Il processore Motorola 68000 (MC68000)

• Il processore Motorola 68000 è stato un processore innovativo, che ha visto la luce all'inizio degli anni '80, ed il cui successo ha posto le basi per lo sviluppo di una apprezzata famiglia di processori Motorola.
• Questo processore è stato utilizzato con soddisfazione nei computer Macintosh della Apple e su alcune macchine SUN.
• Il MC68000 è ancora un processore **CISC** (Complex Instruction Set Computer), nel senso che presenta alcune primitive non banali da realizzare in hardware, ma che facilitano la scrittura dei programmi.
• Un'ulteriore prerogativa del 68000 è la capacità di indirizzare una **vasta area di memoria**, grazie ai registri indirizzi a **32** bit.
• Forse però, dal punto di vista didattico, la caratteristica più interessante di un processore ormai datato come il MC68000 è la relativa **facilità** d'uso dell'assembler messo a disposizione, al contrario della palese difficoltà riscontrata nell'utilizzo della macchina assembler individuata da altri processori, ad esempio della famiglia Intel (8086, 8088, 80836, pentium, ecc.), in parte forse per la necessità di mantenere la compatibilità verso i processori più vecchi. Purtroppo tale facilità è apprezzabile solo dopo avere programmato in entrambi gli ambienti. Ciò non toglie che anche programmando l'assembler del 68000 si incorre costantemente in errori classici, tipicamente relativi alle modalità di indirizzamento.

La macchina assembler del 68000

• Noi lavoreremo sull'interfaccia di programmazione assembler del 68000, ovvero un'interfaccia che consente all'utente di utilizzare una sorta di **macchina astratta**, la cui architettura è descrivibile come segue:

• il 68000 è un'architettura a 32 bit, ma con un bus per il trasferimento dei dati a 16 bit. Il processore è collegato alla memoria ed ai dispositivi di I/O mediante il Bus, su cui viaggiano gli **indirizzi** relativi alle istruzioni e ai dati, i **dati** di cui sono stati specificati gli indirizzi ed alcune **informazioni di controllo**.

• La macchina astratta comprende 16 registri generali (quasi generali) a 32 bit, un registro **SR** di controllo a 16 bit per la verifica dello stato del processore, ed un registro **PC** (**Program Counter**) per individuare la prossima istruzione da eseguire.

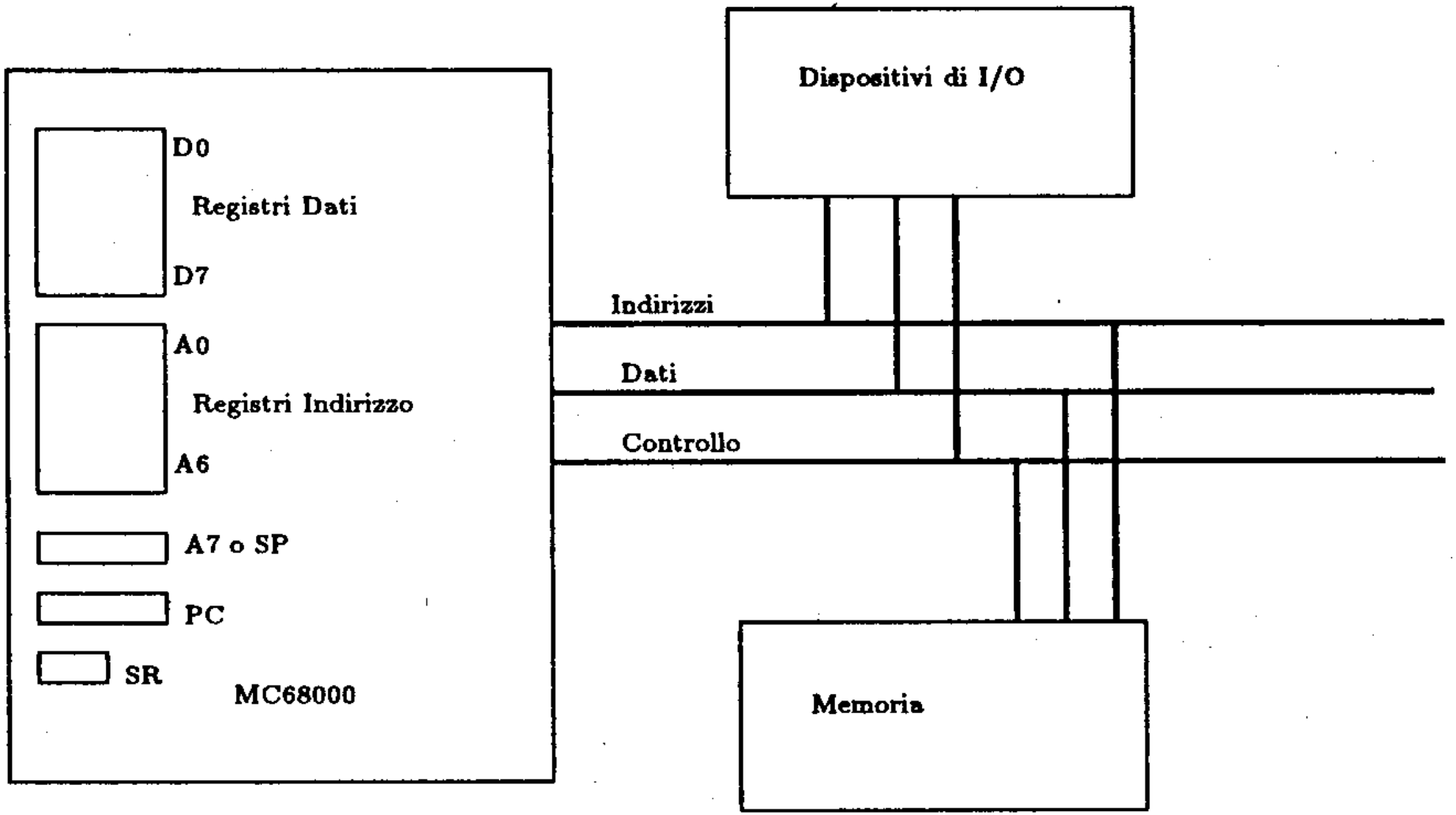
• I primi 8 registri, indicati con i simboli da **D0** a **D7**, sono **registri Dati**, ovvero sono utilizzati per memorizzare **temporaneamente** i dati durante le operazioni, e per **modificarli velocemente**.

