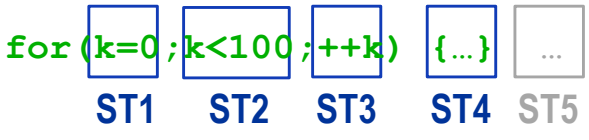
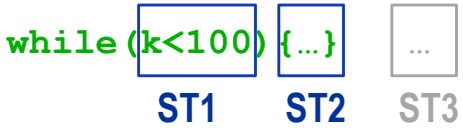
Lezione 11: Interfacciamento del software verso il RISC-V

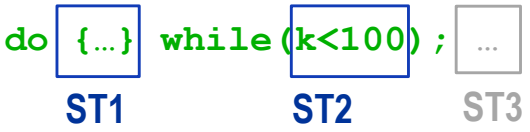
- Cicli FOR, WHILE, DO..WHILE




```
add x8,x0,x0                ST1

ini_for:                    ST2
  slt x5,x8,x10
  beq x5,x0,fin_for

  …                         ST4

  addi x8,x8,1              ST3
  beq x0,x0,ini_for

fin_for:                    ST5
  …
```




```
ini_wh:                     ST1
  slt x5,x8,x10
  beq x5,x0,fin_wh

  …                         ST2

  beq x0,x0,ini_wh

fin_wh:                     ST3
  …
```

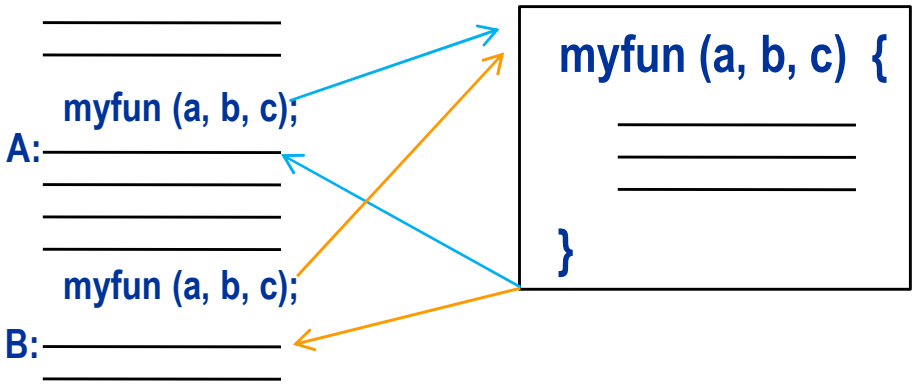



```
ini_do:                     ST1
  …

  slt x5,x8,x10             ST2
  bne x5,x0,ini_do

  …                         ST3
```

Chiamata a funzione

Patterson: 2.8,2.13,2.14

- In un programma C, la funzione rappresenta una porzione di codice richiamabile in uno o più punti del programma



- La chiamata deve prevedere un meccanismo per saltare al codice della funzione + un meccanismo per tornare al chiamante

Nota: l'indirizzo di "myfun" è lo stesso nei due casi ma la funzione deve essere in grado di tornare in due punti *diversi* del programma (A e B)

→La chiamata a funzione quindi
NON PUO' essere implementata con una semplice `bne/beq` → OCCORRE:

i) memorizzare da qualche parte il punto di ritorno, es. x1=&A

ii) saltare all'inizio della funzione "myfun" e al termine saltare a (x1)

- Nel RISC-V si usa quindi la pseudo-istruzione "jump and link" o `jal`:

```
jal label     # x1←PC+4;  PC←PC+OFFSET(&label-PC)
```

→ salta a `label` (es. myfun) mettendo l'indirizzo di ritorno (es. A oppure B) nel registro `x1` . Nota: l'istruzione reale è `jal x1,OFFSET` (v. slide succ.)

- Per tornare indietro uso x1 tramite la pseudo-istruzione "return" o `ret`:

```
ret           # PC←x1
```

Istruzione "JAL rd,offset" e il formato J

- 'JAL label' viene implementata con JAL rd,offset che è una istruzione che memorizza nel registro generico rd l'indirizzo PC+4 mentre somma offset a PC
- Quindi 'JAL label' è un modo più rapido per scrivere 'JAL x1,offset' dove offset=&label-PC; per avere un intervallo di salto ampio uso il formato 'J':
- Il formato J è una variante del formato U in cui, sfruttando bene i 20 bit dell'immediato, posso coprire un offset di 512ki istruzioni*
  - similmente alla BEQ i 20 bit indicano un offset in byte allineato alla half-word (quindi è come se avessi 21 bit in cui il bit0 è 0); contando 4B per istruz.->512ki istruz.

