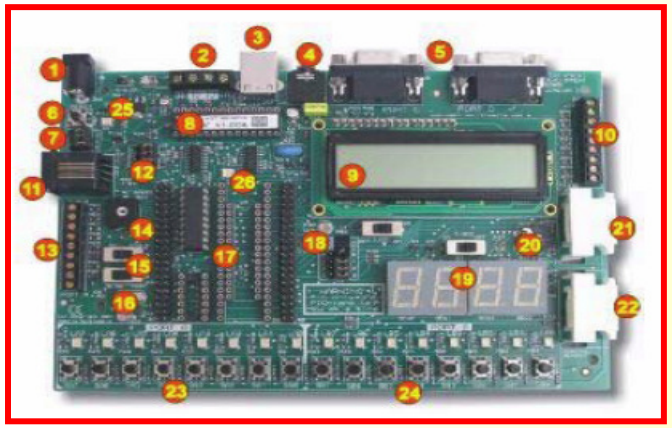
Università degli studi di Cassino

Corso di Laurea in
Ingegneria della Produzione Industriale



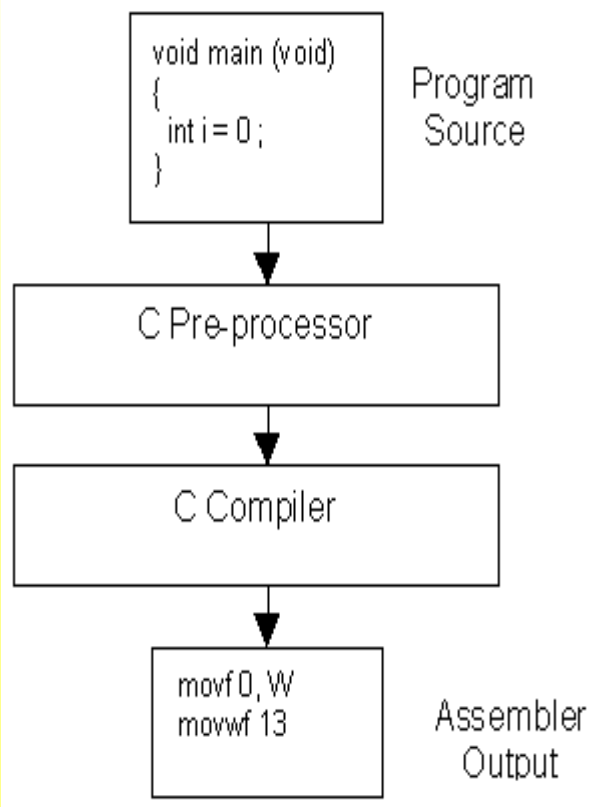
Corso di Informatica Applicata

Lezione 6

Ing. Saverio De Vito
e-mail: saverio.devito@portici.enea.it
Tel.: +39 081 7723364

Linguaggio C - Riepilogo

- Struttura dei programmi C
- Modello di programmazione
- Variabili e tipi semplici (int, char, float[x])
- Esecuzione condizionale (solo if-then-else)
- Iterazione (while, do...while, for)
- Funzioni



Arrays e Puntatori

- Tipicamente un argomento ostico per chi inizia a programmare in C..... quindi Attenzione!
- Strumenti estremamente potenti, ma come tali, uniti alla sintassi C, forieri di bug difficilmente individuabili...
- Appreso quest' argomenti saremo... ½ di programmatore C, :-)

Ma prima....

