UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Survivor (Lazzaro)

Elaborato di esame di: **Pierpaolo Apollonio, Giulio Del Bono, Raffaele Ippolito**

Consegnato il: **20 agosto 2002**

Sommario

Lo scopo del nostro elaborato è quello di realizzare la parte elettronica per il movimento controllato di un prototipo di sedia robot. Abbiamo sviluppato una scheda basata sul microcontrollore PIC16F876A in grado di gestire i segnali provenienti dai 5 sensori all'infrarosso e dalla batteria; inoltre fornisce i comandi ai motori collegati alle zampe della sedia. Abbiamo scritto il programma tramite un linguaggio assembly dedicato che realizza i comportamenti desiderati.

1. Introduzione

La sedia robot è una normale sedia a cui sono state tagliate le "gambe" anteriori per permettere l'aggiunta di due articolazioni, collegate a due motori, che consentono il movimento. Dando dei segnali di comando a ciascun motore è possibile far compiere alla sedia dei passi. Questa in pratica si trascina ed il movimento è reso più scorrevole con l'aggiunta di rotelle all'estremità delle gambe posteriori. Gestendo la sequenza dei segnali di comando è possibile far camminare la sedia avanti (una successione di passi alternati), a destra (una successione di passi con la zampa sinistra) e a sinistra (passi con la zampa destra).

Inoltre controllando la temporizzazione dei passi è possibile far assumere alla sedia un'andatura "sicura" oppure "arrancante". Ciò è realizzato facendo cominciare un passo subito prima che termini l'altro (andatura sicura) oppure aspettando che un passo sia completamente terminato prima di far iniziare l'altro (andatura arrancante).

2. Il problema affrontato

L'idea di partenza era quella di poter far camminare la sedia in qualsiasi ambiente chiuso. Per far questo ci siamo serviti di 5 sensori all'infrarosso per rilevare ostacoli non troppo piccoli (come ad es: gambe dei tavoli e sedie arrotondate e di metallo, ecc…) e di posizione non troppo velocemente variabile.

Il problema principale è stato quello di fornire alla sedia dei comportamenti che simulassero alcuni stati d'animo tipici dell'essere umano: paura e tranquillità.

La paura è stata simulata creando una camminata veloce che comincia dopo che qualcuno o qualcosa sono stati rilevati troppo vicini e c'è spazio attorno per scappare, altrimenti si ferma ed emette un suono che è come un pianto.

La tranquillità invece è lo scorrimento di un tempo casuale di attesa seguita da una camminata lenta e pacata.

3. La soluzione adottata

Per risolvere questi problemi abbiamo adottato un hardware basato su microcontrollore MICROCHIP PIC16F876A perché contiene una memoria FLASH riprogrammabile, un convertitore A/D a 10bit e 5 ingressi analogici.

Dato che il numero di ingressi analogici non è sufficiente per controllare direttamente i 5 sensori e anche la batteria, abbiamo utilizzato anche un multiplexer analogico MAXIM DG408DJ “8 a 1” ad alimentazione singola e con uscita compresa in un range tra 0 e 5 Volt.

Sono così necessari 3 uscite del PIC (bit 1, 2, e 3 del PORT B) per indirizzare il corretto sensore. Il multiplexer è collegato al bit 0 del PORT A e la batteria al bit 1; per fornire un adeguato valore di tensione (minore di 5V) è necessario un partitore di tensione per abbassare i 12 Volt della batteria.

Per quanto riguarda le alimentazioni, i motori funzionano a 12V e la logica a 5V. Quindi sulla scheda si può trovare l’usatissimo 7805 per la corretta alimentazione. I comandi ai motori vengono dati con un impulso di durata minima di 500ms e con tensione di 12V, quindi possono venire direttamente dal PIC tramite un’ apposita rete di transistor; le uscite del microcontrollore il bit 6 del PORT B per il motore destro ed il bit 7 (sempre del PORT B) per il sinistro.

3.1. Programma del PIC

Il programma che abbiamo scritto per il microcontrollore è strutturato come illustrato dal seguente diagramma di flusso (figura 1):

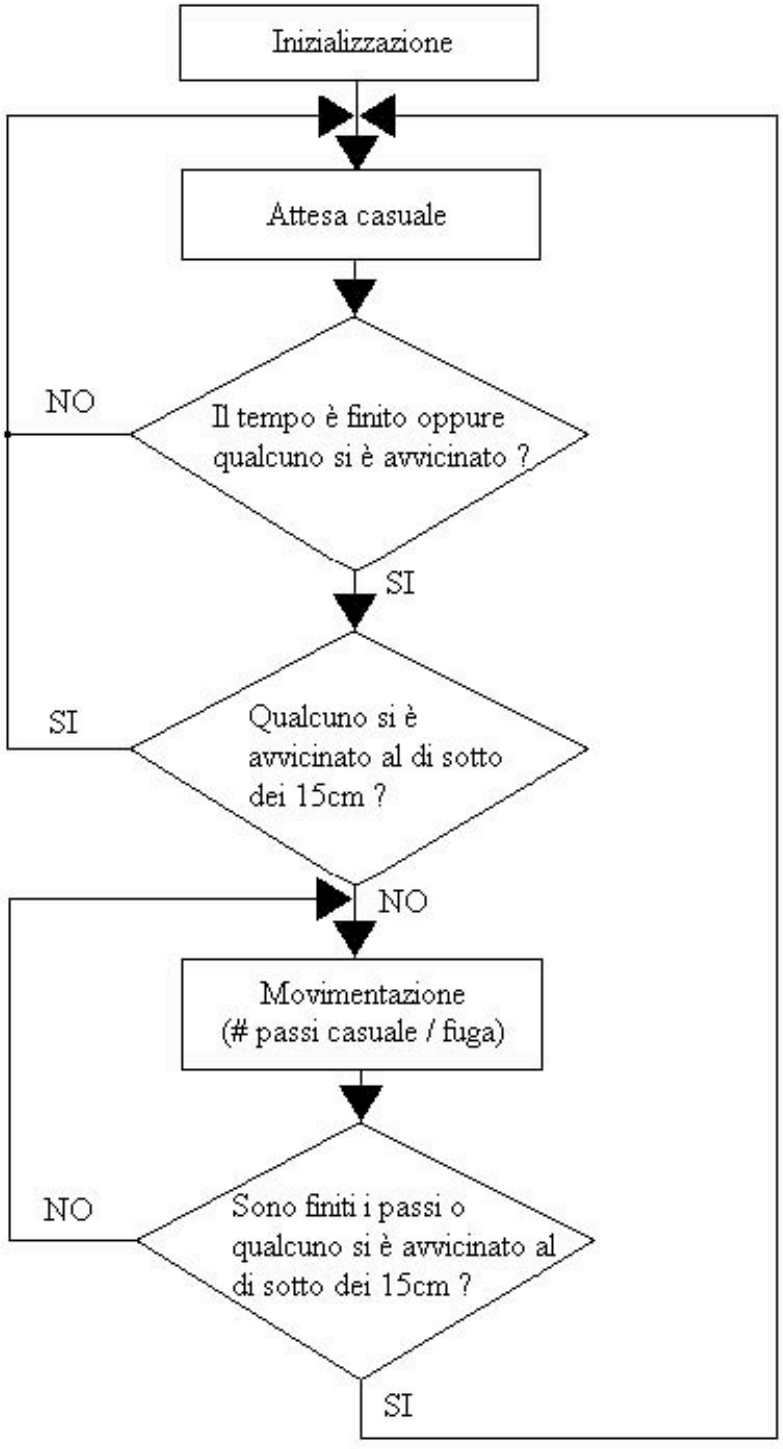


Figura 1

Ecco una descrizione dei vettori di flag utilizzati nel programma:

bit 7	6	5	4	3	2	1	0	decisione

bit 7 : (c'è qualcuno a sinistra) se a 1 indica la presenza di un ostacolo a meno di 80cm sulla sinistra
bit 6 : (c'è qualcuno avanti) se a 1 indica la presenza di un ostacolo a meno di 80cm davanti
bit 5 : (c'è qualcuno a destra) se a 1 indica la presenza di un ostacolo a meno di 80cm sulla destra
bit 4 : (sto scappando) se a 1 indica che a causa della presenza di ostacoli la sedia è in "fuga"
bit 3 : (avanti lento) se a 1 indica che la sedia deve muoversi in avanti con andatura lenta
bit 2 : (SX) se a 1 indica che la sedia deve muoversi girando a sinistra
bit 1 : (avanti veloce) se a 1 indica che la sedia deve muoversi in avanti con andatura veloce (fuga)
bit 0 : (DX) se a 1 indica che la sedia deve muoversi girando a destra

bit 7	6	5	4	3	2	1	0	step_genera

bit 7 : (motore SX) se a 1 indica che il motore sinistro è l'ultimo che è stato azionato
bit 6 : (continuo SX) se a 1 indica che è in corso un moto continuo del motore sinistro
bit 5 : (continuo DX) se a 1 indica che è in corso un moto continuo del motore destro
bit 4 : (devi muoverti) se a 1 indica che è trascorso il tempo casuale oppure che qualcosa si è avvicinato in un raggio di 50cm
bit 3 : (c'è qualcuno) se a 1 indica la presenza di qualcosa in un raggio di 50cm
bit 2 : (finito il tempo) se a 1 indica che è trascorso il tempo di attesa casuale
bit 1 : (utilizzo # casuale) se a 1 indica che è già in uso un numero casuale precedentemente generato
bit 0 : (motore DX) se a 1 indica che il motore destro è l'ultimo che è stato azionato

bit 1	0	corri

bit 1 : (c'è qualcosa TROPPO vicino) se a 1 indica la presenza di qualcosa in un raggio di 15 cm
bit 0 : (continua a correre) se a 1 indica che la sedia deve continuare a muoversi velocemente anche se non ci sono più ostacoli intorno

Figura 2

Il programma è costituito da una prima fase di inizializzazione in cui vengono preparate le variabili in uso nelle varie procedure. Terminata questa fase si entra nel programma vero e proprio costituito dal "main".

3.1.1. main

Il corpo principale del programma è composto da alcune istruzioni di azzeramento di flag (bit 1, 2 e 4 di "step_genera", bit 4 di "decisione", bit 0 di "corri" (vedi fig. 2)), dall'annullamento di eventuali segnali di comando ai motori (bit 6 e 7 di PORTB), dall'azzeramento del timer di conteggio del tempo casuale (TIMER1H E TIMER1L) e da due cicli distinti. Il primo ciclo (il cui inizio è segnalato dall'etichetta "aspetta") è il ciclo di attesa di un tempo casuale compreso tra 16,7s e 67s ed il secondo (che inizia con l'etichetta "lazzaro") è il ciclo di movimentazione. Il ciclo di attesa è costituito da

una continua lettura dei dati provenienti dai sensori (procedura “acquis” che permette inoltre di impostare il valore del tempo di attesa casuale), dalla loro interpretazione (procedure “ghost” e “warning”) e dal polling di due bit (bit 4 di “step_genera” e bit 1 della variabile “corri” (vedi fig. 2)). L’uscita da questo ciclo dipende dal valore assunto dai due bit che vengono testati i quali danno una risposta alle due domande illustrate nel diagramma di flusso di figura 1.

Il ciclo di movimentazione inizia con una nuova lettura dei dati dei sensori (procedura “acquis” che inoltre imposta il numero di passi casuale oppure il numero di passi predefinito per scappare) e prosegue con la loro interpretazione precisa (le procedure “ghostDX”, “ghostSX”, “ghostAV” indicano rispettivamente la presenza o meno nel raggio di 80cm di ostacoli a destra, sinistra e davanti), con la verifica della presenza di ostacoli quasi a contatto (procedura “warning”) e con l’eventuale impostazione del comando di accensione ad uno dei motori (procedura “decis” che stabilisce quale motore attivare e procedura “motori” che effettivamente fornisce il segnale di comando al motore designato). Come si vede dal diagramma sopra l’uscita da questo ciclo è condizionata dal fatto di aver effettuato tutti i passi previsti o dal fatto di essere prossima ad un ostacolo.

3.1.2. acquis

Questa è la procedura che si occupa dell’utilizzo del convertitore A/D integrato nel microcontrollore al fine di avere a disposizione in 5 locazioni di memoria (locazioni “sensore”,…,”sensore+4”) i valori delle tensioni di uscita dei 5 sensori di distanza. Per poter attivare ed utilizzare il convertitore è necessario impostare i vari bit del registro ADCON0 con i valori opportuni (per maggiori dettagli consultare il datasheet del PIC16F876). Con i valori da noi impostati si sceglie il canale 0 del PORTA come sorgente per i segnali analogici e si imposta il minimo tempo di conversione (dipendente dalla frequenza di clock, come indicato nel datasheet). È inoltre necessario impostare anche i bit del registro ADCON1 (vedi datasheet PIC16F876). Con i valori scelti abbiamo deciso di avere il risultato della conversione giustificato a sinistra (in modo da avere nel registro ADRESH gli 8 bit più significativi del numero a 10 bit che risulta) e di avere il porto A costituito da 5 ingressi analogici. A questo punto si entra nel ciclo di acquisizione vera e propria (che inizia con l’etichetta “loop5”). Prima di tutto si inizializza la variabile di appoggio “i” ad un valore diverso da zero e lo si lascia non nullo finché non viene indirizzato l’ultimo canale del multiplexer (“sel” uguale a zero). Al posto della variabile “sel” che contiene gli indirizzi del multiplexer (mux) viene usata la variabile “i” per decidere quando uscire dal ciclo per permetterci di eseguirlo anche nel momento in cui “sel” diventa 0 (dato che è comodo verificare la condizione di uscita al termine del ciclo). In seguito “sel” viene ruotata verso sinistra, in modo da avere i bit che costituiscono l’indirizzo del mux nella posizione appropriata (bit 1, 2 e 3. Non viene usato il bit 0 perché questo bit può essere utilizzato come interrupt esterno), e trasferita sul PORTB indirizzando di fatto il corretto canale del mux. A questo punto è necessario introdurre un ritardo di circa 20 _s per permettere al condensatore di hold che si trova in ingresso all’ A/D di caricarsi al valore della tensione che arriva dal mux. Fatto questo si fa partire la conversione settando il bit 2 (GO/DONE) del registro ADCON0. Quando la conversione è terminata questo bit viene resettato dal microcontrollore e quindi per sapere quando il registro ADRESH contiene un dato valido basta eseguire un ciclo di polling su questo bit. Infatti si esce da questo ciclo solo quando il bit GO/DONE è tornato a 0. Il dato contenuto in ADRESH viene copiato in memoria utilizzando la modalità di indirizzamento indiretto (per maggiori dettagli consultare il datasheet del PIC16F876).