• Altri 8 registri, indicati con i simboli da **A0** ad **A7**, sono invece **registri Indirizzi**, nel senso che servono per contenere gli indirizzi in memoria dei dati che devono essere utilizzati durante le operazioni. Questi registri indirizzo possono essere utilizzati anche come registri per contenere i dati, ma non viceversa, ovvero i registri Dati non possono essere utilizzati come registri indirizzi.

• Quindi anche nel 68000 non c'e' ortogonalità, nel senso cheanche i registri generali in realtà non sono di uso generale, ma presentano alcune particolarità che li caratterizzano.

• Un esempio ancora più evidente di ciò è il registro indirizzo **A7** indicato anche con il simbolo **SP** (**Stack Pointer**), che il processore utilizza per indicare la posizione in memoria dell'ultimo dato collocato sullo stack di sistema.

Architettura della macchina assembler del 68000



La memoria nel 68000

• La memoria è utilizzata per contenere i dati e le istruzioni del programma da eseguire, nonché alcune routine che vengono messe a disposizione dal sistema operativo.

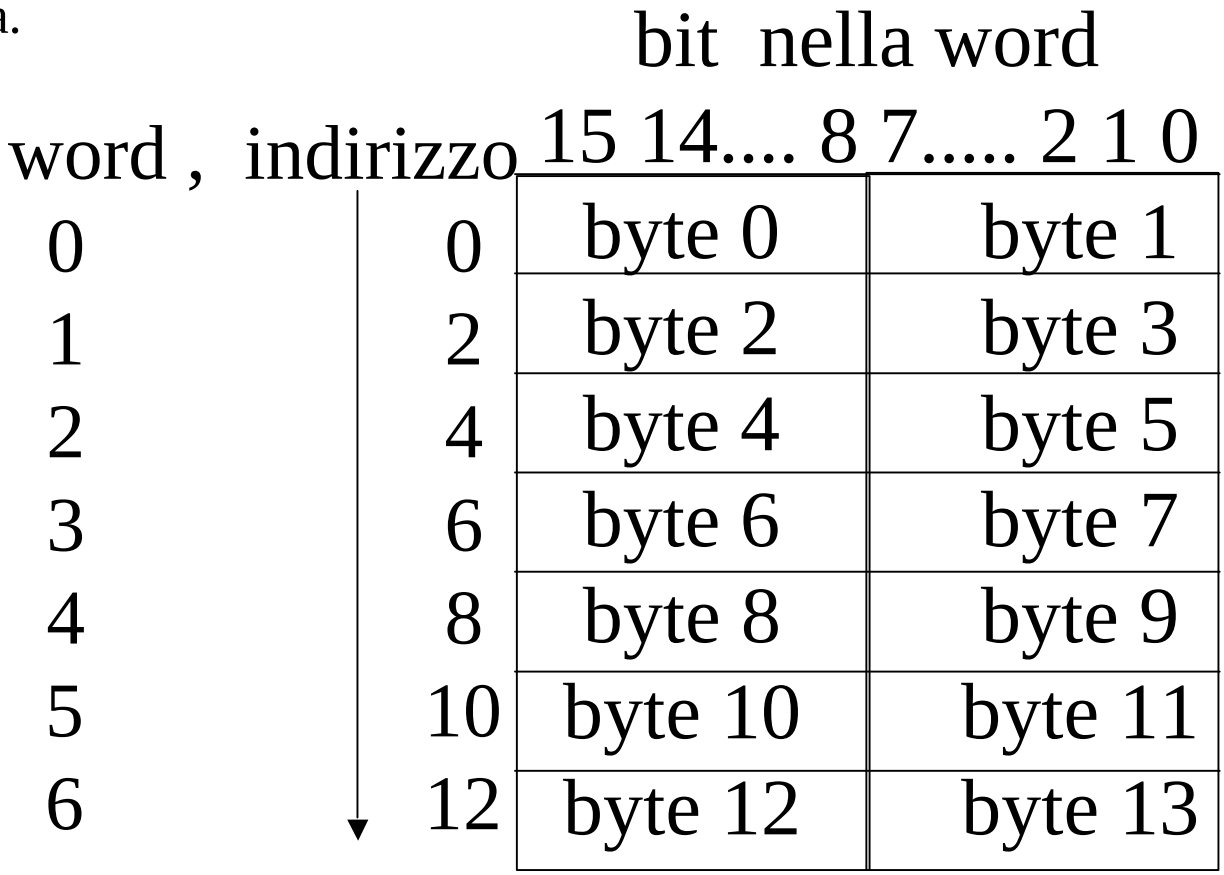
•I tipi di dati disponibili con la macchina assembler del 68000 sono:

• **byte** che occupa 8 bit, ed è la più piccola unità memorizzabile. Il carattere **B** viene utilizzato per indicare che una certa istruzione manipola un dato avente dimensione di un byte.

• **word** che occupa 16 bit (due byte). Viene usato il carattere **W**.

• **long word** che occupa 32 bit (due word ovvero 4 byte). Per indicare la dimensione di long word viene usato il carattere **L**.