- Un altro costrutto di esecuzione condizionale:

```
switch ( control variable )
{
case constant1 :
/* code for this value */
break ;
case constant2 :
case constant3 :
/* code for this value */
break ;
default :
/* code for no matches  */
break ;
}
```

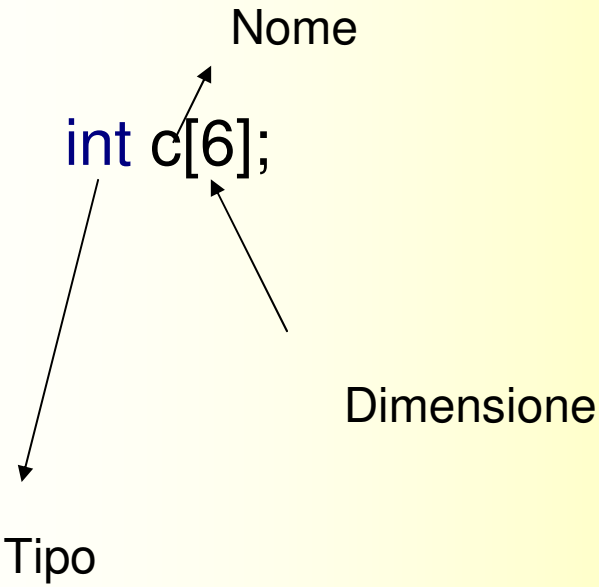
Permette l' esecuzione condizionale di più blocchi basata sul valore assunto da una variabile di controllo, prevede un valore di default.
Ogni case viene attivato se la variabile di controllo assume il valore costante dichiarato.
La parola chiave break permette di interrompere il flusso di esecuzione rimandando alla fine del costrutto switch.

Arrays

- Può presentarsi la necessità di adoperare strutture dati che possano rappresentare insiemi di variabili omogenee in tipo con cardinalità fissata. Es:
  - Studenti di questo corso (matricole)
  - Indice di un libro (argomento, numero di pagina)
  - Porte di I/O di un' architettura a microprocessore

Arrays

- Il C prevede una modalità di dichiarazione che permette di costituire un array di elementi indicizzati:



Dichiariamo un array di 6 elementi di tipo intero

Arrays

- Come 'indicizzare' l' n-esimo valore contenuto nell' array..... ossia come ne leggiamo/scriviamo il contenuto?

- Il C ci fornisce una sintassi per riferirci all' n-esimo valore contenuto nell' array:

```
c[4]=2;  /*l' indice più basso previsto è 0, corrispondente */
         /*al primo elemento dell' array                    */
```

Attenzione: non è previsto in C un controllo esplicito sulla lunghezza di un array, è quindi possibile indicizzare valori posti fuori dai limiti dichiarati per la lunghezza di un array... c[8]=7 ..... nella maggior parte dei casi non riceverete alcun errore

Arrays

- E' possibile ovviamente dichiarare array di differenti tipi:

```
char pippo[5];
    pippo[0]='b';
    pippo[1]='a';
    pippo[2]='u';
    pippo[3]='d';
    pippo[4]='o';
float purpo[6];
```

Arrays

A volte può capitare la necessità di dover lavorare con struttre pluridimensionali, ad esempio matrici:

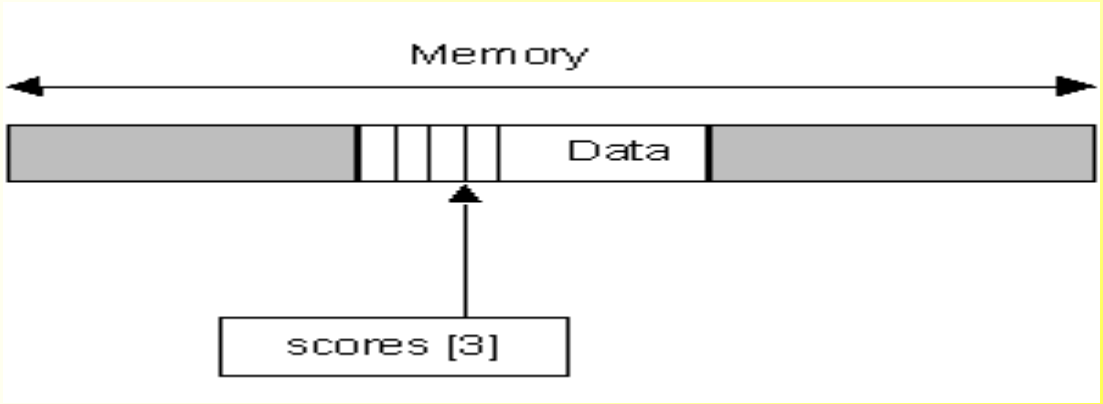
In C è possibile dichiarare e lavorare con array pluridimensionali utilizzando la sintassi:

```
 int matrix [3][4]; /* matrice 3x4 */

matrix[1][0]=matrix[2][3]+2;
```