Durante la fase di scrittura in memoria viene anche costruito in “numerocas” un numero casuale (che viene ottenuto facendo la somma delle letture di tutti i sensori).

Terminate tutte le acquisizioni si spegne il convertitore A/D resettando l'opportuno bit di ADCON0. Fatto ciò, se necessario (bit 1 di "step_genera"), si imposta un nuovo numero casuale (procedura "itc") oppure, se sé in modalità di fuga (bit 4 di "decisione"), si imposta come numero casuale un valore predefinito (si è scelto 100 perché questo numero, che è inteso come un numero di passi da fare, permette di fare una distanza di circa 6m) e si indica che è necessario continuare a scappare (bit 0 di "corri").

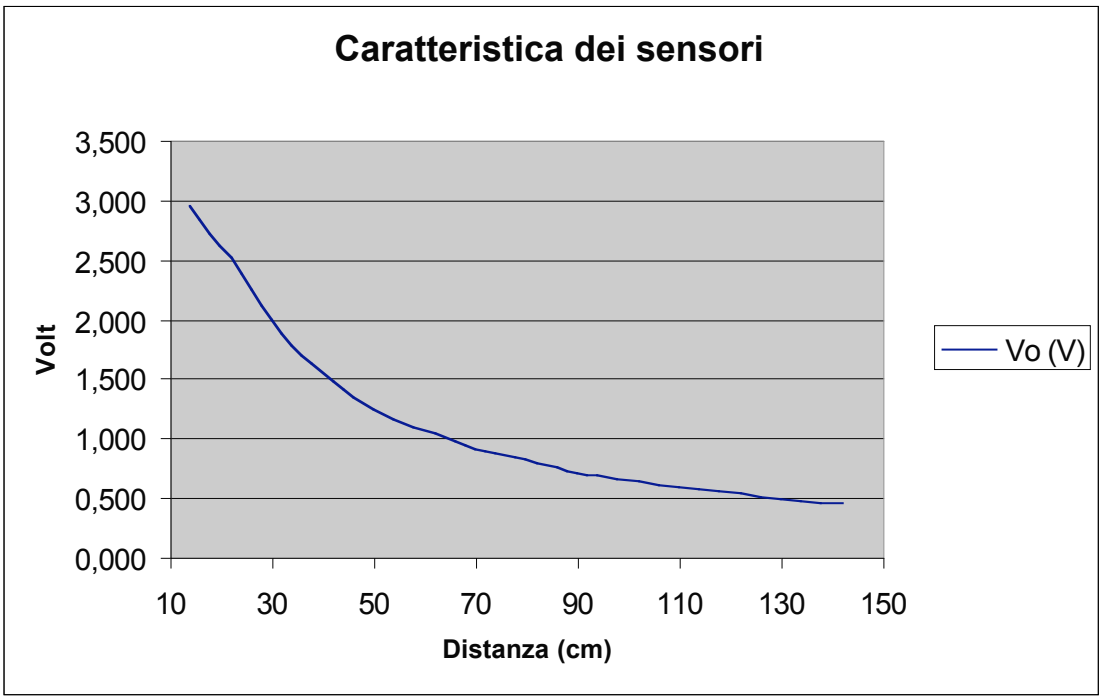


Grafico 1

3.1.3. itc (imposta tempo casuale)

Questa procedura permette di impostare un nuovo numero casuale che può essere utilizzato per generare una attesa casuale oppure un numero casuale di passi da fare.

Il numero casuale è costituito da 13 bit i cui 5 bit più significativi (contenuti in randomH) corrispondono a quelli meno significativi del numero contenuto in "numerocas" (questi 5 bit sono ottenuti mascherando "numerocas"). Gli 8 bit meno significativi (contenuti in randomL) sono posti a 0.

3.1.4. ghost

La procedura ha il compito di rilevare la presenza di un oggetto ad una distanza inferiore ai 50 cm su ognuno dei 5 sensori. Di conseguenza l'ossatura è molto simile a quella di GHOSTDX solo che "j" viene inizializzato a 5 e il registro di flag che utilizziamo è "step_genera"

Il bit 4 di "step_genera" evidenzia il fatto che è successo qualcosa per cui il robot deve spostarsi, mentre il bit 3 informa le procedure seguenti del fatto che bisogna muoversi per la presenza di un ostacolo a una distanza inferiore a 50 cm.

3.1.5. warning

É una procedura che vuole simulare un contatto di qualche oggetto o persona con la sedia.

Dato che non disponevamo di sensori per questo scopo, abbiamo pensato che si possa ritenere "contatto" anche qualcosa che sia avvicini alla sedia al di sotto dei 15cm. La sedia quindi si ferma immediatamente e comincia a "piangere". Per la simulazione del pianto è necessario un buzzer abilitabile con il bit 5 del PORTB. Dato che questo non è stato fisicamente implementato, per ora accendiamo il led rosso della batteria.

La procedura è del tutto simile alle ghost, salvo il fatto della soglia settata ad 85H=15cm ed altre piccole modifiche.

Queste sono il settaggio a 0 di corri(1) che indica che nessuno è troppo vicino e i=5 per scorrere tutti i sensori. Nelle ghost si usavano separatamente quelli del lato destro, sinistro e quello davanti.

Nel "loop12" (identico a quello delle "ghost") setto ad 1 il bit corri(1) se ho trovato qualcosa troppo vicino.

Alla fine della procedura se corri(1)==1 la sedia "piangerebbe" con PORTB(5)==1, ma per ora ci limitiamo, ripetiamo, all'accensione del led rosso della batteria con PORTB(4)==1.

3.1.6. ghost DX

Questa procedura ha la funzione di rilevare la presenza di un oggetto a una distanza inferiore a 82 cm a destra del robot. Questa decisione viene presa facendo la differenza tra l'ultimo valore letto dai sensori di destra con il valore binario (00101000) che corrisponde al valore desiderato. Se viene rilevato un oggetto, la procedura setta il bit 5 del registro "decisione" che ha la funzione di flag per le procedure seguenti.

Nel dettaglio la procedura è composta dalle prime 6 istruzioni di inizializzazione delle variabili ausiliarie: "j" contiene il valore della distanza limite, "i" viene usata come contatore e nel registro FSR viene copiato l'indirizzo di "sensore+3" (73H). Le altre istruzioni invece fanno parte del loop di due, uno per ogni sensore di destra.

