requires Data
PUT	Controller Wants to Send Data

GET Command

This command indicates to the host that the controller re uires specific data. The controller initiates a data drive operation, and the afirtication program waits thirty ds for a response from the host computer. If there ' response, the controller stops execution and the TE light on the control panel lights up (indicating a Transmission trror).

GET is used to get a single counter, a single point, a single counter within a group, and a single figTrir Within a grou7insk well asgragt—tf counters and" points For further information on how the host communicates with the controller, refer to Chapter 8, "Communications." Format of the command is outlined below.

```
GET(counter_name);          -- requests the host to submit value
                            -- for counter name
GET(group_name);            -- gets a value from the host
                            -- for each element in the group
```

Note: If the controller is off-line, the communications cable not attached, or the DSR signal inactive, then a DE will occur when a GET or PUT instruction is performed. **The** error code will be lE - Communications not established (unable to GET/PUT). If the program is started by using the operator panel, the controller has to be in the off-line state. Thus if one of the first

instructions is a GET or PUT, then a DE will occur. Either use the CSTATUS command to ensure communications are established, or start the application remotely using Comaid (see Chapter 8, "Communications").

An example of a GET command is outlined below in a program fragment.

```
PT1:NEW PT(650,0,0,0);
PT2:NEW PT(0,650,0,0);
C: STATIC COUNTER;
P: STATIC GROUP(PT1,PT2);
   GET(C);               -- get value for counter
   GET(P);               -- get values for two points
```

Note: The GET for the group P will only change the points of the group. The values for PT1 and PT2 will remain unchanged.

PUT Command

This command indicates to the host that the controller wants to send specific data.

The controller initiates a data report operation and the application program waits for the host to respond. If no response is received after three secondp, the controller re-transmits the request. After two re-tries (nine seconds), the controller stops program execution and the TE light comes on.

PUT is used to send a single counter, a4ingle_point, a single counter within a group, and a single point within a group as well as groups of coup errand points. Format of the command is outlined below.

```
PUT(name);               -- requests the host to accept value
                         -- for object 'name'
```

An example of a PUT command is outlined below in a program fragment.

```
PT1:NEW PT(650,0,0,0);
PT2:NEW PT(0,650,0,0);
C: STATIC COUNTER;
P: STATIC GROUP(PT1, PT2);
   PUT(C);             -- SEND VALUE FOR COUNTER
   PUT(P);             -- SEND VALUES FOR TWO POINTS
```

Note: If the controller is off-line, the communications cable not attached, or the DSR signal inactive, then a DE will occur when a GET or PUT instruction is performed. The error code will be 1E - Communications not established (unable to GET/PUT). If the program is started by using the operator panel, the controller has to be in the off-line state. Thus if one of the first instructions is a GET or PUT, then a DE will occur. Either use the CSTATUS command to ensure communications are established, or start the application remotely using Comaid (see Chapter 8, "Communications").

An example of a GET and PUT command is outlined below in a program fragment.

```
PUT(INDEX);          -- SEND NEW VALUE OF INDEX TO HOST
GET(LLPT);           -- REQUEST NEW VALUE OF LLPT FROM HOST
PMOVE(LLPT);         -- MOVE TO NEW POINT
GET(P(3));           -- CHANGE THE VALUE OF P(3)
PMOVE(P(3));         -- MOVE TO NEW POINT
```

Variable Identification

Information regarding variable(s) is passed to the host during GET or PUT communication sequences. This information is the variable number specified by the AML/Entry compiler, followed by the count of the number of consecutive variables being passed. AML/Entry refers to each variable by a unique number assigned by the compiler.

XREF Program

The XREF program supplies additional information that is useful when the host is using communications. See "XREF Program" on page 2-34 for a discussion of how to begin the XREF program.

A program example is shown below, followed by an example of the output of the XREF program.

```
*******************************************************
 1 POINT1: NEW PT(0,500,0,0);          --DECLARATION OF POINTS
 2 POINT2: NEW PT(0,600,0,0);
 3 POINT3: NEW PT(50,500,0,0);
 4 POINT4: NEW PT(0,500,-50,180);
 5 ROWS: STATIC GROUP(POINT1,POINT2,POINT3,POINT4); --GROUP OF
 6 ROW: STATIC COUNTER;                               --POINTS
 7 MAIN: SUBR;
 8    SUB1: SUBR(X);
 9       DECR(X);                       --DECREMENT COUNTER
10    END;
11 SETC(ROW,1);                         --SET COUNTER TO 1
12 SUB1(ROW);
13 END;
*********************************************************4
```

When the XREF program is run against the previous program, the following output will be displayed on the screen.

TYPE	NAME	VAR #	SIZE	FTYPE/GSIZE
POINT	P1	34	4	
POINT	P2	38	4	
POINT	P3	42	4	
POINT	P4	46	4	
GROUP-PT	ROWS	50	16	4
COUNTER	ROW	66	1	
SUBR	MAIN	84	0	
SUBR	SUB1	88	1	
FORMAL	X	67	1	COUNTER
END	SUB1	99	0	
END	MAIN	115	0	

This output provides you with the information necessary to use host communications. The NAME field provides you with the name of the variable from the program. VAR # is the address of the variable in controller memory and SIZE is the number of variables associated with the name.

If TYPE is FORMAL, FTYPE/GSIZE is the data type assigned to formal parameters. If the TYPE is GROUP, FTYPE/GSIZE is the number of bytes associated with a GROUP.

In the below example, the controller sends 34 as the variable number and sends 4 as the number of variables r equested during the GET transaction.

```
GET (P1);
```

Pallet and Region Listings

Using GET and PUT in an AML/Entry program allows controller initiated data drive of counters, points, and groups. However the host (IBM PC/XT/AT, 5531 Industrial Computer) can also initiate a data-drive transfer. This allows any variable in controller memory to be changed, including regions and pallets. (See Chapter 8, "Communications" for how to perform host initiated data drive). Regions use 20 controller variables, pallets use 15. The listing produced by XREF indicates each of the components that comprise regions and pallets. The following is a sample listing for a program that contains one pallet and one region.

TYPE	NAME	VAR #	SIZE	FTYPE/GSIZE
PALLET	PAL	34	15	
-PT	LL	34	4	
-PT	LR	38	4	
-PT	UR	42	4	
-CTR	Parts/Row	46	1	
-CTR	# Parts	47	1	
-CTR	Cur Part-1	48	1	
REGION	REG	49	20	
-PT	LL	49	4	

```
     -PT     UL                          53          4
     -PT     LR                          57          4
     -PT     UR                          61          4
     -CTR    LS  Len                     65          1
     -CTR    RS  Len                     66          1
     -CTR    Top Len                     67          1
     -CTR    Bot Len                     68          1
SUBR         MAIN                       123          0
END          MAIN                       124          0
```

The TYPE and NAME fields together indicate the components of regions and pallets. In the above example, The lower right point of the pallet begins at variable number 38 and has a length of 4. The bottom length of the region begins at variable number 68 and has a length of 1. Thus to change a pallet or region using COMAID (host initiated data drive), one would use the enhanced listing to locate the particular component that required changing.

Note that the last component of a pallet is the current part MINUS 1. Thus if COMAID is used to read variable number 48 and it has a value of 5, the current part of the pallet is actually 6 (because 5 is the current part number minus 1). Likewise, if one wanted to set the current part number to 10, he would data drive 9 to variable number 48.

4-92 58X7338

CHAPTER 5. WRITING AML/ENTRY PROGRAMS

In this chapter you will be given two AML/Entry programs that perform the same application. The application is relatively simple, and is intended that way so you may concentrate on the programming style illustrated. This simple program does an easy pick and place operation with no external inputs. A complex program using the full power of AML/Entry Version 4 to assemble small batches of many different cards using data downloaded from a host computer is shown following this simpler example. Your application will probably fall somewhere in between these example applications. Concentrate on the program structure instead of the application itself.

GOOD PROGRAM STRUCTURE

These paragraphs provide information and recommendations on programming techniques that help you create programs that can be easily written and updated.

- Use Declarations Where Possible

  Use declarations where possible for the points and constants of the program. This allows you to refer to the point by a name through out the program making the program easier to read. When the point is taught you only have to put the location data at one place in the program. In addition, a change to a taught point or a constant requires that you make the change only once at the declaration statement.

- Use Subroutines Where Tasks are Repeated

  Use subroutines where tasks are repeated many times in your program. The subroutines may consist of only three or four executable statements, but typing in the subroutine name is easier and faster than typing each line over again. In addition, storage space is saved at the controller when using subroutines.

- Use Indentations

  Use indentations to allow the program to be read easier.

- Use Names With a Meaning

  When using names, attempt to use names that have a meaning you can recall easily. Use the underscore (_) character to separate words in the name if necessary. Meaningful names usually are longer and require a little more time to input, but you will be glad that you spent the extra time when the program needs to be modified.

- Use Blank Lines in the Program

  Leave blank lines in the program as viewed in the editor. The blank space allows easier reading of the program.

- Use Comments Throughout the Program

  Use many comments throughout the program. A program that has been put aside for a period of time may not even look familiar to the developer when comments are not included.

- Use Expressions When Possible

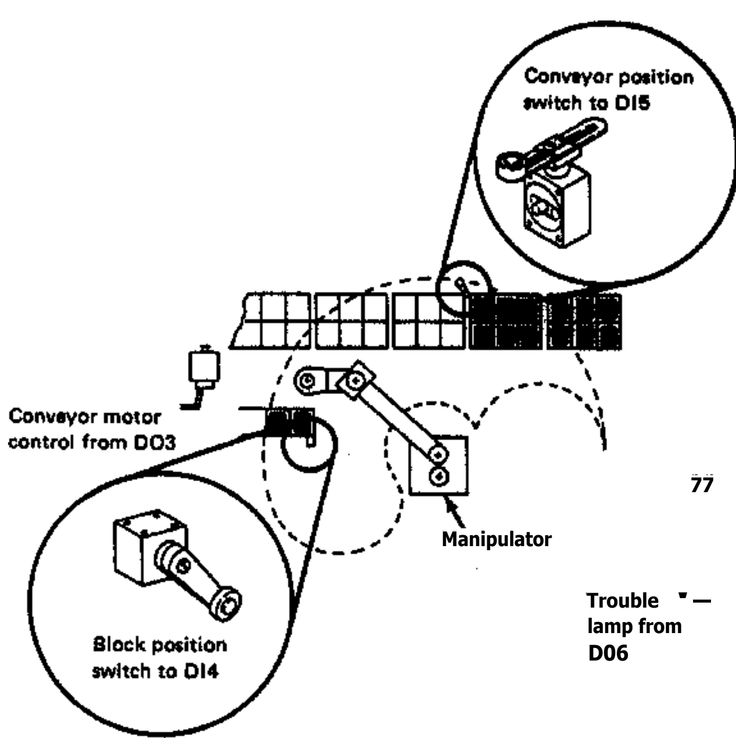
  In many cases, expressions can be used to simplify a program's structure. A good example of this is shown in "Treating DI As Integers" on page 4-56. By using an expression to hold the result of several digital inputs, it is not necessary to perform many TESTI commands. In general, it is better to use the arithmetic function forms of CSTATUS, MSTATUS, TESTI, and TESTP instead of the command forms.

WRITING A SIMPLE AML/ENTRY PROGRAM

The most important step in writing any program is understanding the application. Try to understand the application as well as possible before starting to program the application. Describing the application using words and pictures can be very helpful.

Sample Application

The application appears as:



In the sample application, the manufacturing system picks up one block at a time from a gravity chute and moves the block to a slot in a carton. The carton is on a conveyor that moves the filled carton out and a new one in. The chute has a switch that closes when a block is ready for pickup by the gripper attached to the Z-axis of the arm. The switch at the chute is wired to digital input point 4 (DI4) of the controller.

The conveyor moves when the controller signals it through the relay at digital output point 3 (D03). The controller tests for a switch closure to determine if the conveyor has moved a carton into position. This switch is connected to digital input point 5 (DI5).

If the conveyor does not move a carton to the proper location for loading, the operator is signaled by a lamp controlled by digital output point 6 (D06). The relay at DO point 6 closes causing the lamp to light. The lamp is not part of the manufacturing system.

Application Program Comparison

This section shows two different methods of programming the carton application. The first method uses no subroutines or palletizing; the second program uses both. The difference in the programs is obvious, the second program has a neater appearance and is easier to understand.

```
******* FIRST EXAMPLE ******                                      Page 1
 1 NEWPROG: SUBR;
 2          WRITEO(3,1);
 3          DELAY(3.0);
 4          TESTI(5,0,STOP);
 5          WRITEO(3,0);
 6          BRANCH(GO);
 7    STOP: WRITEO(6,1);
 8          WAITI(5,1,10);
 9          WRITEO(3,0);
10      GO: PMOVE(PT(-500,150,-150,0));      --PICKUP BLOCKS
11          WAITI(4,1,0);                    --WAIT FOR BLOCKS
12          ZMOVE(-250);
13          GRASP;
14          DELAY(1.0);
15          ZMOVE(0);
16          PMOVE(PT(-150,550,0,90));        --SLOT 1 OF CARTON
17          ZMOVE(-100);
18          RELEASE;
19          DELAY(1.0);
20          ZMOVE(0);
21
22          PMOVE(PT(-500,150,-150,0));      --PICKUP BLOCKS
23          WAITI(4,1,0);                    --WAIT FOR BLOCKS
24          ZMOVE(-250);
25          GRASP;
26          DELAY(1.0);
27          ZMOVE(0);
28          PMOVE(PT(-50,550,0,90));         -- SLOT 2 OF CARTON
```

```
29          ZMOVE(-100);
30          RELEASE;
31          DELAY(1.0);
32          ZMOVE(0);
33          PMOVE(PT(-500,150,-150,0));      --PICKUP BLOCKS
34          WAITI(4,1,0);                    --WAIT FOR BLOCKS
35          ZMOVE(-250);
36          GRASP;
37          DELAY(1.0);
38          ZMOVE(0);
39
40          PMOVE(PT(50,550,0,90));          --SLOT 3 OF CARTON
41          ZMOVE(-100);
42          RELEASE;
43          DELAY(1.0);
44          ZMOVE(0);
45          PMOVE(PT(-500,150,-150,0));      --PICKUP BLOCKS
46          WAITI(4,1,0);                    --WAIT FOR BLOCKS
47          ZMOVE(-250);
48          GRASP;
49          DELAY(1.0);
50          ZMOVE(0);
51
52          PMOVE(PT(-150,450,0,90));        --SLOT 4 OF CARTON
53          ZMOVE(-100);
54          RELEASE;
55          DELAY(1.0);
56          ZMOVE(0);
57          PMOVE(PT(-500,150,-150,0));      --PICKUP BLOCKS
58          WAITI(4,1,0);                    --WAIT FOR BLOCKS
59          ZMOVE(-250);
60          GRASP;
61          DELAY(1.0);
62          ZMOVE(0);
63
64          PMOVE(PT(-50,450,0,90));         --SLOT 5 OF CARTON
65          ZMOVE(-250);
66          RELEASE;
67          DELAY(1.0);
68          ZMOVE(0);
69          PMOVE(PT(-500,150,-150,0);       --PICKUP BLOCKS
70          WAITI(4,1,0);                    --WAIT FOR BLOCKS
71          ZMOVE(-250);
72          GRASP;
73          DELAY(1.0);
74          ZMOVE(0);
75
76          PMOVE(PT(50,450,0,90));          --SLOT 6 OF CARTON
77          ZMOVE(-100);
78          RELEASE;
79          DELAY(1.0);
80          ZMOVE(0);
81 END;
```

```
****** SECOND EXAMPLE ******                            Page 1
 1       LL:NEW PT(-150,550,0,90);        --PALLET POINTS
 2       LR:NEW PT(50,550,0,90);
 3       UR:NEW PT(50,450,0,90);
 4      PPR:NEW 3;
 5        N:NEW 6;
 7 CARTON:STATIC PALLET(LL,LR,UR,PPR,N);
 8   PICKUP:NEW PT(-500,150,-150,0);
 9
10 -- ***** DI/DO DECLARATIONS ***********************
11
12 CONVEYOR_ON:NEW 3              -- CONVEYOR MOTOR CONTROL
13    BLOCK_HERE:NEW 4            -- BLOCK PRESENT SENSOR
14 CARTON_HERE:NEW 5              -- CARTON PRESENT SENSOR
15 TROUBLE_LAMP:NEW 6             -- OPERATOR TROUBLE INDICATOR
16
17 -- ***** END OF DECLARATIONS BEGIN SUBROUTINES *****
18
19 NEWPROG:SUBR;
20     LOAD:SUBR;
21            PMOVE(PICKUP);                  --PICKUP BLOCKS
22            WAITI(BLOCK_HERE,1,0);          --WAIT FOR BLOCKS
23            ZMOVE(-250);
24            GRASP;
25            DELAY(1.0);
26            ZMOVE(0);
27            END;
28 DROPOFF:SUBR;                              --MOVEMENTS TO LOAD PALLET
29            ZMOVE(-100);
30            RELEASE;
31            DELAY(1.0);
32            ZMOVE(0);
33 END;
34            SETPART(CARTON,1);              --EXECUTION STARTS HERE
35     NEXT:WRITEO(CONVEYOR_ON,1);            --TEST FOR BLOCK
36            DELAY(3.0);
37            TESTI(5,0,STOP);
38            WRITEO(CONVEYOR_ON,0);
39            BRANCH(GO);
40     STOP:WRITEO(TROUBLE LAMP,1);
41            WAITI(CARTON HER ,1,10);
42            WRITEO(CONVEOR_ON,0);
43       GO:
44            LOAD;                           --LOAD BLOCK
45            GETPART(CARTON);
46            NEXTPART(CARTON);               --MOVE TO CARTON
47            DROPOFF;                        --DROP OFF BLOCK
48            BREAKPOINT;
49            COMPC(TESTP(CARTON) NE 1,G0);   --TEST FOR FULL CARTON
50            BRANCH(NEXT);
51 END;
```

WRITING A COMPLEX AML/ENTRY PROGRAM

An example of how a complex application program is designed and writte using AML/Entry is provided in this section. It demonstrates th ability of this language in the area of data drive, error recovery, an logical organization of task activity. The first step in writing an application program is understanding the application. The more comple the application the more important it is to spend time understanding th application.

Main Application Task

The main task of this program is to insert components into circui boards. Many different circuit boards are required to be assembled i small batches at varying times during the day. Because of this, th data that describes the position of the components on each of thes circuit boards is to be stored on the host computer and sent to th controller before each batch is started.

Printed Circuit Card

Components are inserted into a printed circuit card. The card is place in the production environment by other equipment (probably carried int place by a conveyor to come to rest at a precision stop).

The card is defined as a REGION. This allows the points on the card t be described with respect to the card itself, not in manipulato coordinates. Component insertion locations are obtained directly from computer-assisted design data base contained in a host computer.

Manipulator Gripper

For this application, the manipulator is fitted with a special grippe with two active modes, opening and closing.

There are two sensors to determine whether the gripper is fully opene or fully closed. Gripper can be relaxed by having neither state activ (it is not opening or closing). It also uses a parts presence senso (LED-photodetector pair), and a force sensor that senses for over-pressure condition in the Z direction. If a part does not inser properly, the force sensor is used to prevent the card from bei destroyed.

Component Feeders

The program manages four different types of components, supplied throug four individual feeders.

Each component is supplied from a feeder that has one digital input (DI) and one digital output (DO) point associated with it.

The digital output (DO) point drives an actuator on the feeder that releases one component at a time from the supply bin. The component comes to rest at the feeder part stop location (a taught point). The feeder parts stop positions the component where the manipulator can pick it up.

The DI point is used to sense if a component is present at the feeder's part stop location. If the DO point has been pulsed to release a component and the DI point fails to sense it within the correct time either the feeder is out of parts or it has jammed. When this condition occurs the operator is notified.

Interaction with a Host Computer

Controller interacts with the host computer to control the assembly process, as outlined below.

1. Digital Input/Digital Output (DI/DO)

   - When the points are loaded into the controller, the host signals the controller using a DI point.

   - The controller uses DO points to inform the host of the application's status.

2. Host-Initiated Communication

   - The host loads the component insertion location points into the controller before the application is started.

3. Controller-Initiated communications

   - The application program queries the host for part type and insertion location.

Application Flow

The program uses host communications to control an application program.

1. The host computer first uses host-initiated communications to send the controller the points that define the different printed circuit card locations to be used during the population of the current printed circuit card.

2. These points are obtained from the computer-assisted design data base and are defined with respect to the printed circuit card, not the manipulator work space.

3. When the host completes sending the points to the controller, it uses DI/DO to inform the controller to start the insertion process.

4. After the application program starts, the AML/Entry program queries the host for part (component) type and component insertion location.

5. The part type is a number from 1 to 4 representing the feeder that holds the component.

6. The insertion location is a number from 1 to 10 that specifies which insertion location points to use.

7. The program interacts with an operator to signal various error conditions.

8. The operator may also suspend all motion. This allows the operator to load feeders or clear jams that do not require entry to the work space. In practice, an operator interface panel usually has several lights to display status (from the DO lines) as well as switches for input through DI points.

Define Global Data Types

Now that the application has been described you can start to design your application program. The first step in an application program is defining the global data types.

Taught Points

It is a good idea to put any global taught points at the beginning of the program. This makes it easier to find the points when they are taught. For consistency's sake the other data needed to define the cards and feeders are placed in this area also. The points are initially input with a value of zero because they have not been taught yet.

```
-- Taught Points to Define Location of Card

CARD_UL:   NEW PT(0,0,0,0);        --card's upper-left corner
CARD_LL:   NEW PT(0,0,0,0);        --card's lower-left corner
CARD_UR:   NEW PT(0,0,0,0);        --card's upper right corner
CARD_LR:   NEW PT(0,0,0,0);        --card's lower-right corner

-- Other Data to Define Card

CARD_LS:   NEW 0;                  --card's left side length
CARD_RS:   NEW 0;                  --card's right side length
CARD_TOP: NEW 0;                   --card's top length
CARD_BOT: NEW 0;                   --card's bottom length

-- Taught Points for Fixture Locations

FEEDER1:   NEW PT(0,0,0,0);        -- Z MUST be zero here
FEEDER1Z: NEW 0.0;                 -- This is Z coordinate.
FEEDER2:   NEW PT(0,0,0,0);        -- Z MUST be zero here
FEEDER2Z: NEW 0.0;                 -- This is Z coordinate.
FEEDER3:   NEW PT(0,0,0,0);        -- Z MUST be zero here
FEEDER3Z: NEW 0.0;                 -- This is Z coordinate.
FEEDER4:   NEW PT(0,0,0,0);        -- Z MUST be zero here
FEEDER4Z: NEW 0.0;                 -- This is Z coordinate.
```

Digital Input and Digital Output

DI/DO points do not have to be included in the definitions as constants. The actual numbers can be used in the AML/Entry commands. There are several reasons that make it a good practice to give each DI/DO point a name.

1. Giving each point a name that describes it's function makes the program more readable.

2. The physical number of each point is defined by the hardware and how it is attached to the controller. (If a signal that was supposed to be attached to DI number 3 is actually attached to DI number 4, it may be easier to change one line in the program than to change the hardware.

```
HOST_START:     NEW 03; --Host controlled point.
                         --if 0: inactive.
                         --if 1: start to populate a card.
OPER RETRY:     NEW 04; --Signal from Operator: Retry.
                         --if 0: inactive.
                         --if 1: active.
OPER OK:        NEW 05; --Signal from Operator: OK to move.
                         --if 0: OK.
                         --if 1: Do Not Allow Motion.
OPER ABORT:     NEW 06; --Signal from Operator: Abort the process.
                         --if 0: OK to proceed.
                         --if 1: abort the process.
GRIP OPND:      NEW 07; --Gripper opened sensor
                         --if 0: not fully opened.
                         --if 1: fully opened.
GRIP CLSD:      NEW 08; --Gripper closed sensor
                         --if 0: not fully closed.
                         --if 1: fully closed.
GRIP OBJS:      NEW 09; --Gripper object sensor
                         --if 0: no object detected.
                         --if 1: object detected.
GRIP PRSS:      NEW 10; --Gripper pressure sensor
                         --if 0: no undue pressure detected.
                         --if 1: too much force being exerted.

--parts presence sensor for the feeders

FEEDER1S:       NEW 11; --Fixture 1 sensor.
FEEDER2S:       NEW 12; --Fixture 2 sensor.
FEEDER3S:       NEW 13; --Fixture 3 sensor.
FEEDER4S:       NEW 14; --Fixture 4 sensor.
```

```
HOST_AVAIL:     NEW 03; --Signal to Host: manipulator is available.
                          --if 0: not available.
                          --if 1: available.
HOST_STRTD:     NEW 04; --Signal to Host: process started.
                          --if 0: process is not active.
                          --if 1: card population in progress.
HOST CMPLT:     NEW 05; --Signal to Host: process completed.
                          --if 0: process is not active or incomplete.
                          --if 1: card population completed.
GRIP OPEN:      NEW 06; --Gripper open command
                          --if 0: no effect.
                          --if 1: open.
GRIP CLOSE:     NEW 07; --Gripper close command
                          --if 0: no effect.
                          --if 1: close.
OPER ATTN:      NEW 08; --Signal to Operator: Attention Required.
                          --if 0: no problem.
                          --if 1: problem encountered.
PBLM COMM:      NEW 09; --Signal to Operator: Problem is Comm.
                          --if 0: no problem.
                          --if 1: problem encountered.
PBLM FXTR:      NEW 10; --Signal to Operator: Problem is Fixture.
                          --if 0: no problem.
                          --if 1: problem encountered.
PBLM GRPR:      NEW 11; --Signal to Operator: Problem is Gripper.
                          --if 0: no problem.
                          --if 1: problem encountered.

--parts release activator for the feeders
FEEDER1A:       NEW 12; --Fixture 1 activator.
FEEDER2A:       NEW 13; --Fixture 2 activator.
FEEDER3A:       NEW 14; --Fixture 3 activator.
FEEDER4A:       NEW 15; --Fixture 4 activator.
```

Constants

You may want to give names to numbers that are used many times in your program. If you give these special numbers a descriptive name it will increase the readability of your program. In this program the name ON is given the value 1 and OFF the value 0. This allows you to use the names ON and OFF when using the DI/DO points instead of 1 and 0. The program also assigns a generic point to the name P, allowing a GROUP of points "P" to be defined further on in the program.

```
ON:      NEW 1;
OFF:     NEW 0;
P:       NEW PT(0,0,0,0);       --generic point
```

Variables

The last global declarations in this program are the variables that are used throughout the program. Any variables that are used within specific subroutines are declared as local variables within the individual subroutines.

```
INIT FLAG:      STATIC GROUP
                (OFF);          -- A counter initialized to OFF.
                                -- Will be set to ON after program has
                                -- been initialized.

FEEDERSL:       STATIC GROUP --locations of feeders
                (FEEDER1, FEEDER2, FEEDER3, FEEDER4);
FEEDERSZ:       STATIC GROUP --Z height of feeders
                (FEEDER1Z, FEEDER2Z, FEEDER3Z, FEEDER4Z);
FEEDERSA:       STATIC GROUP --actuator DO point for feeders
                (FEEDER1A, FEEDERSA, FEEDER3A, FEEDER4A);
FEEDERSS:       STATIC GROUP --sensor DI point for feeders
                (FEEDER1S, FEEDER2S, FEEDER3S, FEEDER4S);

CARD:           STATIC REGION(CARD_LL,CARD_UL,CARD_LR, CARD_UR,
                              CARD_LS,CARD_RS,CARD_BOT,CARD_TOP);

LOCATIONS:      STATIC GROUP --component insert locations
                (P,P,P,P,P,P,P,P,P,P);
                                --location points are relative to card

LOC TO CARD:    STATIC COUNTER;--will contain host specified data-
                                --z distance from "location" to "card"

INSTRS:         STATIC GROUP    --insertion instructions
                (0,0);          --part-type , insertion location
```

Define the Global Subroutines

After you have defined all the global data for the program you need to define the global subroutines to used in the program. Careful use of global subroutines can greatly enhance the program.

Analyse the application to see if any actions are repeated several times. These actions should be put into subroutines. When you want to do the action you only have to call the subroutine. Using subroutines also breaks the program into small easy to understand blocks. The example program makes extensive use of subroutines, each of the subroutines is described below.

Utility Subroutines

There are four subroutines used by the program that can be called utility subroutines. These subroutines do simple tasks that are needed by many of the other subroutines in the program. These subroutines are:

- WAIT_TOGGLE (PORT)

  This subroutine waits for the specified DI port to be toggled from OFF to ON to OFF. It can be used any time it is necessary to wait for an input such as the operator pressing a button.

```
--*************************************************************--
                    Subroutine: WAIT TOGGLE( PORT );

      1) Wait for the selected port to be toggled- that is,
         it must be off, then pulse on, then return to off.

         This routine will wait forever.                          --
--*************************************************************--

WAIT_TOGGLE  : SUBR(PORT);
     WAITI( PORT,  OFF,  0);  --insure OFF
     WAITI( PORT,  ON,  0);   --wait for ON
     WAITI( PORT,  OFF,  0); --and OFF again
     END;
```

• PULSE (PORT,TIME)

This subroutine pulses the specified DO port ON then OFF. The time period for each step of the toggle is also specified. This subroutine can be used any time it is necessary to send a signal to an external piece of equipment.

```
--***********************************************************--
                Subroutine: PULSE(PORT,TIME);

--   1) Pulse the selected port from OFF to ON to OFF.
        The pulse will have a duty cycle specified by TIME.      --
--***********************************************************--

PULSE:SUBR(PORT,Y);
     WRITEO(PORT, ON);
     DELAY(Y);
     WRITEO(PORT, OFF);
     END;
```

• PROBLEM (PORT)

This subroutine notifies an operator of a problem within the system. An error indicator is lighted along with the specified DO point to indicate the nature of the error. Notice that this subroutine uses the subroutine WAIT TOGGLE to wait for the operators response to the problem. When the operator has fixed the problem the error indicators are turned off and control is returned to the caller.

```
--************************************************************--
                  Subroutine: PROBLEM( PORT );

-- 1)  Signal the operator that a problem has been encountered. --
-- 2)  Wait for operator to signal OK to retry.           ------
-- 3) Reset problem indicators.                                 --
--*********************************4**************************--

PROBLEM:SUBR(X);
     WRITEO( OPER_ATTN, ON);   --ask for operator's attention
     WRITEO( X, ON);           --indicate where problem is
     WAIT_TOGGLE(OPER_RETRY) ; --wait for retry signal
     WRITEO( OPER_ATTN, OFF) ; --turn off signals
     WRITEO( X, OFF);
     END;
```

- DROP PART (X)

  This subroutine is used to drop a part from the feeder into the fixture where the manipulator can pick it up. This subroutine also checks for feeder jams. If a jam is detected, it is reported to the operator using the PROBLEM subroutine.

```
--*****************************************************--
             Subroutine: DROP PART(X)

       Drop a part at feeder X into the jig.

       1) Activate the feeder DO point.
       2) Wait for part presence sensor to be active,
          if not active in 1.5 second, then it must
--        be jammed, so signal operator.              --
--****************************4**********************--

DROP PART:SUBR(X);
  TRY:
     PULSE(FEEDERSA(X),.2);     --pulse the part release port
     WAITI(FEEDERSS(X), ON, 1.5, ERR);
     BRANCH(OK);                --allow 1.5 sec. for part
  ERR:
     PROBLEM(PBLM_FXTR);        --report problem
     BRANCH(TRY);               --retry
  OK:
     END;
```

Movement Subroutines

The example program uses four movement subroutines that are designed t be used in place of the normal AML/Entry movement commands. Thes subroutines provide a method of suspending all movement of th manipulator arm based on the status of the OPER_OK digital input point This allows the operator to stop the manipulator movement by pressing button. Notice that the subroutines wait for an ON (1) condition befor allowing motion to continue. This makes the sensing "failsoft", meanin that if a wire breaks motion stops. If the subroutines waited for a OFF (0) a broken wire would allow motion to continue.

```
--*****************************************************************-
                         Subroutines:
   PPMOVE(POINT), ZZMOVE(POINT), XXMOVE(POINT), DDZMOVE(DELTA)

     The following four subroutines are used instead of the
     standard move commands. They allow you to monitor a DI
     point to control movement. If OPER_OK is not ON the no
     movement is started, the subroutines wait forever.
--*****************************************************************-

   PPMOVE:SUBR(X);
        WAITI(OPER_OK, ON, 0);    --OK to move?
        PMOVE(X);
        END;


   ZZMOVE:SUBR(X);
        WAITI(OPEROK, ON, 0);     --OK to move?
        ZMOVE(X);
        END;


   XXMOVE:SUBR(R,X);
        WAITI(OPER_OK, ON, 0);    --OK to move?
        XMOVE(R,X);
        END;


   DDZMOVE:SUBR(X);
        WAITI(OPER_OK, ON, 0);    --OK to move?
        DPMOVE(<0,0,0,X>);
        END;
```

Gripper Subroutines

There are two subroutines to control the gripper. This allows you to open and close the gripper using AML/Entry like statements. There is a subroutine named OPENGR to open the gripper and one named CLOSEGR TO close the gripper. Both gripper subroutines ensure that the gripper is in the proper condition, not closing when an open command is given etc., before opening or closing the gripper. This prevents mechanical jams of the gripper. OPENGR checks for a condition that prevents the gripper from opening and uses the PROBLEM subroutine to report a jam if one occurs.

```
--**********A*****************************************************--
--                          Subroutine: OPENGR

            Open the gripper

            1) Activate gripper open DO point.
            2) Wait for open sensor to be active,
               if not active in 1 second, then it must
--             be jammed, so signal operator.                      --
--****************************************************************--

   OPENGR:SUBR;
     TRY:
        WRITEO(GRIP_CLOSE, OFF);       --insure not trying to close
        WRITEO(GRIP_OPEN, ON);         --try to open
        WAITI(GRIPOPND, ON, 1, ERR); --give 1 sec. to open
        BRANCH(OK);
     ERR:
        PROBLEM(PBLM_GRPR);            --report problem
        BRANCH(TRY);                   --retry
     OK:
        END;

--****************************************************************--
                          Subroutine: CLOSEGR

            Close the gripper

--          1) Activate gripper close DO point.                    --
--****************************************************************--

    CLOSEGR:SUBR;
         WRITEO(GRIP_OPEN, OFF);    --insure not trying to open
         DELAY(.1);
         WRITEO(GRIP_CLOSE, ON);    --try to close
         END;
```

Parts Handling Subroutines

There are two parts handling subroutines used in the example. One subroutine GET_PART(PART_TYPE) is used to pick up parts from the specified feeder. The other subroutine PUT_PART(PART_PLACE) is used to place the parts in the circuit board at the specified location. Both of these subroutines make extensive use of the subroutines described earlier. Making use of the subroutines allows the parts handling subroutines to do complex actions without being complex themselves. Using subroutines as building blocks allows you to take a step by step approach to the application.

- GET_PART(PART_TYPE)

  This subroutine is used to pick up the part from the part specified in PART_TYPE. PART_TYPE contains the number of the part to pick up and is used as an index into the variables FEEDERSL and FEEDERSZ. These variables have been previously defined to be groups which allow PART_TYPE to be used as an index into the groups. If PART_TYPE has the value 3 then the third element of each of the variables is used. Notice that almost every statement in the GET_PART routine is the name of a previously defined subroutine.

```
--*************************************************--
          Subroutine: GET PART(PART TYPE)

       Get the part specified by feeder number

       1) Insure Z-stroke is fully up.
       2) Insure gripper is open.
       3) Move to the proper location.
       4) Release a component into Fixture's jig.
       5) Move to proper Z height.
       6) Grasp the part.
       7) Move to Z fully up.
       8) Make sure part was grasped.
--*************************************************--

GET_PART:SUBR(X);
   TRY:
     ZZMOVE(0);                    --insure Z up
     OPENGR;                       --open gripper
     PPMOVE(FEEDERSL(X));          --move over the feeder
     DROP PART(X);                 --activate feeder
                                   --(release a part)
     ZZMOVE(FEEDERSZ(X));          --go to proper height
     CLOSEGR;                      --close gripper
     ZZMOVE(0);                    --move z up to safe height
     TESTI( GRPR OBJS,  ON, OK); --ok if there
     WRITEO(GRPR_OPEN,  OFF);      --relax the gripper
     WRITEO(GRPR_CLOSE, OFF);
     PROBLEM(PBLM_GRPR);           --inform operator
     BRANCH(TRY);                  --try again
   OK:
     END;
```

- PUTPART(PART_PLACE)

  The PUT PART subroutine uses the same structure as the GET PART subroutine. That is, PART PLACE contains an index into variables defined as groups. PUT PART defines a local variable named C to be used only within this subroutine.

  PUT_PART moves to the insertion location using the XXMOVE subroutine. Because XXMOVE uses region coordinates the move is with respect to the card. After the manipulator moves to the insert location, a guarded move is made to insert the parts into the card. If the gripper pressure sensor transfers during the insertion, a problem is reported to the operator. After the operator clears the jam the insertion process is retried.

```
--****************************************************--
          Subroutine: PUT PART(PART PLACE)

       Insert the part specified by location number

       1) Move to the proper location.
       2) Insert the part.
          - turn gripper force sensor guarding on
          - move down until tripped or inserted
       3) Move to Z fully up.
       4) Make sure part was released.
--****************************************************--

PUT_PART:SUBR(X);
   C: STATIC COUNTER; --local variable for move condition

   TRY:
     XXMOVE(CARD, LOCATIONS(X));  --move over the card location
                                  --location is relative to card!
                                  --(location z is above card-
                                  --this move only positions, it
                                  --does not insert the part)
     PAYLOAD (11);                --set to slow for guarded move
     GUARDI(GRIP_PRSS, ON);       --guard from over-pressure
     DDZMOVE(LOC_TO_CARD);        --insert the component
     MSTATUS(C);                  --query the motion complete code
                                  --save code in counter C
     NOGUARD;                     --disable motion guard
     PAYLOAD(0);                  --reset to switch setting
     OPENGR;                      --open gripper
     ZZMOVE(0);                   --insure Z up
     COMPC(C<>0, PROBLEM);        --problem if stopped by guard
     TESTI(GRPR_OBJS, OFF, OK);   --was part in fact inserted
   PROBLEM:
     WRITEO(GRPR_OPEN, OFF);      --relax the gripper
     WRITEO(GRPR_CLOSE, OFF);
     PROBLEM(PBLM_GRPR);          --inform operator
     BRANCH(TRY);                 --try again
   OK:
     END;
```

Initialization Subroutine

The example program uses a subroutine to initialize the external outputs and determine the status of the communications lines. The subroutine uses the variable INIT_FLAG to flag whether initialization is needed. INIT_FLAG is originally set to OFF in the variable declarations, when the INITIA subroutine has finished INIT_FLAG is set to ON. Every time INITIA is called it checks the value of INIT_FLAG. If INIT FLAG is ON a branch is made to the exit. In this way INITIA is only executed the first time it is called.

If INIT_FLAG has the value OFF the initialization is executed and the digital outputs are reset. After the output points are reset, the communications status is checked. If the communications status is incorrect the operator is informed and the program loops until the communications status is correct.

```
--*****************************************************************-.

                    Subroutine: INITIA

     1) Test the init_flag. If clear, perform the initialization,
        otherwise exit.
--   2) Initialization consists of:
           a) set the initialization flag
           b) initialize DO lines
           c) insure communications available
              if not - signal operator, wait for retry
           d) tell host we are available

--******************************************************* *********--


INITIA:SUBR;
   COMM_STAT: STATIC COUNTER;     --holds status of comm. system
   COMM_AVAIL: NEW 15;            --all comm parameters active
                                  --(CTS on, online, XONed state)

     TESTC( INIT_FLAG(1), ON, EXIT); --exit if already initialized

     --initialize DO
     WRITEO( HOST_STRTD, OFF);
     WRITEO( HOST_CMPLT, OFF);
     WRITEO( HOST_AVAIL, OFF);
     WRITEO( OPER_ATTN, OFF);
     WRITEO( PBLM_COMM, OFF);
     WRITEO( PBLM_FXTR, OFF);
     WRITEO( HOST:AVAIL,ON);        -- tell the host we are ready
     DELAY (2) ;                    -- give the host time to respond
     --read comm status, wait for good conditions
  LOOP:
     CSTATUS(COMM_STAT);            --determine communication status
     TESTC( COMM_STAT, COMM_AVAIL,  OK);
     PROBLEM(PBLM_COMM);            --identify problem
     WAIT_TOGGLE(OPER_RETRY);
     BRANCH(LOOP);                  --and retry
  OK:
     SETC( INIT_FLAG(1));           --set flag to indicate initialized

  EXIT:
     END;
```

The Main Subroutine

The final step in writing the program is to write the main subroutine. The MAIN subroutine contains the logic to control the interaction with the host computer and the movement of the manipulator. Because of extensive use of subroutines the MAIN subroutine is short and easy to understand.

The first step in the MAIN subroutine is to call the INITIA to ensure the condition of the manipulator is known and that communications with the host computer is correct. The INITIA subroutine is called every time the MAIN subroutine is executed however INITIA is designed to execute only the first time it is called. After the first time INITIA is called it returns without doing anything.

The controller then waits for the host to signal that it has loaded the correct data into the controller. The controller then signals the host that execution has started using the HOST_STRTD digital output pqint. The controller then uses the GET command to query the host for the component number and insertion information. The controller then inserts the components as instructed by the host until the host signals that all the components have been inserted or the operator stops the process. When all the components have been inserted the controller signals the host using the HOST_CMPLT digital output point and restarts the MAIN subroutine.

```
--*****************************************************************--
                         MAIN PROGRAM LOGIC
     Check initialization.

     Perform the component insert process.

■    1) Wait for Host's start signal.
        The line must be pulsed.
     2) Acknowledge Host
           a) Turn 'completed' signal off
           b) Turn 'started' signal on
     3) GET the Parts Insertion Locations from the host
        These points are defined with respect to the card!
     4) Run the insertion loop:
           a) Read instructions from the host:
              part type and insertion location
           b) Fetch the needed component from the proper feeder
           c) Insert into card at proper location
           d) advance instruction pointer,
        Run loop until all parts inserted.
        or until halted by Host
--   5) Signal Host that the process is finished.                 --
--*****************************************************************--

MAIN: SUBR;                         --start of main program
     INITIA;                        --do initialization
     WAIT_TOGGLE(HOST_START);       --wait for Host start signal
     WRITEO( HOST_CMPLT, OFF);      --process not completed yet
     DELAY(.2);                     --insure CMPLT off before STRTD
     WRITEO( HOST_STRTD, ON);       --indicate process is starting

                                    --the process loop
     GET(LOC_TO_CARD);              --z height of "location" data
  LOOP:
     GET(INSTRS);                   --get instructions from host
                                    --instructions is 2 counters:
                                    -- 1st tells what part type
                                    -- 2nd tells where to put on card
     COMPC(INSTRS(1) = 0, DONE);    --part-type=0 means all done
     TESTI(OPER ABORT, ON, ABORT);  --OK with Operator?
     GET_PART(INSTRS(1));           --get the part
     PUT_PART(INSTRS(2));           --insert it
     BRANCH(LOOP);
  ABORT:
     ZZMOVE(0);                     --move up for safety
     WAIT_TOGGLE(OPER_RETRY);       --wait for operator to signal
  DONE:
    WRITEO( HOST_CMPLT, ON);        --indicate done
    WRITEO( HOST_STRTD, OFF);       --no longer active
     END;                           -- end of program cycle
```

```
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
        GLOBAL TAUGHT POINTS AND THEIR ASSOCIATED DATA
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

-- Taught Points to Define Location of Card

CARD_UL:    NEW PT(0,0,0,0);          --card's upper-left corner
CARD LL:    NEW PT(0,0,0,0);          --card's lower-left corner
CARD_UR:' NEW PT(0,0,0,0):            --card's upper right corner
CARD_LR:    NEW PT(0,0,0,0);          --card's lower-right corner

-- Other Data to Define Card

CARD LS: NEW 0;                       --card's left side length
CARD'RS:   NEW 0;                     --card's right side length
CARD_TOP: NEW 0;                      --card's top length
CARD_BOT: NEW 0;                      --card's bottom length

-- Taught Points for fixture Locations

FEEDER1:    BEN PT(0,0,0,0);             Z MUST be zero here
FEEDER1Z:   NEW 0.0;                  -- This is Z coordinate.
FEEDER2:    NEW P7(0,0,0,0);             Z MUST be zero here
FEEDER2Z:   NEW 0.0;                     This 11:• coordinate.
FgEDER3•    NEW PT(0,0,0,0);             Z MUST be zero here •
FEEDER3Z:   NEW 0.0;                  -- This is Z coordinate.
FEEDER4•    NEW PT(0,0,0,0);             Z MUST be zero here
FEEDER4Z:   NEW 0.0;                  -- This is Z coordinate.

-- NOTE: these points have not yet been taught.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                DI POINT DEFINITIONS
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

HOST START:     NEW 03;   --Host controlled point.
                          --if 0: inactive.
                          --if 1: start to populate a card.

OPER_RETRY:     NEW 04;   --Signal from Operator: Retry.
                          --if 0: inactive.
                          --if 1: active.
OPER_OK:        NEW 05;   --Signal from Operator: OK to move.
                          --If 0: OK.
                          --if 1: Do Not Allow Motion.
OPER ABORT:     NEW 06;   --Signal from Operator: Abort the process.
                          --if 0: OK to proceed.
                          --if 1: abort the process.

CRIP_OPND:      NEW 07;   --Gripper opened sensor
                          --if 0: not fully opened.
                          --if 1: fully opened.
GR1P_CLSD:      NEW 08;   --Gripper Closed sensor
                          --if 0: not fully closed.
                          --If 1: fully closed.
GRIP_OBJS:      NEW 09;   --Gripper object sensor
                          --if 0: no object detected.
                          --If 1: object detected.
GR1P_PRSS:      NEW 10;   --Gripper pressure sensor
                          --if 0: no undue pressure detected.
                          --if 1: too much force being exerted.

--parts presence sensor for the feeders
FEEDER1S:       NEW 11;   --Fixture 1 sensor.
FEEDER2S:       NEW 12;   --Fixture 2 sensor.
FEEDER3S:       NEW 13;   --Fixture 3 sensor.
FEEDERIS:       NEW 14;   --Fixture 4   r.

          ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                DO POINT DEFINITIONS
          ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

HOST AVAIL:     NEW 03;   --Signal to Host: manipulator is available.
                          --if 0: not available.
                          --if 1: available.
HOST_STRTD:     NEW 04;   --Signal to Host: process started.
                          --If 0: process is not active.
                          --if 1: card population in progress.
HOST_CMPLT:     NEW 05;   --Signal to Host: process completed.
                          --if 0: process Is not active or incomplete.
                          --if 1: card population completed.


CRIP_OPEN:      NEW 06;   --Gripper open command
                          --if 0: no effect.
                          --If 1: open.
GRIP_CLOSE:     NEW 07;   --Gripper close command
                          --if 0: no effect.
                          --if 1: close.
```

```
 OPER_ATTN:       NEW 08; --Signal to Operator: Attention Required.
                          --If 0: no problem.
                          --if 1: problem encountered.
 PBLM_COMM:       NEW 09; --Signal to Operator: Problem is Comm.
                          --if 0: no problem.
                          --if 1: problem encountered.
 PBLM_FXTR:       NEW 10; —.Signal to Operator: Problem is Fixture.
                          --if 0: no problem.
                          --if 1: problem encountered.
 PBLM_GRPR:       NEW 11; --Signal to Operator: Problem is Gripper.
                          --If 0: no problem.
                          --If 1: problem encountered.

 --p.rts,release activator for the feeders
 FEEDERSA:        NEW 12; --Fixture 1 activator.
 FEEDERSA:        NEW 13; --Fixture 2 activator.
 FEEDERSA:        NEW 14; --Fixture 3 activator.
 FEEDER4A:        NEW 15; --Fixture 4 activator.

 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                   GLOBAL CONSTANTS
 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

ON:      NEW 1;
OFF:     NEW 0;
P:       NEW PT(0,0,0,0);       --generic point

 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                   GLOBAL VARIABLES
 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

INIT_FLAG:      STATIC GROUP
                (OFF);          -- A counter initialized to OFF.
                                -- Will be set to ON after program has
                                -- been initialized.

FEEDERSL:       STATIC CROUP --locations of feeders
                (FEEDER1, FEEDER2, FEEDER3, FEEDERS);
FEEDERSZ:       STATIC GROUP --Z height of feeders
                (FEEDER1Z, FEEDER2Z, FEEDER3Z, FEEDER42);
FEEDERSA:       STATIC GROUP --actuator DO point for feeders
                (FEEDER1A, FEEDER2A, FEEDERSA, FEEOER4A);
FEEDERSS:       STATIC GROUP --sensor DI point for feeders
                (FEEDER1S, FEEDER2S, FEEDER3S, FEEDER4S);

CARD:           STATIC REGION(CA00_LL,CARD_UL.CARD_LR, CARD_OR,
                              CARD_LS,CARD_RS,CARD_BOT,CARD:TOP);

LOCATIONS:      STATIC GROUP --component insert locations
                (P.P.P.P.P.P,P.P,P.P);
                                --location points are relative to card

LOC_TO_CARD:    STATIC COUNTER;--will contain host specified data-
                                --z          from "location" to "card"

INSTRS:         STATIC GROUP --insertion instructions
                (0,0);          --part-type , insertion location

—*se.................................................
                     Subroutine: WAIT_TOGOLE( PORT );

     1) Wait for the selected port to be toggled- that is,
        It must be off, then pulse on, then return to off.

        This routine will wilt forever.
 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

WAIT_        : SUBR(PORT);
     WAIT1( PORT, OFF, 0);; --Insure OFF
     WAIT1( PORT, ON, 0); --wait for ON
     WAITI( PORT, OFF, 0); --and OFF again
     END;

 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                     Subroutine: OULSE(PORT,TIME);

     1) Pulse the selected port from OFF to ON to OFF.
        The pulse will have a dutx cycle specified by TIME. ■ ■ ■ ■

PULSE:SUBR(PORT,Y);
     WRITEO(PORT, OFF);
     DELAY(Y).
     WRITEO(PORT, ON);
     DELAY(Y);
     WRITEO(PORT, OFF);
     ENO;

••■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
                       Subroutine: PROBLEM( PORT );

     1 Signal the operator that a problem has been encountered.
     2 Wait for operator to Signal ON to retry.
     3)Reset pr:blem Indicators.

PROBLEM:SUBR(X);.
     WRITEO( OPER ATTN, ON); --ask for operator's attention
     WRITEO( X, ON)4           --indicate where problem is
     WAIT TOOGLE(OPER RETRY); --welt for retry signal
     WRITEO            OFF); --turn off signals
     WRITEO( X. °Fr);
     END;
```

■ ■

SubroutineS:
PPMOVE(POINT), UMOVE(POINT), XXMOVE(POINT), DOZMOVE(DELTA)

The following four subroutines are used instead of the standard move Commands. They allow you to monitor a DI point to control movement. If OPER_OK Is not ON the no movement is spirted 4 the subroutines wait forever.

```
PPMOVE:SUBR(X);
     WAIT1(OPER_OK, ON, 0);    --OK to move?
     PMOVE(X);
     END;

ZZMOVE:SUBR(X);
     WAIT1(OPER_OK, ON, 0);    --OK to move?
     ZMOVE(X);
     END;

XXMOVE:SUBR(R,X);
     WAITI(OPER_OK, ON, 0);    --OK to move?
     XMOVE(R,X);
     END;

DOZMOVE:SU(RIX);
     WAIT1(OPER_OK, ON, 0);    --OK to move?
     DPMOVE(<0,0,0,X>);
     END;
```

■ ■

Subroutine: OPENOR

Open the gripper

1) Activate gripper open DO point.
2) Walt for open sensor to be active,
   If not active in 1 second, then it must
   be lammed, so signal operator.

■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■

```
OPENCR:SUBR;
  TRY:
     WRITEO(GRIP_CLOSE, OFF);--insure not trying to close
     WRITEO(CRIP_OPEN, ON); --try to open
     CHECK1(CRIP_OPND, ON, 1, OK); --give 1 sec. to open
  ERR:
     PROBLEM(PBLM_CRPR);        --report problem
     BRANCH(TRY);               --retry
  OK:
     END;
```

■ ■

Subroutine: CLOSECR

Close the gripper

1) Activate gripper close DO point.

■ Mb**■ ■ ■ ■

```
OPENGR:SUBR;
     WRITENCRIP_OPEN, OFF);    --insure not trying to open
     DELAY(.1);
     WROTEO(CRIP_CLOSE. ON);   --try to close
     END;
```

■ ■

Subroutine: DROP_PART(X)

Drop a part at feeder X into the Jig.

1) Activate the feeder DO point.
2) Wait for part presence sensor to be active,
   if not active in 1.5 second, then it must
   be Jammed, so signal operator.

■ ■

```
DROP_PART:SUBR(X);
  TRY:
     PULSE(FEEDERSA(X),.2);     --pulse the part release port
     CHECK1(FEEDERSS(X), ON,  1.5, OK);
                                --allow 1.5 sec. for part
  ERR:
     PROBLEM(PBLM_FXTR);        -report problem
     BRANCH(TRY);               --retry
  OK:
     END;
```

```
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ '
        Subroutine: CET_PART(PART_TVPE)

     Get the part specified by feeder number

     1) Insure Z-stroke Is fully up.
    .2) Insure gripper Is open.
     9) Move to the proper location.
     4) Release a component into Fixture's Jig.
     5) Move to proper Z height.
     6) Grasp the part.
     7) Move to Z fully up.
     B) make sure part was grasped.
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ '

GET ►ART:SUBR(X);
   TRY:
     ZZmOVE(0);                    --insure Z Op
     OPENOR;                       --open gripper
     PMOVE(FEEDERSL(X));           -- move over the feeder
     DROP_PART(X);                 --activate feeder (release a part)
     ZZMOVE(FEERERSZ(X));          --go to proper height
     CLOSEGN;                      --close gripper
     ZZMOVE(0);                    --move z up to safe height
     TESTI( CRPR_OBJS, ON, OK);      ok if there
     WRITEO(GRPR_OPEN, OFF);       -- relax the gripper
     WRITEO(CRPR_CLOSE, OFF);
     PROBLEM(PSLM CRPR);           --inform operator
     BRANCH(TRY);                  --try again
   OK:
     END;
```

```
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ '
        Subroutine: PUT_PART(PART_PLACE)

     Insert the part specified by location number

     1) Move to the proper location.
     2) insert the part.
        - turn gripper force sensor guarding on
        - mo9e down until tripped or inserted
     3) Move to Z fully up.
     4) Make sure part was released.
■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ '

PUT PART:SUBR(X);
   E: STATIC COUNTER; --local variable for move condition

   TRY:
     XXMOVE(CARD, LOCATIONS(X));  --move over the card location
                                  --location is relative to cardl
                                  --(location z is above card-
                                  -- this move only positions, it
                                  -- does not insert the part)
     GUARDI(GRIP_PRSS, ON);       --guard from over-pressure
     DOZMOVE(LOC_TO_CARD);        --insert the component
     MSTATUS(C);                  --query the motion complete code
                                  --save code in counter C
     NOGUARD;                     --disable motion guard
     OPEACR;                      --open gripper
     ZZMOVE(0);                   --insure Z up
     COMPC(C<>0, PROBLEM);        --problem if stopped by guard
     TESTI(CAPR_OBJS, OFF, OK);   --was part in fact Inserted
   PROBLEM:
     WRITEO(GRPR_OPEN, OFF);         relax the gripper
     WRITEO(GRP(_CLOSE, OFF);
     PROBLEM(PSI.M_ORPR);         --inform operator
     BRAMO(TRY);                  --try again
   OK:
     ENO;
```

■ ■

Subroutine: INITIA

1) That the init_flag. If clear, perform the initialization, otherwise exit.
2) Initialization consists of:
   a) set the initialization flag
   b) initialize 00 lines
   c) insure communications available
      if not - signal operator, wait for retry
   d) tell host we are available

■ ■

```
INITIA:SUBR;
  COMM_STAT: STATIC COUNTER;       --holds status of comm. system
  COMM_AVAIL: NEW 7;               --all comm parameters active
                                   --(CTS on, online', XONed state)

    TESTC( INIT_FLAC, ON, EXIT);   --exit if already initialized

    --initialize DO
    WRITEO( HOST STRTD, OFF);
    WRITE0(.HOST_CMPLT, OFF);
    WRITEO( HOST_AVA1L, OFF);
    WRITEO( OPER_ATTN, OFF);
    WRITEO( PBLM_COMM, OFF);
    WRITEO( POLM_FXTR, OFF),
    --read come status, wait for  good conditions
  LOOP:
    CSTATUS(COMM STAT);            --determine communication status
    TESTC( COMM 3TAT, COMM AVAIL,  OK);
    PROBLEM(PBOCOMM);              --identify problem
    WAIT TOGGLE(OPER_RETRY);
    BRANEH(LOOP);                  --and retry
  OK:
    WRITEO(HOST AVAIL,ON);         --tell Host we are ready
    SETC( INIT_TLAG,ON);           --set flag to indicate initialized

  EXIT:
    END;
```

■ ■

MAIN PROGRAM LOGIC

Check initialization.

Perform the component insert process.

1) Wait for Host's start signal.
   The line must be pulsed.
2)' Acknowledge Host
   a) Turn 'completed' signal off
   b) Turn 'started' signal on
3) GET the Parts Insertion Locations from the host
   These points are defined with respect to the card!
4) Run the Insertion loop:
   a) Read instructions from the host:
      part type and insertion location
   b) Fetch the needed component from the proper feeder
   c) 1 .... into cord at proper location
   0) advance instruction pointer,
   Run loop until all parts inserted.
   or until halted by Host
5) Signal Host that the process Is finished.

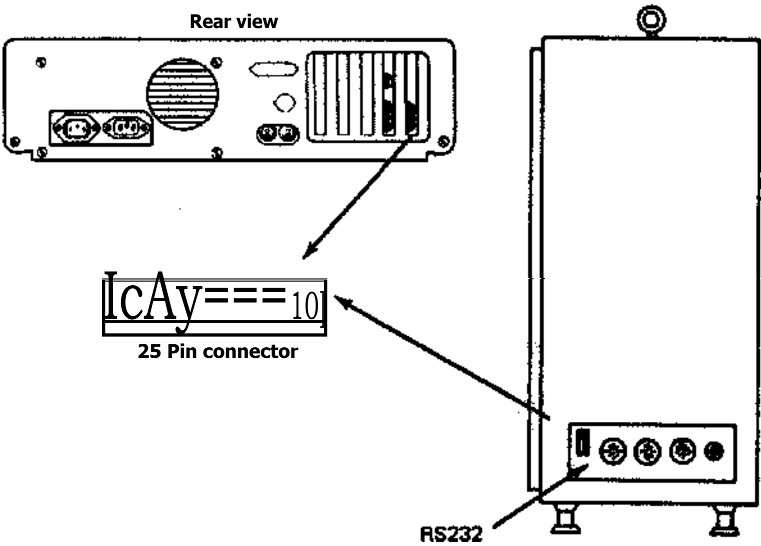
■ ■

```
MAIN : SUBR ;                    --start of main program
    INIT IA;                       --do initialization
    WAIT TOGGLE(HOST_START);       --wait for Host start signal
    WRITEO( HOST CMPLT, OFF);      --process not COmpleted yet
    DELAY(.2);                     --insure CMPLT off before STRTD
    WRITEO( HOST_STRTD, ON);       --indicate process is starting

                                   --the process loop
    GET(LOC_TO_CARD);              --z height of "location" data
  LOOP:
    GET(1NSTRS);                   --get instructions from host
                                   --instructions is 2 counters:
                                   -- 1st tells what part type
                                   -- 2nd tells where to put on card
    COMPC(1NSTRS(1) ■ 0, DONE);    --part-type=0 means all done
    TEST1(OPER_ABORT, ON, ABORT);  --OK with Operator?
    GET_PART(1NSTRS(1));           --get the part
    PUT_PART(1NSTRS(2));           --insert it
    BRANCH(LOOP);
  ABORT:
    2ZMOVE(0);                     --move up for safety
    WAIT_TOGGLE(OPER_RETRY);       --wait for operator to signal
  DONE:
   WRITEO1 HOST_CMPLT, ON);        --indicate done
   WRITEO( HOST_STRTO, OFF);       --no longer active
    ENO;          -- and of program cycle
```

CHAPTER 6. USING THE AML/ENTRY TEACH MODE

This chapter describes the functions of the teach mode and provides yoi with exercises on how to use them. The exercises require that th IBM-supplied blue RS-232 communications cable between the IBM Personal' Computer and the manufacturing system controller be connected. The cablei connection is shown in the following figure.



During the exercises, you are transferring coordinates from the teach mode back to the editor for use in an application program. You will se different uses of the function keys and the cursor control keys on th IBM Personal Computer keyboard. The section titled "Teach Model Exercises" provides a description of how to get ready for teach.

CAUTION

Teach mode moves the manipulator by sending communications requests to the controller. In certain circumstances, the manipulator may move home unexpectedly. This happens whenever one of two events occurs:

1. **The manipulator loses power, whether by an error (i.e. servo error, overrun error, etc.) or because the stop button is hit.**

2. **The controller is taken off-line, the manipulator is moved, and the controller is placed back on-line.**

When either of these events occurs, Teach mode should be exited by hitting the "end" *key* immediately. If this is not done, damage to the manipulator or fixtures could occur because of the unexpected move home. After the move home, the manipulator will return to the current location indicated on the Teach screen.

TEACH MODE

Teach mode is entered from the editor by pressing the F6 key. It suspends editing and displays the work envelope.

CAUTION
The manipulator may return to the home position when the F6 key is pressed. Guide the arm to a position where the home operation does not strike objects and damage the manipulator. Use the Z up key on the control panel to do this.