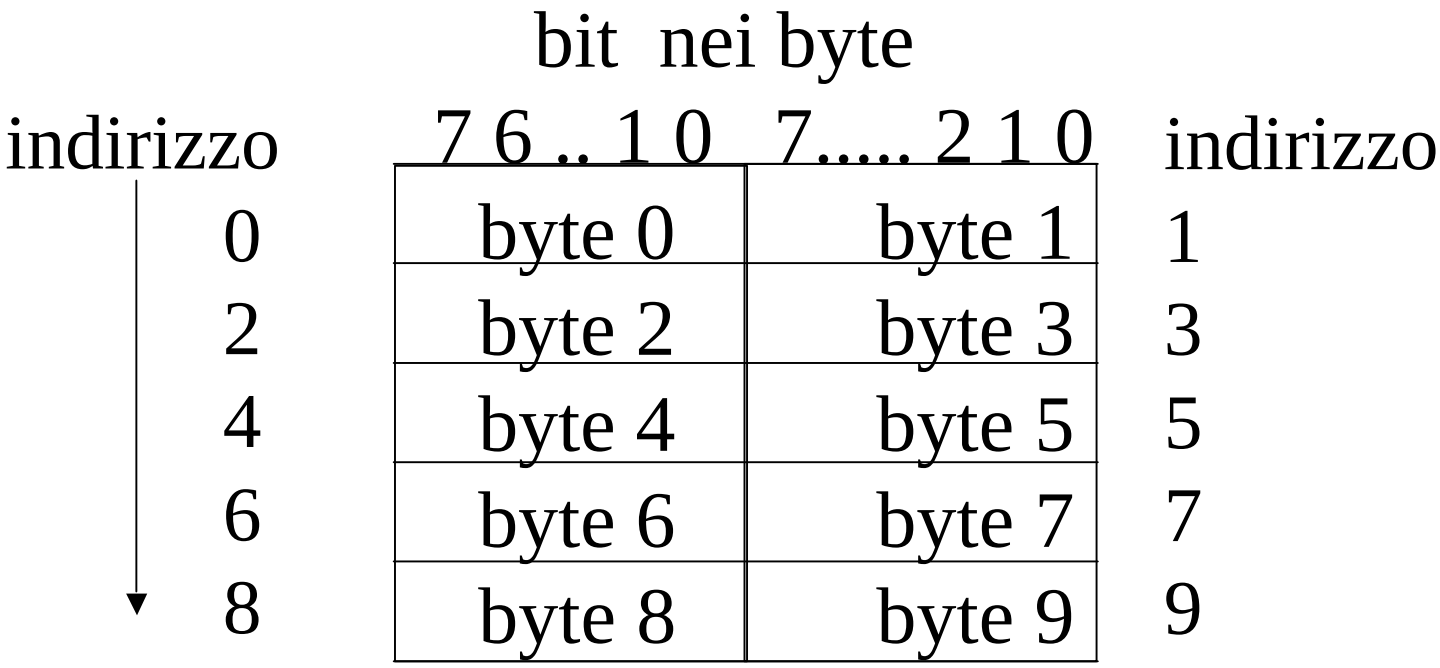
• Poiché il bus dati è in grado di trasferire una word per volta, rappresenteremo la memoria come un array di word, in cui però è possibile indirizzare ciascun byte singolarmente. L'indirizzo del primo byte varrà zero, l'indirizzo del secondo byte varrà 1 e così via.



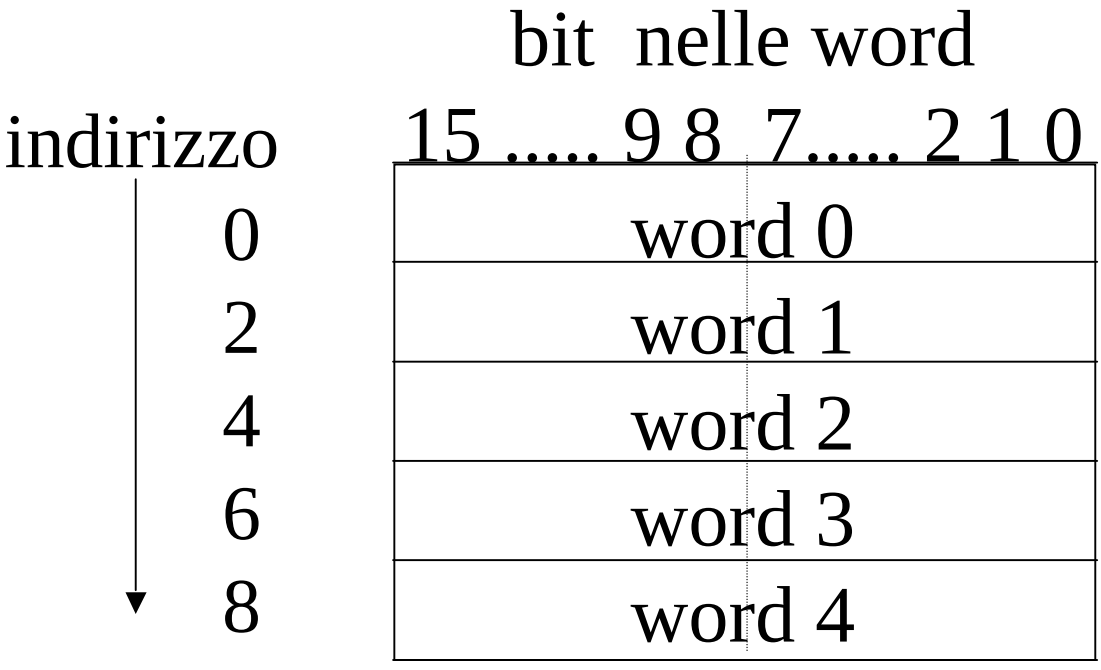
m4

Allineamento dei dati in memoria (1)

• A causa di come è costruito il bus dati, i byte possono essere collocati in uno qualsiasi dei byte della memoria, come in figura.



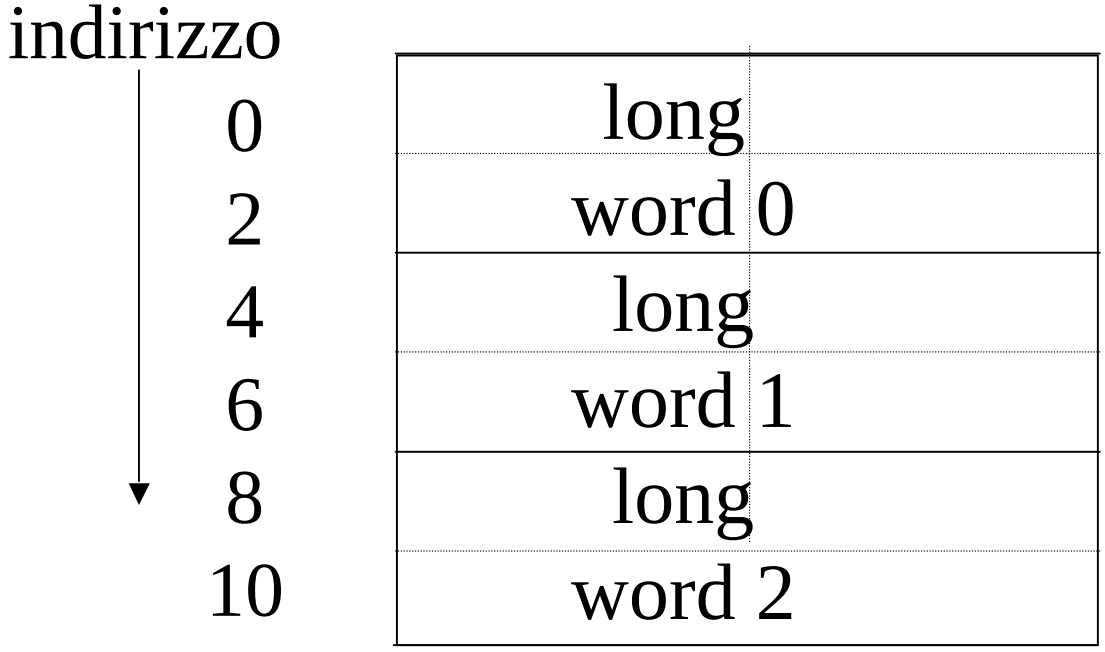
• Invece le word possono essere collocate (ovvero possono avere come indirizzo di partenza (minore)) solo i byte pari. Cioè una word può occupare solo la coppia di byte di indirizzo (2k,2k+1) con $k \in N$.