La prima istruzione del loop copia il valore del registro INDF in "w", cioè scrive il contenuto di "sensore" nel registro di appoggio interno del microcontrollore così da poter realizzare la differenza tra i registri "j" e "w". Quando c'è un oggetto il risultato è negativo (infatti più un oggetto è vicino, più la tensione di uscita dei sensori è elevata!) e per rilevarlo mascheriamo i bit 0 e 2 del registro STATUS (carry e zero) tramite l'istruzione di and e infine testiamo il flag di zero. Se è a 1 allora la sottrazione è negativa e settiamo il bit 5 di decisione (c'è qualcuno a destra), altrimenti salta questa istruzione e prosegue incrementando il registro FSR che ora conterrà l'indirizzo del secondo sensore di destra. Infine decrementiamo "i" di 1 e quando raggiungerà zero usciamo dal loop.

Le ultime due istruzioni servono a settare anche il bit 4 di "decisione" (presenza di un oggetto) qualora ci fosse qualcuno a destra (bit 5). Questo è necessario per coerenza con la procedura GHOST.

3.1.7. ghost SX

La procedura è identica alla precedente tranne per l'indirizzo dei sensori da confrontare (70H) e il bit di "decisione" da settare (bit 7, c'è qualcuno a SX).

3.1.8. ghost AV

Anche questa procedura è molto simile alle prime due, anche se si differenzia ovviamente nei sensori e il bit di "decisione" settato (bit 6).

Qui il loop non è presente perché il sensore anteriore è unico.

3.1.9. decis

Questa procedura determina la strategia del movimento del robot, semplicemente copiando il contenuto della tabella all'indirizzo "move" negli ultimi 4 bit di "decisione". La decisione viene presa osservando gli ultimi 3 bit MSB di "decisione" che contengono le informazioni sulla presenza di oggetti a destra, sinistra o avanti.
La strategia contenuta nella tabella può essere indirizzata per righe tramite i 3 bit di decisione, semplicemente avendo l'accortezza di ruotarli a destra di 5 posizioni e poi sommargli il valore di "move". Per ruotare "decisione", utilizziamo il registro appoggio "i" e infine sommiamo il contenuto della tabella proprio a "decisione". Le ultime 4 istruzioni settano anche i bit 5 e 6 di "step_genera" nel caso in cui la strategia preveda di andare rispettivamente a sinistra e destra. Questi bit servono per il movimento continuo del motore interessato.

Vettore di bit "Decisione"									
spiazzamento	Bit più significativi			direzioni da prendere	Bit meno significativi				indirizzo
	7	6	5		3	2	1	0	
0	0	0	0	avanti (non vede niente)	1	0	0	0	40H
1	0	0	1	sx	0	1	0	0	41H
2	0	1	0	dx (per default)	0	0	0	1	42H
3	0	1	1	sx	0	1	0	0	43H
4	1	0	0	dx	0	0	0	1	44H
5	1	0	1	avanti	0	0	1	0	45H
6	1	1	0	dx	0	0	0	1	46H
7	1	1	1	dx (cerca di girarsi)	0	0	0	1	47H

Tabella 1

3.1.10. motori

Questa procedura è quella che si preoccupa di azionare nel modo corretto il motore giusto. Infatti valutando la decisione di movimento che è stata presa (bit 3, 2, 1 e 0 di "decisione") richiama le procedure adeguate per far andare la sedia verso destra (procedura "mcontSX"), verso sinistra (procedura "mcontDX") o avanti (procedura "avanti").

3.1.11. mcontDX/SX

Abbiamo notato con prove pratiche sulla sedia che se deve continuare a girare da un lato per evitare ostacoli, è meglio tenere alto il comando al motore piuttosto che dargli più volte singoli impulsi, perché così non si corre il rischio di perdere qualche comando.

Con questa idea abbiamo creato due procedure simili che tengono alto l'impulso del motore per tutto il tempo necessario alla variazione di direzione della sedia. Si chiamano "mcontDX" e "mcontSX" perché appunto il comando al motore è continuo.

Anche qui come nella procedura "avanti" prima di muovere un passo devo vedere da che situazione arrivo: se sono nella procedura mcontDX devo assicurarmi di fornire un tempo di sicurezza per far appoggiare l'altra gamba se vengo da un movimento continuo del motore sinistro. Lo controllo col bit 6 di step_genera. Dualmente in mcontSX controllerò step_genera(5).

Una volta essermi assicurato che la sedia non cadrà, passo al movimento vero e proprio con normDX e normSX dove rispettivamente accendo (e li lascio accesi) il motore DX (PORTB(6)) ed il motore SX (PORTB(7)) e setto i bit 7 e 0 di step_genera per sapere che motore ho mosso.

Provvede il "main" (fuori dal ciclo "passi" –altrimenti non ci sarebbe un movimento continuo del motore) a fare il clear dei bit 6 e 7 del PORTB.

Per sicurezza abbiamo messo un ulteriore ritardo di 1,7s che svolge due funzioni: la prima di fornire il tempo minimo per il movimento di almeno un passo, la seconda è quella di dare la sicurezza che la sedia non cada nel caso che nell'esecuzione del programma subito dopo questa procedura venga dato un comando di muovere l'altra gamba.

3.1.12. avanti

Procedura creata per il movimento in avanti dritto, quando cioè non trova ostacoli. Al suo interno vengono inoltre impostate le modalità lavanti lento (normale) od avanti veloce (scappa) in base al bit di flag "decisione(3)".

L'idea è quella di far compiere alla sedia correttamente il passo giusto in base al precedente, e per questo si necessita del flag "step_genera(0)" che mi dice l'ultimo passo compiuto.

La procedura inizia col controllo della situazione: se provengo da un movimento continuo del motore (destro o sinistro) devo, oltre che settare i flag, attendere un tempo di sicurezza (ciclo "att" da 1,6 secondi) per avere la certezza che almeno una gamba sia in appoggio.

Quindi la procedura esegue un test sui bit 5 e 6 di step_genera: se uno dei due è settato vuol dire che provengo da un movimento continuo e chiamo "esciDX" od "esciSX" a seconda della necessità, oppure vado a "norm" se sono in condizione normale e non mi serve l'attesa del tempo di sicurezza.

Le etichette "DX" e "SX" indicano la direzione che si è deciso di prendere in base a step_genera(0) che ricordiamo è il flag dell'ultimo passo compiuto; se è 1 vuol dire che per ultimo ho mosso il motore di destra, cioè stava andando a SX, e quindi ora chiamerò DX.

DX e SX oltre ad abilitare le uscite 6 e 7 del PORTB, settano e puliscono i bit 0 e 7 di step_genera che indicano quale motore è stato mosso.

Prima di togliere il segnale ai motori attendo 500ms ed infine per semplicità pulisco entrambi i bit 6 e 7 del PORTB anche se uno dei due in realtà sarebbe già a 0 (vedi "main").

E' stato poi deciso che la sedia continuasse ad avere un passo veloce per un certo numero di passi (che si contano nel main decrementando RandomH) anche nel movimento avanti dritto per finire di scappare.

