LABORATORIO DI ARCHITETTURA DEI CALCOLATORI

lezione n° 13

Prof. Rosario Cerbone

rosario.cerbone@libero.it

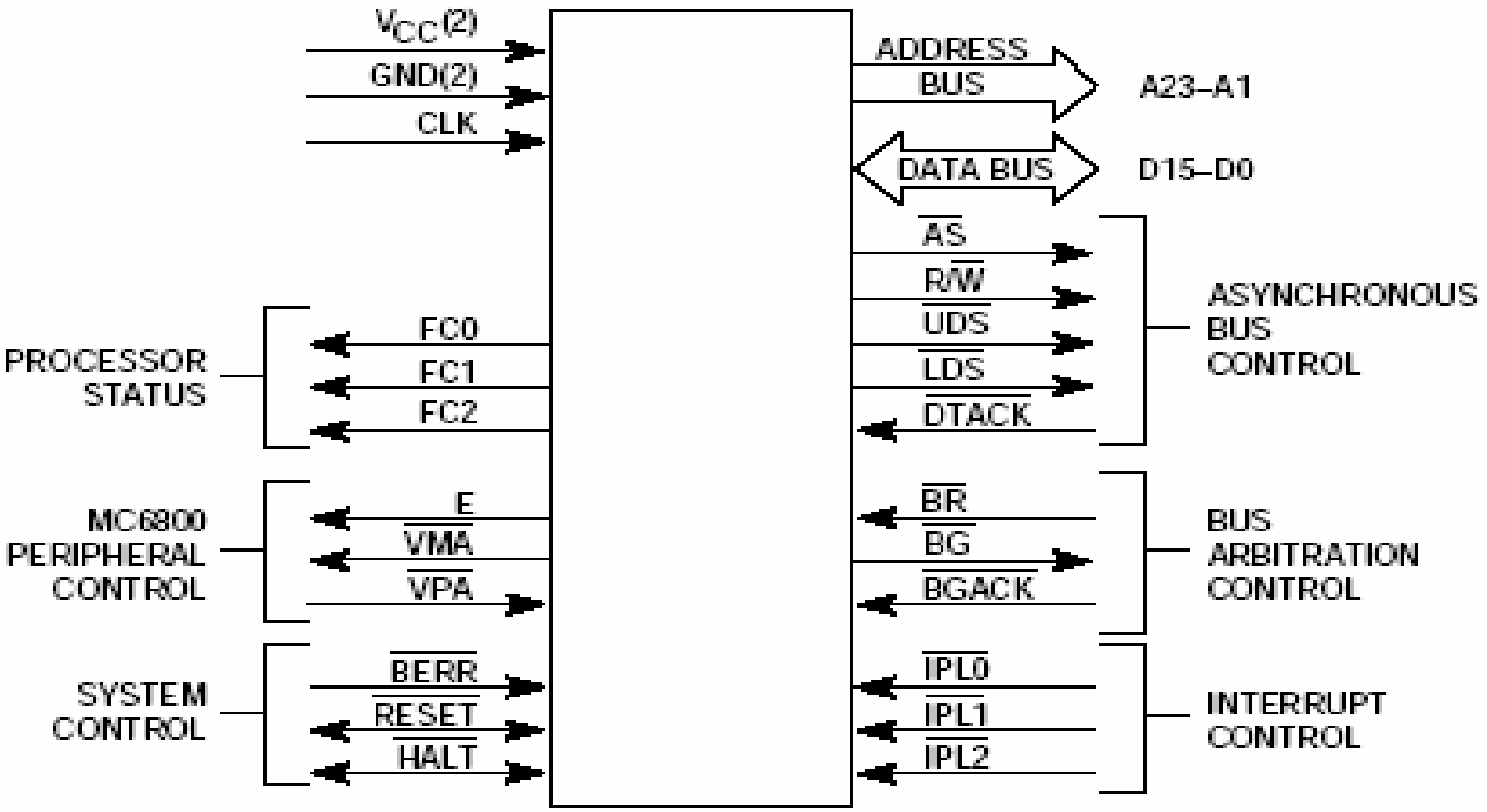
http://digilander.libero.it/rosario.cerbone

a.a. 2005-2006

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.
- Una importante prerogativa del 68000 è la capacità di indirizzare una vasta area di memoria, grazie ai registri indirizzi a 32 bit.
- Forse però, la caratteristica più interessante di un processore come il MC68000 è la relativa **facilità** d'uso dell'assembler messo a disposizione.

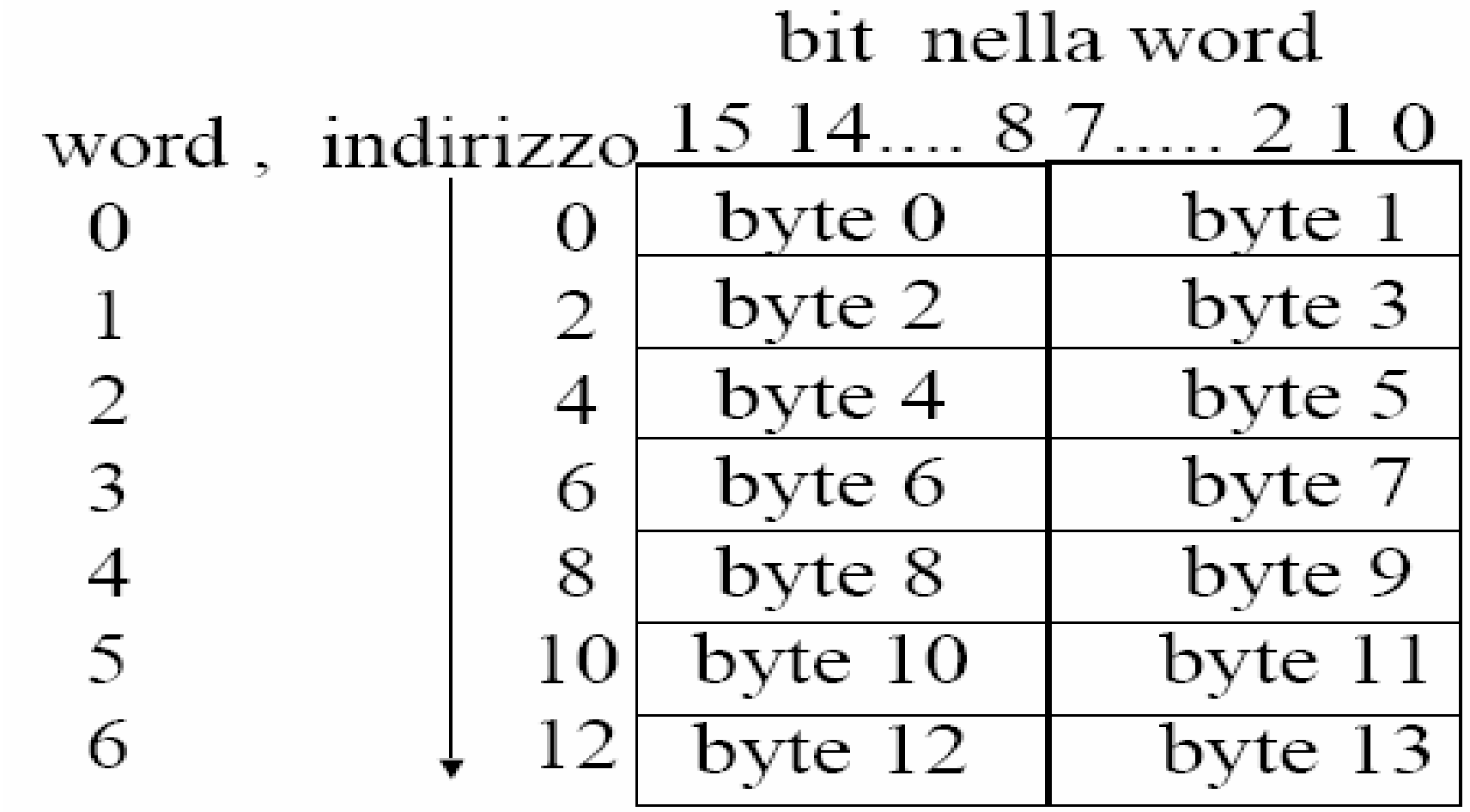
Il processore Motorola 68000 (MC68000)



La memoria nel 68000

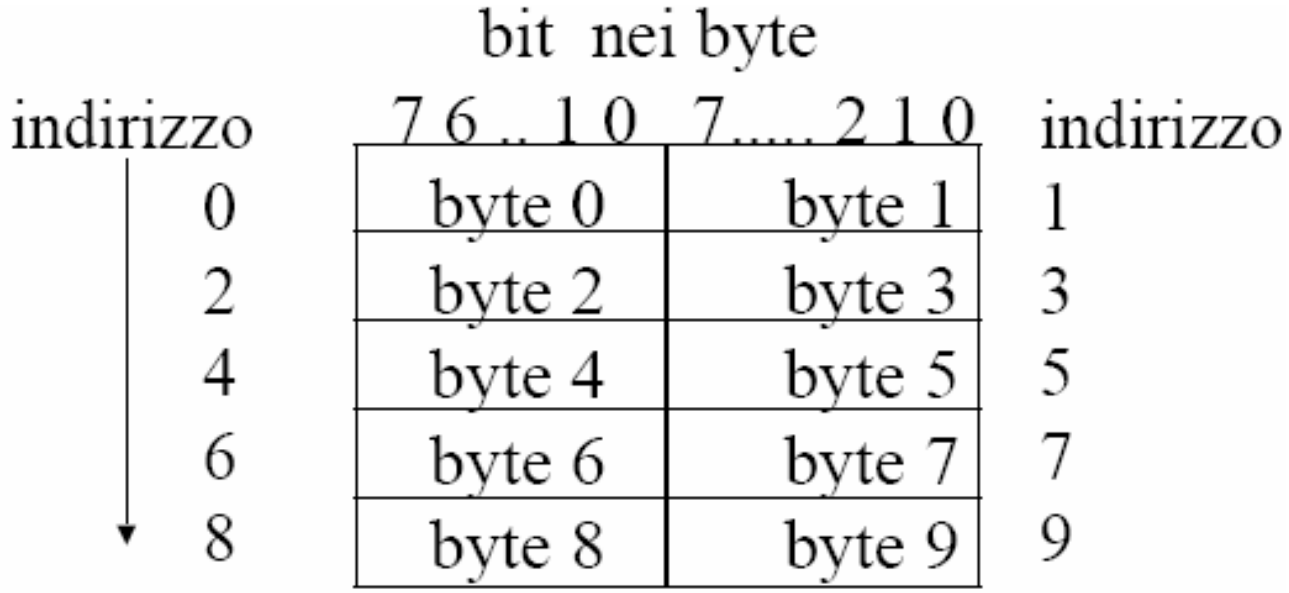
- La memoria è utilizzata per contenere i dati e le istruzioni del programma da eseguire.
- I tipi di dati disponibili con la macchina assembler del 68000 sono:
- **byte** che occupa 8 bit, ed è la più piccola unità memorizzabile. Il carattere **B** viene utilizzato per indicare che una certa istruzione 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**.