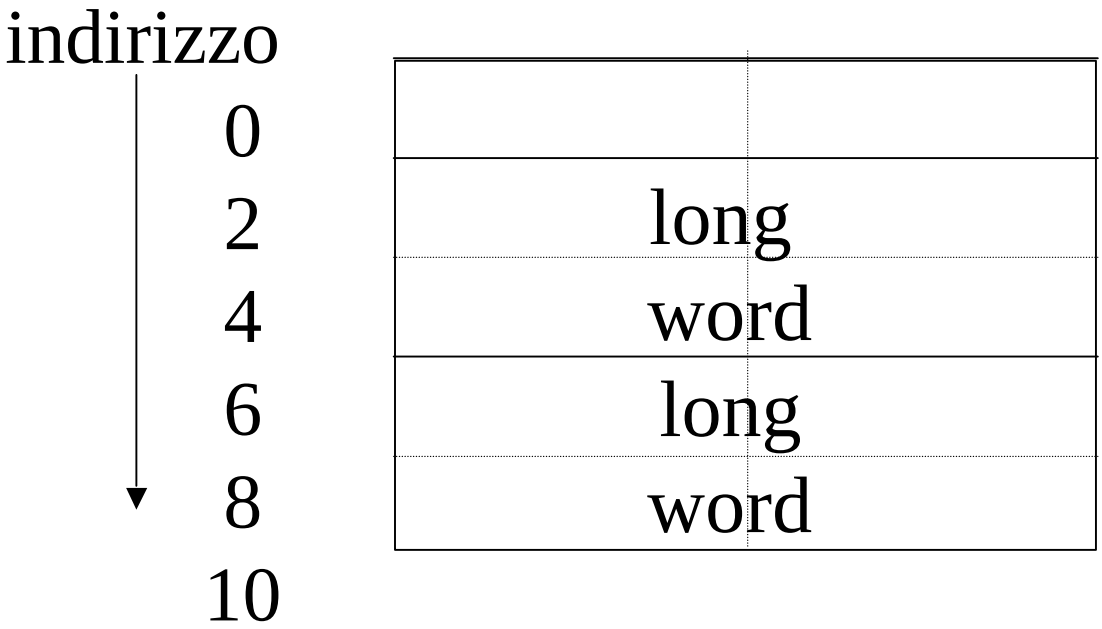
m5

Allineamento dei dati in memoria (2)

• Come per le word, anche le long word possono iniziare solo da un indirizzo pari. Cioè una long word può occupare solo la coppia di byte di indirizzo (2k,2k+1, 2k+2, 2k+3) con $k \in N$.



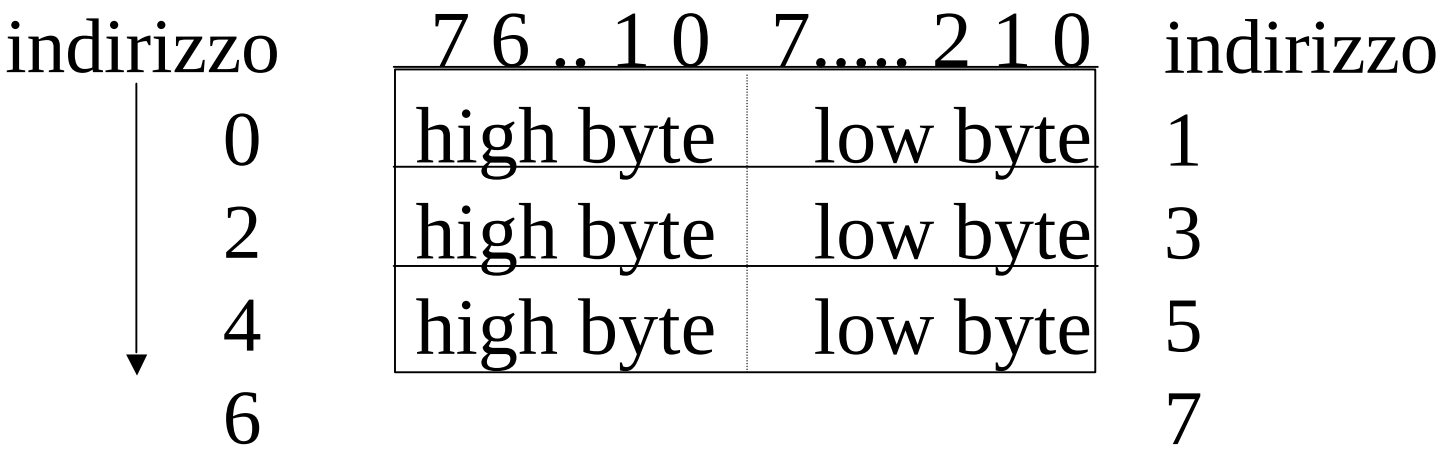
• È quindi anche possibile definire le long word ad indirizzi pari ma non multipli di 4.