Puntatori (non supportati da C2C)

I Puntatori e la loro aritmetica, in pratica il loro utilizzo sono poco distanti dall' utilizzo degli array.
In pratica, quando utilizziamo un indice in un array per riferirci all' i-esimo elemento dell' array non facciamo altro che dichiarare al compilatore di voler lavorare con la locazione di memoria posta i x (k) bytes distante dalla prima, dove k è naturalmente lo spazio occupato dal tipo base dell' array.
Nel caso di un array di tipo char, la notazione c[3] fa riferimento al quarto byte contando dalla locazione di memoria occupata dal primo elemento (c[0])



Puntatori

I puntatori ci permettono di estendere il discorso del riferimento diretto alle celle di memoria per qualsiasi tipo anche non organizzato come array.
Un puntatore di fatto è un intero che descrive l' indirizzo di una locazione di memoria.
Il C ci permette di dichiarare un puntatore ad una variabile di un particolare tipo come:

int ***c;** /*puntatore a variabile di tipo intero */

char ***p;** / puntatore a variabile di tipo carattere */

Il C implementa gli array come blocchi di memoria che partono da un indirizzo base. L' indice di un array rappresenta il *displacement* (spiazzamento) dalla locazione base nomearray che permette di identificare un elemento di un array.

Pippo[4] (pippo array di interi) : Il contenuto della decima (5*2bytes) locazione di memoria a partire dall' indirizzo base. Pippo[0], il contenuto della locazione base.

Aritmetica Puntatori

int ***c;** /*puntatore a variabile di tipo intero */

Se c è un puntatore ad una variabile di tipo intero, è possibile aggiungere un displacement i a c per andare a riferirsi all' i-esima locazione contenente un intero a partire da c.

c+5 è l' indirizzo della 5 locazione contenente un intero a partire dall' indirizzo c. Capite come sia fondamentale l' indicazione del tipo del puntatore, al fine di effettuare correttamente il calcolo della locazione di memoria effetiva (in bytes).

*(c+5) [Operazione di 'dereferenziazione'] rappresenta il contenuto della locazione suddetta.

Aritmetica dei puntatori

In effetti se:

```
int goofy[7];

goofy[0]=3;

if  (*goofy==3) { /*Verrò eseguito */ };
```

se scriviamo `*(goofy+5)=6` allora goofy[5] sarà uguale a 6

Quando dichiariamo un array, appunto, il nome dell' array è il puntatore alla locazione base, l' operazione di indicizzazione è di fatto un operazione di dereferenziazione guidata.

L' operatore & fornisce un metodo per ottenere il puntatore ad una data variabile. `&Target` è il valore del puntatore alla variabile Target.

Aritmetica dei puntatori

Attenzione, i puntatori vanno trattati con molta attenzione, ricordando le distinzioni che la sintassi rende difficili da individuare:

```
/*  compare two pointers  */
if ( ptr1 == ptr2 )

and

/* compare what they point at  */
if ( *ptr1 == *ptr2 )
```

Se due puntatori sono uguali (caso 1) ciò che succede alla locazione di memoria cui puntano si rifletterà, come ovvio su entrambi:
se assegno 2 a *ptr1, anche *ptr2 assumerà il valore 2, perchè entrambi (ptr1 e ptr2) puntano alla stessa locazione di memoria

Il Puntatore nullo

Anche se può sembrare strano, esistono casi in cui è utile avere un puntatore che punta ad una locazione inesistente, per meglio dire in nessun posto.