La memoria nel 68000



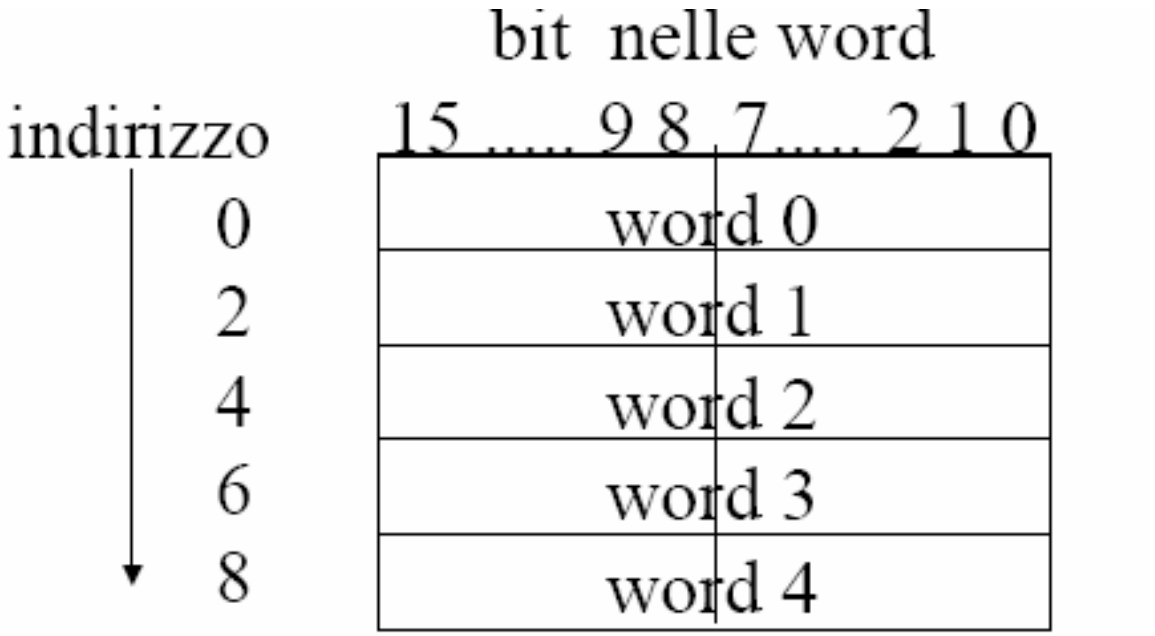
Allineamento dei dati in memoria

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



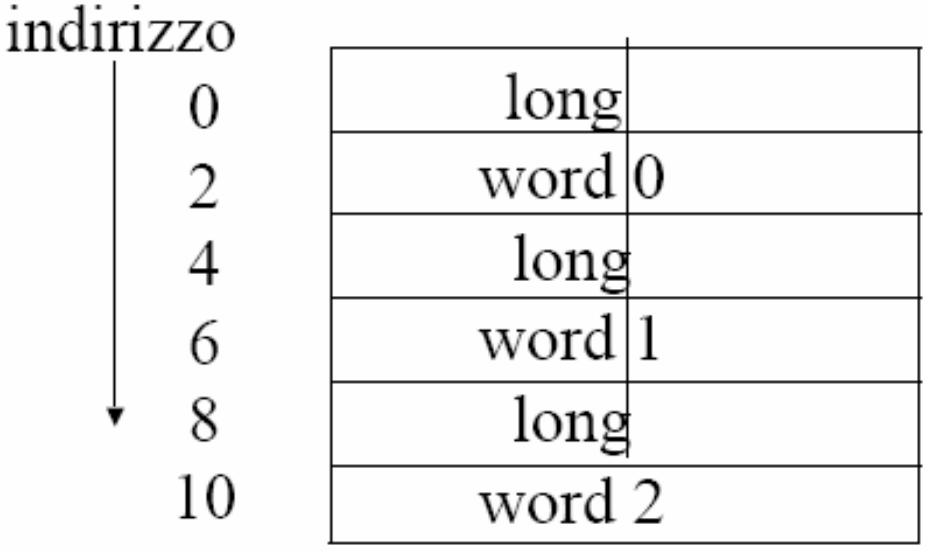
Allineamento dei dati in memoria

- 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.



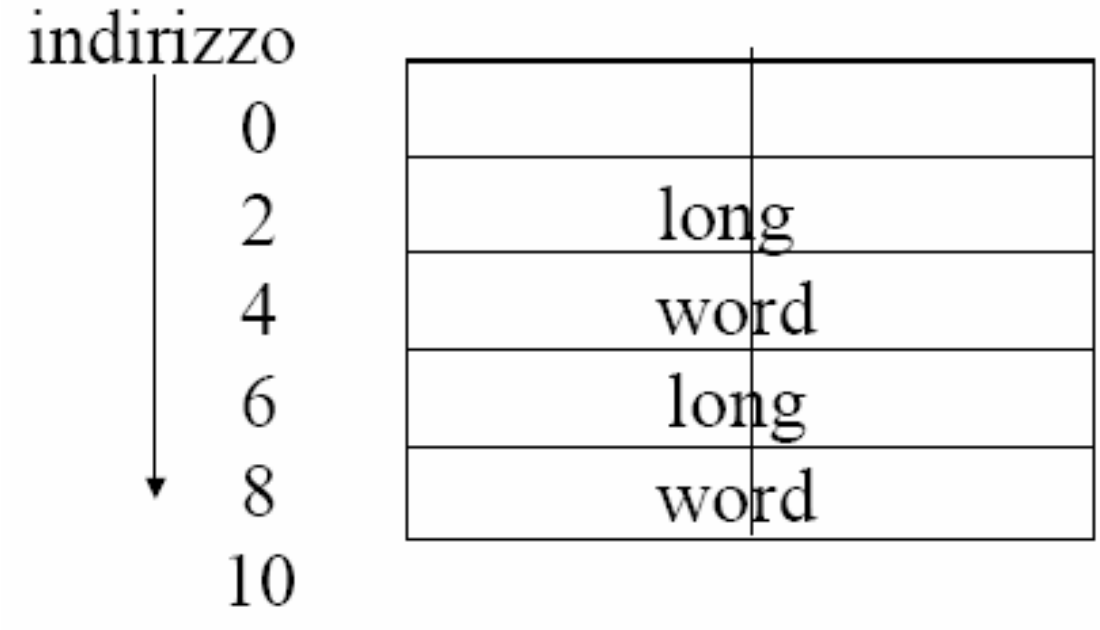
Allineamento dei dati in memoria

- 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.



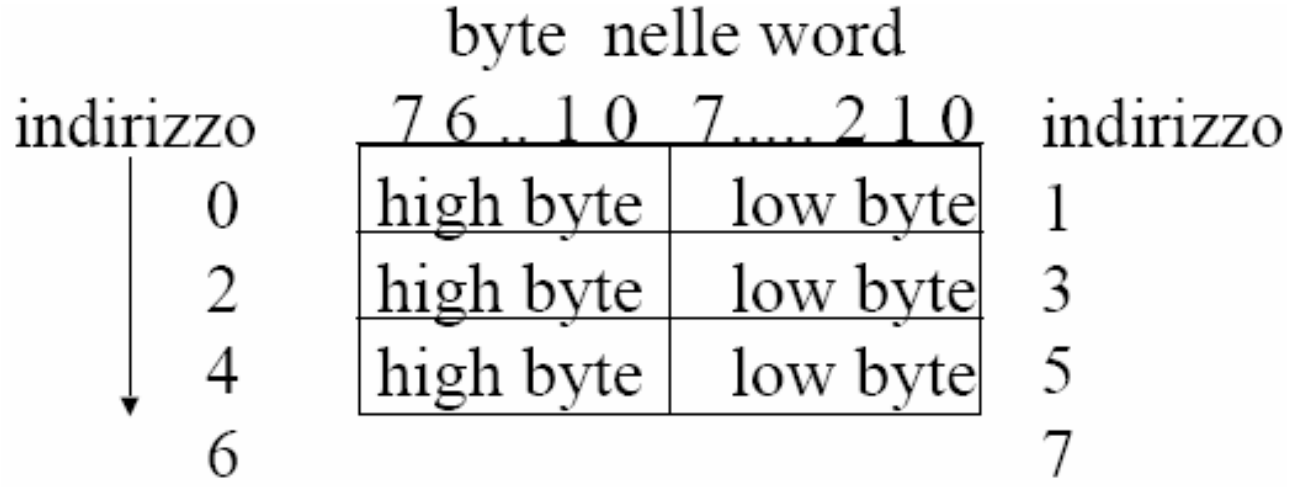
Allineamento dei dati in memoria

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



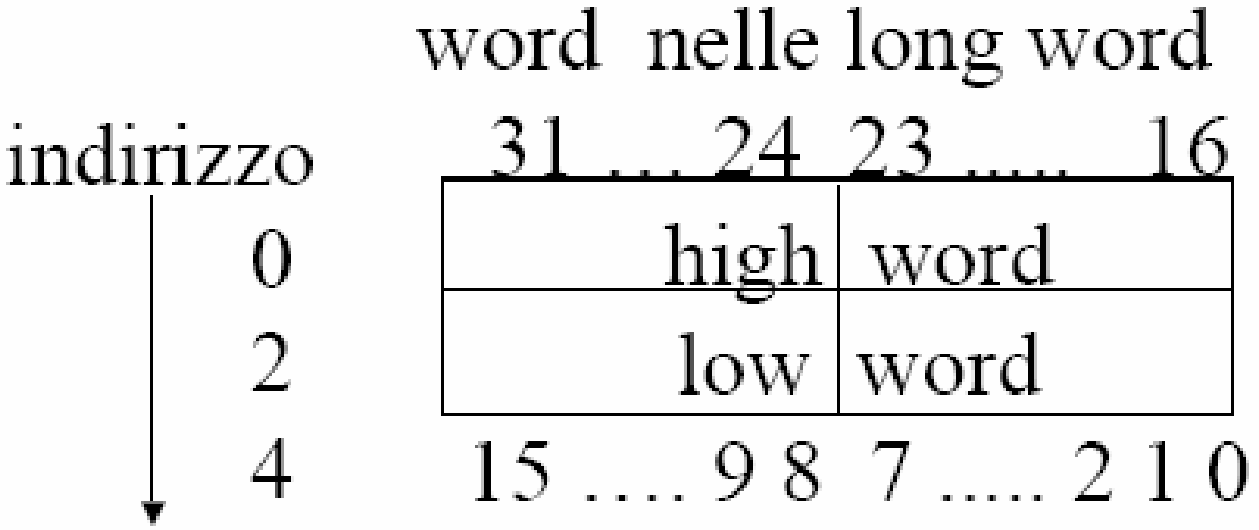
Allineamento dei dati in memoria

- 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 (big endian).