Allineamento dei dati in memoria (3)

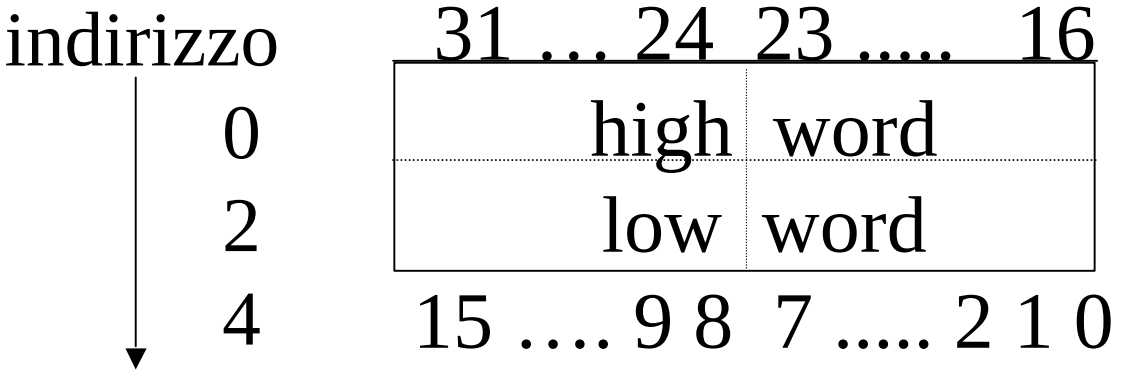
• La questione si complica perché all'interno di ciascuna word, il byte meno significativo, ovvero il byte che contiene i bit da 0 a 7 è il byte che nella word ha l'indirizzo maggiore, mentre il byte più significativo è il byte che ha l'indirizzo minore.

byte nelle word



• Analogamente, all'interno di ciascuna long word, la word più significativa, ovvero la word che contiene i byte 4 e 3 (ovvero i bit da 31 a 16) è quella che ha l'indirizzo minore, mentre l'altra word contiene i byte 1 e 0 (cioè i bit da 15 a 0).

word nelle long word



m7

I registri Dati D0..D7

• L’allineamento in memoria dei dati ha una corrispondenza nelle modalità di utilizzo dei registri che tipicamente contengono i dati, ovvero i registri **D0-D7**.

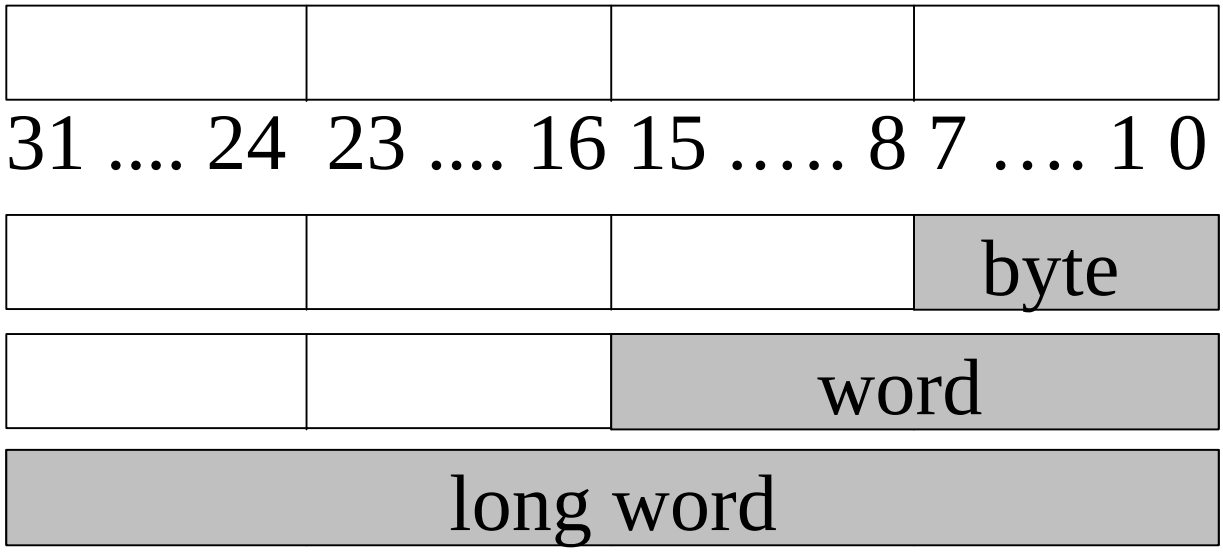
• Ciascuno dei registri dati può essere acceduto in tre differenti modi, ciascuno corrispondente ad una diversa dimensione dei dati che si stanno trattando.

• Ogni registro Dati infatti, essendo costituito da 32 bit, rappresenta una long word, e come tale può essere utilizzato. Cioè quando un’istruzione necessita di usare un dato di tipo long word, utilizza tutti i 32 bit del registro dati interessato all’operazione.

• Se invece un’istruzione deve utilizzare un dato di tipo word, utilizza solo la low word del registro dati, ovvero la word meno significativa del registro. Non è possibile invece utilizzare singolarmente la word più significativa del registro dati.

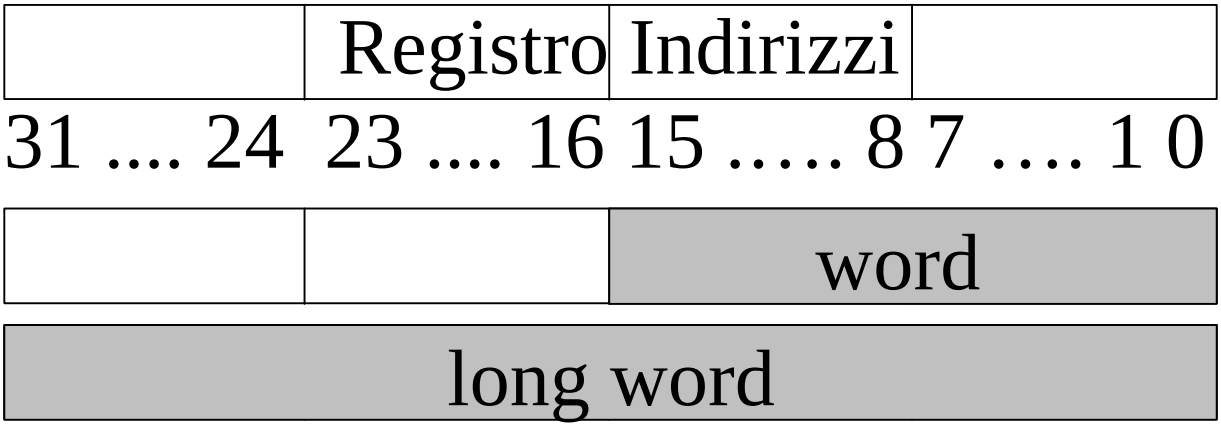
• Invece se un’istruzione deve utilizzare un dato di tipo byte utilizza solo il cosiddetto lower byte, cioè il byte meno significativo della word meno significativa del registro. Non è possibile invece utilizzare singolarmente gli altri tre byte del registro dati.

