Il processore 68000

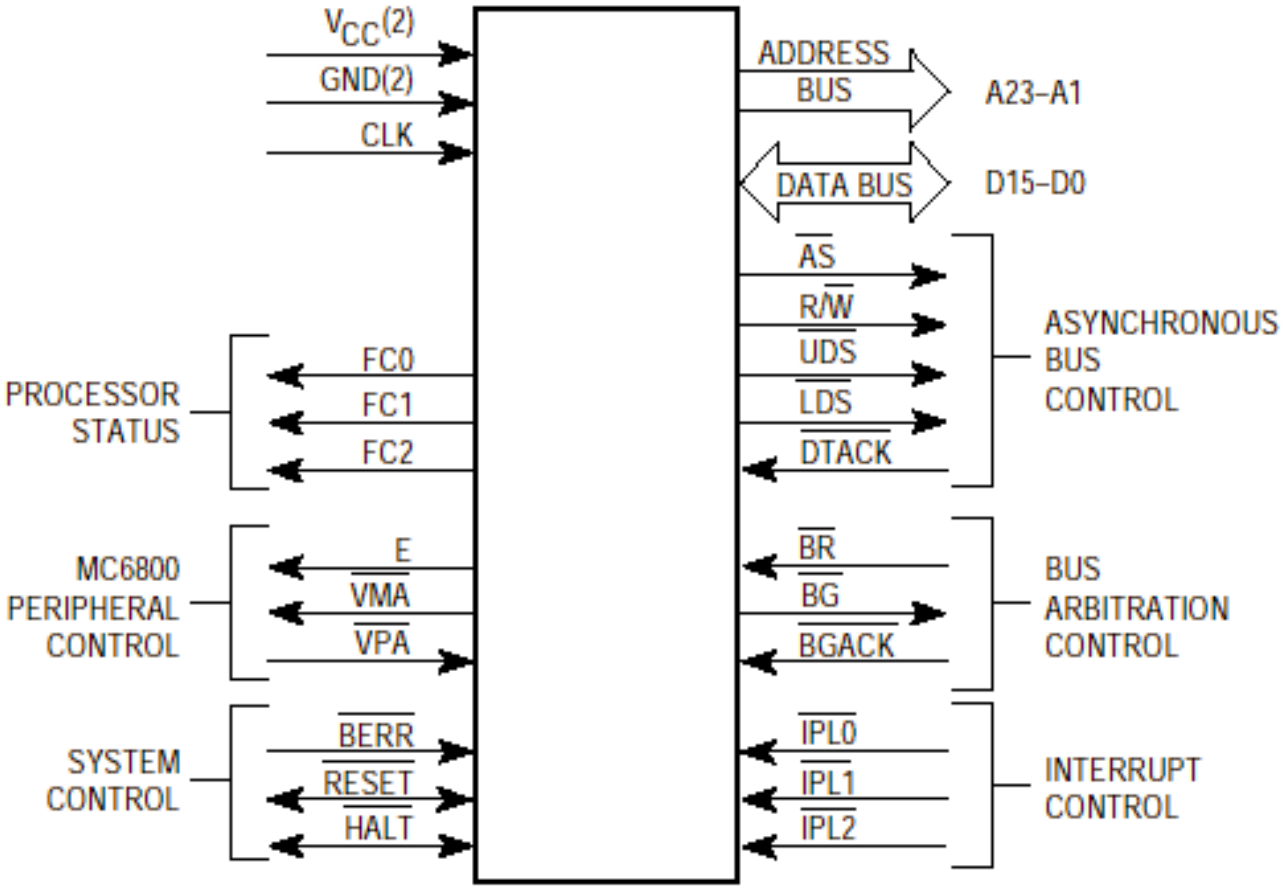


Figure 3-1. Input and Output Signals (MC68000, MC68HC000 and MC68010)

Segnali del processore

- VCC, GND: alimentazione e massa
- CLK: segnale di clock
- A23, A1: bus indirizzi
- D0, D15: bus dati
- FC0/1/2: indicano se l'indirizzo fa riferimento a un dato o programma, se di utente o di supervisore

Gestione periferiche

- E: segnale di sincronizzazione con le periferiche, indipendente dal clock
- VPA: avvisa che l'indirizzo sul bus fa riferimento ad una periferica
- VMA: indirizzo di memoria

Arbitraggio bus

- BR: bus request

  richiede di essere master del bus
- BG: bus grant

  concede il controllo del bus
- ABG: Ack bus grant

  accetta il controllo del bus

Segnali di gestione

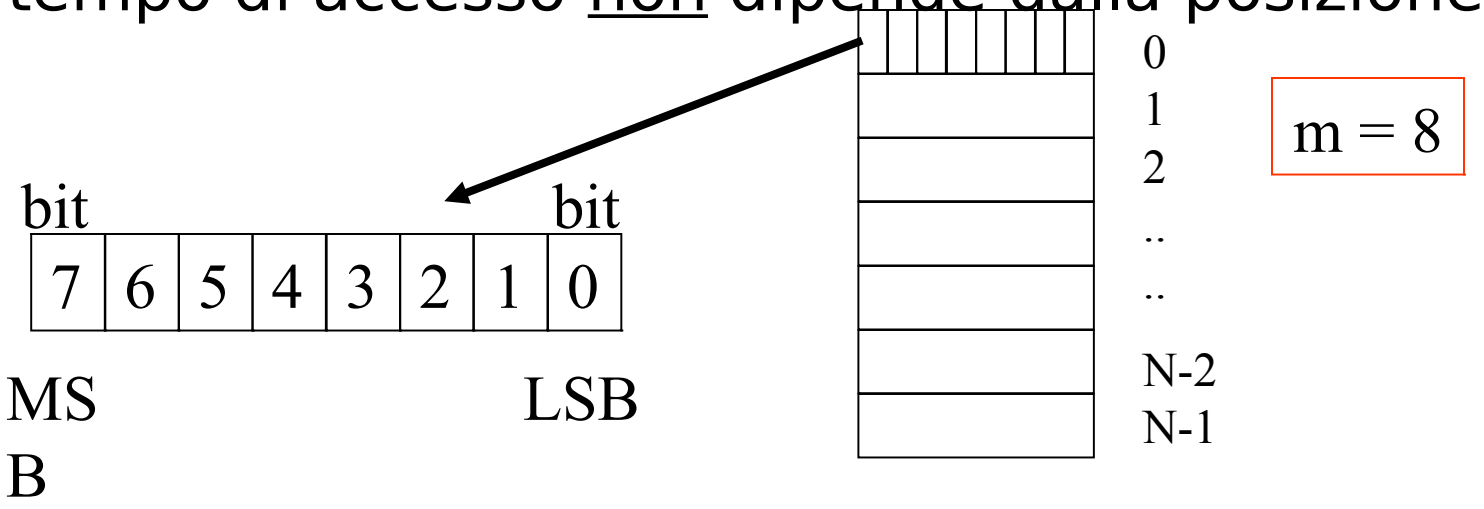
- BERR: comunica al processore un errore sul bus
- RESET: resetta il processore
- ALT: ferma il processore