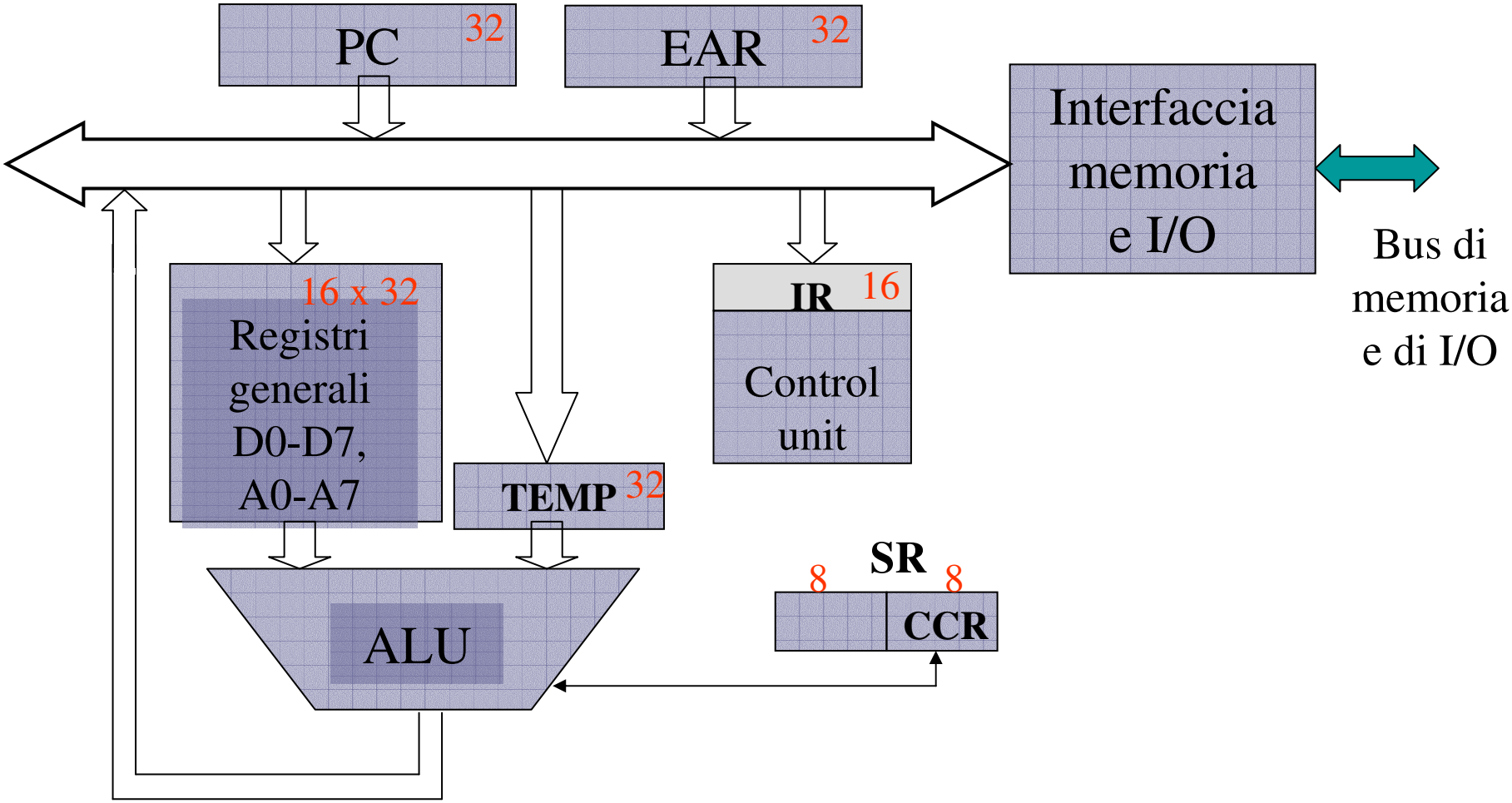


Allineamento dei dati in memoria

- 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).



Il processore Motorola 68000 (MC68000)

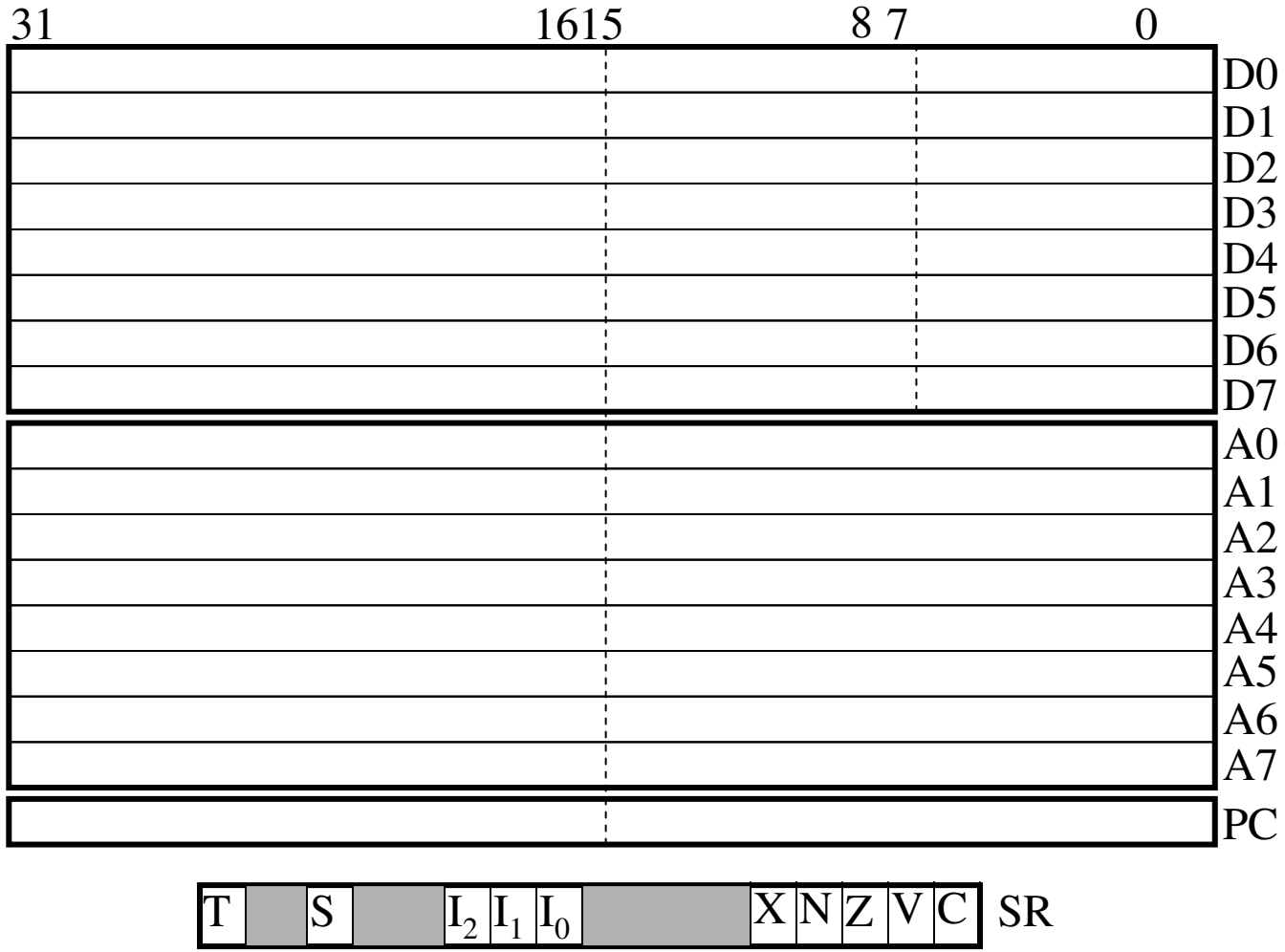


Caratteristiche principali

Il microprocessore 68000 presenta le seguenti caratteristiche:

- 8 registri dati general purpose a 32 bit;
- 8 registri indirizzi general purpose a 32 bit;
- Data bus a 16 bit;
- Address bus a 24 bit;
- Spazio di indirizzamento diretto di 16 MB;
- 5 tipi principali di dati per le operazioni;
- Memory Mapped I/O;
- 14 modalità di indirizzamento;
- 2 livelli di privilegio: user e supervisor;
- 7 livelli di priorità per le interruzioni esterne.

MC68000: modello di programmazione

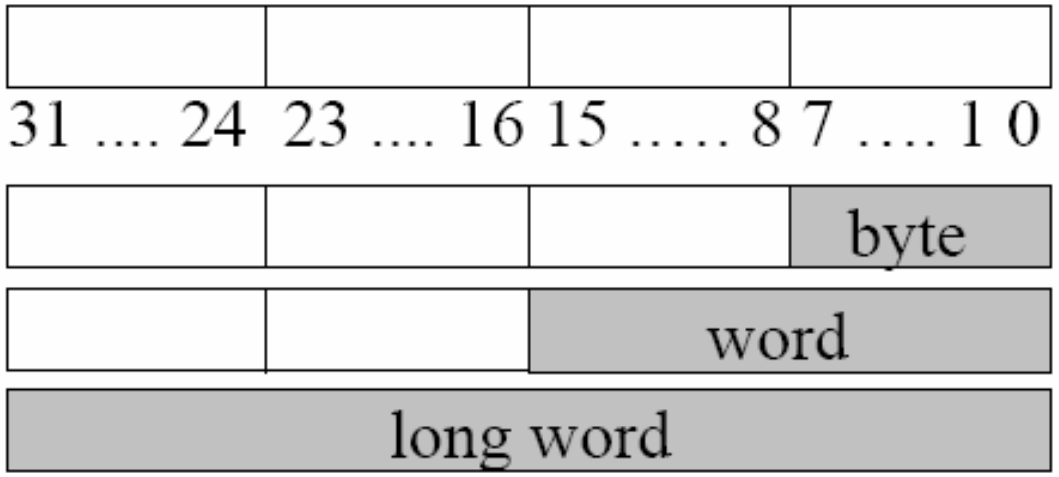


I registri Dati D0..D7

- 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 utilizzato 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.