Teach mode allows you to determine the coordinates of a location for your programs. You do this by moving the manipulator, using the cursor keys of the IBM Personal Computer. When you are satisfied with the' position, you can return to the editor by pressing the **End** key on the numeric keypad. When you return to the program, you can recall the value of the taught point. Some of the features of teach mode are:

- Display the IBM Manufacturing System work envelope on the screen and see the position of the manipulator.
- Move the manipulator under control of the IBM Personal Computer.
- Control the state of any digital output point.
- Lower or raise the Z-axis.
- Open or close a gripper if one is installed.
- Determine a point to be recalled in your AML/Entry program.
- Return to your program in the editor.
- Retrieve a point from a program and display the values on the teach screen.
- Change arm mode (7545-800S only)

When you are in the teach mode, your screen will look similar to the following screen, depending on which manipulator the AML/Entry system is configured for.

```
                    00000000000040000000*
                  0000000 04000000 b000000 0000**
                0006000040000000000000000000004000*
              0000 • 9000000 0000000000000000000000000*
            000000000000000000 000000000000400004000004*
          00000000008 000000000004000000000000000000*
        *0000 00000400000000000040004000004000000000000004*
        00000000000000000000000000000000000000000000000404*
      0006000000'000000 00000000000000000004***** *****000*
     *0060000040000000000 • o o*******o o o 0.*              00*
      000000-0000000000000000          ***                     ***
      '0'00000000 000000000*
     0000000040000000000000                                     #
     *0000000000000-004 00*          4
      e00000000000000004*
     **00000000000060000000*
      *000000000044 0000 00*
       *0000000000000000 0*
        *000000000000000*
          *00000040040**
              **Int

   Current Robot Location e          191.41      0.00

 1HELP                                6 D/0 /DOWN          SP 10 LEASE
```

Note; The screen that shows the work envelope for the 7545-8008 is quite different from those for other models. A figure showing that screen can be found in the discussion "LEFT Command (Valid on 7545-800S Only)" on page 4-9.

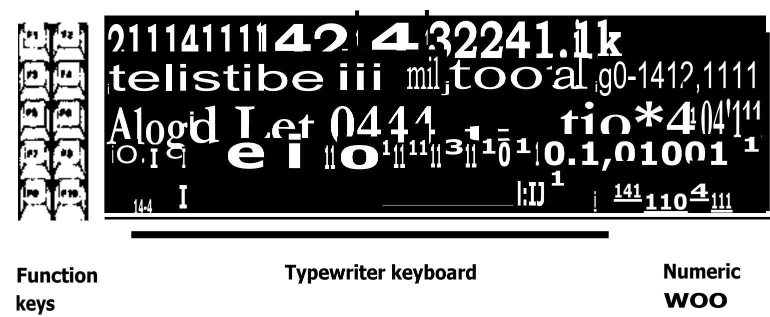
The characters around the work envelope have special meanings when using teach. The section describing the precision and coarse moves contains a description of the work envelope characters.

FUNCTION KEYS

Function keys have special uses while you are in the Teach mode. Settings of these keys are displayed at the bottom of the Teach mode screen.

Key	Description
F1	Displays the Teach mode 14EI_P screen.
F2	Displays the set motion parameter screen.
F3	Retrieves and displays the next global point in a program.
F4	Retrieves and displays the previous global point in a program.
F5	Displays the read DI/DO points screen.
F6	Displays DO control screen.
F7	Lowers the Z-axis shaft.
F8	Raises the Z-axis shaft.
F9	Closes the optional gripper.
F10	Opens the optional gripper.

Note: The Z-axis can be stopped at any position between 0 and -250 mm. The shaft moves at a slow speed, but a coarser movement can be obtained by pressing the shift key simultaneously with the F7 or F8 key.

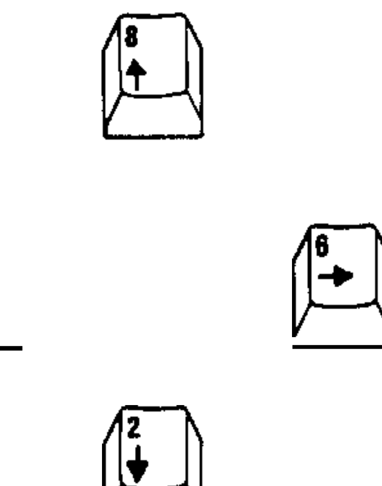


SPECIAL KEYS

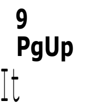
The Esc key allows you to enter point coordinates, and the manipulator then moves to that position. You enter X, Y, Z, and Roll coordinates.

The I key allows you to bypass communications to the controller when you want to display the teach screen without using the manufacturing system. This happens on entry to the Teach system from the editor. If communications cannot be established with the controller, then an Abort, Retry, Ignore message is printed. Option i will then give you the ability to enter Teach mode without moving the manipulator.

The following keys located on the numeric keypad have special applications when you use the teach mode:

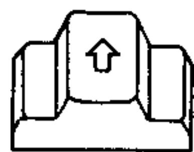


Cursor keys move the cursor across the screen display of the work envelope in the direction indicated by the arrow on the key. The movement of the cursor is transmitted to the controller instructing the manipulator to move to that coordinate of the work envelope. You use th cursor keys for precision adjustments of the manipulator position. I you hold down a cursor key, the speed of manipulator movement increases. The screen displays the position of the manipulator.



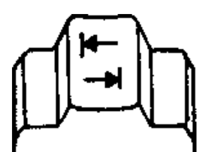
PgUp rotates the roll axis in increments of 0.36° in clockwise direction and accelerates as the key is held down. Roll numbers become more positive.

PgDn rotates the roll axis in increments of 0.36° in a counterclockwise direction and accelerates as the key is held down. Roll numbers become less positive.



Press the shift key at the same time as a cursor key when coarse manipulator movements are desired. Press the shift key at the same time as the F3 or F4 key when you want every 10th point to be recalled.

The **End** key exits the teach mode and returns to the editor. Also use this key when you want to exit the DO control menu without implementing any changes to DO status.



The **TAB** key switches the arm configuration for the 7545-800S manipulator. When a 7545 or 7547 manipulator is attached, this key is inactive. To switch the 7545-800S to right mode, simply strike the **TAB** key. To switch the 7545-800S to left mode, strike the **TAB** and shift key simultaneously. The user is warned of an upcoming change of configuration and can abort the change if desired.

Warning:

The TAB LEFT and TAB RIGHT keys may invoke motion. Whenever the current point is in the region just designated with a TAB LEFT or a TAB RIGHT key, the robot moves.

Note that there is a * region at the bottom of the workspace. Should the user desire a change of modes in this region, the robot essentially makes a full circle around the workspace. Thus the user should not request a flip of modes in the lower section of the workspace.

READ DI/DO POINTS

To read all installed DI and DO points on the system, press F5 in Teach mode. Once a specific value for the DI/DO number to be read is entered, the controller is instructed through communications protocol to read the values of all installed DI/DO points.

After the values of the specifically-requested point are displayed on the screen, the values of the adjacent DI/DO points are viewed by pressing the up-arrow key to examine the value of the next sequential point or the down-arrow to view the previous sequential point. Press the <---(enter) key to return to the initial screen. At this point, you are allowed to enter a new DI/DO point to read. After entering this value, the controller is once again queried for data. Values of all DI/DO points are either On, Off, or -- (not installed). If the End key is pressed any time after the F5 key is pressed, you are returned to Teach mode.

Data is read only when a DI/DO point is entered. If the states of the DI/DO points are changed, the changes are not displayed until a new DI/DO point is entered.

INITIAL SETTING OF MOTION PARAMETERS FOR SAFETY

To further increase the safety features of the manipulators, the motion parameters are preset to the safest values on entry to the Teach system. The settings, made before the homing of the arm, are outlined below.

```
LINEAR = 0
PAYLOAD = 1
ZONE = 1
```

SET MOTION PARAMETERS IN TEACH MODE

CAUTION

Ensure that no person is in the manipulator work space when using Teach. If an error occurs during Teach, the controller returns to the default switch setting speed, which can result in high speed moves that are not anticipated. To regain control of the motion parameters, you must return to the AML/Entry Menu and then re-select the Edit/Teach option.

To both view and modify the motion parameters, press the F2 key in Teach. The current values for LINEAR, PAYLOAD, and ZONE are displayed in a window. To make changes to the values, you need only enter the new values in the appropriate fields and press the <---┘ (enter) key. The values are automatically updated in the controller. To leave the window and return to the teach screen, press the End key.

EXITING THE DIGITAL OUTPUT CONTROL UTILITY IN TEACH

You are only able to exit the digital output control utility in teach by pressing the End key. This is in accord with all other functions in the Teach system that observe the same exit procedure. When the Return key is pressed, the initial Digital Output Control screen is displayed and you are able to enter a new DO point.

IBM MANUFACTURING SYSTEM TEACH RESPONSES TO PREVIOUS CONDITIONS

The manufacturing system responds differently to initial teach signals depending on the previous conditions. The following descriptions provide the different starting conditions and the responses of the manufacturing system upon entering teach mode or changing a status.

Condition 1 (Manipulator Power Off)

Condition 1 - Manipulator Power Off

1. Manipulator power is applied at the control panel but the Return Home key is not pressed.
2. At the IBM Personal Computer, the editor is entered.
3. At the IBM Personal Computer, the **F6** key is pressed to enter teach mode.

   Response - Manipulator returns to the Home position slowly once and then perfornis a second return home at high speed. All digital output (DO) points are set to the off position. The Z-axis assumes the up position and the gripper, if provided, opens.

Note: After the **F6** key is pressed, the Personal Computer issues a message stating the manipulator is going to move to the Home position. It also tells you to make sure the path is clear for manipulator movement, and tells you how to leave teach mode if necessary.

Condition 2 (Manipulator Power On, Return Home Performed)

Condition 2 - Manipulator Power On and Return Home already performed at the control panel

1. At the IBM Personal Computer, the editor is entered.
2. At the IBM Personal Computer, the **F6** key is pressed to enter teach mode.

   Response - Manipulator returns to the Home position at a fast speed and all digital output (DO) points are set to the off position. The Z-axis assumes the up position and the gripper, if provided, opens.

Note: After the **F6** key is pressed, the Personal Computer issues a message stating the manipulator is going to move to the Home position. It also tells you to make sure the path is clear for manipulator movement, and tells you how to leave teach mode if necessary.

Condition 3 (Exit Teach)

Condition 3 - You exit teach mode and the editor after moving the manipulator via a teach operation.

1. At the IBM Personal Computer, the editor is selected at the menu.

2. At the IBM Personal Computer, the **F6** key is pressed to enter teach mode.

3. No operator action is taken at the control panel.

   Response - Manipulator returns to the Home position at a fast speed and digital output (DO) points are **NOT** affected. The Z-axis remains in the position last taught, and the gripper, if provided, remains either open or closed.

   Note: After the **F6** key is pressed, the Personal Computer issues a message stating the manipulator is going to move to the Home position. It also tells you to make sure the path is clear for manipulator movement, and tells you how to leave teach mode if necessary.

Condition 4 (Exit Teach, Remain in Editor _ Control Panel Move)

Condition 4 - You exit teach mode at the IBM Personal Computer but remain in the editor after moving the manipulator to a point. You then use the control panel to control the manipulator position, Z-axis, or gripper.

1. At the IBM Personal Computer, the **F6** key is pressed to enter teach mode. The cursor is located at the last taught point. The manipulator does not move from the position located using the control panel.

2. You press a cursor key to move the manipulator or change a DO status using the function keys.

   Response - Manipulator returns to the Home position at a fast speed and all digital output (DO) points are set to the off position. The Z-axis assumes the up position, and the gripper, if provided, opens before the manipulator moves to the Home position. If cursor key has been pressed, the manipulator moves to the new cursor position rapidly once the manipulator is at the Home position. This occurs in a coordinated move ending when all axes are in their correct position.

If a DO status change was requested, the manipulator stays at the Home position and the DO changes to the requested on or off position. When a cursor key is pressed the manipulator rapidly moves to the cursor position.

DANGER

THE MANIPULATOR CAN UNEXPECTEDLY MOVE TO THE HOME POSITION AND BACK TO THE CURSOR POSITION WITH NO NOTICE GIVEN. ENSURE THE PATH TO AND FROM THE HOME POSITION IS CLEAR BEFORE RETURNING TO TEACH MODE. THE SPEED AND FORCE OF THE MANIPULATOR CAN CAUSE SERIOUS INJURY TO PERSONNEL AND DAMAGE TO EQUIPMENT IN ITS PATH.

<u>Condition 4a</u> - You use the operator panel to enter manual mode from teach mode. You then use the control panel to control the manipulator position, Z-axis, or gripper.

1. Using the operator panel, you put the system on-line so the manipulator can be moved in teach mode.

2. You press a cursor key to move the manipulator or change a DO status using the function keys.

   Response - Manipulator returns to the Home position at a fast speed and all digital output (DO) points are set to the off position. The Z-axis assumes the up position, and the gripper, if provided, opens before the manipulator moves to the Home position. If a cursor key has been pressed, the manipulator moves to the new cursor position rapidly once the manipulator is at the Home position. This occurs in a coordinated move ending when all axes are in their correct position. If a DO status change was requested, the manipulator stays at the Home position and the DO changes to the requested on or off position. When a cursor key is pressed the manipulator rapidly moves to the cursor position.

DANGER

THE MANIPULATOR CAN UNEXPECTEDLY MOVE TO THE HOME POSITION AND BACK TO THE CURSOR POSITION WITH NO NOTICE GIVEN. ENSURE THE PATH TO AND FROM THE HOME POSIT ION IS CLEAR BEFORE RETURNING TO TEACH MODE. THE SPEED AND FORCE OF THE MANIPULATOR CAN CAUSE SERIOUS INJURY TO PERSONNEL AND DAMAGE TO EQUIPMENT IN ITS PATH.

Condition 5 (Exit Teach, Remain in Editor after Manipulator Move)

Condition 5 - You exit teach mode at the IBM Personal Computer but remain in the editor after moving the manipulator to a point. No changes are made at the control panel after teach mode is invoked initially and the communications connection is maintained between the IBM Personal Computer and the manufacturing system.

1. At the IBM Personal Computer, the F6 key is pressed to enter teach mode.

2. You press a cursor key to move manipulator or change a DO status using the function keys.

   Response - If the cursor is moved, the manipulator moves to the new position without returning home. If a DO status change was selected by a function key, the controller responds and no return home occurs.

TEACH MODE EXERCISES

In the following exercise, you are going to use the editor on a new file. Your exercise consists of using the teach features and recalling a point for a small program that only performs manipulator moves. The completed exercise creates a program that can be compiled and loaded into the controller.

The editor should display a new file with no name when you start. Refer to Chapter 3, "Using the AML/Entry Editor" for editor access.

You: In the editor, press the enter (<—J) key.

System: Cursor moves to the TOP•OP•TILE line.

The next instruction inserts four lines in the editor.

You: Type: **14**

System: The cursor moves to the right of line 1.

You: Type: **OUTER:SUBR;**
You: Press: Ctrl and the enter (<—J) key.
You: Type: **PMOVE(PT**
You: Press: Ctrl and the enter (<----') key.
You: Type: PMOVE(PT
You: Press: Ctrl and the enter (<—J) key.
You: Type: **END;**

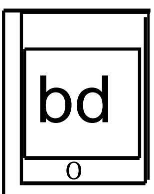
Your file is ready for the exercise.

POWER-UP SEQUENCE FOR TEACH EXERCISES

Use the following steps to power-up the manufacturing system and to establish communications between the IBM Personal Computer and the controller. If an indicator or switch is in the desired state during the power-up sequence, skip that step.

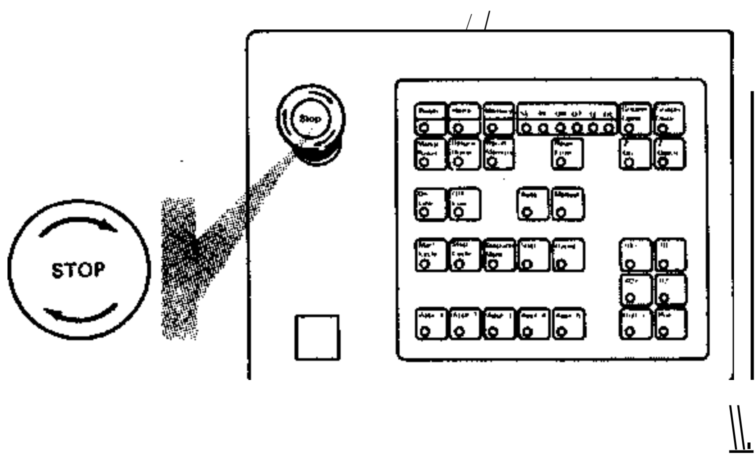
You: At the controller, set the power circuit breaker

to I (on)



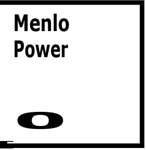
System: The power lamp on the front of the controller lights and the Power, On Line, and Manual LEDs on the control panel light. The fan motor in the controller starts.

You: At the control panel, turn the Stop pushbutton clockwise to the on position.



System: The switch is in the on position (raised).

You: At the control panel, press the **Manip Power** key.



System: The Manip Power indicator lights.

Note: If the On line LED is not lit, you must press the key.

You: Wait 10 minutes for the controller and the manipulator to warm up.

DANGER

BE SURE THE PATH IS CLEAR FOR THE RETURN HOME PERFORMED BY THE MANIPULATOR WHEN THE RETURN HOME KEY IS PRESSED ON THE CONTROL PANEL.

STAND CLEAR OF THE MANIPULATOR WORK ENVELOPE BEFORE THE NEXT STEP. THE SPEED AND FORCE OF THE MANIPULATOR CAN CAUSE SERIOUS INJURY TO PERSONNEL AND DAMAGE TO EQUIPMENT IN ITS PATH. THE MANIPULATOR MAY MAKE TWO RETURN-TO-HOME MOVES WITH THE SECOND RETURN TO HOME USING HIGH SPEED IN ORDER TO CALIBRATE THE SYSTEM.

You: At the control panel, press the **Return Home** key.

System: Manipulator moves to the Home position. After the manipulator stops moving, the Home indicator lights.

DANGER

STAND CLEAR OF THE MANIPULATOR WORK AREA. THE NEXT INSTRUCTION MAY GENERATE A RETURN HOME SIGNAL TO THE MANIPULATOR.

You: At the IBM Personal Computer, press: **F6**

System: Screen blanks and then refreshes, displaying the work envelope of the manipulator.

Coarse Movement

Pressing the shift and cursor keys on the numeric keypad moves the cursor across the screen. If the location of the cursor is within the manipulator work envelope, the manipulator moves to the location indicated by the cursor. The rectangular symbol in the work area displayed on the screen is the actual location of the manipulator within the work envelope. Coarse adjustments to the X- and Y-coordinates are made by using the shift key in combination with the cursor keys.

Coarse moves are restricted to areas of the work envelope display that have the character (u). Precision moves are permitted at locations that have either the above character or the character (*). Coarse adjustments to the roll angle are made by using either the **PgUp** or **PgDn** keys in combination with the shift key.

This exercise demonstrates the movements obtained when pressing the cursor and shift keys. This method is used for coarse adjustments in locating a point in the work area.

You: Using the cursor keys on the numeric keypad, move the cursor within the display area of the work envelope.

You: Observe the movement of the cursor, the rectangular symbol, and the changing current position numbers at the IBM Personal Computer. Observe the movement of the manipulator while a curs or page key is pressed at the same time as the shift key.

Precision Movement

The following exercise uses the cursor keys and the page keys **(PgUp, PgDn)** that move the manipulator when precision movements are needed. Observe the screen of the IBM Personal Computer for changing values of X, Y, and R. The movement of the manipulator is difficult to detect at first.

Precision moves are permitted at locations with either the character (°) or the character (*).

You: Use any cursor or page key on the numeric keypad. Keep the cursor within the display work envelope.

You: Observe the changing current position numbers on the bottom of the teach screen while you press each cursor key. Also, observe the movement rate of the rectangular symbol and the manipulator.

You: The **F7** and **F8** keys can be used to move the Z axis slowly up and down. The only indication on the screen is the changing value for Z.

Note: If you move the cursor out of the work area, the manipulator does not make that move. The screen displays **TARGET NOT WITHIN** ROBOT **WORKSPACE** while the cursor is out of the work area.

DANGER

WHEN THE CURSOR RETURNS TO THE WORK ENVELOPE, THE MANIPULATOR MAKES A FAST MOVE TO THE POSITION OF THE WORKSPACE THAT THE CURSOR INDICATES.

Entering Known Coordinates

This exercise demonstrates the ability to enter a point. Once a valid value (one within the work area) is entered, the manipulator moves tol that location.

You: Press: Esc

System: Screen displays X=

You: Type: -315.5
You: Press the enter (<---I) key.

System: Screen displays **Y=**

You: Type: 100.4
You: Press the enter (<—J) key.

System: Screen displays Z=

You: Type: -100
You: Press the enter (<-----) key.

System: The screen displays **R=**

You: Type: -180
You: Press the enter (<—J) key.

System: Screen displays new coordinates for X, Y, Z, and R,and the manipulator moves to the coordinates. The screen displays the location of the manipulator.

You: Press: Esc

System: Screen displays X=

You: Press the enter (<---I) key.

System: Screen displays **Y=** and the X-coordinate remains the same because no new value was entered.

You: Press the enter (<---) key.

System: Screen displays **Z=** and the Y-coordinate remains the same because no new value was entered.

You: Press the enter (<—J) key.

System: Screen displays **R=** and the coordinates of Z remain the same because a new value was not entered.

You: Type: **180**
You: Press the enter (<---J) key.

System: Screen displays coordinates for X, Y, Z, and R. The manipulator moves to the new R-coordinate.

Return Point Value to Program (Recall)

Now return to the editor where you can put the coordinates of your last manipulator position into your program.

You: Press: **End**

System: Screen blanks and then displays the editor with your program.

You: Using the cursor keys on the numeric keypad, position the cursor after the **PT** of line 2.
You: Press: **F7**

System: Recall key displays the taught point on the screen at the cursor position. The values are enclosed in parentheses ( ) and followed by a semicolon.

You: Press: **Ins**
You: Using the cursor keys on the numeric keypad, position the cursor under the semicolon (;) at the end of line 2.

You: Type a right parenthesis).

System: Your statement should have two left, two right parentheses, and a semicolon.

You: Press: Ins

Note: Always check the quantities of right parentheses) and semicolons (;) dt the end of the statement. If you replace values using the Recall feature with one having fewer characters, an unequal number of parentheses causes errors when compiling the program.

You do not have to use the recall immediately after you have taught a point. You can type other statements and then do a recall.

Obtaining an Additional Point

This exercise returns the IBM Personal Computer and manufacturing system to the teach mode to obtain a second point for your application program. The indicators and switches on the system do not need any additional steps if you have not changed any setting since teach was exited the first time.

You: Press: F6

System: Screen blanks and then displays teach. The manipulator is still in the last position when you exited teach.

You: Use the cursor control keys to locate a new coordinate.

System: Manipulator moves to the new coordinate and the new location is displayed on the screen.

You: Press: **End**

System: Screen blanks and then displays the editor with your program.

You: Use cursor keys to move the cursor after the **PT** of line 3.
You: Press: F7

System: A value for the PT is transferred from teach mode, using the recall feature.

You: Press: **Ins**
You: Using the cursor keys on the numeric keypad, position your cursor under the semicolon (;) at the end of line 3.
You: Type a right parenthesis).

System: Your statement should have two left, two right parentheses, and a semicolon.

You: Press: Ins

The program is complete for manipulator moves between the points. You could save, compile, and load this program into the controller.

Retrieving a Point from a Program

The recall function of teach allows you to retrieve a point location from a program using the **F3** (Recall+) and **F4** (Recall-) keys. You may display the values on the screen, modify any or all of the values, and move the manipulator arm to that point.

Before you begin the exercise, understand the following rules, which apply when retrieving a point from a program:

- The retrieved point value must be global. That is, it must be defined before the first SUBR statement. In this example,

  ```
  POINT1:NEW PT(650,0,0,0);


   OUTER:SUBR;
  ```

  POINT1 is retrievable from the teach screen. But, it is not if it is defined:

  ```
   OUTER:SUBR;
  POINT1:NEW PT(650,0,0,0);
  ```

- Only one PT per line can be recalled. Thus if two PT's appear in the same line, only the first can be recalled.

- The PT definition statement must have a left parenthesis immediately following the PT, with no blank space in between. Teach does not retrieve a PT definition followed by a blank. For example,

  ```
  POINT1:NEW PT(650,0,0,0);
  ```

  is a valid PT definition. But in the following statement,

  ```
  POINT1:NEW PT (650,0,0,0);
  ```

  POINT1 would not be retrieved.

- The point location must contain absolute coordinates; that is, it must be numeric, not a declared point name. For example,

```
POINT1:PT(650,0,0,0);


 OUTER:SUBR;
```

is an acceptable point; the following is not.

```
SIXFIFTY:NEW 650;
    ZERO:NEW 0;
  POINT1:NEW PT(SIXFIFTY,ZERO,ZERO,ZERO);



   OUTER:SUBR;
```

The following exercise demonstrates the recall function of teach mode.

Note: At any time during this procedure, you may press the **End** key to terminate this function. The teach screen remains displayed.

You: Position the cursor on the TOP•OF•FILE line.
You: Type: **i2**
You: Press the enter (<—J) key.

System: Blank lines have been inserted for lines 1 and 2.
The cursor is to the right of line 1.

Your Type: **PT1 : NEW PT(-650,0,0,0);**
You: Press the enter (<—J) key.

System: Cursor moves to the right of line 2.

You: Type: PT2 : NEW PT(0,650,0,0);
You: Press: **F6**

System: Screen blanks and then displays the teach screen.
The manipulator arm is still in the last position when you exited teach.

You: Press: **F3**

System: "Press Enter to Move to PT1: -650 0 0 0" is displayed above the "Current Robot Location" message. This message displays the name of the first global point in your program, and the X, Y, Z, and roll coordinates of that point.

You: Press: **F3**

System: "Press Enter to Move to PT2: 0 650 0 0" is displayed, this time with the point name and coordinates of the next global point in your program.

You: Press the enter (<—J) key.

System: A message is displayed to remind you to check the path of the manipulator because the arm is going to move directly to the global point.

DANGER

BE SURE THE PATH IS CLEAR FOR THE MOVE PERFORMED BY THE MANIPULATOR WHEN THE ENTER KEY IS PRESSED ON THE KEYBOARD.

STAND CLEAR OF THE MANIPULATOR WORK ENVELOPE BEFORE THE NEXT STEP. THE SPEED AND FORCE OF THE MANIPULATOR CAN CAUSE SERIOUS INJURY TO PERSONNEL AND DAMAGE TO EQUIPMENT IN ITS PATH.

You: Press the enter (<---I) key.

System: Manipulator arm moves to the location of PT1, as displayed in the "Press Enter to Move to" message. That message is cleared from the screen. The coordinates of PT1 are now displayed after the "Current Robot Location" message.

You: Press: **F4**

System: The "Press Enter to Move to PT1" message is displayed again, because **F4** recalls the previous point in your program. At this point, you may press the Enter key to move, or:

You: Press: **Esc**

System: The X= prompt is displayed on the last line of the screen.

You: Press the enter (<---┘) key.

System: X coordinate that is displayed in the "Press Enter to Move to" message (-650) is pre-filled after the X= prompt, followed by Y. You may enter a coordinate from the keyboard, or:

You: Press the enter (<--J) key.

System: Y coordinate that is displayed in the "Press Enter to Move to" message (0) is pre-filled after the **Y=** prompt, followed by Z. You may enter a coordinate from the keyboard, or:

You: Press the enter (<--J) key.

System: Z coordinate that is displayed in the "Press Enter to Move to" message is pre-filled after the Z= prompt, followed by **R**. You may enter a coordinate from the keyboard, or:

You: Press the enter (<----┘) key.

System: Roll coordinate that is displayed in the "Press Enter to Move to" message (0) is pre-filled after the **R=** prompt.

You: Press the enter (<—J) key.

System: Manipulator arm moves to the point location shown by the **X.=**, **Y=**, **Z=**, and **R=** prompt.That prompt is then cleared from the screen. The coordinates of that point are now displayed after the "Current Robot Location" message.

You: Press: **F3**

System: "Press Enter to Move to" message is displayed again, with the point name and coordinates of the next global point in your program, which is PT2. You may press the Enter key to move, or:

You: Press: **End**

System: "Press Enter to Move to" message is cleared from the screen. The manipulator arm position does not change.

The F3 and F4 keys recall the next global or the preceding global point from the AML/Entry program. For programs with many global points, using the F3 and F4 keys to recall a particular point can take a long time. To speed the process, hitting the shift key in conjunction with the F3 and F4 keys recalls the 10th next or 10th preceding point. This allows you to step through all the global points much faster. An easy way to remember this is that the shift key always causes a larger step. Hitting the shift key with a cursor key causes the manipulator to make a bigger move, likewise hitting the shift key with the F3 or F$ key causes a bigger step between the recalled points.

Note: If the currently recalled point is one of the last global points, than recalling the 10th next point will cycle back from the beginning. Likewise, recalling the 10th previous point when the currently recalled is one of the first causes a cycle back from the last global point.

Controlling Digital Output (DO) from Teach

This exercise returns to teach mode to demonstrate the digital output control feature. This procedure changes digital output point 2 to the **on** state.

Note: Remember, the DO points are numbered starting with 1. For example, on systems with 16 DO points, the points are numbered 1 to 16.

You: Press: **F6**

System: Screen blanks and then displays the teach screen. The manipulator is still in the last position when you exited teach.

You: Press: **F6**

System: Screen displays the digital output control menu.

You: Type a **2** and then press the down arrow cursor key one time.

System: Screen displays the 2 and the cursor moves below the OFF position.

The left and right cursor keys move the cursor between the ON and OFF position of the control screen.

You: Press the right cursor key one time.

System: Cursor moves to the ON position.

You: Press the enter (<—J) key.

System: Digital output menu disappears and the gripper closes.

Note: If the **End** key is pressed while you are displaying the digital output screen, the screen disappears and the information on the screen is not processed. The remainder teach screen is displayed.

The next instruction ends the teach mode and returns the editor display.

You: Press: **End**

System: Screen blanks and then displays the editor with your program.

You: Exit the editor.

Changing Manipulator Arm Mode

Two key functions are supported for the 7545-800S only. Just below the ESC key, the TAB LEFT key (upper shift) and the TAB RIGHT key (lower shift) can be used to change the configuration of the robot without moving to a new point. The user is warned of an upcoming change ofl configuration and can abort the change if desired.

The TAB LEFT and TAB RIGHT keys can be used at any time, regardless of the current robot position. They designate which portion of the work space is to be considered valid by the editor. If the cursor is moved, to a point within the current work mode region, the robot moves to that, point immediately.

Conversely, if the cursor is moved to a point that is in the workspace of the robot but not in the current mode region, the robot does not move. Instead, a POINT OUT OF WORKSPACE message is displayed. For, example, if the TAB LEFT key is pressed, the editor would recognize only' points in the LEFT mode region. Then, only points accessible in LEFT mode can be reached, and all other points would cause a POINT OUT OF WORKSPACE message to be displayed. If this happens, either of the following could occur:

- If the cursor is moved back into the LEFT mode region, the robot immediately moves to the new point.
- If the user presses the TAB RIGHT key, causing the current point to be recognized by the editor, the robot then moves to the new point.

DANGER

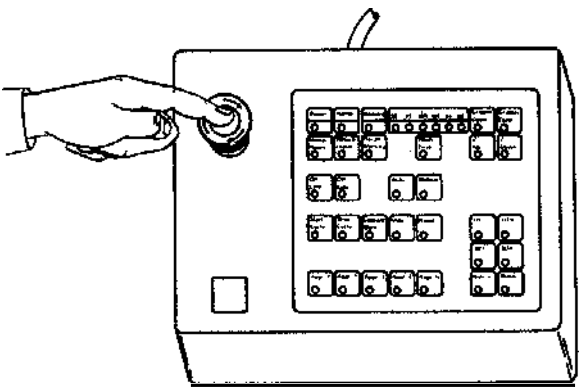
The TAB LEFT and TAB RIGHT keys may invoke motion. ANY TIME THAT THE CURRENT POINT IS IN THE REGION JUST DESIGNATED WITH A TAB LEFT OR A TAB RIGHT KEY, THE ROBOT MOVES.

NOTE THAT THERE IS A * REGION AT THE BOTTOM OF THE WORKSPACE. SHOULD THE USER DESIRE A CHANGE OF MODES IN THIS REGION, THE ROBOT ESSENTIALLY MAKES A FULL CIRCLE AROUND THE WORKSPACE. THUS THE USER SHOULD NOT REQUEST A CHANGE OF MODES IN THE LOWER SECTION OF THE WORKSPACE.

REMOVING POWER AFTER TEACH EXERCISES

The following steps remove the controller power after completing the teach exercises.

You: At the control panel, press the Stop pushbutton down.



System: Manip Power LED goes out.

You: At the controller, set the power switch to 0 .

System: All indicators on the control panel and the controller go out.

CHAPTER 7. OPERATING THE MANUFACTURING SYSTEM

This chapter provides the details for operating the manufacturing system. The chapter includes information relative to transferring application programs from the IBM Personal Computer to the controller and use of the control panel.

Before you begin, read the "Safety" section of this Users' Guide.

The sequence of this chapter is as follows:

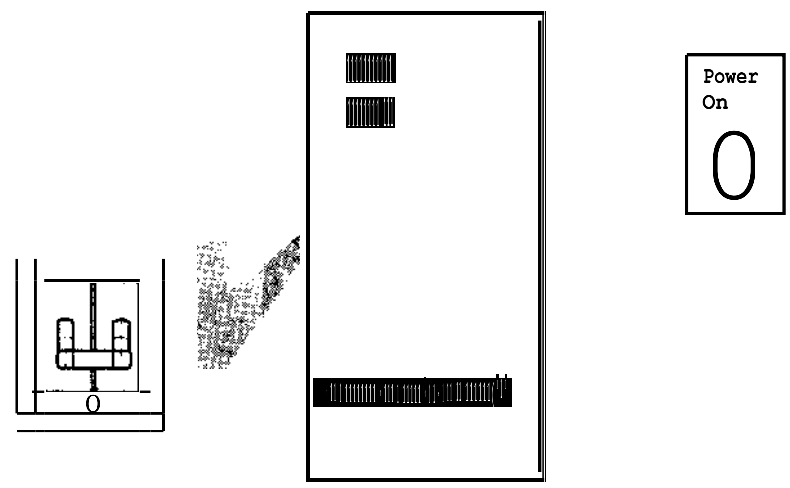
- Controller power switch and power lamp location
- Control panel key and LED indicator description
- System power-on sequence
- Stopping the manipulator and power-off sequences
- Application program transfer
- Testing application programs in manual mode
- Manual operation of the manipulator
  - Control of axis motors
  - Control of Z-axis shaft position
  - Control of gripper
- Automatic operation
  - Starting an application program
  - Recalling an application program
- Clearing error conditions

Note: The procedures in this chapter provide the sequence of pressing keys for the different modes of operation. If you are told to press a key or position a switch that is already in that configuration, skip that instruction.

CONTROLLER

Power On/Off Switch/Circuit Breaker

The power switch is located on the outside of the controller. It is an on/off switch and a circuit breaker. To apply power, place the switch to the 1 (on) position.



Side View

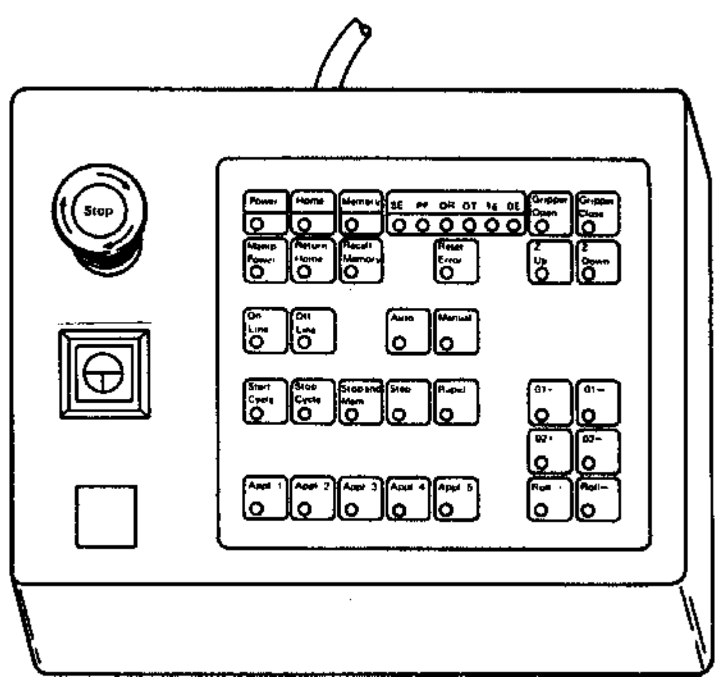
Power On Light

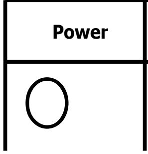
The lamp on the front door lights when the power switch is on.

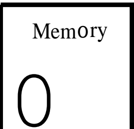
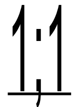
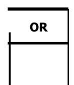
CONTROL PANEL

The control panel contains pressure-sensitive keys and light-emitting diode (LED) indicators. The keys are your interface to the controller.

The LEDs indicate controller and manipulator status. The functions and names of the keys and LEDs are described below:



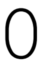
Key/LED	FUNCTION
	When in the down position this latching pushbutton removes power to the manipulator. Manipulator power cannot be energized when this pushbutton is latched in the down position.
	The Power LED lights when the power switch at the controller is in the 1 (on) position.

Key/LED	FUNCTION
	The Home LED lights when the manipulator is in the Home position. At this position, the controller initializes its servo positions through the limit switches.
	The Memory LED lights after you press the Stop and Mem key and when the application, running in automatic mode, has encountered a BREAKPOINT program statement. The LED remains on until you press the Start Cycle key.
EE	The servo error (SE) LED lights when a servo fault is detected. The LED blinks when an encoder fault is detected.
	The power failure (PF) LED lights if an under-voltage condition is detected by the controller.
	The Overrun (OR) LED lights when one of the servoed axes is beyond the maximum working envelope range.
	The over-time (OT) LED lights when a time-limit value for a WAITI in an application program has exceeded its programmed value.
	The transmission error (TE) LED lights when a transmission error occurs between the controller and the IBM Personal Computer or a host computer.

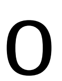
Key/LED	FUNCTION

The data error (DE) LED lights if a data error occurs due to a programming error within an application program.

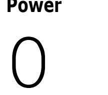
The LED blinks if an error caused by a controller malfunction occurs during execution of an application program.



This key opens the optional gripper when you are operating in manual mode. The LED stays lit only while you are pressing the Gripper Open key.

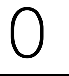


This key closes the optional gripper when you are operating in manual mode. The LED stays lit while the gripper is closed.

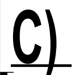


This key energizes the control and servo circuits. The LED Lights when you press the key and remains lit until you press the Stop pushbutton, or remove the power from the controller.

This key causes the controller to return the manipulator to the Home position, where the servo motors and the stepper motor are initialized. When you press the Return Home key, the LED lights and then goes off when the manipulator finds the home position.

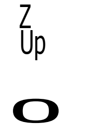


This key instructs the manipulator to resume execution of an application after a BREAKPOINT command. The LED lights when the key is pressed and remains on until the Start Cycle is pressed.



This key resets the error LEDs if the error condition no longer exists.

Key/LED **FUNCTION**



Pressing this key raises the Z-axis shaft when you are using the manual mode. The LED stays lit only while you are pressing the Z Up key.

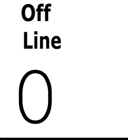
Z
Down

Pressing this key lowers the Z-axis shaft when you are using the manual mode. The LED stays lit only while you are pressing the Z Down key.

On
Line

Pressing this key causes the controller to enter the on-line state. The LED remains lit until the mode is ended. In this mode, a host computer or the IBM Personal Computer can communicate with the controller.

DANGER
THE MANIPULATOR RESPONDS TO THE REMOTE HOST COMMANDS IN THIS MODE. THE MANIPULATOR MAY MOVE WITHOUT WARNING WHEN IN THIS MODE.



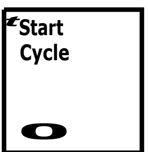
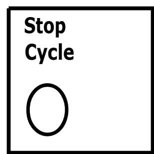
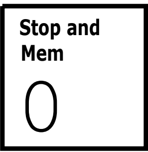
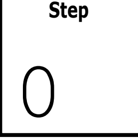
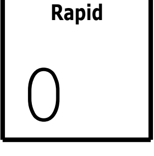
Pressing this key ends the on-line state. Communications with a host or IBM Personal Computer is not possible in this mode. The LED lights when this mode is active.

Auto

This key permits automatic or continuous execution of one of the application programs. The LED lights **when** the key is pressed.

Manual

This key allows operator control of the manipulator, including stepping through an application program one line at a time. The LED lights when this key is pressed.

Key/LED	FUNCTION
	This key starts or resumes an application program. The LED lights when the key is pressed and goes off if the program stops.
	Pressing this key terminates the execution of an application program when the end of the program is reached. The LED lights when the key is pressed and goes off when the program stops or step mode is started.
	This key instructs the controller to stop at a BREAKPOINT statement or at the last line of an executing application program, depending on which one is encountered first. The controller then stores the present state of its memory. This includes instructions, counters, and variables for a subsequent recall by a recall memory command. The LED lights when the key is pressed and goes off when the application program stops. If the Memory LED does not light, the controller has stopped at the end statement of the program and the program must be started without using the Recall Memory key.
	While the controller is in manual or automatic mode, this key instructs the controller to execute a single command of a selected application program and then to wait for the key to be pressed again. The LED lights when the key is pressed.
	Pressing this key in conjunction with a manual move key speeds up the movement of the manipulator during manual mode positioning. The LED stays lit while you press the key.

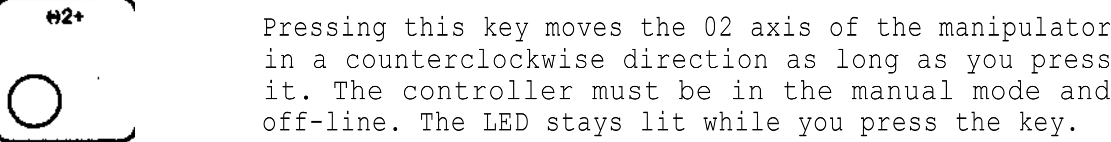
Key/LED **FUNCTION**

θ1+

Pressing this key moves the 01 axis of the manipulator in a counterclockwise direction as long as you hold it. The controller must be in the manual mode and off-line. The LED stays lit while you press the key.

θ1-

Pressing this key moves the 01 axis of the manipulator in a clockwise direction as long as you hold it. The controller must be in the manual mode and off-line. The LED stays lit while you press the key.

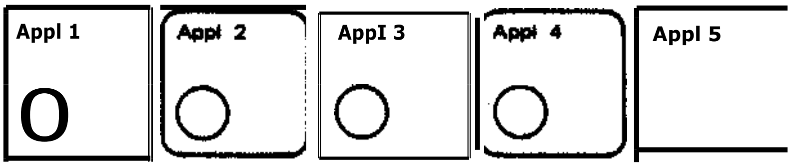


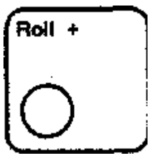
Pressing this key moves the 02 axis of the manipulator in a counterclockwise direction as long as you press it. The controller must be in the manual mode and off-line. The LED stays lit while you press the key.

θ2-

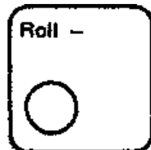
Pressing this key moves the 02 axis of the manipulator in a clockwise direction as long as you press it. The controller must be in the manual mode and off-line. The LED stays lit while you press the key.

Each Appl key selects its respectively stored application for execution when the Start Cycle key is pressed.

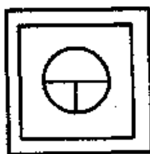




Pressing this key moves the R-axis in a clockwise direction while you press it. The controller must be in manual mode. The LED stays lit when you press the key.



Pressing this key moves the R-axis in a counterclockwise direction while you press the key. The controller must be in the manual mode. The LED stays lit when you press the key.



When an overrun error occurs, pressing and holding this key allows manipulator power to be turned back on with the hitanits Power key. With manipulator power on and while still holding the O.R. Reset key, the axis motion key can be used to move the axis back within its travel range.

The OR Error LED lights when an overrun condition exists on one of the servoed axes and stays lit until the axis causing the overrun is moved to within its travel limit. The Reset Error key can then be used to reset the OR error.

SYSTEM POWER-UP SEQUENCE

The following procedure is the sequence to follow when you power-up the system. Before you apply power, you should be familiar with the Safety section in the front of this document.

Power-up the controller as follows:

You: Check that switches are properly set for the application involved. The application program can control the position zone, and speed, or allow the switches to determine the settings 'for the application. These switches are located on the Motor Control Board on the controller door. Refer to IBM Manufacturing Systems Specifications Guide, 8577126 for the Switch settings on the Motor Control Board.

You: Check that the air regulator (user-supplied) is set at the proper air pressure for the application to be run.

DANGER
MAKE SURE THAT NO ONE IS IN THE WORK AREA OF THE MANIPULATOR WHILE IT IS BEING POWERED UP.

You: Set the power switch to the 1 (on) position.

You: At the control panel, turn the Stop pushbutton clockwise.

System: Stop pushbutton pops up to the on position.

You: Press the **Manip Power** key.

System: Manip Power LED lights.

You: You should allow the controller and manipulator 10 minutes to warm up and stabilize before starting your application. For maximum effectiveness, you should run an exerciser program during the warm-up period.

DANGER
YOU MUST NOT ATTEMPT TO MOVE THE MANIPULATOR BY HAND ONCE THE MANIP POWER LED IS LIT. IF YOU NEED TO MOVE THE MANIPULATOR BY HAND WHEN THE LED IS LIT, PRESS THE STOP PUSHBUTTON TO REMOVE SERVO MOTOR POWER.

DANGER
STAND CLEAR OF THE MANIPULATOR. THE MANIPULATOR MAY MAKE ONE SLOW "RETURN HOME" FOLLOWED BY A FAST "RETURN HOME" AS PART OF AN INITIALIZATION PROCEDURE.

You: Press the **Return Home** key.

System: Manipulator moves to the HOME position and the Return Home LED stays lit until the manipulator reaches the HOME position, lighting the **HOME** LED.

The manipulator is now ready for manual or automatic mode of operation.

MANUAL AND AUTOMATIC STOPPING OF THE MANIPULATOR

Before you begin using the system, be aware of the ways to stop it when using either the automatic or manual modes of operation.

Stop
Pressing this red pushbutton on the control panel instantly stops the manipulator in either automatic mode or manual mode. This is the fastest method of stopping the manipulator during any situation, including emergencies.

Any time it is necessary to use the Stop pushbutton, the controller power should be removed and then reapplied before reactivating the manipulator power.

Stop Cycle
Pressing this key when an application program is executing in the automatic mode stops the program after the last END statement is executed in the program. The operation may be restarted by pressing the Start Cycle key.

Stop and Mem
Pressing this key, when an application program is executing in the automatic mode, stops the program at the first programmed BREAKPOINT statement or at the END statement. The program continues to execute until one of the two statements is encountered. The LED stays lit until the program stops execution.

Step
Pressing the Step key stops automatic execution of the application program and allows each program statement to be executed each time the key is pressed. When the last statement of the program executes, the program cycles to the beginning of the program. Automatic execution resumes when the Start Cycle key is pressed.

POWER-OFF SEQUENCE

This procedure removes power from the manipulator.

You: At the control panel, press the Stop pushbutton.

System: Power LED at the control panel remains on. Depending on the mode of operation, several other LEDs, such as Off Line and Manual, may be lit. The manipulator is disabled.

If all power is to be removed from the system, follow the next instructions. (If the manipulator power is to be restored, return to the power-up sequence, starting at the control panel.)

You: At the controller, set the power switch to the 0 (off) position.

System: Power lamp on the front of the controller goes out and the controller fan stops. All LEDs at the control panel go off.

CONTROLLER STORAGE MANAGEMENT

The controller has the ability to store application programs up to a total of 24000 bytes depending on the configuration of the controller. The controller has the capacity to store up to 5 application programs or the amount of memory available in the controller. The actual number of programs that can be stored depends on the number of bytes of each program.

The controller manages its storage dynamically and utilizes all fragments of user memory.

For example, on a system with 24000 bytes of memory:

1. If all the memory has been cleared,

2. A 13000 byte program is loaded to partition 1, and

3. A 10000 byte program is loaded to partition 2,

4. The controller could still accept one of the following:

    - A 1000 byte program in partition 3, 4, or 5.

    - A 11000 byte program in partition 2.

    - A 14000 byte program in partition 1.

One large program can use the space of all the partitions. If you attempt to load an application program and insufficient storage remains to store the program, the message "Controller memory has been exceeded, unload a partition" is on the IBM Personal Computer. In order to make room for the application, you have to free-up some of the controller memory by unloading other partition(s). The IBM Personal Computer does not prompt you to retry.

Refer to Chapter 8, "Communications" for communication features that allow error message retrieval from the controller.

There are several ways to clear the problem. One of the ways is to evaluate each step of the application program for unnecessary statements. If the program can be shortened, make the changes to the program, compile the program, and then load. The second method involves selecting one or more different storage partitions to be cleared using the **Unload** (Option 4 of the menu) program to each of the selected or all partitions at one time.

The steps for clearing old application programs are:

Your Apply power to the Manufacturing System controller.

You: Access the AML/Entry menu.

You: Connect the RS-232 communication cable between the Personal Computer and the Manufacturing System controller.

You: At the Manufacturing System Control Panel:
Press the On Line *key*

System: Off line LED goes off and the On line LED lights.

You: At the Personal Computer:
Type 4 and press the enter) key.

System: Screen displays: Loading .....

System: Screen displays: Enter partition number to be unloaded

You: Type a single partition number to clear that partition or type all to clear all partitions.

You: Press the enter (<----┘) key.

System: When the unload is complete, the screen displays:
Press any key.

You: Press any key.

System: Screen displays the menu.

You: At the Personal Computer, attempt to load your new application program to the selected partition number.

COMPILE AND LOAD AN APPLICATION PROGRAM

The following procedure provides the steps for converting a program at the Personal Computer and then loading it into a storage partition in the controller. The storage partition numbers correspond to the Appl 1 through Appl 5 keys. This procedure assumes that you are using the menu. You may want to check that your configuration utility reflects the correct system type (units and communication port). Refer to Chapter 2, "Getting Started on the IBM Personal Computer" for details.

Bringing Up the AMLJE Menu On a Standard PC

If you have a standard PC with no fixed disk, perform the following steps to bring up the AML/E System Menu.

You: Open drive A (left drive) on the Personal Computer.

You: Insert your AML/E System Diskette Number 1 into drive A.
Insert your AML/E System Diskette Number 2 into drive B if your PC has a B drive.

You: Set System Unit power switch to ON.

You: When requested, type in the date in the format shown on the screen.

You: Press the enter (<----') key.

You: When requested, type in the time in the format shown on the screen.

You: Press the enter) key.

System: AML/E menu is displayed:

Bringing Up the AML/E Menu On a PC With a Fixed Disk

If you have a fixed disk on your PC, perform the following steps to bring up the AML/E System Menu.

You: Set System Unit power switch to ON and let AUTOEXEC.BAT run.

You: If AUTOEXEC.BAT does not prompt for the date and time, enter the commands **date** and **time** and enter the current date and time in the format required.

You: Change to the directory that contains the AML/Entry system using the **chdir** command.

You: Enter the command **menu**.
The last two steps are done automatically if the AUTOEXEC.BAT file created by the Autoinit procedure is used (and the AML/E system is installed on the root directory).

System: AML/E menu is displayed:

Bringing Up the AML/E Menu On a PC/AT

If you have a PC/AT with a fixed disk, follow the instructions of the previous section. If you have a PC/AT with a high-density diskette drive, then follow these instructions.

You: Open drive A (left drive) on the Personal Computer.

You: Insert your AML/E System Diskette Number 1 into drive A.

You: Set System Unit power switch to ON.

You: When requested, type in the date in the format shown on the screen.

You: Press the enter) key.

You: When requested, type in the time in the format shown on the screen.

You: Press the enter (<---) key.

System: AML/E menu is displayed:

After these instructions are performed, the following menu appears.

```
Select a function:

0. Return to DOS.
1. Edit/Teach a program.
2. Compile a program.
3. Load a program to the controller.
4. Unload a controller program.
5. Set system configuration.
6. Set program name and options.
7. Communicate with controller.
8. Generate Cross Reference Listing.

Enter Option ===>
```

Note: In the steps that follow, the AML/E programs COMPILER.EXE and COMAID.EXE are called by the AML/E Menu. For a dual-sided diskette drive system without a fixed disk, the Compiler is located on the AML/E System Diskette Number 2, and Comaid is located on the AML/E System Diskette Number 1. If the menu displays the message "UNABLE TO READ COMPILER.EXE" or "UNABLE TO READ COMAID.EXE", then the wrong diskette is in the A drive and the correct diskette must be inserted into the A drive. If your PC has an A and B drive, then each of the AML/E System Diskettes can be placed in its own drive.

You: Connect the RS-232 communication cable between the Personal Computer and the Manufacturing System controller.

You: Power on the Manufacturing System and place in "On Line" mode.

You: At the menu, select function 2.

```
Select a function:

0. Return to DOS.
1. Edit/Teach a program.
2. Compile a program.
3. Load a program to the controller.
4. Unload a controller program.
5. Set system configuration.
6. Set program name and options.
7. Communicate with controller.
8. Generate Cross Reference Listing.

Enter Option ===>2
```

System: Screen displays: Loading .....

You: When the screen displays "Enter source file specification:", type the drive where the source program is located followed by a colon and the filename of the program. Then press the enter (<---I) key.

Example:

--> b:programl

In the example, programl is an AML/Entry program located on a diskette in drive B.

You: Answer the Error Report Hardcopy, Generate Listing File, and Generate Symbol Table prompts.

If you are not familiar with these prompts, refer to Chapter 2, "Getting Started on the IBM Personal Computer."

System: After answering the prompts, as the program is compiling, the screen displays the following sequence:

```
Creating Program for Machine Type XXX
Object Module Size =

Reading Input File

Converting AML/E Program

Writing .ASC File

Successful Compilation
Press any Key to continue                    LINE:xxx
```

You: If the On line light on the Manufacturing System control panel is not on, press the On line key.

You: Press any key on the Personal Computer to continue.

You: At the AML/Entry menu select function 3.

```
      Select a function:

   0. Return to DOS.
   1. Edit/Teach a program.
   2. Compile a program.
   3. Load a program to the controller.
   4. Unload a controller program.
   5. Set system configuration..
   6. Set program name and options.
   7. Communicate with controller.
   8. Generate Cross Reference Listing.

      Enter Option ===>3
```

System: Screen displays: Loading .....

You: When the screen displays "Output Filename ==>" type the drive where the compiled program is located followed by a colon and the filename of the program then press the enter (<---┘) key.

Example:

--> **b:programl**

In the example, programl is a compiled AML/Entry program located on a diskette in drive B.

System: Screen asks for the destination partition (Application Number):

Destination Partition ==>

You: Type the number 1, 2, 3, 4, or 5 for the partition where the program is to reside in the Manufacturing System controller storage.

You: Press the enter) key.

System: Screen displays the program lines (Line xxx) as they are being transmitted to the Manufacturing System controller.

System: When transmission is completed, the screen displays:

```
    Successful Transmission to Partition x

Strike any key to continue...
```

System: Menu screen is redisplayed when a key is pressed.

TESTING APPLICATION PROGRAMS IN MANUAL MODE

After you have developed an application program and transferred it to the controller, you may want to step through the execution of the program one instruction at a time. Stepping through your program allows you to observe that it performs each function properly and that the points you have defined are correct.

If you have not already done so, power-up the system and allow the required 10-minute warm-up time. The following indicators must be lit when you start this procedure:

- Power
- Manip Power
- Home

To test the application programs, follow the next instructions:

You: At the control panel, press the **Off Line** key.

System: Off Line LED lights.

You: Press the **Manual** key.

System: Manual LED lights.

You: Press the appropriate **Appl** key corresponding to the partition number for the program stored in the controller.

System: One of the Appl LEDs lights.

You: Press the **Step** key repeatedly to execute instructions.

System: Each time you press the Step key, a program instruction is executed. The Step LED lights when the key is pressed.

Note: Some instructions (e.g., SUBRs, LABELs) contain several internal coded instructions; you need to press the Step key several times to step through these instructions.

If you press the Step key after the last line of the program has executed, the program restarts at the first line.

MANUAL OPERATION OF THE MANIPULATOR

Use the following procedures to position any axis by means of the control panel in manual mode. You may want to use manual mode while you are checking the height and position for feeders and fixtures, and for the gripper's (or other end-of-arm effector) ability to hold objects properly.

Unlike the automatic mode, the manipulator need not be in the Home position to use the manual control keys.

You: At the control panel, press the Manual key.

System: Manual LED lights.

You: Press the Off Line key.

System: Off Line LED lights.

The Manufacturing System is now ready to be used in manual mode.

MANUAL MODE CONTROL OF AXIS MOTORS, Z-AXIS, AND GRIPPER

Eight keys control the manual movement of the Cl axis, the 02 axis, the Z axis, and the R axis. In addition, the speed for any of the axes may be increased by simultaneously using the Rapid key with any of the axis keys.

When using the manual control keys for moving the 01 and 02 axes or a servoed Z axis, it is possible to move the manipulator beyond the work envelope. If you do move the manipulator outside the work envelope, the over-run (OR) LED on the control panel lights. The manipulator stops its movement when the over-run condition is detected. To clear the condition, physically move the manipulator back into the work envelope and press the Reset key to clear the OR indicator or if your system has the O.R. Reset key, it can be used to clear the condition.

> **Note:** Remember the caution in the beginning of Chapter 6, "Using the AML/Entry Teach Mode." If this occurs while in the Teach mode of the AML/E editor, then the manipulator will be moved home by the controller before being moved to the current location specified by the Teach screen.

You should have set the control panel for manual mode (see the first two steps of the previous section). Press the desired **Z_Up** or Z-Down key. Refer to "Control Panel," earlier in this chapter, for a description of the keys.

The optional gripper motion is driven by air pressure. To control the motion of the gripper in manual mode, you must have the air supply on and the Manufacturing System power on. You must have the Manual key active for manual control of the gripper. Refer to the "Control Panel," in this chapter, for a description of the keys.

AUTOMATIC OPERATION

If you are not familiar with stopping the manipulator in the automatic mode, refer to "Stopping the Manipulator", in this chapter, which describes these four keys:

- Stop
- Cycle Stop
- Stop and Mem
- Step

CAUTION
Clear the work area before starting. Review "Safety" in the front of this document.

STARTING AN APPLICATION PROGRAM IN AUTOMATIC MODE

This procedure applies to starting an application from the control panel. Starting an application from a host is described in Chapter 8, "Communications."

The following conditions must be met before you enter the automatic mode:

- The application program has been transferred to the controller.
- The following LEDs on the control panel are lit:
  - Power
  - Manip Power
  - Home
  - Off line
- Allow the required 10-minute warm-up.

The following procedure starts the system in the automatic mode from the control panel. Remote start-up is described in Chapter 8, "Communications."

You: At the control panel, press the **Auto** key.

System: Auto LED lights.

You: Press the appropriate **Appl** key where the application program is stored.

System: LED for the selected APPL key lights.

You: Press the **Start Cycle** key.

System: Application program starts.

RESUMING AN APPLICATION PROGRAM FROM A BREAKPOINT

Use this procedure if you stopped an application program using the Stop and Mem feature and the Memory LED was lit. The Stop and Mem feature allows you to power-off the controller and then resume the application without starting over. If the Memory LED was not lit, the program stopped at the last line of the application program and you need to use the "Starting an Application Program in Automatic Mode" section.

If power was removed, you must restore power using the system Power-up sequence. The Memory LED at the control box goes on when power is applied.

The following conditions must exist when you wish to resume a program that was stopped using the Stop and Mem feature:

- Power-up and warm-up complete
- The following LEDs must be on at the control panel:
  - Power
  - Manip Power
  - Home
  - Memory

CAUTION
Clear the work area of obstacles between the home position and the next point of a move execution.

You: At the control panel, press the **Auto** key.

System: Auto LED lights.

You: Press the **Appl** number that is to be recalled.
This must be the application in which the breakpoint occurred.

System: Appl LED is lit.

You: Press the **Recall Memory** key.

System: Recall Memory LED lights.

You: Press the **Start Cycle** key.

System: Application program starts execution. The Memory LED and the Recall Memory LED go off.

CLEARING ERROR CONDITIONS

This section describes how to clear control panel error LED indicators.

If the problems do not clear after following these procedures, refer to the IBM Manufacturing System Maintenance Information for your system.

DE

The DE (Data Error) LED lights when the controller detects an error in the stored application program.

Press the **Reset** key to clear the indication.

> **Note:** If the Reset key does not clear the DE LED, turn off at the controller breaker for at least 10 seconds. Turn on the circuit breaker. If the error occurred during start up, your problem may be the result of the method used to start the application. Refer to the "Automatic Operation" section for the correct sequence of starting an application program.

Blinking DE (Data Error) LED - To clear a blinking DE LED, the application program may have to be reloaded.

Refer to Chapter 8, "Communications" for details of reading error condition at the controller if the condition repeats.

PF

The PF LED lights when the controller has detected an interruption or drop in the line power.