Segnali di comunicazione

- AS: indirizzo valido
- UDS,LDS: dice quale parte del bus dati è valida
- R/W dice se il comando e di scrittura o lettura
- DTACK: operazione terminata

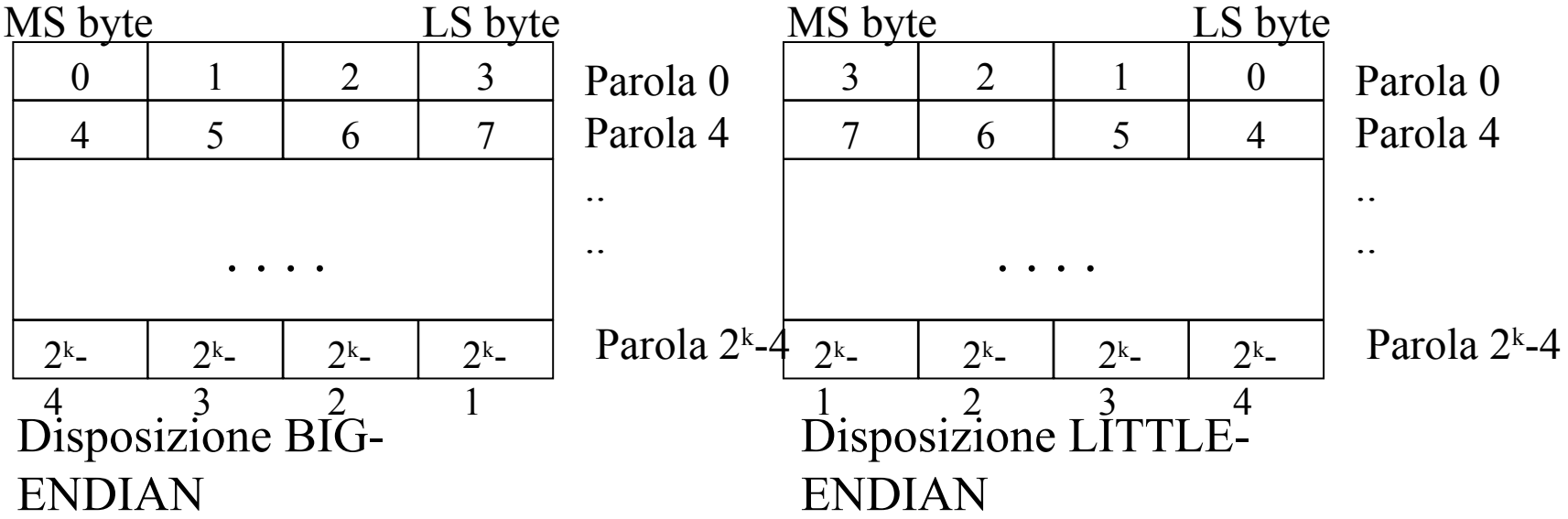
La memoria centrale del M68000

- La memoria centrale di un computer è organizzata come un array di stringhe di bit di lunghezza m, dette *parole* o *word* (m = LUNGHEZZA DI PAROLA)
- Gli m bit di una parola sono accessibili dal processore (in lettura/scrittura) mediante un'unica operazione
- Ogni parola è individuata da un *indirizzo*, cioè un intero compreso tra 0 e N-1 (SPAZIO DI INDIRIZZAMENTO), con $N = 2^c$
- La memoria centrale è *ad accesso casuale* (RAM) cioè il tempo di accesso non dipende dalla posizione del dato



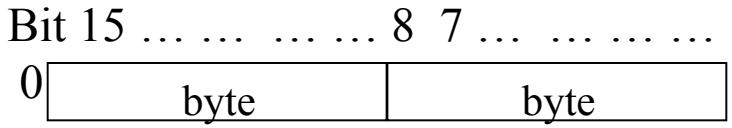
Organizzazione della memoria: indirizzamento di parola

- I processori “a parola” accedono alla memoria con un parallelismo di 16 bit, 32 bit o 64 bit
- Talora, la memoria è *byte-addressable*, cioè la più piccola unità di memoria indirizzabile è il byte
- I byte costituenti una word possono essere disposti in due modi alternativi: big endian e little endian (anche detti inverso e diretto, rispettivamente)