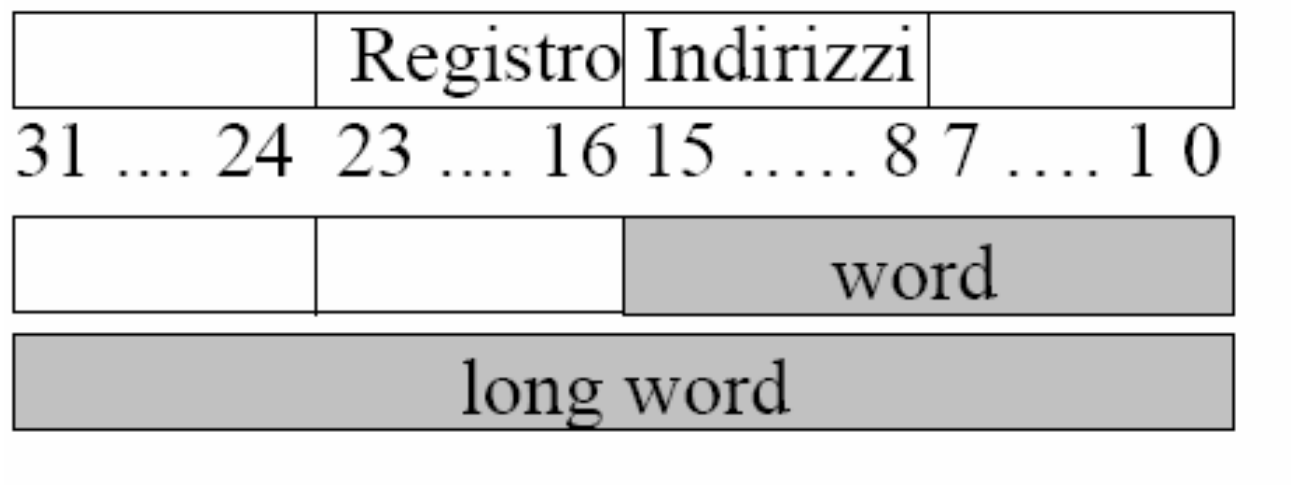
I registri Dati D0..D7

- 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.



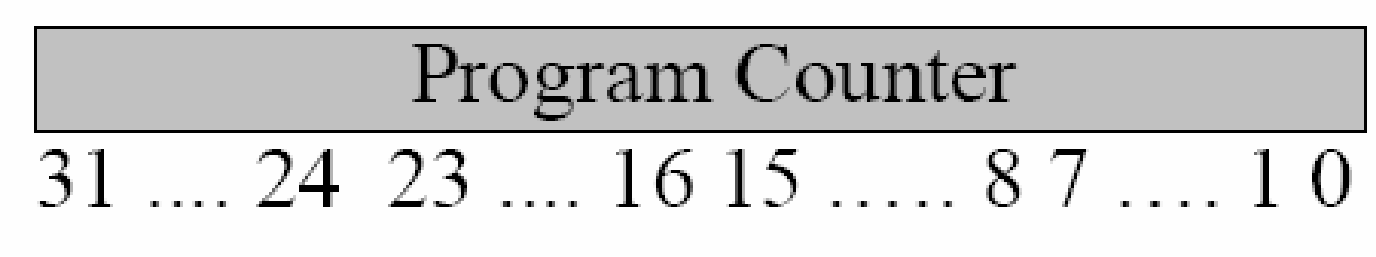
I registri Indirizzi A0..A7

- I registri Indirizzi A**0…**A**7** sono anch'essi a 32 bit ma, a differenza dei registri dati, possono essere utilizzati solo a blocchi di 32 o a 16 bit, e nel caso dell'accesso a 16 bit solo la low word può essere acceduta.
- In scrittura la rimanente parte di registro viene modificata (est. segno)
- Le stesse modalità valgono per il registro A7 ovvero lo Stack Pointer SP.



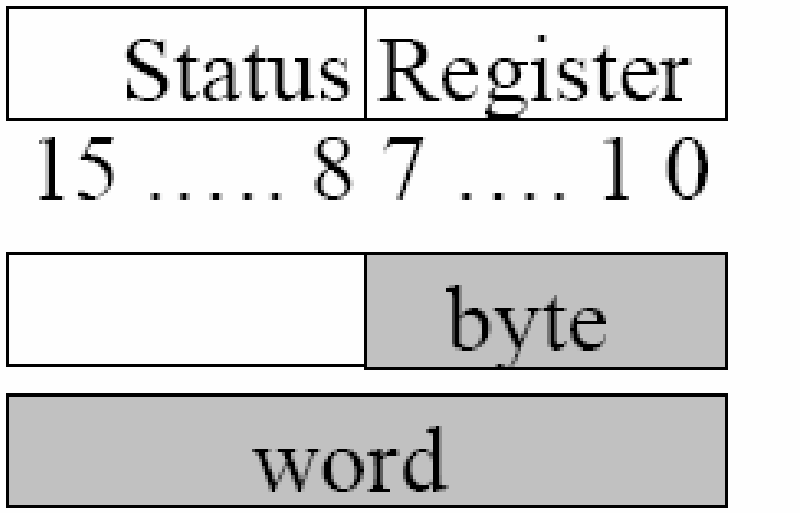
Il registro Program Counter PC

- 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

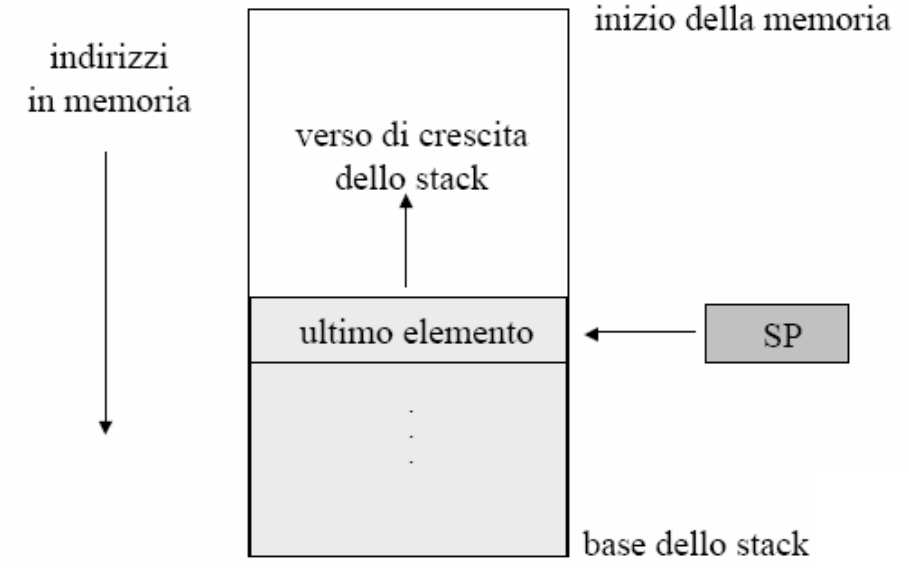
- Il 68000 può operare in due modi diversi, detti modalità **supervisore** ed **utente**. Il modo in cui opera è stabilito da un bit nel registro di controllo di stato SR. In modalità supervisore il processore può eseguire alcune istruzioni privilegiate, di solito utilizzate nell'implementazione del sistema operativo, ed utilizza un registro diverso per contenere il puntatore allo stack (A7').
- Ad eccezione dei registri A7=SP, SR e PC tutti gli altri registri **possono** essere utilizzati liberamente dal programmatore.

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.

Lo stack di sistema

- 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.



Introduzione all'assembly

- Confrontando le potenzialità del linguaggio assembly con quelle di un linguaggio di programmazione ad alto livello come Java o il C, si possono individuare i seguenti vantaggi:
  - è possibile accedere ai registri della CPU;
  - è possibile scrivere codice ottimizzato per una specifica architettura di CPU;
  - è possibile ottimizzare accuratamente le sezioni "critiche" dei programmi.
- Viceversa, i principali svantaggi derivanti dall'utilizzo dell'assembly sono i seguenti:
  - possono essere richieste molte righe di codice assembly per riprodurre il comportamento di poche righe di Java o C;
  - è facile introdurre dei bug nei programmi perche' la programmazione e' piu' complessa;
  - i bug sono difficili da trovare;
  - non è garantita la compatibilità del codice per versioni successive dell'hardware;

Introduzione all'assembly

- Ma allora… a cosa serve conoscere l'assembly?
- L'utilizzo del linguaggio assembly, rispetto all'uso dei tradizionali linguaggi ad alto livello, è talvolta giustificato dalla maggiore efficienza del codice prodotto. I programmi assembly, una volta compilati, sono tipicamente più veloci e più piccoli dei programmi scritti in linguaggi ad alto livello anche se il numero di righe di codice scritte dal programmatore e' maggiore. Inoltre, la programmazione assembly è indispensabile per la scrittura di driver per hardware specifici. Non va infine dimenticato che conoscere l'assembly consente di scrivere codice ad alto livello di qualità migliore.

Assemblare, verificare ed eseguire un programma assembly

- Il processo di creazione di un programma Assembly passa attraverso le seguenti fasi:
- Scrittura di uno o più file ASCII contenenti il programma *sorgente*, tramite un normale *editor di testo*.
- Assemblaggio dei file sorgenti, e generazione dei file *oggetto* tramite un *assemblatore*.
- Creazione, del file *eseguibile*, tramite un *linker*.
- Verifica del funzionamento e correzione degli eventuali errori, tramite un *debugger*.

L’assemblatore

- L’Assemblatore trasforma i file contenenti il programma sorgente in altrettanti file oggetto contenenti il codice in linguaggio macchina.