At the controller, set the power switch to the 0 (off) position. Wait at least 10 seconds and then set the switch to the I (on) position.

OR

The OR LED lights when the manipulator has moved out of the work envelope.

Physically move the manipulator el or 02 axis back into the work envelope. Press the **Reset** key; then press the **Manip Power** key.

If your manipulator has an **O.R. Reset:**

You: Press and hold the O.R. Reset key.

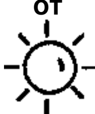
You: Press the Mardi) Power key.

You: Return the axis causing the overrun back within its travel limits by using its axis key. The O.R. Reset light goes off when the axis is back within the work envelope.

You: Press the Reset Error key.

Note: The 01 or 02 axis can still be returned back into the work envelope by hand without using this procedure.

Refer to Chapter 8, "Communications" for details on reading error condition at the controller if the condition repeats.

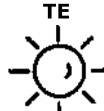


The OT LED lights when a timer function detects an overtime condition. One of the instructions in your program is waiting for a DI switch condition that does not occur within the time limit specified.

Refer to Chapter 8, "Communications" for details on reading error condition at the controller if the condition repeats.

The SE LED lights when a servo error has occurred on any axis. The SE LED flashes when a encoder error has occurred on any axis.

Press Reset to clear the indication. If the indicator stays lit or continues to flash, refer to the IBM Manufacturing System Maintenance Information for your system.



The TE LED lights when a transmission error has occurred between the Personal Computer and the controller.

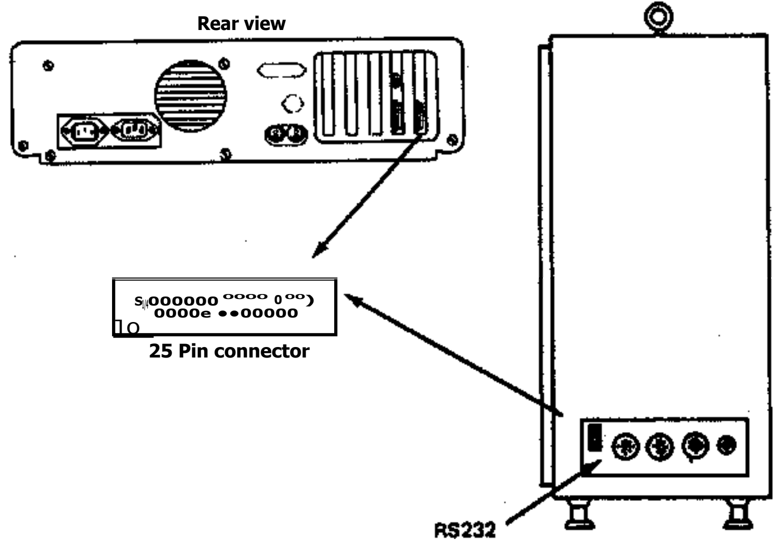
If the problem occurred during a program transfer, press the Reset key to clear the indication. Refer to "Controller Memory Management" in this chapter for possible solutions.

Refer to the Chapter 8, "Communications" for details on reading error condition at the controller if the condition repeats.

If the condition occurred during the change from On Line to Off Line, press the Reset key to clear the indication. This is not a program transmission problem.

Controller Communications Connector

The 25-pin connector on the controller with the label **Cl RS232C** is the connector for the communications interface. Some pins in the connector have special meanings for RS-232 communications and some have special meaning for RS-422. The cable you connect activates the correct interface. For example, the RS-422 cable connects pin 18 in the controller connector to ground, which tells the controller to use the RS-422 interface. For the cable wiring diagrams, refer to Appendix F, "Communications Cable Wiring Diagrams."



Communication Startup Sequence

1. The controller On-Line LED is lit.
2. The host raises "data terminal ready."
3. The host waits for the controller to raise "data set ready"; this should happen immediately.
4. Startup is complete.

After startup is complete, the operator control panel only responds to:

- Manipulator Power On
- Reset Error
- Off-line (This drops the controller DTR at the end of a transaction or immediately if no sequence is in process)
- Emergency power off

The controller side of the cable should have RTS wrapped back to CTS

COMMUNICATION CAPABILITIES

Because the manipulator is controlled by a special controller, a communications interface exists so that the IBM Personal Computer or the IBM Industrial Computer can talk with the controller. The IBM PC or Industrial Computer is called the "host" computer. The "host" can initiate a wide variety of communications requests to which the controller responds. Downloading a compiled AML/Entry application, as discussed in Chapter 7, "Operating the Manufacturing System," is an example of a communications request. All of these requests can be performed by the user from his IBM terminal by using Comaid (see "COMAID" on page 8-14). The AML/Entry communications requests include:

- Download Request

  This request downloads a .ASC file to any of the five controller partitions. The .ASC file must come from the AML/Entry compiler, as this contains the format of the records needed by the communications request. Attempting to download a .AML file will cause an error.

- Unload Request

  This request unloads a controller partition. It is possible to download a .ASC file and overwrite an existing file within a partition. By performing an unload request, the application key on the control panel corresponding to the unloaded partition will be deactivated, as will the Execute communications request that selects the unloaded partition.

- Read Request (R record protocol)

  An often used request, this allows the host to read values from the controller. Some of the read requests read values pertaining to a running application; others read values pertaining to the state of the manipulator. The AML/Entry Read requests include:

  - Read Machine Status -- This reads error information when an error has occurred in a running application (e.g., a data error or AML/Entry error).

  - Read Reject Status -- This reads the error code for a communications error. When a communications error occurs because the host has requested an invalid communications request (for example an Unload request when the given partition does not contain an AML/Entry application), the controller will send an Eot (reject data) signal to the host. This Read request will give further information as to why the Eot was sent by the controller.

  - Read Micro-Code Level and Machine Type -- This request returns three values. The first two values contain the major and minor microcode level numbers, the third value contains the machine type.

Read Robot Parameter Table -- This request returns twelve values which characterize the attached manipulator. See "The Other Read Transactions" on page 8-43 for a list of the twelve values.

- Read Instruction Address -- This request returns the current hardware instruction address in the current application. By generating a listing file when the AML/Entry program is compiled, it is then possible to determine where in the original AML/Entry program the controller actually is.

  Note: The AML/Entry compiler uses an internal subroutine that performs any manipulator motion, regardless of the actual AML/Entry command that invokes the motion. Reading the instruction address will return the address of the internal subroutine when this request is performed during or immediately after manipulator motion.

- Read Digital I/O -- This request returns the state of all the digital I/O attached to the manipulator system.

- Read All Program Variables -- This request returns the values of all the variables in controller memory for the current application.

- Read Current Manipulator Position -- This request returns the location of the manipulator in terms of the motor encoder pulses. The user must convert these to the actual X, Y, Z, and Roll coordinates.

- Read Specific Program Variables -- This request also reads the values of variables in controller memory for the current application. This request, however, allows the user to specify which variables to read. For large applications, this is far more efficient than reading all the program variables.

• Execute Request (X record protocol)

  This request allows the user to control the state of the manipulator via communications. Virtually all of the keys on the control panel have a corresponding Execute request. Thus rather then having to require an operator to strike keys on the control panel, this can be done remotely via communications. A list of the Execute requests can be found in "X - Transmit Execute Command" on. page 8-16.

• Control Request (C record protocol)

  This request gives the user the ability to control the current application. Whereas Execute requests control, the state of the manipulator,. Control requests require a current application. The available Control requests include:

  - Suspend Program Execution -- This request suspends program execution as soon as possible. Any motion must complete before this Control request is honored.

- Restart Program Execution -- This request restarts program execution after it has been suspended with the preceding Control request.

- Execute Until Next Terminator -- This Control request allows the user to single step through the application. This is similar to the Step Execute request.

- Set Debug Breakpoint -- This request allows the user to set up a breakpoint in the existing application. The address given should be obtained from a listing that is generated by the AML/Entry Compiler. Once a breakpoint is set, the host is noted of its occurrence by a controller initiated communication transaction. In order for this to be properly received, data drive mode must be entered. See "Data Drive Mode" on page 8-8 for a discussion of data drive mode.

- Reset Application -- This request resets the controller. It clears all errors, resets all communication timers, deselects the current application, and places the controller in manual mode. Sometimes this is the only way to stop an application from running.

- Change Variables in Memory -- This request allows the host to asynchronously change any of the variables in the controller memory. Points, regions, pallets, counters, and groups can all be changed. The user must use the XREF program to determine the variable numbers in controller memory each AML/Entry variable occupies.

• Teach Requests (T record protocol)

  This request allows the user to move the manipulator without compiling, downloading, and starting an AML/Entry program. The user may move the manipulator to a new point, set the LINEAR, PAYLOAD, and ZONE parameters for any future motion started by a Teach request, turn on or off any digital output, or switch the arm configuration from right to left or vice versa (for the 7545-800S only).

  Warning: Using Teach requests and the control panel at the same time is likely to cause the manipulator to move to home unexpectedly. Whenever a Teach request is sent, if any of the axis movement keys has been touched, the manipulator will move home before honoring the Teach request.

• Enable/Disable Data Drive Mode

  AML/Entry provides a feature called controller initiated communications. This feature makes the controller the initiator of any communications transaction. For example, when an existing application encounters a GET or PUT statement, a controller initiated data drive transaction begins. Another example of a controller initiated data drive transaction is when a debug breakpoint is encountered. When either of these three events

occurs, the controller initiates a communications transaction by sending a message to the host.

With the current communications protocol, only the host or controller can be enabled as the initiator of communications. When the system is powered on, the IBM Personal Computer or Industrial Computer is made the initiator. This means that all of the communication requests described in the preceding list can be performed. But once data drive mode is enabled, the controller is made the initiator of any requests. If the host tries a communications request when the controller is the initiator, an error will occur (the controller will send an Eot back to the host). If the controller reaches one of the three situations that cause a controller initiated request to begin and the controller is not the initiator, then a wait state is entered. Once data drive mode is enabled, the controller initiated communications request will occur. The host can turn off data drive mode at any time, enabling itself as the initiator.

The important thing to remember is that only the host or the controller is enabled as the initiator. Originally the host is enabled. To perform controller initiated data drive, the controller must be enabled. While the controller is enabled, all requests by the host are rejected until data drive mode is disabled, enabling the host as the initiator.

Most users will not need to learn the AML/Entry communications protocol because AML/Entry Version 4 provides two programs that give the user two different levels of interface to the communications transactions. Comaid provides the user a menu-driven interface to the above communication requests. Comaid may be invoked by selecting option 7 from the AML/Entry menu. AMLECOMM provides the user a "device-driver" level of interface. AMLECOMM consists of BASIC modules which may be imbedded into a user cell controller applications. By calling the BASIC modules, any of the communication requests can be performed. For the more curious user, Appendix H, "Advanced Communications" contains a technical description of the communications protocol, along with sample transactions. This appendix should only be read by users who cannot use AMLECOMM and must write their own program to perform the host side of the communications protocol.

DATA DRIVE MODE

As just discussed, when the system is first powered up, the IBM PC, or host, is the initiator of all communications requests. In order for the controller to be the initiator of communications requests, data drive mode must first be enabled. When data drive mode is enabled, only the controller can initiate communications requests. It will do so when a GET, PUT, or debug breakpoint is encountered. If any of these three events occurs and the controller is not the initiator of communications requests, then the controller enters a wait state. In terms of the communications protocol, these states are called the Xon state, Xoff state, Wxo (Waiting for Xoff) state, and the Xto (Xoff time out state). These are discussed in greater detail in this section.

Controller States

The controller is considered to be in one of two communication states, the Xon state or the Xoff state. The Xoff state has two sub-states, Xoff Time-Out (Xto) and Waiting For Xon (Wxo). The Xon state is entered when the host sends an Xon signal to the controller. This signal tells the controller that it may initiate a communications request. The Xoff state is the initial state in which the host is the initiator of communications requests. The two Xoff substates occur when the controller hits a GET, PUT, or debug breakpoint and the host is still the initiator of communications requests. The Wxo mode is entered. Then, unless an Xon signal is received within 30 seconds, the Xto error state is entered.

Xon State

In the Xon state, the controller is allowed to initiate communications. When in the Xon state, the controller returns an Eot with the status set to code A1 for all host-initiated communication. While in the Xon state, should a GET, PUT, or debug breakpoint be encountered by the running application, a communications request is initiated by the controller. The actual data sent across the communications line is discussed in Appendix H, "Advanced Communications." The details of this can be avoided if AMLECOMM is used. See the discussion of AMLECOMM later in this chapter.

Xoff State

In the Xoff state, the controller cannot initiate communications. It responds only to host communication requests.

Waiting for Xon (Wxo) State

The controller enters the Wxo state if it needs to initiate a communications transaction (e.g., GET, PUT, or debug breakpoint encountered) but cannot, since it is in an Xoff state. The Wxo state waits 30 seconds for an Xon.

Xoff Time-Out (Xto) State

The controller enters the Xoff Time-Out state when the Wxo state times out (when the 30 second timer expires). Although this is an error state, host communications are allowed.

Transitions Between States

There are additional rules governing the transition from one state to another.

Simple State Transitions

Simple state transitions govern the transitions between the Xon and Xoff state. All other state transitions are governed by the new rules for complex state transitions.

When it is powered on, the controller is in the Xoff state. It only enters the Xon state when it receives an Xon from the host. If the controller receives an Xon while in the Xon state, it ignores the received Xon. When it receives an Xoff from the host, the controller enters the Xoff state. When the controller is in the Xon state, it can begin a communications request at any time. It begins a request by sending a special record called a D record. Should the host send an Xoff signal at the same time that a D record is sent by the controller, or should the host send an Xoff instead of acknowledging the D record according to the requested protocol (see Appendix H, "Advanced Communications"), then the Xoff is honored by the controller and the Wxo mode is entered. When the host sends the Xon, provided this is done within 30 seconds so the Xto mode is not entered, the controller resends the D record that begins the controller initiated communications request.

> **Note:** The data drive scenario just discussed requires microcode level 15 or 16. If a lower level of microcode is installed and an Xoff is sent after the controller sends a D record indicating the start of a communications request, then manipulator power will be lost. The error code returned would be 48 (Hex 30), Communications Not Established (unable to GET/PUT). To avoid this, the host application should only-send an Xoff when it knows that the controller will not attempt to simultaneously begin a communications request.

If an Xoff is received while in the Xoff state, it is ignored. If the controller is in the Xon state, and the operator presses the "off-line" key, the controller enters the Xoff state, It automatically transmits

an Eot when this event occurs. When the "off-line" key is again pressed, the Eot code is changed to 'AA'.

Complex State Transitions

When the controller encounters a situation where it needs to initiate communications (executes a GET), but the controller is in the Xoff state, it is referred to as "frustrated initiation". In this case, the following rules apply:

1. The controller waits up to 30 seconds for an Xon to be received. This is the Wxo (waiting for an Xon) state

   - If an Xon is received, the controller enters the Xon state and initiates its communication.

   - If any other communication is received, the controller may respond to it. There are two classes of communication requests, allowed and not allowed. If you attempt to perform an operation that is not allowed, an Eat error with an 'A2' code (communication not allowed) results. Transactions allowed during this state are:

     All Read (R) Record operations

     Execute (X) Record operation
     13 - Reset error

     Control (C) Record operations
     10 - Install debug address stop
     20 - Reset controller
     80 - Change controller variables

   Note: The thirty-second timer is reset by performing these host communication operations.

2. If the 30 seconds expires, the controller enters the Xto (Xoff Time-Out) state and the transmission error (TE) lamp comes on.

The following rules govern the Xto (Xoff Time-Out) state.

1. If an Xon is received, the controller returns an Eot and the Eot status is set to AO (Xto mode).

2. If any other communication is received, the controller may respond to it. There are two classes of communication requests, allowed and not allowed. If you attempt to perform an operation that is not allowed an Eot error with an A2 op-code (communication not allowed) results. Transactions allowed during this state are:

All Read (R) Record operations

Execute (X) Record operation
    13 - Reset error

Control (C) Record operations
    10 - Install debug address stop
    20 - Reset controller
    80 - Change controller variables

3. If the error is reset (either from the control panel or by using communications) and enters the Xoff Error Reset state, the controller restarts the thirty-second timer and waits for an Xon. The controller has re-entered the Wxo (waiting for Xon) state.

Comaid provides a very friendly data drive interface. The followin application uses a GET statement to fetch two points that the manipulator moves between, and a PUT statement to send to the host counter that indicates the number of moves performed by the manipulator. The data drive is only performed if the counter DO_DATA_DR is set to 1. Initially this counter is set to 0, so the GET and PUT instructions are not performed. By using Comaid, change variable 34 (which corresponds to DO_DATA_DR) to 1. Once this is done, the GET and PUT statements will be performed. The PUT statement sends a value for variable number 35 (which corresponds to the counter NUM_MOVES). The GET statement requests values for variables 36-43. The first four values correspond to the X, Y, Z, and Roll locations for the first point. The second four values correspond to the X, Y, Z, and Roll locations for the second point.

```
-- Sample program to illustrate Data Drive

DO_ DATA DR: STATIC COUNTER;      -- When 0, the GET and PUT are not
                                  -- performed. When 1, they are.
NUM_MOVES: STATIC COUNTER;        -- Counts the number of PMOVES
TWO_POINTS: STATIC GROUP(PT(400 ,400,0,0) , PT(-400,400,0,0));
                                  -- Robot moves between these points
MAIN:SUBR;
   PMOVE(TWO POINTS(1));          -- Move to first point
   SETC(NUM_MOV S,NUM_MOVES+1);   -- Increment NUMMOVES
   TESTC(DO_DATA DR,O,NOPUT);     -- Don't do PUT if DO DATA_DR=0
     PUT(NUM_MOVE );              -- Send NUM_MOVES to host
NOPUT:
   PMOVE(TWO POINTS(2));          -- Move to second point
   SETC(NUM_MOV S,NUM_MOVES+1);   -- Increment NUM_MOVES again
   TESTC(DO_DATA DR,O,NOGET);     -- Don't do GET if DO DATA_DR=0
     GET(TWO_POINTS);             -- Fetch two new points
NOGET:
   END;                           -- End of application
```

To run the application and at the same time better understand controller initiated data drive, follow these steps:

1. Enter the program into a file.

2. Compile the program with the AML/Entry Compiler. Request the symbol table file so that the XREF program can be used to generate a cross reference listing of the program variables by their numbers.

3. Select option 8 from the main menu to run the XREF program. Verify that the variable numbers indeed match those listed above.

4. Choose option 4 to download the compiled program to partition 1.

5. Start the application. See Chapter 7, "Operating the Manufacturing System." Let the application run a minute or so to allow the counter NUM MOVES to accumulate.

6. Choose option,7 from the AML/Entry menu to invoke Comaid.

7. Choose option D to enter the data drive menu and select option 99 to prepare for a GET.

8. Enter values for variables 36-43, which will correspond to two new points between which the manipulator will move.

9. Exit the data drive menu and return to the communications menu.

10. Choose option C to enter the Control Commands menu.

11. Choose option 80 to change variable 34 to a value other than O.

12. Note that as soon as this is done, the manipulator stops moving. This is because either the GET or PUT statement has now been encountered and the controller wants to perform a controller initiated communications request. The controller is now in the Wxo state.

13. Choose option D to enter the data drive menu.

14. Select option 02 to send an Xon signal to the controller. Once this is done, either the GET or PUT command will be performed, either causing the values you entered for variables 36-43 to be sent to the controller or causing the value for variable 35 to be sent to the host. (Choose option C when Comaid prompts you since you have already prepared your data to be sent to the controller).

15. Comaid will now loop indefinitely, printing the requests from the controller as they arrive. Notice how the GET and PUT commands are flashed on the screen as they arrive from the controller. When a key is struck, an Xoff is sent to the controller and data drive mode is exited. This reinstates the host as the initiator of any communications requests.

16. If the above steps were done correctly, the manipulator will have been moving between the two new points you specified when you prepared for the GET, but when data drive mode was exited the manipulator stopped. This is because the controller is in the Wxo state, either wanting to perform the GET or PUT instruction.

17. If you wait 30 seconds, the TE light will come on. This is because the Xto mode has been entered.

18. Now choose option C followed by option 80. Change variable 34 (DO_DATA_DR) back to O. Thus controller initiated data drive requests will be disabled.

19. Now choose option X followed by option 13. This Execute comman resets the Xto error and returns the controller to the Wxo state.

20. Now enable data drive once again by choosing option D followed by option 02. This sends an Xon signal to the controller. The Wxo mode is exited and the data drive command that is blocking the controller is performed.

21. Note now that as Comaid loops indefinitely, the GET and PUT messages are not flashing on the screen. This is because DO_DATA_DR has been set back to 0, disabling the data drive.

22. Strike a key at any time to exit data drive mode and reinstate the host as the initiator of all communications requests.

23. If you choose option D from the communications menu, followed by option 98, you can display variable number 35 sent from the controller. The value will be the number of moves (NUM_MOVES) that had been performed when the last PUT command was encountered.

24. The above steps can be repeated as often as desired.

COMA ID

Comaid is the 7th option from the AML/E Main Menu. It is a program that provides the user a menu driven communications interface to the controller. The main menu is shown in the following figure.

```
                    COMMUNICATIONS MENU

    Choose one of the following:

          P - Display last 51 communication transactions
          L - Load a Program to the Controller
          U - Unload Controller Partition
          R - Transmit Read Command
          X - Transmit Execute Command
          D - Controller Initiated Communications
          C - Control Executing Program
          T - Transmit Teach Command
          F - Execute a Command File
          E - End Program

    Select an option _
```

To perform a communication request with the controller, the user must first learn the submenus. For instance, to send the manipulator home requires an Execute Command, to turn on a DO port requires a Teach Command, and to set a debug breakpoint requires a Control Command. In the sections that follow, each of the options of this primary menu will be covered in greater detail.

P Display the last 51 communication transactions

As the user performs communications requests with Comaid, information is being sent between the controller and the personal computer. As information is sent or received, it is placed in a "circular" communications buffer capable of holding 51 elements. That is, when the 51st element of the buffer is filled, the first element is overwritten with the next request, then the second element, etc. The user thus can ***always*** look at the last 51 transactions. Seventeen transactions are displayed at a time, until all 51 have been displayed.

The P option also allows the communications transactions to be printed on the printer. After the P option is selected, you will be asked if you would like the communication transactions also printed on the printer. If you select yes (option "Y") and the printer is not functional for any reason, then the message "Printer not available - hardcopy request cancelled" will appear on the screen.

Note: Sometimes, there will be as long as a 30-second delay before this message appears.

To ease the viewing of the communication transactions, Comaid places line of dashes (----) in the buffer every time a new request is begun A line of dashes is also placed into the buffer every time a new dat drive request is initiated by the controller when data drive mode enabled.

L Load a Program to the Controller

This option allows the user to download a compiled AML/E program to th controller. The user is prompted for a filename and a partition. Th filename must have a .ASC extension. Comaid will assume this as th default file extension if none is given. A file may not be downloade with an extension other than .ASC. Comaid does not check that th compiled program was indeed compiled for the manipulator that attached. The partition must be 1, 2, 3, 4, or 5.

If the specified file does not exist, the user is prompted for anothe file. A carriage return will abort the load request.

As the file is downloaded, the current line number of the .ASC file is printed on the screen.

U _ Unload Controller Partition

This option allows the user to unload a controller partition. The user must enter either a valid partition number (1, 2, 3, 4, or 5), or "ALL". Selecting a single partition number will cause only that partition to be unloaded. Selecting "ALL" will cause all 5 partitions to be unloaded.

R _ Transmit Read Command

This option allows the user to read different values from the controller. When this option is selected, a submenu immediately appears. The options available are:

```
            VALID READ COMMANDS ARE:

    00 - Return to the Communications Menu
    01 - Read machine status
    02 - Read reject status
    03 - Read microcode level and machine type
    04 - Read robot param table
    08 - Read current instruction address
    10 - Read DI/DO
    20 - Read all program variables
    40 - Read position in pulses
    80 - Read specific program variable(s)
```

Only option "80" requires further input. All the other options perform the read request as soon as they are selected. The result of the read request is displayed on the screen in a user friendly format. For example, reading the machine or reject status will result in a message being printed which is the definition of the status code that the controller returns.

Option "80" prompts for the starting variable to be read and the number of variables to be read. The variable number that is entered is produced by the AML/E cross reference program (XREF). For example, if a counter C is indicated as variable number 48 by XREF, then to read the value of C, one would enter 48 for the variable number and 1 for the number of variables to be read. If more variables are requested than the number that actually exist in the current application, then only the actual number of variables that exist will be displayed. For example, if the existing application has 100 variables (numbered from 0-99), and the user requests 50 variable to be read starting from variable number 75, then only variables 75-99 will be displayed. If no variables exist in the range given by the user (i.e., the starting variable is greater than the actual number of variables in the current application), then no variables are displayed. Entering 0 for both the starting offset and the number of variables will cause all the variables in the current application to be read.

Note: Comaid only reserves storage space for 400 variables. Thus the user should never ask for more than 400 variables to be read. Likewise, if option "20" is used to read all the program variables, an error will occur if the existing application contains more than 400 variables. If this occurs, then the variables must be read 400 at a time.

X _ Transmit Execute Command

This option gives the user the ability to control the state of the controller from the host. Virtually all the buttons on the control panel have a corresponding X command which will perform the identical action from the host. When the X option is chosen, the following new options appear as a submenu:

```
VALID EXECUTE COMMANDS ARE

00 - Return to the Communications Menu
11 - Return home
12 - Recall memory
13 - Reset error
20 - Auto
22 - Start cycle
23 - Stop cycle
24 - Stop and memory
25 - Step
31 - Application 1
32 - Application 2
33 - Application 3
34 - Application 4
35 - Application 5
```

None of the X commands require further input. When they are selected, their action is performed. For many of the X commands, a delay **will** exist from when the option is selected until when the command has actually completed executing. For example, the Return Home (option "11") and Stop Cycle (option "23")" can be virtually instantaneous or can take over a minute to complete.

D _ Controller Initiated Communications

This option is chosen either when

1. The running application performs controller initiated data drive (i.e., a GET or PUT statement is used).

2. A debug breakpoint is set by using Control Command 10.

When this option is chosen, a new submenu appears as follows:

```
VALID COMMANDS ARE:

00 - Return to the Communications Menu
01 - Disable Data Drive (send an Xoff)
02 - Enable Data Drive (send an Xon)
98 - Display values received from controller by a PUT
99 - Prepare for GET
```

Note: The option "98" is chosen to display values that have been sent from the controller by a PUT statement. If a PUT has not yet

been encountered, this option will not appear; only after a PUT has been encountered will this option become available.

The controller is enabled for data drive by sending an Xon (option "02"). Once the Xon is sent to the controller, the host can no longer perform communication requests because the controller is enabled as the initiator (hence controller initiated communications), the host hooked to the controller is the slave. Since the host cannot initiate any communications transaction to the controller once an Xon is sent, control does not return to the main menu. Instead, Comaid loops indefinitely, printing the following messages:

```
In Data Drive Mode
Strike any key to exit Data Drive mode
Last Data Drive operation was: GET - variables 34 - 37
                               PUT - variable 38
                               GETC - variable 37
                               DEBUG
```

Note: Comaid supports the GETC instruction of AML/Entry Version 3. This instruction does not exist in AML/Entry Version 4, and has been replaced by the GET instruction.

The variable numbers shown above are examples. The actual values will differ, based on the actual GET, PUT, or GETC instruction encountered.

Once data drive mode is enabled, the host can only regain control from the controller if an Xoff is sent. Comaid thus loops until the user strikes any key. Upon striking a key, an Xoff is sent to the controller and data drive mode is exited. The user should strike the key to send an Xoff when the application is not near a GET, PUT, or the debug breakpoint. This is one reason that the data drive requests are displayed as they are received from the controller. By doing this, the user can monitor the application, so the key can be struck at the proper time.

Because the user only has 30 seconds to respond to a GET request, the user first prepares for a GET (or GETC). When option "99" is selected, the user is prompted for the values to be sent to the controller when a GET (or GETC) is requested. The values are later sent when the controller performs its request. Once the Xon is sent to the controller, any GET or PUT statement that is reached will be performed immediately. Thus the user must prepare for any GET the controller may perform. Option "99" can be selected as many times as necessary to load values that will be sent to the controller. The values that are entered remain valid until they are changed. Thus if data drive mode is exited, and entered after other communications requests, the values that were entered previously via option "99" are still in effect.

If variables are received from the controller via a PUT command, after exiting data drive the user may display the values sent from the controller by selecting option "98".

Note: To use the AML/E Version 4 controller initiated data drive (GET/PUT), level 15 or 16 microcode must be installed. If a lower

level is installed, then upon exiting data drive mode, manipulator power is likely to be lost.

Note: The data drive feature of Comaid will only work for the first 400 variables of an application. Data drive will not be correctly performed if a GET or PUT accesses any variable beyond variable number 400. If a PUT request accesses variables beyond variable number 400, then the values are not accessible by the user. If a GET request requires variables beyond variable number 400, only variables up to variable number 400 receive a value from Comaid. Thus variables that the controller has requested will not be given a value, and a transmission error (TE) occurs.

C – Control Executing Program

This option is chosen when the user wants to "control" the executing application. The options are the AML/E Version 4 Control requests. They appear in a new submenu:

```
        VALID CONTROL COMMANDS ARE:

 00 - Return to the Communications Menu
 01 - Suspend program execution
 02 - Restart program execution
 04 - Execute until next terminator encountered
 10 - Set debug address stop
 20 - Reset Application Program
 80 - Change value of variables in controller memory
```

Two of the control options require further input data. To set a debug breakpoint (option "10"), the user must enter the address where a breakpoint will be set. This address can be gotten from the left hand side of the .LST file created by the AHL/E compiler when the AML/E program was compiled. Once a breakpoint is set, the controller will stop before the AML/E statement at the specified address is executed.

> **Note:** Compiled AML/E programs use a hidden internal subroutine to perform all motion (i.e., PMOVE, MOVE, DPMOVE, GETPART, XMOVE). Internal address 72 corresponds to the actual motion statement of this internal subroutine. To make the program break immediately before motion to the next point, set a breakpoint address of 72.

Changing the values in controller memory (option "80") also requires additional data. The user gives the starting variable number (as indicated by the XREF program), the number of variables to be changed, and then each value. When all the data has been entered, the values are sent to the controller.

Note: As with the R 80 command, only 400 variables may be sent to the controller per request using option "80". When more than 400 variables mast be sent, break the request into several requests, each containing less than 400 variables.

T _ Transmit Teach Command

This option gives the user the ability to move the manipulator, change the motion settings, or activate/deactivate the digital outputs. A new submenu appears when this option is selected:

```
       VALID TEACH COMMANDS ARE:

00 - Return to the Communications Menu
01 - Teach a new point
65 - Turn a DO port ON or OFF
68 - Set linear
69 - Set payload
6A - Set zone
4E - Switch to right mode (7545-800S only)
4F - Switch to left mode (7545-800S only)
```

Almost all the Teach options require further data. The only exceptions are options "4E" and "4F". When teaching a new point, the user enters the X, Y, Z, and roll coordinates. When specifying a digital output (DO), the user enters the port number and a 0 or 1 (off or on). When specifying linear, payload, or zone, the user enters the integer value to be used. Remember the following ranges:

Parameter	Range
LINEAR	0-50
PAYLOAD	0-19
ZONE	0-15

Options "4E" and "4F" are only valid for the 7545-800S, because this is the only manipulator that is symmetrical. An Eot will result if either of these is chosen when a standard 7545 or a 7547 is installed.

F _ Execute a Command File

This option allows the user to place sequences of commonly used commands into a file and then execute the commands in the file. The commands are executed from the file in sequential order, with no user interaction required. When the end of the command file is reached, the user then strikes a key to return to the main menu. Virtually all of the commands are allowed to appear in a command file. The only exceptions are:

- Data drive commands may not be used (the D option).

- A debug breakpoint may not be set (the C 10 option).
- The communications buffer may not be printed (the P option).
- A command file may not be started from within a command file (the F option).

The following is a command file that will start an application named FASTEN.AML in partition 1.

```
C 20             Reset Controller
U 1              Unload Partition 1
L FASTEN 1       Download FASTEN.ASC to partition 1
X 11             Return Home
X 20             Automatic Mode
X 31             Select Application 1
X 22             Start Cycle
```

The commands appear just as they would be entered into Comaid interactively. These commands are entered into a file with the AML/E editor. For example, these lines could be placed into a file named FASTEN.BEG (begin fasten). Then by selecting the F option from the Comaid menu and entering FASTEN.BEG as the command file to be used, these command will be executed.

As the commands are processed, the results are scrolled on the screen instead of being printed in a menu-like style. The user can include comments following each request by using two dashes as in the AML/E language. This is strongly recommended so the user can more easily monitor the actions of the command file.

DOS Command Line Processing

The user may use Comaid to invoke a single request and immediately return back to DOS. The user simply gives the parameters, as they would be entered if run interactively, following Comaid on the DOS command line. Spaces are used to separate entries. For example, to download the file FASTEN.ASC to partition 1, the following would used:

```
COMAID L FASTEN 1       -- Download FASTEN.ASC to partition 3
```

As shown, a comment may be given on the command line -- indicated by two dashes, as in the AML/Entry language.

Almost all of the Comaid options may be given on the DOS command line. The only exceptions are

- Data drive may not be used (the D option).
- Setting a debug breakpoint may not be used (the C 10 option)
- The P option to print the communications buffer may not be used.

A command file can be invoked from the DOS command line. So instead of just downloading the file FASTEN.ASC, one may start it running by using the above command file with the instruction

```
COMAID F FASTEN.BEG          -- Start Command File FASTEN.BEG
```

DEBUGGING MLA APPLICATIONS

Once an AML/E application has been downloaded to the controller, it may be executed. However due to the complexity of robotic applications, there will probably be errors in the AML/E program. These errors are run-time errors that the compiler cannot catch at compile-time. The error may be an application error (e.g., the program is not doing what it is supposed to, yet continues to run), or a controller error (e.g., servo error). Locating the errors in the original program and correcting them is called "debugging" the program. This section describes how to use Comaid to debug application programs.

Using Read Requests

Three of the Comaid read requests are particularly valuable when an error occurs, they are:

- Read machine status
- Read current instruction address
- Read specific program variables

Reading the Machine Status

The most common errors in an AML/Entry program are AML/Entry errors and Data Errors. AML/Entry errors occur when an application uses a value that is illegal in the AML/Entry context in which it is used (e.g., using an invalid index for a group). Data errors occur when the application program uses a value that is illegal for the controller (e.g., division by zero). When the machine status is read, one of eight messages will be printed by Comaid. The messages correspond to the different errors that could have occurred in the controller. They are:

- No Machine Status Errors -- There is no error in the controller.
- Servo error -- The servos of the manipulator became constrained during a move; so power was shut down to protect the manipulator.
- Power failure -- The manipulator has lost power.
- Overrun -- The manipulator is not in the work area.
- Transmission error -- A communications error has occurred while talking to the host.
- Over Time -- A WAITI instruction has reached its time limit, and no label was given.
- AML/Entry error -- the AML/E program used an illegal value for AML/E.
- Data error -- the AML/E program used an illegal value for the controller.

When an AML/Entry error occurs,

the DE (Data Error) light on the operator panel will become lit and power will be lost. In addition to printing that an AML/Entry error occurred, Comaid will also print what caused the error. For AML/E Version 4, there are 7 AML/E errors:

- Part number too small for pallet -- The part number for pallets must be greater than O. This will only be generated by the SETPART command, since this is the only command that can initialize a part number. If communications are used to change the current part number, no error test is done to make sure the part number is legal.

- Part number too large for pallet -- The part number for pallets must be less than or equal to the number of parts in the pallet. This will only be generated by the SETPART command, since this is the only command that can initialize a part number. If communications are used to change the current part number, no error test is done to make sure the part number is legal.

- Invalid index for a group -- The index for a group must be greater than 0 and less than or equal to the number of points or counters in the group being referenced.

- Communications not established (unable to GET/PUT) -- This occurs when the controller encounters a GET, PUT or debug breakpoint, and either the controller is off-line, the communications cable is not attached, or the DSR signal is inactive. As long as Comaid is running, the DSR signal remains active. This usually occurs when the controller is off-line. If level 15.0 or lower microcode is installed, this can also occur when the user exits data drive mode.

- Invalid index for FROMPT function -- The second value in the FROMPT function must be 1, 2, 3, or 4. The actual index that caused the error can also be read using option "80" of Comaid. It resides in variable number 6. (Manipulator power will have to be turned on and the correct application selected if R "80" is to-be used).

- Square root of a negative number -- The argument for the square root function must be non-negative. The actual index that caused the error can also be read using option "80" of Comaid. It resides in variable number 4. (Manipulator power must be turned on and the correct application selected if R "80" is to be used).

- Invalid arguments for the ATAN2 function -- Both arguments for the ATAN2 function were zero.

Data errors will also cause additional information to be printed by Comaid. There are 11 possible data errors; however only the following four would likely appear. If any other data error appears, try powering off the controller for 10 seconds, powering it back on, and reloading the application program. If the data error can be reproduced, contact your IBM Field Representative.

- Arithmetic error -- This occurs when an expression causes numerical error in the controller. This can happen when the TAN of 90 degrees is taken or division by 0 occurs, for example.

- Invalid op-code -- This should only appear if a program that was compiled for the 7545-800S manipulator that contains a LEFT or RIG command is downloaded by mistake to a 7545 or 7547 manipulator.

- Invalid port number -- This appears if a port number was referenced in a sensor command that is illegal (i.e., less than or equal to zero or greater than the number of installed ports).

- Point out of workspace -- This appears if the manipulator tries to move to a point not within its workspace. This error can occur b any of the five AML/E motion commands (PMOVE, DPMOVE, ZMOVE, XMOVE, or GETPART). The point that the manipulator wants to move to resides in variables 0 - 3 (i.e., 0 contains the X coordinate, 1 the Y, 2 the Z, and 3 the roll). These can be read by using option "80" from Comaid. (Manipulator power will first have to be turned on an the correct application reselected before the R "80" command can be given).

When an error occurs in an AML/E application, reading the machine status is likely to give the cause of the error. Even so, sometimes it is har to determine which AML/E statement caused the error. The next step is to read the instruction address of the application.

Reading the Current Instruction Address

To read the instruction address, simply select option "08" from the Rea request menu. The address that is printed by Comaid is the decimal address of the current instruction or the instruction that caused error. By generating a listing file with the AML/E compiler, it is usually possible to determine the statement that caused the error. Th addresses that are listed on each line correspond to the hardwar address for the beginning of the line. If the line does not generat any compiled code, note that the line following the line has the sam address. The line that causes the error will be the line that has th address closest to, but not greater than the address of the error. Fo example, suppose an error occurs at address 113, and one AML/E line ha address 111 and the next AML/E line has address 114. The AML/E lin that has address 111 was responsible for the error.

Because of the implementation of motion in the controller, an interna subroutine is created by the AML/E controller that is responsible fo all motion. When a point out of workspace error occurs, one cannot rea the instruction address to determine the line that caused the error because the line will always be the line of the internal subroutine When a point out of workspace error occurs, the only recourse is to rea the value of the point that caused the error (variables 0 - 3). If i is still not possible to determine which motion statement caused th error, then try stepping through the application. The internal routin resides in addresses 1 - 87. If the instruction address is read and i falls in this range, the controller is within special compiler code.

Reading Specific Program Variables

The last important read request is option "80", which is used to read specific program variables. The manipulator power must be on and an application selected for this command to be accepted by the controller. Using option "80", it is possible to read any variable stored in controller memory. Since AML/E applications can contain expressions, it is valuable to be able to read the value of counters that appear in or are set by expressions.

When you select option "80", Comaid will prompt you for the starting variable number and the number of variables to be read. The first 34 variables (numbered 0 - 33) are used internally for various purposes. For example, variables 0 - 3 always contain the last point to which the program moved the manipulator. If a counter ever gets a wrong value, it is possible to step through the application, reading the value for the counter at each step. The erroneous statement can then be found.

Reading Other Values

It may be necessary to read the reject status while debugging. The reject status indicates why a transmission error occurred. If an application does not communicate with the host, then a transmission error cannot occur. However if data drive is being used, then transmission errors are likely. A list of the reject status return codes can be found on page H-21.

It may also be necessary to read the DI/DO during debugging. To read the DI/DO, simply select option "10" from the read command menu. The values of all the digital inputs and outputs will be displayed. You may discover that the wrong digital port is being used.

Using Control Requests

Some errors cannot be found by using just read requests. Read requests are most useful when an error occurs that causes power to be lost. However if the application continues to run, yet not perform the correct steps, then control requests are more valuable. By using control requests, you can suspend execution, step through an application, set debug breakpoints, and change values in controller memory should variables get incorrect values. To execute a control request, an application must be selected, otherwise a transmission error will occur.

Suspending Program Execution

Suppose the program is not doing what you thought it would do. You could execute this command by selecting option "01" from the control command menu. After suspending program execution, you can read the machine address to determine where in the application the controller currently is. This request suspends the controller at its current instruction, which may not necessarily correspond to the beginning or ending of an AML/E statement. However if you then select option "04"

execute until next terminator, the controller will then continue to the end of the next instruction, label, or subroutine call.

Restarting Program Execution

After suspending execution, you can perform read and step requests through each statement of your program. After determining the cause of an error, you may wish to continue execution. Selecting option "02" allows the application to continue from where it left off.

Executing Until the Next Terminator

This performs the same function as X "25" or the step button of the operator panel. It causes the current application to continue until the next terminator is encountered. Terminators are placed at the end of every AML/E statement, at the beginning of a subroutine, and at every label. By stepping through the application, it is possible to see if the error is caused by an incorrect flow of control statement (e.g., BRANCH, TESTC, COMPC,' TESTI, TESTP, etc.). As you step through the application, you can read program variables, the current instruction address, and the DI/DO. This is a lengthy process, but doing this will almost always detect the error. If the error is found to be an incorrect value for a counter, pallet, or point, it is possible to change the value using option "80". Doing this then allows the application to be continued so you can continue to look for more errors before returning to the AML/E editor to make the corrections.

Note: It is not possible to step through a GET, PUT, or debug breakpoint. Doing this will cause Comaid to hang and a TE error in the controller. If this should happen, manipulator power will have to be turned off and back on again to clear the error. The PC will have to be rebooted by striking the Ctrl, Alt, and Del keys simultaneously.

Setting a Debug Breakpoint

A faster way to step through many statements is to set a debug breakpoint via control option "10". When you select this option, Comaid will prompt you for the address where a debug breakpoint will be installed. Only one debug breakpoint may be installed at a time, and installing a new breakpoint cancels any previous one. When the debug breakpoint is reached, it is cleared. So the next time the same statement is encountered, the program will not stop execution. The address you give should be one of the addresses listed in the compiler listing file (.LST). If this is not the case, then unpredictable results will occur. There is one exception to this rule; internal address 72 is the address of any motion statement. If a breakpoint is set at this address, the controller will reach the breakpoint when the next motion statement is reached.

If the controller is currently suspended, and you set a breakpoint address for the current address, the breakpoint will be reached immediately and no instructions will be executed. Control does not

continue and will stop the next time the same address is reached. To do this, you should first step until the next terminator, and then set a debug breakpoint for the desired address.

Once a debug breakpoint has been set, the user should enter the data drive menu of Comaid and enable data drive. When the controller reaches the debug breakpoint, the message:

Last data drive operation was: DEBUG

will appear on the screen. After this appears, strike any key to exit the data drive menu and complete the debug breakpoint transaction. The user should estimate the amount of time necessary for the debug breakpoint to be reached for two reasons:

1. Other communications requests can continue to be performed. It is only when the application gets near the debug breakpoint that data drive mode must be enabled. In fact, you can even let the controller run until a transmission error occurs (provided the program does not contain a GET or PUT instruction). After this occurs, you can send an X "13" to reset the error and then enter the data drive mode to complete the debug transaction.

2. If you have an estimate for when the breakpoint should be reached, you can wait for that time period to elapse. If the breakpoint is not encountered, then the data drive mode can be exited, and you can suspend program execution. It is entirely possible that the error is a flow of control problem, and the breakpoint will never be reached. After breaking execution, you can step through the program, reading the instruction address after each step, to determine the flow of the program.

Note: Remember not to step through a debug breakpoint. This will cause a TE error in the controller and cause Comaid to hang. Should this accidently be done, you will have to reboot the PC by striking the **Alt**, **Ctrl**, and Del keys simultaneously. The manipulator power will have to be turned off and back on again to clear the TE in the controller.

Resetting the Controller

Suppose you wish to stop the application and restart it from the beginning. The quickest way to do this from Comaid is to use control option "20". This deselects the current application and clears any errors. This is better to use than stop cycle, because this command stops execution as soon as any current motion is complete. Stop cycle does not necessarily stop the program from running, because the program could contain a BRANCH to the first statement as its last statement. This option cannot be used to stop a program that is hung on a WAITI. If a program is hung on a WAITI, the only way to clear this condition is to either set the DI oh or off according to the value specified in the WAITI, wait for the time limit to expire (a time limit of zero means "forever" to the controller however), or to turn off manipulator power with the Stop Button on the operator panel.

Changing Variable Values in Controller Memory

Suppose that by stepping through a program and reading variables values, you have found a statement that sets the value of a counter incorrectly. At this point you could exit Comaid, make the change in the original program, recompile the program, download the program again, and rerun the program. Doing this may take many minutes however, and you may find that the statement after the erroneous one also contains an error. When the first error is discovered, rather than exiting Comaid, it is possible to change the counter to its correct value and continue execution. Doing this allows you to find as many errors in the source program as possible, and fix them all at once. This can be done by selecting control option "80".

After selecting control option "80", the user is prompted for the starting variable number and the number of variables to be changed. This information can be gotten from a cross reference listing produced by the XREF program (see "XREF Program" on page 2-34 and "XREF Program" on page 4-89). You can change up to 400 variables in a single request, as long as they are contiguous in controller memory.

Between the read requests and the control requests, the AML/E programmer has many debugging tools at his disposal. Debugging AML/E applications that do not use data drive will consist primarily of read and control requests. Users should avoid Teach requests, however tempting they are, because they will cause the manipulator to move to home position before their action is invoked.

Introduction to the AMLECOMM System

AMLECOMM is a system consisting of many BASIC modules, which can be configured to create a BASIC program that is capable of communicating with the IBM 75xx manipulator family. AMLECOMM supports all the AML/E Version 4 communications. The user simply sets the proper input variables, calls AMLECOMM and then examines the return code and output variables for completion information. All the routines that provide these functions are called using a BASIC GOSUB statement from the user's application program. Thus the typical AMLECOMM user might write a BASIC program to automatically down-load an application, return the manipulator to home, select the application and start the cycle -- all by simply calling AMLECOMM with the appropriate inputs.

In addition, full error support is provided via error text files. For any error return code, there is a corresponding error explanation message available to provide instant error identification.

AMLECOMM supports interpretive and compiled BASIC, though compiled Basic is recommended because of the slow speed of the interpreter. AMLECOMM even has the ability to begin selected operations and return to the user's program while the operation is being executed by the controller. Upon successive poll operations the user is notified of the current status of the outstanding operation request. This allows the user's program to do other useful work while the manipulator is performing potentially slow operations, like return home.

The AMLECOMM System Files

The AMLECOMM system is on the third AML/E ship diskette. It consists of many files:

- COMPILE.BAT -- This is a batch file that shows how to create an executable (.EXE) version using the BASIC Compiler and DOS Linker.
- CONVERT1.COM and CONVERT2.COM -- These files must be executed before an interpreter version of AMLECOMM can be run. See "Interpreter vs. Compiler" on page 8-32 for more on these.
- CCVTFLT.OBJ and CFLTCVT.OBJ -- These files must be linked with the compiled source code should the user desire a compiled version of AMLECOMM.
- MSGCOM.TXT -- A message file which contains the error messages for AMLECOMM, and the error messages for the AML/E version 4 communications.
- CONFIG.BAS -- The configuration program then creates an AMLECOMM program that meets the user's needs.

- AMLECOMO.BAS -- The only AMLECOMM system module the user shoul change. This module contains the parameters that AMLECOMM needs i.e., array bounds, error handling, etc. The user should chang this file for each application, as some applications have differen needs. This must be changed BEFORE the configuration utility run. These are briefly documented in the following section, an more completely in Appendix G, "Configuration Parameters fo AMLECOMM."

- AMLECOM1.BAS-AMLECOMK.BAS -- The AMLECOMM system modules. Thes should not be modified.

- MENUCOMM.BAS -- A program which when compiled with a version o AMLECOMM configured with all the available options, gives the user menu-driven communications program. This program is basically th source code for Comaid, except certain features have been removed i order to make MENUCOMM smaller and easier to view. This progra demonstrates how to call the AMLECOMM program.

Installation Procedure

Before AMLECOMM can be used, the user must configure a working versio of AMLECOMM from the AMLECOMM system modules. To do this, the user mus first determine which of the AML/E Version 4 communication requests th application needs. After doing this, the utilities diskette is inserte into the a: drive (if there is no fixed disk installed), and the use enters

```
BASICA a:config
```

The configuration utility will prompt the user with a series of yes/n questions that determine which communication requests the create working version of AMLECOMM can handle. When finished, the working cop of AMLECOMM is loaded in memory, and the user is returned to the BASI prompt level.

The configuration utility will print the commands necessary to exi correctly. The user must first enter the commands:

```
16310
16320
16330
```

If the compiler is to be used, the file must be saved in ASCII format. Thus the command

```
SAVE "filename"
```

is used, where filename denotes the name of the .BAS file you wish t have this working copy of AMLECOMM stored in. If the interpreter is t be used, the source file may now be merged into memory and be run. Provided the source file has already been created and is stored in ASCII format, the following command may be used:

MERGE "filename"

The filename used above is the name of the .BAS file that your application is in. If you have not yet created your application program, then store the working copy of AMLECOMM in ASCII format, so it can later be merged. See "Interpreter vs. Compiler" for more on this.

As the questions are answered by the user, only certain modules of the AMLECOMM system are merged into the working version. The configuration utility looks for the modules on the default directory; thus the user must either copy the AMLECOMM system files to the directory containing BASICA.COM, copy BASICA.COM to the directory containing the AMLECOMM system files, or use the DOS path feature to begin BASICA (see the DOS user manual).

Interpreter vs. Compiler

AMLECOMM can be used with either compiler or advanced interpreter basic (BASICA). It is recommended, however, that compiled basic be used. This is because the interpreter is vastly slower. The AML/E communications protocol requires retries of sending D records if a response is not received within three seconds. BASICA can sometimes compute and verify the accuracy of a received D record within this time limit, but usually it cannot. When it cannot acknowledge a D record within three seconds, the controller sends another copy. AMLECOMM must then use more processing cycles to accurately ignore the retry. Thus it often takes as long as six to nine seconds to have AMLECOMM correctly receive a D record from the controller. AMLECOMM still functions correctly, just slower. For this reason, IBM Compiler Basic is recommended.

If Compiler Basic is used, the object modules CCVTFLT.OBJ and CFLTCVT.OBJ must be linked to the compiled source program. This is shown in the batch file COMPILE.BAT that is shipped with the AML/E system.

If BASICA is used, two modules must first be loaded. The modules are CONVERT1 and CONVERT2. The user simply enters these names as if they were DOS commands. Each of these will print a status message on the screen to inform the user they are correctly loaded. If these are not loaded, AMLECOMM will return an initialization error with an error code of 2.

AMLECOMM Line Numbers

AMLECOMM uses Basic line numbers 51-19999. Thus an application may not use these line numbers. The user's application begins executing at the first line greater than 19999. Thus a user application may contain 50 lines of header comments, but the first executable statement must be no less than line number 20000.

Using Compiler Basic

The user application must "include" the AMLECOMM program using the $Include metacommand of Compiler Basic. This may be done as follows:

```
1 'First Comment
2 'Second Comment



49 'Forty Ninth Comment
50 REM $INCLUDE: 'AMLECOMM'
20000 'Start of user's Code
```

To use the $Include metacommand, the AMLECOMM working copy must be saved in ASCII format. At the end of the AMLECOMM configuration utility, the user is informed to save the program with:

```
SAVE "filename",a
```

This saves the working copy of AMLECOMM in filename.bas in ASCII format. The user may use any legal DOS filename to store the working copy of AMLECOMM. The above example assumes it was stored in AMLECOMM.BAS.

AMLECOMM will automatically insure that the code begins executing at the first user application statement greater than 19999. Also, AMLECOMM turns off the listing of the AMLECOMM source, so this will not appear in the user's listing.

If the user is compiling AMLECOMM then two object files must be linked together with the compiled source. These files are CCVTFLT.OBJ and CFLTCVT.OBJ. In addition, a third object file called IBMCOM.OBJ is also necessary and is supplied with the BASIC compiler. The file COMPILE.BAT may be used to compile and link a user application.

Using BASICA

In order to use BASICA, the working copy of AMLECOMM must be merged together with the user's source code for the application. The instructions at the end of the AMLECOMM configuration utility assume the user has already written the source code. Since the AMLECOMM configuration utility terminates in the BASICA monitor with AMLECOMM already loaded, all that is necessary is to merge the source code with:

```
MERGE "filename"
```

where filename.bas is the file that contains the source code for the user's application. This file must be stored in ASCII format, otherwise the merge will fail. The file being merged must be stored in ASCII format. If the source file is not stored in ASCII format (i.e., it was entered line by line in BASICA, and saved without the ,a), or if the source file has not yet been created, it is best to exit BASICA, saving the working copy of AMLECOMM with:

```
SAVE "filename",a
```

This will save the working copy of AMLECOMM in filename.bas. At a later time, the user can then combine this file as follows (assuming filename above is replace with AMLECOMM):

```
BASICA
LOAD "application"
MERGE "AMLECOMM"
SAVE "application",a
RUN
```

BASICA also requires the installing of two COM files called CONVERT1.COM and CONVERT2.COM. These modules must be installed prior to running the application that is merged with AMLECOMM. The user simply enters the commands CONVERT1 and CONVERT2 to DOS and these routines will be installed. If the system is rebooted, then these routines are lost and must be reloaded. If the user intends to use BASICA frequently on AMLECOMM, then the two commands CONVERT1 and CONVERT2 should be placed in the AUTOEXEC.BAT file (See DOS reference). By doing this, the user need not worry about installing these COM files. CONVERT1 and CONVERT2 use software interrupts 65 and 67 (Hex). The user application should not use these interrupts.

Initialization and Configuration Parameters

Before the configuration utility for AMLECOMM is run, there are many variables that must be initialized that depend on the applications for which AMLECOMM is to be used. Their meaning and default values are given here with a more complete discussion to be found in Appendix G, "Configuration Parameters for AMLECOMM."

- POSTING.A% -- If set to Y.A%, then an X record request will return to the user application before completing the request. If set to N.A% then AMLECOMM will wait for the X record request to complete before returning to the user application. The default is N.A%. POSTING.A% only affects X records.

- COMM.A -- This variable needs to be set to a number that may be used as a DOS file number for the communications file. The default is file number 1.

- OPENFCOMM.A -- This is similar to COMM.A, except this is a file number used for a file to be downloaded. The default is 3.

- VARMAX.A -- This variable allocates storage for the maximum number of variables that exist in an application running in the controller. The default is 400.

- VARMAXRECS.A -- This is used internally, and should be set to INT(VARMAX.A/4)+1.

- ROBOT.A$ -- This variable defines the type of manipulator that is connected to the manufacturing system. Set ROBOT.A$ to "7545" or

"7547". The default is "7545". This is only used to determine h many coordinates will be sent for a "Teach a Point" request. If 7545-800S is attached, leave ROBOT.A$ as "7545".

- ADAPT.A$ -- This variable is set to the communication port th application will use. The default is "COM1:".
- ARRAYMODE.A% -- This variable indicates whether the arrays used fo reading variables, poking variables, and data drive are 0 or based. The default is 0 based so the variable numbers coincide wit those produced by the cross reference utility.
- RRVARS.A$(VARMAXRECS.A) -- This array is used internally, and mus be dimensioned with a size equal to VARMAXRECS.A. Unexpecte results will occur if this is not the case. The declaration fo RRVARS.A$ should not be removed.
- RVARS.A(VARMAX.A) -- This array is used to return variable value from a R80 request (read variables). Its size must be set equal t VARMAX.A. Unexpected results will occur if this is not the case The declaration for RVARS.A should not be removed, even if no rea variables request is being used. The configuration utility wil automatically remove it.
- GVARS.A(VARMAX.A),PVARS.A(VARMAX.A) -- These arrays are used for GE and PUT operations in data drive. They should be dimensioned wit size equal to VARMAX.A. Unexpected results will occur if this i not the case. The declarations for these should not be removed even if no data drive is being used. The configuration utility wil automatically remove them.
- C8OVARS.A(VARMAX.A) -- This array is used to poke variables int controller memory using the C80 communication protocol. The size o this array must be set to VARMAX.A. Unexpected results will occu if this is not the case. As with RVARS.A, GVARS.A, and PVARS.A, th DIM statement for C8OVARS.A should not be removed, even though th C80 feature may not be needed.
- ON ERROR GOTO 0 AMLECOMM must install its own error handler whe active to trap communication errors. This will cancel any erro handler the user has established for his own program. AMLECOMM wil restore a single error handler for the user. The user changes th two ON ERROR statements to point to his error handler. If no use error handler is needed, then ON ERROR GOTO 0 statements are used.

The file AMLECOMO.BAS is shown in its entirety i Appendix. G, "Configuration Parameters for AMLECOMM." Should this fil ever become destroyed (e.g., a line deleted), you may get a backup cop from Volume 3 of the AML/Entry Ship Diskettes.

Calling AMLECOMM

The AMLECOMM program consists of one BASIC module entry point (regardless of which communications requests were chosen at configuration time). All calls to this module must pass the required information in special input and output BASIC variables. OPERATION.A$ contains the request that AMLECOMM is to perform.
For example, to initialize AMLECOMM one uses:

```
20000 OPERATION.A$=""
20010 GOSUB 100
```

Type these lines in as they are shown (even with the two double quotes). To initialize AMLECOMM, OPERATION.A$ must contain the null string. This performs the required initializations for AMLECOMM. RTRN.A% contains the return code of any AMLECOMM request, a 0 means that the operation (in this case initialization) was performed correctly. A non-zero return code in RTRN.A% indicates an error occurred. See "AMLECOMM/COMAID Error Messages" on page B-60 for a list of the AMLECOMM errors.

After initializing AMLECOMM, the user application should call AMLECOMM to open the communications port designated by ADAPT.A$.
This is done by:

```
21000 OPERATION.A$=OPEN.A$
21010 GOSUB 100
```

OPEN.A$ is an AMLECOMM constant. The value of OPEN.A$ is immaterial to the user's application; its value is critical only to AMLECOMM. OPEN.A$ is initialized on the first call to AMLECOMM. If the initialization fails, OPEN.A$ may not be properly initialized. Once again, a non-zero return code in RTRN.A% indicates that the communication port was not correctly initialized. Either COM1: or COM2: can be opened, depending on the value of ADAPT.A$ when AMLECOMM is called. The default is COM1:. If an application desires COM2:, then perform the assignment:

```
ADAPT.A$="COM2:"
```

before the call to AMLECOMM.

After initializing and opening the communications port, the user may request any AML/E Version 4 communication request. The user simply sets up the appropriate input variables, calls AMLECOMM via GOSUB 100, and looks at the appropriate output variables and the return code in RTRN.A%. All the AMLECOMM variables have a ".A" suffix to distinguish them from any user application variables. Thus a user application may use any variable name that does not end in ".A". Examples are the operation variable, OPERATION.A$, the return code, RTRN.A%, and the read variables array RVARS.A(n).

The following table contains a list of the input and output variables for each communication request. The first column indicates the operation. The values listed in this column are constants that are placed in the variable OPERATION.A$. The second column indicates the operand. The values listing in this column are constants that are

placed in the variable OPERAND.A$. The third column list inpu variables that are needed for the given operation and operand. The *las* column lists the output variable for each operand. For example, to *rea* variables 10-15, one would use:

```
21000 OPERATION.A$=R.A$
21010 OPERAND.A$ =VARS.A$
21020 VAROFFS.A% = 10
21030 VARCNT.A% = 6    '15-10+1=6
21040 GOSUB 100
```

The output variables would be in RVARS(0)..RVARS(5). Note that th operation is similar to the first menu of Comaid, and the operand i similar to the submenus of Comaid. Examples for each request are give following the table.