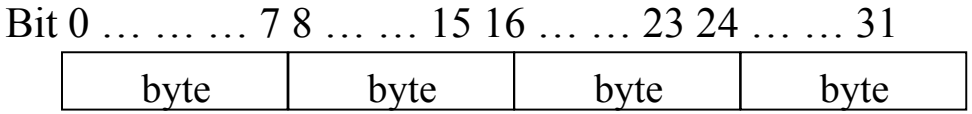
Organizzazione della memoria

Identificazione dei bit in una word



- b_0 = LSBit
- Motorola 68000, Intel 8086

oppure



- b_0 = MSBit
- PowerPC

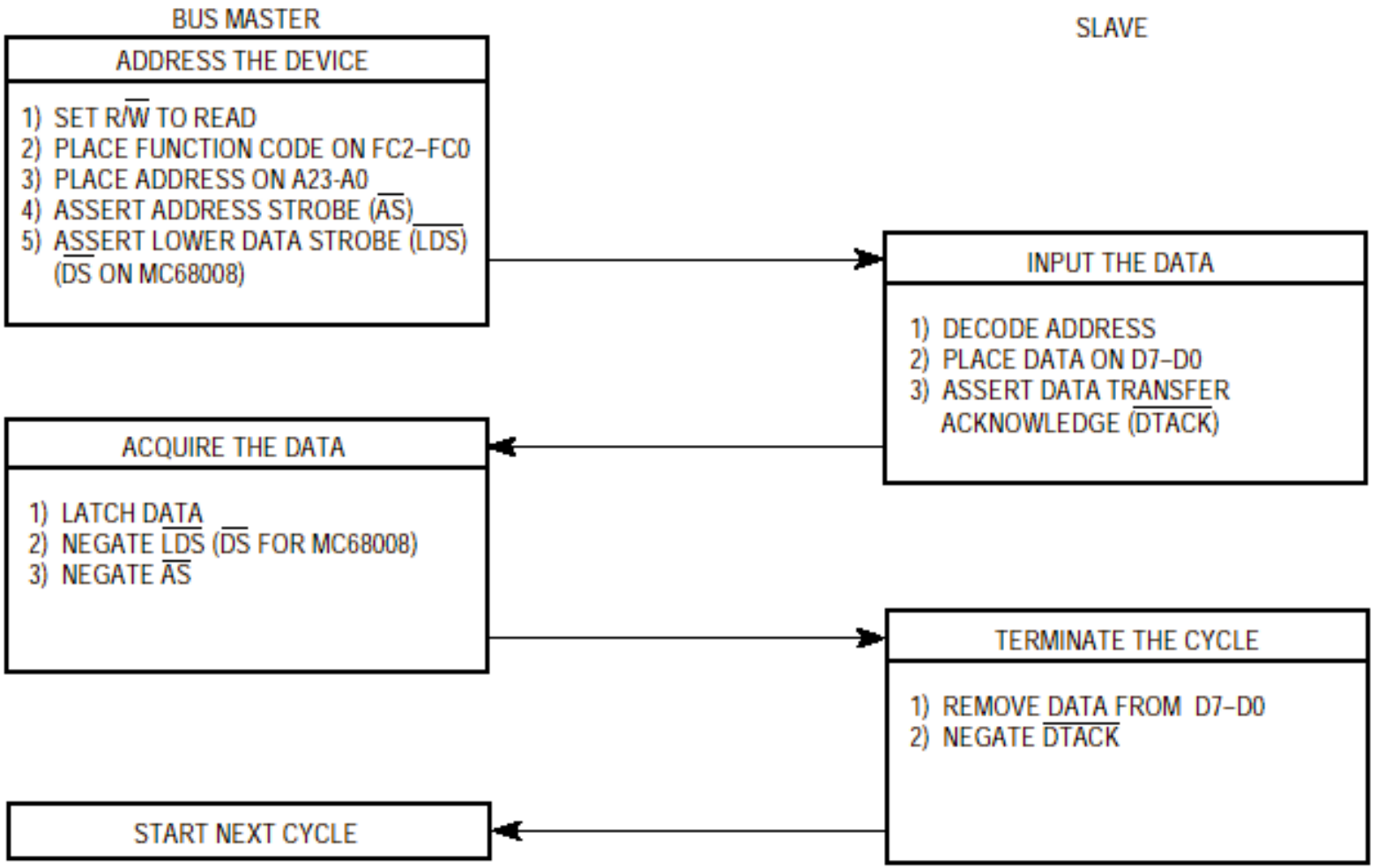
Memoria: parole allineate e non allineate (*)

- Per un processore a 16 bit, una parola che inizia ad un indirizzo pari si dice “allineata sul limite di parola”
- Tipicamente, un processore è in grado di accedere ai byte che costituiscono una parola allineata mediante una sola operazione di lettura (*Read* o *Fetch*)
- Il processore 8086 consente l’utilizzo di parole non allineate, cioè parole che iniziano ad un indirizzo dispari, ma in tal caso sono necessari 2 accessi in memoria
- Il processore 68000 NON consente parole non allineate

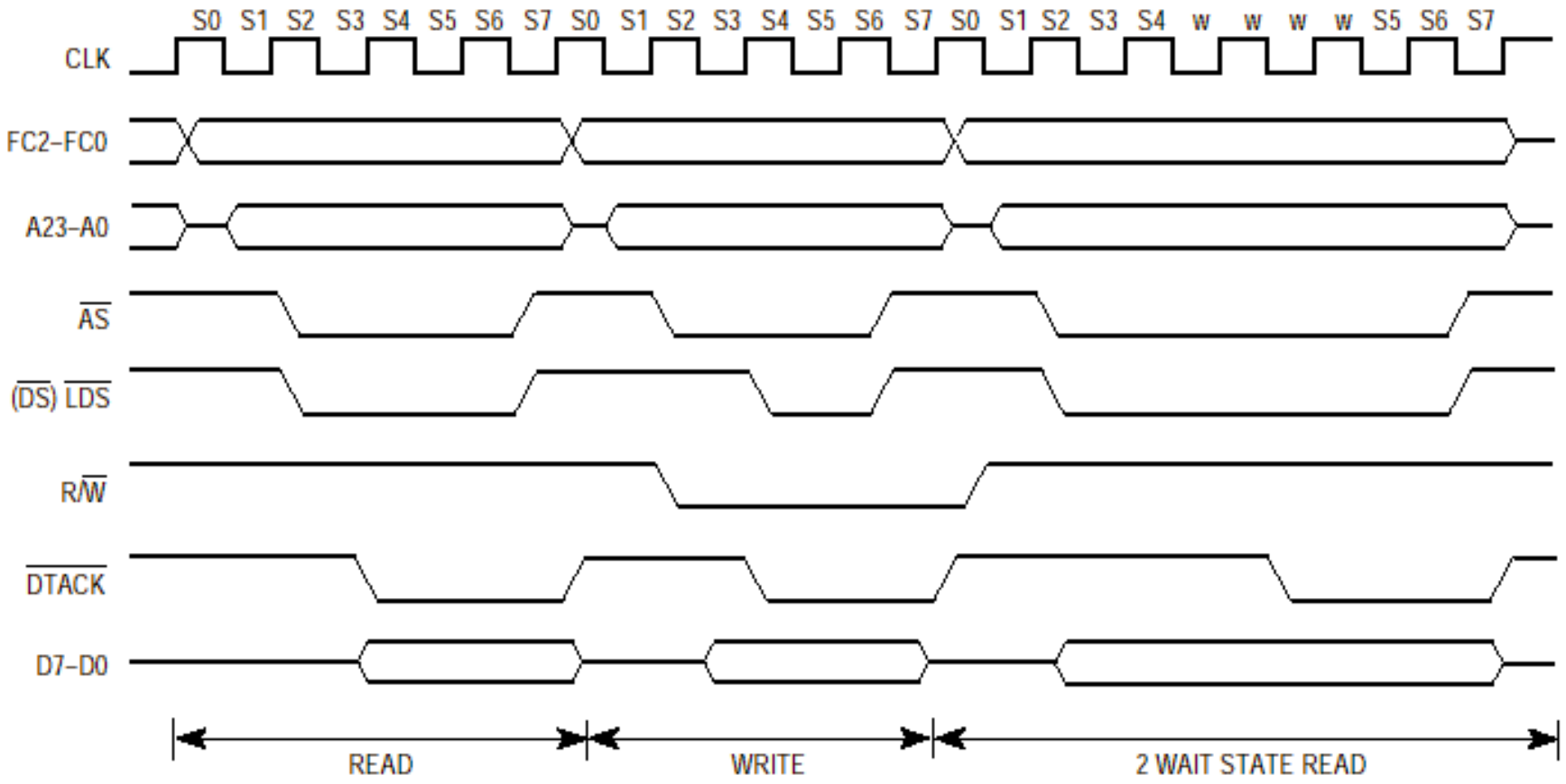
(X pari)