Quindi in questa procedura vado a leggere il contenuto del flag corri(0), settato in altre parti del programma, e se vale 1 (cioè devo correre per finire di scappare, non vado lento), metto a 0 decisione(3) (= Avanti Veloce). In base a questo flag il ritardo tra un passo e l'altro è di 1,1s per il passo veloce o 2,5s per quello lento.

3.1.13. pausa

E' la procedura che introduce un "ritardo base" in quanto per diminuire il ritardo basta modificare "y" secondo la relazione: $t_{delay}=1.28*(255-y)$ ms. All'inizio pulisco y1 (variabile d'appoggio interna); in seguito mettendo y=177 (Decimale) si ottiene in totale circa 100ms di pausa. Il ciclo termina quando y supera il valore di 255: ecco perché incrementando y il ritardo diminuisce (raggiungo più velocemente il valore 255).

3.1.14. batt

Questa procedura utilizza il convertitore A/D per valutare il livello di tensione della batteria. La struttura dell'acquisizione è simile a quella descritta nella procedura "acquis" con la differenza che si deve indirizzare il canale 1 del porto A su cui è presente il segnale di tensione della batteria (opportunamente scalato tramite un partitore di tensione) e che va fatta una singola acquisizione. Il numero contenuto in ADRESH va confrontato con il valore di soglia presente nella variabile "batteria" (il valore preimpostato è A3h che corrisponde ad una tensione di ingresso di 10V). Il confronto è effettuato facendo una sottrazione tra il valore di soglia e quello effettivo. Se il risultato è minore o uguale a zero (per fare questa verifica si utilizzano i flag Z e C del registro STATUS (vedi il datasheet del PIC16F876)) significa che il livello di tensione della batteria è inferiore a quello della soglia e quindi viene data una segnalazione mettendo a 1 il canale numero 4 del porto B a cui è collegato un led rosso.

3.1.15. interrupt

Il nostro programma sfrutta alcuni interrupt interni generati dai registri di TIMER quando hanno raggiunto il loro valore massimo.
Nei microcontrollori della Microchip, quando occorre un interrupt (se sono attivati), il program counter viene salvato nello stack, successivamente viene impostato all'indirizzo 04H e vengono disattivati tutti gli altri interrupt. Da questo indirizzo la gestione di quale interruzione è avvenuta deve essere fatta tramite polling dei flag degli interrupt attivati. Una volta determinato cosa è successo, il flag corrispondente deve essere azzerato via software dell'utente.

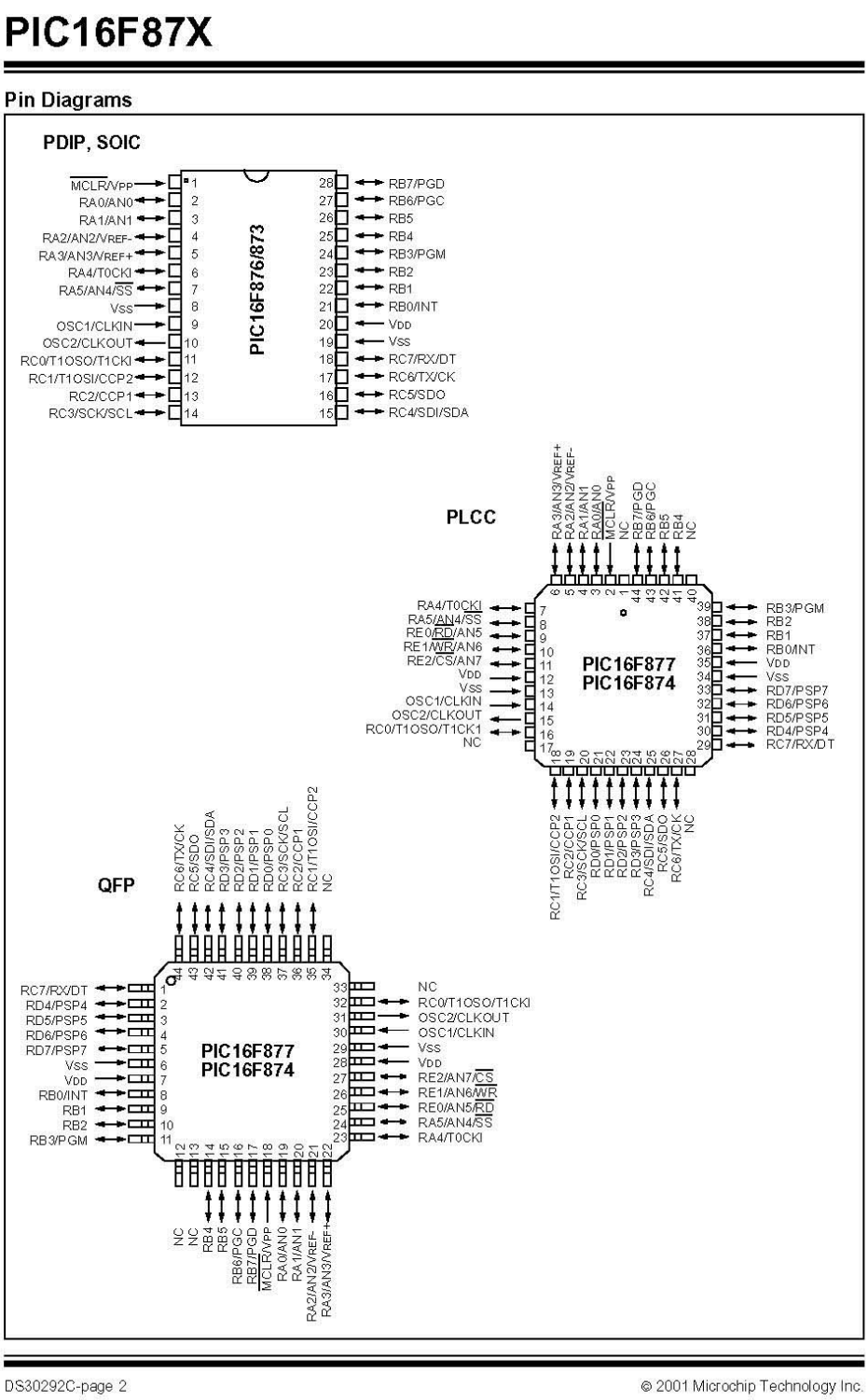
Nel dettaglio utilizziamo l'interrupt del Timer1 (registro a 16 bit) per aspettare la fine del tempo casuale, gestito nella parte di codice dopo l'etichetta 'rand' (bit 0 del registro PIR1). In questa parte di codice aggiorniamo "randomH" e "randomL" che servono come registri di appoggio per il conteggio del numero casuale.

Utilizziamo il ritardo generato dal TIMER2 (registro a 8 bit) come base dei tempi per controllare la tensione sulla batteria (bit 1 del registro PIR1), infatti in questa parte di codice c'è la chiamata alla procedura "batt" che acquisisce e gestisce il led di segnalazione.

3.2. Listato del programma

Il listato del programma si può trovare sul CD nel file “listato.txt” e si può consultare direttamente tramite il link seguente: listato.txt .