- - - - - - -	- - •aNOZ u.avouva WIMUNFI %V"IVAIHOd %wwnmixoa iru'v-VIT*Vv.x	(AIII0 S009-5¹751) STIHOIN = (API() S009-5,50 STLITI = twalloz = STOVICAVd = SVINHNIU = tv.moa = STINIOd =	(Hovel) SV'I =
-	S•RINN tiviliva	-	(puoTumoa) tv.N =
-	t•Ilivd	-	(Pnalun) tv•n =
(47) •atusod WINONVA ` WXYPRIVA)TSHVAN Ora' 'Oria ti•muu (z0v auplua (£)V'rIff.LONOIN STNIIIII OUNHIN	- WSLIONVA 'WINONVA - - - - - -	Oi•sod = STSNVA = tv•oma = tiruaav = STWINd = SV'OHOIN = STIVISH = STIVISN =	(P8921) ST' U =
- %rsnIvIs	%Tian = %V'NO = WONIISOd	tv saadv . STIfIddV = STE'IddV = tv zaddv = Orvidav . Oral's = STNaNdOIS = S•A0dOIS = STAOINVIS = troInv . trunsu = ti•wamx = WHWOH =	(94noexa) S•X =
STDIVIIMA Llano ONIONOdSH2RIO0	STD:MINYA IOdNI HUHIO	tiramvHado	STNOLINHad0

NO113M1A A9 S319VI2IVA **indinoandm** V01033111V

OPERATION.A$	OPERAND.A$	OTHER INPUT VARIABLES	CORRESPONDING OUTPUT VARIABLES
= C.A$ (Control)	= SUSPEND.A$	-	-
	= RESTART.A$	-	-
	= TERM.A$	-	-
	= DEBUG.A$	ADDRESS.A%	-
	= RESET.A$	-	-
	= POKE.A$	CVARCNT.A% CVAROFFS.A% C8OVARS.A(VARMAX.A)	-
= D.A$ (Data Drive)	-	DDSWITCH.A% = OFF.A% = ON.A%	- STATUS.A% RTRN.A$= NODD.A$ = GETC.A$ = GET.A$ = PUT.A$ = DEBUG.A$ DDOFFS.A%(for GETC.A$ GET.A$ PUT.A$) DDCNT.A% (for GET.A$ PUT.A$)
		RTRN.A$=GETC.A$ GETCVAR.A	STATUS.A%
		RTRN.A$=GET.A$ GVARS.A(VARMAX.A)	STATUS.A%
		RTRN.A$=PUT.A$	STATUS.A% PVARS.A(VARMAX.A)
		RTRN.A$=DEBUG.A$	STATUS.A%

Note: All module calls return the following variables:

```
RTRN.A%    - Integer Error return code
STATUS.A% - Integer status variable indicating the
             state of AMLECOMM:
             - IDLE.A%    - Able to accept new
                            operation requests.
             - PENDING.A% - Operation pending, poll for completion.
             - READY.A% - Data drive operation ready
                            for data from user's application.
             STATUS.A% can only enter the PENDING.A% or READY.A%
             state if an X record with posting or data drive mode is
             selected. Thus STATUS.A% can be considered an output
             variable of an X record transaction and of a data drive
             transaction, and so is shown as such in the preceding
             table.
```

AMLECOMM calls are performed using a BASIC GOSUB 100 statement. AMLECOMM has to be merged into the user's application program as already described. The following describes the steps used to set up the input variables and pass control to the AMLECOMM interface module. The symbols "2nnna", "2nnnb", "2nnnc" and so forth refer to BASIC line numbers for code segments in the user's application program. Remember the statements in a user application must have line numbers greater than 19999, thus these symbolize line numbers with this constraint. Line numbers may of course be in the 30000's, 40000's, 50000's, or 60000's, so this notation does not imply line numbers must be in the 20000's.

Initialize AMLECOMM for Use

```
2nnna OPERATION.A$=""            'Call AMLECOMM initialization
2nnnb GOSUB 100
```

This GOSUB command initializes and configures AMLECOMM to its initial state. This call is the very first call to AMLECOMM and it is only done once.

Opening the ADAPT.A$ Communications Port

```
2nnna OPERATION.A$=OPEN.A$       'Open the COM port
2nnnb ADAPT.AWCOM1:"             'Specify COM1:
2nnnc GOSUB 100                  'Call AMLECOMM
```

This GOSUB command can only be done after a successful initialization. It will open the communications port specified by ADAPT.A$. If the controller is off-line or the cable not connected, an error will occur. Thus the user application should test for a non-zero return code, and retry this call should an error occur.

Execute Operation

```
2nnna OPERATION.A$ = X.A$         'Specify "Execute" operation
2nnnb OPERAND.A$ = HOME.A$        'Specify "Move Home"
2nnnc GOSUB 100                   'Call AMLECOMM
```

This command causes the attached manipulator to execute a "Move Home". Control will not return to the caller until the move is complete.

```
2nnna OPERATION.A$ = X.A$         'Specify "Execute" operation
2nnnb OPERAND.A$ = HOME.A$        'Specify "Move Home"
2nnnc POSTING.A% = ON.A%          'Specify asynchronous-
                                  'completion posting
2nnnd GOSUB 100                   'Call AMLECOMM
```

This example causes the attached manipulator to execute a "Move Home". Control will return to the caller once the move has started. The progr can then do other things while the move is completing.
The AMLECOMM program enters the PENDING.A% state (as indicated by th variable STATUS.A%), and will reject any new AMLECOMM requests until th preceding X record request has completed. There are two ways t complete the preceding X record transaction. The first is to have th user application continue, and only when it is necessary to cal AMLECOMM with a new request, the application waits for the preceding request to terminate. The following lines of code achieve this:

```
2nnne WHILE STATUS.A% = PENDING.A%
2nnnf    GOSUB 100                'Call AMLECOMM
2nnng WEND
2nnnh POSTING.A% = OFF.A%         'Specify no posting
```

OPERATION.A$ must still be set to X.A$, and OPERAND.A$ must still be se to the last X record request (in this case HOME.A$).
The application will wait until AMLECOMM has entered the IDLE.A% state The user should test RTRN.A% upon exiting from the loop to make sur that the protocol completed normally. The second way to end th protocol is to use the Basic ON COM statement. The AMLECOMM state c only change from PENDING.A% to IDLE.A% when the controller has sent a Xon following a previous Xoff. By using the ON COM statement, the use application can be interrupted when there is an Xoff or Xon from th controller on the line.

The following lines of code perform this:

```
2nnne ON COM(COMM.A) GOSUB 2nnnf : GOTO 2nnnk
2nnnf    GOSUB 100              'Call AMLECOMM to check for change
2nnng    IF STATUS.A%=PENDING.A% THEN 2nnnj
2nnnh    COM(COMM.A) OFF        'Shut off future trapping
2nnnf    POSTING.A% = OFF.A%    'Turn off posting
2nnnj    RETURN
2nnnk .
2nnnf .
2nnnm .
```

OPERATION.A$ must still be set to X.A$, and OPERAND.A$ must still be set to the last X record request (in this case HOME.A$) whenever AMLECOMM is called.
Note that POSTING.A% should not be changed until the previous request has been completed, hence we include it with the interrupt handler. Using this approach, the user need not worry about imbedding wait loops throughout his application code. The application may continue at its own pace, and the next AMLECOMM request need not be preceded by a wait loop. If the protocol has not been completed, a return code of RTRN.A%=8 will be returned when a new function is requested (see "AMLECOMM/COMAID Error Messages" on page B-60).

The other X record requests are called identically to HOME.A$. Their meanings are:

OPERAND.A$	ACTION
HOME.A$	Causes manipulator to return home.
RMEM.A$	Recalls memory.
RSTERR.A$	Reset error.
AUTO.A$	Enter auto mode instead of manual mode.
STARTCY.A$	Start cycle.
STOPCY.A$	Stop cycle.
STOPMEM.A$	Stop and set memory.
STEP.A$	Perform a single step of the selected application.
APPL1.A$	Select application 1.
APPL2.A$	Select application 2.
APPL3.A$	Select application 3.
APPL4.A$	Select application 4.
APPL5.A$	Select application 5.

A Simple Read Operation

```
2nnna OPERATION.A$ = R.A$         'Specify "Read" operation
2nnnb OPERAND.A$ = RSTAT.A$       'Specify read reject (Eot) status
2nnnc GOSUB 100                   'Call AMLECOMM
```

This example causes the controller to return the reason code for a previous Eot response via communications. The reason code (two numeric characters) is returned in the RTRN.A$ variable. The English explanation of the error can be found in MSGCOM.TXT (See "MSGCOM.TXT" on page B-66). Reading the machine status or the current instruction address is similar, except OPERAND.A$ is set to MSTAT.A$ or ADDR.A$, and the return value in RTRN.A$ is four characters long instead of two. The return value of MSTAT.A$ is discussed in the AML/E System Documentation. For ADDR.A$, the four characters represent the decimal address. For example, if the current instruction address were 72, then RTRN.A$ would be equal to the string "0072".

The Other Read Operations

```
2nnna OPERATION.A$ = R.A$         'Specify "Read" operation
2nnnb OPERAND.A$ = MICRO.A$       'Specify read microcode level and
2nnnc GOSUB 100                   'machine type and call AMLECOMM
```

This example will read the microcode level and manipulator type into the array MICROTBL.A. MICROTBL.A(1) will contain the major microcode level, MICROTBL.A(2) will contain the minor microcode level, and MICROTBL.A(3) will contain the manipulator type.

Reading the manipulator parameter table or the current manipulator position is similar to this except the return values are in the arrays PARMTBL.A and POSTBL.A. The 12 values returned in PARMTBL.A are:

Index Number	Description
1	First Link Length (in mm)
2	Second Link Length (in mm)
3	Theta 1 Maximum Pulse
4	Theta 2 Maximum Pulse
5	Roll Motor +/- Maximum Pulse
6	Theta 1 Offset(in radians)
7	Theta 2 Offset (in radians)
8	Theta 1 Pulse Rate
9	Theta 2 Pulse Rate
10	Roll Motor Pulse Rate
11	Z-Axis Pulse Rate
12	Z-Axis Maximum Stroke

The 4 values returned in POSTBL.A correspond to the current position in pulses of the Theta 1, Theta 2, Z, and Roll axes.

Reading variable values from an application running on the controller is slightly more complex. The user application must set up VARCNT.A% to have the number of consecutive variables to be read, and VAROFFS.A% must contain the starting variable number. Note that VAROFFS.A% depends on

the value of ARRAYMODE.A% set up by the user in the file AMLECOMO.BAS (see Appendix G, "Configuration Parameters for AMLECOMM"), but using the default of 0 allows VAROFFS.A% to be set to the exact number produced by the AML/E XREF utility. For example, to read variables numbered 34-43 on the cross reference listing, one would use

```
2nnna  OPERATION.A$ = R.A$      'Specify "Read" operation
2nnnb  OPERAND.A$   = VARS.A$   'Specify read variables
2nnnc  VAROFFS.A%   = 34        'Specify returning variables-
                                'starting with the 34th variable.
2nnnd  VARCNT.A%    = 10        'Specify a count of ten variables-
                                'to return.
2nnne  GOSUB 100                'Call AMLECOMM
```

If the return code is 0 (RTRN.A%=0), then the read operation completed normally, and the 10 values read are in the array RVARS.A. The values read will be placed in RVARS.A(0) through RVARS.A(9). If fewer variables existed in the current application than the number requested, then the actual number of variables read will be returned in VARCNT.A%. If in the above example the last variable according to the cross reference listing was numbered 40, then on return VARCNT.A% would equal 7 (40-34+1). To read all the variables in controller memory, the application sets VARCNT.A%=0 and VAROFFS.A%=0. As above, the actual number of variables read will be placed in VARCNT.A% and the actual values will be placed in RVARS.A(0) through RVARS.A(VARCNT.A%-1).

Unload Operation

```
2nnna OPERATION.A$ = U.A$         'Specify "Unload" operation
2nnnb PART.A$        = "1"        'Specify the partition to unload
2nnnc GOSUB 100                   'Call AMLECOMM
```

This example causes the controller to unload the specified partition in preparation for a subsequent download to that partition.

Download Operation

```
2nnna OPERATION.A$ = N.A$         'Specify "Download" operation
2nnnb PART.A$      = "1"          'Specify the partition to load
2nnne NAME.A$      = "AMLEPROG"   'Specify the program to download
2nnnd GOSUB 100                   'Call AMLECOMM
```

This example causes the specified program to be downloaded to the specified partition. AMLECOMM simply passes NAME.A$ to the Basic OPEN command, so the drive number, path, and filename must be compatible with your level of DOS. The only exception to this is that AMLECOMM will append a file extension of ".ASC" if none is given. Thus one can download a file that does not have a ".ASC" extension, and it is the user's responsibility to make sure an actual ASCII file produced by the AML/E Compiler is downloaded. If this is not the case, then a transmission error (TE) will occur.

Teaching a Point Operation

```
2nnna  OPERATION.A$ = T.A$       'Specify "Teach" operation
2nnnb  OPERAND.A$   = POINT.A$   'Specify Teach a point
2nnne  X.A!         = 400        'Specify the "X" coordinate
2nnnd  Y.A!         = 300        'Specify the "Y" coordinate
2nnne  Z.A!         = -100       'Specify the "Z" coordinate
2nnnf  R.A!         = 0          'Specify the "R" coordinate
2nnng  GOSUB 100                 'Call AMLECOMM
```

This example causes a Teach operation to be performed to the specified coordinates. The manipulator will immediately move to the specified point . T records can only be issued at certain times. T records cannot be issued when an application is running, stopped by C01 (suspend), C04 (execute to next terminator), or in step mode. The application must first be stopped by C20 (reset) or X23 (stop cycle).

Warning: Due to the implementation of Teach Records in the controller, if the manipulator has been taken off-line, moved, and put back on line, then a T Record wilt cause the manipulator to move home before invoking its action.

Turning on a DO Port

```
2nnna  OPERATION.A$ = T.A$       'Specify "Teach" operation
2nnnb  OPERAND.A$   = PORT.A$    'Specify Teach a port
2nnne  PORTNUM.A%   = 5          'Specify DO port number 5
2nnnd  PORTVAL.A%   = 1          'Specify Turn ON (0 means OFF)
2nnne  GOSUB 100                 'Call AMLECOMM
```

This example causes a Teach operation to be performed that will turn on Digital Output port number 5.

Changing LINEAR, PAYLOAD, or ZONE

```
2nnna   OPERATION:AS = T.A$         'Specify "Teach" operation
2nnnb   OPERAND.A$ = ZONE.A$        'Specify Teach a port
2nnnc   ZONE.A%      = 10           'Specify a Zone of 10
2nnnd   GOSUB 100                   'Call AMLECOMM
```

This example causes a Teach operation to be performed that will set zone to 10.
LINEAR and PAYLOAD are changed in a similar manner. Using a value of 0 for LINEAR, PAYLOAD, or ZONE will set the value back to its default switch setting.

Changing Arm Configuration (7545 _800S Only)

```
2nnna OPERATION.A$ = T.A$         'Specify "Teach" operation
2nnnb OPERAND.A$ = LEFT.A$        'Specify Left Mode
2nnnc GOSUB 100                   'Call AMLECOMM
```

This example causes a Teach operation to be performed that will change the arm configuration of the 7545-800S to left mode. Changing the mode to right is done similarly. The protocol for switching the arm configuration of the 7545-800S uses a P record instead of a T record, but AMLECOMM hides this from the user.

A Simple Control Operation

```
2nnna OPERATION.A$ = C.A$         'Specify "Control" operation
2nnnb OPERAND.A$ = SUSPEND.A$     'Suspend current application
2nnnc GOSUB 100                   'Call AMLECOMM
```

This example will suspend the current application as soon as possible (i.e., as soon as any motion completes). The application can then be restarted using the RESTART.A$ operand. The RESTART.A$, TERM.A$ (execute until next terminator), and RESET.A$ are performed like the SUSPEND.A$ operand.

Setting a Debug Breakpoint

```
2nnna OPERATION.A$ = C.A$         'Specify "Control" operation
2nnnb OPERAND.A$ = DEBUG.A$       'Suspend current application
2nnnc ADDRESS.A% = 72             'Break after next motion
2nnnd GOSUB 100                   'Call AMLECOMM
```

This example will cause a breakpoint to be set at address 72 (decimal). The controller will stop execution and send a controller initiated communications request to the host when this address is encountered. Address 72 is a special address to the AML/E compiler which is at the end of its internal motion subroutine. Thus as soon as the next move is complete, execution will stop. All other addresses should be attained

from the .LST file created by the AML/E compiler. Execution will halt before the statement at the given address is executed.

Because the controller initiates a communications request when the breakpoint is reached, the application must enter data drive mode to await the arrival of the controller's message. See the forthcoming discussion of data drive.

Changing Variables in Controller Memory

```
2nnna OPERATION.A$ = C.A$          'Specify "Control" operation
2nnnb OPERAND.A$ = POKE.A$         'Poke variables into memory
2nnnc CVAROFFS.A%= 34              'Starting from variable 34
2nnnd CVARCNT.A% = 4               ' to variable 37 (34+4-1)
2nnne C8OVARS.A(0)=650             'Send 650 to variable 34
2nnnf C8OVARS.A(1)=0               'Send 0 to variable 35
2nnng C8OVARS.A(2)=0               'Send 0 to variable 36
2nnnh C8OVARS.A(3)=0               'Send 0 to variable 37
2nnni GOSUB 100                    'Call AMLECOMM
```

This example will send value 650 to variable 34, and 0's to variables 35-37. In order to determine which AML/E program variables to change, a symbol table must be created during compilation of the program. The XREF program can then be run on the .SYM file created to give a listing of the variables' numbers. See "XREF Program" on page 2-34 and "XREF Program" on page 4-89.

Data Drive Operation

```
2nnna OPERATION.A$ = D.A$          'Specify "Data Drive" operation
2nnnb DDSWITCH.A% = ON.A%          'Specify turn on data drive mode
2nnnc GOSUB 100                    'Call AMLECOMM
```

This example causes the controller to become the communications initiator by placing the controller in the'Xon state. The host must now be ready to respond to requests from the controller to avoid communication timeouts. The return code in RTRN.A% will either be a 0 (all was well and data drive mode has been entered), or 19 (the controller has sent an Eot, and data drive mode has not been entered). The latter will occur when data drive mode was entered too late, and the controller is already in the Xto state. This occurs when the running controller application hits a GET, PUT or DEBUG breakpoint, and the controller is in the Xoff state. The controller enters the Wxo state (Waiting for Xon), and unless an Xon is received within 30 seconds, the Xto mode is entered. The user host application then tries to use the above AMLECOMM call to enter data drive mode, but it is too late. When the return code of 19 is received, a single X record (RSTERR.A$) will place the controller application back in the Wxo state, and then the above AMLECOM call can be used to enter data drive.

When using data drive, the user application sees one of two possible AMLECOMM states (as indicated by STATUS.A%). When the Xon is sent to

the controller, AMLECOMM enters the PENDING.A% state. AMLECOMM will stay in the PENDING.A% state until one of two events occurs:

1. The user application shuts off data drive by setting DDSWITCH.A% to OFF.A% and calling AMLECOMM. This would return the user to the IDLE.A% state, and reinstate the host has the initiator of communications.

2. A record request is received from the controller signifying the start of a controller initiated data drive request. When this occurs AMLECOMM enters the READY.A% state. Depending on the D record received from the controller, the host has a different amount of time to respond before a TE (Transmission Error) results. For a GET request the host has 30 seconds from when the D record is sent, for a PUT request or DEBUG encountered notification the host has three seconds.

The user application should poll AMLECOMM every 1 second to see if the STATUS.A% has changed from PENDING.A% to READY.A%. When the user application notices that AMLECOMM is in the READY.A%, state, the data drive request initiated by the controller is given in RTRN.A$ (which is set to GET.A$, GETC.A$, PUT.A$, or DEBUG.A$). The user application then calls AMLECOMM one more time (without changing the value in RTRN.A$) and the data drive operation is completed, and AMLECOMM returns to the PENDING.A% state.

Note: AMLECOMM supports the GETC instruction of AML/Entry Version 3. This instruction is no longer available in AML/Entry Version 4, having been replaced by the GET instruction.

The following example illustrates how to complete each data drive transaction. The example simply waits for AMLECOMM to enter the READY.A% state or encounter an error. Using ON COM as described with X records could also be used to recognize when the controller has sent the D record that initiates a data drive transaction and places AMLECOMM in the READY.A% state.

```
2nnna OPERATION.A$ = D.A$          'Specify data drive operation
2nnnb DDSWITCH.A% = ON.A%         'Keep data drive on
2nnnc GOSUB 100                   'Call AMLECOMM to turn on Data Drive
2nnnd WHILE STATUS.A%=PENDING.A% AND RTRN.A%=0
2nnne     GOSUB 100               'Call AMLECOMM for DD request
2nnnf     WEND
2nnng IF RTRN.A%<>0 THEN PRINT "ERROR in Data Drive": GOTO 2nnnd
2nnnh IF RTRN.A$=GETC.A$ THEN GETCVAR.A=10*DD0FFS.A% : GOSUB 100
2nnni IF RTRN.A$=DEBUG.A$ THEN GOSUB 100
2nnnj IF RTRN.A$<>PUT.A$ THEN 2nnnq       'Handle PUT request
2nnnk    FIRST%=DDOFFS.A%:NUM%=DDCNT.A%  'Save offset and count
2nnnl    GOSUB 100                        'Receive vars from robot
2nnnm    WHILE NUM%<>0                    'Print each value received
2nnnn       PRINT "VAR# ";FIRST%;" = ";PVARS.A(FIRST%)
2nnno       FIRST%=FIRST%+1 : NUM%=NUM%-1
2nnnp       WEND
2nnnq IF RTRN.A$<>GET.A$ THEN 2nnnd       'Handle GET request
2nnnr    FOR I=0 TO DDCNT.A%-1            'Load vars to send
2nnns       GVARS.A(DDOFFS.A%+I)=I        'Send 0,1,2,3....
2nnnt       NEXT I
2nnnu    GOSUB 100                       'Call AMLECOMM to send value
2nnnv    GOTO 2nnnd                       'Handle next request
```

If an error occurs while polling for the READY.A% state, then RTRN.A% will be set to a non-zero return code, but AMLECOMM will remain in the PENDING.A% state. Thus an application must loop until a non-zero return code is received or the READY.A% state is entered. When the READY.A% state is entered, the above example tests RTRN.A$ to see which request the controller has initiated.

For a GETC request, the user application can inspect DDOFFS.A% to see which variable the application requests. The user application then places the value to be sent for this variable in GETCVAR.A, and calls AMLECOMM. In the above example, the value sent is 10 times the variable number. For example AMLECOMM would data drive a 70 for a GETC for variable 7.

For a DEBUG encountered notification, nothing needs to be done except to call AMLECOMM once again to complete the protocol.

For a PUT the values to be sent are specified by the starting variable number in DDOFFS.A% and the number to be sent in DDCNT.A%. The user application should save the values of these variables before calling AMLECOMM again for a reason to be discussed momentarily. The user then calls AMLECOMM again and the values sent by the controller will be placed into PVARS.A in the corresponding locations. For example, if variables 34-43 are sent via a PUT, the values would be placed in PVARS.A(34) through PVARS.A(43). This is called "memory map". Note this is different from the technique use by RVARS.A for the read variables request.

For a GET the user must load the values to be sent into the array GVARS.A using a memory map. Thus if the controller requests 10 variables starting from variable number 50, the user application would have to load the appropriate values into GVARS.A(50) through

GVARS.A(59). After the values are loaded, the user application calls AMLECOMM and the values are sent to the controller.

When calling AMLECOMM in the READY.A% state, the user application must not change RTRN.A$, because this tells AMLECOMM which data drive protocol to use. This is why RTRN.A$ is shown as an output variable in the first half of the data drive protocol and as an input variable in the second half of the data drive protocol in the table earlier in this section. When AMLECOMM is called in the READY.A% state, the protocol is completed and AMLECOMM will either return to the PENDING.A% state or remain in the READY.A% state (if another data drive request has been made immediately by the executing controller application). This is why, in the example code shown for a PUT transaction, the values for DDCNT.A% and DDOFFS.A% must be saved in NUM% and FIRST% -- if after completing the PUT protocol, the next request is immediately received, then the new values may overwrite the old ones in DDCNT.A% and DDOFFS.A%. Note that the example may actually perform both a PUT and a GET before the loop at 2nnnd is re-entered.

Note: If a PUT command is requested which accesses variables beyond VARMAX.A, then the variables beyond VARMAX.A are not saved in PVARS.A. It is the responsibility of the application to inspect DDOFFS.A% and DDCNT.A% to ensure they are within VARMAX.A. Likewise, if a GET command is requested which accesses variables beyond VARMAX.A, then only the variables up through VARMAX.A are sent a value. The remainder of the variables are not sent values. This will cause a transmission error (TE) that may be reset by using the X 13 request.

To end the data drive mode of communications and reinstate the host as the communications initiator, the following should be performed:

```
2nnna OPERATION.A$ = D.A$        'Specify "Data Drive" operation
2nnnb DDSWITCH.A% = OFF.A%       'Specify turn off data drive mode
2nnnc GOSUB 100                  'Call AMLECOMM
```

AMLECOMM requires level 15 or 16 microcode in order to shut off data drive properly. With this microcode, the above code will either return a return code of 0 (all went well, data drive mode has exited), or 19 (the controller has sent an Eot). A return code of 19 means that the controller had already sent a D record, and the user application did not call AMLECOMM quickly enough to complete the protocol (30 seconds for a GET, nine seconds for a PUT or DEBUG (three seconds per try, two retries)). When this happens, the above code will place AMLECOMM back into the IDLE.A% state, and X records can now be used to clear the error. Two consecutive X record requests (RSTERR.A$ followed by STARTCY.A$) will clear the error and restart the controller application. The controller application will then immediately enter the Wxo state (waiting for Xon), and the data drive transaction that failed will be retried as soon as an Xon is sent.

With pre-15.1 level microcode, the user application must only exit data drive when the controller application will not perform a simultaneous data drive request. Thus the controller application should include several "windows" consisting of three to four consecutive DELAY statements, and the cell application running on the host must exit data

APPENDIX A. COMMAND/KEYWORD REFERENCE

This appendix provides an alphabetic listing of Edit commands, AML/Entry commands and AML/Entry keywords. Each command and keyword is described briefly with its format and purpose. AML/Entry command descriptions include a category called "Manipulators." The type of command is also stated. Examples are included where applicable.

> **Note:** In this appendix, the AML/Entry program examples are, except where marked, written for use on a manipulator with a Home position of (650,0,0,0). If the Home position on the manipulator is different, you may need to change the points in the examples to produce valid results.

AML/Entry commands are instructions to the system that usually resemble words in the English language. Commands are included in an AML/Entry statement and often contain parameters enclosed in parentheses, and are always ended with a semicolon.

AML/Entry keywords are used to define constants, reserve storage for variables, invoke arithmetic functions, and identify the beginning and ending of a subroutine.

Editor commands aid in the creation, modification, and management of AML/Entry programs on the Personal Computer. Editor commends are classified as Line Edit commands and Primary Edit commands. Eine Edit commands are entered to the left of the line numbers on the screen. Primary Edit commands are entered following the COMMAND INPUT ---> prompt of the editor.

Refer to Chapter 3, "Editor" in this manual for detailed descriptions of the editor commands. Chapter 4, "A Manufacturing Language/Entry", contains detailed descriptions of all AML/Entry commands, keywords, and rules for AML/Entry statements.

Line Edit Command

Format: A

Purpose: This line command designates where the copied or moved line(s) are put when a copy or move command executes. The text is placed after the line where the A is placed. The command is used in conjunction with the line commands C , CC , M , or MM .

AML/Entry Arithmetic Function

Format: ABS(expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the absolute value of an expression. The absolute value of a number is defined as the positive value of the number. For numbers which are already positive, the ABS function returns the number itself. For numbers which are negative, the ABS function returns -1 multiplied by the number,

Remarks: This function is useful for determining how close one expression is to another. For example, to compare two counters which contain real values, the ABS of the difference of the counters should be compared to a small, positive number (i.e. 0.0001). See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
ABS(0)       = 0
ABS(3.2)     = 3.2
ABS(3150)    = 3150
ABS(-1)      = 1
ABS(-2.999) = 2.999
COMPC(ABS(A-B) < .0001,EQUAL);   --Checks to see if the
                                 --counters A and B are
                                 --within .0001 of each
                                 --other. If so, branch
                                 --to the label EQUAL.
```

AML/Entry Arithmetic Function

Format: `ATAN(expression)`

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the arctangent of an expression. The arctangent of a number is defined as the angle (in degrees) whose tangent is the number given. The result will always be in the range of -90 through 90.

Remarks: The tangent of an angle is equal to the sine of the angle divided by the cosine. The arctangent "undoes" the tangent function. The ATAN2 function performs a similar function, except a value in the range of -180 through 180 is returned. See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
ATAN(0)           = 0      --Thus TAN(0)=0
ATAN(1)           = 45     --Thus TAN(45)=1
ATAN(-SQRT(3)/3) = -30     --Thus TAN(-30)=-SQRT(3)/3
```

ATAN2

AML/Entry Arithmetic Function

Format: ATAN2(expression1,expression2)

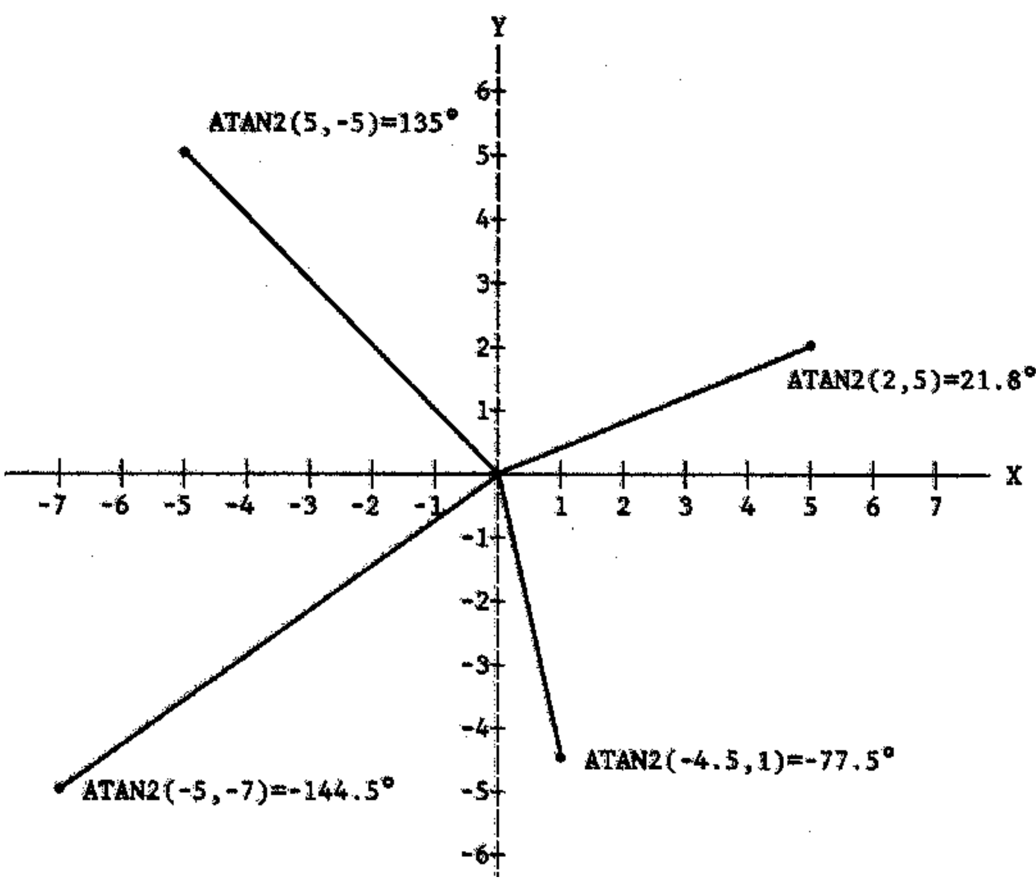
Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the arctangent of expression1 divided by expression2. The arctangent of a number is defined as the angle (in degrees) whose tangent is the number given. Thus the tangent of the result of the ATAN2 function will give expression1/expression2. The result will always be in the range of -180 through 180.

Remarks: The tangent of an angle is equal to the sine of the angle divided by the cosine. The arctangent "undoes" the tangent function. The ATAN2 function essentially uses expression1 as the sine value, and expression2 as the cosine value. In general, however, any Y coordinate can be used as expression1 and any X coordinate as expression2. The result of the ATAN2 function will then be the angle made by the line segment (0,0)-(X,Y) and the X-axis. The angle will have the same sign as expression1. If both expression1 and expression2 are 0, then a run-time AML/Entry error occurs. If COMAID is used to perform a R 01 (read machine status), the error code will be Hex 34 (Invalid arguments for ATAN2 function). See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
ATAN2(1,-1)       = 135       --Thus TAN(135)=1/-1
ATAN2(3,3)        = 45        --Thus TAN(45)=3/3
ATAN2(-.5,-SQRT(3)/2) = -150 --Thus TAN(-150)=-.5/(-SQRT(3)/2)
ATAN2(-7,0)       = -90
ATAN2(0,0)        = AML/Entry Error
```



Graphic Representation of the AT 42 lFuntt ion

Line Edit Command

Format: B

Purpose: This line command designates where the copied or moved line(s) are put when a copy or move command executes. The text is placed before the line where the B is placed. The command is used in conjunction with the line commands C , CC , M , or MM.

BRANCH

AML/Entry Command

Format: BRANCH(label);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This instruction unconditionally transfers control to the program line containing the specified label.

Remarks: The label must appear in the same subroutine as the BRANCH statement.

Example: As shown in the example, the branch statement returns control to the statement that checks for a part in the feeder. The statement with the GO label is not executed until the test at DI point 8 indicates that a part is present. The program then transfers control to the statement with the label GO, avoiding the branch statement.

```
FEEDER:PMOVE (PT (-400,400,-80,180));
CHECK:TESTI(8,1,GO);            --TEST FOR PART PRESENT
      WRITEO(3,1);              --OPEN FEEDER GATE
      DELAY(2);
      WRITEO(3,0);              --CLOSE FEEDER GATE
      BRANCH(CHECK);            --RETURN TO CHECK
   GO:ZMOVE(-250);
```

BREAKPOINT

AML/Entry Command

Format: BREAKPOINT;

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This instruction allows the operator at the control panel to interrupt the application program at a specific point and then resume the application from that point at a later time. If the **Stop and Mem** key has been pressed, the controller stops executing the program at the next BREAKPOINT instruction or at the last END keyword for the application program.

Remarks: The operator uses the **Stop and Mem** key and the **Recall Memory** key with this feature. You can use this instruction in an application program where you may want to stop the application without completing the cycle. Multiple BREAKPOINT commands can be provided in the program.

When you use the **Recall Memory** key to resume an application after Manip Power has been turned off, remember that the manipulator starts at the home position, with the Z-axis shaft up and the optional gripper open. If the statements previous to a BREAKPOINT involved DOWN commands or closing the optional gripper, the controller does not remember those commands occurred.

When using BREAKPOINT commands, the important things to remember are:

- DOWN or GRASP commands are not retained during the power down and future power up.
- DO states are not retained during BREAKPOINT if Manip Power is turned off. All DO's become inactive.
- If you execute the BREAKPOINT command without removing Manip Power you can restart execution at the place it was broken off. However, if Manip Power is turned off the manipulator must be returned to the Home position before execution can resume.

Example:

Assume that an application puts 10 identical parts into a carton. A simple way to program this application is to write a subroutine M that executes repeatedly. If it is occasionally necessary to stop the application before the carton is loaded, you may want to resume the application and load the remaining parts. Inserting a BREAKPOINT command in M allows you to do this.

```
COMMAND INPUT -->

    1 DEMO:SUBR;
    2    M:SUBR;
    3       PMOVE (PT (300,400,-100,0));
    4 -
    5 -
    6 -
    7       BREAKPOINT;
    8 END;
    9       M;
   10 END;
```

C

Line Edit Command

Format:

Purpose: This line command copies a single line to a location following the line where the line command A appears or before the line where the line command B appears. The line numbers are put in numeric order when the command is executed.

Remarks: Type the A on the line you want the copied line to follow and press the enter key. Or, type B on the line you want the copied line to come before and press the enter key.

CANCEL

Primary Edit Command

Format: CANCEL

Purpose: This primary command allows you to exit the editor without saving the current program on the diskette.

Remarks: You may want to use this command in the following cases:

- You want to exit the editor without saving any information from the editing session on diskette.
- You entered the editor without specifying a file to edit.
- You deleted the file you are editing from the diskette, and you want to prevent the copy in the editor from being saved.
- You renamed the file that you are editing and want to prevent the copy in the editor from being saved under the old file name.
- Lines were truncated to 72 characters by the editor, and you want to cancel the editing session to leave the original file intact.

CAPS

Primary Edit Command

Format: CAPS

Purpose: This primary command allows you to convert lowercase characters to uppercase characters.

Remarks: This command converts AML files created with other editors to uppercase. This is necessary if the AML file is to be modified in the AML/Entry editor and some of the primary commands are used during the edit session (such as CHANGE or FIND).

Line Edit Command

Format: CC

Purpose: This line command designates the first or last line of a block of lines to be copied. The line numbers are put in numeric order when the command executes.

Remarks: The command must be used in conjunction with either the **A** or the **B** line command and another CC.

Note: The **A** or the **B** command cannot be placed between the two CC commands.

CHANGE

Primary Edit Command

Format: C /stringl/string2/ [col-1] [col-2] [ALL]

Purposes This primary command allows you to search for a particular character string in the file currently being edited and change it to another string of characters.

Remarks: This primary command changes a character string located between col-1 and col-2 to another string of characters. Embedded blank characters are permitted in the strings. The search for the string begins at the top line of the program window and continues to the end of the program.

The required parameters are stringl and string2, preceded and followed by the delimiter character / (slash). There must be three delimiter characters present, as shown above. Stringl and string2 may not contain the slash character. If a slash is not used, the delimiter character is assumed to be a blank and the command still works. The col-1 and col-2 parameters in the format limit the changes to the characters within the area specified by the two boundaries. The col-1 parameter can be specified without the col-2 parameter, but col-2 can not be specified without col-1. If you enter the col-1 parameter without the col-2 parameter, the search begins in the first column specified and continues to the end of the line. If omitted, the entire screen is searched (all 72 columns).

The ALL parameter specifies that the change occur every place in the program from the top of the program window to the end of the file. It may be specified alone or with the column parameters. If you do not specify the ALL option, only the first occurrence of the desired change takes place.

You can enter an abbreviated command, as shown in the format, or use the entire command.

You can repeat a change by using the F5 key.

Example: An example of a Change command follows.

```
COMMAND INPUT --> C /POINT/PT/ ALL
     lelkAA''''AANA
     1       P1:NEW POINT(300,300,0);
     2       P2:NEW POINT(0,650,0,0);
     3       GOHOME:SUBR;                             --MOVE HOME
     4         PMOVE(POINT(650,0,0,0));               --HOME FOR 7545
     5         END;
     6       PMOVE(P1);                               --GO TO POINT 1
     7       PMOVE(P2);                               --GO TO POINT 2
     8       GOHOME;                                  --GO HOME
     9       END;
```

RESULT:

```
COMMAND INPUT -->
     *************
     1       P1:NEW PT(300,300,0);
     2       P2:NEW PT(0,650,0,0);
     3       GOHOME:SUBR;                             --MOVE HOME
     4         PMOVE(PT(650,0,0,0));                  --HOME FOR 7545
     5         END;
     6       PMOVE(P1);                               --GO TO POINT 1
     7       PMOVE(P2);                               --GO TO POINT 2
     8       GOHOME;                                  --GO HOME
     9       END;
```

COMPC

AML/Entry Command

Format: `COMPC(expression1 condition expression2,label);`

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command allows you to compare a mathematical expression to another mathematical expression. See "Expressions" on page 4-48 for a discussion of expressions.

Remarks: If the condition is met, the statement branches to the specified label. Allowable conditions are listed below. Enter exactly as shown.

Condition	Meaning
LT or <	less than
LE or <=	less than or equal to
GT or >	greater than
GE or >=	greater than or equal to
EQ or =	equal to
NE or <>	not equal to

Do not enter '=<' for '<=', '=>' for '>=', or >< for <>; the order is important.

When comparing real-valued expressions for equality or inequality, small round-off errors could cause unexpected results. It is better to use the ABS function to see if the two expressions are within a small amount (i.e. 0.0001) of each other.

For the compiler to be able to discern the alphabetic operators from their surrounding expressions, the alphabetic operators must be surrounded by at least one blank one each side. The following statements are identical:

```
COMPC(CTR EQ 0,CONTINUE);
COMPC(CTR=0,CONTINUE);
```

Example: An example of a COMPC command is shown below.

```
MAIN: SUBR;
   CARD_POINT :   NEW PT(0,450,0,0);
   CARD_POINT2  : NEW PT(0,350,0,0);
   STOP_POINT :   NEW 5;      -- the DI point to guard
   NEW RACE :     NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);         -- move to the start point
   GUARDI(STOP_POINT,1):      -- guard for the stop point
   PAYLOAD(11);
   LINEAR(1);
   PMOVE(CARD_POINT2);        -- move to the new point
   COMPC(MSTATUSO<>0,HERE);-- if stopped by guard
                              - send point to host
   NOGUARD;                   -- disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW_PLACE);       -- read the stop location
   PUT(NEW_PLACE);            -- send location to host
EN:                           -- the remainder of the program
   END;
```

AML/Entry Arithmetic Function

Format: COS(expression)

Manipulators: This command is applicable to all systems using An/Entry Version 4.

Purpose: This arithmetic function returns the cosine of an expression.

Remarks: This function is useful for making the manipulator move in a circle. A circle is described by the following parametric equations:

```
X=X0 + R*COS(THETA)
Y=Y0 + R*SIN(THETA)
```

The center of the circle is at (X0,Y0), the radius is R, and THETA is a parameter that traces a circle as it is varied from 0 to 360 degrees. See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
COS(0)      = 1
COS(45)     = SQRT(2)/2
COS(-90)    = 0
```

AML/Entry Keyword

Format: name:STATIC COUNTER;

Purpose: This keyword defines a name to be a counter.

Remarks: A counter can hold either an integer or real value, but all integers are stored in real number format. The range for a counter is approximately from -9.2E18 to +9.2E18. As long as a counter is assigned an integer value (or an expression consisting only of integer constants, integer arithmetic functions, and the operators +, -, and *), then the exact value will be stored. Counters which hold real values are subject to a small round-off error which can almost always be ignored. The round-off error only comes into play when a COMPC or TESTC command is executed.

When your program is loaded, the initial value for the counter is 0. The counter's value can be changed during program execution by the counter commands INCR, DECR, GET, and SETC.

A counter retains its last value from one invocation of the program to the next. You can restore the starting value of the counter by one of these methods:

- Reloading the application program from the Personal Computer.
- Using a digital input to activate a subroutine in your application program to set the counter to a starting value when required.
- Using the host initiated data drive (option C "80") from Comaid.

Example: CTR1 in the example is a global counter used to build 200 parts before going to the next assembly process (which is not shown). In this application each time a starting counter value is desired, DI 16 receives an input to set the counter to 0. The program branches to line 6 of the program if line 4 does not receive a DI 16 signal with a value of 1.

```
COMMAND INPUT -->
     *************
    1      CTR1:STATIC COUNTER;
    2     START:SUBR;
    3       SET:SUBR;
    4            TESTI(16,0,NOCHANGE) ;     IS DI 16 ON
    5            SETC(CTR1,0);           --SET COUNTER TO ZERO
    6 NOCHANGE:                          --BYPASS RESET
    7            END;
    8            SET;     --CALL SET SUBROUTINE FOR COUNTER
          PART1:
   10            SETC(CTR1,CTR1+1);      --ADD 1 TO THE COUNTER
   11
   12               PARTS ASSEMBLE STATEMENTS
   13            -- PARTS ASSEMBLE STATEMENTS
   14
   15            COMPC(CTR1 NE O,PART1);--TEST COUNT
   16     END;
```

CSTATUS

AML/Entry Command and Arithmetic Function

Format (command): CSTATUS (count er_name) ;

Format (arithmetic function): CSTATUS()

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command/function allows you to determine if the controller is able to initiate a data transfer in an AML/Entry program.

Remarks: The CSTATUS command is used to monitor communication system status. The command form assigns the CSTATUS to a counter. The arithmetic function form returns a value which can be used in an AML/Entry expression. See "Expressions" on page 4-48 for a discussion of expressions. The following commands show how the different forms of CSTATUS may be used to attain identical results.

```
CSTATUS(counter_name);
SETC(counter_name,CSTATUS());
```

The advantage of using the arithmetic function form is that the CSTATUS may appear immediately in a TESTC or COMPC, because these commands allow expressions. The command form requires a counter be declared to hold the CSTATUS value.

The value returned into a counter when a CSTATUS is performed represents the state of the communication line, as outlined below.

```
     15 = controller enabled to communicate
 Not 15 = controller not enabled to communicate
```

The below listed conditions must be present before a 15 is returned.

- Host Connection (DSR) Active
- Communications Cable Connected
- Controller On-Line
- Controller In Xon'ed State

Example 1: An example of a CSTATUS command is shown below in a program fragment.

```
  TOP: WRITEO(COM_CK,1);
       TESTC(CSTATUS(),15, GOOD1); -- IS THE CONTROLLER
       WAITI(OP_OK,1,0);            -- ABLE TO COMMUNICATE?
GOOD1:                              -- CONTINUE PROGRAM
```

Example 2: The following program fragment shows how the CSTATUS function can be used to ensure data drive will be performed by a GET command.

```
LBL: COMPC(CSTATUS() NE 15, LBL);
     GET(PT1);
```

Line Edit Command

Format:

Purpose: This line command deletes the single line where the command has been entered. The line numbers are put in numeric order when the command is executed.

Line Edit Command

Format: DD

Purpose: This line command designates the first or last line of a block of lines inclusively to be deleted. The line numbers are put in numeric order when the command is executed.

Remarks: Use this command in conjunction with another CO.

AML/Entry Command

Format: DECR(counter_name);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command decrements the named counter by the value of 1.

Remarks: The counter name can be passed to a subroutine as a formal parameter; if it is, the **DECR** command does not change the value of the calling argument. In other words, the counter only changes its value in the subroutine it is passed to. The counter does not change its value outside the subroutine it is passed to.

The following two statements are identical.

```
DECR(counter_name);
SETC(counter_name,counter_name-1);
```

Example:

CTR1 in the example is a counter to build 200 parts before going to the next assembly process (which is not shown). In this application, each time a starting counter value is desired, DI 16 receives an input to set the counter to 200. The program branches to line 6 of the program if line 4 does not receive a DI 16 signal with a value of 1. Line 9 of the program reduces the counter each time partl subroutine is called. Line 15 tests to determine if 200 partl parts have been built, allowing a branch to part2.

```
COMMAND INPUT -->
     *************
   1     CTR1:STATIC COUNTER;
   2    START:SUBR;
   3      SET:SUBR;
   4          TESTI(16,0,NOCHANGE);     -- IS DI 16 ON
   5          SETC(CTR1,200);     --SET COUNTER TO 200
   6          NOCHANGE:     --BYPASS COUNTER RESET
   7          END;
   8    PART1:SUBR;         --SUBROUTINE FOR PART1
   9          DECR(CTR1);      --REDUCE COUNTER BY ONE
  10                 PART1 ASSEMBLY STATEMENT
  11          END;
  12          SET; --CALL SET SUBROUTINE FOR COUNTER
  14 CONTINUE:PART1; --CALL SUBROUTINE FOR PART1
  15          TESTC(CTR1,0,PART2); --IS COUNTER ZERO
  16          BRANCH(CONTINUE);       --LESS THAN 200
  17    PART2:
  18              --PART2 STATEMENTS
  19      END;
```

Primary Edit Command

Format: DEL device:filename.filetype

Purpose: This primary command deletes the specified file from a diskette (or fixed disk drive if your system has one).

Remarks: In the special case where you delete a file from the diskette that you are currently editing, you should use the **CANCEL** command to exit the editing session. This action prevents the copy in the editor from being saved on the diskette. The DEL command should **not** be used to delete any of the AML/E system files.

DELAY

AML/Entry Command

Format: DELAY(seconds);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This instruction delays the execution of the next statement for a specified time.

Remarks: The instruction should be used when instructing a slow auxiliary device, such as a gripper, to open or close.

The delay allows the mechanical motion of those devices to complete before making another move. The time range for the delay is 0 seconds to 25.5 seconds in increments of 0.1 seconds. The seconds can be a simple constant or a counter.

Example: This example uses the DELAY command to allow the gripper to complete its movement before the ZMOVE command.

```
 *********************
1 PICKUP: SUBR;       --SUBROUTINE TO PICKUP PART
2     PMOVE(PT(250,300,0,0));    --MOVE TO PART BIN
3     ZMOVE(-250);
4     GRASP;                              --PICKUP PART
5     DELAY(1); --DELAY TO ALLOW GRIPPER TO CLOSE
6     ZMOVE(0);
7 END;
```

AML/Entry Command

Format: DPMOVE(<x,y,z,r>);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4. The change in coordinates is in terms of X, Y, Z, and R.

Purpose: The DPMOVE command is used to move the arm by a small amount in a specified direction. DPMOVE differs from PMOVE because PMOVE moves to a specified location, and DPMOVE moves a specified distance. The argument supplied to DPMOVE is a aggregate. The aggregate represents the change in position for the arm. The change is in the X, Y, Z, and R directions.

Remarks: The DPMOVE command interprets the aggregate in the manipulator coordinate frame. Because it is difficult to establish the precise orientation of this frame, DPMOVE should not be used in an application that requires accurate positioning of the arm.

Note: The DPMOVE command can not be used within an ITERATE statement. Also, the aggregate needed by DPMOVE can not be passed as a parameter to a subroutine.

Example:

The following examples show the DPMOVE command.

The first DPMOVE has offsets of 2 in the X direction, 5 in the Y direction and 0 in the Z and R directions. The second movement is perpendicular to the first, and has offsets of -5 in the X direction, 2 in the Y direction and 0 in the Z and R directions.

```
DELTA_ROLL: NEW 0;
VECTOR:     NEW <2,5,0,DELTA_ROLL>;
     DPMOVE(VECTOR);
     DPMOVE(<-5,2,0,DELTA_ROLL>);
```

The following code fragment illustrates a technique for passing arguments to a subroutine to be used in a DPMOVE command. Once passed to the subroutine the arguments are put into an aggregate, and this aggregate is then supplied to the DPMOVE command.

```
DELTA_ MOVE: SUBR( DX, DY, DZ );
    VECTOR: NEW <DX, DY, DZ, 0>;
            DPMOVE(VECTOR);
     END;     -- END OF DELTA MOVE

DELTA MOVE(5.2, WIDTH, -2);    --LATER IN THE PROGRAM
```

AML/Entry Keyword

Format: END;

Purpose: This keyword is required for each **SUBR** keyword used. The keyword indicates the last line of a subroutine.

Remarks: No other AML/Entry statement or label may follow the keyword **END** on the same line. Only Comments are permitted on the same line as the END statement.

FILES

Primary Edit Command

Format: FILES [parameter]

Purpose: This primary command displays a listing of files contained on a diskette.

Remarks: The command may be entered in several ways depending on the type of listing desired. The methods used are the same as those used by DOS. Some methods for entering the command are:

FILES	List all the files on the default drive
FILES A:*.*	List all the files on drive A
FILES *.AML	List all AML/Entry files on the default drive
FILES name?.AML	List all files on the default drive that start with "name" and have one other character in the name
FILES B:*.*	List all the files on drive B
FILES B:*.AML	List all the AML/Entry files files on drive B

FIND

Primary Edit Command

Format: F /string/ [col-1] [col-2]

Purpose: This primary command searches for the first occurrence of the specified string of characters in the text currently being edited. If the string is located in the program, the program scrolls to display the line with the string at the top of the edit window. Blank characters are permitted in the string.

Remarks: This primary command finds a character string located between col-1 and col-2. Embedded blank characters are permitted in the strings. The search for the string begins at the top line of the program window and continues to the end of the program.

The only required parameter is the string, preceded the delimiter character / (slash). The string is terminated by a second delimiter character / (slash) or the last non-blank character. A blank character must follow the Find or F command. If the slashes are omitted, the command still works. If you enter the col-1 parameter without the col-2 parameter, the search begins in the first column specified and continues to the end of the line. If omitted, the entire screen is searched (all 72 columns).

You can enter an abbreviated command, as shown in the format or, use the entire command.

You can repeat a find by using the **F4** key.

Example: An example of a FIND command is outlined below.

```
F PT        --finds the string 'PT'
F /PT (/    --finds the string 'PT ('
```

FROMPT

AML/Entry Arithmetic Function

Format: FROMPT(point,expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the specified coordinate of the given point. The expression will automatically be truncated to an integer. If the resulting integer is 1, then the X coordinate of the point is returned; if it is 2, then the Y coordinate is returned; if it is 3, then the Z coordinate is returned; if it is 4, then the Roll coordinate is returned.

Remarks: If the expression does not equal 1, 2, 3, or 4 (after truncation) then an AML/Entry error occurs. If COMAID is used to perform a R 01 (read machine status), the error code will be Hex 32 (Invalid index for FROMPT function). See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
PT1:NEW PT(400,400,-250,180);
HOME:NEW PT(650,0,0,0);
FROMPT(PT1,1) = 400
FROMPT(PT1,3) = -250
FROMPT(PT1,5) = AML/Entry Error
FROMPT(PT2,SQRT(2)) = 650 (SQRT(2)=1.414 which is
                               truncated to 1)
```

AML/Entry Command

Format: GET(counter name);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command indicates to the host that the controller requires specific data.

Remarks: The controller initiates a data drive operation, and the application program waits until the host satisfies the request. If the host does not respond within thirty seconds, the application program stops and the TE (transmission error) lamp comes on.

If a GET command is executed when the controller is not on-line and connected to the host, then a DE (data error) results. See -- Heading id 'fixit' unknown -- for a discussion of how this can be avoided.

GET is used to get a single counter, a single point, a single counter within a group, or a single point within a group as well as a group of counters or points.

Example: An example of a GET command is shown below.

```
MAIN: SUBR;
   P : NEW PT(0,0,0,0);          -- a generic point
   INSTRS : STATIC GROUP(P,P);
   FLAG : STATIC COUNTER;
   LOOP :
      GET(FLAG);
      COMPC(FLAG=0,DONE);           flag=0 means all done
      GET(INSTRS);               -- get instructions from host
                                    INSTRS is 2 counters
                                 -- 1st tells the start point
                                 -- 2nd tells the end point
      PMOVE(INSTRS(1));          -- move to start point
      PMOVE(INSTRS(2));          -- move to end point
      BRANCH(LOOP);
DONE:
                                 -- continue with the program
   END;
```

GETFILE

Primary Edit Command

Format: GETFILE filename

Purpose: This primary command allows you to insert other files into the file currently being edited.

Remarks: In addition to filename (if no extension is entered, the Editor assumes .AML), you must specify where the included text is to be placed. This is accomplished by entering an A (after) or a B (before) in the line command area. A or B must be specified before the GETFILE command is entered on the primary command line and the <--(enter) key is pressed. If either A or B is not specified when GETFILE is entered, an error results and the command must be re-entered (unless, the ? command is used to recall the command). Errors are also issued for an improper file name, file not found, or file name not specified. In all cases, whatever was entered in the line command area remains both visible and active.

Example: An example of a GETFILE command is shown below.

```
--STATION1.AML - - - - - - - - 07-08-1985 - - - - - - - - - - - -
COMMAND INPUT ---> GETFILE STATION2.AML
 - - **********************> Top•o•ILE <***********************
     1 EASY TO PROGRAM
     2 EASY TO PROGRAM
A 3 THE IBM MANUFACTURING SYSTEM
     4 PICK AND PLACE ROBOT
     5 IS VERSATILE
     6 EASY TO PROGRAM
     7 THE IBM MANUFACTURING SYSTEM
     8 PICK AND PLACE ROBOT
     9 IS VERSATILE
 - - **********************> BaTom•u•FILE <********************
```

This example copies the contents of the STATION2.AML file into the STATION1.AML file after line 3.

GETPART

AML/Entry Command

Format: GETPART(name);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This palletizing command moves the manipulator to the current part of the specified pallet.

Remarks: When GETPART is executed, the manipulator will move to the location specified by the current part number of the given pallet. The X and Y coordinates depend on the current part number. The Z coordinate is not changed from its current position. The Roll coordinate is the Roll coordinate for the lower left point of the pallet.

The pallet does not have a current part number when the program is loaded into the controller. Your program controls the current part counter by using the other pallet commands:

- name:STATIC PALLET(ll,lr,ur,ppr,parts);
- NEXTPART(name);
- PREVPART(name);
- SETPART(name,value);
- TESTP(name,value,label);

Example: In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** keywords use all the above information, as shown in the example. The counter for the pallet is set in line 8. The move to the current part position is in line 9. Line 13 increments the current part number. Line 14 checks to see if the current part is back at the first part. If so, the program branches to the label "LOOP."

```
COMMAND INPUT -->
     *************

  1      LL:NEW PT(300,200,0,0);
  2      LR:NEW PT(300,300,0,0);
  3      UR:NEW PT(500,300,0,0);
  4     PPR:NEW 8;                  --PARTS PER ROW
  5  PARTS:NEW 32;              --NUMBER OF PARTS
  6  POINTS:STATIC PALLET(LL,LR,UR,PPR,PARTS);
  7  MAIN:SUBR;
  8        SETPART(POINTS,1);       --SET COUNTER
  9  LOOP:GETPART(POINTS);          --PALLET MOVE
 10        ZMOVE(-250); GRASP; DELAY(1); ZMOVE(0);
           --PICK UP PART
 11        PMOVE( PT(250,300,0,0)); --MOVE FROM PALLET
 12        ZMOVE(-250); RELEASE; DELAY(1); ZMOVE(0);
           --DROP OFF PART
 13        NEXTPART(POINTS);        --CHANGE COUNTER
 14        COMPC(TESTP(POINTS) NE 1,LOOP);
 15        WAIT(16,1,0);            --WAIT FOR INPUT
 16  END;
```

GRASP

AML/Entry Command

Format: GRASP;

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This instruction closes digital output (DO) point 2. In an unmodified system, DO point 2 controls the air supply for a gripper.

Remarks: A delay instruction may be required following the **GRASP** instruction to allow the mechanical motion of a gripper to complete before the next move.

Example: This example uses the GRASP command to allow the gripper to pick up a part.

```
      *********************
    1 PICKUP: SUBR;               --SUBR TO PICKUP PART
    2     PMOVE(PT(250,300,0,0)); --MOVE TO PART BIN
    3     ZMOVE(-250);
    4     GRASP;                  --PICKUP PART
    5     DELAY(1);        --DELAY SO GRIPPER CAN CLOSE
    6     ZMOVE(0);
    7 END;
```

GROUP

AML/Entry Keyword

Format:

```
GROUP(PT1,PT2,...PTn);
GROUP(value,value,...value);
```

Manipulators: This keyword is applicable to all systems using AML/Entry Version 4.

Purpose: This keyword allows you to group data together and then refer to individual elements using an index.

Remarks: A group must be defined as STATIC. It consists of either points, single-valued counters, or constants. A group consists of only one type of entity, a group of points or a group of counters. The index into a Group is 1 based. For example, 'INCR (COUNT(0));' would cause an error. All elements of a Group must contain initial values. A group must contain at least one element. The maximum number of elements is limited only by available controller memory.

Example 1: An example of a GROUP keyword is shown below.

```
   POINT1: NEW PT(0,500,0,0) ;        -- location of fixture 1
   POINT2: NEW PT(0,600,0,0) ;        -- location of fixture 2
   POINT3: NEW PT(50,500,0,0) ;       -- location of fixture 3
 FIXTURES: STATIC GROUP (POINT1,POINT2 ,POINT3) ;
    INDEX: STATIC COUNTER ;           -- the index into fixtures

     MAIN: SUBR ;
           SETC (INDEX,1) ;            – start at the first fixture
      TRY: PMOVE (FIXTURES(INDEX)) ; - - move to fixture
           INCR (INDEX) ;              – go to next fixture
           TESTC (INDEX,4,EN) ;        – moved to all 3 fixtures
           BRANCH (TRY) ;              – if not move again
       EN:                            -- the rest of the program
           END;
```

Example 2: When an individual point or counter of a group is referenced, an index must be given. The only time a group may be referenced without an index is in the GET or PUT commands. For example, suppose a global group of 4 counters is used to declare a point, PT1. The following would be have to be used:

```
GR:STATIC GROUP(650,0,0,0);
PT1:NEW PT(GR(1),GR(2),GR(3),GR(4));
```

It is tempting to want to use GR without indices, but this will cause a compiler error. Each time the program cycles back to the beginning of the program, PT1 will get reassigned new values based on the current values of GR.

GUARDI

AML/Entry Command

Format: `GUARDI(digital_input_port,value);`

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command allows you to treat a DI port as a motion guard. It provides the ability to interrupt a motion, based on an external input.

Remarks: When motion occurs, the controller checks the digital input (DI) port value. If it attains the specified value, motion is halted. Version 4 allows a DI port to be treated as a motion guard. The manipulator does not stop program execution, but regards the move as completed. The digital_input_port and value can consist of constants, formal parameters, or counters. A value of zero guards for an "open" state of the digital_input_port, a nonzero value guards for a "closed" state.

Only one DI point is used as a motion guard at any one time. Whenever the GUARDI command is encountered, monitoring is changed to the newly-specified point. If a previous value was in effect, it is lost. However, the controller does not lose the current value when it is changed by a subroutine. For example, if GUARDI is set to a particular DI point and value, and a subroutine that changes the guard data is called, the new data is used for monitoring while the subroutine remains active. When the subroutine ends, and control is returned to the caller, the controller automatically restores the caller's guard parameters.

If a move is stopped by a DI point, the system automatically updates its internal location variables so that a DPMOVE executed immediately after a DI guarded motion acts as expected (moves a delta amount from where the arm came to rest.)

Example: An example of a GUARDI command is shown below.

```
MAIN: SUBR;
   CARD_POINT : NEW PT(0,450,0,0);
   CARD_POINT2 : NEW PT(0,350,0,0);
   STOP POINT : NEW 5; -- the DI point to guard
   NEW_PLACE :   NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);         -- move to the start point
   GUARDI(STOP_POINT,1):      -- guard for the stop point
   PAYLOAD(11);               -- set low speed for quick stopping
   PMOVE(CARD_POINT2);        -- move to the new point
   COMPC(MSTATUSO<>0,HERE);-- if stopped by guard
                            - send point to host
   NOGUARD;                   -- disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW_PLACE);       -- read the stop location
   PUT(NEW_PLACE);            -- send location to host
EN:                           -- the remainder of the program
   END;
```

Line Edit Command

Format: I[n]

Purpose: This line command inserts **n** number of blank line(s) following the line where the command is entered. The number substituted for the **n** must be a single digit number; the maximum number for **n** is 9. The number 1 is not needed when a single line is desired. The line numbers are put in numeric order when the command is executed.

AML/Entry Command

Format: INCR(counter name);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command increments the specified counter by 1.

Remarks: The counter name can be passed to a subroutine as a formal parameter; if it is, the **INCR** command does not change the value of the calling argument. In other words, the counter only changes its value in the subroutine it is passed to. The counter does not change its value outside that subroutine.

The following two statements are identical.