Parola non allineata sul limite di parola

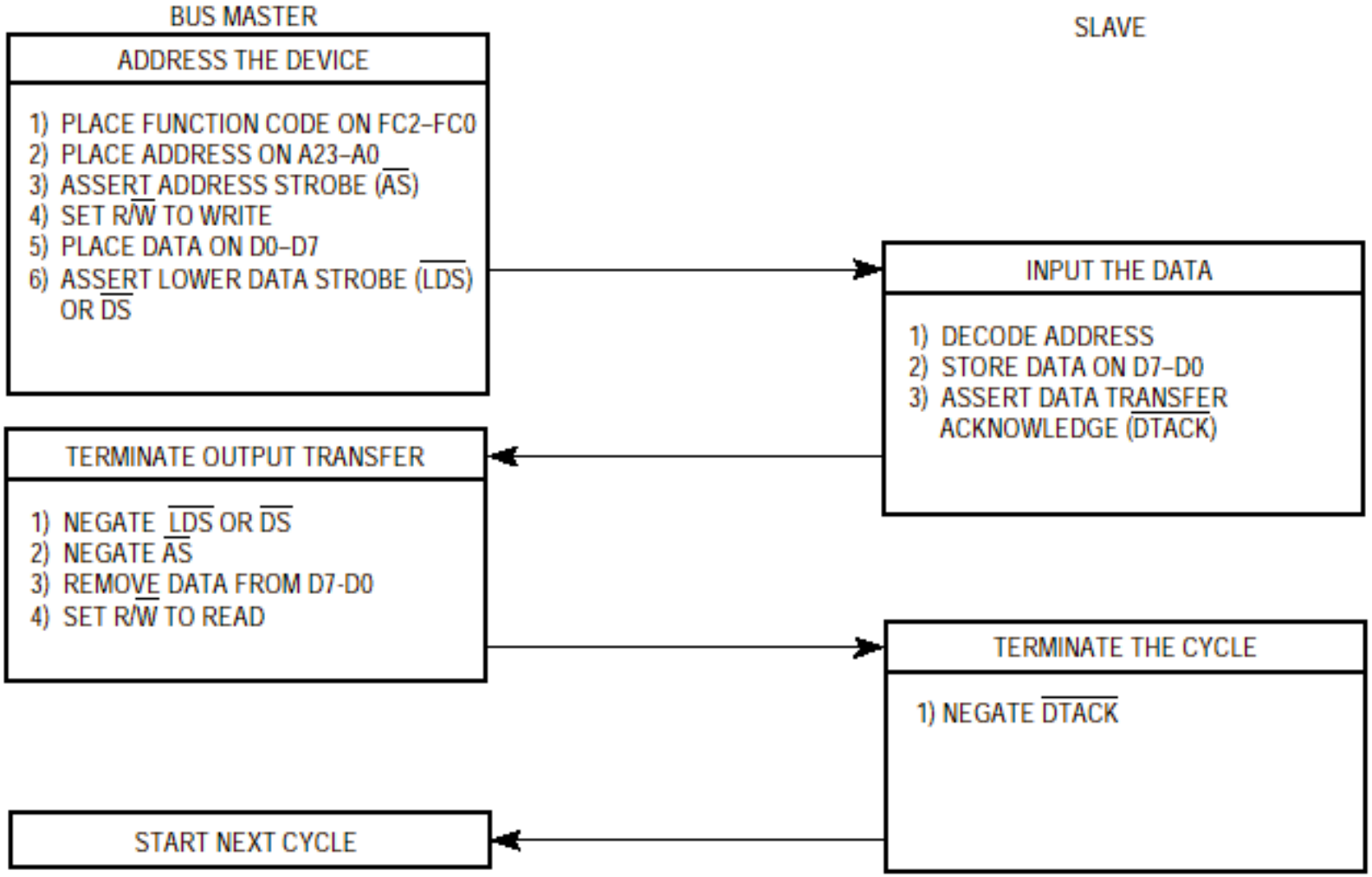
Read in memoria di un byte



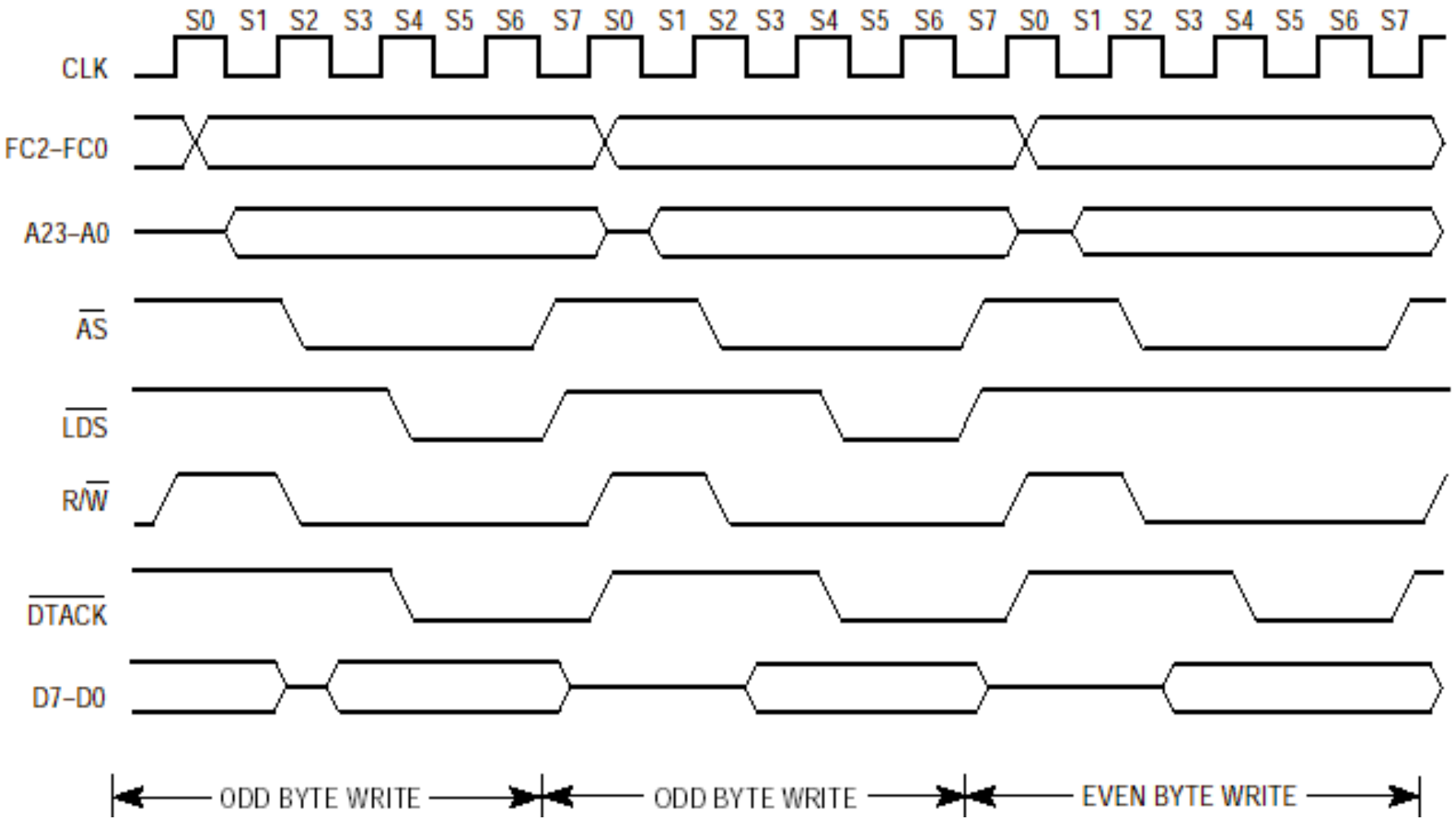
Read in memoria di un byte