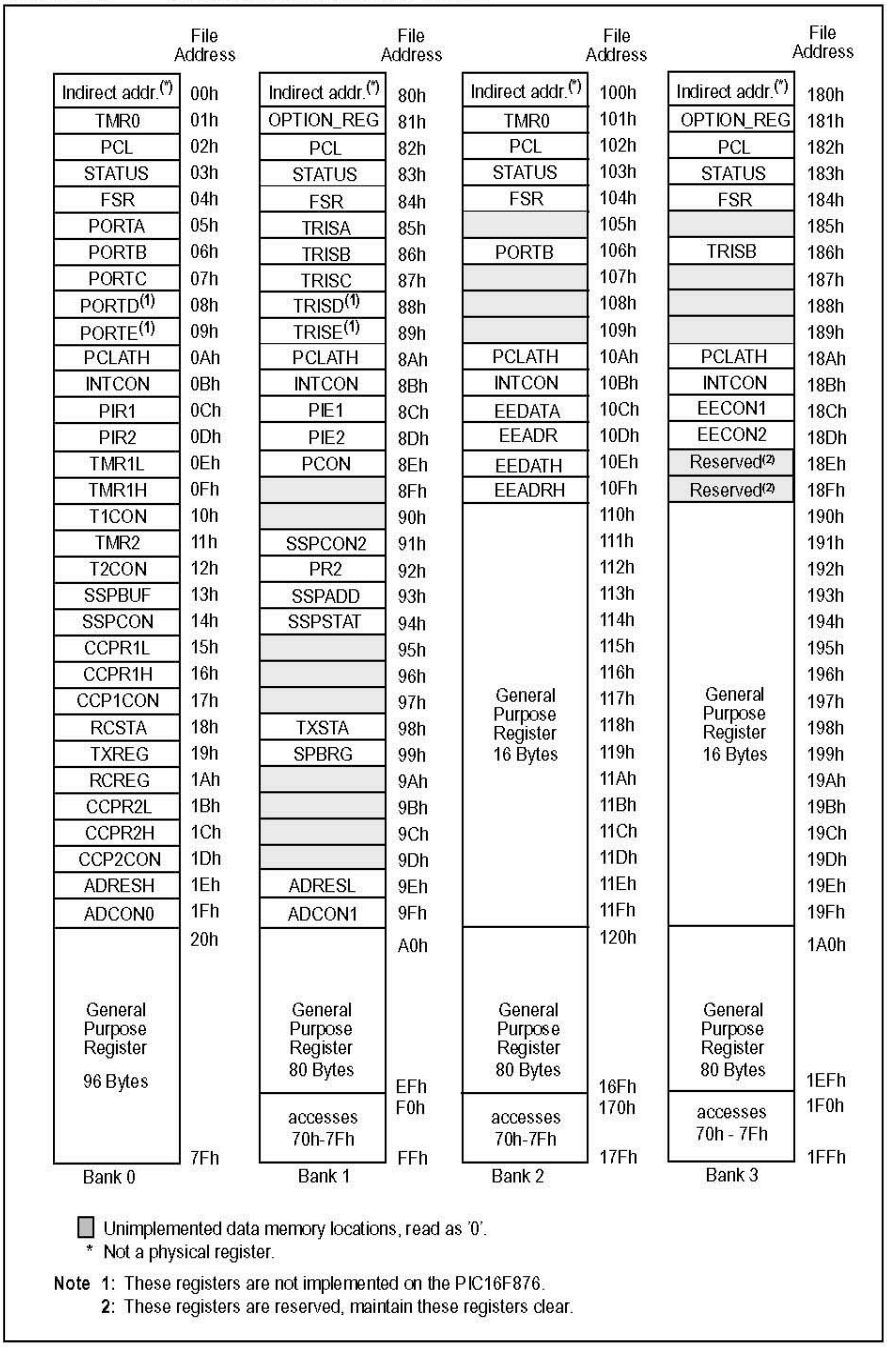
3.3. Estratti dal datasheet PIC16F876



DS30292C-page 2 © 2001 Microchip Technology Inc.

Figura 3

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP



Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

Figura 4

2.2.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 2-1.

The Special Function Registers can be classified into two sets: core (CPU) and peripheral. Those registers associated with the core functions are described in detail in this section. Those related to the operation of the peripheral features are described in detail in the peripheral features section.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
01h	TMR0	Timer0 Module Register								xxxx xxxx	47
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	29
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	33
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	35
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	36
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
0Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	22
0Dh	PIR2	—	(5)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0	24
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	52
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	51
11h	TMR2	Timer2 Module Register								0000 0000	55
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	55
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	70, 73
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	67
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx	57
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx	57
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	58
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	96
19h	TXREG	USART Transmit Data Register								0000 0000	99
1Ah	RCREG	USART Receive Data Register								0000 0000	101
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx	57
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx	57
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	58
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	116
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	111

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

Figura 5

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	29
86h	TRISB	PORTB Data Direction Register								1111 1111	31
87h	TRISC	PORTC Data Direction Register								1111 1111	33
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	35
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	37
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
8Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	21
8Dh	PIE2	—	(5)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0	23
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	25
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	68
92h	PR2	Timer2 Period Register								1111 1111	55
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	73, 74
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	66
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	95
99h	SPBRG	Baud Rate Generator Register								0000 0000	97
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	—	Unimplemented								—	—
9Dh	—	Unimplemented								—	—
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	116
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000	112

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

Figura 6

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
101h	TMR0	Timer0 Module Register								xxxx xxxx	47
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	26
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	31
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
10Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	41
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	41
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						xxxx xxxx	41
10Fh	EEADRH	—	—	—	EEPROM Address Register High Byte					xxxx xxxx	41
Bank 3											
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	27
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	19
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	26
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	18
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	27
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	31
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	26
18Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	20
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	41, 42
18Dh	EECON2	EEPROM Control Register2 (not a physical register)								---- ----	41
18Eh	—	Reserved maintain clear								0000 0000	—
18Fh	—	Reserved maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.

5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

Figura 7

2.2.2.1 STATUS Register

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bits for data memory.

The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable, therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u uluu` (where `u` = unchanged).

It is recommended, therefore, that only `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions are used to alter the STATUS register, because these instructions do not affect the Z, C or DC bits from the STATUS register. For other instructions not affecting any status bits, see the "Instruction Set Summary."

Note:	The C and DC bits operate as a $\overline{borrow}$ and digit $\overline{borrow}$ bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP**: Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 **$\overline{TO}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 8

2.2.2.2 OPTION_REG Register

The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler/WDT postscaler (single assignable register known also as the prescaler), the External INT Interrupt, TMR0 and the weak pull-ups on PORTB.

Note: To achieve a 1:1 prescaler assignment for the TMR0 register, assign the prescaler to the Watchdog Timer.

REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: When using low voltage ICSP programming (LVP) and the pull-ups on PORTB are enabled, bit 3 in the TRISB register must be cleared to disable the pull-up on RB3 and ensure the proper operation of the device

Figura 9

2.2.2.3 INTCON Register

The INTCON Register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow, RB Port change and External RB0/INT pin interrupts.