FORMATO U (già visto e usato per la LUI)

0000 0000 0001 0000 0000	00001	0110011
imm20[19:0]	rd	opcode

es. LUI x1,0x00100 (imm20)

FORMATO J (anche chiamato FORMATO UJ)

0010 0000 0000 0000 0000	00001	1101111
{imm21[20],imm21[10:1],imm21[11],imm21[19:12]}	rd	opcode

es. JAL 0x00200 (imm21)

Complessivamente sono i bit 20:1 di imm21, sparpagliati opportunamente all'interno dei 20 bit dell'istruzione

imm21 = 0x00200= (in base2) 0 0000 0000 0010 0000 0000

20 19:12 11 10:1

0010 0000 0000 0000 0000

Il bit meno significativo è sempre zero, quindi non occorre codificarlo

- Il registro rd codifica l'indice del registro x1 (ovvero 00001); x1 conterrà PC+4 (punti «A» e «B» della slide precedente)

*Nota: 512ki=512*1024=524288 istruzioni

Pseudo-istruzione JAL, istruzione JAL, istruzione JALR

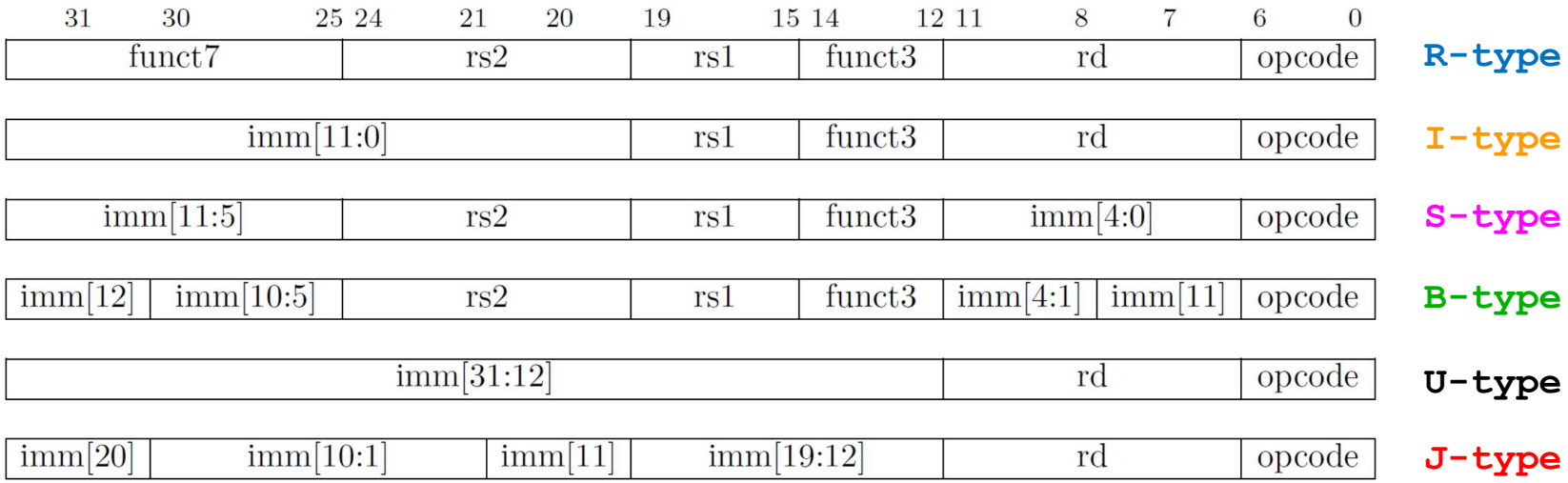
- Abbiamo quindi due possibili sintassi per JAL:
  - `JAL label` → denota la pseudo-istruzione (basata su `JAL ra,offset`)
  - `JAL rd,offset` → denota una ISTRUZIONE dove rd∈ {x0,..,x31}
    - Normalmente noi faremo riferimento alla pseudo-istruzione `JAL label`
    - `BEQ/BNE` e `JAL` permettono di generare codice «PIC» (Position Independent Code) che può essere caricato a qualsiasi indirizzo
- Per usare offset di ampiezza maggiore di 2^{20} (e quindi poter chiamare funzioni o fare salti a qualsiasi indirizzo) occorre far riferimento a un registro → istruzione `JALR`
  - `JALR rd,offset(rs1)` → `x[rd]=PC+4; PC=x[rs1]+sext(offset)&~1`
  - «`sext(offset)`» denota che offset viene esteso di segno a 64 bit
  - «`&~1`» denota che il bit 0 viene azzerato