• La parte di registro non interessata dall’istruzione resta invariata.



I registri Indirizzi A0..A7

• I registri Indirizzi A**0**-A**7** sono anch'essi a 32 bit ma, a differenza dei registri dati, possono essere acceduti solo a blocchi di 32 o a 16 bit, e nel caso dell'accesso a 16 bit solo la low word può essere acceduta.
• La parte di registro non interessata dall'istruzione resta invariata.
• Le stesse modalità valgono per il registro A7 ovvero lo Stack Pointer SP

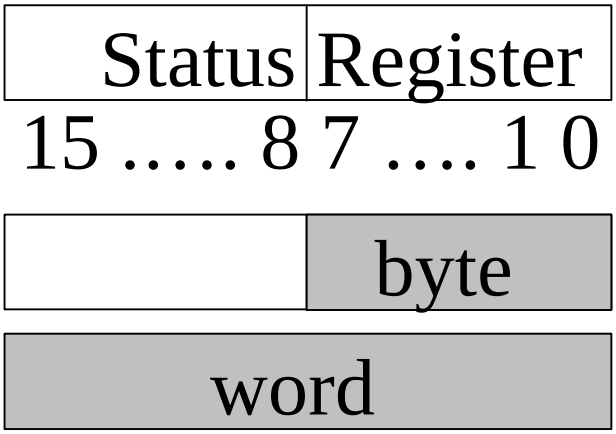


Il registro Program Counter PC

• questo registro è composto da 32 bit, ed è accedibile solo nella sua interezza. **Individua la prossima istruzione che deve essere eseguita dal processore**.

Il Registro di Stato del Processore SR

• questo registro è composto da 16 bit, ed in modalità utente solo il byte meno significativo è accedibile, mentre in modalità supervisor è accedibile integralmente.



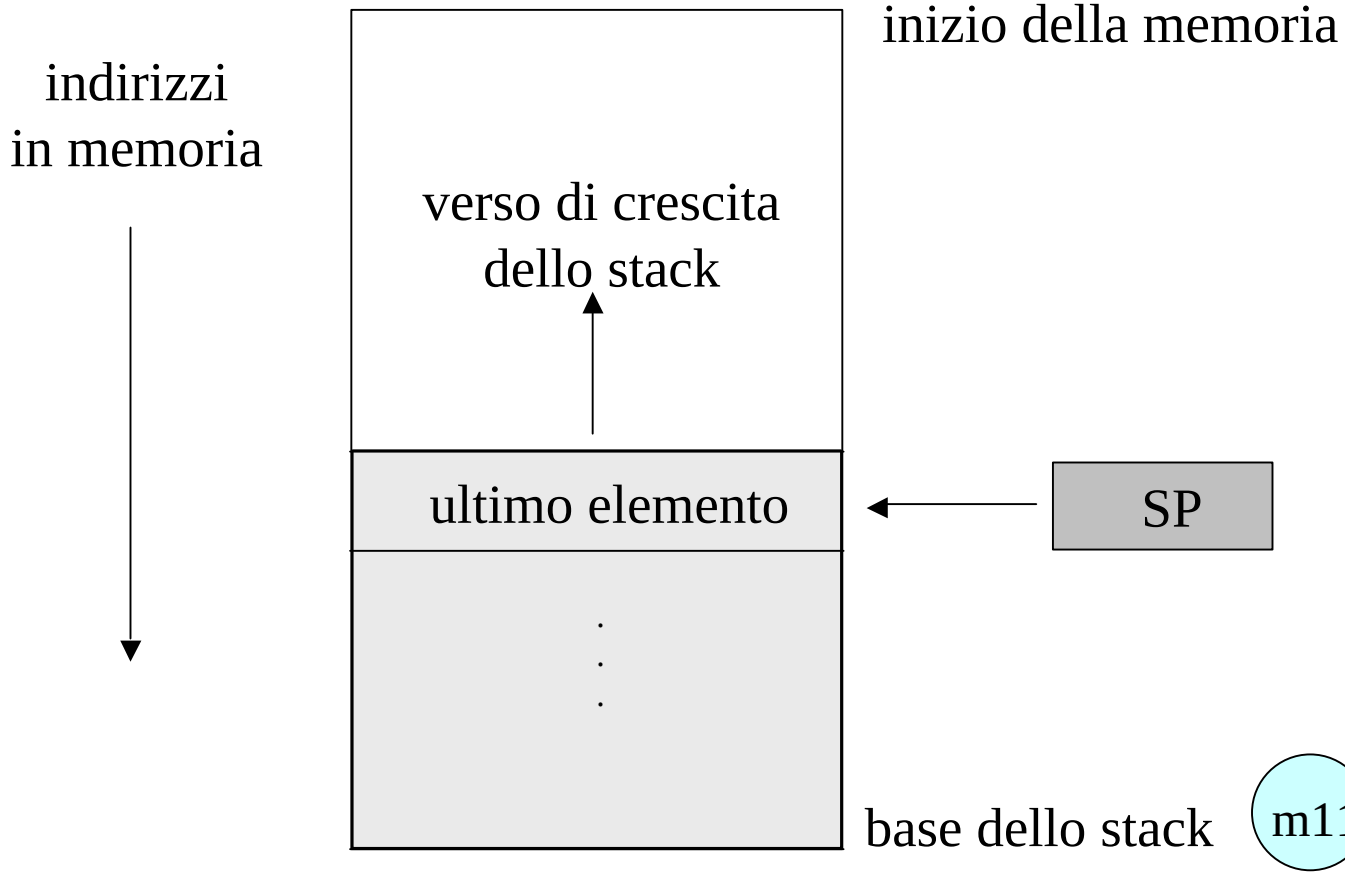
Modalità di esecuzione del 68000

• il 68000 può operare in due modi diversi, detti modalità **supervisore** ed **utente**. Il modo in cui opera è stabilito da un bit nel registro di controllo di stato SR. In modalità supervisore il processore può eseguire alcune istruzioni privilegiate, di solito utilizzate nell'implementazione del sistema operativo, ed utilizza un registro diverso per contenere il puntatore allo stack. Per ora ci limiterem ad analizzare il funzionamento in modalità utente, che è la modalità tipica con cui vengono eseguite le applicazioni.