Il valore che questo puntatore assume è detto NULL ed è un valore predefinito.

If (c==NULL) {} else {}

Puntatori e funzioni

Abbiamo visto come in C, per gli argomenti delle funzioni sia previsto il solo passaggio per valore, ciò significa in pratica che non è possibile per una funzione cambiare il valore di una variabile che è nello scope del solo blocco chiamante. Le funzioni in C, hanno solo "parametri di ingresso" ad eccezioine ovviamente del valore che ritornano (return).

Un modo ("il" modo) per bypassare questa limitazione, è il passaggio come parametri, dei puntatori alle variabili il cui valore si intende modificare. Essi verranno copiati, come accase per ogni parametro di una funzione C nello scope della funzione, ma la locazioie di memoria da essi puntata sarà ovviamente la stessa (il loro valore non cambia).

```
/*funzione swapper */
void    swap(int  *a,  int
*b)
{
  int c;
  c=*b;
  *b=*a;
  *a=c;
}
```

Stringhe

In C standard ANSi sono disponibili numerose librerie (standardizzate) per il trattamento di sequenze di caratteri terminate denominate stringhe. Tramite queste librerie è possibile effettuare diverse operazioni (ad es.: estrazioni di sottostringhe) su stringhe. In C2C non sono disponibili. E' possibile però dichiarare array costanti di caratteri dotati di carattere di terminazione ed utilizzarli esattamente come stringhe.

***const* char *p =”Hello”**

si può poi operare con questa stringa alla stregua di un array

char c;
c=p[0]; /* il contenuto della variabile c è ora 'H' */
.....infatti.....

p[0]='H'
p[1]='e'
p[2]='l'
p[3]='l'
p[4]='o'
p[5]=0x00 <- terminatore di stringa

questo modo di operare è utile per la costruzione di messaggi per l' output su particolari dispositivi di I/O a caratteri, ad esempio il display della scheda HP488 (tesina).

Stringhe

Ad esempio supponiamo di disporre di una funzione in grado di visualizzare sul display una stringa qualsiasi di opportuna lunghezza:

void display(const char* message);

sarà possibile dichiarando una stringa qualsiasi:

const char *p=”Messaggio”;

richiamare la funzione passando come parametro la stringa dichiarata per ottenerne la visualizzazione:

display(p);