  Nota: la JALR viene codificata usando il formato I

  Nota2: la JALR ha bisogno di precaricare nel registro x[rs1] l'indirizzo a cui saltare
- Con `JALR x0,0(rs1)` posso supportare la pseudo-istruzione `JR rs1` per fare un salto a un indirizzo qualsiasi (contenuto in rs1)
- Con `JALR x0,0(ra)` posso supportare la pseudo-istruzione `RET` per realizzare il ritorno al chiamante, al termine di una funzione

Noi useremo solo la coppia: JAL etichetta + RET

Ricapitolazione dei formati e istruzioni fin qui viste



- **Istruzione** **Significato** **Variante** **Significato**

```
add x5,x6,x7    x5 = x6 + x7          addi x5,x6,10     x5 = x6 + 10
sub x5,x6,x7    x5 = x6 - x7
slt x5,x6,x7    x5 = (x6<x7) ? 1 : 0  slti x5,x6,10     x5 = (x6<10) ? 1 : 0

lw  x5,100(x6)  x5 = Memory[x6+100] (32 bit)  ld  x5,100(x6)   legge  64 bit
sw  x5,100(x6)  Memory[x6+100] = x5 (32 bit)  sd  x5,100(x6)   scrive 64 bit

bne x6,x7,L     salta a L se x6!=x7 (L si trova a offset imm13=(&L-PC))
beq x6,x7,L     salta a L se x5==x7 (L si trova a offset imm13=(&L-PC))

lui x5,imm20    carica imm20 nei 20 bit alti

jal FUN         esegue la funzione FUN (che si trova a offset imm21=(&FUN-PC))
ret             ritorna da funzione (jalr x0,0(x1))
```

Regole di allocazione dei registri o ABI (Application Binary Interface)

Num. registro	Scopo	Nome	invariato su "call"?
x0	la costante 0	zero	SI
x1	return address (il chiamato DEVE salvarlo**)	ra	SI
x2	stack pointer (il chiamato DEVE salvarlo**)	sp	SI
x3	global pointer (il chiamato DEVE salvarlo**)	gp	SI
x4	thread pointer (il chiamato DEVE salvarlo**)	tp	SI
x5	temporaneo / return-address-alternativo	t0	no
x6-x7	temporanei	t1-t2	no
x8	registro **salvato** o frame pointer (il chiamato DEVE salvarlo**)	s0/fp	SI
x9	registro **salvato** (il chiamato DEVE salvarlo**)	s1	SI
x10-x11	argomenti della funzione / valori di ritorno	a0-a1	no
x12-x17	argomenti della funzione	a2-a7	no
x18-x27	registro **salvati** (il chiamato DEVE salvarli**)	s2-s11	SI
x28-x31	temporanei (il chiamato li puo' sovrascrivere)	t3-t6	no

** nel caso in cui tale registro sia utilizzato

** N.B.: «salvati» significa «salvati-prima», ovvero
devono essere salvati nel frame i valori precedenti all'inizio della funzione e prima dell'uso all'interno della funzione