Note:	Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE**: Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 10

2.2.2.4 PIE1 Register

The PIE1 register contains the individual enable bits for the peripheral interrupts.

Note: Bit PEIE (INTCON<6>) must be set to enable any peripheral interrupt.

REGISTER 2-4: PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE(1)**: Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Note 1: PSPIE is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 11

2.2.2.5 PIR1 Register

The PIR1 register contains the individual flag bits for the peripheral interrupts.

Note:	Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt bits are clear prior to enabling an interrupt.

REGISTER 2-5: PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF(1)**: Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag
1 = The SSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI
  - A transmission/reception has taken place.
- I^2C Slave
  - A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated START condition was completed by the SSP module.
  - The initiated STOP condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A START condition occurred while the SSP module was idle (Multi-Master system).
  - A STOP condition occurred while the SSP module was idle (Multi-Master system).

0 = No SSP interrupt condition has occurred.

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: PSPIF is reserved on PIC16F873/876 devices; always maintain this bit clear.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 12

2.5 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

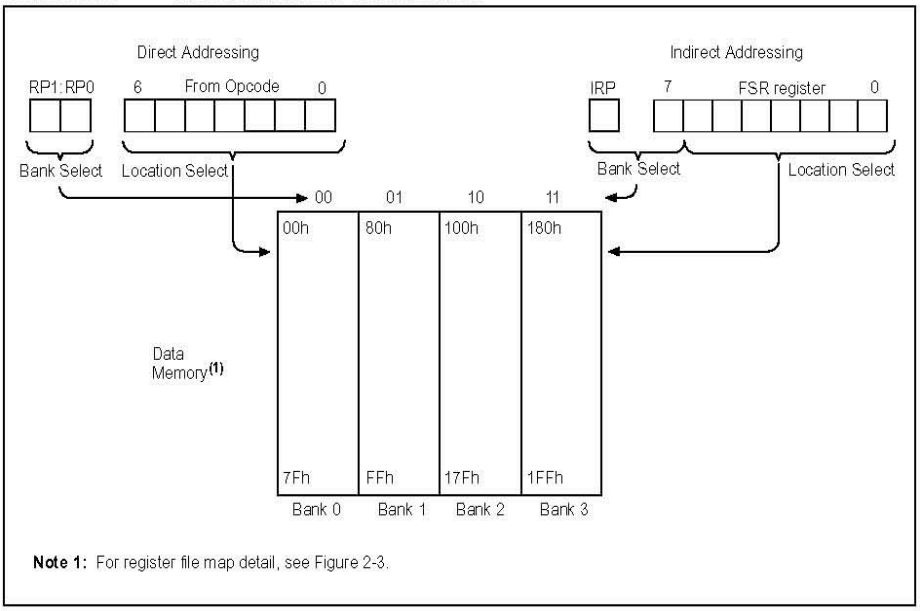
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (STATUS<7>), as shown in Figure 2-6.

A simple program to clear RAM locations 20h-2Fh using indirect addressing is shown in Example 2-2.

EXAMPLE 2-2: INDIRECT ADDRESSING

```
        MOVLW  0x20    ;initialize pointer
        MOVWF  FSR     ;to RAM
NEXT    CLRF   INDF    ;clear INDF register
        INCF   FSR,F   ;inc pointer
        BTFSS  FSR,4   ;all done?
        GOTO   NEXT    ;no clear next
CONTINUE
        :              ;yes continue
```

FIGURE 2-6: DIRECT/INDIRECT ADDRESSING



Note 1: For register file map detail, see Figure 2-3.

Figura 13

11.0 ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) Converter module has five inputs for the 28-pin devices and eight for the other devices.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. The A/D conversion of the analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low voltage reference input that is software selectable to some combination of VDD, VSS, RA2, or RA3.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D clock must be derived from the A/D's internal RC oscillator.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register, shown in Register 11-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 11-2, configures the functions of the port pins. The port pins can be configured as analog inputs (RA3 can also be the voltage reference), or as digital I/O.

Additional information on using the A/D module can be found in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

REGISTER 11-1: ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0**: Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$**: A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented**: Read as '0'

bit 0 **ADON**: A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 14

REGISTER 11-2: ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.
2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

The ADRESH:ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair, the GO/$\overline{\text{DONE}}$ bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set. The block diagram of the A/D module is shown in Figure 11-1.

After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs.

To determine sample time, see Section 11.1. After this acquisition time has elapsed, the A/D conversion can be started.

Figura 15

TABLE 13-2: PIC16F87X INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from interrupt	2	00 0000 0000 1001			
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into standby mode	1	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk		Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Note: Additional information on the mid-range instruction set is available in the PICmicro™ Mid-Range MCU Family Reference Manual (DS33023).

Figura 16

4. Modalità operative

Per programmare il PIC è necessario un circuito ed un software (ICprog) dedicato scaricabile via internet all'indirizzo www.icprog.com .

Un esempio di circuito elettrico programmatore (da noi utilizzato) è riportato in figura con il dimensionamento dei componenti:

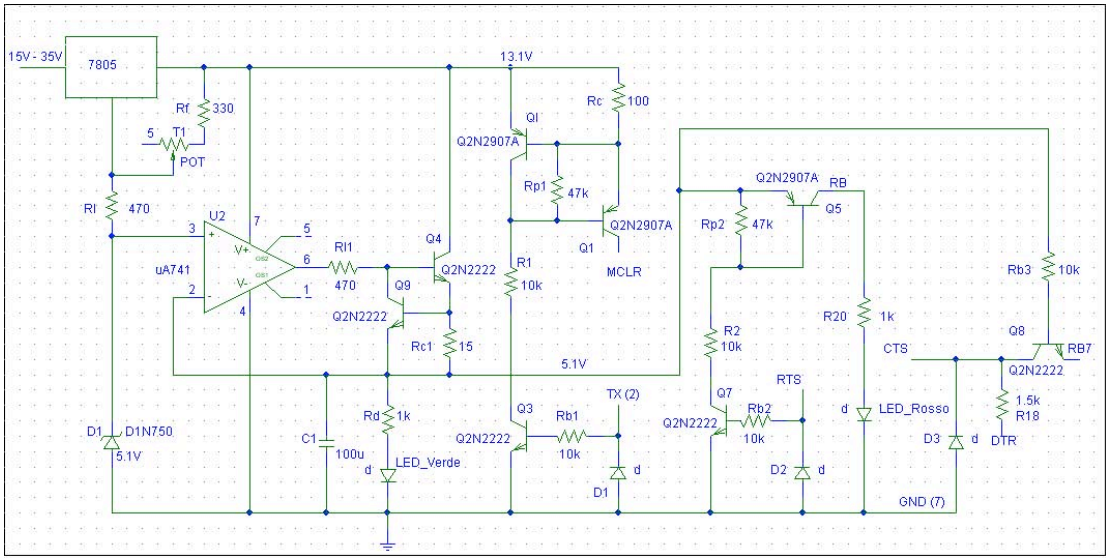


Figura 17

4.1. Componenti necessari

La realizzazione della scheda comporta l'acquisto di pochi componenti, piuttosto comuni e quindi reperibili in qualunque negozio di elettronica di consumo. Sono necessari: un microcontrollore della Microchip PIC16F876, un multiplexer analogico Maxim DG408DJ, un 7805, due transistor NPN (2n2222) e due PNP (2n2907) (questi BJT sono sovradimensionati ma erano disponibili in laboratorio), un quarzo da 4 MHz, led, resistori e condensatori vari (vedi schema elettrico).