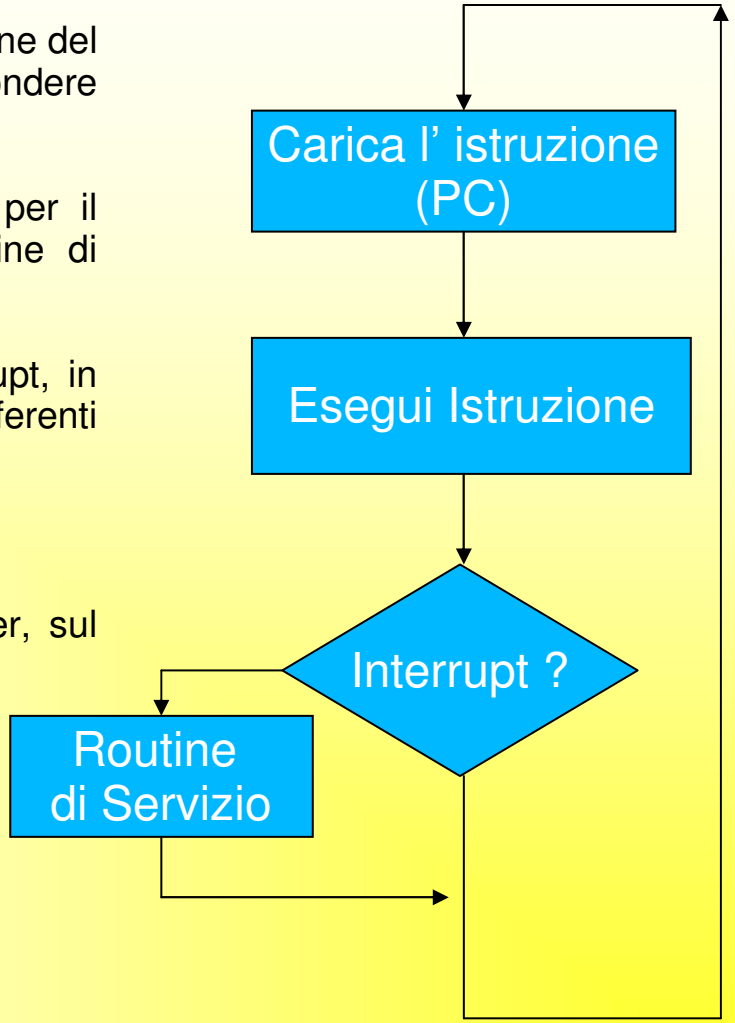
Recall: Interrupts

Abbiamo già visto gli interrupts come modalità di interruzione del flusso primario di esecuzione per risolvere problemi o rispondere a stimoli impellenti.

Abbiamo introdotti i vettori di interrupt come modalità per il processore di identificare il tipo di interrupt e la routine di servizio del particolare interrupt.

Il PIC16F84A è disegnato per possedere un solo interrupt, in compenso l' interrupt può essere generato in alcuni differenti modi:

- External interrupt signal (on PORTB bit 0)
- Timer Overflow - (Il pic è dotato di un registro timer, sul rollover viene generato un segnale di interrupt)
- Change on inputs (on PORTB bits 4-7)
- EEPROM write complete (ne riparleremo a breve)



Recall: Interrupts

Il PIC 16F84A è dotato di un registro speciale denominato INTCONT mediante il quale il programmatore è in grado di controllare l' attivazione delle routine di interrupt da parte dei singoli segnali che la potrebbero attivare. In pratica è possibile mascherare (disabilitare) un interrupt, ossia fare in modo che un segnale noto non “triggeri” un interrupt.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

GIE: Porre a 0 questo bit significa rendere insensibileil PIC agli interrupt

EEIE: Porre a = questo bit significa rendere il PIC insensibile all' interrupt dovuto al completamento di operazioni EEPROM (I/=)

TOIE: Abilita/Disabilita l' interrupt generato dal wrap around del timer (255->0)

RBIE: Abilita/Disabilita l' interrupt generato dal cambiamento negli input 0-4

Routine di Interrupt in C2C

void interrupt (void)

è la signature predefinita della function rappresentante la routine di servizio dell' interrupt nel C del nostro PIC16F84A.
All' interno della routine, tipicamente, è incluso un costrutto di esecuzione condizionale preposto alla discriminazione, mediante il test degli opportuni bit meno significativi del registro INTCON, della tipologia di segnale che ha causato l' invocazione della routine di interrupt stessa.

I/O : Input and Output

L' I/O rappresenta l' insieme delle modalità di comunicazione di un microprocessore con l' ambiente esterno caratterizzato dalle sue periferiche.

Es.:

Scrivere un dato su un disco o su una qualsiasi memoria secondaria

Ricevere un valore carattere di input da una tastiera

Campionare un valore acquisito da un sensore

Scrivere un carattere su un display LCD

Attivare il blocco o lo sblocco di una porta

Le modalità di comunicazione differiscono notevolmente tra le architetture, alcune di queste prevedono la gestione dell' I/O con diverse modalità.