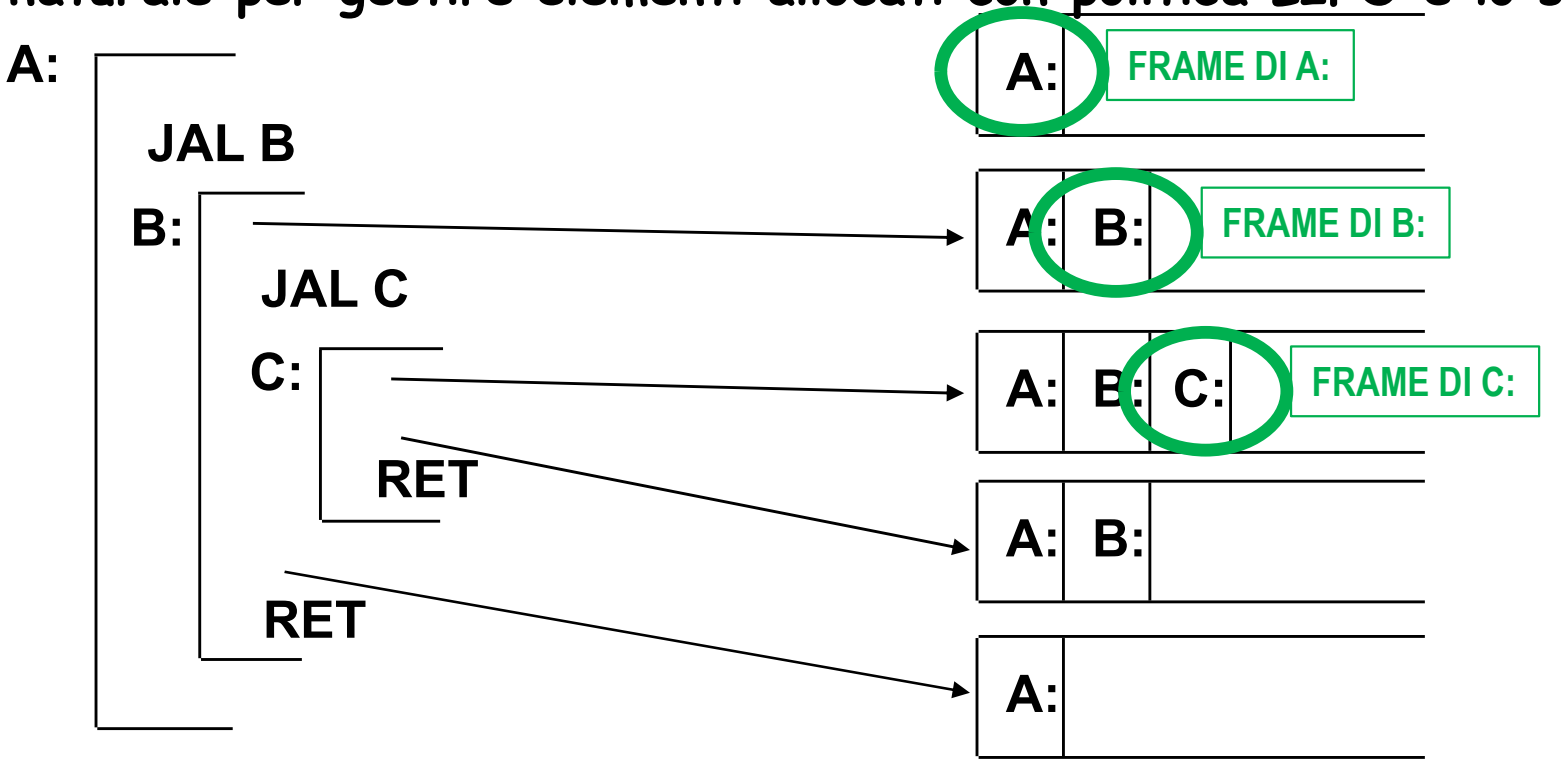
Operazioni connesse alla chiamata di funzione

1) eventuale salvataggio registri da preservare (es. t0-t6, a0-a7, nello stack param. oltre l'ottavo) 2) preparazione dei parametri di ingresso (nuovi a0-a7)	pre-chiamata (lato chiamante)
3) chiamata della funzione	`JAL MYFUN`
4) allocazione del call-frame nello stack 5) eventuali salvataggi vari (es. (old) a0-a7, (old) ra, (old) fp/s0, (old) s1-s11) 6) eventuale inizializzazione nuovo fp	prologo (lato chiamato)
7) esecuzione del codice della funzione	corpo (lato chiamato)
8) preparazione dei parametri di uscita (a0-a1) 9) ripristino dei parametri salvati (salvati al punto 5)	epilogo (lato chiamato)
10) ritorno al codice originario	`RET`
11) uso dei valori di uscita della funzione 12) eventuale ripristino dei vecchi valori (t0-t6,a0-a7) salvati al punto 1	post-chiamata (lato chiamante)

- I "salvataggi" sono fatti in una zona di memoria chiamata "record di attivazione" ("call frame")

Chiamate annidate

- Per ogni chiamata alloco un nuovo frame (i parametri passati possono essere diversi)
- Per chiamate annidate, il frame precedente non viene buttato e quindi si ha necessità di avere una politica LIFO (Last-In First-Out) per gestire i frame
- Il modo naturale per gestire elementi allocati con politica LIFO è lo stack