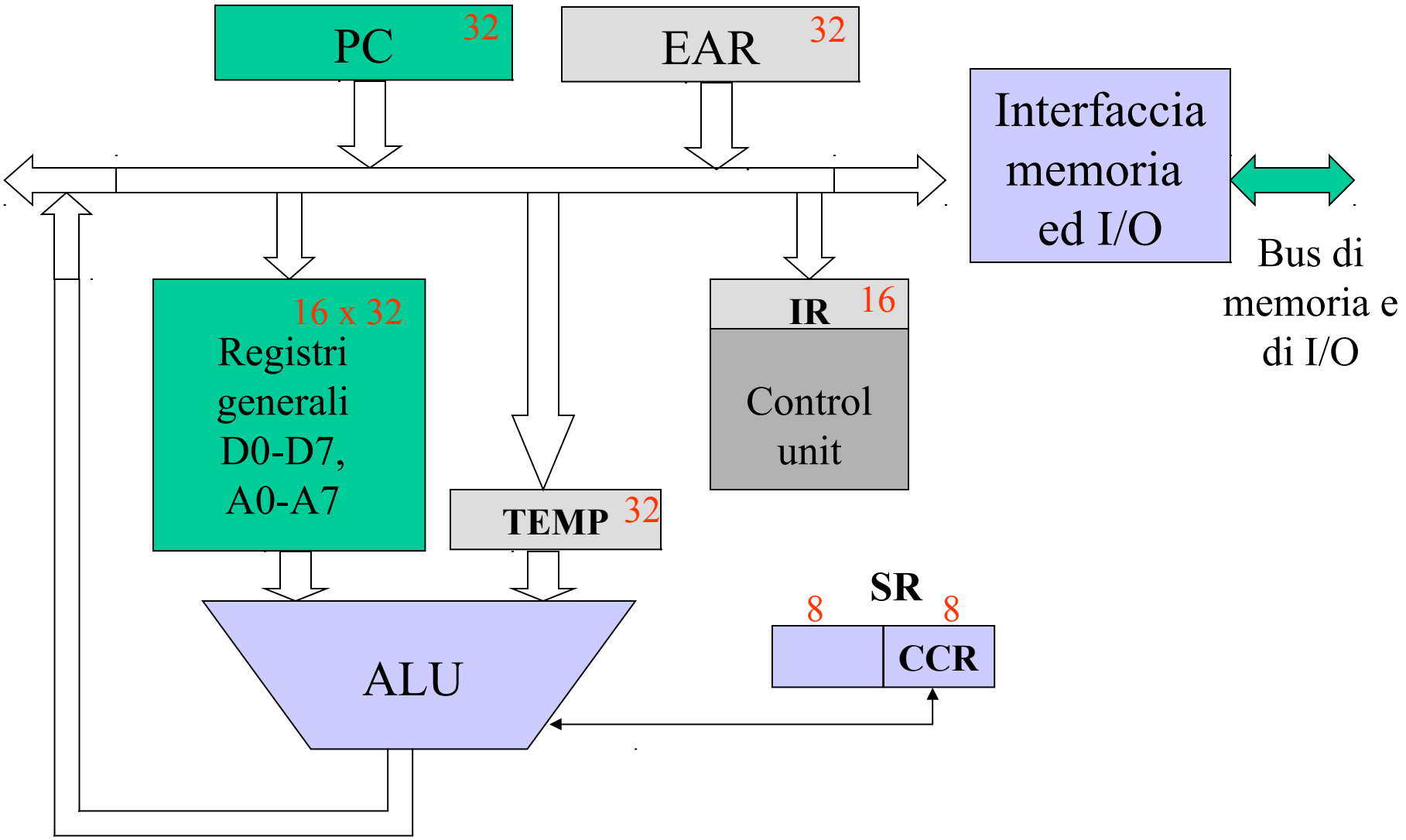
Write in memoria di un byte



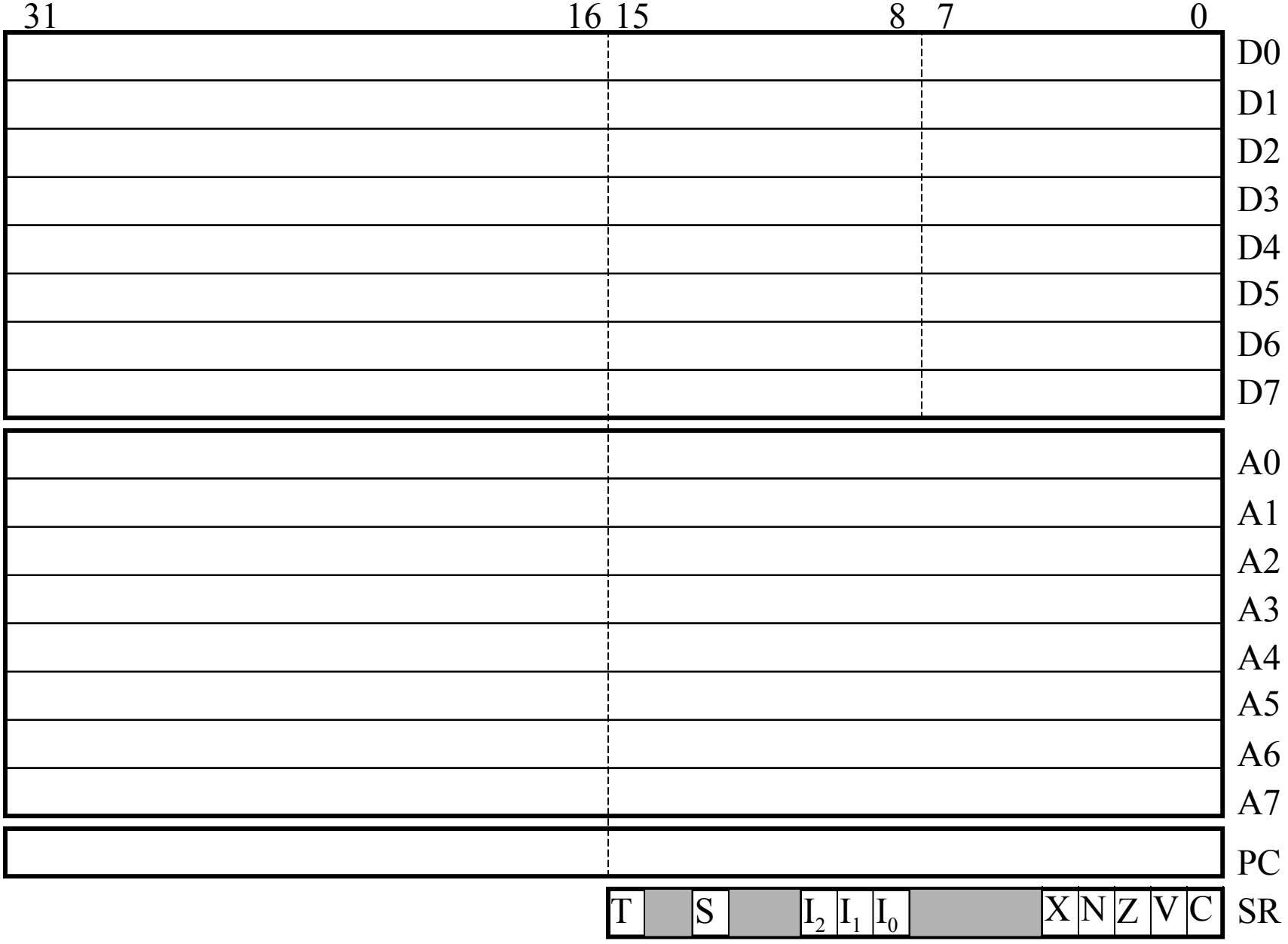
Write in memoria di un byte



Architettura di un processore reale