I/O : Input and Output

Come si indirizza una periferica di I/0? Sostanzialmente le soluzioni sono due:

1. Gli indirizzi delle porte di I/O fanno parte integrante dello spazio di indirizzamento della memoria dati del processore. [I/O Memory Mapped]

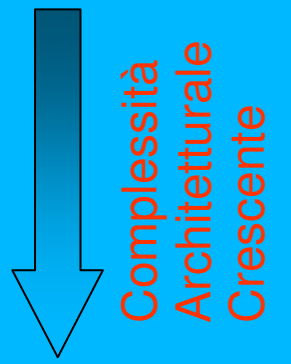
2. L' indirizzamento delle porte avviene in modalità isolata, l' indirizzo delle porte di I/O non è incluso nello spazio di indirizzamento della memoria dati. [I/O mapped o Isolated I/O o port addressed I/O]

Nel primo caso le operazioni di I/O a livello della macchina standard vengono effettuate tramite operazioni del tutto identiche a quelle relative allo spostamento valori da e per la memoria dati. Nel secondo si utilizzano tipicamente delle istruzioni speciali tipo IN o OUT che utilizzano valori particolari per l' indirizzamento delle diverse porte I/O.

I/O: Input e Output

Anche per quanto riguarda la gestione dell' I/O vengono utilizzate delle tecniche differenti a seconda dell' architettura di riferimento, le più comuni sono senz' altro:

1. I/O programmato
2. I/O gestito tramite interrupt
3. I/O con DMA
4. I/O con canali



I/O: Input e Output

L' I/O programmato è la modalità tipicamente utilizzata da architetture di fascia bassa o comunque destinate al controllo industriale con basse necessità di I/O multiplo.

Per ogni scrittura il processore eseguirà una istruzione di I/O, per la scrittura di un blocco di n caratteri ad esempio, il processore eseguirà n istruzioni di output carattere.

In lettura il discorso è duale, per l' attesa di un carattere ad esempio il processore leggerà in un ciclo stretto la porta di I/O in attesa che essa renda disponibile il carattere immesso dall' esterno. L' informazione di input disponibile viene tipicamente affidata ad un bit di controllo.

L' I/O programmato è estremamente semplice e permette numerose semplificazioni architetturali. E' infatti utilizzato dall' architettura PIC per la lettura scrittura su memoria dati, in particolare sulle porte A e B del PIC 16F84A.

Come drawback, l' I/O programmato non è ottimale nell' ipotesi in cui l' I/O di quantità di dati relativamente grosse occupi una parte rilevante del tempo macchina. In questo caso il microprocessore spenderà molto del suo tempo in operazioni di I/O come per esempio l' attesa di un carattere senza poter meglio utilizzare il suo tempo……

I/O: Input e Output

Introdurre una gestione basata su interrupt permette di migliorare notevolmente la situazione specialmente nei casi di letture/scritture multiple su dispositivi lenti.

In questo schema di gestione, la CPU può lanciare un operazione di scrittura e contemporaneamente continuare il suo flusso di esecuzione in attesa che un interrupt segnali la terminazione dell' operazione di I/O. Dualmente la presenza di un input può essere segnalata tramite interrupt.

Nel PIC, tale modalità può essere utilizzata in quanto l' unico interrupt presente può essere attivato dal cambiamento di stato di taluni bit della porta B (lettura pilotata via interrupt). In attesa di questo input la CPU può entrare ad esempio in uno stato di risparmio energetico che può essere richiamato tramite l' esecuzione di una opportuna istruzione (funzione sleep() in C2C) permettendo così l' estensione della vita operativa di dispositivi alimentati a batteria (Wireless sensor networks).

Allo stesso modo è possibile lanciare un' operazione di scrittura sulla EEPROM del PIC16F84 e (ad esempio) valutarne il corretto funzionamento al momento della terminazione, evento che può essere associato ad un apposito interrupt.

I/O: Input e Output

Le due restanti architetture sono di interesse limitato per questo corso. Si basano sull' utilizzo di integrati che svolgono funzioni di accesso al bus e controllo diretto di operazioni di I/O. La CPU di fatto, programma questi dispositivi per l' esecuzione di operazioni di I/O, le quali vengono eseguite in paralleo al flusso dii esecuzione di foreground.