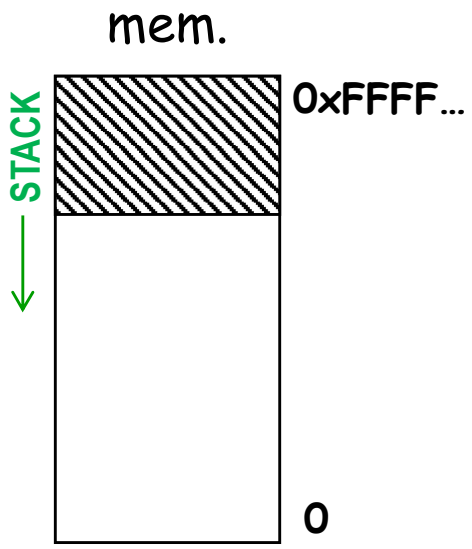
- Alcune macchine forniscono le istruzioni per la gestione di stack come parte della architettura stessa (e.g. VAX, processori Intel)
  - Istruzioni PUSH e POP
- Gli stack possono però anche essere implementati via software (e.g. MIPS, RISC-V)

Convenzione sugli stack (1): "verso di crescita"

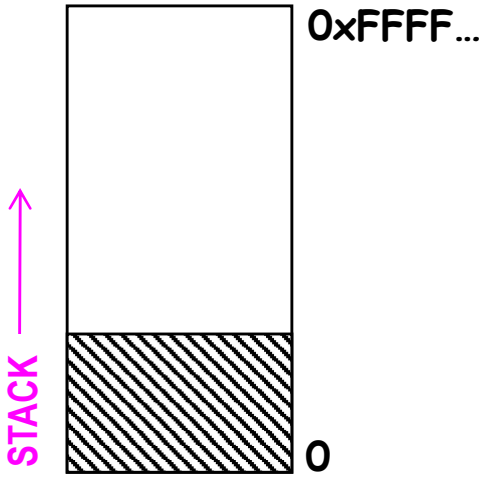
- Assumendo che nella rappresentazione della memoria gli indirizzi alti siano in alto e quelli bassi in basso…
  - Con '0xFFFF…' indico gli indirizzi alti in memoria
  - Con '0' indico gli indirizzi bassi in memoria

…gli stack possono crescere verso l'alto oppure verso il basso

Convenzione
grow down



Convenzione
grow up



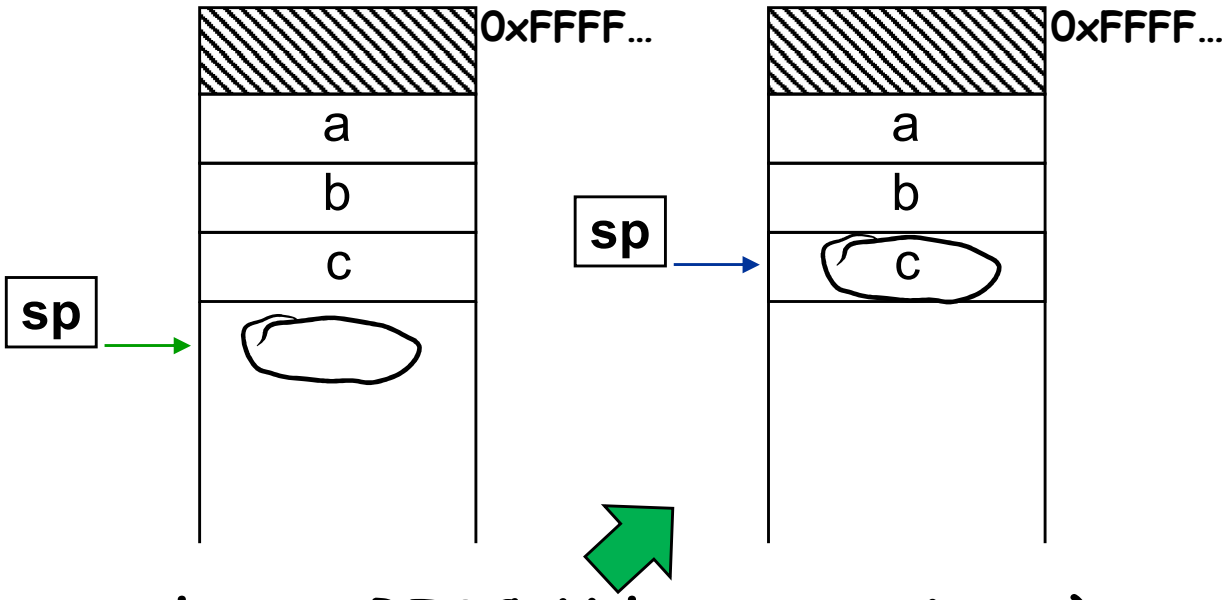
RISC-V

Convenzione sugli stack (2): "a cosa punta sp"