```
INCR(counter_name);
SETC(counter_name,counter_name+1);
```

Example: CTR1 in the example is a counter to build 200 parts before going to the next assembly process (which is not shown). In this application each time a starting counter value is desired, DI 16 receives an input to set the counter to 0. The program branches to line 6 of the program if line 4 does not receive a DI 16 signal with a value of 1. Line 9 of the program increases the counter each time the partl subroutine is called. Line 15 tests to determine if 200 partl parts have been built, allowing code execution to fall through to the "PART2" label.

```
COMMAND INPUT -->
     *************
    1     CTR1:STATIC COUNTER;
    2    START:SUBR;
    3      SET:SUBR;
    4          TESTI(16,O,NOCHANGE);     --IS DI 16 ON
    5          SETC(CTR1,0);    --SET COUNTER TO ZERO
    6          NOCHANGE:        --BYPASS COUNTER RESET
    7          END;
    8    PART1:SUBR;            --SUBROUTINE FOR PART1
    9          INCR(CTR1);      --INCREASE COUNTER BY ONE
   10               PART1 ASSEMBLY STATEMENT
   11          END;
   12          SET; --CALL SET SUBROUTINE FOR COUNTER
   13 CONTINUE:
   14          PART1;       --CALL SUBROUTINE FOR PART1
   15          COMPC(CTR1<200,CONTINUE);--IS COUNTER 200
   16    PART2:
   17           --PART2 STATEMENTS
   18           END;
```

ITERATE

AML/Entry Command

Format:

```
ITERATE('command',<aggregate>,<aggregate>,...);

ITERATE('subr',<aggregate>,<aggregate>,...);
```

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This instruction repeatedly executes an AML/Entry command or subroutine with different values until all the values are used in the command or subroutine. The values are put in an aggregate and the aggregate is a parameter of the ITERATE statement. When all values in the aggregate have been used, the next statement executes.

The AML/Entry command or name of a subroutine to be repeated is defined as a character string and enclosed in single quotes ('). The ITERATE command does not allow expressions to appear as arguments. Thus even though an AML/Entry command may allow expressions to appear as arguments, when the command is repeated by an ITERATE, expressions may not be used. In this case, the arguments must be integer constants, real constants, or counters, depending on the particular command.

Example 1:

The PMOVE statement is repeated until all the values in PATH are exercised one time. The values of PATH are P1 through P15.

```
COMMAND INPUT -->
     *************
    1 DEMO:SUBR;
    2   P1:NEW PT(500,400,0,0);
    3   P2:NEW PT(450,400,0,0);
    4   P3:NEW PT(400,400,0,0);
    5   P4:NEW PT(350,400,0,0);
    6   P5:NEW PT(300,400,0,0);
    7   P6:NEW PT(250,400,0,0);
    8   P7:NEW PT(200,400,0,0);
    9   P8:NEW PT(150,400,0,0);
   10   P9:NEW PT(100,400,0,0);
   11 P10:NEW PT(50,400„00);
   12 PH:NEW PT(0,400,0,0);
   13 P12:NEW PT(-50,400,0,0);
   14 P13:NEW PT(-100,400,0,0);
   15 P14:NEW PT(-150,400,0,0);
   16 P15:NEW PT(-200,400,0,0);
   17 PATH:NEW <P1,P2,P3,P4,P5,P6,P7,
   18      P8,P9,P10,P11,P12,P13,P14,P15>;
   19      ITERATE('PMOVE',PATH);
   20      END;
```

Example 2:

This example does the same task as the previous example, but the method of programming shortens the program. A different value of X is substituted in the subroutine SLMOVE. The ITERATE statement loops on the subroutine SLMOVE until all values of X are exercised.

```
COMMAND INPUT -->
     *************
    1   DEMO:SUBR;
    2      X:NEW <500,450,400,350,300,250,200
    3        150,100,50,0,-50,-100,-150,-200>;
    4 SLMOVE:SUBR(X);
    5      P:NEW PT(X,400,0,0);
    6        PMOVE (P);
    7        END;
    8        ITERATE('SLMOVE',X);
    9        END;
```

LEFT

AML/Entry Command

Format: `LEFT;`

Manipulators: This command is only applicable to the 7545-800S manufacturing system.

Purpose: Switches the manipulator to left mode.

Remarks: The 7545-8005 is a symmetric arm manipulator, thus there are points that can be reached in two different ways (in either left or right mode). There are also points that can only be reached in left mode or only in right mode. The LEFT command specifies that all subsequent motion commands (i.e. PMOVE, DPMOVE, GETPART, XMOVE, ZMOVE) will be performed in left mode. If a point is specified that can only be reached in right mode, then a data error (point out of workspace) occurs.

Example: The point PT1 in the following example can only be reached in left mode. When the 7545-800S is returned home, it is placed in right mode. Thus before a move to PT1, the LEFT command must be given.

```
COMMAND INPUT -->
     *************
    1 DEMO:SUBR;
    2 PT1:NEW PT(500,-500,0,0);
    3       LEFT;
    4       PMOVE(PT1);
    5       END;
```

LINEAR

AML/Entry Command

Format: LINEAR(quality);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command changes the motion between goals from an arc to straight line motion or back to an arc.

Remarks: A 1 for the quality gives the straightest achievable manipulator move along a line between two points. A quality of 50 gives the greatest deviation from straight and the fastest manipulator move along a line between two points. A quality of 0 disables the linear motion. The "quality" in the statement determines the number of calculated points made by the controller between the current location and the next goal to maintain a straight move. For example, a quality of 50 results in 50 millimeters between each point calculated by the controller until it arrives at the goal.

Stated simply, the lower the quality value, the straighter the line, and the slower the manipulator's arm moves. A slower movement is more accurate.

The quality can consist of an integer constant, a formal parameter, or a counter.

Note: When the controller reaches the last **END;** statement in the program, it automatically defaults to the standard arc movements. If you want to maintain linear motion, the **LINEAR** statement must precede the first **PMOVE** statement in the program.

The programmed linear moves must remain within the borders of the linear area defined in IBM Manufacturing Systems Specification Guide, 8577126. Linear moves extending outside the linear work space can cause error conditions and unpredictable results.

Note: The Home position is not in the valid linear workspace. The arm cannot make LINEAR movements from Home.

Example:

This example shows simple movements between points. Different speeds and accuracies occur depending on the condition set by the LINEAR statement.

```
COMMAND INPUT -->
     *************
    1 DEMO:SUBR;
    2 START:SUBR;
    3       PMOVE (PT(400,400,-100,0));
    4       LINEAR(1);             -- SLOW PATH
    5       PMOVE (PT(500,300,-50,0));
    6       LINEAR(0);             -- DISABLE STRAIGHT
    7       END;
    8       PMOVE (PT(300,300,-75,0));
    9       LINEAR(50);            -- FAST PATH
   10       START;
   11       END;
```

LOCATE

Primary Edit Command

Format:

```
LOCATE n

L n
```

Purpose: This primary command places the indicated line number at the top of the program window.

Remarks: You can enter an abbreviated command, as shown in the format, or use the entire command.

Line Edit Command

Format:

Purpose: This line command indicates a single line to be moved. The line is relocated after the line that contains the **A** line command or before the line that contains the **B** line command. The line numbers are put in numeric order when the command is executed.

Remarks: This command is used in conjunction with the **A** or **B** line commands.

Line Edit Command

Format: MM

Purpose: This line command indicates a block of lines to be moved. The block is moved after the **A** or before the **B** command. The line numbers are put in numeric order when the command is executed.

Remarks: This command is used in conjunction with another **MM** and either the **A** or **B** commands.

Note: The **A** or **B** commands cannot be placed between the **MM** commands.

MSTATUS

AML/Entry Command and Arithmetic Function

Format (command): MSTATUS(counter_name);

Format (arithmetic function): MSTATUS()

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command/function allows you to determine the completion status of the current move.

Remarks: The command form assigns the motion status into a counter. The arithmetic function form returns a value which can be used in an AML/Entry expression. See "Expressions" on page 4-48 for a discussion of expressions. The following commands show how the different forms of MSTATUS may be used to attain identical results.

```
MSTATUS(counter_name);
SETC(counter_name,MSTATUS());
```

The advantage of using the arithmetic function form is that the MSTATUS may appear immediately in a TESTC or COMPC, because these commands allow expressions. The command form requires a counter be declared to hold the MSTATUS value.

The controller maintains a special status byte to monitor the completion of an entered command move. Data returned by MSTATUS is as outlined below.

0 = move completed normally
1 = move terminated by GUARDI
2 = move never started due to a GUARDI

Example: An example of an MSTATUS command follows.

```
MAIN: SUBR;
   CARD POINT : NEW PT(0,450,0,0);
   CARD POINT2 : NEW PT(0,350,0,0);
   STOP POINT : NEW 5; -- the DI point to guard
   NEW PLACE :   NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);            -- move to the start point
   GUARDI(STOP_POINT,1):         -- guard for the stop point
   PAYLOAD(11);
   LINEAR(1);
   PMOVE(CARD_POINT2);           -- move to the new point
   COMPC(MSTATUSO<>0,HERE);      -- if stopped by guard
                                 -- send point to host
   NOGUARD;                      -- disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW PLACE);          -- read the stop location
   PUT(NEW_PLACE)                -- send location to host
EN:                              -- the remainder of the program
   END;
```

AML/Entry Keyword

Format:

```
name:NEW PT(coordinates);

name:NEW 'string';

name:NEW <aggregate>;

name:NEW n;
```

Purpose: This keyword identifies a constant. A constant may be a number, point, character string, or an aggregate. An aggregate is multiple points, numbers, or character strings.

Example: The following program fragment shows the declaration of a **NEW** aggregate, constant, number constant, point constant, and string constant.

```
COMMAND INPUT -->
     *************
    1 PORTS:NEW <3,4,5,6>;       --AGGREGATE
    2 ZERO:NEW 0;                --NUMBER
    3 PORT:NEW 8;                --NUMBER
    4 POINT:NEW PT(650,0,0,0); --POINT
    5 POINT2:NEW PT(650,ZERO,ZERO,ZERO);
    6        --POINT2 USES CONSTANT FROM LINE 2
    7   NAME:NEW 'POINT';        --STRING
```

NEXTPART

AML/Entry Command

Format: `NEXTPART(name);`

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command is a palletizing command that increases the current part by 1.

Remarks: The pallet name used in the NEXTPART command can be passed to the command as a formal parameter, but since parameters are passed by value, the changed part number is only affected in the subroutine that is called. The part number of the calling subroutine is not changed.

Note: When NEXTPART is used to advance the part number beyond the maximum number of parts specified for the pallet, the current part "wraps around" and becomes 1. NEXTPART does **not** set the part number to a number greater than the maximum number of parts in the pallet.

In addition to the **NEXTPART** command, other pallet statements are:

- name:STATIC PALLET(ll,lr,ur,ppr,parts);
- GETPART(name);
- PREVPART(name);
- SETPART(name,value);
- TESTP(name,value,label);

Example: In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** uses all the above information, as shown in the example. The counter is advanced from 0 to 1 the first time statement 8 executes. Each following execution, the counter is increased by 1. The move to the current part position is done in line 9. Line 17 advances the current part, and line 18 checks to see if the current part is back to 1 (in which case all the parts have been processed).

When all the parts have been processed, the program waits for an input to start over at part number 1.

```
COMMAND INPUT -->
     *************
    1     LL:NEW PT(300,200,0,0);
    2     LR:NEW PT(300,300,0,0);
    3     UR:NEW PT(500,300,0,0);
    4    PPR:NEW 8;                          --PARTS PER ROW
    5 PARTS:NEW 32;                          --TOTAL # OF PARTS
    6 POINTS:STATIC PALLET(LL,LR,UR,PPR,PA RTS);
    7 MAIN:SUBR;
    8       SETPART(POINTS,1);               --SET PALLET
    9 LOOP:GETPART(POINTS);                  --PALLET MOVE
   10       ZMOVE(-250);
   11       GRASP; DELAY(1);                 --PICK UP PART
   12       ZMOVE(0);                        --RAISE Z
   13       PMOVE( PT(250,300,0,0));         --MOVE FROM PALLET
   14       ZMOVE(-250);                     --LOWER Z
   15       RELEASE; DELAY(1);               --DROP OFF PART
   16       ZMOVE(0);                        --RAISE Z
   17       NEXTPART(POINTS);                --CHANGE COUNTER
   18       COMPC(TESTP(POINTS)<>1,LOOP);    --TEST COUNTER
   19 DONE :WAITI(16,1,0);                   --WAIT FOR INPUT
   20 END;
```

NOGUARD

AML/Entry Command

Format: NOGUARD;

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: The NOGUARD command is used to cancel the GUARDI (motion guard) feature.

Remarks: None

Example: An example of a NOGUARD command is shown below.

```
MAIN: SUBR;
   CARD POINT : NEW PT(0,450,0,0);
   CARD—POINT2 : NEW PT(0,350,0,0);
   STOP—POINT : NEW 5; -- the DI point to guard
   NEW PLACE :    NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);          - move to the start point
   GUARDI(STOPPOINT,1):        - guard for the stop point
   PAYLOAD(11);
   PMOVE(CARD_POINT2);        -- move to the new point
   COMPC(MSTATUSO<>0,HERE);- - if stopped by guard
                               - send point to host
   NOGUARD;                   -- disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW_PLACE);       -- read the stop location
   PUT(NEW_PLACE);            -- send location to host
EN:                           -- the remainder of the program
   END;
```

PALLET

AML/Entry Keyword

Format: `name:STATIC PALLET(LL,LR,UR,PPR,PARTS);`

Purpose: This keyword defines a "name" to be a pallet.

Remarks: The parameters in the parentheses following the pallet keyword provide location information of the coordinates for three corner goals of the pallet, the number of parts per row, and the number of parts. The **LL** is the lower left part location. The **LR** is the lower right part location. The **UR** is the upper right part location. The **PPR** is the number of parts per row, and the **PARTS** is the number of parts.

The controller keeps track of which part is the current part. When you load your application program, the current part is not defined. You must define the first part with a SETPART statement. The part numbers in the control map start at the corner that you define as the lower left increasing to the corner you define as the lower right. The example shows two arrangements of a pallet with 12 parts and four parts per row.

```
                UR     | LL           LR
 9   10   11   12      | 1    2    3    4
 5    6    7    8      | 5    6    7    8
 1    2    3    4      | 9   10   11   12
 LL             LR     |              UR
```

A one dimensional pallet can be defined in two different ways, either as a row or a column. To define a one dimensional pallet as a row, the LR and UR points must be the same, and the PPR must equal PARTS. To define a one dimensional pallet as a column, the LL and LR points must be the same, and the PPR must equal 1. In either case, the LL point will correspond to the first part and the UR point will correspond to the last part. As far as the AML/Entry Compiler is concerned, they are identical.

Your program must set the pallet to a valid part position before attempting to use the pallet move command the first time. Refer to the following commands for details about counter control and move commands:

- GETPART(name);
- NEXTPART(name);
- PREVPART(name);
- SETPART(name,value);
- TESTP(name,value,label);

Example 1: In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** uses all the above information as shown in the example. The NEXTPART statement in line 8 advances the counter each time the statement is encountered. The move to the current part position is done in line 9. Line 17 advances the current part, and line 18 checks to see if the current part is back to 1 (in which case all the parts have been processed). When all the parts have been processed, the program waits for an input to start over at part number 1

Note: The Z coordinate in pallet point declarations is ignored; the Z coordinate from the last position is maintained. Therefore, it is important to move the Z arm before issuing a GETPART, to avoid a collision in the workspace. For palletizing to work properly, it is assumed the pallet is perpendicular to the Z-axis. It should also be noted that the Roll coordinate is always the Roll of the lower left point. It is not possible to change the degree of roll in the arm during pallet execution (except by using separate DPMOVE command).

```
COMMAND INPUT -->
     *************
    1     LL:NEW PT(300,200,0,0);
    2     LR:NEW PT(300,300,0,0);
    3     UR:NEW PT(500,300,0,0);
    4    PPR:NEW 8;                          --PARTS PER ROW
    5 PARTS:NEW 32;                          --TOTAL # OF PARTS
    6 POINTS:STATIC PALLET(LL,LR,UR,PPR,PARTS);
    7  MAIN:SUBR;
    8       SETPART(POINTS,1);               --SET PALLET
    9  LOOP:GETPART(POINTS);                 --PALLET MOVE
   10       ZMOVE(-250);
   11       GRASP; DELAY(1);                 --PICK UP PART
   12       ZMOVE(0);                        --RAISE Z
   13       PMOVE( PT(250,300,0,0) );        --MOVE FROM PALLET
   14       ZMOVE(-250);                     --LOWER Z
   15       RELEASE; DELAY(1);               --DROP OFF PART
   16       ZMOVE(0);                        --RAISE Z
   17       NEXTPART(POINTS);                --CHANGE COUNTER
   18       COMPC(TESTP(POINTS<>1, LOOP);    --TEST COUNTER
   19  DONE :WAITI(16,1,0);                  --WAIT FOR INPUT
   20  END;
```

Example 2: The following program shows how to declare a one dimensional pallet.

```
LL: NEW PT(0,500,0,0);
LR: NEW PT(-300,500,0,0);
UR: NEW PT(-300,200,0,0);
N: NEW 5;
P:  STATIC PALLET(LL,LR,LR,N,N);
Q:  STATIC PALLET(LR,LR,UR,1,N);
```

P and Q are both one dimensional pallets with N parts. P is defined as a row, Q as a column.

PAYLOAD

AML/Entry Command

Format: `PAYLOAD(value);`

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command controls tool-tip speed by overriding the speed switches or defaulting to those switches.

Remarks: The number values for speed range from the number 1 for the slowest speed to the number 10 for the fastest speed. The value 0 causes the switch setting to be used. Additional values 11 through 19 are provided in Version 4. All are slower than PAYLOAD 1. Example tool-tip speeds are listed below.

```
PAYLOAD(12); = 20 percent of PAYLOAD(1);
PAYLOAD(19); = 90 percent of PAYLOAD(1);
```

Using payloads 11 through 19 allows shorter stopping distances when GUARDI is used to interrupt motion. This is accomplished by using a different monitoring scheme, no deceleration ramp, and slOwer speeds.

The value can be an integer constant, a formal parameter, or a counter.

You must have some type of movement command between PAYLOAD commands. If you assign a value to the PAYLOAD command, that value can not change until some type of movement command has been executed. If two PAYLOAD commands are executed without a movement command in between, a DE error can sometimes occur.

> **Note:** When the controller reaches the last END statement in the program, it automatically defaults to the speed switches. If you want to maintain a certain programmed speed, the PAYLOAD statement must precede the first PMOVE statement in the program.

Example An example of a PAYLOAD command is shown below.

```
MAIN: SUBR;
   CARD_POINT : NEW PT(0,450,0,0);
   CARD POINT2 : NEW PT(0,350,0,0);
   STOP POINT : NEW 5; -- the DI point to guard
   NEW PLACE :   NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);         -- move to the start point
   GUARDI(STOP_POINT,1):      -- guard for the stop point
   PAYLOAD(11);
   LINEAR(1);
   PMOVE(CARD_POINT2);        -- move to the new point
   COMPC(MSTATUSO<>0,HERE);-- if stopped by guard
                             - send point to host
   NOGUARD;                   -- disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW_PLACE);       -- read the stop location
   PUT(NEW_PLACE);            -- send location to host
EN:                           -- the remainder of the program
   END;
```

AML/Entry Command

Format:

```
PMOVE (PT(x,y,z,r));
PMOVE (point name);
```

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This instruction moves the manipulator to a specified position and rotation. The point coordinates are in millimeters or inches and the roll is in degrees. The manipulators need X, Y, Z, and R values for the command to execute. The values may be declared names.

Remarks: The coordinates of the position may be part of the statement if the keyword **PT** is included in the statement, as shown in the example. The X, Y, Z, and R values may be counters or constants, provided they are declared in the program. You can also supply a name that has been declared as a PT.

Example: The following program fragments show the PMOVE command used in the forms described above.

```
 ZERO:NEW 0
START:NEW PT(-650.00,0,ZERO,-180);
      PMOVE (PT (650,0,-100,50));
      PMOVE (START);
      PMOVE (PT(650,0,-75,ZERO));
```

PREVPART

AML/Entry Command

Format: PREVPART(name);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This palletizing command decreases the current part by 1.

Remarks: The pallet name used in the PREVPART command can be passed to the command as a formal parameter, but since parameters are passed by value, the changed part number is only affected in the subroutine that is called. The part number of the calling subroutine is not changed.

Note: When PREVPART is used to decrease the part number and the current part is 1, then the current part "wraps around" and becomes the maximum part number. PREVPART does **not** set the part number to 0.

In addition to the **PREVPART** command, other pallet statements are:

- name:STATIC PALLET(ll,lr,ur,ppr,parts);
- GETPART(name);
- NEXTPART(name);
- SETPART(name,value);
- TESTP(name,value,label);

Example: In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** uses all the above information, as shown in the example. The counter for the pallet is set in line 8. Line 9 changes the current part number, allowing the program to go to the last part number and then decrease that number each time the statement is executed. The move to the current part position is done in line 10. Line 13 decrements the current part; line 14 tests to see if the current part has not become reset to 32, and if so, the program branches to the label "LOOP."

```
COMMAND INPUT -->
     *************
    1     LL:NEW PT(300,200,0,0);
    2     LR:NEW PT(300,300,0,0);
    3     UR:NEW PT(500,300,0,0);
    4    PPR:NEW 8;                   --PARTS PER ROW
    5 PARTS:NEW 32;                 --NUMBER OF PARTS
    6 POINTS:STATIC PALLET(LL,LR,UR,PPR,PARTS);
    7   MAIN:SUBR;
    8        SETPART(POINTS,32);           --SET PALLET
    9 LOOP: GETPART(POINTS);               --MOVE TO PART
   10        ZMOVE(-250); GRASP; DELAY(1); ZMOVE(0);
   11        PMOVE( PT(250,300,0,0));
   12        ZMOVE(-250); RELEASE; DELAY(1); ZMOVE(0);
   13        PREVPART(POINTS);             --CHANGE COUNTER
   14        COMPC(TESTP(POINTS)<>32,LOOP);--TEST COUNTER
   15 DONE:WAIT(16,1,0);                  --WAIT FOR INPUT
   16        END;
```

PRINT

Primary Edit Command

Format: PRINT

Purpose: This primary command prints the program on the optional printer.

Remarks: You can interrupt the PRINT command by pressing the **Esc** key. (The print screen method is not interrupted using the **Esc** key.) If the optional printer is on, the contents of the screen may be printed by using the shift key in conjunction with the **PrtSc** key.

AML/Entry Keyword

Format:

```
name:NEW PT(x,y,z,r);

PMOVE(PT(x,y,z,r));
```

Purpose: This keyword indicates that the values following it describe a position and rotation of a point within the work envelope. The values within the parentheses are real or integer numbers.

Remarks: When you declare a position and rotation using a name, you use this keyword following the term "NEW." A space is required between the two keywords. The name is then used in your program when referring to that point. Using names for your coordinates makes the program easier to understand, because:

- Names for a point are easier to remember than numbers when building your application program
- Teaching or changing points that are declared at the top of the program saves more time than searching the program for each point.

When you use the **PMOVE** command with coordinates rather than a referenced name, you must include the keyword **PT** in the outer set of parentheses, with the coordinates in the inner set. (See "Format" above.)

Examples: The following program fragment shows uses of PT.

```
COMMAND INPUT -->
     *************
    1 POINT:NEW PT(650,0,0,0); --named point declaration for
    2 DEMO:SUBR;               --servoed Z manipulators
    3       PMOVE(POINT);               --name used
    4       PMOVE( PT(400,300,-100,0)); --coordinates
```

AML/Entry Command

Format: PUT(counter_name);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command indicates to the host that the controller wants to send specific data.

Remarks: The controller initiates a data report operation and the application program waits until the host accepts the data. If the host does not respond within three seconds, the controller re-transmits the D record. If the controller does not respond after two retries (nine seconds total), the application program is stopped and the TE (transmission error) lamp comes on.

If a PUT command is executed when the controller is not on-line and connected to the host, a DE (data error) results. See -- Heading id 'fixit' unknown -- for a discussion of how this can be avoided.

PUT is used to send a single counter, a single point, a single counter within a group, or a single point within a group as well as either a group of counters or points.

Example: An example of a PUT command is shown below.

```
C: STATIC COUNTER;
P: STATIC GROUP(PT1,PT2);
   PUT(C);                      -- send value for counter
   PUT(P);                      -- send values for two points
```

PUTFILE

Primary Edit Command

Format: PUTFILE filename

Purpose: This primary command allows you to extract all or part of the file currently being edited and place it into another file.

Remarks: In addition to filename (if no extension is entered, the Editor assumes .AML), you must specify which lines of text currently being edited are to be extracted and placed into the new file. This is accomplished by entering the CC or C line command in the line command area. If you want to extract a single line, the C must be placed adjacent to that line before entering a PUTFILE command. If a block of lines are to be extracted, specify the CC command in the appropriate locations.

C or a pair of CC identifiers must be specified before the PUTFILE command is entered on the primary command line and the <---I(enter) key is pressed.

If either C or CC is not specified when PUTFILE is entered, an error results and the command must be re-entered (unless, the ? command is used to recall the command). Error messages are also issued for an improper filename or for the name of a file that already exists. Appending to an existing file is not allowed.

Example: An example of a PUTFILE command is shown below.

```
--STATION1.AML - - - - - -  07-08-1985 - - - - - - - - - - -
COMMAND INPUT ---> PUTFILE STATION2.AML
 -   ***********************> Top•0F•FILE <************************
    1 EASY TO PROGRAM
    2 EASY TO PROGRAM
CC 3 THE IBM MANUFACTURING SYSTEM
    4 PICK AND PLACE ROBOT
CC 5 IS VERSATILE
    6 EASY TO PROGRAM
    7 THE IBM MANUFACTURING SYSTEM
    8 PICK AND PLACE ROBOT
    9 IS VERSATILE
 -   ***********************> goTTom•u•FILE <*********************
```

This example copies the marked contents (lines 3 through 5) of the STATION1.AML file into a new file labelled STATION2.AML.

Line Edit Command

Format: R[n]

Purpose: This line command repeats the line on which the command is entered. The repeated line is located below the original line. You may repeat a line up to nine times by specifying the number after the command. If the number (n) is omitted, a single repeat of a line is executed. The line numbers are put in numeric order when the command is executed.

AML/Entry Keyword

Format:

```
R: STATIC REGION
 (LL,UL,LR,UR,LS_LEN,RS_LEN,TOP_LEN,BOT_LEN);
```

Manipulators: This keyword is applicable to all systems using AML/Entry Version 4.

Purpose: This keyword allows you to define a region of space as an independently existing entity.

Remarks: Version 4 allows you to define a frame of reference in the manipulator workspace that corresponds to some external coordinate system. A frame of reference allows you to describe motions in this area that are relative to the region itself, not to the manipulator coordinate systems. Primary use of this feature is to allow application programs to be constructed to accept host data that refer to some engineering abstract of the assembly object (such as computer-aided design data), rather than requiring taught point data. Because of this, the program may be replicated across multiple machines without changes in the host data base (all applications use the same set of host data).

Moving to taught points is unaffected by regions. Also listed below, are the arguments required to show where the spatial coordinates are with respect to the manipulator frame (taught points).

LL = taught lower-left corner coordinate
LR = taught lower-right corner coordinate
UL = taught upper-left corner coordinate
UR = taught upper-right corner coordinate
LS_LEN = user-defined left side length
RS_LEN = user-defined right side length
TOP LEN = user-defined top length
BOT LEN = user-defined bottom length

The definition accepts formal parameters. Note that the region need not be rectangular. In cases where the region is rectangular, its use is easily understood. Skewed parallelograms function in a logical manner as an extension of rectangular behavior.

You are able to define points relative'to a region. This is especially useful in moving to coordinates that have been generated externally. For example, consider a

rectangular region somewhere in the workspace. A circuit board is placed in this region so that the card is aligned with the X and Y directions of the region (not the manipulator). If the card is to be populated, the external system knows the locations of the components to be inserted in that coordinate system. If a point is down loaded from the host, you are able to cause the manipulator to move to the correct location in the workspace by entering the below outlined command.

```
XMOVE( region_name,point_name);
```

Here, region_name is the name used in the REGION definition and point_name is a point in REGION coordinates.

The roll values of the LL, UL, LR, and UR points are entirely ignored. The roll orientation is determined by the line drawn from the LL point to the LR point. A positive roll value in point_name corresponds to a counterclockwise rotation from this line. A negative roll value in point_name corresponds to a clockwise rotation from this line.

Example: An example of the REGION keyword is shown below.

```
        LLPT:  NEW PT(0,350,0,0);
        LRPT:  NEW PT(100,350,0,0);
        ULPT:  NEW PT(0,550,0,0,0);
        URPT:  NEW PT(100,550,0,0);
        REG1:  STATIC REGION(LLPT,ULPT,LRPT,URPT,4,4,5,5);--4x5 REGION
        CTR1:STATIC COUNTER;
        CTR2:STATIC COUNTER;
        MAIN:  SUBR;

        MOVE: SUBR(X,Y);       -- MOVE TO A POINT WITHIN A REGION
        XMPT: NEW PT(X,Y,0,0);
     XMPT DN: NEW PT(X,Y,-100,0);
              XMOVE(REG1,XMPT);
              XMOVE(REG1,XMPT_DN);
              END;               -- END MOVE SUBR

  — —*** BEGINNING OF PROGRAM *** — -
              SETC(CTR1,0);          INITIALIZE COUNTERS
              SETC(CTR2,0);          THAT ARE THE REGION MOVE POINTS
       LOOP1: TESTC(CTR1,6,NEXT1);   END OF ROW ?
              MOVE(CTR1,CTR2);       MOVE TO POINT IN REGION
              INCR(CTR1);
              BRANCH(LOOP1);
       NEXT1: TESTC(CTR2,4,NEXT2);   LAST COLUMN ?
              SETC(CTR1,0);          RESET ROW COUNTER
              INCR(CTR2);            INCREMENT COLUMN COUNTER
              BRANCH(LOOP1);
       END;
```

RELEASE

AML/Entry Command

Format: RELEASE;

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This instruction opens the digital output point 2 (DO 2). In an unmodified system, the air supplied to a gripper or other device attached to Z-axis shaft, is controlled by DO 2.

Remarks: A delay may be required after this statement to allow the mechanical motion of opening the gripper to complete.

Example: This example uses the RELEASE command to allow the gripper to drop off the part it picked up.

```
     *********************
 1 PICKUP: SUBR;        --SUBROUTINE TO PICKUP PART
 2     PMOVE(PT(250,300,0,0));    --MOVE TO PART BIN
 3     ZMOVE(-250);
 4     GRASP;                     --PICKUP PART
 5     DELAY(1); --DELAY TO ALLOW GRIPPER TO CLOSE
 6     ZMOVE(0);
 7     PMOVE(PT(400,-300,0,0)); --DROPOFF POINT
 8     ZMOVE(-250);
 9     RELEASE;                   --DROPOFF PART
10     DELAY(2);--DELAY TO ALLOW GRIPPER TO RELEASE
11     ZMOVE(0);
12 END;
```

RENAME

Primary Edit Command

Format: RENAME devicename:filename.filetype filename.filetype

Purpose: This primary command renames a file.

Remarks: The device name precedes the file name and they are both separated from the file name by a colon. You do not need a device name if the file is located on the diskette in the default drive.

If the file to be renamed is in the editor, the RENAME command renames the copy located on the diskette; the copy in the editor is considered to have the old name. To prevent the editor copy of the file being saved under the old name on the diskette also, use the **CANCEL** command when you exit the editing session.

Rules for naming files are as follows:

- Maximum name length is eight characters.
- No special characters can be used for the first character of the name.
- The filetype must be specified; it does not default to .AML.

Example: The following example changes file "EXAMPLELAML" to "EXAMPLE2.AML".

RENAME EXAMPLE1.AML EXAMPLE2.AML

RIGHT

AML/Entry Command

Format: RIGHT;

Manipulators: This command is only applicable to the 7545-800S manufacturing system.

Purpose: Switches the manipulator to right mode.

Remarks: The 7545-800S is a symmetric arm manipulator, thus there are points that can be reached in two different ways (in either left or right mode). There are also points that can only be reached in left mode or only in right mode. The RIGHT command specifies that all subsequent motion commands (i.e. PMOVE, DPMOVE, GETPART, XMOVE, ZMOVE) will be performed in right mode. If a point is specified that can only be reached in left mode, then a data error (point out of workspace) occurs.

Example: The point PT1 in the following example can only be reached in left mode, the point PT2 only in right mode. When the 7545-800S is returned home, it is placed in right mode. After the move to PT1, a RIGHT command must be given to switch the manipulator to right mode.

```
COMMAND INPUT -->
     *************
    1 DEMO:SUBR;
    2 PT1:NEW PT(500,-500,0,0);
    3 PT2:NEW PT(-500,-500,0,0);
    4       LEFT;
    5       PMOVE(PT1);
    6       RIGHT;
    7       PMOVE(PT2);
    5       END;
```

SAVE

Primary Edit Command

Format: SAVE devicename:filename.filetype

Purpose: This primary command saves the program in the editor onto a diskette.

Remarks: A program that does not have a name is named when you use this command. A name may be entered for a program without a name when you enter the command or when **ENTER FILESPEC -->** is displayed. After the SAVE command has executed, the program name appears in the top left corner of the editor screen.

When a name is displayed in the top left corner of the screen, you do not have to enter the name to save it. Subsequent save commands use the name provided unless a new name is provided after the SAVE command.

When using the AML/Entry menu, you must specify the device name preceding the filename if the program is not saved on the same diskette as the editor.

The SAVE command does not warn you if you are saving the current file into a file that already exists. Care must be taken to not accidently overwrite another file.

SETC

AML/Entry Command

Format: SETC(counter_name,expression);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This command sets the named counter to a specified value.

Remarks: Counters can hold either real or integer values. The range of counters is approximately from -9.2E18 to 9.2E18. To assign an integer value to a counter, an integer constant is used for the value. To assign a real value to a counter, a real constant (one with a fractional portion) is used for the value. See "Expressions" on page 4-48 for a discussion of expressions.

Note: If this command is used within the ITERATE command, then the value cannot be an expression. In this case, the value must be a constant or a counter.

The name can be a formal parameter; if it is, the **SETC** command does not change the value of the calling argument.

Example: ZHEIGHT in the example is a global counter used to keep track of the height of a current part in a vertical pallet. In this application, the manipulator will move to the pallet, go to the proper height, grasp a part, and drop it off at home. Parts in the pallet reside at heights -240, -220, -200,...,-20. The expression on line 6 is used to add 20 to the height each time through the loop.

```
COMMAND INPUT -->
     *************

  1      ZHEIGHT:STATIC COUNTER;
  2      MAIN:SUBR;
  3         SETC(ZHEIGHT,-240);         --INITIALIZE PALLET
  4    MORE:PMOVE(PT(-300,300,0,0));    --MOVE TO PALLET
  5         ZMOVE(ZHEIGHT);             --GO TO PROPER HEIGHT
  6         SETC(ZHEIGHT,ZHEIGHT+20);   --NEXT LOCATION
  7         LINEAR(1);                  --TURN LINEAR ON
  8         DPMOVE(<30,0,0,0>);         --GO TO MAGAZINE
  9         GRASP; DELAY(1);            --GRAB PART
 10         DPMOVE(<-30,0,0,0>);        --LEAVE MAGAZINE
 11         LINEAR(0);                  --TURN LINEAR OFF
 12         PMOVE(PT(650,0,0,0));       --MOVE HOME
 13         RELEASE; DELAY(1);          --RELEASE PART
 14         COMPC(ZHEIGHT<O,MORE);      --GO UNTIL TOP
 15         END;
```

SETPART

AML/Entry Command

Format: `SETPART(name,value);`

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This palletizing command sets the specified pallet's current part to a specified value.

Remarks: Each time the program executes the SETPART statement, the statement sets the value in the current part counter to the number in the SETPART statement. Refer to "Pallet" in this appendix for details.

The pallet name used in the SETPART command can be passed to the command as a formal parameter, but since parameters are passed by value, the set part number is only affected in the subroutine that is called. The part number of the calling subroutine is not set.

Initially, the controller assumes the pallet part number is zero. This is why a SETPART command must be issued before any other palletizing statements.

The value can be an integer constant, formal parameter, or counter. A formal parameter or counter must contain an integer value, otherwise the manipulator may move to the wrong point.

Other palletizing statements are:

- `name:STATIC PALLET(ll,lr,ur,ppr,parts);`
- `GETPART(name);`
- `PREVPART(name);`
- `NEXTPART(name);`
- `TESTP(name,value,label);`

Example:

In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** uses all the above information as shown in the example. The counter is set the first time the program executes after being loaded in the controller. This is accomplished by testing for the controller initialized to 0 in the counter. If the 0 is present, the program executes line 10 to set the counter to 1. The move to the current part position is accomplished in line 11. Line 15 increments the current part number. Line 16 tests to see the current part has not yet been reset to 1. If so, the program branches to the label "LOOP."

```
COMMAND INPUT -->
     *************
    1     LL:NEW PT(300,200,0,0);
    2     LR:NEW PT(300,300,0,0);
    3     UR:NEW PT(500,300,0,0);
    4    PPR:NEW 8;               --PARTS PER ROW
    5 PARTS:NEW 32;      --TOTAL NUMBER OF PARTS
    6 POINTS:STATIC PALLET(LL,LR,UR,PPR,PARTS);
    7   MAIN:SUBR;
    8        TESTP(POINTS,O,SET);    --FIRST TIME ONLY
    9        BRANCH(LOOP);     --BRANCH WHEN NOT ZERO
   10    SET:SETPART(POINTS,1);      --SET COUNTER
   11 LOOP:GETPART(POINTS);        --MOVE TO PART
   12        ZMOVE(-250); GRASP; DELAY(1); ZMOVE(0);
   13        PMOVE( PT(250,300,0,0));
   14       DOWN;RELEASE;UP;
   15        NEXTPART(POINTS);       --CHANGE COUNTER
   16        COMPC(TESTP(POINTS)<>1,LOOP); --TEST COUNTER
   17   DONE:WAIT(16,1,0);           --WAIT FOR INPUT
   18        END;
```

AML/Entry Arithmetic Function

Format: SIN(expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the sine of an expression.

Remarks: This function is useful for making the manipulator move in a circle. A circle is described by the following parametric equations:

```
X=X0 + R*COS(THETA)
Y=YO + R*SIN(THETA)
```

The center of the circle is at (X0,Y0), the radius is R, and THETA is a parameter that traces a circle as it is varied from 0 to 360 degrees. See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
SIN(0)       =  0
SIN(45)      = SQRT(2)/2
SIN(-90)     =
```

SQRT

AML/Entry Arithmetic Function

Format: SQRT(expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the positive square root of an expression. The square root of a number is defined as that number which when multiplied by itself returns the original number.

Remarks: The SQRT of a negative expression will cause a run-time AML/Entry error. If COMAID is used to perform a R 01 (read machine status), the return code will be Hex 33 (Square root of a negative number). The SQRT returns the positive square root of its argument. See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
SQRT(0)         0
SQRT(4)         2
SQRT(25)        5
SQRT(-10)       AML/Entry Error
SQRT(-7*-7) =   7
```

STATIC

AML/Entry Keyword

Format:

```
name:STATIC COUNTER;

name:STATIC PALLET(ll,lr,ur,ppr,parts);

name:STATIC REGION(11,u1,1r,ur,ls,rs,top,bot);
```

Purpose: This keyword reserves controller storage for a counter or a pallet. When a variable is declared with the STATIC keyword you are saying that variable does not lose its value, even across power downs. When a variable is declared with the keyword STATIC it has no initial value. The value you give it may change during execution, but the last value it has is retained if execution starts over for any reason.

Remarks: Refer to "Counter", "Pallet" and "Region" in this appendix for more details about these data types.

AML/Entry Keyword

Format:

```
id:SUBR;

id: SUBR(parameter);

id:SUBR(parameterl,parameter2,...);
```

Purpose: This keyword identifies a subroutine. The ID is an identifier used to call the subroutine. The parameter(s) is used when you want to pass a variable from another part of your program into the subroutine. If no formal parameter is to be passed, the semicolon (;) is located after the term SUBR.

Remarks: Each time you use a SUBR keyword, you must have an END keyword as the last line of the subroutine. The SUBR keyword must be on a line separate from other AML/Entry statements.

Example 1: In the following example, the variable X is declared in line 2 to be an aggregate of four numbers. Each time the ITERATE statement calls the subroutine SLMOVE, a new value for X is passed until all values have been used. Within the subroutine SLMOVE, a new value of X is substituted each time into line 4, and a move is executed in line 5. Line 6 contains the END keyword for the SLMOVE SUBR line. Line 7 calls the subroutine SLMOVE. Line 8 contains the END keyword for line 1.

```
COMMAND INPUT -->
     *************
    1   DEMO:SUBR;
    2      X:NEW <500,400,300,200>;
    3 SLMOVE:SUBR(X);
    4      P:NEW PT(X,400,0,0);
    5        PMOVE (P);
    6        END;
    7        ITERATE('SLMOVE',X);
    8        END;
```

Example 2: The following application program has a subroutine that does not use parameters when it is called to execute. The subroutine is identified as START. The program executes the move to coordinates 650,0,0,0 one time, and the subroutine START on line 2 is called by its name, located on line 7. After the subroutine executes the move to the two points in the subroutine, the program returns control to the next line following the statement that called the subroutine (line 8). Because line 8 is the end of the program, the program returns to the first line and starts the sequence over.

```
COMMAND INPUT -->
     *************
    1 DEMO:SUBR;
    2 START:SUBR;
    3       PMOVE (PT(400,400,-50,180));
    4       PMOVE (PT(400,400,-50,-180));
    5       END;
    6       PMOVE (PT(650,0,0,0));
    7       START;
    8       END;
```

AML/Entry Arithmetic Function

Format: TAN(expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the tangent of an expression. The tangent of a number is defined as the sine of the number divided by the cosine of the number.

Remarks: The TAN function may cause a data error (DE). Because the tangent of a number is equal to the sine of the number divided by the cosine of the number, whenever the cosine of the number is 0 (or sufficiently close to 0), a data error occurs. This happens when the number is a multiple of 90 degrees (i.e. ... -270, -90, 90, 270

See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
TAN(0)      =  0
TAN(45)     =  1
TAN(-60)    = -(SQRT(3)/2)/.5
TAN(450)    = Data Error
```

TESTC

AML/Entry Command

Format: TESTC(expression1,expression2,1abel);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This command tests the first value with the second value.

Remarks: See "Expressions" on page 4-48 for a discussion of expressions. If the expression are equal, then the program branches to the program statement label.

Note: If this command is used within the ITERATE command, then the values cannot be expressions. The first value must be a counter and the second value must be a constant or a counter.

Example: CTR1 in the example is a counter to build 200 parts before going to the next assembly process (which is not shown). In this application, each time a starting counter value is desired, DI 16 receives an input to set the counter to 0. The program branches to line 6 of the program if line 4 does not receive a DI 16 signal with a value of 1.

```
COMMAND INPUT -->
     *************
    1     CTR1:STATIC COUNTER;
    2    START:SUBR;
    3      SET:SUBR;
    4          TESTI(16,0,NOCHANGE);       --IS DI 16 ON
    5          SETC(CTR1,0);       --SET COUNTER TO ZERO
    6 NOCHANGE:                       --BYPASS RESET
    7          END;
    8          SET;    --CALL SET SUBROUTINE FOR COUNTER
    9    PART1:
   10          INCR(CTR1);        --ADD 1 TO THE COUNTER
   11             -- PARTS ASSEMBLE STATEMENTS
   12             -- PARTS ASSEMBLE STATEMENTS
   13          TESTC(CTR1,200,PART2);       --TEST COUNT
   14          BRANCH(PART1);           --CONTINUE PART1
   15    PART2:                       --ASSEMBLE PART2
   16           --PART2 STATEMENTS
   17          END;
```

AML/Entry Command and Arithmetic Function

Format (command): TESTI (digital_input_point , value, label) ;

Format (arithmetic function): TESTI(digital_input_point)

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: The command form checks a digital input point for an open- or closed- switch condition. If the condition of the DI is the same as the specified value in the statement, a branch is made to a labeled statement. If the condition at the DI point does not meet the condition for a branch, the program executes the statement that follows the TESTI statement. The function returns either a 0 or 1, corresponding to an open- or closed- switch condition. This value may then be used in any AML/Entry expression.

Remarks: The TESTI statement can be used with any of the DI points. Both the digital_input_point and the value can be integer constants, formal parameters, or counters. In the arithmetic function form, the digital_input_point can even be a complex expression. See "Expressions" on page 4-48 for a discussion of expressions. An expression for digital_input_point should evaluate to an integer. Any nonzero expression for value maps to a closed- switch position, while a zero for value maps to an open- switch position.

The arithmetic function form is more convenient, as it allows expressions to be used to specify the port number and the return value to be used in an expression. This allows one to easily check if all of a group of digital inputs are on or off.

Example: The following example shows how the TESTI can be used to test for certain conditions and then branch, if required. If FEEDER1 is empty, then the controller waits for it to be filled. After it is refilled, the program continues. Digital inputs 5, 6, and 7 are all connected to a different relay on the feeder. If any 2 of the three are on, then a part exists.

```
**k.W.L**************
 1 TEST5:NEW 5; --DI POINT 5
 2 FEEDER1:PMOVE(PT(400,400,0,180));
 3 EMPTY:    COMPC(TESTI(5)+TESTI(6)+TESTI(7)<=1,EMPTY);
 4
 5 MAIN:SUBR;
 6          ZMOVE(-250);
 7          GRASP;DELAY(1.0);     --PICK PART
 8          ZMOVE(0);
 9
10 DROPOFF:PMOVE(PT(650,0,0,0));
11          ZMOVE(-250);
12          RELEASE;DELAY(3.0); --PLACE PART
13          ZMOVE(0);
14          END;
```

Remarks: In this example, the program loops to the label EMPTY as long as two of the relays 5, 6, and 7 are open. As soon as two of them become closed, the statement ZMOVE(-250) executes on the next line (line 6).

TESTP

AML/Entry Command and Arithmetic Function

Format (command): TESTP (pal let_name , value , label) ;

Format (arithmetic function): TESTP(pallet_name)

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: The command form compares the current part of the pallet pallet_name to the given value. If the value matches, the program branches to the statement label. If the value does not match, the program executes the next statement. The name in the statement is the name used in the **STATIC** statement for the pallet. The arithmetic function form returns the current part number of pallet_name, which may be used in an AML/Entry Expression. See "Expressions" on page 4-48 for a discussion of expressions.

Remarks: In general, the arithmetic function form should be preferred over the command form. This is because the arithmetic function form can be used in expressions and the COMPC command.

In addition to the **TESTP** command, other pallet control statements are:

- GETPART(name);
- NEXTPART(name);
- PREVPART(name);
- SETPART(name,value);

Example: In the following pallet example, the pallet location declaration is contained in lines 1 through 3. The number of parts per row is 8 and the total number of parts is 32. The **STATIC PALLET** uses all the above information, as shown in the example. The counter for the pallet is set in line 8. The move to the current part position is done in line 9. Line 13 increments the current part number. Line 14 test to see if the current part number is not equal to 1. If so, the program branches to the label "LOOP."

```
COMMAND INPUT -->
     *************
   1     LL:NEW PT(300,200,0,0);
   2     LR:NEW PT(300,300,0,0);
   3     UR:NEW PT(500,300,0,0);
   4    PPR:NEW 8;                       --PARTS PER ROW
   5 PARTS:NEW 32;                       --NUMBER OF PARTS
   6 POINTS:STATIC PALLET(LL,LR,UR,PPR,PARTS);
   7 MAIN:SUBR;
   8        SETPART(POINTS,1);           --SET COUNTER
   9 LOOP:GETPART(POINTS);               --MOVE TO PART
  10       ZMOVE(-250); GRASP; DELAY(1); ZMOVE(0);
  11       PMOVE( PT(250,300,0));
  12       ZMOVE(-250); RELEASE; DELAY(1); ZMOVE(0);
  13       NEXTPART(POINTS);             --CHANGE COUNTER
  14       COMPC(TESTP(POINTS)<>1,LOOP);--TEST COUNTER
  15 DONE:WAIT(16,1,0);                  --WAIT FOR INPUT
  16       END;
```

TRUNC

AML/Entry Arithmetic Function

Format: TRUNC(expression)

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This arithmetic function returns the integer that is less than or equal to the given expression. The expression is "rounded downwards". Thus if the expression is positive, the fractional portion may be "thrown away". If the expression is negative, the fractional portion is "thrown away", but the integer portion is decremented by 1.

Remarks: It is possible to "round-off" a number by taking the TRUNC of the number + 0.5. This will "round-up" if the fractional portion is greater than or equal to 0.5, and "round-down" if the fractional portion is less than 0.5. See "Expressions" on page 4-48 for a discussion of expressions.

Examples:

```
TRUNC(0)      = 0
TRUNC(1.499) = 1
TRUNC(1.501) = 1
TRUNC(7.9)      7
TRUNC(-0.1)     -1
TRUNC(-7.6)   = -8
TRUNC(1.499+0.5) = 1   --Note how 1.499 is rounded
TRUNC(1.501+0.5) = 2   --Note how 1.501 is rounded
```

AML/Entry Command

Format: WAITI(digital_input_port,value,time_limit,[label]);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command allows you to respond to device time-out, rather than have the controller generate an OT (overtime) error.

Remarks: WAITI is designed to optimize throughput in an application that employs devices with time-limited behavior (such as grippers and feeders).

If the digital input (DI) point does not attain the specified value within the time-limit, control branches to the specified label. If the digital input (DI) point attains the specified value within the time limit, control falls through and the next instruction is executed. A time_limit of 0 tells the controller to wait forever for the DI to attain the specified value. That is, control will never branch to the optional label or an OT will never occur.

A label parameter may be used, but it is optional. If the label field is empty, the command continues to operate.

Example: An example of a WAITI command is shown below in a program fragment.

```
    GRIPPER : NEW 2;         -- the gripper close DO point
    GRIPPER_CLOSED : NEW 1; -- the gripper closed feedback point
    PROBLEM : NEW 3;         -- the problem light

 MAIN: SUBR;
  TRY: WRITEO(GRIPPER,1);              -- close the gripper
    WAITI(GRIPPER_CLOSED,1,1 .5,ERR); -- wait for the feedback
    BRANCH(OK);
  ERR:
    WRITEO(PROBLEM,1);                 -- report problem
    DELAY(3);
    BRANCH(TRY);                       -- retry the gripper
   OK:                                 -- continue program execution
    END;
```

WHERE

AML/Entry Command

Format: WHERE(pointname);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command allows you to update a point definition, based on the current X, Y, Z, and Roll position of the arm.

Remarks: The point must not contain any counters or formal parameters, otherwise the values returned from the WHERE command may be overwritten the next time the subroutine is invoked.

Example: An example of a WHERE command is shown below.

```
MAIN: SUBR;
   CARD POINT: NEW PT(0,450,0,0);
   CARD POINT2:NEW PT(0,350,0,0);
   STOP POINT: NEW 5;.       -- the DI point to guard
   NEW PLACE: NEW PT(0,0,0,0);

   PMOVE(CARD_POINT);        -- move to the start point
   GUARDI(STOP_POINTO.):     -- guard for the stop point
   PAYLOAD(ll);
   LINEAR(1);
   PMOVE(CARD_POINT2);       - - move to the new point
   COMPC(MSTATUSO<>0,HERE);- - if stopped by guard
                              - send point to host
   NOGUARD;                  - - disable motion guard
   LINEAR(5);
   PAYLOAD(5);
   BRANCH(EN);
HERE: WHERE(NEW PLACE);      -- read the stop location
   PUT(NEW PLACE);           -- send location to host
EN:                          -- the remainder of the program
   END;
```

AML/Entry Command

Format: WRITEO(digital output point,value);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4.

Purpose: This instruction controls a digital output point (DO) relay.

Remarks: There is a relay for each of the digital output points. You can specify the relay by using the DO point number in the statement. You can also name the DO point using the NEW keyword in a statement and then using the name in the WRITEO statement.

Once the command is given to open or close a DO relay, the relay remains in that state until the command to that DO point changes its state. The following are program fragments of each case.

```
FEEDER1:NEW 5;               --LABEL 5
        WRITEO(FEEDER1,1);    --WRITE POINT 5
        WRITE0(8,0);          --WRITE POINT 8
```

You can specify a zero value to open a DO relay, and.a nonzero to close DO relay. In the example above, DO point 5 relay is to close. DO point 8 relay is to open.

XMOVE

AML/Entry Command

Format: XMOVE(region_name,region_point);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command allows you to perform moves to a calculated point within a region.

Remarks: The point is in respect to region coordinates, not manipulator coordinates. **The roll values of the LL, UL, LR, and UR points of the region are ignored.** The roll orientation is determined by the line drawn from the LL point to the LR point. A positive roll value in region_name corresponds to a counterclockwise rotation from this line. A negative roll value in region_name corresponds to a clockwise rotation from this line.

Example: An example of an XMOVE command follows.

```
    LLPT: NEW PT(0,350,0,0);
    LRPT: NEW PT(100,350,0,0);
    ULPT: NEW PT(0,550,0,0,0);
    URPT: NEW PT(100,550,0,0);
    REG1: STATIC REGION(LLPT,ULPT,LRPT,URPT,4,4,5,5);--4x5 REGION
    CTR1:STATIC COUNTER;
    CTR2:STATIC COUNTER;
    MAIN: SUBR;

    MOVE: SUBR(X,Y);             -- MOVE TO A POINT WITHIN A REGION
    XMPT: NEW PT(X,Y,0,0);
 XMPT_DN: NEW PT(X,Y,-100,0);
          XMOVE(REG1,XMPT);
          XMOVE(REG1,XMPT_DN);
          END;                   -- END MOVE SUBR

-- *** BEGINNING OF PROGRAM *** --
          SETC(CTR1,0);          -- INITIALIZE COUNTERS
          SETC(CTR2,0);          -- THAT ARE THE REGION MOVE POINTS
   LOOP1: TESTC(CTR1,6,NEXT1);   -- END OF ROW ?
          MOVE(CTR1,CTR2);       -- MOVE TO POINT IN REGION
          INCR(CTR1);
          BRANCH(LOOP1);
   NEXT1: TESTC(CTR2,4,NEXT2);   -- LAST COLUMN ?
          SETC(CTR1,0);          -- RESET ROW COUNTER
          INCR(CTR2);            -- INCREMENT COLUMN COUNTER
          BRANCH(LOOP1);
   NEXT2:
   END;
```

ZMOVE

AML/Entry Command

Format: ZMOVE(position);

Manipulators: This command is applicable to all systems using AML/Entry Version 4.

Purpose: This command is used to perform absolute Z-axis movement. In millimeters the range is between 0 mm (fully retracted) and -250 mm (fully extended). In inches the range is between 0 in (fully retracted) and -9.8 in (fully extended). This command has no time limit to reach its specified goal.

The position can be a simple constant or complex expression. See "Expressions" on page 4-48 for a discussion of expressions.

Note: If this command is used within the ITERATE command, then the position cannot be an expression. The position must be a constant, formal parameter, or counter.

Example: This example uses the ZMOVE command to allow the gripper to pick up a part.

```
*********************
1 PICKUP: SUBR;          --SUBROUTINE TO PICKUP PART
2     PMOVE(PT(250,300,0,0));    --MOVE TO PART BIN
3     ZMOVE(-150);
4     GRASP;                              --PICKUP PART
5     DELAY(1); --DELAY TO ALLOW GRIPPER TO CLOSE
6     ZMOVE(0);
7 END;
```

ZONE

AML/Entry Command

Format: ZONE(value);

Manipulators: This command is applicable to all systems that use AML/Entry Version 4. However, reduction in settle time varies between machine types.

Purpose: This command changes the settle at a goal or defaults control to the zone switches in the controller. The amount of settle affects the precision of locating a goal and it affects the throughput. There is trade-off between the precision and the time required to execute program statements.

Remarks: The value for zone ranges from 1, for a long settle, to 15, for the minimum settle. A higher value in the ZONE command shortens the time that the manipulator takes in finding your desired points. It also lessens the exactness in those points. You give up a degree of accuracy to improve speed and time.

A 0 (zero) specifies that the switch settings are used. There is no program value equivalent to the switches when they are all in the off position, but if you have all the switches in the off position and specify a ZONE(0);, you get maximum accuracy in the manipulator's arm movement.

The following conditions automatically change a programmed zone setting:

- The controller reached the last **END** statement in the program and defaults to the zone switch settings in the controller
- A different zone statement was encountered in the program

To maintain a certain settle the **ZONE** statement must precede the first **PMOVE** statement in the program. This prevents an automatic default each time the program cycles to the beginning statements. In addition, consider resetting the zone as needed after you complete a linear move.

Example: This example shows how different values in the ZONE command affect simple movement.

```
COMMAND INPUT -->
     *************

  1 DEMO:SUBR;
  4 START:SUBR;
  5       ZONE(15);             -- SHORT SETTLE
  6       PMOVE (PT(400,400 ,0,180));
  7       ZONE(0);              -- USE ZONE SWITCHES
  8       PMOVE (PT(300,300 ,-50,-180));
  9   END;
  2       ZONE(1);              -- LONG SETTLE TIME
  3       PMOVE (PT(500,300 ,-100,0));
 10       START;
 11 END;
```

Primary Edit Command

Format:

Purpose: This primary command recalls the last primary command to the command input line.

Compiler Directive

Format:

Purpose: AML/Entry allows you to Include a file. Using the --%Include command, a file will cause additional files to be read by the compiler. The Included files' lines are included in the output (.ASC) and the listing (.LST) file.

Remarks. The --%I designates an Include function. (Both upper and lower case I can be used.) Nested Include files and embedded blanks are not allowed. The filespec of the Included file must contain the file's extension because the compiler will not assume the extension to be .AML. A path name may not be given with the filespec, but a drive may be given. If a drive is specified, then the included file must reside on the designated drive. If no drive is specified, then the drive that contains the source file is used as the default.

An "Including file" message is issued whenever a file is included. The lines of the Included file are identified in the .LST file by a % sign following the line address indicator.

Example: An example of the Include compiler directive is shown below. The example shows two lines of code that are included in the subroutine MAIN by the command --WA: DATA.AML'.

```
****MAIN.AML file**

MAIN: SUBR;
--%I'A: DATE.AML'
PMOVE PT1;
PMOVE PT2;
END;
```

```
****DATA.AML file****

PT1:NEW PT(650,0,0,0);
PT2:NEW PT(0,550,0,0);
```

```
****MAIN.LST file****

MAIN: SUBR;
        DATA.AML'
%PT1:NEW PT(650,0,0,0);
%PT2:NEW PT(0,550,0,0);
PMOVE PT1;
PMOVE PT2;
END;
```

Compiler Directive

Format: **--%P**

Purpose: The Page compiler directive causes a page break in the listing file. It allows you to control the paging within the listing file.

A-110 58X7338

APPENDIX B. AML/ENTRY MESSAGES

This appendix contains an alphabetic-order list of all messages that do not have a message number and a numeric-order list of all error messages that contain error numbers.

Error messages can be the result of working with a file in an edit session, or attempting to transmit a program to the controller. In AML/Entry, the computer attempts to inform you where the error occurred. AML/Entry run-time error messages are described in Chapter 8, "Communications." Information on error messages encountered when you are using DOS is provided in DOS documentation.

MESSAGES WITHOUT NUMBERS

A, R, or I PLEASE

Cause: In response to an Abort, Retry, or Ignore question, an input other than A, R, or I was entered.

Recovery: Input A, R, or I and press the enter key.

Component: Compiler

ABORT LOAD? *(y* or n)

Cause: This message appears when an empty filename is entered, an empty partition number is entered, or the requested filename is not found for a download operation. Rather than aborting the load request all together, this allows the operation to be retried with a new filename or partition number.

Recovery: If the operation is to be retried, select option "n". If the load is to be aborted, select option "y".

Component: Comaid

ABORT(A), RETRY(R), or IGNORE(I)

Cause: This is a normal message following an operations error, such as the printer not available. The Abort option, specified by entering the single letter A, cancels the compiler; the Retry option, specified by entering the single letter R, retries the operation (after the problem has been fixed); the Ignore option, specified by entering the single letter I, ignores the current operation, but continues with the compile.

Recovery: Enter the correct response and press the enter key.

Component: Compiler

---

xxxx ALREADY EXISTS

Cause: A rename command specified a new file name for a file, but the new name has already been used for another file on the diskette.

Recovery: You should rename the file to an unused name on the diskette.

Component: Editor

ATTENTION!!!
THE ROBOT IS ABOUT TO GO HOME. IF THE SHAFT IS EXTENDED IT MAY COLLIDE WITH FIXTURES IN YOUR WORKSPACE. PRESS Q TO ABORT TEACH OR ANY OTHER KEY TO CONTINUE.

Cause: This message appears whenever Teach mode is entered for the first time or after a communications error. The robot must first move home in order to zero its encoders.

Recovery: First move the Z axis up by using the control panel if the Z axis is extended. Then press any key to enter Teach mode and cause the robot to move home or a Q to abort Teach mode.

Component: Editor

BAD FILE NAME

Cause: An improper file name was used.

Recovery: Rename the file using these rules.