Il linker

- Il linker combina i moduli oggetto e produce un unico file eseguibile. In particolare:
- unisce i moduli oggetto, risolvendo i riferimenti a simboli esterni; ricerca i file di libreria contenenti le procedure esterne utilizzate dai vari moduli e produce un modulo rilocabile ed eseguibile.
- Notare che l'operazione di linking deve essere effettuata anche se il programma è composto da un solo modulo oggetto.
- Non è previsto l'uso di un linker, data la finalità didattica del corso.

Il debugger

- Il debugger è uno strumento software che permette di verificare l'esecuzione di altri programmi. Il suo utilizzo risulta indispensabile per trovare errori (bug, da qui il nome debugger) in programmi di complessità elevata. Le principali caratteristiche di un debugger sono:
- possibilità di eseguire il programma "passo a passo";
- possibilità di arrestare in modo condizionato l'esecuzione del programma tramite l'inserimento di punti di arresto detti *breakpoint*;
- possibilità di visualizzare ed eventualmente modificare il contenuto dei registri e della memoria.
- L'ASIM possiede tutte le caratteristiche elencate assieme ad altre che permettono lo sviluppo di programmi.

Introduzione all'assembly

- Un processore, durante il suo funzionamento, esegue senza sosta le fasi di fetch, operand assembly ed execute

- Fase fetch:
  - preleva dalla memoria l'istruzione (sequenza di bit) correntemente indirizzata dal registro PC
  - copia l'istruzione nel registro IR
  - incrementa il PC in modo da "puntare" all'istruzione seguente
- L'unità di controllo decodifica l'istruzione ed avvia la serie di ***micro-operazioni*** che producono l'esecuzione dell'istruzione.

Esecuzione in sequenza lineare

Esecuzione in sequenza lineare: in seguito ad ogni fase fetch, il PC viene incrementato per puntare alla locazione successiva.

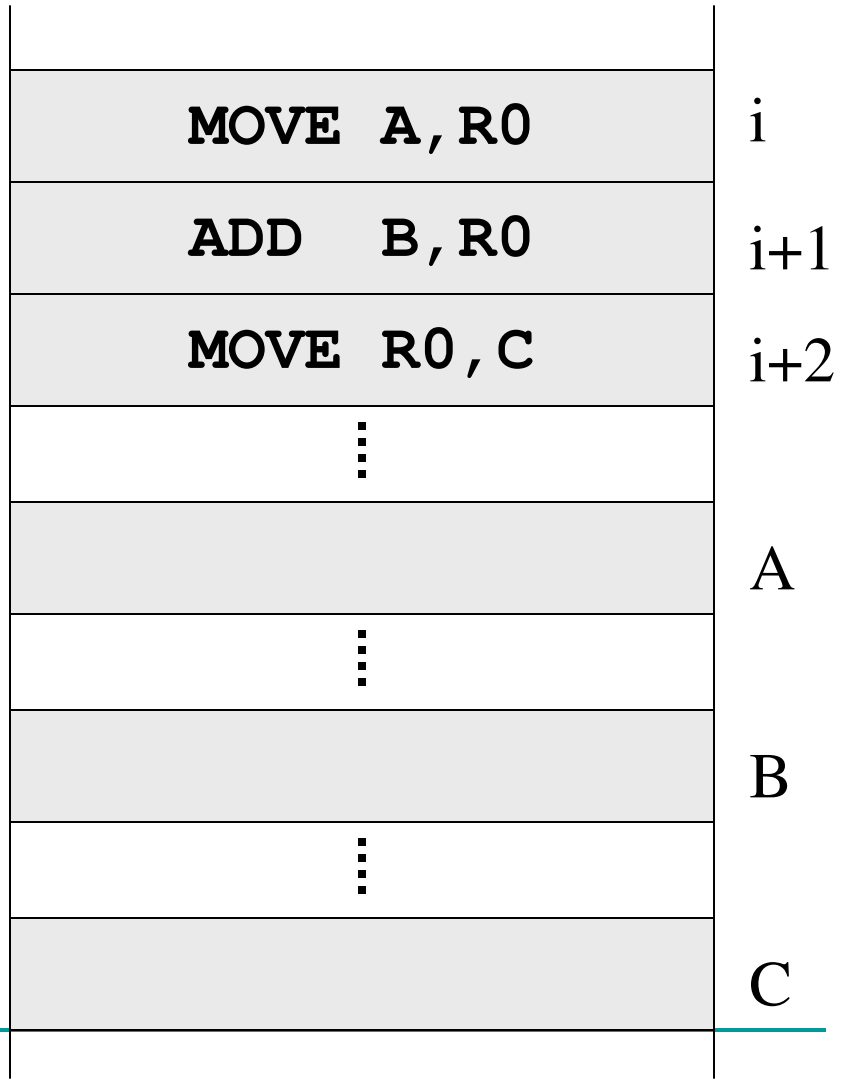
- All'inizio (PC) contiene l'indirizzo **`i`**.

*Fase fetch: il contenuto della **`i`**-esima locazione di memoria passa in (IR).

*Subito dopo aver prelevato l'istruzione, il PC si incrementa per puntare alla successiva locazione di memoria (**`i+1`**).

.....

- Dopo che la **MOVE** alla locazione **i+2** è stata eseguita, il PC contiene il valore **i+3**, che è l'indirizzo della prima istruzione del segmento di programma seguente.



Il linguaggio macchina

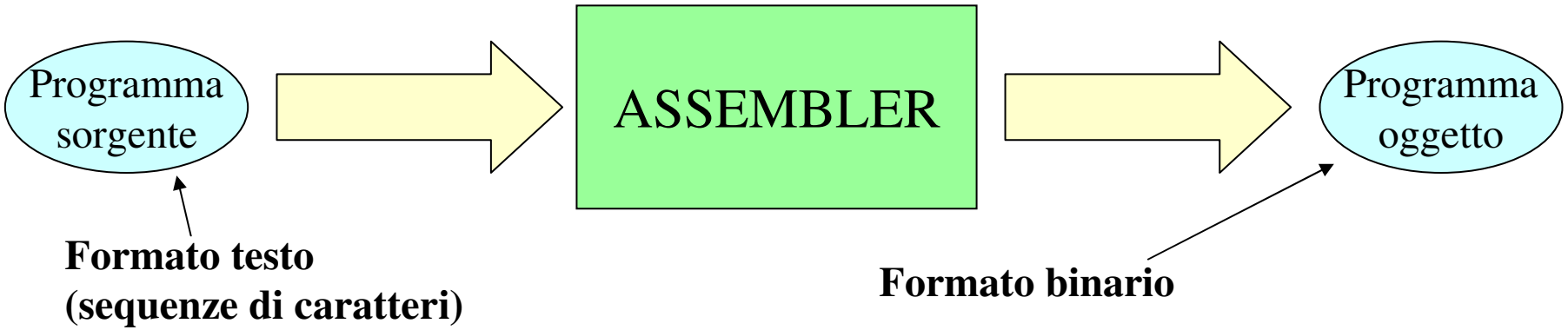
- Esempi di *micro-operazioni*:
  - abilitazione di un registro alla scrittura sul bus interno
  - abilitazione di un registro alla lettura dal bus interno
  - abilitazione della memoria alla scrittura nel registro
  - invocazione di una funzionalità dell'ALU
- Ad ogni istruzione in linguaggio macchina è abbinata una ben precisa e specifica serie di micro-operazioni
- Il programmatore non ha accesso alle micro-operazioni (l'istruzione in linguaggio macchina è vista come atomica)

Il linguaggio Assembly

- Le istruzioni in linguaggio macchina hanno cardinalità finita, e sono strettamente dipendente dal particolare processore.
- ***Programma***: un insieme di istruzioni in linguaggio macchina più i dati su cui le istruzioni devono operare.
- Linguaggio Assembly: il programmatore non utilizza sequenze di bit ma, attraverso l'assembler, identifica:
  - i codici operativi con codici mnemonici;
  - gli indirizzi di memoria ed i registri con identificatori testuali

Il linguaggio Assembly

- Un programma assembler per essere eseguito, deve essere tradotto da un assemblatore in linguaggio macchina



Esempio: un listato assembly…

```
* Programma per sommare i primi 17 interi
*
                        ORG     $8000
START                   CLR.W   SUM
                        MOVE.W  ICNT,D0
ALOOP                   MOVE.W  D0,CNT
                        ADD.W   SUM,D0
                        MOVE.W  D0,SUM
                        MOVE.W  CNT,D0
                        ADD.W   #-1,D0
                        BNE     ALOOP
                        JMP     SYSA
SYSA            EQU     $8100
CNT             DS.W    1
SUM             DS.W    1
IVAL            EQU     17
ICNT            DC.W    IVAL
                        END     START
```

…esempio: il lavoro dell'assembler

```
00000000                    1  * Programma per sommare i primi 17 interi
00000000                    2  *
00008000                    3          ORG  $8000
00008000  4279 00008032     4  START   CLR.W      SUM
00008006  3039 00008034     5          MOVE.W     ICNT,D0
0000800C  33C0 00008030     6  ALOOP   MOVE.W     D0,CNT
00008012  D079 00008032     7          ADD.W      SUM,D0
00008018  33C0 00008032     8          MOVE.W     D0,SUM
0000801E  3039 00008030     9          MOVE.W     CNT,D0
00008024  0640 FFFF         10         ADD.W      #-1,D0
00008028  66E2              11         BNE        ALOOP
0000802A  4EF9 00008100     12         JMP        SYSA
00008030  =00008100         13 SYSA    EQU        $8100
00008030                    14 CNT     DS.W       1
00008032                    15 SUM     DS.W       1
00008034  =00000011         16 IVAL    EQU        17
00008034  0011              17 ICNT    DC.W       IVAL
00008036                    18         END START
```

Classi di istruzioni

- Un calcolatore deve avere istruzioni in grado di effettuare quattro tipi di operazioni
  - trasferimento dei dati tra la memoria e i registri di CPU;
  - operazioni aritmetiche e logiche sui dati;
  - controllo di flusso di un programma;
  - trasferimento dei dati in ingresso/uscita (I/O).