- Il punto di inserimento dello stack viene puntato dal registro sp
- Ci sono due opzioni per ciò a cui punta sp: Next-Empty vs. Last-Full

Convenzione (grow-down +) next-empty

Convenzione (grow-down+) last-full



Operazioni sullo stack:

Grow-down + Next-Empty

POP: Incrementa sp
Legge da Mem[sp]

PUSH: Scrive in Mem[sp]
Decrementa sp

Grow-down + Last-Full

POP: Legge da Mem[sp]
Incrementa sp

PUSH: Decrementa sp
Scrive in Mem[sp]

- nel caso RISC-V la convenzione è:
  - Grow-Down+Last-Full

Nota: lo stack viene usato non solo per i call-frame ma anche per:
- allocazione VARIABILI LOCALI della funzione
- MEMORIZZAZIONE TEMPORANEA di valori in "esubero" dai registri (es. nella valutazione di espressioni)

Convenzione RISC-V per le chiamate a funzioni (1)

PRE-CHIAMATA (LATO CHIAMANTE)

1) Eventuale salvataggio registri <u>da preservare nel chiamante</u>
   - Si assume che a0-a7,t0-t6, possano essere sovrascritti
     - <u>se</u> li si vuole preservare vanno salvati nello stack (dal chiamante) (in particolare i vecchi valori di a0-a7)

2) Preparazione degli <u>argomenti della funzione</u>
   - I primi 8 argomenti vengono posti in a0-a7 (nuovi valori)
   - Gli eventuali altri argomenti oltre l'ottavo vanno salvati nello stack (EXTRA_ARGS), così che si trovino subito sopra il frame della funzione chiamata

CHIAMATA

3) `jal MYFUN`

NOTA: la `jal` mette innanzitutto in `ra` l'indirizzo di ritorno (ovvero l'indirizzo dell'istruzione successiva alla `jal` stessa); Dopodiché salta all'indirizzo specificato da MYFUN

Convenzione RISC-V per le chiamate a funzioni (2)

PROLOGO (LATO CHIAMATO)

4) Eventuale allocazione del call-frame sullo stack
(→ aggiornare sp)

5) Eventuale salvataggio registri che si intende sovrascrivere

- Salvataggio degli argomenti a0-a7 *solo se la funzione ha necessità di riusarli nel corpo di questa funzione, successivamente a ulteriori chiamate a funzione che usino tali registri,*
(nota: negli altri casi a0-a7 possono essere sovrascritti)
- Devo salvare il vecchio ra, *solo se la funzione chiama altre funzioni…*
- Devo salvare il vecchio fp, *solo se ho bisogno effettivamente del call-frame pointer (e devo quindi sovrascriverlo)*
- Devo infine SEMPRE salvare i vecchi s0-s11 *se intendo usare tali registri*
(il chiamante si aspetta di trovarli intatti)

6) Eventuale inizializzazione di fp : punta al nuovo call-frame

CORPO DELLA FUNZIONE

7) CODICE EFFETTIVO DELLA FUNZIONE

Convenzione RISC-V per le chiamate a funzioni (3)

EPILOGO (LATO CHIAMATO)

8) Se deve essere restituito un valore dalla funzione
   - Tale valore viene posto in a0 (e a1)

9) I registri (se salvati) devono essere ripristinati
   - a0-a7 (nel caso siano stati salvati all'interno della funzione)
   - s0-s11
   - ra
   - fp

Inoltre, notare che sp deve solo essere aumentato di opportuno offeset (lo stesso sottratto nel punto 4)

RITORNO AL CHIAMANTE

10) `ret`

POST-CHIAMATA (LATO CHIAMANTE)