- Only A, B, C (fixed disk drive), and D (fixed disk drive) may be used as a device name.
- The file name must be less than eight characters and not contain illegal characters.

Component: Editor

CAN NOT FIND FILE

Cause: One of the files needed by the configuration utility is missing.

Recovery: Ensure the correct diskettes are in the diskette drive.

Component: Configuration utility

CAN NOT OPEN .ASC FILE

Cause: The compiler can not open the output .ASC file.

Recovery: Ensure the diskette used to store the output file has enough space to record another file.

Component: Compiler

CAN NOT OPEN INCLUDE FILE: filespec

Cause: This message is produced when the Include command is encountered, but the specified file (filespec) can not be opened.

Recovery: Make sure the file to be included is on the device specified. Also make sure that the file CONFIG.SYS contained in your root directory contains the line "FILES=12". This is done automatically by the Autoinit procedure.

Component: Compiler

CAN NOT OPEN .LST FILE

Cause: The compiler can not open the listing .LST file.

Recovery: Ensure the diskette used to store the output file has enough space to record another file.

Component: Compiler

CAN NOT OPEN .SYM FILE

Cause: The compiler can not open the symbol table .SYM file.

Recovery: Ensure the diskette used to store the symbol table has enough space to record another file.

Component: Compiler

CAN NOT TEACH

Cause: The editor is unable to enter teach mode because the manipulator is in a condition that does not allow it to enter teach mode.

Recovery: Ensure that there are no errors present at the manipulator, and that the manipulator is not running an application program.

Component: Editor

CHECK THAT CONTROLLER IS ON-LINE AND CABLE IS ATTACHED. ABORT TEACH? (y[yes], n[no], i[ignore])

Cause: This message occurs when the editor is unable to establish communications between the Personal Computer and the controller.

Recovery: Choose option "y" to return to the editor, option "n" to retry establishing communications, or option "i" to enter a simulated version of teach.

Component: Editor

COMMAND FILE TERMINATED BY ERROR

Cause: This message appears when one of the lines in a Comaid command file caused an error. The error could be because of an invalid command line or a communications error from the controller. In either case, processing of the command file is terminated.

Recovery: None necessary.

Component: Comaid

COMMAND PENDING

Cause: Only one part of a line command that uses two parts has been input. This message occurs whenever you must scroll forward or backward to enter paired line command(s). The line commands listed here are used together:

```
C used with A or B
CC used with CC and A or B
DD used with DD
M used with A or B
MM used with MM and A or B
```

If you enter pairs of commands that do not fit on one screen this message is normal.

Recovery: You may clear the condition by pressing function key **F3** and typing in all the required commands. You can also enter the remaining command on the desired line and press the enter key. This allows the command to complete.

Component: Editor

COMMUNICATIONS ERROR

Cause: The Personal Computer has encountered an error when it tried to communicate with the controller. This error occurs during program loading and teach functions.

Recovery: Ensure that the communications cable is attached between the controller and the Personal Computer, and the "On Line" LED is on at the controller. If the **TE** LED on the control panel is lit, press the **Reset** key.

Component: Editor

COMPILATION ABORTED/ABNORMAL TERMINATION

Cause: A severe program error occurred causing the compiler to terminate. This message also occurs with the Write Protect Violation, Drive Not Ready, and the Device Full messages.

Recovery: Correct the program error and retry the compiler. Ensure the diskette containing the compiler is not removed from the drive after the compiler is loaded.

Component: Compiler

COMPILATION TERMINATED

Cause: Due to one of the following reasons the compiler has terminated operation:

An Abort was entered.
Errors were encountered in the program that did not allow the compiler to continue.
A disk error occurred while compiling.

Recovery: Correct the error and compile the program again.

Component: Compiler

CONFIGURED CONTROLLER IS INCOMPATIBLE WITH THIS VERSION OF EDIT/TEACH

Cause: This message is displayed if the machine type in the configuration file is not to be used with this version of the Editor.

Recovery: Reconfigure using Version 4.

Component: Editor

CONFIGURED MACHINE TYPE IS INVALID FOR THIS COMPILER.

Cause: This message is displayed if the machine type in the configuration file is not to be used with this compiler.

Recovery: Make sure that the files supplied with Versions 3 and 4 are not mixed.

Component: Compiler

CONVERTING AML/E PROGRAM

Cause: This is a normal message indicating the progress of the compilation process. This message tells you that the second of three steps for the compiler is executing.

Recovery: None necessary.

Component: Compiler

CREATING PROGRAM FOR MACHINE TYPE 7545

Cause: This message is displayed if the configuration is set for this machine type.

Recovery: None necessary.

Component: Compiler

CREATING PROGRAM FOR MACHINE TYPE 7545-S

Cause: This message is displayed if the configuration is set for this machine type.

Recovery: None necessary.

Component: Compiler

CREATING PROGRAM FOR MACHINE TYPE 7547

Cause: This message is displayed if the configuration is set for this machine type.

Recovery: None necessary.

Component: Compiler

CURRENT ROBOT CONFIGURATION - LEFT or RIGHT

Cause: This appears at the top of the Teach screen if the AML/E system is configured for a 7545-800S. It indicates the current arm configuration (left or right).

Recovery; None necessary..

Component: Editor

CURRENTLY PRINTING

Cause: This message is, displayed while a file is being printed.

Recovery: None necessary.

Component: Editor

DATA DRIVE MODE EXITED

Cause: This message appears after the Xoff has been sent to the controller to exit data drive mode. If a data drive request has been received, then it too is printed.

Recovery: None necessary.

Component: Comaid

DESTINATION PARTITION

Cause: Comaid is requesting the number of the partition in the controller (1-5) that is to receive the output file.

Recovery: Type in the correct partition number and press the enter key.

Component: Comaid

DEVICE FULL

Cause: The compiler attempted to create a file, or write to a file, on a disk device that ran out of room.

Recovery: Use a diskette with enough space to hold the file or erase files from the existing diskette to make room for the file.

Component: **Compiler**

DISK ERROR

Cause: A read/write error occurred when the compiler tried to use the diskette drive.

Recovery: Retry the operation. If the error reoccurs, try the operation using a different diskette that has been properly formatted. If the error still occurs, check the Personal Computer operation.

Component: Compiler

DISK FULL

Cause: An attempt was made to write or load a file that required more space than was available on the diskette.

Recovery: Erase files from the diskette until there is enough space for the file or use another diskette.

Component: Configuration utility, Editor

DISK IS WRITE PROTECTED

Cause: You attempted to write data on a diskette with the write protect notch covered.

Recovery: Remove the tab covering the write protect notch and retry the operation, or use a diskette that is not write-protected.

Component: Configuration utility, Editor

DISK I/O ERROR

Cause: The Personal Computer had an error when reading or writing to the diskette.

Recovery: Retry the operation. If the error reoccurs retry the operation using a different diskette that has been properly formatted. If the error still occurs, check the Personal Computer operation.

Component: Configuration utility, Editor, Menu

DISK NOT READY

Cause: The Personal Computer sensed that a diskette was not present when it tried to read or write to a diskette drive.

Recovery: Ensure that the diskette is properly inserted in the correct diskette drive. Ensure the door on the diskette drive is closed.

Component: Configuration utility, Editor

DO YOU WISH TO OVERWRITE?

Cause: This message occurs with the "TERMINATING EDITOR SESSION" message during the F8 save operation if you enter the name of a file that already exists.

Recovery: Enter Y or N.

Component: Editor

DRIVE NOT READY

Cause: A diskette drive was not ready when the compiler tried to read or write to it.

Recovery: Ensure that the diskette was installed in the correct diskette drive. Ensure the door on the diskette drive is closed.

Component: Compiler

END OF COMMAND FILE REACHED

Cause: This message appears after the last line in a Comaid command file has been processed.

Recovery: None necessary.

Component: Comaid

ENTER BREAKPOINT ADDRESS

Cause: This message prompts the user for the address where a breakpoint will be set. The mapping of AML/Entry program lines to physical addresses can be generated from the program listing created by the AML/Entry Compiler.

Recovery: Enter the address where the breakpoint will be set. This will cancel any previous breakpoint.

Component: Comaid

ENTER CHANGE COMMAND

Cause: The "Repeat Change" key **F5** was used before an initial **CHANGE** primary command was entered.

Recovery: Enter a **CHANGE** primary command then use the "Repeat" key.

Component: Editor

ENTER COMMAND FILE NAME

Cause: This message prompts the user for the name of a command file that contains a list of Comaid commands, one per line.

Recovery: Enter the name of the command file. No default file extension is assumed, thus enter the full file specification.

Component: Comaid

ENTER FILESPEC

Cause: The editor is prompting for a file to be edited (if entering the editor) or the filename under which the current file should be saved (if exiting the editor).

Recovery: Enter either the filename of the file to be loaded or the filename under which the current file should be saved. Hitting enter without giving a filename will return you back into the editor. Use the CANCEL command to abort the editor.

Component: Editor

ENTER FIND COMMAND

Cause: The "Repeat Find" key **F4** was used before an initial **FIND** primary command was entered.

Recovery: Enter a **FIND** primary command then use the "Repeat" key.

Component: Editor

ENTER NEW SOURCE FILE SPECIFICATION:

Cause: The file name given as the source file for the compiler is not in the correct form.

Recovery: The source file must have a file extension of .AML. Enter the file name again with the form: **device:name.AML.** If you do not specify a file extension the compiler defaults to the .AML extension.

Component: Compiler

ENTER NUMBER OF VARIABLES

Cause: This message appears when Comaid needs to know the number of variables that are to be read with the R 80 read command or the number of variables that are to be sent with the C 80 control command.

Recovery: Enter the number of variables that are to be read or sent.

Component: Comaid

ENTER PARTITION NUMBER TO BE UNLOADED

Cause: This proepts the user for the partition number to be unloaded for an unload operation.

Recovery: Enter the number 1, 2, 3, 4, or 5 to unload the respective partition of the controller. If desired, "ALL" may be entered to cause all 5 partitions to be unloaded.

Component: Comaid

ENTER "Q" TO QUIT, OR ANY OTHER KEY TO RETRY

Cause: If "Q" i$ pressed, the AML/Entry menu is displayed. Any other key runs the configuration utility again.

Recovery: Press the desired key.

Component: Configuration utility

ENTER SOURCE FILE SPECIFICATION:

Cause: This is a normal message requesting the name of the input source file for the compiler.

Recovery: Type in the name of the file to be compiled. The format of the file specification is: **device:name.AML.** If you do not specify a file extension the compiler defaults to the .AML extension.

Component: Compiler

ENTER STARTING VARIABLE

Cause: This message appears when Comaid needs to know the starting variable to be used with the R 80 read command or the C 80 control command.

Recovery: Enter the starting variable's number. This can be gotten directly from the XREF program.

Component: Comaid

ENTER VALUE FOR VARIABLE NUMBER

Cause: This message appears when either the C 80 control command is being performed or the user is preparing for a GET command.

Recovery: Enter the value for the variable that will be sent during the control command or if a GET is received.

Component: Comaid

ERROR IN DATA DRIVE MODE

Cause: This message appears after data drive mode has been exited and an error has occurred. The cause of the error will be printed on the line following this line.

Recovery: Take error recovery based on the cause of the error. Usually this means sending the reset error (X 13) request.

Component: Comaid

ERROR IN INCLUDE FILE: INCLUDE ENCOUNTERED FOR FILE: filespec

Cause: This message is produced when the Include command is encountered within an open include file. Nested Includes are not allowed.

Recovery: Remove nested Include file and re-compile.

Component: Compiler

ERROR ON DOWNLOADING

Cause: This message appears when an error has occurred during the download operation. The line following this gives additional information about the error.

Recovery: Retry the download operation if necessary.

Recovery: Comaid

ERROR ON LAST DATA DRIVE

Cause: Comaid will always print the last data drive request that was received when data drive mode is exited. If the last request was invalid, then this message is printed. This should never occur. If it is repeatable, IBM should be contacted.

Recovery: None necessary.

Component: Comaid

ERROR ON xxxx COMMAND

Cause: This message appears when an error occurs during a read, execute, control, or teach command. The following line will contain additional information about the error.

Recovery: Retry the operation if necessary. The controller may not be able to accept the request due to its current state.

Component: Comaid

ERROR ON UNLOADING

Cause: This message appears when an error has occurred during the unload operation. The line following this gives additional information about the error.

Recovery: Retry the unload operation if necessary.

Recovery: Comaid

ERROR REPORT HARDCOPY?

Cause: This is a normal message from the compiler. If you answer "Y" or "y" the error messages are printed on the printer and the screen. If you give any other answer, the error messages are displayed on the screen only.

Recovery: Type in the answer and press enter.

Component: Compiler

ERRORS:

Cause: This message gives the total number of errors encountered by the compiler when it processed the AML source program.

Recovery: None necessary

Component: Compiler

ERROR: AN UNRECOVERABLE INITIALIZATION ERROR HAS OCCURRED.

Cause: This message appears when Comaid cannot initialize its variables. This message should never appear for a Personal Computer with 192K memory.

Recovery: Retry calling Comaid.

Component: Comaid

ERROR: CANNOT FIND xxxx.TXT

Cause: This message appears if Comaid cannot find either MSGCOM.TXT or MSGCOM2.TXT.

Recovery: Insert a diskette containing the missing file into one of the diskette drives or copy the missing file into the current directory.

Component: Comaid

ERROR: COMMAND FILE NOT FOUND

Cause: This message appears when a specified command file cannot be found by Comaid.

Recovery: Check that the filename was correctly entered, and conforms with the DOS filespec naming conventions.

Component: Comaid

ERROR: INVALID COMAID COMMAND LINE

Cause: This message appears if a Comaid line in a command file contains an error. This message will not appear if an invalid option is selected, but will appear if an invalid operand or incorrect number of operands appears.

Recovery: Correct the command line in the Comaid command file that contains the error with the AML/Entry editor.

Component: Comaid

ERROR: INVALID OPTION IN COMMAND FILE

Cause: This message appears when an invalid option appears in a command file. For example, recursive command file processing is not allowed, so the F option is invalid in a Comaid command file.

Recovery: Remove the command line in the Comaid command file that contains the erroneous request.

Component: Comaid

ERROR: INVALID OPTION ON DOS COMMAND LINE

Cause: This message appears when an invalid option appears on the DOS command line. For example, the P request is not allowed on the DOS command line.

Recovery: This usually happens when a typographical error occurs (e.g., the Z option is requested instead of the X option). See "COMAID" on page 8-14 for a discussion of the Comaid options.

Component: Comaid

ERROR: xxxx.TXT HAS BEEN CHANGED.

Cause: This message appears when some of the lines have been deleted from either MSGCOM.TXT or MSGCOM2.TXT. These files should never be edited by the user.

Recovery: Get a new copy of the file that has been changed from either a backup diskette or volume 2 of the AML/Entry ship diskettes.

Component: Comaid

ERROR: STILL CANNOT OPEN COMM PORT, EXECUTION TERMINATED.

Cause: This message appears if communications still cannot be established between the Personal Computer and the controller. The first time communications cannot be established, the next error message is printed.

Recovery: Once again, check that the cable is correctly attached, the AML/Entry system is properly configured, and the controller is on-line.

Component: Comaid

ERROR: UNABLE TO OPEN COMMAND FILE

Cause: This message appears when the entered command file cannot be opened.

Recovery: Retry the F option, making sure the file exists and is correctly specified to Comaid.

Component: Comaid

ERROR: UNABLE TO OPEN PORT COMn: CHECK HARDWARE AND STRIKE ANY KEY TO CONTINUE...

Cause: This message appears when Comaid is invoked and communications cannot be established between the Personal Computer and the controller.

Recovery: Check to make sure that the communications cable is correctly attached, the AML/Entry system is correctly configured for the correct communications port (either COM1: or COM2:), and that the controller is on-line.

Component: Comaid

EXTENSION MUST BE .AML

Cause: The compiler was given an input file that does not have an .AML extension.

Recovery: Rename the input file with an .AML extension

Component: Compiler

FILE ALREADY EXISTS

Cause: This message occurs if you specify an already existing filename in PUTFILE, or when you terminate the Editor session using F8 and the file you are editing is a new file.

Recovery: Specify a different filename.

Component: Editor

FILE COPIED

Cause: This message is displayed when a GETFILE/PUTFILE has successfully completed.

Recovery: None necessary

Component: Editor/Teach

FILENAME MISSING

Cause: This message occurs if GETFILE/PUTFILE is entered without specifying a filename.

Recovery: Specify a filename when entering GETFILE/PUTFILE.

Component: Editor

FILE NOT ACCESSIBLE

Cause: The file name supplied as the input file to the compiler can not be found on the diskette.

Recovery: Ensure the correct diskette is in the diskette drive and that you are using the correct file name.

Component: Compiler

FILE TOO LARGE

Cause: Your program is too large for the editor. Program size is limited to 500 lines for Personal Computers with 192 KBytes of memory, 800 lines for Personal Computers with 256 KBytes.

Recovery: Consider combining logic into subroutines or placing multiple statements on a line.

Component: Editor

FILE NOT FOUND

Cause: A file name was given which can not be found on the specified drive.

Recovery: Ensure that the correct diskette is in the diskette drive. Ensure that the correct drive was specified in the file name.

Component: Editor

FILENAME MUST HAVE .ASC EXTENSION

Cause: You tried to load a program to the controller that did not have an .ASC extension.

Recovery: Ensure that the correct diskette is in the diskette drive. Ensure that there were no errors when the program compiled.

Component: Comaid

FILESPEC TOO LONG

Cause: The file name supplied as the input file to the compiler is more than 14 characters long.

Recovery: Ensure the correct file name was entered. Rename the file with a name less than 14 characters long.

Component: Compiler

GENERATE LISTING FILE (.LST)?

Cause: This is a normal message from the compiler. If you answer "Y" or "y" the compiler generates a listing file containing the original AML program and any error messages. Any other input is interpreted as a "no". If the file is created, it has the same name as the input file, but the extension is .LST.

Recovery: Type in the answer and press the enter key.

Component: Compiler

GENERATE SYMBOL TABLE (.SYM)?

Cause: This is a normal message from the compiler. If you answer "Y" or "y", a symbol table is produced. It contains a map of the AML/Entry program structure and a list of the variables in the program. Any other input is interpreted as a "no." If the file is created, it has the same name as the input file, but the extension is .SYM.

Recovery: Type in the answer and press the enter key.

Component: Compiler

ILLEGAL FILESPEC GIVEN

Cause: A file name was used that does not follow the rules for naming files.

Recovery: Ensure the file name has less than eight letters and no special characters.

Component: Editor

ILLEGAL OP ON FIRST LINE

Cause: A line command was used on the first line of the file.

Recovery: You may have entered a D , M , C ,or R line command on line number 1. This line can not be moved, deleted, copied, or repeated.

Component: Editpr

ILLEGAL OP ON LAST LINE

Cause: A line command was used on the last line of the file. This line can not be moved, deleted, copied, or repeated.

Recovery: You wly have entered a D , M , C ,or R line command on the last line.

Component: EditOr

ILLEGAL VALUE ENTERED. REENTER.

Cause: This meseage appears when the user enters either a negative offset or number of variables to be printed that were received from the controller via POT operations.

Recovery: Enter new values, making sure both are non-negative.

Component: Comaid

INCLUDING FILE: filespec

Cause: This message is produced during "Reading Input File" when an Include command is encountered. It indicates that the compiler is now reading from that include file.

Recovery: None necessary.

Component: Compiler

INPUT FILE IS EMPTY

Cause: This message is produced when an .AML file that is empty is compiled.

Recovery: Check to make sure that the correct file was specified and that it contains an AML/Entry program.

Component: Compiler

INSUFFICIENT MEMORY 192K IS MINIMUM

Cause: The IBM Personal Computer does not have enough memory.

Recovery: 192KBytes of IBM Personal Computer memory is the minimum required for AML/Entry Version 4. Ensure the Personal Computer has the correct configuration.

Component: Menu

INVALID CHANGE COMMAND

Cause: This message is produced by an improper format for the CHANGE command.

Recovery: Re-enter command using correct format.

Component: Editor

INVALID COLUMN SPEC.

Cause: You have used a number that is not within the range of 1 to 72 for column values as part of an **F** or C primary command.

Recovery: Specify a column between 1 and 72.

Component: Editor

INVALID HOME CALCULATION

Cause: This message occurs if upon entry into Teach from the Editor using PF6, it was determined by Teach that the arm could not be homed due to hardware Modifications.

Recovery: Contact your IBM representative.

Component: Editor/Teach

INVALID LINE COMMAND

Cause: An invalid line command has been entered in the line command input field.

Recovery: Use the space bar and blank out the invalid command, then press F3 (reset key).

Component: Editor

INVALID PORT NUMBER

Cause: A digital output port outside of the allowed range for your system has been specified.

Recovery: Change the port number to a number within the range for your system, or press the End key to abort the DO control mode.

Component: Editor

INVALID PRIMARY COMMAND

Cause: An incorrect command has been entered on the command input field.

Recovery: Retype the command.

Component: Editor

LAST DATA DRIVE OPERATION WAS

Cause: As data drive requests are received from the controller, they are printed on the screen. This informs the user what the most recent data drive command is.

Recovery: None necessary.

Component: Comaid

LINE:

Cause: LINE is used to show the current line number of the AML program being processed.

Recovery: None required

Component: Compiler

LINE COMMAND CONFLICT

Cause: Several conflicting line commands were entered at the same time. For example, two **A** line commands are entered at the same time.

Recovery: Use the space bar to blank out the line commands and retype the correct commands.

Component: Editor

LINE NOT FOUND

Cause: A line was requested, using the **L** line command, which does not exist in the file.

Recovery: Enter the correct line number.

Component: Editor

MANIPULATOR POWER IS OFF

Cause: The manipulator is unable to teach because Manip Power is off.

Recovery: Press the Manip Power button.

Component: EditOr

MESSAGE FILE xxxx HAS BEEN DAMAGED. GET A NEW COPY FROM YOUR BACKUP DISKETTE.

Cause: This message occurs when one of the three message files, MSGCMP.TXT, MSGZED2.TXT, or MSGCOM2.TXT has been altered by the user.

Recovery: When this happens, reload the file from a backup diskette or the original AMI4Entry ship diskettes.

Component: Configuration Utility

NO DATA DRIVE PERFORMED

Cause: This message appears when data drive mode is exited and no data drive request was received from the controller.

Recovery: None necessary.

Component: Comaid

NO LINE(S) SPECIFIED

Cause: This message occurs when a GETFILE/PUTFILE is entered and C, CC, A, or B have not been specified. message file.

Recovery: Specify C, CC, A, or B before entering GETFILE/PUTFILE.

Component: Editor

NOW DOWNLOADING

Cause: This message flashes on the screen as Comaid downloads a program.

Recovery: None necessary.

Component: Comaid

NOW PERFORMING:

Cause: This appears when a command file is invoked from Comaid. Each line that is read in from the file is echoed to the screen, following this message.

Recovery: None necessary.

Component: Comaid

NOW PERFORMING xxxx COMMAND

Cause: This message flashes on the screen as Comaid performs a read, execute, control, or teach command.

Recovery: None necessary.

Component: Comaid

NOW PERFORMING R02 COMMAND TO READ CAUSE OF EOT

Cause: When an Eot is received from the controller, Comaid will automatically petform an R 02 (read reject status) to read the cause of the Eot. This is done so that a more informative error message can be printed.

Recovery: None necessary.

Component: Comaid

NOW UNLOADING PARTITION

Cause: This message flashes on the screen as Comaid unloads a partition.

Recovery: None necessary.

Component: Comaid

OBJECT MODULE SIZE=

Cause: This is the size of the output .ASC module created by the compiler. This is the amount of memory the program takes in the controller.

Recovery: None required

Component: Compiler

ONCE YOU ENABLE DATA DRIVE, YOU CANNOT SPECIFY THE VARIABLES FOR A GET COMMAND. YOU MUST FIRST PREPARE FOR THE GET. PRESS C TO CONTINUE, OR P TO PREPARE

Cause: This reminds that user that should a GET command be encountered by the executing controller application, the values that are sent to the controller must be preloaded into Comaid. Due to timing constraints, this cannot be done dynamically.

Recovery: If you have not already entered values for a GET command, and if the executing application has GET commands in it, then choose option "P" to prepare. If the executing application does not have GET commands in it, or if you have already prepared for the GET command, choose option "C" to enter data drive mode.

Component: Comaid

OUT OF PC MEMORY

Cause: This error rarely occurs, but it can occur at any time in the compilation process. It means that the compiler has run out of space in its 64K data segment and cannot continue the compilation.

Recovery: Try to reduce the number of variables and constants required by the AML/E program, the number of subroutines, and the number of PT's.

Component: Compiler

OUTPUT FILENAME

Cause: Comaid is requesting the name of the output (.ASC) file to be sent to the controller.

Recovery: Type in the filename for the output (.ASC) file and press enter.

Component: Comaid

PARTITION UNLOADED

Cause: Comaid has successfully unloaded the partition.

Recovery: None required

Component: Comaid

PLEASE CHECK FILE DEVICE

Cause: The compiler tried to read the AML source file from the specified drive and the drive was not working. This message occurs with the "CANNOT OPEN .ASC FILE" message.

Recovery: Ensure the drive for the source file is correct. Ensure the door is closed on the diskette drive.

Component: Compiler

PRESS ANY KEY TO CONTINUE

Cause: This is a normal message used to delay operation until you are ready to continue.

Recovery: Press any key when you are ready to continue.

Component: Comaid, Compiler, Configuration Utility, Editor

PRESS ANY KEY TO EXIT DATA DRIVE MODE

Cause: This message appears while the controller is in data drive mode (i.e. the Xon state). While the controller is in data drive mode, no Comaid requests can be honored, so the Comaid main menu does not appear. Instead, the controller initiated request appear on the screen as they are received.

Recovery: When ready, press any key. This will send an Xoff to the controller, causing data drive mode to be exited.

Component: Comaid

PRESS END TO EXIT

Cause: This appears at the bottom of the submenus of the Teach Utility. It reminds the user that the submenu may be exited by pressing the **End** key.

Recovery: When done specifying values for the submenu (and pressing enter to send them to the controller), press this key to return to the main Teach screen.

Component: Editor

PRESS ENTER TO MOVE TO

Cause: This appears when the RECALL+ or RECALL- feature is used in Teach mode. This tells you where the currently recalled point is located.

Recovery: Press enter. After which a second verification will appear. Confirming the second verification will cause the robot to move.

Component: Editor

PRESS ENTER TO SPECIFY

Cause: This appears at the bottom of the submenus of the Teach Utility.

Recovery: Values are first entered into the fields of the submenus (motion parameter submenu, DI/DO submenu, or DO submenu). Pressing enter will then cause the new values to be sent to the controller.

Component: Editor

PRESS "Q" TO QUIT OR ANY OTHER KEY TO RETRY ...

Cause: If "Q" is pressed the AML/Entry menu is displayed. Any other key runs the configuration utility again.

Recovery: Press the desired key.

Component: Configuration utility

PRESS "Q" TO QUIT THE UTILITY OR PRESS ANY OTHER KEY TO ENTER THE UTILITY WITH A STANDARD CONFIGURATION AND RECTIFY THE CONFLICT

Cause: The configuration utility has encountered a conflict in the files on the diskette. If you press "Q" you return to the main menu. If you press any other key the utility continues and sets the configuration to a standard configuration.

Recovery: Ensure the correct diskettes are in the diskette drive.

Component: Configuration utility

PRINT ABORTED

Cause: The print operation was aborted by the user pressing the escape (Esc) key.

Recovery: Restart the print operation if desired.

Component: Editor

PRINT COMPLETED

Cause: The print operation completed normally.

Recovery: None necessary.

Component: Editor

PRINTER NOT AVAILABLE

Cause: The compiler tried to initialize the printer and the printer did not respond.

Recovery: Ensure the printer is properly configured, attached, and turned on. Also, ensure the system board switches are properly set for your printer.

Component: Compiler

PRINTER NOT AVAILABLE- HARDCOPY REQUEST CANCELLED

Cause: While Comaid was printing the communications buffer to the printer, a printer error occurred.

Recovery: Ensure the printer is properly configured, attached, and turned on. Also, ensure the system board switches are properly set for your printer.

Component: Comaid

READING INPUT FILE

Cause: This is a normal message indicating the progress of the compilation process. This message tells you that the first of three steps for the compiler is executing.

Recovery: None necessary.

Component: Compiler

READING MESSAGE FILES

Cause: This message occurs at the beginning of the configuration utility. It is an informational message.

Recovery: None necessary.

Component: Configuration Utility

ROBOT TYPE CONFLICT

Cause: This message occurs when trying to get into Teach from the Editor using PF6 and Edit/Teach is configured for a different robot. You are unable to get into Teach.

Recovery: Reconfigure Editor/Teach for the robot you want to Teach.

Component: Editor/Teach

SHUTTING OFF DATA DRIVE

Cause: This message appears after a key is pressed in Comaid to exit data drive mode. It informs the user that the Xoff is now being sent to the controller.

Recovery: None necessary.

Component: Comaid

— — -> SKIPPING TEXT UNTIL END OF DIGIT STRING.

Cause: A digit string (that is, a number) has too many digits. The compiler is skipping to the end of the digit sequence.

Recovery: Locate the error that caused the compiler to skip ahead and retry the compiler.

Component: Compiler

— — -> SKIPPING TEXT UNTIL NEXT SEMICOLON.

Cause: An error in an AML/Entry program has caused the compiler to not understand what the programmer intended. The compiler skips ahead to the next semicolon.

Recovery: Locate the error that caused the compiler to skip ahead and retry the compiler.

Component: Compiler

STRING FOUND

Cause: The string of characters specified in the F or C command was correctly found.

Recovery: None necessary.

Component: Editor

STRING NOT FOUND

Cause: The string of characters specified in the **F** or C command cannot be located.

Recovery: Ensure the correct string has been specified.

Component: Editor

SUCCESSFUL COMPILATION

Cause: The compile operation has ended normally. An output file with the extension .ASC has been produced.

Recovery: None necessary

Component: Compiler

SUCCESSFUL TRANSMISSION TO PARTITION

Cause: This message indicates that the output file has been correctly loaded to the controller and is ready to be executed.

Recovery: None required

Component: Comaid

SYSTEM WILL NOT RUN WITH DOS 1.1 OR 1.0

Cause: The AML/Entry diskette was created using DOS 1.1 or 1.0.

Recovery: Recreate the AML/Entry diskette using DOS 2.0 or 2.1.

Component: Menu

TARGET NOT WITHIN ROBOT WORKSPACE

Cause: This message appears whenever the user attempts to move the robot to a point out of the workspace.

Recovery: When the robot is moved back within the robot workspace, by any of the mechanisms provided by the Teach Utility, this message will disappear.

Component: Editor

TEACH ABORTED

Cause: This message appears when the user aborts Teach.

Recovery: After returning to the editor, press F6 to re-enter Teach mode.

Component: Editor

TERMINATING EDITOR SESSION

Cause: This message occurs when you attempt to save a new file using the F8 key.

Recovery: None necessary.

Component: Editor

THE 75xx IS ABOUT TO MOVE TO POINT xxxx PRESS ENTER TO MOVE OR ANY OTHER KEY TO ABORT.

Cause: This message appears when the RECALL+ or RECALL- feature is chosen (keys **F3** or **F4**). It is warning you that the robot is about to move.

Recovery: Pressing enter will cause the move to be performed.

Component: Editor

THE 7545-S IS ABOUT TO SWITCH ARM CONFIGURATION PRESS ENTER TO MOVE OR ANY OTHER KEY TO ABORT.

Cause: This appears when the **TAB** key has been used to request a switch of arm configuration for the 7545-S.

Recovery: Pressing enter will cause the 7545-S to switch arm configuration. The robot will temporarily move from the current point, but will end up back at the same point.

Component: Editor

THERE IS A CONFLICT BETWEEN THE CONFIGURATIONS SHOWN IN THE THREE MESSAGE FILES

Cause: The message files from the compiler, the editor, and comaid do not reflect the same configuration.

Recovery: Use the configuration utility to specify the proper configuration for your system.

Component: Configuration Utility

TOO MANY FILES

Cause: The diskette specified by a SAVE command has the maximum number of files allowed already on it.

Recovery: Designate the other drive to receive the file, substitute another formatted diskette or delete some files.

This message can also result from an invalid file name during a SAVE when you exit from the editor.

Component: Editor

TOO MANY VARIABLES REQUESTED - PLEASE REENTER

Cause: This message appears when the user requests more than 400 variables to be sent with the C 80 control command or read with the R 80 read command.

Recovery: Perform two separate C 80 or R 80 commands.

Component: Comaid

UNABLE TO PRINT

Cause: The Personal Computer attempted to print a program listing or an error message but was unable to because of one of the following:

- No printer is available.
- The printer power-on switch is not in the ON position.
- The printer ONLINE key is not activated.
- The printer cable is not attached.
- The printer is out of paper.

Recovery: Correct the problem with the printer and retry the operation.

Component: Editor

UNABLE TO READ CAUSE OF EOT

Cause: This message appears when Comaid attempts to perform a R 02 command to read the cause of an Eot, and the controller disallows the R 02 command.

Recovery: None necessary.

Component: Comaid

UNABLE TO READ CFG.EXE TRY AGAIN

Cause: The configuration option was called from the AML/Entry menu, but the CFG.EXE file is not on any diskette installed in the Personal Computer.

Recovery: Place the AML/Entry diskette containing the CFG.EXE file in the Personal Computer and retry the configuration option.

Component: Menu

UNABLE TO READ COMAID.EXE TRY AGAIN

Cause: The Comaid option was selected from the AML/Entry menu, but the COMAID.EXE file is not on any diskette installed in the Personal Computer.

Recovery: Place the AML/Entry diskette containing the COMAID.EXE file in the Personal Computer and retry the compiler option.

Component: Menu

UNABLE TO READ COMPILER.EXE TRY AGAIN

Cause: The compiler option was selected from the AML/Entry menu, but the COMPILER.EXE file is not on any diskette installed in the Personal Computer.

Recovery: Place the AML/Entry diskette containing the COMPILER.EXE file in the Personal Computer and retry the compiler option.

Component: Menu

UNABLE TO READ EDIT.EXE TRY AGAIN

Cause: The edit option was selected from the AML/Entry menu, but the EDIT.EXE file is not on any diskette installed in the Personal Computer.

Recovery: Place the AML/Entry diskette containing the EDIT.EXE file in the Personal Computer and retry the edit option.

Component: Menu

UNABLE TO READ MESSAGE FILE

Cause: The edit option was selected but either the msgzedl.txt or the msgzed2.txt file is not in any diskette installed in the Personal Computer.

Recovery: Ensure the correct diskette is in the drive.

Component: Editor

UNABLE TO READ XREF.EXE TRY AGAIN

Cause: The XREF option was selected from the AML/Entry menu, but the XREF.EXE file is not on any diskette installed in the Personal Computer.

Recovery: Pl'ace the AML/Entry diskette containing the XREF.EXE file in the Personal Computer and retry the edit option.

Component: Menu

UNKNOWN UNIT

Cause: This error occurs when an attempt is made to access a file on an unknown unit.

Recovery: Retry compiling the AML/Entry program.

Component: Compiler

WARNING: AGGREGATE OF ALL FORMALS IS TYPED AS REAL

Cause: An *aggregate* that contains only formal parameters, none of which have been used before this definition. In this situation, the formal parameters default to the type "real," and the aggregate is considered a set of real numbers.

Recovery: This is a warning only. Ensure that the default typing is correct for your program.

Component: Compiler

WARNING: FORMAL PARAMETER LIST ITEM NOT USED:

Cause: In a call to a subroutine, one of the parameters being passed is not used by the called subroutine. The number that follows the colon indicates which parameter was not used; parameters are numbered from left to right, starting with 1.

Recovery: This is a warning only, the compiler will still complete. Ensure the parameter that is not being used is intended to be skipped.

Component: Compiler

WARNING: ROBOT MOVING TO TAUGHT POINT

Cause: This message is printed when the robot is sent a Teach command from Comaid.

Recovery: Make sure that no one is in the workspace of the manipulator when a Teach command is performed.

Component: Comaid

WARNING: THIS COMMAND SHOULD NOT BE USED WITH THIS MACHINE TYPE.

Cause: The program contains a **LEFT** or **RIGHT** command, but the system is configured for a 7545 or 7547. Only the 7545-800S permits the use of these commands.

Recovery: Delete the **LEFT** or **RIGHT** commands or reconfigure the system for the 7545-800S manipulator. This is just a warning message. No code will be produced by the AML/Entry Compiler for the **LEFT** or **RIGHT** command that caused the warning.

Component: Compiler

WARNING: N LINES WERE TRUNCATED TO 72 CHARACTERS

Cause: This message appears when the editor loads a file that contains a line which is longer than 72 characters (because the AML/E editor can only process 72 characters per line).

Recovery: This is a warning only. When this message appears, data from the original file has been lost; only the first 72 characters of each line will be included in the new file. If you want to keep the original file intact, use the CANCEL command to cancel the editing session.

Component: Editor

WOULD YOU LIKE A HARDCOPY OF THE COMMUNICATIONS TRANSACTIONS (YIN)?

Cause: This message appears when the P option of Comaid is selected. If so chosen, the communications transactions will be printed on the printer in addition to on the screen.

Recovery: Enter "Y" if you would like the communications transactions to be printed on the printer; otherwise enter "N".

Component: Comaid

WRITE-PROTECT VIOLATION

Cause: The compiler tried to write on a diskette that is write-protected.

Recovery: Remove the write-protect tab from the diskette, or use a diskette that is not write-protected.

Component: Compiler

WRITING .ASC FILE

Cause: This is a normal message indicating the progress of the compilation process. This message tells you that the last of three steps for the compiler is executing.

Recovery: None necessary.

Component: Compiler

NUMBERED MESSAGES

ERR1: missing left parenthesis

Explanation: A left parenthesis '(' was expected, but not found.

ERR2: missing right parenthesis.

Explanation: A right parenthesis ')' was expected, but not found.

ERR3: missing semicolon

Explanation: a semicolon ';' was expected, but not found.

ERR4: missing comma

Explanation: A comma ',' was expected, but not found.

ERR005: missing single quote

Explanation: A quote mark (') was expected, but not found.

ERR10: SUBR declaration must be alone on a line.

Explanation: Only a comment may follow a SUBR declaration. Any other statements must appear on the following line(s).

ERR11: END must be alone on a line

Explanation: Only a comment may follow an END statement. Any other statements must appear on the following line(s).

ERR012: declaration not allowed in code section

Explanation: The organization rules for a subroutine require that all variable declarations be made before any commands are used. This message identifies a declaration that has occurred after a command statement.

ERR13: END does not have a matching SUBR

Explanation: An END statement was encountered, but no subroutine is open. Check to ensure that there is only one END for each SUBR statement.

ERR14: executable may not precede first SUBR

Explanation: An executable command (for example, PMOVE, WAITI, etc.) was found outside a subroutine. These commands must be inside a subroutine.

ERR015: outermost subroutine may not have formal parameters

Explanation: The first SUBR may not have a list of parameters because there is no way for the subroutine to be called.

ERR21: invalid use of a symbol

Explanation: A symbolic name was used in a way that is not allowed.

ERR22: no name for definition

Explanation: A variable was defined, but no name was supplied for it.

ERR23: referenced name is not in scope

Explanation: A symbolic name that is not defined in the current scope was used. Only global or local symbols, formal parameters, and accessible subroutines are considered to be in the current scope.

ERR24: referenced type can not be used for NEW assignment

Explanation: A variable was defined with the NEW command to be the same as another already defined variable; however, the existing type is not suitable. For example, a counter may not be used to define a NEW variable.

ERR25: definition object already defined

Explanation: An attempt was made to define a variable that is already defined.

ERR26: invalid definition

Explanation: The form of the definition is improper. Refer to Chapter 4 which describes the proper way to define a variable.

ERR27: global symbol may not be redefined

Explanation: An attempt to define a symbolic name is not permitted because the name is a global variable. Global variables may not be redefined.

ERR28: name belongs to an accessible subroutine

Explanation: An attempt to define a symbolic name is not permitted because it is the name of a subroutine that is defined in the current scope.

ERR30: too many digits

Explanation: Numbers may only have seven significant digits.

ERR31: expecting numeric type

Explanation: In a location where a number is expected, a plus or minus sign preceded a variable.

ERR032: expecting numeric type: formal parameter not allowed

Explanation: In a location where a number is expected, a plus or minus sign preceded a symbol; the symbol was a formal parameter. This is not allowed.

ERR035: symbol too long – has been truncated

Explanation: A symbol appeared with more than 72 characters. Only the first 72 characters are being used to identify the symbol.

ERR40: bad aggregate structure

Explanation: An aggregate has a structural error. Refer to Chapter 4 for the description of declaring an aggregate.

ERR41: nested aggregate not allowed.

Explanation: An aggregate may not contain another aggregate.

ERR42: premature end of aggregate.

Explanation: An aggregate was terminated without a right bracket '>'

ERR43: illegal type in aggregate

Explanation: An aggregate can be composed of certain data types. A data type not allowed for use in aggregates was used. An aggregate can only contain numbers, points, or counters.

ERR044: element does not match aggregate type

Explanation: All the elements of an aggregate must be the same data type. The current element does not match that of the aggregate.

ERR45: formal parameter can not be assigned current type

Explanation: A formal parameter appearing in an aggregate can not be used in this particular aggregate because the aggregate type is not allowed for a formal parameter; for example, an aggregate of labels.

ERR46: previously used formal parameter does not match type

Explanation: A formal parameter appearing in an aggregate has been used previously, and its type does not match the aggregate type.

ERR050: illegal form _ must be a single symbolic name

Explanation: A string type must be a single name; that is, no spaces are allowed within the two quote marks.

ERR60: too few arguments supplied in a PT definition

Explanation: Less than the correct number of parameters were used to specify a PT. A PT is defined with four coordinates: X, Y, Z, and roll on the 7545, 7547, and 7545-800S manipulator.

ERR61: too many arguments supplied in a PT definition

Explanation: More than the correct number of parameters were used to specify a PT. A PT is defined with four coordinates: X, Y, Z, and roll on the 7545, 7547, and 7545-800S manipulators.

ERR80: DPMOVE: aggregate elements are improper type

Explanation: The aggregate supplied to DPMOVE was not an aggregate of numbers or counters and it must be.

ERR81: DPMOVE: aggregate contains too many elements

Explanation: The aggregate supplied to DPMOVE contained more than the correct number of elements. Four elements (X, Y, Z, and roll) are needed on the 7545, 7547, and 7545-800S.

ERR082: DPMOVE: aggregate contains too few elements

Explanation: The aggregate supplied to DPMOVE contained less than the required four elements (X, Y, Z, and roll). Ensure the AML/Entry system is configured for the correct manipulator type.

ERR100: empty parameter list

Explanation: A parameter list was opened with a left parenthesis, but there were no entries in the list.

ERR101: parameter list structure is invalid

Explanation: The structure of a parameter list is erroneous. Refer to Chapter 4 for a description of lists.

ERR104: empty field in parameter list/extra separator

Explanation: An empty field was found in a parameter list. This could be either a missing argument or an extra comma.

ERR110: iterate: function must be a string type

Explanation: The function or subroutine specification for the Iterate command was not specified as a string. It may be either a constant that has been previously defined as a string, or in the Iterate statement within single quote marks.

ERR111: iterate: function not legal for use with iterate

Explanation: An AML/Entry function which is not allowed for use in an Iterate statement has been used.

ERR112: iterate: aggregate of conflicting size

Explanation: All of the aggregates in an Iterate statement must have the same number of elements. The indicated aggregate does not have the correct number.

ERR150: parameter name already used in this formal list

Explanation: The formal parameter used has already appeared in the list in a SUBR statement.

ERR151: actual and formal parameter lists are of different size

Explanation: In a subroutine call, the calling statement does not supply the same number of parameters as the subroutine expects.

ERR152: actual and formal parameter list item do not match:

Explanation: In a subroutine call, the indicated parameter does not have the same type as the subroutine expects.

ERR153: actual parameter is not a passable type:

Explanation: In a subroutine call, the indicated parameter is not allowed to be passed to a subroutine.

ERR200: can not call self

Explanation: A call to a subroutine can not be made from within that subroutine.

ERR201: can not call owner

Explanation: A call to a subroutine can not be made if the called subroutine has defined the present subroutine.

ERR210: called subroutine is not reachable

Explanation: A call to a subroutine can not be made because the called subroutine is not defined in the current scope.

ERR300: too few arguments supplied for this function

Explanation: The function requires more parameters than it was given.

ERR301: too many arguments supplied for this function

Explanation: The function requires less parameters than it was given.

ERR310: formal parameter is not legal here

Explanation: The indicated function argument is a formal parameter, however this particular function argument may not be formal paramqer.

ERR311: already used formal parameter is improper type

Explanation: A formal parameter that has been used as one data type is now being used where that data type is not allowed.

ERR312: formal parameter type is not yet determined.

Explanation: This message is produced if a subroutine uses a formal parameter in a command that does not assign the type before the type is established. For example, the commands GET and PUT do not assign a formal parameter a data type. If a formal parameter is used in one of these commands before being used anywhere else, this message results.

ERR320: argument type is illegal here

Explanation: The data type of a function argument is not appropriate for the function parameter; for example, a DELAY command with a point argument.

ERR330: argument value is out of range

Explanation: A numeric value either too large or too small has been used in a function.

ERR331: argument of type real requires rounding

Explanation: A real value that has more precision than is allowed with this function has been used; this occurs most often when a real number is used where an integer is required.

ERR340: invalid operator.

Explanation: This message is produced if an unknown operator is used in the COMPC command (such as <<).

ERR341: operator expected but not found.

Explanation: This message is produced if an operator is not found in the COMPC statement.

ERR400: BRANCH:target is not a label

Explanation: The name used in a BRANCH statement is previously defined, but is not a label.

ERR410: BRANCH:target may not be a formal parameter

Explanation: The name used in a BRANCH statement is a formal parameter; labels cannot be passed as parameters.

ERR420: BRANCH:forward reference not resolved in line:

Explanation: The name used in a BRANCH statement on the indicated line never appeared in the subroutine as a label.

ERR500: group argument has no index.

Explanation: This message is produced when a group data type is used, but no index is supplied. The only place that a group may be used without an index is with GET and PUT.

ERR501: index to group is not valid data type.

Explanation: This message is produced when the index to a group is an invalid data type. An index can only be an integer, a counter, or the element of a group of counters (assume 'c' is a counter and 'g' a group of counters, then the following is legal: g(g(c)), but if a 'p' is a pt, then g(p) is invalid).

ERR505: argument does not match group type

Explanation: This message is produced when an element within a group is not the same as the initial element in the group. For example, mixing points and constants or points and counters.

ERR600: Illegal expression - operator expected.

Explanation: This error message indicates that an operator is missing in an expression. This will happen when two of the following appear consecutively - a counter, a function call, or a constant. An operator must appear between the items.

ERR601: Illegal expression – invalid left parenthesis.

Explanation: This occurs when a left parenthesis occurs in an invalid location. If a left parenthesis follows a constant or a counter, then this error is printed. Another situation which will cause this to appear is if a program attempts to use a formal parameter as a group. Formal parameters are not allowed to be groups, thus the left parenthesis is invalid.

ERR602: Illegal expression – unbalanced parentheses.

Explanation: This occurs when the end of an expression is reached and the number of right parentheses is less than the number of left parentheses.

ERR603: Illegal expression – invalid term after right parenthesis.

Explanation: This occurs when a function call, constant, or counter immediately follows a right parenthesis (one that was not part of a function call). If a function call, constant, or counter immediately follows the right parenthesis of a function call, then ERR600 is generated. As with ERR600, an operator is probably missing.

ERR604: Illegal expression – operator expected between parentheses.

Explanation: This occurs when a right parenthesis is followed by a left parenthesis. If the right parenthesis was part of a function call, then ERR601 is generated. For example, (A+B)(C+D) will cause this error. Either an operator may be missing or one of the parenthesis may accidentally be the wrong one.

ERR605: Illegal expression - multiplication operation illegal here.

Explanation: This occurs when a multiplication operator occurs in an illegal location. Multiplication operators must always follow a counter, right parenthesis, or a constant.

ER11606: Illegal expression _ division operation illegal here.

Explanation: This occurs when a division operator occurs in an illegal location. Division operators must always follow a counter, right parenthesis, or a constant.

ER11607: Illegal expression _ invalid right parenthesis.

Explanation: This occurs when a right parenthesis follows an addition, subtraction, multiplication, or division operator. Either the operator needs to be removed or an operand (counter, constant, function call, etc) needs to be inserted.

ERR608: Illegal group reference _ right parenthesis expected.

Explanation: This occurs when a group reference is not properly terminated by a right parenthesis.

ERR609: Illegal function call _ right parenthesis expected.

Explanation: This occurs when a function call is not properly terminated by a right parenthesis.

ERR610: Expression expected.

Explanation: This occurs when expression is missing. Null expressions are not permitted.

ERR611: Expression too large, break into subexpresslons.

Explanation: This error occurs when the expression is too complex to allow code to be generated for the robot controller. When this error occurs, break the expression into smaller components. This error rarely **Occurs.**

ERR612: Illegal expression - extra right parenthesis.

Explanation: This occurs when an expression contains an extra right parenthesis. The user should check to make sure a left parenthesis is not missing.

ERR613: Illegal expression - unexpected term.

Explanation: This occurs when an illegal term appears in an expression. An expression must consist of symbols (e.g., counters), operators (e.g., +, *, and /), built-in functions, constants, and parentheses.

ERR800: unexpected: can not be properly interpreted

Explanation: The indicated item is out of place; the compiler can not interpret what it is supposed to be.

ERR900: premature end-of-file (i.e. insufficient ENDs)

Explanation: The end of the file has been reached, but a number of subroutines remain open; there is not an END for every SUBR.

ERR910: closing END is not last in file

Explanation: The END statement that corresponds to the first (outermost) subroutine has been encountered, but more data remains in the AML file.

ERR950: **** Object Module Too Large

Explanation: The program is larger than the maximum amount of memory available in the controller.

ERR951: Object Module too large for PC.

Explanation: The .ASC file that the compiler was building became too large for the compiler to continue. This can occur when the AML/Entry system is being used with a PC that has less than 256K, or when the user has installed several resident programs in memory in addition to the AML/Entry system (i.e. VDISK or RDT). Either decrease the size of the AML/Entry program, remove some of the other modules from resident memory, or add more memory to your computer.

ERR970 to ERR999

Explanation: These messages are generated if the compiler detects an internal error condition. If one appears, please report this to your IBM representative. The messages are all of a similar form:

```
ERR9xx: compiler error - PLEASE REPORT TO IBM
```

AMLECOMM/COMAID ERROR MESSAGES

This section contains the AMLECOMM error messages. It does not contain a discussion of the errors that could occur in the executing application controlling the manipulator, as these are discussed in the AML/E System Documentation. Because COMAID uses AMLECOMM, these error messages will also appear during the execution of COMAID.

ERROR 1 - Initialization Error

Explanation: This error occurs either when the user application calls AMLECOMM for initialization (OPERATION.A$="") and AMLECOMM has already been initialized or if the communications port specified by ADAPT.A$ cannot be opened.

ERROR 2 - CONVERT1 or CONVERT2 Not Loaded

Explanation: This error will occur only if AMLECOMM is being used interpretively with BASICA, and the user forgot to install one or both of the numeric conversion routines CONVERT1 and CONVERT2. The remedy is to install these by returning to DOS and entering each of these commands to invoke the corresponding .COM files to install each of these modules.

ERROR 3 - Illegal Function Requested

Explanation: The user application has requested an illegal function. This primarily occurs because the user did not properly configure the version of AMLECOMM that is being used. For example, requesting a R record from a version of AMLECOMM not configured for R records will cause this error.

ERROR 4 - Illegal Partition Number Specified

Explanation: This error occurs when a partition specified in PART.A$ to either an unload or download request was illegal. PART.A$ must be either "1", "2", "3", "4", or "5".

ERROR 5 - Basic File Error (52 or 55) using COMM.A or OPENFCOMM.A

Explanation: This error occurs when Basic Error number 52 or 55 occurs. This error will occur most commonly if AMLECOMM is called and the port specified by ADAPT.A$ was not opened or abnormally closed. See the description of these errors in Appendix A of the Basic reference manual. When this occurs, the user should CLOSE COMM.A and reopen the COM port by setting OPERATION.A$=OPEN.A$

BAUD 4800
PARITY E
DATABIT 7
STOPBIT 2

ERROR 6 - Disk I/O Error (71 or 72)

Explanation: This error occurs when Basic Error number 71 or 72 occurs. This error occurs whenever a disk I/O error occurs. See the description of these errors in Appendix A of the Basic reference manual. When this occurs, the user should make sure the diskette is inserted properly in the disk drive and retry the operation.

ERROR 7 - Data Drive Must First Be Turned Off

Explanation: This error occurs when the user application calls AMLECOMM with OPERATION.A$ not equal to D.A$ and data drive mode is active. That is, once the user places AMLECOMM in data drive mode, no request other than a data drive request can be honored. The remedy is to turn data drive off by setting OPERATION.A$=D.A$ and DDSWITCH.A%=OFF.A%.

ERROR 8 - New Function Denied: Old Funct. Pending

Explanation: This error is similar to number 7. It occurs when an old request is still pending (i.e. an X request with posting) and the user application is requesting a new function. The user application must allow the old request to complete before a new one can be accepted. The remedy is to poll AMLECOMM until STATUS.A%=IDLE.A%.

ERROR 9 - Illegal Port Number or Value for DO

Explanation: This error occurs when the user application is attempting to set either an illegal DO port or a legal DO port to an illegal value. The DO port specified by PORTNUM.A% must be from 1,2,3...127. The value specified by PORTVAL.A% must bo 0 or 1. Note that this error will not occur if the user application specifies a DO port greater than the number installed, but still within the 1 to 127 range. In this case the controller will send back an EOT and ERROR 19 is returned to the user.

ERROR 10 - File to Download Not Found

Explanation: This error occurs when the file specified in NAME.A$ on a download request cannot be found. The string in NAME.A$ is simply passed to the Basic OPEN command, so check your level of DOS to make sure the filename meets the requirements.

ERROR 11 - Too Many Variables to Fit in RVARS or PVARS

Explanation: This error occurs when more variables are sent from the controller than can be fit into RVARS.A or PVARS.A. The remedy is to increase the value of VARMAX.A in AMLECOMO.BAS, adjust the DIM statements in AMLECOMO.BAS that depend on VARMAX.A (i.e. RVARS.A,PVARS.A,GVARS.A, and C8OVARS.A), and re-adjust VARMAXRECS.A. The configuration utility for AMLECOMM will have to be rerun to incorporate the updated values into the program.

ERROR 12 - Too Many D Records to Fit in RRVARS$

Explanation: When D records are received from the controller, they are placed in RRVARS.A$. The value of VARMAXRECS.A indicates the size of this array. When this error occurs, raise the value of VARMAXRECS.A and change the DIM statement for RRVARS.A$ in AMLECOMO.BAS. VARMAXRECS.A should be set to INT(VARMAX.A/4)+1. It must also be greater than or equal to 8.

This message can also appear from within Comaid, since Comaid uses Amlecomm. Comaid is configured to provide access to 400 variables (which corresponds to an RRVARS of size 101). If more than 404 variables are sent from the controller to the host (either from a PUT command or a read all program variables request), then this error appears.

ERROR 13 - Illegal VARCNT or VAROFFS

Explanation: This error occurs when either VARCNT.A% or VAROFFS.A% is negative on a request to read variables. Each of these values must be non-negative.

ERROR 14 - Hardware Error

Explanation: This error occurs when one of the signals from the RS-232 cable are lost. It occurs when Basic Error numbers 24, 25, 57, or 68 occur. When this occurs, the user application can attempt error recovery by closing file COMM.A, and trying to reopen the ADAPT.A$ communcations port by setting OPERATION.A$=OPEN.A$. If this works then a reset operation should be considered to clear any errors that may have resulted from the hardware error in the controller. If opening the port fails, then a true hardware error has occurred (e.g., the cable has been removed or manipUlator power has been lost), and an operator should be notified. Please see the Basic Reference for more on errors 24, 25, 57, and 68.

ERROR 15 - Communicationes Line Data Error

Explanation: This occurs when a parity error occurs on a record received from the controller. This communication error is rare, and when it occurs the user application should call AMLECOMM to reset any TE error that may have resulted (via the RSTERR.A$ X request). If this occurs often, a faulty cable could be the problem.

ERROR 16 - Controller Data Drive Response Wrong

Explanation: This occurs when the controller has responded incorrectly in a data drive scenario. When this occurs the circular transaction buffer BUFFER.A$ should be inspected to verify that indeed the controller responded incorrectly. If the error can be recreated, than it should be reported to IBM.

ERROR 17 - Receive COntroller Record Error

Explanation: Thisl error occurs when on a third retry to send data to the controller, a 3 second timeout occurs and no response is received from the controller. The user should make sure the controller is ON-LINE, and no error state exists (no LED's on).

ERROR 18 - Checksum 'Error on Received Cntrlr Recd

Explanation: This error occurs when a checksum error occurs on a received D record or the controller NAK's a D record during the download operation 3 consecutive times. If this error occurs because of the download operation, than the .ASC file has probably been corrupted and they corresponding source file should be recompiled. If this error occurs because of a checksum error on a received D record from the controller, than the user application should clear any TE error that may have resulted (by using the RSTERR X request), and retry the request that caused this error.

ERROR 19 - Controller Has Sent an EOT to You

Explanation: This is the most common error, and occurs whenever an EOT has been sent by the controller. An EOT is sent when by the controller when a message received from the PC, but it cannot be honored. Examples are sending a T record when an application is running, or sending a R request that is in error (i.e. R13). The user should issue the RSTAT request (R02) which will return the cause of the EOT.

ERROR 20 - Robot Controller Response Timeout

Explanation: This occurs when we are awaiting data from the controller and it never arrives. Check that the controller is ON-LINE and no errors exist (no LED's lit). If this is the case and the error can be recreated, than it should be reported to IBM.

ERROR 21 - Three NAKs Received From the Controller

Explanation: This error occurs when the controller sends a NAK to each retry of a transaction in the communication protocol. If this error occurs often, then a faulty cable could be the problem.

ERROR 22 – Controller XOFF Timeout

Explanation: This error occurs when 30 seconds expire after an XOFF has been received from the controller, and another XOFF or an XON does not arrive. If this error can be recreated than it should be reported to IBM.

ERROR 23 – Illegal Data Drive Request

Explanation: This error occurs when the controller sends a D record beginning a data drive transaction that is not supported by AML/E V4.0. This should never be the case, and if it can be recreated, than it should be reported to IBM.

ERROR 24 – Incorrect Length Record Received From Controller

Explanation: This error occurs when the size of a D record sent by the controller does not coincide with the size specified by the length field of the D record. When this error occurs, the user application should reset any TE that may have resulted by using the RSTERR X request and retry the operation. If it can be recreated, than it should be reported to IBM.

ERRORS 25–29 – Unused

Explanation: These error numbers will never be returned. They are reserved for future expansion.

ERROR 30 – AMLECOMM Error, Please Report to IBM.

Explanation: This error number will be returned when the AMLECOMM error trap is invoked with an ERR that is not 24, 25, 52, 55, 57, 68, 71, or 72 and an ERL that is one of the I/O statements. This "soft" error is one that could be recovered from by the user's application had a more specific error number been returned. The user should report this to IBM so that the error handler can be enhanced to cover the error that has been discovered. The actual error number will be in IBMERR.A and the actual error line will be in IBMERL.A.

MSGCOM.TXT

The file MSGCOM.TXT is provided to the user to give his application on-line access to many of the AML/E, and AMLECOMM errors. Five sets of data reside in MSGCOM.TXT.

- The first 30 lines contain explanations for the 30 errors described in the preceding section.
- The next 7 lines contain explanations for the 7 bits of the machine status error word. They are given from the most significant bit (HEX "80" - Servo Error) to the least significant bit (HEX "02" - AML/Entry Error). It should be mentioned that when an AML/Entry error occurs, the control word will also flag the data error bit, giving a value of HEX "06" instead of just HEX "02".
- The next 22 lines contain the RSTAT (R02) reject status error codes. Each line is prefixed by the two character code and a blank before the actual English explanation.
- The last 11 lines contain the AML/E Data errors.
- The last 7 lines contain the AML/E Version 4 errors. As with the R02 messages, each line is prefixed by a two character code and a blank.

It is recommended that the user browse the file MENUCOMM.BAS to see how to read, store, and later print out appropriate error messages.

APPENDIX C. VALUES FOR THE LINEAR COMMAND

This appendix contains the linear rate values for the IBM7545, 7545-800S, and 7547 Manufacturing Systems. The following table gives arm speed at tool tip and the straight line error for the different LINEAR command values in both millimeters and inches. Refer to Chapter 4, "Learning the AML/Entry Language" and to Appendix A, "Command/Keyword Reference" for a description of the LINEAR command.

IBM Manufacturing Systems LINEAR Rate Values		
Programmed rate	Arm speed at tool tip mm/sec. (in./sec)	Straight line error mm (in.)
1	60 (2.4)	3.0 (0.12)
	100 (3.9)	3.7 (0.15)
3	140 (5.5)	4.4 (0.17)
4	180 (7.1)	5.3 (0.21)
5	225 (8.9)	6.2 (0.24)
6	265 (10.4)	6.9 (0.27)
7	305 (12.0)	7.6 (0.30)
8	345 (13.6)	8.4 (0.33)
9	385 (15.2)	9.3 (0.37)
10	430 (16.9)	10.0 (0.39)
20	430 (16.9)	11.5 (0.45)
30	430 (16.9)	11.5 (0.45)
50	430 (16.9)	11.5 (0.45)
0	Exit linear speed and motion	

Notes:

1. The straight line error in the table is the maximum deviation from the straight line (defined by two end points) that the system travels while moving in one direction.