PIC 16F84A: I/O su EEPROM

- EEPROM è l' acronimo di Electrically Erasable Programmable Read Only Memory. Questa parte speciale dello spazio di indirizzamento del PIC è in grado di ritenere i valori in esso contenuti anche a fronte di un prolungato periodo di power-off.
- In pratica, la EEPROM può essere vista come un dispositivo di memoria secondaria per il PIC, anche dal punto di vista prestazionale.

EEPROM PIC16F84

- La EEPROM del PIC viene indirizzata utilizzando una tecnica di indirizzamento indiretto tramite registro speciale EEADR in maniera analoga all'indirizzamento indiretto dei registri GP mediante il registro FSR.
- I valori letti o da scrivere vengono porti in un particolare registro speciale EEDATA.
- La procedura di scrittura su EEPROM è particolarmente laboriosa ed è standardizzata, ciò è dovuto alle modalità di accesso proprie della EEPROM e alle sue performance.
- Di converso le operazioni di lettura sono abbastanza semplici.

EEPROM PIC16F84

Sequenza consigliata “Writing to EEPROM”

• Porre l’ indirizzo di destinazione in EEADR	(**bank 0**)	**Indirizzamento**
• Porre i dati in EEDATA	(**bank 0**)	**Fase Preparazione**
• Disabilitare gli interrupts (clear GIE in INTCON)	(**bank 0**)	**Fase Preparazione**
• settare il write enable bit (WREN) in EECON1	(**bank 1**)	**Abilitare la scrittura**
• Scrivere il valore 0x55 in EECON2	(**bank 1**)	**Programmare la EEPROM**
• Scrivere il valore 0xAA in EECON2	(**bank 1**)	**Programmare la EEPROM**
• Settare il write bit (WR) in EECON1	(**bank 1**)	**Esecuzione della Scrittura**
• Aspettare che l’ EEIF bit in EECON1 diventi 1	(**bank 1**)	**Operazione in Corso….**
• clear EEIF bit in EECON1	(**bank 1**)	**Fase Terminazione**
• clear WREN bit in EECON1	(**bank 1**)	**Fase Terminazione**
• Abilitare gli interrupts (set GIE in INTCON)	(**bank 1**)	**Fase Terminazione**

EEPROM PIC16F84

Sequenza consigliata “Reading from EEPROM”

• Porre l’ indirizzo di destinazione in EEADR	(**bank 0)**	**Indirizzamento**
• settare il RD bit in EECON1	(**bank 1)**	**Comandare la lettura**
• Leggere il valore in EEDATA	(**bank 0)**	**Lettura del valore**

Esercitazione C (Analisi Listato)

Scrittura di un valore sulla EEPROM

Versione C

Testo function

```
void write_eeprom
    ( unsigned char address,
    unsigned char data )
{
    /* set the destination address  */
    eeadr = address ;

    /* turn off all interrupts      */
    clear_bit ( intcon, GIE ) ;

    /* set the write enable bit     */
    set_bit ( eecon1, WREN ) ;

    /* put the data in the write    */
    /* register                     */

    eedata = data ;

    /*  send the activation         */
    /*  sequence to EECON2          */
    eecon2 = 0x55 ;
    eecon2 = 0xAA ;

    /*  set the write bit           */
    set_bit ( eecon1, WR ) ;

    /* wait for the write to        */
    /* complete                     */

    while  ( ( eecon1 & EEIFM ) == 0 ) ;

    /* when we get here the bit has */
    /* gone high - indicating the   */
    /* write is complete            */

    /* turn off the write enable    */
    clear_bit ( eecon1, WREN ) ;

    /* turn off the interrupt bit   */
    clear_bit ( eecon1, EEIF ) ;

    /* turn off the interrupt flag  */
    clear_bit ( intcon, EEIE ) ;

    /* turn on all interrupts       */
    set_bit ( intcon, GIE ) ;
}
```