• **NOTA BENE**: Ad eccezione dei registri A7=SP, SR e PC tutti gli altri registri **possono** essere utilizzati liberamente dal programmatore, fatti salvi i problemi di compatibilità con gli usi standard dei registri adoperati dai compilatori, ad es. nel passaggio di parametri alle funzioni.

Lo stack di sistema

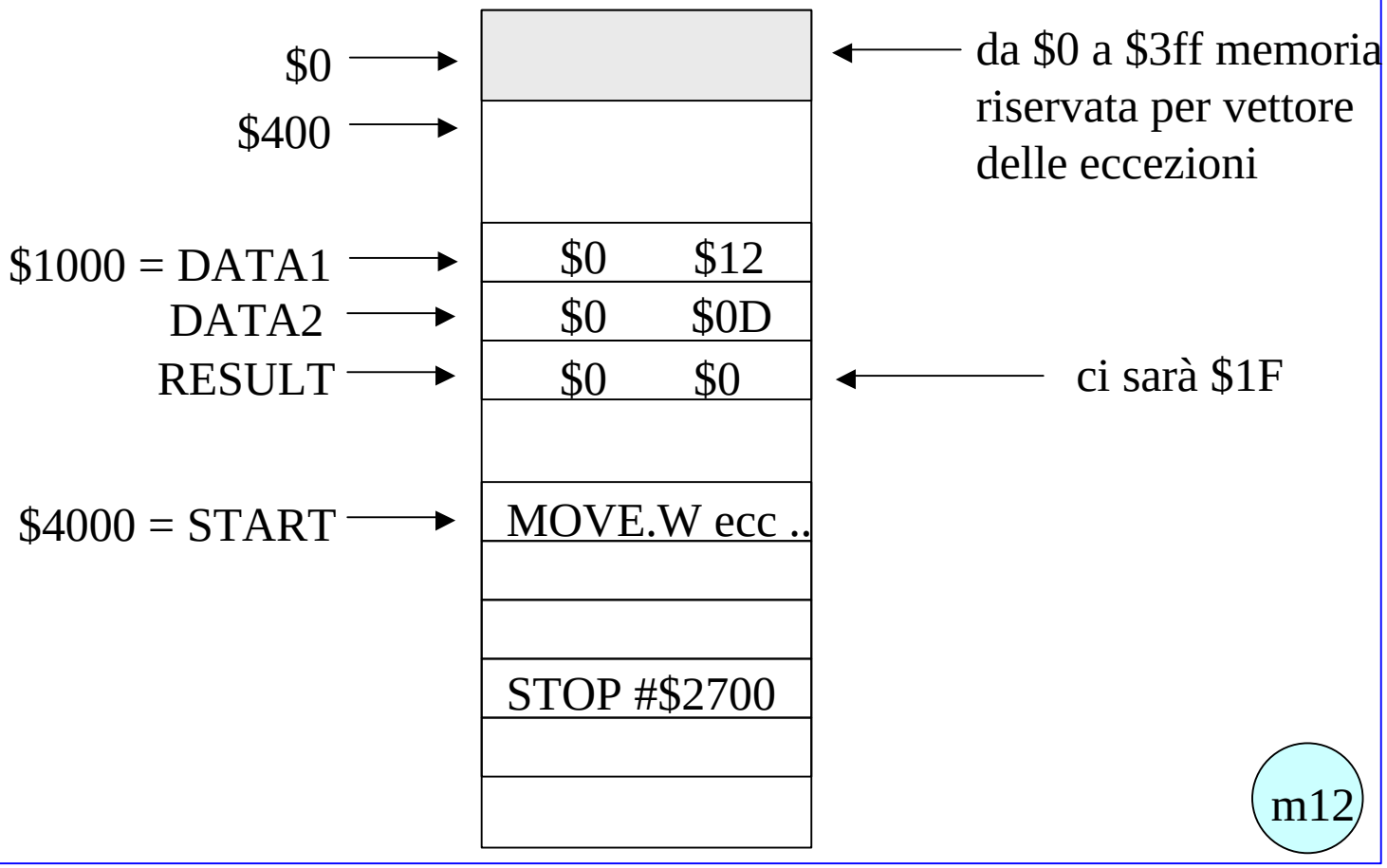
• Il Motorola 68000 è predisposto per utilizzare una struttura dati a pila, detto **Stack di Sistema**, come supporto per implementare le chiamate a sottoprogramma e la ricorsione.
• Lo stack è un'area di memoria gestita con una politica **LIFO** (Last In First Out) in cui cioè l'ultimo elemento ad entrare è il primo ad uscire.
• Lo stack è collocato in memoria, tipicamente **la base dello stack è collocata alla fine della memoria disponibile** (cioè in corrispondenza degli indirizzi maggiori). Lo stack cresce in direzione dell'inizio della memoria, cioè aggiungendo elementi sullo stack ci avviciniamo agli indirizzi più piccoli.
• Il registro SP mantiene l'indirizzo dell'ultimo elemento aggiunto allo stack, cioè dell'indirizzo più piccolo occupato dallo stack.
• Aggiungendo elementi sullo stack lo SP diminuisce, togliendo elementi dallo stack lo SP cresce.



Un primo esempio di programma

```
*       somma di due numeri contenuti in DATA1 e DATA2
*       risultato viene messo in RESULT
            ORG         $1000
DATA1   DC.W        $12
DATA2   DC.W        $0D
RESULT  DC.W         0

            ORG         $4000
*           Main Program
START:  MOVE.W    DATA1,D5
            ADD.W       DATA2,D5
            MOVE.W    D5,RESULT
            STOP        #$2700
            END         START
```



Considerazioni sul primo esempio (1)

Linea di commento. ogni linea che inizia con un asterisco * è una linea di soli commenti (e come il carattere c del fortran messo ad inizio linea).

Commenti a destra. si può mettere un commento anche a destra di ogni istruzione assembler, semplicemente separando l'istruzione con uno spazio.