2. Speeds in the table are for planning purposes only and are typical minimum values. Linear speed can not be controlled by the user. Linear speed values vary in different locations in the workspace. Therefore, you cannot expect movement at a constant speed when performing long moves using the LINEAR command.

APPENDIX D. VALUES FOR THE PAYLOAD COMMAND

7545 PROGRAM SPEED VALUES FOR PAYLOAD COMMAND

7545 Program Speed Values ter PAYLOAD Command					
Prorate speed values	Speed of 91 at the kW tip mm/sec fln/aec)	Speed of 92 at the tool tip mm/sec (In/sec)	Roll axis speed deg/sec	Z-axis speed mm/sec On/ac)	Maximum payload for speed AI MI
1	300 (11.8)	200 (7.9)	100	60 (2.4)	10 (22)
2	500 (19.7)	380 (15.0)	170	100 (3.9)	10 (22)
3	700 (27.6)	530 (20.9)	240	140 (5.5)	10 (22)
4	820 (32.3)	620 (24.4)	280	165 (6.5)	10 (22)
5	900 (35.4)	700 (27.6)	310	180 (7.1)	8 (17.6)
6	1000 (39.4)	720 (28.3)	330	190 (7.5)	7 (15.4)
7	1100 (43.3)	850 (34.7)	370	220 (8.7)	6 (13.2)
8	1200 (47.2)	900 (35.4)	400	235 (9.3)	4 (8.8)
9	1300 (51.2)	970 (38.2)	430	255 (10)	3 (6.6)
10	1400 (55.1)	1050 (41.3)	480	280 (11)	1 (2.2)
0	Default to speed switches				

Speeds in the table are for planning purposes only and are typical minimum values. Speed values only consider a single joint moving. Speed at the end of the arm is greater when multiple joints are used on a single move.

7545-800S PROGRAM SPEED VALUES FOR PAYLOAD COMMAND

7545 with APO R00107 Program Speed Values ter PAYLOAD Command					
Program speed values	Speed of 01 at the tool tip mm/sec (in/sec)	Speed of 02 at the tool tip mm/sec (in/sec)	Roll axis speed deg/sec	Z-axis speed mm/sec (in/see)	Maximum mind for speed kg (11)
1	294 (11.6)	294 (11.6)	103	61 (2.4)	5 (11.0)
2	490 (19.3)	490 (19.3)	171	102 (4.0)	5 (11.0)
3	687 (27.0)	687 (27.0)	240	140 (5.5)	5 (11.0)
4	768 (30.2)	768 (30.2)	278	166 (6.5)	5 (11.0)
5	887 (34.9)	887 (34.9)	310	185 (7.3)	4 (8.8)
6	941 (37.0)	941 (37.0)	329	197 (7.8)	3.5 (7.7)
7	1073 (42.2)	1073 (42.2)	372	210 (8.3)	3 (6.6)
8	1189 (46.8)	1189 (46.8)	416	248 (9.8)	2 (4.4)
9	1285 (50.6)	1285 (50.6)	449	268 (10.6)	1.5 (3.3)
10	1356 (53.4)	1356 (53.4)	480	280 (11.0)	1 (2.2)
0	Default to speed switches				

Speeds in the table are for planning purposes only and are typical minimum values. Speed values only consider a single joint moving. Speed at the end of the arm is greater when multiple joints are used on a single move.

7547 PROGRAM SPEED VALUES FOR PAYLOAD COMMAND

7647 Program Speed Values for PAYLOAD Csmasad					
Prig == spud rains	Spade 81 at the tool tip mm/au MA	Spud of 82 st tho tool tip mm/see Mu)	Roll axis spud lite/su	1-axis spud maisse pi/see)	Maximum Mind for spud kg fib)
1	310 (12.2)	185 (7.3)	105	60 (2.4)	20 (44)
2	520 (20.5)	310 (12.2)	175	100 (3.9)	20 (44)
3	730 (28.7)	440 (17.3)	245	140 (5.5)	20 (44)
4	850 (33.5)	505 (19.9)	285	165 (6.5)	14 (30.8)
5	950 (37.4)	565 (22.2)	320	180 (7.0)	12 (26.4)
6	1000 (39.4)	600 (23.6)	340	195 (7.7)	11 (24.2)
7	1140 (44.9)	680 (26.8)	385	220 (8.7)	10 (22)
8	1230 (48.4)	735 (28.9)	410	235 (9.3)	6 (13.2)
9	1330 (53.4)	795 (31.2)	445	255 (10)	4 (8.8)
10	1450 (57)	865 (34)	485	280 (11)	3 (6.6)
0	Default to speed switches				

Speeds in the table are for planning purposes only and are typical minimum values. Speed values only consider a single joint moving. Speed at the end of the arm is greater when multiple joints are used on a single move.

APPENDIX E. SPEED/WEIGHT VALUES BASED ON Z POSITION

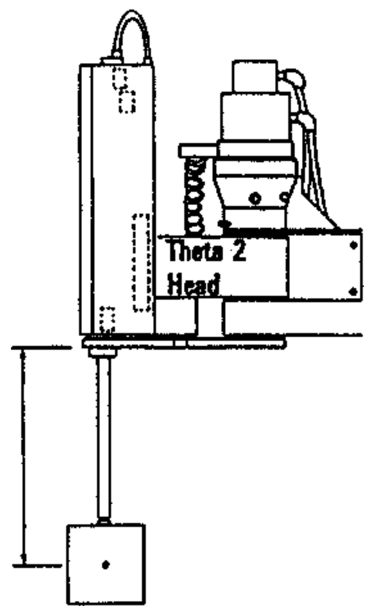
The tables in this section provide the the maximum program speed setting based on the height of the Z axis and the payload.

Speeds in the table are guidelines for planning purposes only, not specifications of appropriate speeds for all systems. Actual speeds vary, depending on the stability and weight distribution of your specific attachment.

To use these tables, find the weight of your payload on the vertical column of the chart. Along the horizontal find the Z-axis length. The length is the extension of the Z-axis plus the distance to the center of mass of the payload. The intersection of these two numbers is the maximum value you can safely use in the PAYLOAD command.

nr indicates that it is not recommended to run the manipulator with a Z-axis of that length and that payload.

The Z-arm height is illustrated below.



7545 SPEED/WEIGHT RELATIONSHIP BASED ON Z POSITION

Payload kg (lb)	Z-axis length to the cantor of mass of the payload (la mm)																	
	0 .	10	20	30	40	50	60	70	10	90	100	110	120	130	140	151	100	170
1 (2.2)	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2 (4.4)	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
3 (5.6)	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
4 (1.8)	8	•	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	
5 (11.0)	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
5 (13.2)	7	7	7	7	7	7	7	6	6	6	6	6	8	6	6	6	6	6
7 115.41	6	6	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5
1 (17.5)	5	5	5	5	5	5	5	5	5	5	5	5	4	3	3	3	3	3
9 (19.8)	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3
10 122.01	4	4	4	4	4	4	4	4	4	4	4	3	2	2	2	2	1	1

Payload kg (lb)	Z-axls length to the center of mass of the payload (11 mm)																	
	180	190	200	210	220	230	240	250	210	270	210	200	300	310	320	330	340	350
1 (2.2)	10	10	10	10	10	10	10	10	10	9	8	7	6	5	4	3	2	1
2 (4.4)	9	9	9	9	9	9	9	9	9	8	7	6	5	4	3	2	1	nr
3 (5.1)	9	9	9	9	9	9	9	9	9	8	7	6	5	4	3	2	1	nr
4 (1.0)	8	8	8	8	8	8	8	8	8	8	6	5	4	3	2	1	nr	nr
5(11.0)	6	6	6	6	6	6	6	4	4	4	3	2	1	IW	11(	re	nr	nr
1(13.2)	6	5	5	3	3	3	3	3	3	3	3	1	1	nr	nr	nr	nr	nr
7(15.4)	5	3	3	3	3	3	3	3	2	2	re	re	nr	nr	nr	re	nr	nr
0(17.5)	3	3	3	2	2	1	1	1	1	1	nr	nr	nr	nr	re	re	re	nr
11(11.01	2	2	1	1	1	1	1	1	1	re	nr	re	nr	IV	VW	nr	WI'	it
10(22.0)	1	1	1	1	1	nr	re	nr	re	nr	nr	nr	nr	nr	IV	it	IV	nr

Note: "nr" means not recommended.

Notes:

- nr means not recommended. It is not recommended to run the manipulator with a Z-axis of that length and that payload.

- Speeds in the table are guidelines for planning purposes only, not specifications of appropriate speeds for all systems. Actual speeds vary, depending on the stability and weight distribution of your specific attachments.

7545-800S SPEED/WEIGHT RELATIONSHIP BASED ON Z POSITION

Payload kg Oh)	Z-axis length to the center of mass of the payload II. mm)																	
	0	10	20	30	40	50	60	70	80	00	100	110	120	130	140	150	160	170
1 (2.2)	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2 14.4)	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
3 (6.6)	7	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
4 (8.8)	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
5 (11.0)	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Payload kg (lb)	Z-axis length to the center of mass of the payload ill mm)																	
	180	190	200	210	220	230	240	250	260	270	280	290	300	310	320	330	340	350
1 (2.2)	10	10	10	10	10	10	10	10	10	9	8	7	6	5	4	3	2	1
2 (4.4)	8	8	8	8	8	8	8	8	8	7	6	5	4	3	2	1	nr	nr
3 16.6)	6	6	6	6	6	6	6	6	6	5	4	3	2	1	nr	nr	nr	nr
4 (8.8)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	nr	nr	nr	nr
5 111.0)	1	1	1	1	1	1	1	1	1	1	1	nr	nr	nr	nr	nr	nr	nr

Notes:

- nr" means not recommended. It is not recommended to run the manipulator with a Z-axis of that length and that payload.
- Speeds in the table are guidelines for planning purposes only, not specifications of appropriate speeds for all systems. Actual speeds vary, depending on the stability and weight distribution of your specific attachments.

7547 SPEED/WEIGHT RELATIONSHIP BASED ON Z POSITION

hylud he (lb)	1-uls length legs enter el mess of the pulsed (Is mml													
	0	10	20	30	40	50	60	70	80	90	100	110	120	130
1 (2.2)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2 (4.4)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
3 (8.5)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
4 (11.8)	9	9	9	9	9	9	9	9	9	9	9	9	9	9
5 (11.0)	8	8	8	8	8	8	8	8	8	8	8	8	8	8
8(13.2)	8	8	8	8	8	8	8	8	8	8	8	8	8	8
8(17.8)	7	7	7	7	7	7	7	7	7	7	7	7	7	7
10 (22.0)	7	7	7	7	7	7	7	7	7	7	7	7	7	7
13 (28.5)	4	4	4	4	4	4	4	4	4	4	4	4	4	3
18 (35.2)	3	3	3	3	3	3	3	3	3	3	3	2	2	2
20 (44.0)	3	3	3	3	3	3	3	2	2	2	2	2	2	2

hylud 14 PM	Z-exls Meth to the cuter el mess if the pulsed (le mm)													
	140	150	160	170	180	190	200	210	220	230	240	250	280	270
1 (22)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2 (4.4)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
3 (5.8)	10	10	10	10	10	10	10	10	10	10	10	10	10	10
4(8.8)	9	9	9	9	9	9	9	9	9	9	9	9	9	9
5 (11.0)	8	8	8	8	8	8	8	8	8	8	8	8	8	8
5 (13.2)	8	8	8	8	8	8	8	8	8	8	8	8	8	8
8(17.6)	7	7	7	7	7	7	7	7	7	7	7	7	7	7
10 (22.0)	7	7	7	7	7	7	7	7	7	7	7	7	6	6
13 (28.5)	3	2	2	2	2	2	2	2	2	2	2	2	2	2
18 (35.2)	2	2	2	2	1	1	1	nr	nr	nr	nr	nr	nr	nr
10 (44.0)	2	1	1	nr	nr	nr	nr	nr	IN	nr	nr	nr	nr	nr

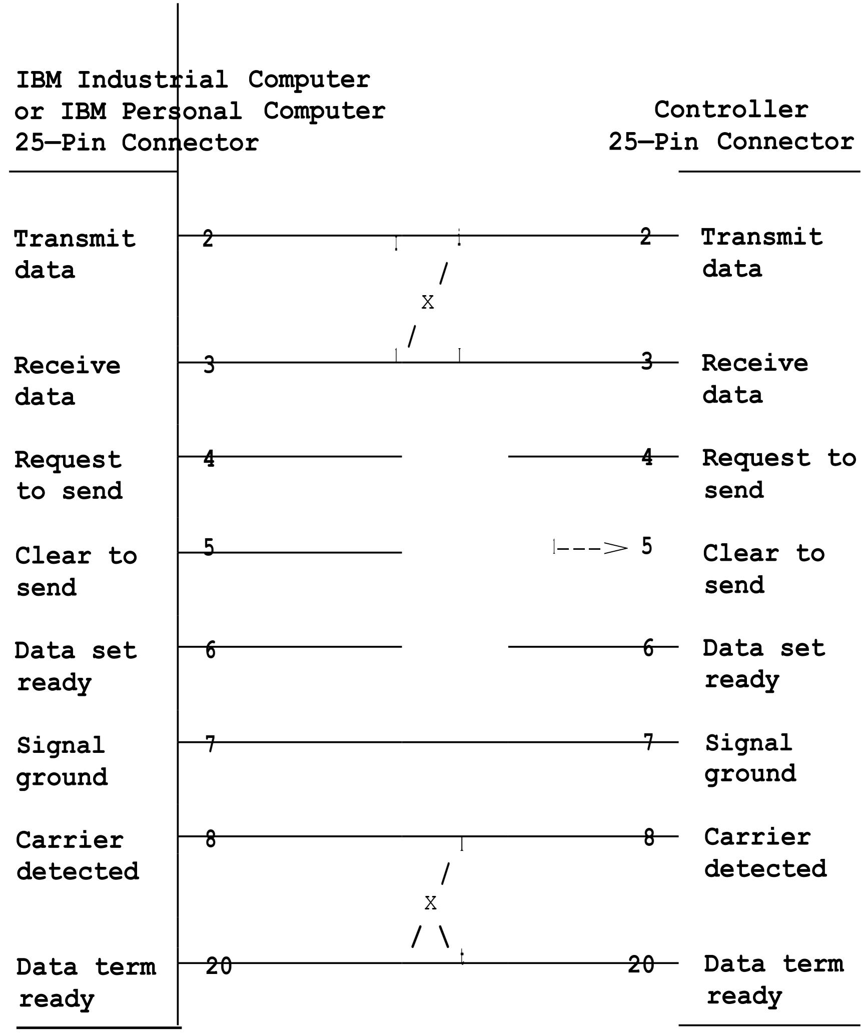
Payload ks ON	Z-axis booth to the cuter st taus of Oa payload po lug													
	280	290	300	310	320	330	340	350	380	370	380	390	400	410
1 (2.2)	10	10	10	10	10	10	9	8	7	6	5	4	3	nr
2 (4.4)	10	10	10	10	10	9	8	7	6	5	4	3	2	
3 (8.8)	10	10	10	10	9	8	7	6	5	4	3	2	1	nr
4 (8.8)	9	9	9	9	8	7	6	5	4	3	2	1	nr	nr
5 (11.0)	8	8	8	8	7	6	5	4	3	2	1	nr	nr	nr
8 (13.2)	8	8	8	7	6	5	4	3	2	1	nr	nr	nr	nr
1 (17.8)	7	7	7	6	5	4	3	2	1	nr	nr	fW	nr	1W
18 (22.0)	5	4	3	2	1	nr	nr	nr	nr	nr	nr	1W	11f	nr
13 (28.8)	1	1	1	nr	nr	nr	nr	IT	nr	rw	nr	nr	nr	nr
18 (35.2)	nr	nr	nr	nr	nr	nr	nr	nr	nr	nr	nr	nr	nr	
20 (44.0)	nr	nr	nr	nr	nr	nr	nr	nr	nr	nr	rtr	nr	iv	nr

Notes:

- nr" means not recommended. It is not recommended to run the manipulator with a Z-axis of that length and that payload.
- Speeds in the table are guidelines for planning purposes only, not specifications of appropriate speeds for all systems. Actual speeds vary, depending on the stability and weight distribution of your specific attachments.

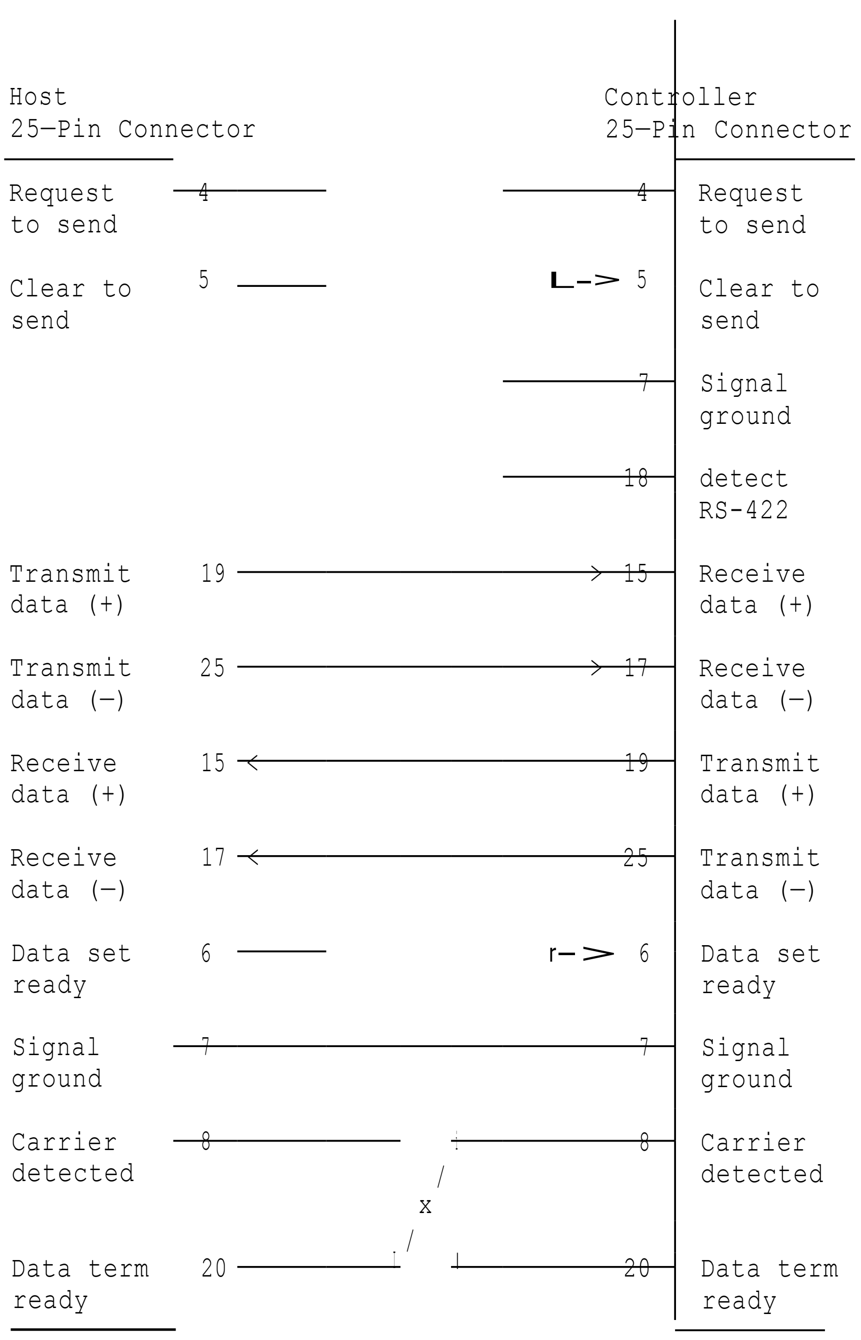
APPENDIX F. COMMUNICATIONS CABLE WIRING DIAGRAMS

LOCAL RS-232-C CABLE WIRING



Local RS-232—C Cable Wiring

LOCAL RS-422 CABLE WIRING



Local RS-422 Cable Wiring

APPENDIX G. CONFIGURATION PARAMETERS FOR AMLECOMM

This appendix discusses the configuration parameters for AMLECOMM that are contained in the file AMLECOMO.BAS. The user must modify this file according to the needs of his application. The user must never delete, reorder, or renumber any of these lines. The user should only change the values contained in this file.

```
1200 '
1210 'DEFAULT INSTALLATION CONFIGURATION:
1220 '
1230 POSTING.A% = N.A%           'Default to no state posting
1240 COMM.A       = 1            'Base file number for AMLECOMM
1250 OPENFCOMM.A = 3             'File number for file to download
1260 VARMAX.A     = 400          'Maximum number of variables
1265 VARMAXRECS.A = 101          'Set equal to INT(VARMXA.A/4)+1
1270 ROBOT.A$     = "7545"       'Default to a 7545
1280 ADAPT.A$     = "COM1:"      'Default to COM1
1330 ARRAYMODE.A% = 0            'Use 0 based arrays for R,G,P,C8OVARS
1340
2000 DIM RRVARS.A$(101)           Set size at least equal to VARMAXREC
2001                              Must be >=8
2010 DIM RVARS.A(400)             Set size equal to VARMAX.A
2020 DIM GVARS.A(400),PVARS.A(400)  'Set size equal to VARMAX.A
2030 DIM C8OVARS.A(400)           Set size equal to VARMAX.A
3265 ON ERROR GOTO 0              Replace 0 with application error trap
3405 ON ERROR GOTO 0              Replace 0 with application error trap
```

- POSTING.A% This variable allows the user to begin an execute command and then poll AMLECOMM to check for completion status. By using this posting feature, an execute command that may take many seconds allows other processing to be done while waiting for completion. A good example is the return home execute command. The elapsed time for a return home command can approach 90 seconds. It would be nice if during the 90 seconds the user's application program could also do other work as well. IF POSTING.A% is set to Y.A% then the execute return home command would be started and the status of the return home would be returned to the user via the output variable STATUS.A%. Upon subsequent polls to AMLECOMM, the user could determine when the return home actually completed. Between polls the user's application program could perform other routines. An alternative to polling is to use the ON COM statement, and is discussed in the body. POSTING.A% only affects X records.

- COMM.A This variable needs to be set to a number that may be used as a DOS file number for the communications file. Under DOS, communications lines are treated as files, thus a file number is assigned. If an application program uses file numbers 1 and 2, then COMM.A could be set to 3. Remember that over 3 files requires the

explicit specification of the number of files, using the /f: BASICA parameter.

- OPENFCOMM.A is similar to COMM.A, except this is a file number used for a file to be downloaded. When a user requests a download operation from AMLECOMM, AMLECOMM will use this file number to open the requested file.

- VARMAX.A This variable allocates storage for the maximum number of variables that exist in an application running in the robot controller. Depending on the number of variables in the AML/E program VARMAX.A allows storage to be saved by eliminating a fixed maximum value. Remember that even if there are no variables in the AML/E program there are still variables that are returned if a read variables command is executed. For example, AML/E version 4 returns 34 variables even if no counters are defined in the AML/E program. Any declared counters come after this base set.

- VARMAXRECS.A is used internally, and should be set to INT(VARMAX.A/4)+1. For example, if VARMAX.A were set to 39, then VARMAXRECS.A would be set to 10. If VARMAX.A were set to 40, then VARMAXRECS.A would be set to 11. VARMAXRECS.A must be given a value >=8, otherwise the configuration will fail.

- ROBOT.A$ This variable defines the type of robot that is being communicated to via AMLECOMM. Set ROBOT.A$ to "7545", "7545-800S", or "7547".

- ADAPT.A$ This variable is set to either "COM:" or "COM2:", depending on which asynchronous communication card through which the user intends to communicate with the robot controller.

- ARRAYMODE.A% This variable indicates whether the arrays used for reading variables, poking variables, and data drive are 0 or 1 based. If 0 is chosen, then the first variable in controller memory must be accessed as variable number 0, the second as variable number 1, etc. If arraymode 1 is chosen, then the first variable in controller memory must be accessed by variable number 1, the second by 2, etc. Note that the AML/E compiler produces its cross reference listing using a 0 based mode, thus arraymode should be set to 0 if the user intends to refer to variables as they appear in the cross reference listing produced by the XREF program. If arraymode 1 is chosen, then the user must add 1 to the number of a variable listed in the cross reference. For example a variable assigned number 40 by the cross reference program is accessed as 40 for ARRAYMODE.A%=0 and 41 for ARRAYMODE.A%=1.

  RVARS.A and C8OVARS.A are affected by ARRAYMODE.A%. The first variable read by the read variables request (VARS.A$) will be placed in RVARS.A(ARRAYMODE.A%), the second in RVARS.A(ARRAYMODE.A%+1), etc. Likewise variables to be sent to the controller using the poke variables request (POKE.A$) should be loaded into C8OVARS.A(ARRAYMODE.A%), C8OVARS.A(ARRAYMODE.A%+1), etc. Thus not only does ARRAYMODE.A% affect the numbering of the variables, but also whether the first value is put in the zero-position or the one-position of the arrays.

GVARS.A and PVARS.A are also affected by ARRAYMODE.A%. For example, should the user want the values 1 and 2 sent for variables 34 and 35 according to the cross reference listing and ARRAYMODE.A% 0, then GVARS.A(34) must be set to 1 and GVARS.A(35) must be set to 2. If ARRAYMODE.A%=1 then variables 34 and 35 according to the cross reference listing are really the 35th and 36th variables. Thus the user would set GVARS.A(35) to 1 and GVARS.A(36) to 2. If ARRAYMODE.A=1, then GVARS.A(0) and PVARS.A(0) are ignored entirely.

- RRVARS.A$(VARMAXRECS.A) This array is used internally, and must be dimensioned with a size equal to VARMAXRECS.A. VARMAXRECS.A (and thus the size of this array) must be at least 8 in size. Due to limitations in BASICA and the Basic Compiler this cannot be DIMensioned as a function of VARMAXRECS, and thus the user make sure the size of this array and VARMAXRECS.A agree. Unexpected results will occur if this is not the case. The DIM statement for this array should not be removed from AMLECOMO.BAS, even if the user does not need the R80 facility. The config utility will fail if this is removed. The config utility will strip the unneeded arrays from the configured AMLECOMM program.

- RVARS.A(VARMAX.A) This array is used to return variable values from a R80 request (read variables). As just described, the first value from a read request is placed either in RVARS.A(0) or RVARS.A(1) depending on the value of ARRAYMODE.A%. The size of RVARS.A must be set equal to VARMAX.A. Unexpected results will occur if this is not the case. The DIM statement for this array should not be removed from AMLECOMO.BAS, even if the user does not need the R80 facility. The config utility will fail if this is removed. The config utility will strip the unneeded arrays from the configured AMLECOMM program.

- GVARS.A(VARMAX.A),PVARS.A(VARMAX.A) These arrays are used for GET and PUT operations in data drive. They should be dimensioned with size equal to VARMAX.A. Unexpected results will occur if this is not the case. As with RRVARS.A$ and RVARS.A, the user should not delete this DIM statement, even though he may not need data drive.

- C8OVARS.A(VARMAX.A) This array is used to poke variables into controller memory using the C80 communication protocol. The size of this array must be set to VARMAX.A. Unexpected results will occur if this is not the case. As with the other arrays DIMensioned in AMLECOMO.BAS, the DIM statement for C8OVARS.A should not be removed, even though the C80 feature may not be needed.

- ON ERROR GOTO 0 AMLECOMM must install its own error handler when active to trap communication errors. This will cancel any error handler the user has established for his own program. AMLECOMM will restore a single error handler for the user. The user changes the two ON ERROR statements to point to his error handler. If no user error handler is needed, then ON ERROR GOTO 0 statements are used.

APPENDIX H. ADVANCE] COMMUNICATIONS

This appendix describes the communications support supplied by AML/Entry Version 4 at an extremely technical level. Users of AML/Entry Version 4 should use AMLECOMM to perform communications with the controller, because AMLECOMM handles all the details of the complex protocol. This appendix should only be read by users wishing to rewrite or modify AMLECOMM. Refer to chapter 8 for a high level discussion of the AML/Entry Version 4 communications, including a discussion of AMLECOMM.

The communications interface supplied with AML/Entry Version 4 is for the most part a low level interface at the machine language level. For this reason much of the information in this chapter is written for a reader with an understanding of machine language concepts.

COMMUNICATIONS HARDWARE INTERFACE

There are two types of interfaces used to communicate with the controller. The IBM Personal Computer communicates with the controller using the RS-232 interface. The asynchronous communications protocol is used with the following characteristics:

1. Full duplex transmission with a half-duplex (flip/flop) end-to-end user protocol.

2. baud rate = 4800

3. parity = even

4. data bits = 7

5. stop bits = 2

AML/Entry only allows the IBM Personal Computer communications port (COM1: or COM2:) to be configured, all other communications characteristics are not configurable. The controller characteristics are switch selectable. See the IBM Manufacturing Systems Specifications Guide, 8577126 for the controller switch settings.

The second type of interface available is RS-422. The RS-422 interface is provided to allow the controller to communicate to host computers. The controller automatically selects an RS-422 interface by sensing the cable type. This interface permits local operation between the controller and a host separated by up to 4000 feet. The cable effectively serves as a modem eliminator because:

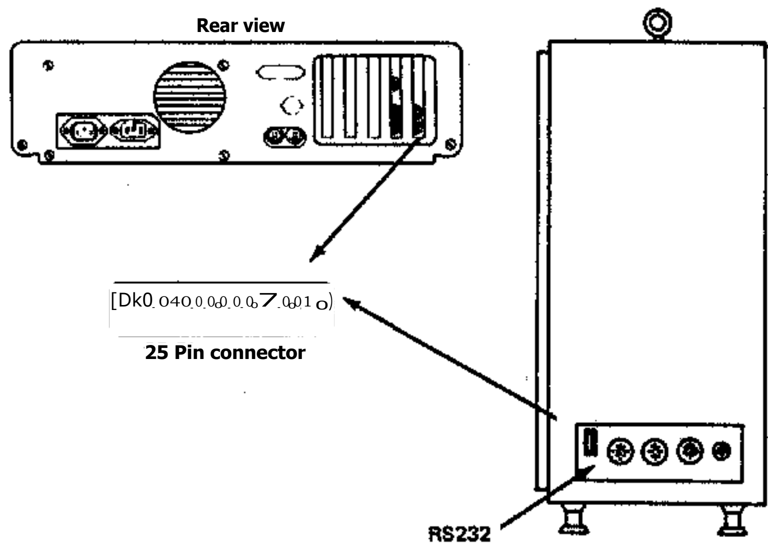
1. "Transmit data" is wired to "receive data" at the other end.

2. "Data terminal ready" is wired to "data set ready" at the other end.

3. "Request to send" is wired to "clear to send" at each end.

Controller Communications Connector

The 25 pin connector on the controller with the label **Cl RS232C** is the connector for the communications interface. Some pins in the connector have special meanings for RS-232 communications and some have special meaning for RS-422. The cable you connect activates the correct interface. For example, the RS-422 cable connects pin 18 in the controller connector to ground which tells the controller to use the RS-422 interface. Refer to Appendix F, "Communications Cable Wiring Diagrams" on page F-1 for the cable wiring diagrams.



COMMUNICATIONS PROTOCOL

The communications protocol used by the controller is a transaction based protocol with the host controlling all transactions.

Transactions

A transaction consists of:

- An identifier
- A record
- An optional collection of identifiers, records, requests, and responses.

Transactions are started by the host sending an identifier to the controller. This starts the transaction process. After the transaction has been started the identifiers are appended to the beginning of each record. The only time an identifier is sent by itself is to start a transaction.

If the controller does not receive a response to a transmission within 3 seconds it transmits again. After 3 attempts to transmit, the controller sets an error condition LED (TE) on the control panel.

The communications Startup precedes any transaction. A transaction begins with the start of transmission from the host. This can be either a request that data be sent to the host or a command from the host for the manipulator to do something. It may terminate normally, or abnormally. At the end of a transaction, the controller remains in the on-line state to accept further commands or requests from the host. It stays in the on-line state until off-line is pressed, or the host drops "data terminal ready".

Note: The on-line state is not to be confused with the On Line LED. The On Line LED signals a condition that the controller is able to enter the on-line state (line startup enabled).

Identifiers

When the host communicates with the controller, the host can send one of several types of records. Every record has an identifier associated with it. The types of identifiers you can use are:

- Read (R) - the host requests status or variable data values.
- Execute (X) - the host directs the controller to perform a remote function related to the operator control panel.
- New (N) Program - this identifier is used to start sending a compiled application program to the controller.
- Data (D) - this identifies subsequent records of multi-record transactions, such as an application program or reading variables where more than 16 bytes are expected.
- End (E) - this identifies the end of data transmissions.
- Control (C) - the host wants the controller to alter the status of a running program.
- Teach (T) - the host wants to move the manipulator, change a DO, or change LINEAR, PAYLOAD, or ZONE.
- Present Configuration (P) - the host wants to change the present configuration mode of the 7545-800S manipulator. This is not allowed for the 7545 and 7547 manipulators.

```
  id

One character
   |
   |
   |
   |
   |
   |
 Identifier:
   X - execute
   R - read
   N - program
   D - data
   E - end
   C - control
   T - teach
   P - present configuration (7545-800S only)
```

Records and Record format

The general format of a record is:

id bc **data** cs Cr Lf

A transaction is initiated by the transmission of an identifier. After an acknowledgement is received, the body of the record is transmitted. An acknowledgement is required for each record transmitted.

In the case of a transaction that consists of multiple records, all subsequent records are sent with an identifier and the record together. No acknowledgement should be expected between the ID and the record until the initiation of a new transaction.

Identifier

The **id** is a single character that identifies the type of transaction taking place.

Byte Count

The **bc** is two hexadecimal (hex) digits representing the length, in bytes, of the data. It takes values of 01 to 10 (1 to 16 decimal).

Data

The **data** is 1 to 16 bytes of hexadecimal data. Each byte of data is equivalent to two hex digits. All data is sent in hex digit pairs.

Check Sum

The **CS** is the checksum, which is not counted as part of the data. The checksum is the modulo 256 of all the hex digit pairs in the **data** bytes. For example, if the data that is being sent is '090A', to compute the checksum you would:

1. Add the hex pairs.

   09 + OA = 13

2. Take the remainder after dividing by 256.

   13 modulo 256 = 13

3. The checksum is 13.

Note: A quick method for finding the checksum of only a few records is to add up the hex pairs and take the last (low order) hex pair of the sum for the checksum. That is the modulo 256 of the sum.

Record Termination

To be able to process the record properly you have to know when the record ends. It is very easy with AML/Entry to look for the carriage return (Cr) character followed by the line feed (Lf) character. This is how the controller indicates the end of the record. It is also how you must indicate the end of the record to the controller.

Just remember, all records, except identifiers sent alone to start a transaction, and control codes (Ack, Xoff, etc.) end with a carriage return and a line feed.

Data Rules

All data sent to the controller must conform to the following conventions:

- All numeric data must be in hexadecimal.

  All data must be transmitted as the hexadecimal representation of the ASCII code for each byte.

  For example, if you wanted to send the decimal number "20" to the controller you would:

  1. Convert 20 to hexadecimal.

     20 = 14 hex

  2. Convert each digit of the hex number to its ASCII code.

     The ASCII code for 1 = "31"
     The ASCII code for 4 = "34"

  3. The actual data sent to the controller would be '3134'

  4. Many computers automatically convert any character sent to an asynchronous communications port to the correct ASCII code. Check the documentation for your host computer to find out if it automatically converts the data for you. If the host doesn't convert the data, you must convert it. For example, IBM Basic automatically does this conversion. Thus if the character string "14" is sent to either COM1: or COM2:, then it is automatically converted to the hex data '3134'.

     Note: To make explanations easier and less confusing, all examples in the remainder of this chapter assume that the correct ASCII conversion is done before sending the data to the controller. The examples show the correct data before conversion to ASCII format, or in the case of received data, after decoding the ASCII format.

- The ASCII Nul (hex 00) is ignored by the controller. The Nul character allows you to pad data if you are using a host that cannot send a single byte of data.

Data Representation

Numeric data returned by the controller is represented as either fixed point data or floating point data.

1. Fixed point is four bytes long in two's complement notation (the high order bit is the sign bit).

2. Floating point notation takes four bytes and is in the following format:

   a. The left most bit is the mantissa sign bit. The mantissa is positive if the sign bit is 0 and negative if the sign bit is 1.

   b. The next seven bits represent the exponent in two's complement notation (the high order bit is the sign bit for the exponent).

   c. The remaining 24 bits are the fractional mantissa. The mantissa must be 0 or between 0.5 and 1; therefore for values other than zero the first bit is always set to 1. Because this is a fractional mantissa, each bit in the mantissa is a negative power of 2.

X	XXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXX
Mantiss4 sign bit	Exponent -64 to 63	Fractional mantissa 0, or 0.5 to 1

Note: All numeric data that corresponds to application variables in controller memory are stored in floating point forma t. Thus, whenever reading variables from an application in memory or writing variables to an application in memory, the numeric data must be in floating point format.

Floating Point Examples

This section gives examples on how to convert numbers to and from the floating point format.

Example 1.

To decode the hex number 06A0 0000 you would:

```
Hex number      06A0 0000
    Binary      0000 0110 1010 0000 0000 0000 0000 0000

         Mantissa
         sign bit Exponent    Fractional mantissa
             0 0000110 101000000000000000000000

              mantissa sign = 0, mantissa is positive
              exponent = 4 + 2 = 6
```

number = mantissa x 2 raised to the exponent

$= (2^{-1} + 2^{-3}) \times 2^{6}$ add exponents to multiply

$= (2^{5} + 2^{3})$

= 32 + 8

= 40 decimal

Example 2.

To decode the hex number F8F0 0000 you would:

```
Hex number     F8F0 0000
    Binary     1111 1000 1111 0000 0000 0000 0000 0000

     Mantissa
     sign bit Exponent    Fractional mantissa
         1    1111000    11110000000000000000000

       Mantissa sign = 1      - The mantissa is negative
       Exponent sign = 1      - The exponent is negative-
                                and two's complement.

       Exponent = 1111000 Take the complement of 1111000
                  0000111 add 1 giving
                  0001000

                    8 exponent (absolute value)
       Exponent = -8
```

number = mantissa x 2 raised to the exponent

$= (2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}) \times 2^{-8}$ add exponents

$= (2^{-9}\ 2^{-10} + 2^{-11} + 2^{-12})$

= .00019531 + .00097656 + .00048828 +.00024414

= -.0036621 decimal

To convert decimal numbers to floating point format requires a different process.

Example 3.

To convert 650.0 to floating point format you would:

Find the exponent by doing step 1.

1. In the "Powers Of Two Table" on page H-14, find the N with a value that is just larger than the number to be converted. For 650.0, N would be 10, which has a value of 1024. Ten is the exponent (exp) to be used in the floating point format (bits 2 to 8).

The remainder of the procedure is an iterative process to determine the fractional mantissa.

2. Find N in the table with a value just smaller or equal to the difference. For the first time through, the difference is the starting number. In this case N = 9 has a value of 512 which is just smaller than 650.0. Find the number B, which when subtracted from exp to get 9, as shown in the following formula:

   exp-B = N   Remember exp = 10 and N = 9. Thus:
   10-B = 9
   B = 1

   - B = 1 shows that bit 1 of the mantissa is on.

3. Subtract the value chosen as being just smaller than the previous difference, from the previous difference, getting a new difference.

   650.0 - 512 = 138

   Take the new difference and use it as the difference in step 2. Continue this process until the difference is 0 or less than .001 x the original number.

From the table, the value that is just less than or equal to 138 is 128 (N = 7).

Using the formula   exp-B = N   exp = 10 and N = 7
10-B = 7
B = 3

- B = 3 shows that bit 3 of the mantissa is on.

Find the new difference and start again

138-128 = 10

From the table, the value that is just less than or equal to 10 is 8 (N = 3)

Using the formula exp-B = N exp = 10 and N = 3

10-B = 3

B = 7

• B = 7 shows that bit 7 of the mantissa is on.

Find the new difference and start again

10-8 = 2

From the table, the value that is less than or equal to 2 is 2 (N = 1)

Using the formula exp-B = N exp = 10 and N = 1

10-B = 1

B = 9

• B = 9 shows that bit 9 of the mantissa is on.

Find the new difference and start again

2-2 = 0

The difference is 0, the conversion is finished.

To get the hex digits

```
Mantissa sign bit = 0        The number is positive
        Exponent = 0001010 The exponent (exp) is 10
        Mantissa = 10100010100000000000000
                   | |   |
                   | I    I bit 9 on
                   | 1   bit 7 on
                   1 bit 3 on
                   bit 1 on
```

Put the fields together and convert to hex.

```
         0000 1 1010 1 1010 1 0010 1 1000 1 0000 1 0000 1 0000
Hex   =    0      A      A      2      8      0      0      0
      =    0AA28000
```

Example 4.

To convert -0.003663 to floating point format you would:

Ignore the sign and find the value in the "Powers Of Two Table" on page H-14 that is just larger than .003663 (exp.).

The value is .0039062 exp = -8

Find the fractional mantissa.

```
.0019531 .5_ .003663 N = -9
   exp-B = N    exp = -8 and N = -9
    -8-B = -9
       B = 1
```

• Bit 1 is on.

```
.003663 - .0019532 = .0017098
.00097656 5. .0017098 N = -10
   exp-B = N exp = -8 and N = -10
    -8-B = -10
       B = 2
```

• Bit 2 is on.

```
.0017098 - .00097656 = .00073324
.00048829 5. .00073324 N = -11
   exp-B = N exp = -8 and N = -11
    -8-B = -11
       B = 3
```

• Bit 3 is on.

```
.00073324 - .00048829 = .00024495
.00024414 ^ .00024495 N = -12
   exp-B = N exp = -8 and N = -12
    -8-B = -12
       B = 4
```

• Bit 4 is on.

```
.00024495 - .00024414 = .00000081
.00000081 < .001 x .003663
The mantissa calculation is finished.
```

To get the hex digits:

```
Mantissa sign bit = 1        The number is negative
         Exponent = 1111000 The exponent (exp) is -8
                             in 2's complement
         Mantissa = 111100000000000000000000
                    1111
                    1111
                    Illbit 4 on
                    llbit 3 on
                    'bit 2 on
                    bit 1 on
```

Put the fields together and convert to hex.

```
            1111 1 1000 1 1111 1 0000 1 0000 1 0000 1 0000 1 0000
  Hex    =    F      8      F      0      0      0      0      0
         =    F8F00000
```

Powers Of Two Table

N	Value (2^N)
16	65536
15	32768
14	16384
13	8192
12	4096
11	2048
10	1024
9	512
8	256
7	128
6	64
5	32
4	16
3	8
2	4
1	2
0	1
-1	0.5
-2	0.25
-3	0.125
-4	0.0625
-5	0.03125
-6	0.015625
-7	0.0078125
-8	0.00390625
-9	0.001953125
-10	0.0009765625
-11	0.00048828125
-12	0.000244140625
-13	0.0001220703125
-14	0.00006103515625
-15	0.000030517578125
-16	0.0000152587890625
-17	0.00000762939453125
-18	0.000003814697265625
-19	0.0000019073486328125
-20	0.00000095367431640625
-21	0.000000476837158203125
-22	0.0000002384185791015625
-23	0.00000011920928955078125
-24	0.000000059604644775390625

Data Line Control

The controller uses special ASCII characters to control transactions. The line control characters allow the host and the controller to acknowledge a transmission, etc. The line control characters used by AML/Entry Version 4 are:

Ack Acknowledgement for a good transmission

Cr Carriage return

Eot Reject data

Lf Line feed

Nak Error in transmission

Nul Fill Character

Xoff Not ready

Xon Ready

The ASCII characters for these line signals are shown in the table that follows.

Line Control Characters

Character	ASCII *
Ack	06
Cr	OD
Eot	04
Lf	OA
Nak	15
Nul	00
Xoff (DC3)	13
Xon (DC1)	11

* All values are in hexadecimal

For example, to send an Ack using IBM Basic, one would use

```
PRINT #1,CHR$(6);
```

(Assuming the communications port had been opened with file number 1).

Ack

The controller or host sends the Ack line control character under the following conditions:

1. Received data has the correct checksum and parity.
2. Controller is in a state that allows it to accept another request. The controller can delay sending the Ack up to 2 seconds.
3. The request is valid.

Note: All the previous conditions must be satisfied in order to send an Ack.

Xoff

The controller sends the Xoff line control character under the following conditions:

1. Received data has the correct checksum and parity.
2. The request is valid.
3. Due to other conditions (motion required or in progress, processing required) the request cannot be fulfilled immediately.

Note: All the previous conditions must be satisfied in order to send an Xoff.

If an.Xoff is sent, an additional response is sent within 30 seconds (Xon, or another Xoff or, Eot). For example, if the manipulator has just been powered up and the return home command is issued by the host, the controller responds with an Xoff signal approximately every 25 seconds until Home is reached. The controller then responds with an Xon signal.

Xon

The controller sends the Xon line control character under the following conditions:

1. The last response transmitted was an Xoff.

2. If the Xoff was in response to a record identifier, a receipt by the host of an Xon signals the host to transmit the body of the record.

3. If the Xoff was in response to a command record, receipt of Xon by the host means that the controller performed the request and can accept another record or transaction.

Note: In case numbers 2 and 3 above, the Xon can be considered by the host to be a positive response to the request.

Eot

The controller sends the Eot line control character under the following conditions:

1. The received data is a valid request. However, due to a condition in the controller, the process cannot be honored (such as attempting to select an application if the manipulator is currently running one, or attempting to return home with manipulator power off, or if an error condition, such as TE, DE, OT, etc, is present at the controller).

2. The received data is not a valid request (incorrect format, op code, and so on).

Nak

The controller or host sends the Nak line control character under one of the following conditions:

1. Incorrect checksum.

2. Bad parity.

Note: Up to 3 retries are permitted after which the controller sets an error condition LED (TE) on the control panel.

Nul

The Nul character is used as a fill character to allow host computers that can not send a single byte to pad data until the data is the size the host can handle. Nul characters are completely ignored by the controller.

Communication Startup Sequence

1. The controller On-Line LED is lit.
2. The host raises "data terminal ready."
3. The host waits for the controller to raise "data set ready"; this should happen immediately.
4. Startup is complete.

- If startup is complete, the operator control panel only responds to:

  Manipulator Power On

  Reset Error

  Off-line (This drops the controller DTR at the end of a transaction or immediately if no sequence is in process)

  — Emergency Power Off

- The controller side of the cable should have RTS wrapped back to CTS.

Record Descriptions

As we have described earlier in this chapter all data is communicated in the form of records. There are eight basic types of records used by AML/Entry. They are:

1. Read (R) Records - the host wants status or variable data values.

2. Execute (X) Records - the host wants the controller to perform a remote function related to the operator control panel.

3. New (N) Program Record - this record is used to start sending a compiled application program to the controller.

4. Data (D) Records - this identifies subsequent records as multi-record transactions, such as an application program or reading variables where more than 16 bytes are expected.

5. End (E) Records - this record indicates the end of a data transfer transaction.

6. Control (C) Records - the host wants the controller to alter the status of a running application program.

7. Teach (T) Records - the host wants to move the manipulator, change a DO, or change LINEAR, PAYLOAD, or ZONE.

8. Present Configuration (P) Records - the host wants to change the present configuration mode of the 7545-800S manipulator. This is not allowed for the 7545 and 7547 manipulators.

The easiest way to learn each record format is to use the COMAID program. By using option P, the communications records can be printed on the screen. The detailed layout of each record type follows:

R - (Read) Record

Data returned from the controller in response to a **R** (read) record is in **D** (data) record format.

> **Note:** The controller does not send an Ack in response to a read record. The requested data is immediately returned in a **D** - record (for all read requests except R 80 - read specific program variables). For R 80, an Ack is sent by the controller in response to the read record. The host then sends a special **D** record which contains the starting variable that is to be read and the number of variables to read. The controller responds to this with a **D** record.

operand 1	CrLf
Two characters	Two characters

Read operand:

01 - Read machine status
02 - Read reject status
(why the controller sent Eot)
03 - Read robot type and micro code level
04 - Read robot parameter table
08 - Read current instruction address
10 - Read DI/DO
20 - Read all variable values
40 - Read current position in pulses
80 - Read specific variable values

Note: The read record may temporarily interrupt a running application when the controller is responding to the host request.

R 01 - Read Machine Statue

Response to op-code 01 (read machine status) returns additional hexadecimal codes in the data field of the D-record, as identified below.

First byte of data field in D-record

80 - Servo error
40 - Power failure
20 - Overrun
10 - Over Time
08 - Transmission error
06 - AML/Entry error (the error code is in the second byte)
04 - Data error (the error code is in the second byte)

If the first byte contains 06 (AML/Entry error), the second byte contains additional information about the error as listed below.

OA - Part number too small for pallet
OB - Part number too large for pallet
14 - Invalid index for a group
lE - Communications not established (unable to GET/PUT)
32 - Invalid index for FROMPT function
33 - Square Root of a negative number
34 - Invalid arguments for ATAN2 function

If the first byte contains 04 (data error), the second byte contains additional information about the error as listed below.

00 - No data error present
01 - Bus error
02 - Memory test error
11 - Arithmetic error
12 - Programming error
13 - Invalid op Code
14 - Invalid data
15 - Invalid port number
17 - Stack error
18 - Address error
40 - Point out of work space

The functions provided by AML/Entry Version 4 limit the ability of the compiler to detect run-time errors. Some errors occur only when the application program is executing. If a run-time error is found, the DE (data error) light comes on and execution of the application is suspended.

The ability to determine the type of error is provided as part of the read machine status transaction. In Version 4, the error code (06) indicates that an AML/Entry error was encountered. If the first byte returned is (06), the second byte contains the particular AML/Entry run-time error code.

R 02 - Read Reject Status

Response to Op-code 02 (read reject status), or why the controller sent Eot, have additional codes as identified below.

10 - Record format error
15 - Invalid port number
20 - Undefined record
30 - UnacCeptable condition
Improper application startup sequence
31 - UnacOeptable condition
C-Record but no application
40 - Point out of workspace
50 - Insufficient Memory
51 - Invalid robot type
52 - Double select error
53 - Invalid application number
54 - C-Record format error
60 - Invalid identifier
sent. N record
70 - Xoff Time-Out (30 seconds)
74 - Invalid data
75 - Too much data
76 - Too little data
80 - Manipulator power off
AO - Xoff. Time-Out State
Al - Can not accept command (Xon State)
A2 - Communication operation not allowed (Xto or Wxo mode)
AA - Controller has gone off-line

R 03 - Read Micro-code Level and Machine Type

Response to op-code 03 (Read micro-code level and machine type) is a D record of the following format:

D	06 1				00 I 00 I	check	CrLf 1
	length in bytes	two HEX digits Major Microcode Level	two HEX digits Minor Microcode Level	four HEX digits Machine Type	four zeros always returned	2 HEX digits Check Sum	2 ASCII chars Carriage Return and Line Feed

R 04 - Read Robot Parameter Table

Response to op-code 04 (read robot parameter table) is three D records, each containing four pieces of configuration data. The format and contents of the three records is as follows

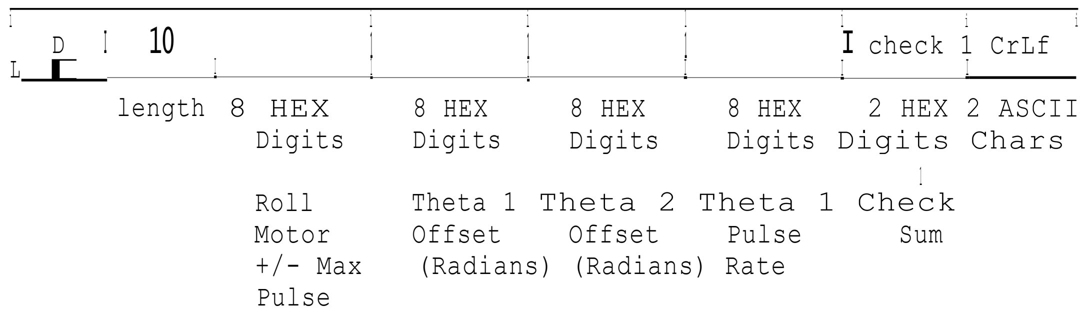
RECORD 1

D 10, 1 check 1 CrLf 1

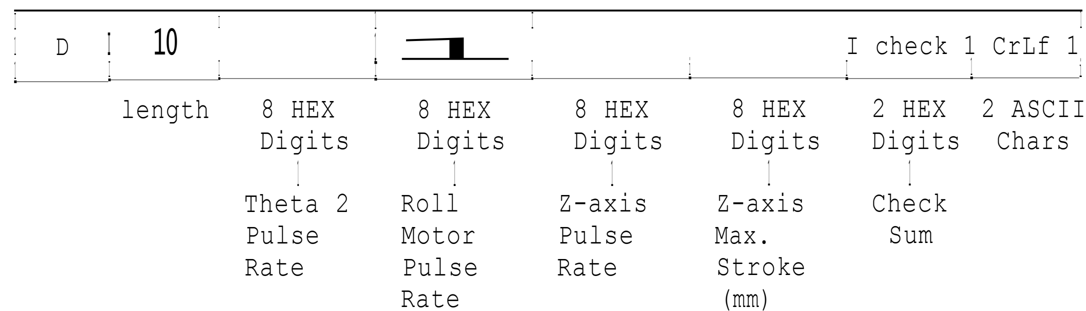
length 8 HEX Digits | 8 HEX Digits | 8 HEX Digits | 8 HEX Digits | 2 HEX Digits | 2 ASCII Chars

Theta 1 Arm Length (mm) | Theta 2 Arm Length (mm) | Theta 1 Max Pulse | Theta 2 Max Pulse | Check Sum

RECORD 2



RECORD 3



R 08 - Read Current Instruction Address

Response to op-code 08 (read current instruction address) is a single **D** which contains the current instruction address in 4 hex digits. By converting this to decimal and looking at a listing generated by the AML/Entry compiler, it is possible, to determine the source statement.

Note: All motion of the manipulator is performed by an internal subroutine. If the current instruction address is read during motion or immediately following motion, the controller will return the address of

the internal subroutine, not the address of the actual PMOVE, DPMOVE, XMOVE, GETPART, or ZMOVE that started the motion.

R 10 - Read DI/DO

Response to op-code 10 (read DI.DO) is three **D** records. The status of the DI/DO is in hex format with two bytes (4 hex digits) for every 16 points of DI/DO installed on the system.

1. The first record contains the information on the first 16 bits of DI and DO installed. The data is returned in the following format:

    Data bytes 1-2 16 input ports.
    Data bytes 3-4 16 output ports.

    For example, if a "Read DI/DO" request is sent to the controller, the following data is returned in the data field of the first record.

    8000FF00

    Breaking the data into fields:

```
8000 FF00
 |    |
 |    16 DO
 16 DI
```

First data byte in hex	8	0	0	0
First data byte in binary	1000	0000	0000	0000
	1111	1111	1111	1111
DI port	1234	5678	9111	1111
			012	3456

    DI port 1 is on.

2. The second **D** record contains the information on any additional DI installed on the system. The first byte of the data field contains the number of additional 16 point increments installed on the system. The remaining data bytes contain the state of the points with each bit representing one point.

3. The third **D** record is identical to the second record except it contains information on the additional DO points installed on the system.

Three records are always returned. If no additional DI/DO is installed, the second and third records contain zero for the number of additional increments installed, but they are still returned.

R 20 - Read All Program Variables

Response to op-code 20 (read all program variables) is many **D** records. Each **D** record contains 4 values (stored in floating point format) representing 4 values of the user's application program. To determine which values in the **D** records map to which user variables, you must know how the compiler arranged them in the controller. This information is supplied as an offset into the data for each variable. The offsets are provided by the XREF program. See "XREF Program" on page 2-34 and "XREF Program on page 4-89 for more on XREF.

R 40 - Read Current Position in Pulses

Response to op-code 40 (read current position in pulses), is a single **D** record that contains 4 values representing the 01, 02, Z, and Roll axes. Each value is a 6 hex digits long 2's complement number. See the IBM Manufacturing System Specifications Guide for the number of pulses per degree for the 01, 02, and Roll axes, and the number of pulses per millimeter for the Z axis.

R 80 - Read Specific Program Variables

Response to op-code 80 (read specific program variables) is many **D** records. Each **D** record contains 4 values (stored in floating point format) representing 4 values of the user's application program. To determine which values in the **D** records map to which user variables, you must know how the compiler arranged them in the controller. This information is supplied as an offset into the data for each variable. The offsets ate provided by the XREF program. See "XREF Program" on page 2-34 and "XREF Program" on page 4-89 for more on XREF.

This request requires that the host send the starting variable number and number of variables to read. After starting the request by sending a **R** identifier with an op-code of 80, the controller acknowledges the **R** record. After which the host sends **D** record containing the data outlined below.

vvvv	nnnn
Four characters	Four characters

The number of contiguous variables requested

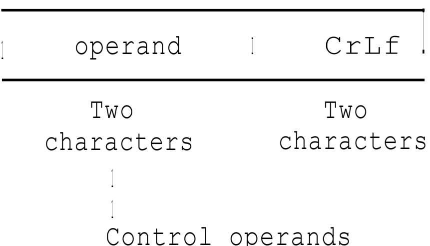
Starting variable number

If the request can not be honored, the controller responds with an Eot and sets the appropriate Eot code. For example, if the request extends beyond the end of the partition, the controller sets the Eot code to '76', "too little data" not enough available. When all of the requested

data has been transmitted the controller sends an **EG** record to indicate the end of the data.

C (Control) - Records

C - records allow you to control the execution of an application program from the host computer. Format and op-codes for the **C** record are outlined below.



Operands allowed in the C record are as listed below.

01 - Suspend program execution
02 - Restart program execution
04 - Execute Until Next Terminator
10 - Set debug address stop
20 - Reset controller
80 - Change controller variables

Op-codes 01, 02, 04, and 20 do not require further data from the host. As soon as the op-code is received by the controller, the request is performed (or if motion is being performed, the request is blocked until the motion completes).

Op-code 10 requires further data to be sent to the controller. One additional **D** record is that contains a 4 hex digit address indicating the address where the debug breakpoint is to be installed. This is gotten from the listing (.LST) file created by the AML/E compiler. Only one debug breakpoint may be installed, and the installation of a new breakpoint supersedes the previous breakpoint. Once the debug breakpoint is established, the host must send an Xon to the controller to enable data drive mode. If the controller is in the Xoff state when a debug breakpoint is reached, the controller follows the rules for complex state transitions (see "Complex State Transitions" on page 8-10). When the controller encounters the debug breakpoint and is in the Xon state, a Q record is sent from the controller to the host. This **D** record contains only one byte to data in the body of that identifies the record as a "debug encountered" message.

Op-code 20 performs the following actions:

- Clears all errors
- Resets all communication timers
- The current application selection is cancelled

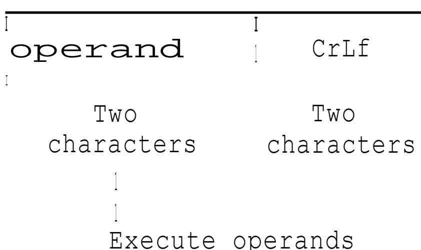
• Controller is placed in manual mode

Op-code 80 requires many **D** records to be sent to the controller. The first **D** record is of the same format required for the R 80 request. It contains the starting variable number and the number of variables to be sent. After the controller responds with an Ack, then the values are sent 4 at a time, in **D** record format until the proper number ov variables have been sent. The controller will respond to each of the **D** records with an Ack. After the last **D** record is sent, the host should send an "EG".

> **Note: It** is important to remember that the process is not synchronized with the application program. Care must be exercised that the data is not sent to the controller at a time that adversely effects the application program.

X (Execute) - Records

The format for the execute (X) appears as follows:



The operands allowed in the X record are as follows:

11 Return Home
12 Recall Memory
13 Reset Error
20 Auto
22 Start Cycle
23 Stop Cycle
24 Stop and Mem
25 Step
31 Select Application 1
32 Select Application 2
33 Select Application 3
34 Select Application 4
35 Select Application 5

No further data is required from the host. When the X is sent to the controller, it will respond with Xoff's until it can handle the request, upon which it will send an Xon. Upon receiving the Xon, the host then sends the op-code. If the request can be handled immediately, the controller responds with an Ack, otherwise the controller will respond with an Xoff every 25 seconds until the request has been fulfilled, and then an Xon will be sent.

N (Compiled Program) - Record

The **N** record tells the controller the name of an application program and the partition to load the program into. The format for a **N** record appears as:

```
N RECORD:

 09  1 1* 1    20 0202020202020    1 **        CrLf

 Two    One     'Length-1'          Two        Two
chars   char    characters         chars      chars
  |      |      of data              |          |
Length   |                        Checksum   Carriage return
  of    Partition                            and line feed
 data
```

* Equals the partition number for controller storage. The compiler always puts a 1 for the partition. However, you can specify a 1, 2, 3, 4, or 5 for the partition number.

** If a name is placed in the program name field, the checksum must be computed for the record using modulo 256.

The program name is not used, therefore, the field is prefilled with blanks. This does not mean that the name field can be deleted. The name field must always be filled with something.

The compiled program generated by the AML/Entry compiler includes the first record to be transmitted to the controller. The N-record generated by ithe compiler defines partition 1 as the partition for transmission to the controller. If you want any other partition, you must alter the record where any * is located. **No other part of the record should be altered unless a name is to be transmitted.**

The host starts the sequence by transmitting the N identifier and then waits for an acknowledgement (it could be an Xoff followed by an Xon sequence, or an Ack) from the controller. The host sends each record along with a CrLf and waits for an Ack signal from the controller. When the host has sent all the compiled program, it sends an E record to indicate the end of the program data.