Assembler del processore MC68000

Formato delle istruzioni ASM68K:

etichetta	**cod. operativo**	**operando/i**	**commento**
(*label)*	(*opcode)*	(*operand(s))*	(*remark)*

Le istruzioni assembler si dividono in tre categorie:

- **Assembly-time operations** - sono eseguite dal programma assemblatore una volta soltanto, all'atto della generazione del codice oggetto
- **Load-time operations** - sono eseguite dal programma caricatore, all'atto del caricamento in memoria del codice oggetto
- **Run-time operations** - sono eseguite dal programma in linguaggio macchina, all'atto dell'esecuzione delle istruzioni corrispondenti

Esempio:

- **MOVE R0, SOMMA**
- **MOVE**: codice operativo (o OP-CODE)
- **R0** : operando sorgente (registro)
- **SOMMA**: operando destinazione (locazione di memoria il cui indirizzo è rappresentato dal nome simbolico **SOMMA)**
- Nel listato del programma occorre definire tutti i nomi simbolici in modo da creare un'associazione **SIMBOLO → VALORE**
- L'assemblatore traduce l'istruzione assembler in codice binario

Natura degli operandi

- **Operando costante**: (solo operando sorgente)
  - ***Esplicito o immediato***: espresso direttamente nell'istruzione
  - ***Implicito***: è determinato dal codice operativo (**CLR D0** la costante 0 è implicitamente indicata dal codice operativo CLR=CLEAR)

- **Operando memoria**: locazione della memoria centrale, è individuato mediante un indirizzo

- **Operando registro**: (sorgente o destinazione)
  - ***Esplicito***: nelle macchine a registri generali è indicato dall'indice del registro
  - ***Implicito***: è implicitamente espresso dal codice operativo

Natura degli operandi:

- Esempi:
- **ADD #9,R0**
  - il primo operando è immediato (il valore è espresso nell'istruzione), il secondo è il registro **R0**
- **ADD R0,R1**
  - entrambi gli operandi sono registri
- ~~**ADD R0,#9**~~
  - l'operando destinazione non può essere una costante.

Registri interni

L'uso di registri interni produce

- elaborazioni più rapide: l'accesso ai registri interni è molto più veloce dell'accesso in memoria;
- istruzioni più corte: l'operando registro richiede un minor numero di bit.

Poiché non si può operare direttamente su locazioni di memoria, occorre trasferire esplicitamente i dati dalla memoria ai registri e viceversa.

Il linguaggio Assembler del processore MC 68000

L'assemblatore mette a disposizione:

- **codici mnemonici** – forma simbolica delle istruzioni del processor
  - CLR (CLEAR)
  - MOVE (MOVE)
  - BEQ (BRANCH ON EQUAL)
  - ADD (ADD)
- **pseudo codici** - *direttive* per il processo di assemblaggio del programma
  - ORG (ORIGIN)
  - DS (DEFINE STORAGE)
  - DC (DEFINE CONSTANT)
  - END (END ASSEMBLY)

Direttive di assemblaggio: EQU…

Direttive di assemblaggio o **pseudo-operatori:** informazioni utilizzate dall'assemblatore per la traduzione del programma da sorgente ad oggetto.

- Direttiva **EQU**

HOURS EQU 24

comanda dell'assemblatore la sostituzione**,** ovunque compaia nel programma, del simbolo **HOURS** con il valore **24.**

- Direttiva **ORG** (***origin***):

ORG $8000

indica l'indirizzo di memoria a partire dal quale l'assemblatore deve allocare il codice che segue.

Etichetta (***label***): nella traduzione la ***label***, che se presente occupa il primo campo dell'istruzione, viene sostituita con il valore corrispondente all'indirizzo di memoria in cui è contenuta l'istruzione.

Processo di assemblaggio in due passi

L'assemblatore scandisce il programma sorgente due volte:

- Nel **primo passo** costruisce la ***tabella dei simboli***
- Nel **secondo passo** risolve i riferimenti “in avanti”

Riferimento “in avanti”– l’operando compare prima che il suo **valore** sia stato definito

Convenzioni usate dall'assemblatore

- Gli ***spazi tra i campi*** fungono esclusivamente da ***separatori*** (vengono ignorati dall'assemblatore)
- Una linea che inizi con un ***asterisco*** (*) è una linea di ***commento***
- Nelle espressioni assembly, gli argomenti di tipo numerico si intendono espressi
  - in notazione decimale, se non diversamente specificato
  - in ***notazione esadecimale***, se preceduti dal ***simbolo "$"***
- Nell'indicazione degli operandi, il ***simbolo "#"*** denota un ***indirizzamento immediato***

Un programma assembly

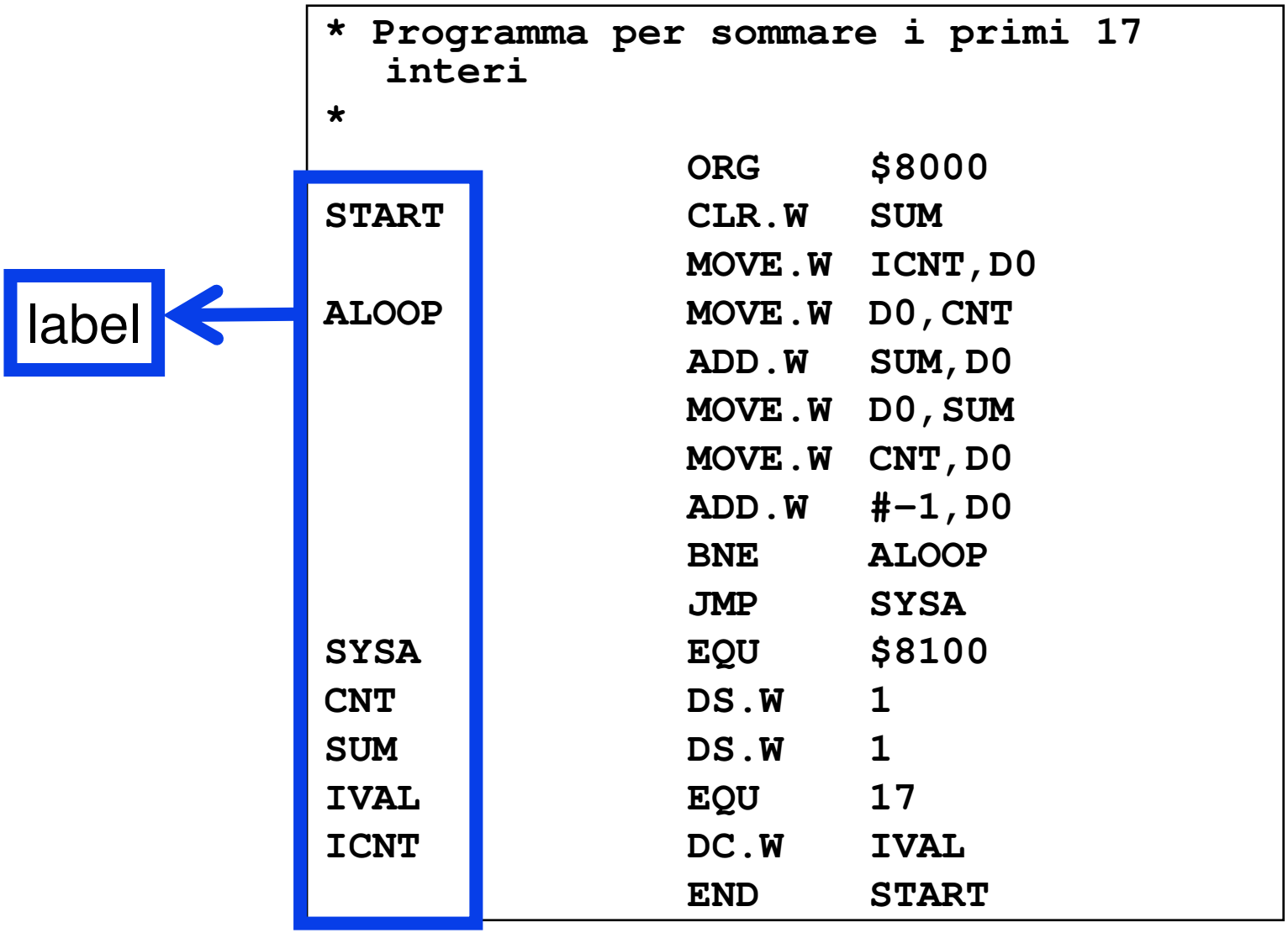
```
* Programma per sommare i primi 17
   interi
*
                    ORG     $8000
START               CLR.W   SUM
                    MOVE.W  ICNT,D0
ALOOP               MOVE.W  D0,CNT
                    ADD.W   SUM,D0
                    MOVE.W  D0,SUM
                    MOVE.W  CNT,D0
                    ADD.W   #-1,D0
                    BNE     ALOOP
                    JMP     SYSA
SYSA                EQU     $8100
CNT                 DS.W    1
SUM                 DS.W    1
IVAL                EQU     17
ICNT                DC.W    IVAL
                    END     START
```

Un programma assembly

```
* Programma per sommare i primi 17
   interi
*
                    ORG     $8000
START               CLR.W   SUM
                    MOVE.W  ICNT,D0
ALOOP               MOVE.W  D0,CNT
                    ADD.W   SUM,D0
                    MOVE.W  D0,SUM
                    MOVE.W  CNT,D0
                    ADD.W   #-1,D0
                    BNE     ALOOP
                    JMP     SYSA
SYSA                EQU     $8100
CNT                 DS.W    1
SUM                 DS.W    1
IVAL                EQU     17
ICNT                DC.W    IVAL
                    END     START
```

commento

Un programma assembly




```
* Programma per sommare i primi 17
   interi
*
                ORG     $8000
START           CLR.W   SUM
                MOVE.W  ICNT,D0
ALOOP           MOVE.W  D0,CNT
                ADD.W   SUM,D0
                MOVE.W  D0,SUM
                MOVE.W  CNT,D0
                ADD.W   #-1,D0
                BNE     ALOOP
                JMP     SYSA
SYSA            EQU     $8100
CNT             DS.W    1
SUM             DS.W    1
IVAL            EQU     17
ICNT            DC.W    IVAL
                END     START
```