Label per istruzioni e dati. le etichette devono essere composte esclusivamente di caratteri alfanumerici, e devono iniziare con una lettera, a partire dall'inizio della riga. Una label in corrispondenza di un'istruzione può (o no) essere seguita dai due punti (:). Una label in corrispondenza di un dato non deve essere seguito dai due punti. Ogni etichetta rappresenta l'indirizzo in cui è collocato (da cui inizia) ciò che segue l'etichetta, sia esso una variabile o un'istruzione.

Label per definizione di costanti. un'etichetta può anche essere utilizzata in un costrutto detto EQU, che corrisponde ad una sorta di define. In questo caso la label non deve essere seguita dai due punti, e non rappresenta l'indirizzo di ciò che segue, in quanto l'assemblatore, ogni qualvolta incontra la label la sostituisce con la costante a cui la label è associata.

Evidenziare le etichette e distinguerle dalle istruzioni. E' buona norma scrivere le istruzioni non all'inizio della linea ma dopo almeno un carattere di tabulazione.

Direttiva all'assemblatore per la definizione di Dati. L'assemblatore permette di definire ed inizializzare dei dati mediante le direttive DC.B DC.W e DC.L che riservano spazio rispettivamente per un byte, una word ed una long word. Questa direttiva deve prevedere forzatamente l'inizializzazione del dato definito. Invece la label precedente la direttiva è opzionale, ma quasi sembre utilizzata, per poter accedere al dato.

Considerazioni sul primo esempio (2)

$. il carattere $ seguito da un numero indica che il numero è espresso in forma esadecimale.

Direttiva ORG. la direttiva ORG seguita da un numero indica che le istruzioni o i dati che seguono (fino alla prossima direttiva ORG) devono essere memorizzate a partire dall'indirizzo specificato dalla direttiva ORG stessa. Poichè la zona di memoria all'inizio, in particolare la zona compresa tra gli indirizzi $0 e $3FF è riservata per la conservazione del vettore per la gestione delle eccezioni, dovrà essere presente una direttiva che specifichi un indirizzo almeno ORG $400, per salvaguardare il vettore delle eccezioni. Questo tranne casi particolarissimi.

Istruzione END. Questa istruzione END seguita da un numero o da un'etichetta indica al processore che valore deve essere assegnato al Program Counter PC, ovvero indica quale deve essere la prima istruzione da fare eseguire non appena il programma è stato caricato in memoria. Deve sempre essere presente una ed una sola istruzione END, e deve essere posta alla fine del programma.

Istruzione Move. L'istruzione MOVE prevede il seguene formato:
MOVE.SIZE SORGENTE DESTINAZIONE
SIZE può essere B o W o L ad indicare un byte o una word ouna long word. SORGENTE può essere un valore immediato o un indirizzo (calcolato in svariati modi) o un registro. DESTINAZIONE deve essere un indirizzo (calcolato in svariati modi) o un registro. L'effetto dell'istruzione MOVE è di copiare il dato (delle dimensioni specificate) indicato da SORGENTE nella locazione indicata da DESTINAZIONE.

#. Il carattere # seguito da una label o da un numero indica che quello che segue è egli stesso il dato da utilizzare. Se il carattere # non c'è ma c'è solo un numero o una label, significa che va utilizzato il dato contenuto nella zona di memoria specificata dal numero o dalla label che quindi vanno utilizzati come indirizzo del valore da usare

Un secondo esempio di programma

si vuole qui far vedere la differenza tra
l'uso di un valore contenuto in un certo indirizzo
e l'uso di un valore immediato (#)

```
*       somma di due numeri contenuti in DATA1 e DATA2
*       inizializzati durante l'esecuzione
*       risultato viene messo ancora in RESULT

            ORG         $1000
DATA1       DC.W        $0
DATA2       DC.W        $0
RESULT      DC.W        $0

            ORG         $4000
*           Main Program
START:
*           inizializzo i dati
            MOVE.W      #$12,DATA1      il # significa usa il valore stesso
            MOVE.W      #$0D,DATA2      non il contenuto dell'indirizzo

*           sommo
            MOVE.W      DATA1,D5
            ADD.W       DATA2,D5
            MOVE.W      D5,RESULT
            STOP        #$2700
            END         START
```

Un terzo esempio dello stesso programma

si vuole qui far vedere come definire delle costanti immediate, non delle variabili, cioè come usare un qualcosa di equivalente alla #define del linguaggio C

* somma di due numeri contenuti in DATA1 e DATA2
* inizializzati durante l'esecuzione
* risultato viene messo ancora in RESULT

```
VAL1    EQU     $12             e' come una define del C
VAL2    EQU     $0D

        ORG     $1000
DATA1   DC.W    $0
DATA2   DC.W    $0
RESULT  DC.W    $0

        ORG     $4000
START:
*       inizializzo i dati
        MOVE.W  #VAL1,DATA1
        MOVE.W  #VAL2,DATA2

*       sommo
        MOVE.W  DATA1,D5
        ADD.W   DATA2,D5
        MOVE.W  D5,RESULT
        STOP    #$2700
        END     START
```

Un esempio più completo di programma

```
WRCHAR    EQU     6
CR        EQU     $0D
LF        EQU     $0A

          ORG     $1000
DATA0     DC.B    $2
DATA00    DC.B    $5        inutile
DATA1     DC.W    $12
DATA2     DC.W    $0D
RESULT    DC.W    0
MSG1      DC.B    'Inizio programma',CR,LF,0
MSG2:     DC.B    'Fine programma',CR,LF,0

          ORG     $4000
*         Main Program

START:
PRINT1:   MOVEA.L  #MSG1,A0
NEXT1:    MOVE.B   (A0)+,D1
          BEQ      DOSUM
          MOVE.B   #WRCHAR,D0
          TRAP     #15
          BRA      NEXT1

DOSUM:    MOVE.W   DATA1,D5
          ADD.W    DATA2,D5
          MOVE.W   D5,RESULT

PRINT2:   MOVEA.L  #MSG2,A0
NEXT2:    MOVE.B   (A0)+,D1
          BEQ      FINISH
          MOVE.B   #WRCHAR,D0
          TRAP     #15
          BRA      NEXT2

FINISH:   STOP     #$2700
          END      START
```

solo per vedere allineamento