MC 68000



MC68000: modello di programmazione



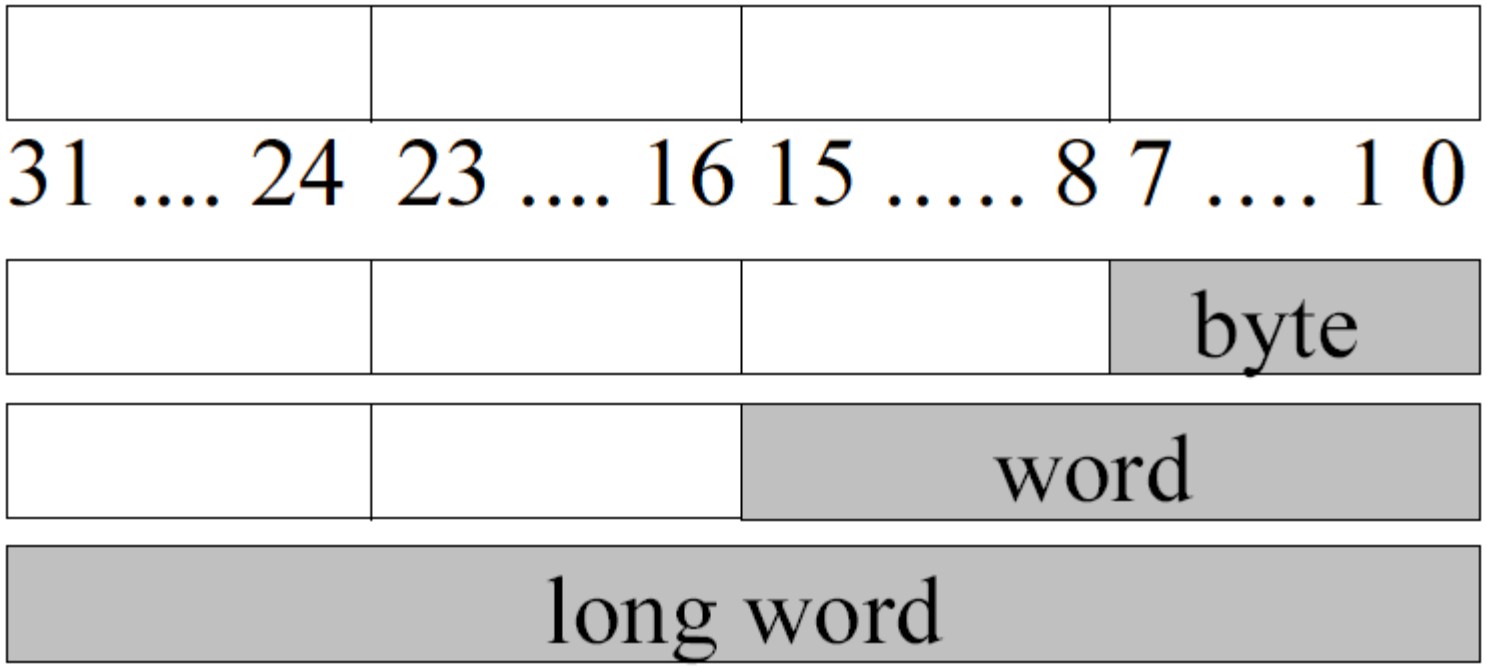
Caratteristiche del processore MC68000

- Dati
  - All'esterno:

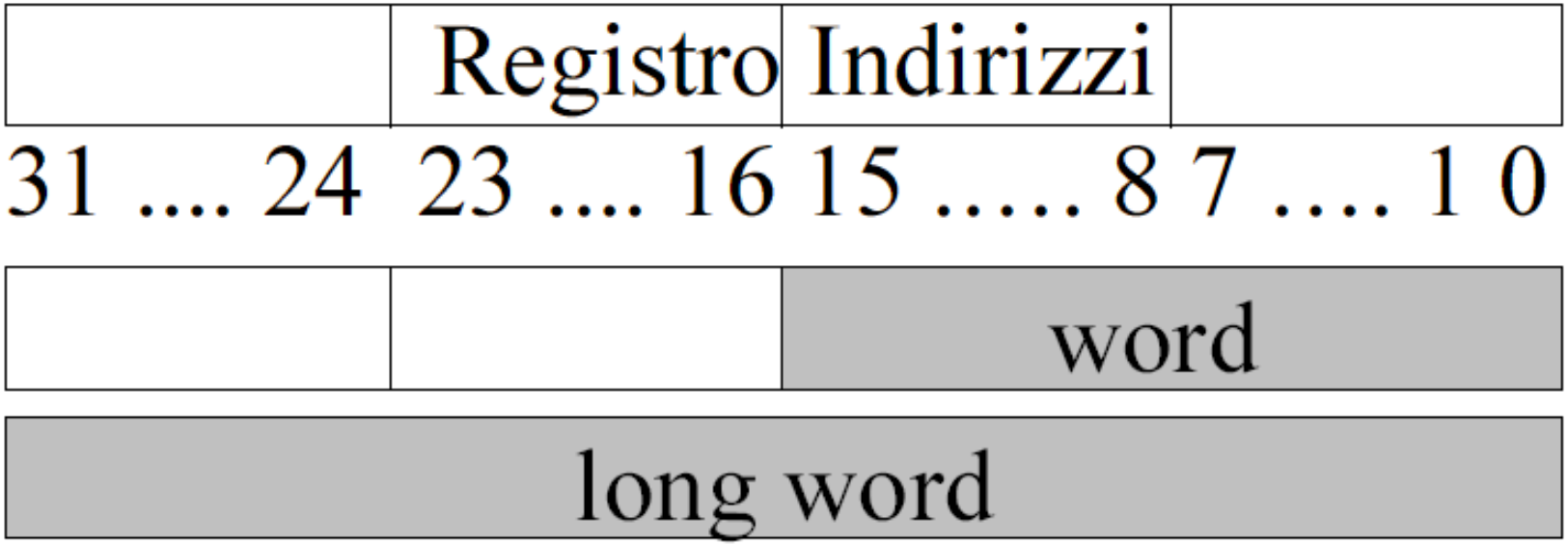
    parola di 16 bit (16 pin per i dati)
  - All'interno:

    registri di 32 bit
- Indirizzi
  - di 24 bit (spazio di indirizzamento 2^{24} = 16M)
  - parole di 16 bit, ognuna costituita da due byte con indirizzi distinti (*memoria byte addressable*)
  - una parola **deve** essere allineata ad un indirizzo pari (*even boundary*)
  - convenzione big-endian