Un programma assembly

```
* Programma per sommare i primi 17
   interi
*
                ORG     $8000
START           CLR.W   SUM
                MOVE.W  ICNT,D0
ALOOP           MOVE.W  D0,CNT
                ADD.W   SUM,D0
                MOVE.W  D0,SUM
                MOVE.W  CNT,D0
                ADD.W   #-1,D0
                BNE     ALOOP
                JMP     SYSA
SYSA            EQU     $8100
CNT             DS.W    1
SUM             DS.W    1
IVAL            EQU     17
ICNT            DC.W    IVAL
                END     START
```

codici operativi
e pseudo codici

Un programma assembly

```
* Programma per sommare i primi 17
   interi
*
                    ORG     $8000
START               CLR.W   SUM
                    MOVE.W  ICNT,D0
ALOOP               MOVE.W  D0,CNT
                    ADD.W   SUM,D0
                    MOVE.W  D0,SUM
                    MOVE.W  CNT,D0
                    ADD.W   #-1,D0
                    BNE     ALOOP
                    JMP     SYSA
SYSA                EQU     $8100
CNT                 DS.W    1
SUM                 DS.W    1
IVAL                EQU     17
ICNT                DC.W    IVAL
                    END     START
```

operandi

Il lavoro dell'assemblatore

```
00000000                             1  * Programma per sommare i primi 17
   interi
00000000                             2  *
00008000                             3          ORG   $8000
00008000  4279 00008032              4  START   CLR.W       SUM
00008006  3039 00008034              5          MOVE.W      ICNT,D0
0000800C  33C0 00008030              6  ALOOP   MOVE.W      D0,CNT
00008012  D079 00008032              7          ADD.W       SUM,D0
00008018  33C0 00008032              8          MOVE.W      D0,SUM
0000801E  3039 00008030              9          MOVE.W      CNT,D0
00008024  0640 FFFF                 10          ADD.W       #-1,D0
00008028  66E2                      11          BNE         ALOOP
0000802A  4EF9 00008100             12          JMP         SYSA
00008030  =00008100                 13  SYSA    EQU         $8100
00008030                            14  CNT     DS.W        1
00008032                            15  SUM     DS.W        1
00008034  =00000011                 16  IVAL    EQU         17
00008034  0011                      17  ICNT    DC.W        IVAL
00008036                            18          END         START
```

Il lavoro dell'assemblatore

La tabella dei simboli:

Symbol Table

ALOOP	800C	CNT	8030	IVAL	0011
START	8000	SUM	8032	ICNT	8034
SYSA	8100				

Il Program Location Counter (PLC)

PLC (variabile interna dell'assemblatore):

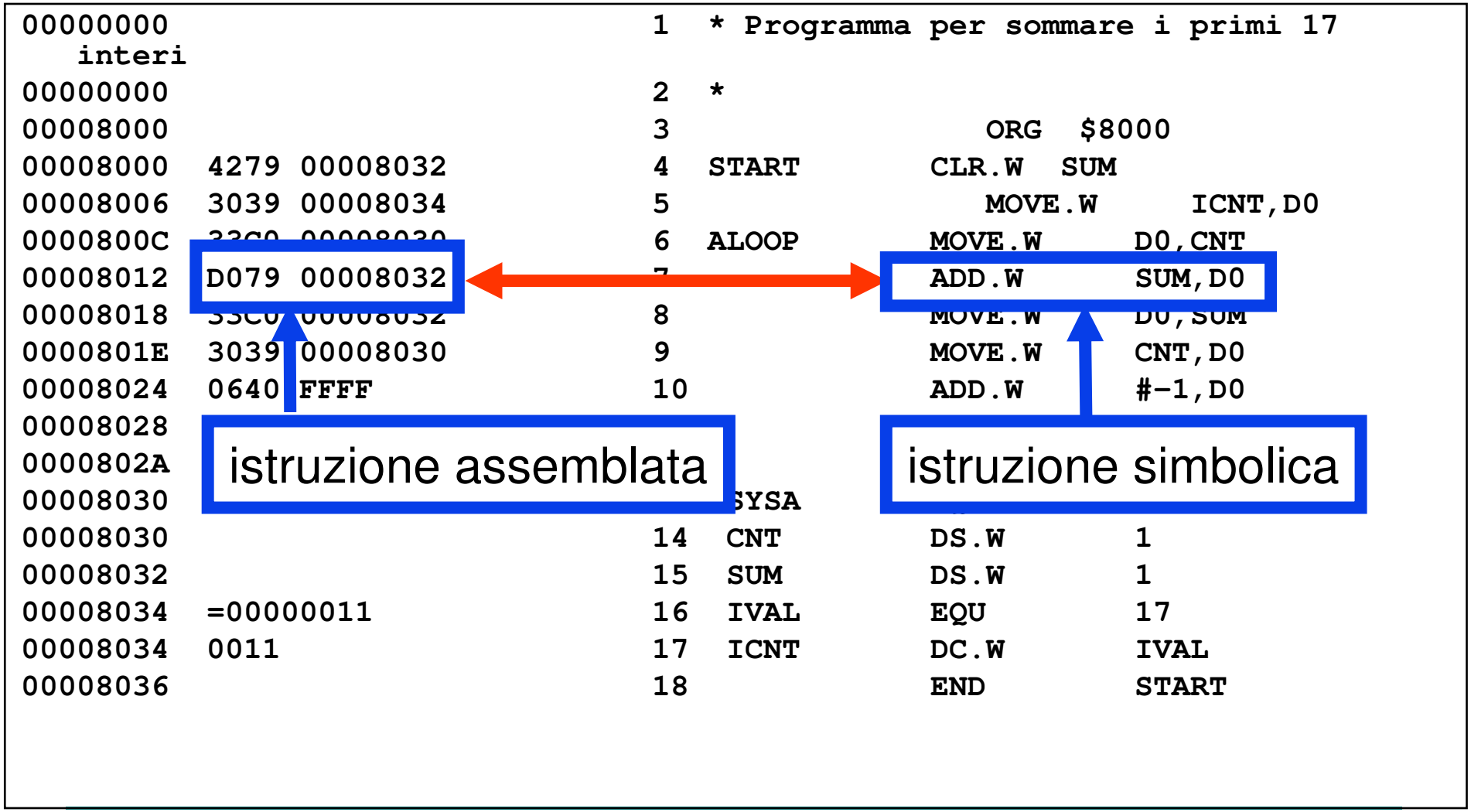
- Punta alla locazione di memoria in cui andrà caricata l'istruzione assemblata
- Viene inizializzata dallo pseudo-operatore "origin" (ORG)
- Durante il processo di assemblaggio, il suo valore è aggiornato in funzione del codice operativo o pseudo-codice
- E' possibile, all'interno di un programma, fare riferimento al suo valore corrente mediante il simbolo " ***** "

Il lavoro dell'assemblatore

```
00000000                          1  * Programma per sommare i primi 17
   interi
00000000                          2  *
00008000                          3              ORG   $8000
00008000  4279 00008032           4  START       CLR.W  SUM
00008006  3039 00008034           5              MOVE.W     ICNT,D0
0000800C  33C0 00008030           6  ALOOP       MOVE.W     D0,CNT
00008012  D079 00008032           7              ADD.W      SUM,D0
00008018  33C0 00008032                          D0,SUM
0000801E  3039 00008030                          CNT,D0
00008024  0640 FFFF              10              ADD.W      #-1,D0
00008028  66E2                   11              BNE        ALOOP
0000802A  4EF9 00008100          12              JMP        SYSA
00008030  =00008100              13  SYSA        EQU        $8100
00008030                         14  CNT         DS.W       1
00008032                         15  SUM         DS.W       1
00008034  =00000011              16  IVAL        EQU        17
00008034  0011                   17  ICNT        DC.W       IVAL
00008036                         18              END        START
```

valore corrente del PLC

Il lavoro dell'assemblatore



Pseudo-operatori ORG e END

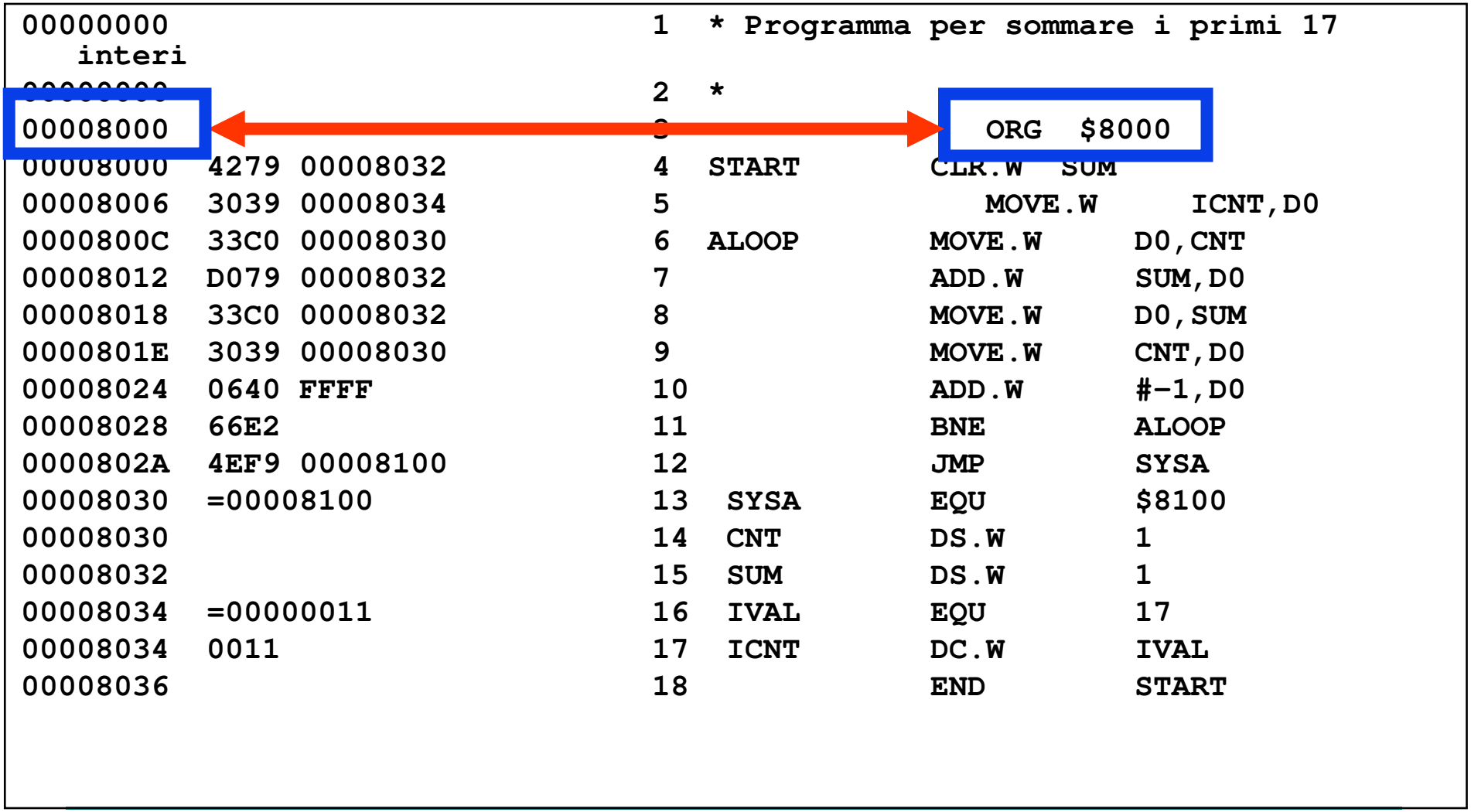
ORG viene usato per inizializzare il PLC