E (End) - Record

The AML/Entry compiler generates an End Good (EG) record to indicate that the compiled program is good and should be loaded into memory. If the controller receives an EN record, it unloads the partition that was specified in the original 'N' record. The file can be altered to have the controller unload a partition.

E RECORD:

```
 ______________________
   status 1    CrLf
            ___
    One          Two
 character  characters

                 Carriage return + line feed
```

Operand:
G - data is good
N - unload the partition

D (Data) - Record

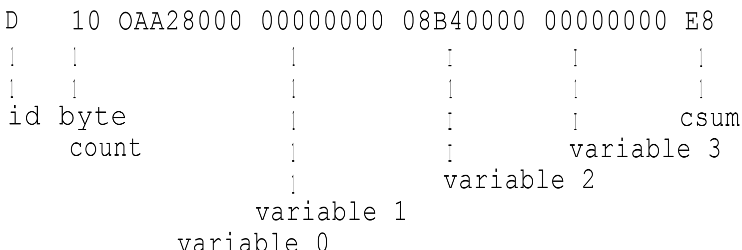
The data record is used to return information requested by a read record. Because the data record is always in response to a request for data and never to start a transaction, the identifier can be thought of as being part of the record. If there is more data than can be put in one record additional data records are sent following the first data record. The D-record format is:

```
 ________________________________________
 I D I be I data        check     CrLf
                 ________________________
   |    |
   |    |
   |    |
  One  Two   'Length'    Two       Two
  char chars  chars      chars     chars
  id  byte    requested  checksum Carriage return
       count  data                 line feed
```

Responses to the 20, read variable, return a string of hex bytes in D record format. Example:

```
D100AA280000000000008B4000000000000E8
```

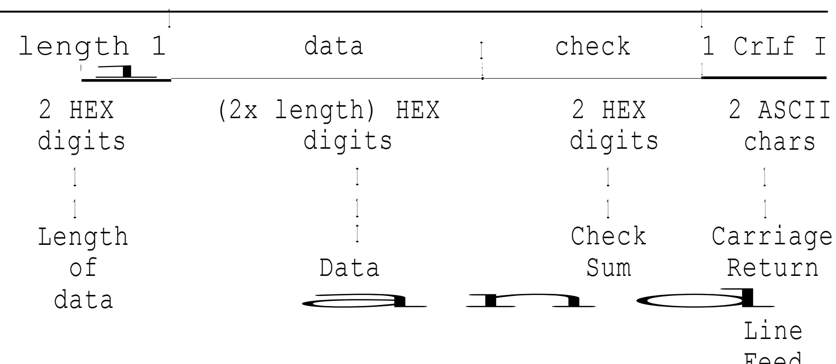
Breaking the number into data fields:



See "Reading Program Variables" in this chapter for details on how to interpret the data about the variables.

T (Teach) - Record

The generalized format of a T record is shown below. All types of teach transactions use a T record with this structure. The contents of the data field, however, will vary from transaction to transaction as specified below.



Warning: Sending a Teach record will cause the manipulator to move home if the manipulator has been taken off-line, moved, and placed back on-line. Thus Teach records should not be used in conjunction with the control panel.

A "T" is first sent to begin the Teach protocol. After the controller responds, then the Teach record is sent. Only teaching a point requires a second record to be sent. The second teach records have a "T" prepended to the data.

Motion Parameters

The three motion parameters (Linear, Velocity and Zone) are set using T records. The data field for this type of teach record is composed of three parts. The first is a single byte code (two HEX digits) that correspond to the operation that is to be performed. The valid op-codes corresponding to each of the three parameters are

X'68'- Modify Linear setting
X'69'- Modify Velocity setting
X'6A' - Modify Zone setting

The second part of the data field is the actual setting. This value should be in HEX format and occupy one byte (two HEX digits).

The third part of the data field is the terminator. This terminator is always X'3E' and immediately follows the op-code.

These parameters only affect motion that is invoked by Teaching a point. They cannot be used to change the parameters of an executing application.

Motion Control

The manipulator is instructed to move to a specific location using teach records. While the general format of the record is the same, as that for the motion parameters, the data portion of the record differs slightly.

The data field for this type of teach record is composed of four parts. The first is a single byte code (two HEX digits) that correspond to the operation to be performed. The valid op-codes are

X'70'- X value
X'71'- Y value
X'72'- Z value
X'73'- Roll Value

The second part of the data field in a teach record is a code which specifies the format of the data contained in the record. The codes and their corresponding meanings are

X'21'- Floating Point
X'22'- Fixed Point

The third part of the data field in this teach record is the actual value in the format specified by the previous byte of data.

The fourth part of the data field is the terminator. This terminator is always X'3D' and immediately follows the actual value.

Multiple occurrences of this four part format may occur in a single teach record, provided that the total number of bytes in the data field does not exceed 16. Normally a teach operation to direct the motion of the arm is done in two parts: the first record teaching the X and Y

positions, the second teaching the Z and Roll positions. The second (and any subsequent) Teach Records have a "T" prepended to the data. After the last separator (the X'30' flagging the end of the roll value), a terminator of X'3E' must be appended. After the X'3E', the checksum and CrLf appear. For example, to move the manipulator to the point (650,0,0,0), a "T" is first sent to begin the protocol. After which two records are sent:

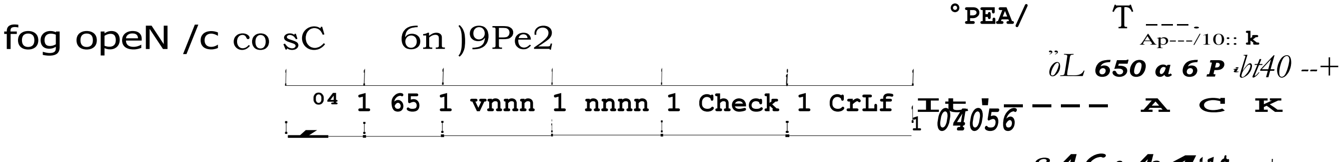
```
1 OE 17021 1 OAA28000 1 3D 17121    00000000 1 3D 1 C9 1 Crlf
```

```
T I OF 1 7221 1 00000000 1 3D 1 7321 1 00000000 I 3D3E I DF I Crlf 1
```

The controller will send an Ack to the first record. After the second record is received, the motion is performed.

Changing Digital Outputs

In order to change a digital output, after sending a "T" to begin the Teach protocol, a record is sent with the following format:



The 7 n's hold the binary representation of the port nunfti. If the port is to be turned on, then v is 1. If the port is to be turned off, then v is O. The v field and n field together combine to give 2 Hex digits.

P (Present Configuration) – Record

A P record is used to switch the present configuration of the 7545-800S to either left or right mode. This class of records is only valid for the 7545-800S manipulator. A "P" is first sent to the controller, which responds with Xoff's every 25 seconds until the request can be honored. Once it can be honored, an Xon is sent to the host. Upon receiving the Xon, the host sends one of the following records:

```
FOR LEFT MODE: 014F4F CrLf
FOR RIGHT MODE: 014E4E CrLf
```

The first byte, 01, is the length of the data. The second byte is the data that indicates whether the 7545-800S should be set to left mode or right mode. A X'4F' signifies left mode, a X'4E' signifies right mode. The third byte is the checksum. The last byte is CrLf.

Controller-Initiated Communications

Controller-initiated communications allow the controller to send as well as receive information from the host computer. This allows report information to be sent to the host from the application program. Examples of GET and PUT transactions are given in "Typical Communications Sequences" on page H-36. This information could include the items listed below.

- The number of parts produced.
- Error conditions.
- Completion of a cycle.

Controller-initiated communications use the AML/Entry commands GET and PUT. For a detailed description of GET and PUT, refer to Chapter 8, "Communications." The controller must be in the Xon State for the GET and PUT commands to execute. If the controller is in the Xoff State when the commands are encountered, the rules for complex state transitions apply. See Chapter 8, "Communications" on page 8-1 for a discussion of the Xoff state, Xon state, and complex state transitions.

PUT Transaction

When the PUT command is encountered, the controller sends a **D**_ record with the data in the format outlined below.

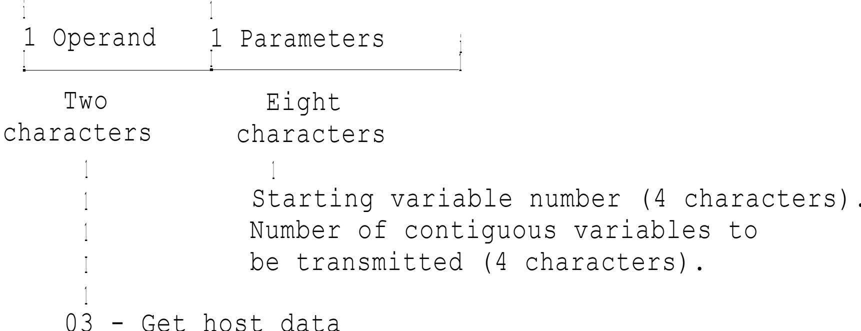
Operand	I Parameters
Two characters	Eight characters

```
     |          |
     |        Starting variable number (four characters).
     |        Number of contiguous variables to
     |        be transmitted (four characters).
     |
   02 - PUT data to host
```

The host should respond to this record with an Ack. After sending the Ack, the controller will proceed to send all the variables identified in the first **D** record in subsequent **D** records. The host should Ack each **D** from the controller if it is received with correct length, checksum, and parity. After all the data has been sent, the controller will send an "EG".

GET Transaction

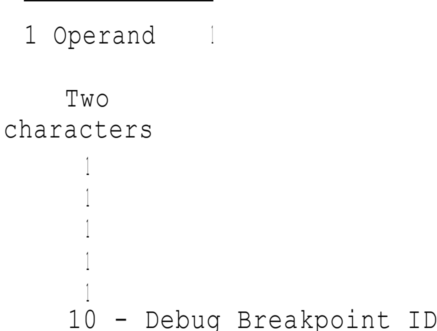
When the GET command is encountered, the controller sends a **D** record in the format outlined below.



Once the GET is received, the host should send the requested variables to the controller in **D** records. The controller will send an Ack for each **D** record. After all the data has been sent, the host should send an "EG" to the controller. If too little or too much data has been sent, the controller will respond with an Eot, otherwise it will respond with an Ack.

DEBUG Transaction

When a debug breakpoint is encountered, the controller sends a **D** record in the format outlined below.



The host should simply Ack this **D** record to complete the debug breakpoint protocol. The controller will now be blocked. To continue execution, first an Xoff should be sent to disable data drive. After this has been done, a C 02 record will restart execution.

Application Startup Sequence

This sequence is used to start an application from the host once the system power has been turned on and the manipulator power is up.

Host	Controller	Explanation
		The host system sends the controller an X identifier (hex 58) to indicate the start of an execute transaction. The X tells the controller that the next thing sent is an X record.
	Xoff	The controller always responds with an initial Xoff (hex 13) If the controller is able to accept the record, an Xon (hex 11) is sent following the Xoff.
	Xon	The controller is able to accept the X record.
"11"CrLf		The host sends the return home operand '11' (hex 31, hex 31) to the controller followed by the Cr (hex OD) and the Lf (hex OA) to indicate the end of the record.
	Xoff	The controller returns home sending a Xoff every 25 seconds until home is reached and the return home command is completed.
	Xon	The controller sends an Xon to indicate the completion of the return home command. This ends this transaction and the controller waits for another identifier from the host.
"X"		The host system sends the controller an X identifier (hex 58) to indicate the start of an execute transaction. The X tells the controller that the next thing sent is an X record.
	Xoff	The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff.
	Xon	The controller is able to accept the X record.
"31"CrLf		The host sends the select application 1 operand '31' (hex 33, hex 31) to the controller followed by the Cr (hex OD) and the Lf (hex OA) to indicate the end of the record.
	Ack	The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that it has completed the command.

Manipulator Stop Cycle Sequence

This sequence is used to stop the currently running application

Host	Controller	Explanation
"X"		The host system sends the controller an X identifier (hex 58) to indicate the start of an execute transaction. The X tells the controller that the next thing sent is an X record.
	Xoff	The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record, an Xon (hex 11) is sent following the Xoff.
	Xon	The controller is able to accept the X record.
"23"CrLf		The host sends the stop cycle operand '23' (hex 32, hex 33) to the controller followed by the Cr (hex OD) and the Lf (hex OA) to indicate the end of the record.
	Xoff	The host sends an Xoff indicating that it received the command.
	Xoff	The controller goes to the end of the program sending an Xoff every 25 seconds until the end of the program is reached and the stop cycle command is completed.
	Xon	The controller sends an Xon to indicate the completion of the stop cycle command. This ends this transaction and the controller waits for another identifier from the host.

Reason For Data Error

This sequence is used to query the controller for the reason for a data error.

Host	Controller	Explanation
"R"		The host system sends the controller an R identifier (hex 52) to start a read transaction. The R tells the controller that the next thing sent is an R record.
	Xoff	The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. If the controller is not ready additional Xoffs are sent about every 25 seconds until the controller is ready.
	Xon	The controller is able to accept the X record.
"01"CrLf		The host sends the read controller status operand '01' (hex 30, hex 31) to the controller followed by the Cr (hex OD) and the Lf (hex OA) which indicates the end of the record.
	"D02044044"CrLf	The controller responds with a record with a D to indicate a data record, the byte count (02), the data (0440), the check sum (44), and the CrLf to indicate the end of the record. The 04 in the data shows a data error, the 40 shows a point out of work space error.
Ack		The host sends an Ack (hex 06) to tell the controller that the record was received correctly.
	"EG"CrLf	The controller responds with an E record to show the end of the data. The record is as follows: an E to show this is an E record, a G shows the data is good, the CrLf shows the end of the record.
Ack		The host sends an Ack (hex 06) to tell the controller that the record was received correctly.

"X" The host system sends the controller an X identifier (hex 58) to indicate the start of an execute transaction. The X tells the controller that the next thing sent is an X record.

Xoff The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff.

Xon The controller is able to accept the X record.

"20"CrLf The host sends the auto operand '20' (hex 32, hex 30) to the controller followed by the Cr (hex OD) and the Lf (hex OA) which indicates the end of the record.

Ack The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed.

"X" The host system sends the controller an X identifier (hex 58) to indicate the start of an execute transaction. The X tells the controller **that** the next thing sent is an X record.

Xoff The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff.

Xon The controller is able to accept the X record.

"22"CrLf The host sends the start cycle operand '22' (hex 32, hex 32) to the controller followed by the Cr (hex OD) and the Lf (hex OA) to indicate the end of the record.

Ack The controller responds with an Ack (hex 06) to indicate that the record was received correctly, and that the command has completed.

Reason For Data Error

This sequence is used to query the controller for the reason for a data error.

Host	Controller	Explanation
"R"		The host system sends the controller an R identifier (hex 52) to start a read transaction. The R tells the controller that the next thing sent is an R record.
	Xoff	The controller always responds with an initial Xoff (hex 13). If the controller is able to accept the record an Xon (hex 11) is sent following the Xoff. If the controller is not ready additional Xoffs are sent about every 25 seconds until the controller is ready.
	Xon	The controller is able to accept the X record.
"01"CrLf		The host sends the read controller status operand '01' (hex 30, hex 31) to the controller followed by the Cr (hex OD) and the Lf (hex OA) which indicates the end of the record.
	"D02044044"CrLf	The controller responds with a record with a D to indicate a data record, the byte count (02), the data (0440), the check sum (44), and the CrLf to indicate the end of the record. The 04 in the data shows a data error, the 40 shows a point out of work space error.
Ack		The host sends an Ack (hex 06) to tell the controller that the record was received correctly.
<	"EG"CrLf	The controller responds with an E record to show the end of the data. The record is as follows: an E to show this is an E record, a G shows the data is good, the CrLf shows the end of the record.
Ack		The host sends an Ack (hex 06) to tell the controller that the record was received correctly.

Program Transmit Sequence

This sequence is used to send a compiled program to the controller.

Host	Controller	Explanation
"N"		The host system sends the controller an *N* identifier (hex 4E) to indicate the start of a program transmission transaction. The N tells the controller that the next thing sent is an N record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly.
"0911202020202020202011"C rLf		The host sends the first record of the compiled program with the data length (09), the partition number (11), program name (20), checksum (11), Cr (hex OD), and the Lf (hex OA) to indicate the end of the record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly.
"D02010102"CrLf		The host sends the first record of the compiled program with the identifier(D), the data length (02), data (0101), checksum (02), Cr (hex OD), and the Lf (hex OA) which indicates the end of the record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly.
Data record 2	>	As many records as needed to transmit the program are sent. Each record is acknowledged by the host with an Ack (hex 06).
"EG"CrLf		The host sends an E record with the identifier E (hex 45), the G operand (hex 47) to indicate the end of the data and that the program compiled correctly and should be loaded.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly. This is the end of the program transmission.

Unload a Partition

This sequence is used to unload a partition at the controller.

Host	Controller	Explanation
		The host system sends the controller an N identifier (hex 4E) to indicate the start of a program transmission transaction. The N tells the controller that the next thing sent is an N record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly.
"091 120202020202020201 1 "C rLf		The host sends the first record of the compiled program with the data length (09), the partition number (11), program name (20), checksum (11), Cr (hex OD), and the Lf (hex OA) to indicate the end of the record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly.
"EN"CrLf		The host sends an E record with the identifier E (hex 45), the N operand (hex 47) to tell the controller to unload the partition specified in the N record.
	Ack	The controller responds with an Ack (hex 06). This indicates to the host that the record was received correctly. This is the end of the program transmission.

Put Transaction

A typical PUT host data transaction is provided below:

Host	Controller	Explanation
Xon ———		Controller must be in the Xon State.
	———"D0502vvvvrinnn"csCrLf	When the application program encounters the PUT command, it sends a D record with byte count (05), put op-code (02), starting variable number (vvvv), number of continuous variables (nnnn), check sum (cs), and terminator (Cr,Lf).
Ack ———		The host acknowledges the D record.
	——— D - record	The controller sends a D record with the value for each of the variables.
Ack ———		The host acknowledges the D record.
	——— D - record	The controller sends additional D records until all the variables have been transmitted. Each D record is Acked by the host.
Ack ———		The host acknowledges the D record.
	——— "EG"CrLf	When all the data has been transmitted, the controller sends a E record. The record contains the identifier (E), an operand (G), and the terminator (Cr,Lf).
Ack ———		The host acknowledges the E record ending the transaction.

Get Transaction

A typical GET host data transaction is given below:

Host	Controller	Explanation
Xon		The controller must be in the Xon State.
	"D0503vvvynnnn"csCrLf	When the application program encounters the GET command, it sends a D record with the byte count (05), an operand (03), the starting variable number (vvvv), the number of variables (nnnn), the check sum (cs), and the terminator (Cr,Lf).
D record		The host sends a D record containing the variables requested by the controller.
	Ack	The controller sends an acknowledgment of the record.
D record		The host continues to send D records until all the variables are transmitted.
	Ack	The controller acknowledges the D records when received.
"EG"CrLf		When all the data has been transmitted, the host sends an E record with the identifier (E), an operand (G), and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the E record, ending the transaction. If too much or too little data is received from the host, the controller responds with an Eot with the status set to 75 or 76.

Read Transaction

A typical read transaction is given below:

Host	Controller	Explanation
"R"		The host starts the transaction by sending a **R** identifier
	Xoff	The controller always sends an Xoff first.
	Xon	The controller sends an Xon to acknowledge that it is ready to receive the R record.
"80"C rLf		The host sends a record with the read specific variables op code (80) and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the record.
"DO4vvvvrinnn"csCrLf		The host sends a **D** record with the identifier (D) the byte count (04), the starting variable number (vvvv), the number of variables to be read (nnnn), the check sum (cs), and the terminator (Cr,Lf).
	D record	The controller responds with a **D** record containing the variables.
Ack		The host acknowledges the **D** record.
	D record	If additional **D** records are necessary, the records are sent after each acknowledgement.
Ack		The host acknowledges each **D** record.
	"EG"CrLf	When all the data has been transmitted, the controller sends an **E** record. The record contains an identifier (E), an op-code (G), and the terminator (Cr,Lf).
Ack		The host acknowledges the **EG** (end good) record, ending the transaction.

Read Instruction Address

A typical read instruction address is outlined below.

Host	Controller	Explanation
"R" ———		The host starts the transaction by sending a **R** identifier
	——— **Xoff**	The controller always sends an Xoff first.
	Xon	The controller sends an Xon to acknowledge that it is ready to receive the **R** record.
"08"CrLf ———		The host sends a record with the read instruction address (08) op code, and the terminator (Cr,Lf).
	"DO2dddd"csCrLf	The controller responds with a **D** record with the byte count (02), two bytes of data (dddd), the check sum (cs), and the terminator (Cr,Lf). The data is the current instruction address as provided in the listing created by the AML/Entry compiler.
Ack ———		The host acknowledges the record.
	——— **"EG"CrLf**	The controller sends an **E** record to indicate the end of the data. The record contains an identifier (E), an operand (G), and the terminator (Cr,Lf).
Ack ———		The host acknowledges the **E** record ending the transaction.

Debug Transaction

A typical debug operation is outlined below.

Host	Controller	Explanation
		The host starts the transaction by sending a C identifier.
	Xoff	The controller always sends an Xoff first.
	Xon	The controller sends an Xon to acknowledge that it is ready to receive the remainder of the C record.
"10"CrLf		The host sends the remainder of the record with the Set Debug Stop (10) op code and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the record.
"DO2xxxx"csCrLf		The host sends as a **D** record with the identifier (D), the byte count (02), the hex address for the stop code (xxxx), and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment when the stop code is installed.
Xon		The host sends an Xon to place the controller in the Xon State (able to initiate communications).
	"D011010"CrLf	When a Stop Code is encountered, the controller sends a **D** record with the byte count (01), one byte of data (10), the check sum (10), and the terminator (Cr,Lf).
Ack		The host acknowledges the **D** record.
	"EG"CrLf	The controller sends an **E** record containing an E to indicate that this is an E record, a G to denote that the data is good, and the terminator (Cr,Lf).
Ack		The host acknowledges the **E** record, ending the transaction.

Write Controller Data Transaction

A typical write transaction is given below:

Host	Controller	Explanation
"C"		The host starts the transaction by sending a C identifier.
	Xoff	The controller always sends an Xoff first.
	Xon	The controller sends an Xon to acknowledge that it is ready to receive the **R** record.
"80"CrLf		The host sends the remainder of the record with the op-code (80) and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the record.
"DO4vvvvnnnn"csCrLf		The host sends a **D** record with the identifier (D), the byte count (04), the starting variable number (vvvv), the number of variables to be read (nnnn), and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the record.
D-record		The host sends a **D** record containing the variables requested by the controller.
	Ack	The controller sends an acknowledgment of the record.
D-record		The host sends as many additional **D** records as required to send the requested data.
	Ack	The controller sends an acknowledgment of each additional record
"EG"CrLf		When all the data has been transmitted, the host sends an **E** record. The record contains the identifier (E), an operand (G), and the terminator (Cr,Lf).
	Ack	The controller sends an acknowledgment of the **E** record. This ends the transaction.

Example Application Sequence

Host	Controller	Explanation
"C" ———		The first step of the application is using host-initiated communications to send the computer-generated points to the controller. The host starts the transaction by sending a C identifier.
	——— Xoff	The controller always sends an Xoff first.
	——— Xon	The controller sends an Xon to acknowledge that it is ready to receive the C record.
"80"CrLf ———		The host sends the remainder of the record with the op-code (80), and the terminator (Cr,Lf).
	——— Ack	The controller sends an acknowledgment of the record.
"D040010000818"CrLf ———		The host sends a D record with the identifier (D) the byte count (04), the starting variable number (0010) , the number of variables (from the XREF program) to be sent (0008), and the terminator (Cr,Lf).
	——— Ack	The controller sends an acknowledgment of the record.
"D1003A0000001800000 000000000000000024" C r Lf ———>		The host sends a D record containing the identifier (D) the byte count (10), the first four variables(03A...), the check sum (24), and the terminator (Cr,Lf).
	——— Ack	The controller sends an acknowledgment of the record.
" D1003A0000003A 00000 000000000000000046" C r Lf ———>		The host sends a D record containing the identifier (D) the byte count (08), the final four variables(03A...), the check sum (46), and the terminator (Cr,Lf).
	——— Ack	The controller sends an acknowledgment of each additional record
"EG"CrLf ———		The host sends a E record. The record contains the identifier (E), the op-code (G), and the terminator (Cr,Lf).

Ack — The controller sends an acknowledgment of the **E** record. This ends the write data transaction for the host. The host then sends the controller a signal using a DO point. This allows the application program to start.

> **Note:** The DO signal is used by this application example only. It is not a requirement of the communications protocol.

Xon —> The host sends the controller an Xon to enable the controller to execute controller-initiated communications.

"D05030015000119"CrLf

When the application program encounters the GET (LOC_TO_CARD) command, it sends a **D** record with the byte count (05), an operand (03), the starting variable number (from the XREF program) (0015), the number of variables (0001), the check sum (15), and the terminator (Cr,Lf).

"D0488C8000050"CrLf — The host sends a **D** record containing the byte count (04), the variable (88C...), the check sum (50), and the terminator (Cr,Lf).

Ack — The controller sends an acknowledgment of the record.

"EG"CrLf — The host sends a **E** record with the identifier (E), operand (G), and the terminator (Cr,Lf).

<— Ack — The controller sends an acknowledgment of the **E** record ending the transaction. If the incorrect amount of data is received from the host, the controller responds with an Eot with the status set to 75 or 76.

"D05030020000225"CrLf

When the application program encounters the GET (INSTRS) command it sends a **D** record with the byte count (05), an operand (03), the starting variable number (0020), the number of variables (0002), the check sum (25), and the terminator (Cr,Lf).

')0801800000
0180000002"CrLf ——— The host sends a **D** record containing the byte count (08), the values for each variable (018...), the check sum (02), and the terminator (Cr,Lf).

——— Ack ——— The controller sends an acknowledgment of the record.

"EG"CrLf ——— The host sends a **E** record with the identifier (E), operand (G), and the terminator (Cr,Lf).

——— Ack ——— The controller sends an acknowledgment of the **E** record ending the transaction. If too much or too little data is received from the host, the controller responds with an Eot with the status set to 75 or 76 respectively.

<——— **"D05030020000225"CrLf**

Each time the program encounters the GET (INSTRS) command, it sends a **D** record with the byte count (05), an operand (03), the starting variable number (0020), the number of variables (0002), the check sum (25), and the terminator (Cr,Lf).

"D0800000000
0000000000"CrLf ——— The host sends a **D** record containing the byte count (08), the values for each variable (000...), the check sum (00), and the terminator (Cr,Lf). When the application program receives zero for the first value it ends the loop.

——— **Ack** ——— The controller sends an acknowledgment of the record.

"EG"CrLf ——— The host sends a **E** record with the identifier (E), an operand (G), and the terminator (Cr,Lf).

<——— **Ack** ——— The controller sends an acknowledgment of the **E** record, ending the transaction. If too much or too little data is received from the host, the controller responds with an Eot with the status set to 75 or 76. This ends this application sequence.

INDEX

1 Special Characters 1

--II A-108
--%P A-109
? (Recall) 3-48
? primary command A-107

A

a structural overview 4-2
A, line command 3-18, 3-22, A-2
ABS 4-50, A-3
Ack H-15, H-16
actual parameter assignment restrictions 4-64
additional point 6-22
additional topics for program enhancement 4-74
  host communications 4-87
  pallet 4-74
  region 4-82
aggregate constants 4-35
AML/E Entry Run-Time Errors 8-23
AML/Entry commands 4-6
  abs 4-6, A-3
  atan 4-6, A-4
  atan2 4-6, A-5
  BRANCH 4-7, 4-24, A-8
  BREAKPOINT 4-7, 4-24, A-9
  Commands That Allow Expressions 4-54
  COMPC 4-7, A-17
  cos 4-6, A-19
  counter 4-6, A-20
  cstatus 4-6, 4-7, 4-17, A-22
  DECR 4-7, A-26
  DELAY 4-7, 4-8, A-29
  DPMOVE 4-7, 4-8, A-30
  end 4-6, A-32
  frompt 4-6, A-35
  GET 4-7, 4-87, A-36
  GETPART 4-7, 4-9, 4-77, A-38
  GRASP 4-7, 4-9, A-40
  group 4-6, 4-45, A-41
  GUARDI 4-7, 4-18, A-43
  INCR 4-7, A-46
  ITERATE 4-7, 4-24, A-48
  LEFT 4-9, A-50
  LINEAR 4-7, 4-11, A-51
  mstatus 4-6, 4-7, 4-19, A-56
  new 4-6, A-58
  NEXTPART 4-7, 4-77, A-59
  NOGUARD 4-7, 4-20, A-61
  pallet 4-6, A-62
  PAYLOAD 4-7, 4-11, A-65
  PMOVE 4-7, 4-12, A-67
  PREVPART 4-7, 4-78, A-68
  pt 4-6, A-71
  PUT 4-7, A-72
  region 4-6, A-76
  RELEASE 4-7, 4-12, A-79
  RIGHT 4-13, A-81
  SETC 4-7, A-83
  SETPART 4-7, 4-78, A-85
  sin 4-6, A-87
  sqrt 4-6, A-88
  static 4-6, A-89
  subr 4-6, A-90
  tan 4-6, A-92
  TESTC 4-7, A-93
  testi 4-6, 4-7, 4-20, 4-25, A-94
  testp 4-6, 4-7, 4-78, A-96
  trunc 4-6, A-98
  WAITI 4-7, 4-20, A-99
  WHERE 4-7, 4-21, A-100
  WRITEO 4-21, A-101
  WRITEP 4-7
  XMOVE 4-7, 4-13, 4-84, 4-85, A-102
  ZMOVE 4-7, 4-13, A-104
  ZONE 4-7, 4-13, A-105
AML/Entry diskettes 2-10, 2-38
AML/Entry language programs 2-10
AML/Entry menu 2-23, 7-16, 7-17, 7-20
  option 0 (return to DOS) 2-24
  option 1 (edit/teach) 2-25
  option 2 (compile) 2-26
  option 3 (load) 2-28
  option 4 (unload) 2-29, 7-14
  option 5 (set system configuration) 2-30
  option 6 (set name and options) 2-32
  option 7 (communicate with controller) 2-33
  option 8 (xref) 2-34
AML/Entry messages
  AMLECOMM Error Messages B-60, B-66

COMAID Error Messages B-60
error messages without numbers B-1
messages with error numbers B-47
AML/Entry reserved words 4-6
AML/Entry user subroutines 4-61
AML/Entry utility programs 2-38
AMLECOMM Modules 2-38
OFFSET.EXE 2-38
75XXexX.AML 2-38
800S-exX.AML 2-38
AML/Entry Version 4
application program 4-2
beginning and ending program 4-3
comments 4-2
definition operator 4-5
identifier 4-4
keyword/command 4-5
line number 4-4
statement delimiter 4-5
AMLECOMM 8-30, G-1
BUFFER.A$ 8-51
Calling AMLECOMM 8-36
Configuration Parameters 8-34
Control Operation 8-46, 8-47
Data Drive Operation 8-47
Download Operation 8-45
Execute Operation 8-41
Initialization 8-40
Initialization Parameters 8-34
Installation 8-31
Introduction 8-30
Line Numbers 8-32
Opening COMM port 8-40
Read Operation 8-43
System Files 8-30
Tabular Listing 8-38
Teach Operation 8-45, 8-46
Unload Operation 8-44
Using the Compiler 8-32, 8-33
Using the Interpreter 8-32, 8-33
AMLECOMM Modules 2-38
AMLECOMO.BAS G-1
configuration parameters G-1
AMLECOMO.BAS G-1
appl 7-22, 7-25, 7-26
application program comparison 5-4
application program starting-automatic mode 7-25
application startup sequence H-38
Arithmetic Functions 4-50
ABS(exp) 4-50
ATAN(exp) 4-50
ATAN2 4-50
COS 4-51
CSTATUS 4-52
FROMPT 4-52
MSTATUS 4-52
SIN 4-52
SQRT 4-52
TAN 4-53
TESTI 4-53
TESTP 4-53
TRUNC 4-53
ASCII characters H-15
Ack H-15
Cr H-15
Eot H-15
Lf H-15
Nak H-15
Nul H-15
Xoff H-15
Xon H-15
asynchronous communications 8-2, H-2
ATAN 4-50, A-4
ATAN2 A-5
auto key 7-6, 7-25, 7-26
autoinit 2-10, 2-12, 2-14, 2-15, 2-16, 2-17
automatic operation 7-24
AUX 2-18

B

B, line command 3-18, 3-22, A-7
backspace key 3-6
baud rate
baud rate 8-2, H-2
data bits 8-2, H-2
full duplex 8-2, H-2
parity 8-2, H-2
stop bits 8-2, H-2
beginning and ending your program 4-3
BRANCH 4-24, 4-26, A-8
breaking data into fields H-24
BREAKPOINT 4-24, 4-26, A-9
byte count H-6

C

C, line command 3-22, A-11
CANCEL 3-57, A-12
CAPS 3-58, A-13
CC, line command 3-22, A-14
CHANGE 3-36, A-15
changing manipulator arm mode 6-29
check sum H-6

- Circular Motion 4-55
- clearing error conditions 7-27
- coarse movement 6-17
- Column Number of a Part in a Pallet 4-57
- COMAID 2-33, 8-14
  - C Option 8-19
  - D Option 8-17
  - Debugging AML/E Applications 8-23
  - DOS Command Line 8-21
  - F Option 8-20
  - L Option 8-15
  - P Option 8-14
  - R Option 8-15
  - T Option 8-20
  - U Option 8-15
  - X Option 8-16
- command input 3-30
- command pending message 3-11, 3-12
- Commands That Allow Expressions 4-54
- comments 4-2
- communication commands
  - GET A-36
  - PUT A-72
- communication startup sequence 8-3, H-18
  - data terminal ready 8-3, H-18
- Communication with controller 2-33
- communications 2-4, 8-1
  - AMLECOMM 8-30
  - asynchronous communication 8-2, H-2
  - COMAID 8-14
  - Communication Capabilities 8-4
  - communication startup sequence 8-3, H-18
  - communications hardware interface 8-2, H-2
  - communications protocol H-4
  - controller communications connector 8-3, H-3
  - Data Drive Mode 8-8
  - data line control H-15
  - interface 8-2, H-2
  - reading program variables H-25
  - record descriptions H-19
  - typical communications sequences H-36
- communications hardware interface 8-2, H-2
- communications protocol H-4
- communications sequences H-36
- COMPC 4-19, 4-42, 4-54, A-17
- compile and load application program 7-16
  - with a fixed disk PC 7-16
  - with a PC 7-16
  - with a PC/AT 7-17
- compiler
  - .ASC File 2-26, 2-27
  - .LST File 2-27
  - .SYM File 2-27
  - --%I include file 4-58, A-108
  - --%P page command 4-59, A-109
  - AUX 2-18
  - batch program example 2-37
  - compiler directive 4-58, 4-59
  - compiler errors 2-26
  - COM1 2-18
  - COM2 2-18
  - CON 2-18
  - Converting AML/E Program 2-26
  - displayed information 2-26
  - DOS batch support 2-36
  - error level 2-36
  - Include Files (--%I) 4-58
  - invoking the compiler 2-36
  - LINE 2-18
  - listing file 2-27
  - Load 2-36
  - LPT1 2-18
  - NUL 2-18
  - Page Ejects (--%P) 4-59
  - phase messages 2-26
  - PRN 2-18
  - Reading Input File 2-26
  - set program name and options 2-32
  - symbol file 2-27
  - USER 2-18
  - Writing .ASC File 2-26
- compiler directive 4-58
  - --%I A-108
  - --%1[3] A-109
- compiler errors 2-26
- compiler phase messages 2-26
- COM1 2-18
- COM2 2-18
- CON 2-18
- Condition 1 (teach) 6-10
- Condition 2 (Teach) 6-10
- Condition 3 (Teach) 6-11
- Condition 4 (Teach) 6-11
- Condition 5 (Teach) 6-13
- Configuration Utility menu 2-30
- constants 4-31, 5-13
  - aggregate constants 4-35
  - declaring constants 4-31
  - global constants 4-32
  - global constants vs. local constants 4-32
  - local constants 4-32
  - using constants 4-32
  - using global constants 4-34

using local constants 4-33
using the ITERATE with aggregates 4-35
contrast and brightness 2-5
control key 3-6
control keys 3-8
Ctrl- 3-8
Ctrl-End 3-8
Ctrl-PgUp 3-8
control lights 2-9
control panel 7-3
control panel keys 7-3
auto 7-6
gripper close 7-5
gripper open 7-5
home 7-11
manip power 7-5
manual 7-6
memory 7-4
o.r. reset 7-9
off line 7-6
on line 7-6
power 7-3
rapid 7-7
recall memory 7-5
reset error 7-5
return home 7-5
roll - 7-9
roll + 7-9
start cycle 7-7
step 7-7
stop 7-3
stop and mem 7-7
stop cycle 7-7
THETA 1- 7-8
THETA 1+ 7-8
THETA 2- 7-8
THETA 2+ 7-8
Z down 7-6
Z up 7-6
control switches 2-9
controller application program 4-2
controller communications
controller states 8-8
Data Drive Example 8-11
Data Drive Mode 8-6
DEBUG transaction H-35
GET 4-87
GET transaction H-35
PUT 4-88
PUT transaction H-34
Use with Comaid 8-17
using AMLECOMM 8-47
controller communications connector 8-3, H-3
controller initiated communications 4-87, H-34
controller storage management 7-14
controller switches and lamps 7-2
controlling digital output from teach 6-28
coordinates 6-19
copying DOS and AML/Entry shipped diskettes on to work diskettes 2-12
copying the AML/Entry diskettes on fixed disk drives 2-15
COS 4-51, A-19
COUNTER commands 4-41
counters 4-39, A-20
DECR 4-41
GROUP 4-45
INCR 4-41
PT's defined in terms of counters 4-44
SETC 4-41
TESTC 4-43
using counter statements 4-43
cr-carriage return H-15
Creating Program screen 7-20
creating self-booting AML/Entry diskettes 2-10
CSTATUS 4-17, 4-52, A-22
~~Ctrl- 3-8~~
Ctrl-End 3-8
Ctrl-PgUp 3-8
cts-clear to send 8-3, H-18
cursor 2-5
cursor keys 2-6

D

D, line command 3-15, A-24
data H-6
data rules H-8
Data Drive Mode 8-6, 8-8
controller states 8-8
example 8-11
transitions between states 8-9
Data Errors 8-23
data line control H-15
data representation H-9
data set ready 8-3, H-18
cts-clear to send 8-3, H-18
rts-request to send 8-3, H-18
data terminal ready 8-2, 8-3, H-2, H-18
DD, line command 3-15, A-25
DE-data error 7-5
DEBUG transaction H-35, H-47

Debugging AML/E Applications 8-23
   Control Requests 8-26
   Read Requests 8-23
declarations 4-30
   declaring constants 4-31
   declaring variables 4-38
   using declarations 4-30
declaring aggregate constants 4-35
declaring constants 4-31
declaring variables 4-38
DECR 4-41, A-26
define global data types 5-10
define global subroutines 5-14
definition operator 4-5
DEL 3-56, A-28
Del key 3-7
DEL, primary command 2-19
DELAY 4-8, A-29
description of a pallet 4-74
design application program
   constants 5-13
   define global data types 5-10
   define global subroutines 5-14
   digital input and output 5-11
   gripper subroutines 5-18
   initialization subroutine 5-22
   main subroutine 5-24
   movement subroutine 5-17
   parts handling subroutines 5-19
   taught points 5-10
   utility subroutines 5-14
   variables 5-13
development of user subroutines 4-61
device name 2-17
digital input and output 5-11
disk operating system (DOS) 2-10
diskcopy 2-17
diskettes 2-11, 2-19
display 2-3, 2-5
   contrast and brightness controls 2-5
   cursor 2-5
displayed information 2-26
DOS (XREF program) 2-34
DOS batch support 2-36
DPMOVE 4-8, A-30
drive door 2-8

I E

edit/teach 2-25
editor
   ? 3-48
   exiting the editor 3-10
   keyboard usage for the editor 3-4
   set program name and options 2-32
editor commands
   ? 3-48
   CANCEL 3-57
   CAPS 3-58
   CHANGE 3-36
   DEL 3-56
   FILES 3-46
   FIND 3-32
   GETFILE 3-52
   LOCATE 3-42
   PRINT 3-54
   PUTFILE 3-53
   RENAME 3-50
   SAVE 3-44
editor exercises 3-11
Editor File Sizes 3-1
editor function key settings 3-4
editor help screens 3-4
editor special key 3-6
   backspace key 3-6
   control key 3-6
   enter key 3-6
   PrtSc key 3-6
   shift key 3-6
   tab key 3-6
END A-32
end key 3-7
enter key 2-6, 3-6
entering known coordinates from the
 keyboard 6-19
Eot H-17
error LEDS 7-3
   DE-data error 7-5
   OR-overrun 7-4
   OT-over-time 7-4
   PF-power failure 7-4
   SE-servo error 7-4
   TE-transmission error 7-4
error level 2-36
example batch program 2-37
example of a subroutine with formal
 parameters 4-63
example of palletization 4-79
exercises for teach mode 6-14
exiting the editor 3-10
Expressions 4-48
   Arithmetic Functions 4-50
      ABS(exp) 4-50
      ATAN(exp) 4-50
      ATAN2 4-50
      COS 4-51
      CSTATUS 4-52
      FROMPT 4-52
      MSTATUS 4-52

SIN 4-52
SQRT 4-52
TAN 4-53
TESTI 4-53
TESTP 4-53
TRUNC 4-53
Circular Motion Example 4-55
Commands That Allow Expressions 4-54
DI as Integers Example 4-56
Precision 4-39
Round-off Error 4-39
Row and Column of a Part in a Pallet Example 4-57

F

FILE and ?, primary commands 3-46
file types
.AML 2-19
.ASC file 2-28
.LST 2-19, 2-27
.SYM 2-19, 2-27
filename 2-17
files 2-17, A-33
FIND, primary command 3-32, A-34
Fixed diskettes 2-15
floating point examples H-10
flow-of-control commands 4-23
BRANCH 4-24
BREAKPOINT 4-24
COMPC 4-42, A-17
ITERATE 4-24
Labels 4-23
TESTI 4-25
using flow-of-control commands 4-26
formal parameter names restrictions 4-64
formal parameters 4-79
formal parameters in subroutines 4-63
frame of reference 4-82
FROMPT 4-21, 4-52, A-35
full screen editing 3-1
function key settings, editor 3-4
Fl HELP 3-5
F10 BOTTOM 3-5
F2 RESHOW 3-5
F3 RESET 3-5
F4 FIND 3-5
F5 CHANGE 3-5
F6 TEACH 3-5
F7 RECALL 3-5
F8 EXIT 3-5
F9 TOP 3-5
function keyboard usage, teach
exiting DO control (F6) 6-9
Fl 6-5
F10 6-5
F2 6-5
F3 6-5
F4 6-5
F5 6-5
F6 6-5
F7 6-5
F8 6-5
F9 6-5
read DI/DO points (F5) 6-8
set motion parameters (F2) 6-9
setting motion parameters for safety 6-8
Functions 4-50
ABS(exp) 4-50
ATAN(exp) 4-50
ATAN2 4-50
COS 4-51
CSTATUS 4-52
FROMPT 4-52
MSTATUS 4-52
SIN 4-52
SQRT 4-52
TAN 4-53
TESTI 4-53
TESTP 4-53
TRUNC 4-53

G

GET 4-87, A-36
GET transaction H-35, H-44
GETFILE 3-52, A-37
GETPART 4-9, 4-77, A-38
getting to the editor from the main menu 3-9
global constants 4-32
global constants vs.local constants 4-32
good program structure 5-1
use blank lines 5-2
use comments 5-2
use declarations 5-1
use expressions 5-2
use indentations 5-1
use names 5-2
use subroutines 5-1
GRASP 4-9, A-40
gripper close key 7-5
gripper open key 7-5

gripper subroutines 5-18
GROUP 4-45, 5-13, A-41
GUARDI 4-18, A-43

H

handling diskettes 2-20
help screens, editor 3-4
home key 3-7, 7-4
host communications 4-74, 4-87
communications sequences H-36
Control Request 8-19, 8-46, 8-47, H-26
data drive 4-87
data reporting 4-87
Download Request 8-4, 8-15, 8-45, H-29
Execute Request 8-5, 8-16, 8-41, H-28
Present Configuration Request H-33
Read Request 8-4, 8-15, 8-43, H-19
Teach Request 8-6, 8-20, 8-45, 8-46, H-31
Unload Request 8-4, 8-15, 8-44
variable identification 4-89
XREF program 4-89

I

I, line command 3-13, A-45
IBM Manufacturing System teach responses 6-10
Condition 1 (manipulator power off) 6-10
Condition 2 (Manipulator Power Off) 6-10
Condition 3 (Exit teach and Editor) 6-11
Condition 4 (Exit teach) 6-11
Condition 5 (Exit teach) 6-13
IBM PC 2-2
in-use conditions 2-2
storage conditions 2-2
IBM PC configuration 2-3
IBM PC requirements 2-2
identifier 4-4
identifiers H-5, H-6
Control (C) H-5
data H-5
end H-5
execute H-5
new H-5
Present Configuration (p) H-5
read H-5
Teach (T) H-5
in-use light 2-8
Include file compiler directive A-108
Include Files (--%I) 4-58
including comments 4-2
INCR 4-41, A-46
indexing 4-46
information about primary commands 3-30
initialization subroutine 5-22
Ins key 3-7
inserting diskettes 2-11
interface
RS-232 8-2, H-2
RS-422 8-2, H-2
invoking the compiler 2-36
/B batch mode operation 2-36
/E aborts compile 2-36
/H hard copy report 2-36
/L listing file 2-36
/S symbol file 2-36
ITERATE 4-24, 4-35, 4-54, A-48
ITERATE to repeat a subroutine 4-70

KI

key functions, control panel 7-3
auto 7-6
gripper close 7-5
gripper open 7-5
home 7-11
manip power 7-5
manual 7-6
memory 7-4
o.r. reset 7-9
off line 7-6
on line 7-6
power 7-3
rapid 7-7
recall Memory 7-5
reset error 7-5
return home 7-5
roll - 7-9
roll + 7-9
start cycle 7-7
step 7-7
stop 7-3
stop and mem 7-7
stop cycle 7-7
THETA 1- 7-8

THETA 1+ 7-8
THETA 2- 7-8
THETA 2+ 7-8
Z down 7-6
Z up 7-6
keyboard 2-3, 2-6
Alt 2-7
Ctrl 2-7
cursor keys 2-6
Del 2-7
enter key 2-6
function keyboard 2-6
numeric keypad 2-6
typewriter keyboard 2-6
keyboarding known coordinates 6-19
keys 2-6
keyword/command 4-5
keywords 4-6
abs 4-6
atan 4-6
atan2 4-6
cos 4-6
counter 4-6, A-20
cstatus 4-6
end 4-6, A-32
frompt 4-6
group 4-6, A-41
mstatus 4-6
new 4-6, A-58
pallet 4-6, A-62
pt 4-6, A-71
region 4-6, A-76
sin 4-6
sqrt 4-6
static 4-6, A-89
subr 4-6, A-90
tan 4-6
testi 4-6
testp 4-6
trunc 4-6

L

labels 4-23
language structure 4-2
LEDS, control panel 7-3
DE-data error 7-5
home 7-4
memory 7-4
OR-overrun 7-4
OT-over-time 7-4
PF-power failure 7-4
SE-servo error 7-4
TE-transmission error 7-4
LEFT 4-9,:A-50
Using Groups with 7545-800S 4-47
LEFT mode 6-29
if-line feed H-15
LINE 2-18
line command conflicts 3-12
line command I 3-13
line command R 3-26
line commands 3-11
block copy 3-11
block delete 3-11
block move 3-11
copy line 3-11
copy/move after 3-11
delete line(s) 3-11
insert line(s) 3-11
move line 3-11
repeat line(s) 3-11
line commands C, CC, with A or B 3-22
line commands D and DD 3-15
line commands M, MM, with A or B 3-18
line commands that cross screens 3-12
line edit commands
A 3-18, 3-22, A-2
B 3-18, 3-22, A-7
C 3-22, A-11
CC 3-22, A-14
D 3-15, A-24
DD 3-15, A-25
I 3-13, A-45
M 3-18, A-54
MM 3-18, A-55
R 3-26, A-75
line number 4-4
LINEAR 4-11, A-51
listing file 2-27
load 2-36
load a program 2-28
load file (.ASC) 2-26
loading the self-booting work diskette 2-21
local constants 4-32
local RS-232-C cable wiring F-1
local RS-422 cable wiring F-2
LOCATE 3-42, A-53
LPT1 2-18

M

M, line command 3-18, A-54
main subroutine 5-24
Making Backup Copies of AML/Entry Work Diskettes 2-17
manip power key 7-5, 7-10
manipulator arm mode, changing 6-29
manipulator stop cycle sequence H-37
manual key 7-6, 7-22, 7-23
manual mode control of axis motors, Z-axis and gripper 7-24
manual mode motors, Z-axis and gripper 7-24
manual operation of the manipulator 7-23
memory key 7-4
Minimum PC requirements 2-2
MM, line command 3-18, A-55
motion commands 4-8
- DELAY 4-8
- DPMOVE 4-8
- GETPART 4-9
- GRASP 4-9
- LEFT 4-9
- LINEAR 4-11
- PAYLOAD 4-11, A-65
- PMOVE 4-12
- RELEASE 4-12
- RIGHT 4-13
- using motion statements 4-14
- XMOVE 4-13, A-102
- ZMOVE 4-13
- ZONE 4-13

movement subroutines 5-17
MSTATUS 4-19, 4-52, A-56
multiple statements on a line 4-29

N

nak H-15, H-17
NEW A-58
NEXTPART 4-77, A-59
NOGUARD 4-20, A-61
NUL 2-18, H-8, H-15, H-18
num lock key 3-7
numeric keypad 3-7
- del key 3-7
- end key 3-7
- home key 3-7
- ins key 3-7
- num lock key 3-7
- page down key 3-7
- page up key 3-7

O

o.r. reset 7-9
obtaining an additional point 6-22
off line key 7-6, 7-22, 7-23
on line key 7-6, 7-15, 7-20
on/off power switch 2-8, 2-9
Option 0 (AML/Entry menu) 2-24
Option 1 (AML/Entry Menu) 2-25
Option 2 (AML/Entry menu) 2-26
Option 3 (AML/Entry menu) 2-28
Option 4 (AML/Entry menu) 2-29
Option 5 (AML/Entry menu) 2-30
Option 6 (AML/Entry menu) 2-32
Option 7 (AML/Entry menu) 2-33
Option 8 (AML/Entry Menu) 2-34
OR-overrun LED 7-4
OT-over-time LED 7-4
ownership and multiple name occurrence 4-72

P

Page compiler directive 4-59, A-109
page down key 3-7
Page Ejects (--%P) 4-59
page up key 3-7
pallet 4-74, A-62
palletization example 4-79
palletizing 4-79
Palletizing Commands 4-77
- GETPART 4-77
- NEXTPART 4-77
- PREVPART 4-78
- SETPART 4-78
- TESTP 4-78
- using palletizing statements 4-79

panel, control 7-3
parameter passing 4-63
parameters in subroutines 4-68
parameters, restrictions 4-64
parts handling subroutines 5-19
PAYLOAD 4-11, A-65
PC configuration 2-3
- communications 2-4
- display 2-3

keyboard 2-3
printer 2-4
system unit 2-4
PC environmental considerations 2-2
PF-power failure 7-4
phase messages 2-26
PMOVE 4-12, A-30, A-67
power LED 7-3
power-off sequence 7-13
power switch locations 2-8
power-up sequence for teach exercises 6-15
coarse movement 6-17
controlling digital output from teach 6-28
entering known coordinates 6-19
precision movement 6-18
retrieving a point from a program 6-23
return point value to program (recall) 6-21
powers of two table H-14
precision movement 6-18
Precision of Counters 4-39
PREVPART 4-78, A-68
primary commands, information 3-30
primary edit commands
? 3-46, 3-48, A-107
CANCEL 3-57, A-12
CAPS 3-31, 3-58, A-13
CHANGE 3-36, A-15
DEL 3-56, A-28
FILES 3-46, A-33
FIND 3-32, A-34
GETFILE 3-52, A-37
LOCATE 3-42, A-53
PRINT 3-54, A-70
PUTFILE 3-53, A-73
RENAME 3-50, A-80
SAVE 3-44, A-82
PRINT 3-54, A-70
printer 2-4, 2-9
control lights 2-9
control switches 2-9
on/off power switch 2-9
PRN 2-18
program transmit sequence H-41
Programming System Options menu 2-32
PT A-71
PT's (formals and/or counters) 4-44
PUT 4-88, A-72
PUT transaction H-34, H-43
PUTFILE 3-53, A-73

R

R, line command 3-26, A-75
rapid key 7-7
read instruction address H-46
read transaction H-45
reading program variables H-25
C (control) - records H-26
d (data) - record H-30
data representation H-9
e (end) - record H-30
floating point examples H-10
n (compiled program) - record H-29
p (present configuration) - record H-33
t (teach) - record H-31
X (execute) H-28
recall memory key 7-5, 7-26
record descriptions H-19
control H-19
data H-19
end H-19
execute H-19
new H-19
present configuration H-19
✓ - (read) record H-19
read H-19
teach H-19
record termination H-7
records and record format H-6
byte count H-6
check sum H-6
data H-6
data rules H-8
identifier H-6
record termination H-7
region 4-13, 4-74, 4-85, 5-7
application flow 5-9
component feeders 5-7
interaction with host computer 5-8
manipulator gripper 5-7
REGION A-76
region command 4-82, 4-84
coordinate generation 4-84
external coordinate system 4-82
roll coordinate 4-85
using REGION 4-85
X and Y coordinates 4-84
XMOVE 4-84
Z coordinate 4-84
RELEASE 4-12, A-79
removing power after teach exercises 6-30
RENAME 3-50, A-80

request to send 8-2, H-2
reserved words and commands
   AML/Entry reserved words 4-6
   arithmetic functions in
    expressions 4-6
   commands 4-6
   functions 4-6
   keywords 4-6
reset error key 7-5
restrictions on parameters 4-64
resuming an application program from a
 breakpoint 7-26
retrieving a point from a program 6-23
return home key 7-5, 7-11
return point value to program
 (Recall) 6-21
RIGHT 4-13, A-81
   Using Groups with 7545-800S 4-47
RIGHT mode 6-29
roll - 7-9
roll + 7-9
Roll coordinate 4-85
Round-off Error 4-39
Row Number of a Part in a Pallet 4-57
RS-232 8-2, F-1, H-2
RS-422 8-2, F-2, H-2
rts-request to send 8-3, H-18
   = refid=hcom.Control Request 8-5
rules for calling subroutines 4-66
Run-Time Errors 8-23

S

sample application 5-3
SAVE 3-44, A-82
SE-servo error LED 7-4
self-booting AML/Entry work
 diskette 2-21
   method 1 - power switch off 2-21
   method 2 - system RESET 2-22
sensor commands 4-17
   CSTATUS 4-17, A-22
   GUARDI 4-18, A-43
   MSTATUS 4-19, A-56
   NOGUARD 4-20, A-61
   TESTI 4-20
   using sensor statements 4-21
   WAITI 4-20, A-99
   WHERE 4-21, A-100
   WRITEO 4-21
sequences, communications H-36
   application startup sequence H-38
   debug transaction H-47
   get transaction H-44
   manipulator stop cycle sequence H-37
   program transmit sequence H-41
   put transaction H-43
   read instruction address H-46
   read transaction H-45
   reason for data error H-40
   unload a partition H-42
   write transaction H-48
set name and options 2-32
set system configuration 2-30
SETC 4-41, 4-54, A-83
SETPART 4-78, A-85
setup for your editor exercises 3-9
shift key 3-6
shipped diskettes 2-12
SIN 4-52, A-87
special keys, editor
   backspace key 3-6
   control key 3-6
   enter key 3-6
   PrtSc key 3-6
   shift key 3-6
   tab key 3-6
special keys, teach
   Esc 6-6
   I 6-6
speed/weight values
   7545 Speed/Weight Relationship based
    on Z Position E-2
   7545-800S Speed/Weight Relationship
    based on Z Position E-4
   7547 Speed/Weight Relationship based
    on Z Position E-5
SQRT 4-52, A-88
start cycle 7-7, 7-25, 7-26
starting an application
 program-automatic mode 7-25
statement delimiter 4-5
STATIC A-89
step 7-7
stop and mem 7-7, 7-12
stop button 7-3, 7-10, 7-13
stop cycle 7-7, 7-12
stop key 7-12
stopping the manipulator 7-12
   step 7-12
   stop 7-12
   stop and mem 7-12
   stop cycle 7-12
SUBR A-90
subroutine with formal parameters 4-63
subroutine, initialization 5-22
subroutine, main 5-24
subroutine, movement 5-17
subroutines 4-60, 4-65

development of user subroutines 4-61
example of subroutine with formal parameters 4-63
formal parameters in subroutines 4-63
ownership and multiple name occurrence 4-72
parameter passing 4-63
rules for calling subroutines 4-66
system subroutines 4-60
user subroutines 4-60
user subroutines in the AML/Entry program 4-61
using ITERATE to repeat a subroutine 4-70
using subroutines 4-65
using subroutines with parameters 4-68
subroutines with parameters 4-68
subroutines, development for user 4-61
subroutines, global 5-14
subroutines, gripper 5-18
subroutines, parts handling 5-19
subroutines, rules for calling 4-66
subroutines, utility 5-14
switch locations 2-8
symbol file 2-27
system power-up sequence 7-10
system subroutines 4-60
system unit 2-4, 2-8
drive door 2-8
in-use light 2-8
on/off power switch 2-8

T

tab key 3-6
TAN 4-53, A-92
taught points 5-10
TE-transmission error 7-4
teach function keys
Fl 6-5
F10 6-5
F2 6-5
F3 6-5
F4 6-5
F5 6-5
F6 6-5
F7 6-5
F8 6-5
F9 6-5
teach mode 6-1, 6-3
teach mode exercises 6-14
teach mode, power-up sequence 6-15
teach responses, IBM manufacturing system 6-10
teach, controlling digital output 6-28
teach, removing power after teach exercises 6-30
teach, special keys
Esc 6-6
I 6-6
techniques to simplify programming 4-28
declarations 4-30
declaring constants 4-31
multiple statements on a line 4-29
variables 4-38
TESTC 4-19, 4-43, 4-54, A-93
TESTI 4-20, 4-25, 4-26, 4-53, A-94
testing application programs in manual mode 7-22
TESTP 4-53, 4-78, A-96
THETA 1- key 7-8
THETA 1+ key 7-8
THETA 2- key 7-8
THETA 2+ key 7-8
transactions H-4
identifier H-4
record H-4
transmit data 8-2, H-2
Treating DI As Integers 4-56
TRUNC 4-53, A-98
typical communications sequences H-36
application startup sequence H-38
debug transaction H-47
get transaction H-44
manipulator stop cycle sequence H-37
program transmit sequence H-41
put transaction H-43
read instruction address H-46
read transaction H-45
reason for data error H-40
unload a partition H-42
write transaction H-48

unload a controller program 2-29
unload a partition H-42
USER 2-18
user subroutines 4-60
user subroutines in the AML/Entry program 4-61
using constants 4-32
using counter statements 4-43
using declarations 4-30

using flow-of-control commands 4-26
using global constants 4-34
using linear, speed, and precise motions 4-16
using local constants 4-33
using motion statements 4-14
using move, z-axis, delay and gripper commands 4-14
using sensor statements 4-21
using the ITERATE command with aggregates 4-35
utility subroutines 5-14

V

Values for the LINEAR Command C-1
Values for the PAYLOAD Command D-1
variable identification 4-89
variable structures
   counters 4-39
   frame of reference 4-82
   group 4-45
   GROUP keyword A-41
   indexing 4-46
   pallets 4-79
variables 5-13
variables structures 4-38

W

WAITI 4-20, 4-21, A-99
WHERE 4-21, A-100
work diskettes 2-10, 2-12
write controller data transaction H-48
   write transaction
      example application sequence H-49
WRITEO 4-21, A-101
writing a complex AML/Entry program 5-7
   main application task 5-7
   printed circuit card 5-7
writing a simple AML/Entry program 5-3

X

X and Y coordinates 4-84
XMOVE 4-13, 4-84, 4-85, A-102
xoff H-15, H-16
xon H-15, H-17
XREF program 2-34, 4-89
   Output Listing 4-89
   Pallet Listing 4-90
   piping output to printer 2-35
   Region Listing 4-90

Z

Z-axis and gripper 7-24
Z coordinate 4-84
Z down 7-6
Z up 7-6
ZMOVE 4-13, 4-54, A-104
ZONE 4-13, A-105

Numerics

7545 Program Speed Values For PAYLOAD Command D-1
7545 Speed/Weight Relationship based on Z Position E-2
7545-800S Program Speed Values For PAYLOAD Command D-2
7545-800S Speed/Weight Relationship based on Z Position E-4
7547 Program Speed Values For PAYLOAD Command D-3
7547 Speed/Weight Relationship based on Z Position E-5

AML/Entry Version 4 User's Guide
(Second Edition)
Order No. 58X7338

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply, in any way it believes appropriate without incurring any obligation to you.
Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system. to your IBM representative or to the IBM branch office serving your locality.*

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

58X7338

Printed in U.S.A.

Reader's Comment Form

Fold and tape — Please Do Not Staple — Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK. N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
RS Information Development, Department 9C9
P.O. Box 1328
Boca Raton, Florida 33432

Fold and tape — Please Do Not Staple — Fold and tape