Registri dato D0,D7

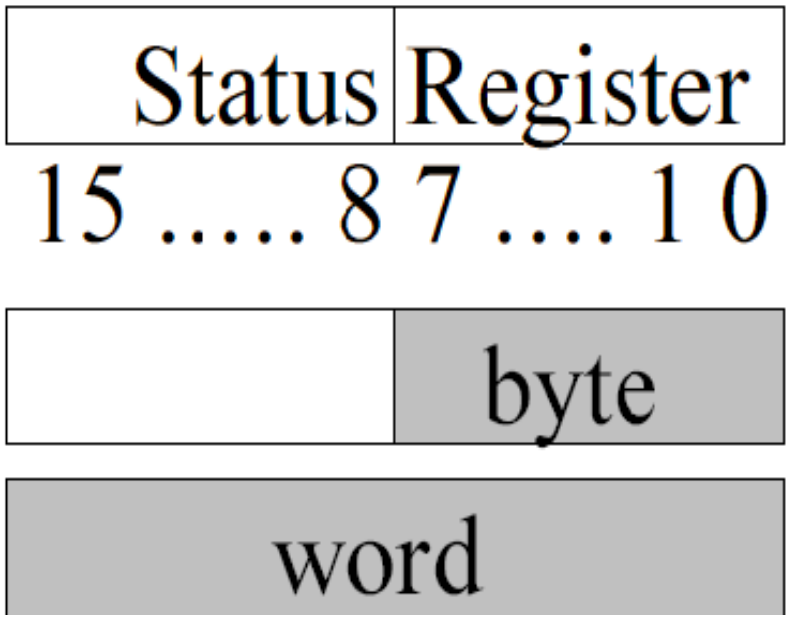


Registri indirizzo A0-A7

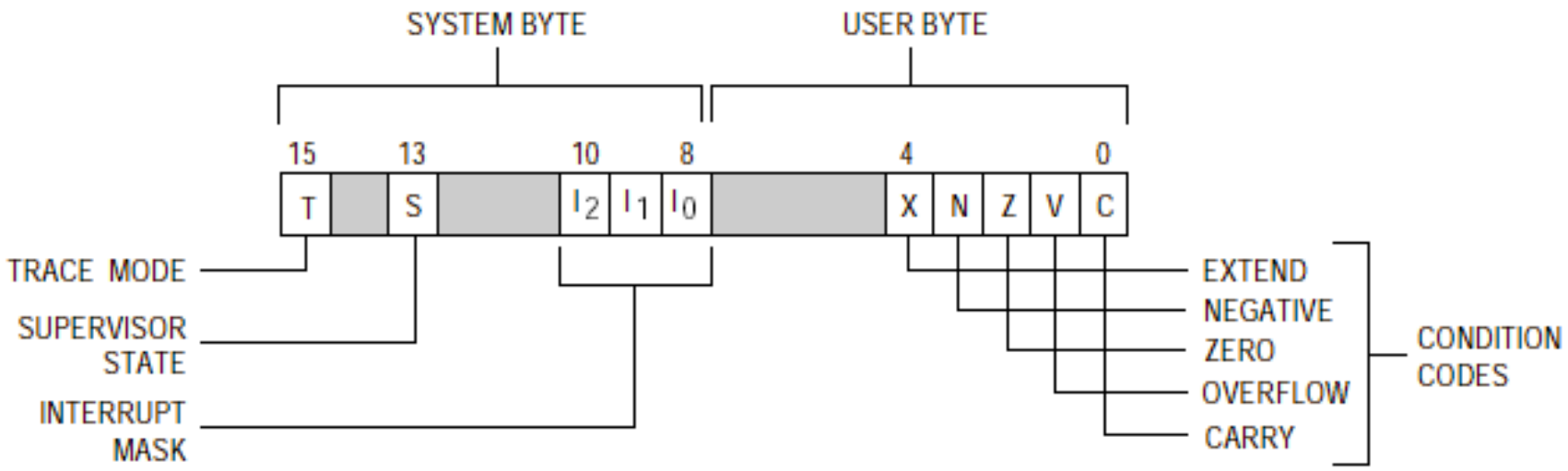


Registro di stato

• 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.



Registro di stato



Linguaggio macchina ed assemblativo

- Il processore preleva dalla memoria le istruzioni
- Le istruzioni sono codificate in binario, come stringhe di bit
- L'insieme delle istruzioni che il processore è in grado di eseguire, insieme con la loro codifica in binario, costituisce il linguaggio macchina del processore
- Per semplificare la scrittura dei programmi, i programmi in linguaggio macchina sono espressi in *linguaggio assemblativo* e poi tradotti in stringhe di bit da un *assemblatore*
- Il linguaggio assemblativo rappresenta le istruzioni del processore mediante nomi mnemonici (*opcodes*) e consente l'utilizzo di simboli per rappresentare indirizzi di memoria e costanti
- Esempio: l'istruzione "somma 3 al contenuto del registro D1",

in linguaggio macchina potrebbe essere:

```
0101 011 000000 001
```

ed in linguaggio assemblativo:

```
ADD #ADJ,D1
```

dove il simbolo `ADJ` è una costante precedentemente posta uguale a 3

Esempi di istruzioni assembler

- istruzioni a tre indirizzi:

  OPERAZIONE Sorgente1, Sorgente2, Destinazione

Es: ADD A,B,C con A,B,C indirizzi di word in memoria,
che corrisponde a: C ← [A] + [B]

- istruzioni a due indirizzi:

  OPERAZIONE Sorgente, Destinazione

Es: ADD A,B con A,B indirizzi di word in memoria,
che corrisponde a: B ← [A] + [B]
oppure MOV A,B che corrisponde a B ← [A]

Esempio – Assembly X86 a 32 bit

```
DES_std_crypt:
    movl 4(%esp),%edx
    pushl %ebx
    movl DES_count,%ecx
    xorl %ebx,%ebx
    movq (%edx),K1
    movq 32(%edx),K2
    movq K1,tmp1
    movq 8(%edx),K3
    movq 16(%edx),K4
    DES_copy(24, 40)
    …
    DES_copy(112, 120)
    movq DES_IV,R
    xorl %edx,%edx
    movq DES_IV+8,L
DES_loop:
    …
```

Esempio – Assembly Alpha

```
DES_std_crypt:
    ldgp $29,0($27)
DES_std_crypt..ng:
    subq $30,56,$30
    lda tmp1,DES_IV
    lda tmp2,DES_count
    lda SPE,DES_SPE_F
    ldq R,0(tmp1)
    ldq L,8(tmp1)
    ldq count,0(tmp2)
    ldq K1,0(kp)
    ldq K2,8(kp)
    ldq K3,16(kp)
    ldq K4,24(kp)
    xor K1,R,D
    ldq K5,32(kp)
    ldq K6,40(kp)
    ldq K7,48(kp)
    …
```

```
        ldq K8,56(kp)
        stq K9,0($30)
        stq K10,8($30)
        stq K11,16($30)
        stq K12,24($30)
        stq K13,32($30)
        stq K14,40($30)
        stq K15,48($30)
        ldq K9,64(kp)
        ldq K10,72(kp)
        ldq K11,80(kp)
        ldq K12,88(kp)
        ldq K13,96(kp)
        ldq K14,104(kp)
        ldq K15,112(kp)
        ldq K16,120(kp)
DES_loop:
        DES_2_ROUNDS(K2, K3)
        …
```

Esempio – Assembly Sparc

```
DES_std_crypt:
    …
    save %sp,-120,%sp
    st %i7,[%fp-24]
    sethi %hi(DES_SPE_L),SPE_L_0
    sethi %hi(DES_SPE_L+0x400),SPE_L_4
    add SPE_L_0,0x808,SPE_H_0
    …
    ldd [kp],Dl
    ldd [SPE_L_4+0xC08],Rl
    …
    ld  [SPE_L_4+0xC18],count
DES_loop:
    DES_2_ROUNDS(kp)
    …
    std Rl,[out]
    std Ll,[out+8]
    ret
    restore
    …
```

Esempio – Assembly MC68000

```
* BSORT.ASM - Un semplice programma di ordinamento bubblesort
* Ordina in sense crescente 5 numeri alle locazioni $1000-$1004
N  equ  5    array size
   org  $400
* Qui comincia il programma
PROG   lea  array,A0 carica in A0 indirizzo array
   clr  D2   azzera il flag di scambio
   moveq    #N-1,D3  inizializza contatore iterazioni
loop1  move.b   (A0),D0  carica in D0 il 1° el. della coppia
   move.b   1(A0),D1 carica in D1 il 2° el. della coppia
   cmp.b    D0,D1    confronto
   bge.s    noswap   se il 2° è minore del 1°, scambiali
SWAP   move.b   D0,1(A0) sostituisci il 2° con il 1°
   move.b   D1,(A0)  sostituisci il 1° con il 2°
   moveq    #1,D2    metti ad 1 il flag di scambio
noswap addq.l   #1,A0    fa puntare A0 alla coppia seguente
   subq#1,D3    decrementa contatore di iterazioni
   bne  loop1    ripeti se ci sono ancora coppie
   tst  D2   test del flag di scambio
   bne  progripeti se fatto almeno uno scambio
   stop#$2000   arresta processore

   org  $1000
array  dc.b5,4,3,2,1    array, dopo ordinamento: 1,2,3,4,5
   end  prog
```