Tutti i componenti sono reperibili da RS componenti, tranne i sensori di distanza GP2Y0A02YK acquistati sul sito www.acroname.com.

Ecco lo schema della scheda:

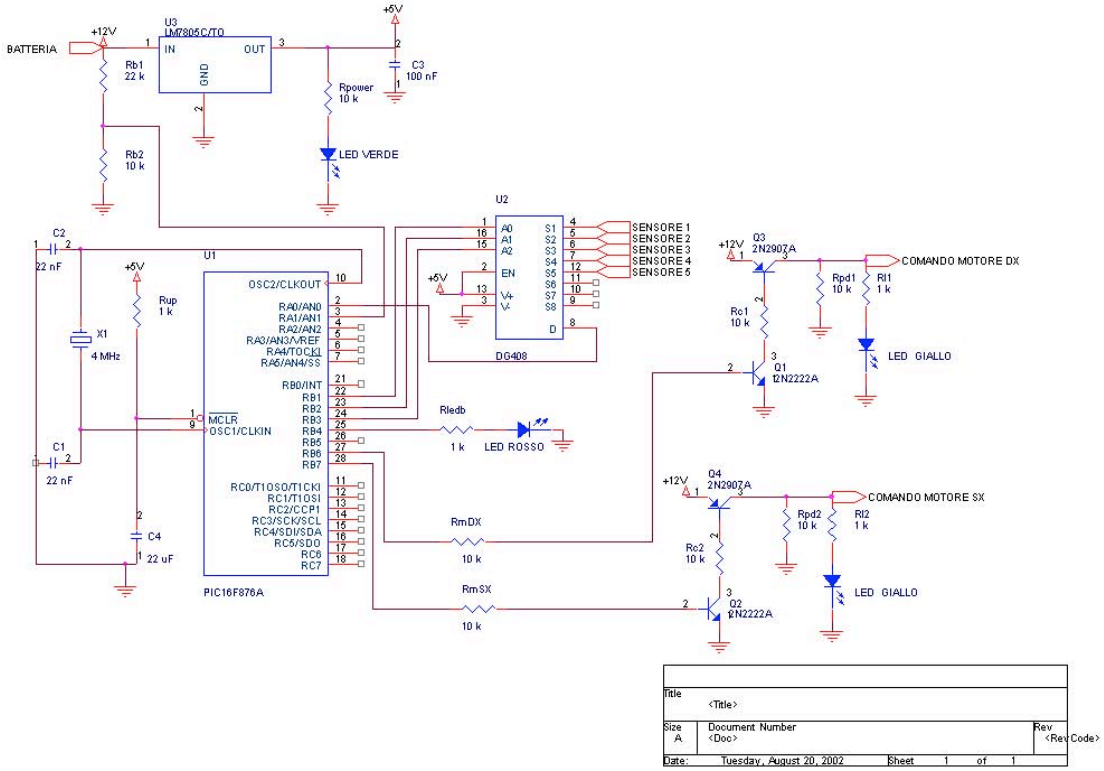


Figura 18

4.2. Modalità di installazione

Abbiamo raccolto tutti i segnali e le alimentazioni in un unico connettore la cui piedinatura è la seguente:

Connettore std 26 poli	
Segnali	*Piedini*
GND	1-2
sensore 1	3
sensore 2	4
sensore 3	5
sensore 4	6
sensore 5	7
NC	8
+5 V	9-10
NC	11
NC	12
Motore SX	13-14
Motore DX	15-16
+12 V	17-18-19-20
NC	21
NC	22
GND	23-24-25-26

Tabella 2

La disposizione dei sensori deve essere come quella riportata in figura 17 che mostra la sedia vista dall'alto (la freccia indica la direzione del moto in avanti):

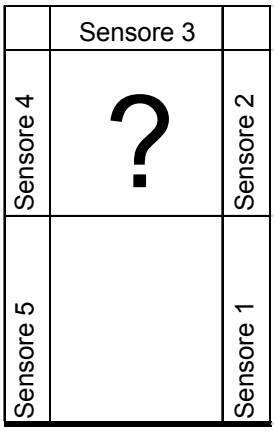


Figura 19

5. Conclusioni e sviluppi futuri

Come prototipo dimostra bene le funzioni che sono state richieste e l'effetto visivo è notevole; uno sviluppo futuro può essere quello di creare più sedie e cercare di far condividere loro uno stesso spazio d'azione.

Il programma nella versione finale è piuttosto semplice sia nel comportamento sia nella gestione dei segnali esterni; sarebbe possibile migliorarlo scrivendo apposite routine più specifiche per il comportamento e per l'acquisizione più precisa dei valori dei sensori.

Inoltre abbiamo notato che la sedia funziona bene su superfici ruvide (moquette,tappeti,ecc…), mentre ha delle difficoltà su superfici lisce: questo perché le ruote applicate alle gambe posteriori scivolano troppo: una miglioria sarebbe il montaggio di ruote monodirezionali.

Bibliografia

Tutta la documentazione necessaria al progetto è reperibile sui datasheet dei vari componenti che si possono trovare sui siti internet sotto elencati (reperibili anche sul CD allegato):

[1] Microchip: “PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers”, www.microchip.com .

[2] Microchip: “PICmicro™ Mid-Range MCU Family Reference Manual”, www.microchip.com .

[3] Maxim: “Improved 8-Channel,CMOS Analog Multiplexer”, www.maxim-ic.com .

[4] Sharp: ”GP2Y0A02YK Long Distance Measuring Sensor”, www.sharpsma.com

Indice

SOMMARIO 1

1. INTRODUZIONE 1

2. IL PROBLEMA AFFRONTATO 1

3. LA SOLUZIONE ADOTTATA 2
3.1. Programma del PIC 3
3.1.1. main 4
3.1.2. acquis 5
3.1.3. itc (imposta tempo casuale) 6
3.1.4. ghost 6
3.1.5. warning 7
3.1.6. ghost DX 7
3.1.7. ghost SX 7
3.1.8. ghost AV 8
3.1.9. decis 8
3.1.10. motori 8
3.1.11. mcontDX/SX 8
3.1.12. avanti 9
3.1.13. pausa 10
3.1.14. batt 10
3.1.15. interrupt 10
3.2. Listato del programma 11
3.3. Estratti dal datasheet PIC16F876 12

4. MODALITÀ OPERATIVE 26
4.1. Componenti necessari 26
4.2. Modalità di installazione 27

5. CONCLUSIONI E SVILUPPI FUTURI 28

BIBLIOGRAFIA 29

INDICE 30