Sintassi: ORG $HEXADDR

END viene usato per terminare il processo di assemblaggio e saltare all'entry point del programma

Sintassi: END LABEL

Il lavoro dell'assemblatore



```
00000000                          1   * Programma per sommare i primi 17
   interi
00000000                          2   *
00008000                          3           ORG   $8000
00008000  4279 00008032           4   START   CLR.W   SUM
00008006  3039 00008034           5           MOVE.W      ICNT,D0
0000800C  33C0 00008030           6   ALOOP   MOVE.W      D0,CNT
00008012  D079 00008032           7           ADD.W       SUM,D0
00008018  33C0 00008032           8           MOVE.W      D0,SUM
0000801E  3039 00008030           9           MOVE.W      CNT,D0
00008024  0640 FFFF               10          ADD.W       #-1,D0
00008028  66E2                    11          BNE         ALOOP
0000802A  4EF9 00008100           12          JMP         SYSA
00008030  =00008100               13  SYSA    EQU         $8100
00008030                          14  CNT     DS.W        1
00008032                          15  SUM     DS.W        1
00008034  =00000011               16  IVAL    EQU         17
00008034  0011                    17  ICNT    DC.W        IVAL
00008036                          18          END         START
```

Il lavoro dell'assemblatore

```
00000000                             1  * Programma per sommare i primi 17
   interi
00000000                             2  *
00008000                             3           ORG   $8000
00008000  4279 00008032              4  START    CLR.W  SUM
00008006  3039 00008034              5           MOVE.W ICNT,D0
0000800C  33C0 00008030              6  ALOOP    MOVE.W D0,CNT
00008012  D079 00008032              7           ADD.W  SUM,D0
00008018  33C0 00008032              8           MOVE.W D0,SUM
0000801E  3039 00008030              9           MOVE.W CNT,D0
00008024  0640 FFFF                  10          ADD.W  #-1,D0
00008028  66E2                       11          BNE    ALOOP
0000802A  4EF9 00008100              12          JMP    SYSA
00008030  =00008100                  13 SYSA     EQU    $8100
00008030                             14 CNT      DS.W   1
00008032                             15 SUM      DS.W   1
00008034  =00000011                  16 IVAL     EQU    17
00008034  0011                       17 ICNT     DC.W   IVAL
00008036                             18          END    START
```

Pseudo-operatori EQU - DS - DC

EQU viene usato per stabilire un'identità

Sintassi: LABEL EQU VALUE

DS (Define Storage) viene usato per incrementare il PLC in modo da riservare spazio di memoria per una variabile

Sintassi: LABEL DS.W AMOUNT

DC (Define Constant) viene usato per inizializzare il valore di una variabile

Sintassi: LABEL DC.B VALUE

Il lavoro dell'assemblatore

```
00000000                                1  * Programma per sommare i
   primi 17 interi
00000000                                2  *
00008000                                3           ORG   $8000
00008000  4279 00008032                 4  START       CLR.W   SUM
00008006  3039 00008034                 5           MOVE.W      ICNT,D0
0000800C  33C0 00008030                 6  ALOOP       MOVE.W      D0,CNT
00008012  D079 00008032                 7              ADD.W       SUM,D0
00008018  33C0 00008032                 8              MOVE.W      D0,SUM
0000801E  3039 00008030                 9              MOVE.W      CNT,D0
00008024  0640 FFFF                    10              ADD.W       #-1,D0
00008028  66E2                         11              BNE         ALOOP
0000802A  4EF9 00008100                12              JMP         SYSA
00008030  =00008100                    13  SYSA        EQU         $8100
00008030                               14  CNT         DS.W        1
00008032                               15  SUM         DS.W        1
00008034  =00000011                    16  IVAL        EQU         17
00008034  0011                         17  ICNT        DC.W        IVAL
00008036                               18              END         START
```

Il lavoro dell'assemblatore

```
00000000                                1  * Programma per sommare i
  primi 17 interi
00000000                                2  *
00008000                                3           ORG   $8000
00008000  4279 00008032                 4  START       CLR.W   SUM
00008006  3039 00008034                 5           MOVE.W      ICNT,D0
0000800C  33C0 00008030                 6  ALOOP       MOVE.W      D0,CNT
00008012  D079 00008032                 7              ADD.W       SUM,D0
00008018  33C0 00008032                 8              MOVE.W      D0,SUM
0000801E  3039 00008030                 9              MOVE.W      CNT,D0
00008024  0640 FFFF                    10              ADD.W       #-1,D0
00008028  66E2                         11              BNE         ALOOP
0000802A  4EF9 00008100                12              JMP         SYSA
00008030  =00008100                    13    SYSA      EQU         $8100
00008030                               14    CNT       DS.W        1
00008032                               15    SUM       DS.W        1
00008034  =00000011                    16    IVAL      EQU         17
00008034  0011                         17    ICNT      DC.W        IVAL
00008036                               18              END         START
```

Il lavoro dell'assemblatore

```
00000000                             1  * Programma per sommare i
   primi 17 interi
00000000                             2  *
00008000                             3           ORG   $8000
00008000  4279 00008032              4  START       CLR.W   SUM
00008006  3039 00008034              5           MOVE.W      ICNT,D0
0000800C  33C0 00008030              6  ALOOP       MOVE.W      D0,CNT
00008012  D079 0000803[illegible]                   ADD.W       SUM,D0
00008018  33C0 0000[illegible]                      MOVE.W      D0,SUM
0000801E  3039 0000[illegible]                      MOVE.W      CNT,D0
00008024  0640 FFFF                                 ADD.W       #-1,D0
00008028  66E2                                      BNE         ALOOP
0000802A  4EF9 00008100             12              JMP         SYSA
00008030  =00008100                 13  SYSA        EQU         $8100
00008030                            14  CNT         DS.W        1
00008032                            15  SUM         DS.W        1
00008034  =00000011                 16  IVAL        EQU         17
00008034  0011                      17  ICNT        DC.W        IVAL
00008036                            18              END         START
```

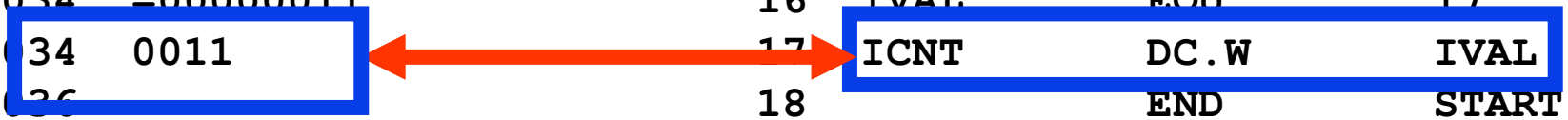
$11_{16} = 17_{10}$

Il lavoro dell'assemblatore

```
00000000                             1  * Programma per sommare i
  primi 17 interi
00000000                             2  *
00008000                             3          ORG   $8000
00008000  4279 00008032              4  START       CLR.W   SUM
00008006  3039 00008034              5          MOVE.W     ICNT,D0
0000800C  33C0 00008030              6  ALOOP       MOVE.W     D0,CNT
00008012  D079 00008032              7              ADD.W      SUM,D0
00008018  33C0 00008032              8              MOVE.W     D0,SUM
0000801E  3039 00008030              9              MOVE.W     CNT,D0
00008024  0640 FFFF                 10              ADD.W      #-1,D0
00008028  66E2                      11              BNE        ALOOP
0000802A  4EF9 00008100             12              JMP        SYSA
0000802E  =00008100                 13  SYSA        EQU        $8100
00008030                            14  CNT         DS.W       1
00008032                            15  SUM         DS.W       1
00008034  =00000011                 16  IVAL        EQU        17
00008034  0011                      17  ICNT        DC.W       IVAL
00008036                            18              END        START
```

Il lavoro dell'assemblatore

```
00000000                             1  * Programma per sommare i
  primi 17 interi
00000000                             2  *
00008000                             3           ORG   $8000
00008000  4279 00008032              4  START       CLR.W   SUM
00008006  3039 00008034              5           MOVE.W     ICNT,D0
0000800C  33C0 00008030              6  ALOOP       MOVE.W     D0,CNT
00008012  D079 00008032              7              ADD.W      SUM,D0
00008018  33C0 00008032              8              MOVE.W     D0,SUM
0000801E  3039 00008030              9              MOVE.W     CNT,D0
00008024  0640 FFFF                 10              ADD.W      #-1,D0
00008028  66E2                      11              BNE        ALOOP
0000802A  4EF9 00008100             12              JMP        SYSA
00008030  =00008100                 13  SYSA        EQU        $8100
00008030                            14  CNT         DS.W       1
00008032                            15  SUM         DS.W       1
00008034  =00000011                 16  IVAL        EQU        17
00008034  0011                      17  ICNT        DC.W       IVAL
00008036                            18              END        START
```



Cosa fa questo programma?

```
00000000                             1  * Programma per sommare i primi 17
   interi
00000000                             2  *
00008000                             3              ORG   $8000
00008000  4279 00008032              4  START       CLR.W     SUM
00008006  3039 00008034              5              MOVE.W    ICNT,D0
0000800C  33C0 00008030              6  ALOOP       MOVE.W    D0,CNT
00008012  D079 00008032              7              ADD.W     SUM,D0
00008018  33C0 00008032              8              MOVE.W    D0,SUM
0000801E  3039 00008030              9              MOVE.W    CNT,D0
00008024  0640 FFFF                 10              ADD.W     #-1,D0
00008028  66E2                      11              BNE       ALOOP
0000802A  4EF9 00008100             12              JMP       SYSA
00008030  =00008100                 13  SYSA        EQU       $8100
00008030                            14  CNT         DS.W      1
00008032                            15  SUM         DS.W      1
00008034  =00000011                 16  IVAL        EQU       17
00008034  0011                      17  ICNT        DC.W      IVAL
00008036                            18              END       START
```