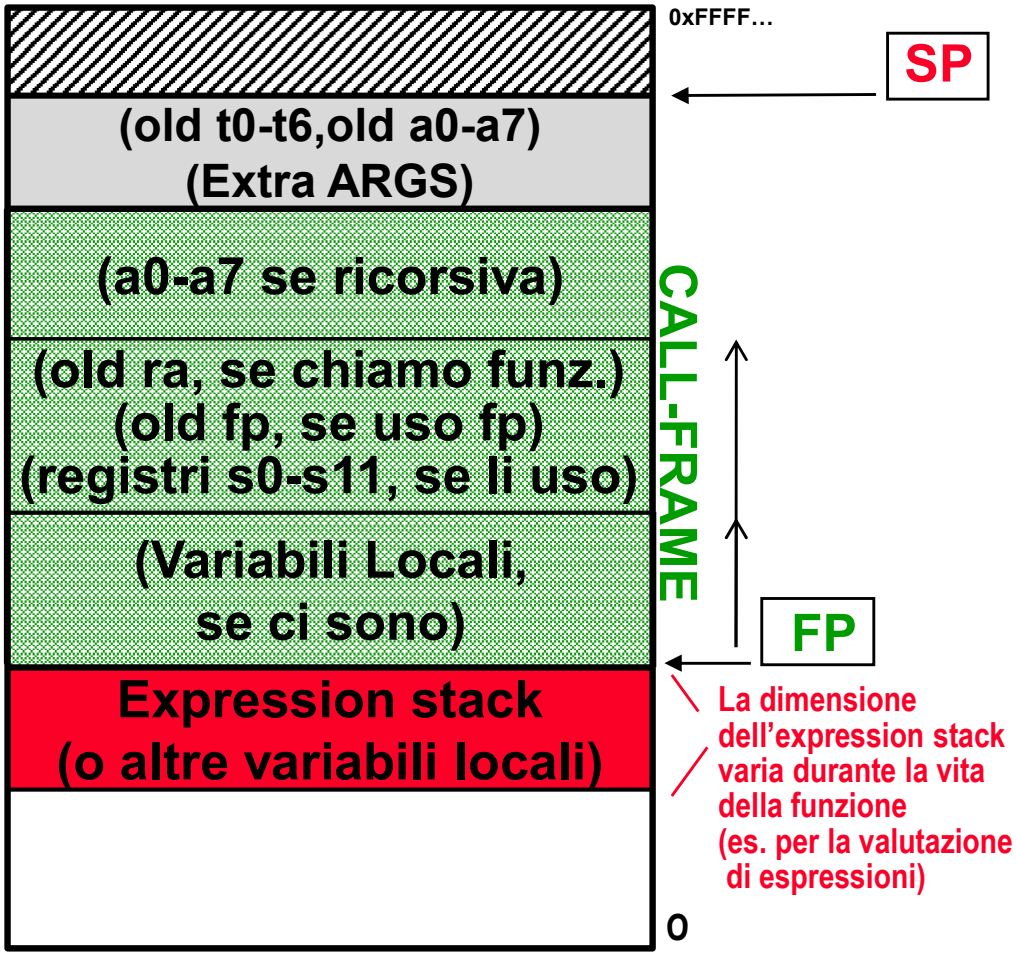
11) Eventuale uso del risultato della funzione (in a0 (e a1))

12) Ripristino dei valori t0-t6, a0-a7 (vecchi) eventualmente salvati

Riepilogo: struttura del Call-Frame

- Grow-Down + Last-Full Stack
- FP=Frame Pointer (i.e., s0)
- SP=Stack Pointer

```
                        addi  sp,sp-16
                        sd   t0,0(sp) # salva t0
                        sd   xx,8(sp) # salva 9^arg
....                    call fun
n=fun(a,b,c,d,e,f,g,h,xx)
    fun(a,b,c,d,e,f,g,h,i) {                    fun:
        int x,y,z;                              addi  sp,sp-144
        int v[20];                              sd    a0,136(sp)
        …                                       sd    a1,128(sp)
                                                …
                                                sd    ra,120(sp)
                                                sd    s0,112(sp)
                                                sd    s1,104(sp)
                                                …
                                                li a0,1
                                                li a1,2
                                                …
        x=fun(1,2,3,4,5,6,7,8,9)                call fun
        …                                       …
        return(100);                            ld ra,120(sp)
                                                li  a0,100
    }                                           ret
....                    <uso a0>
                        ld   t0,0(sp)
                        addi sp,sp,16
```

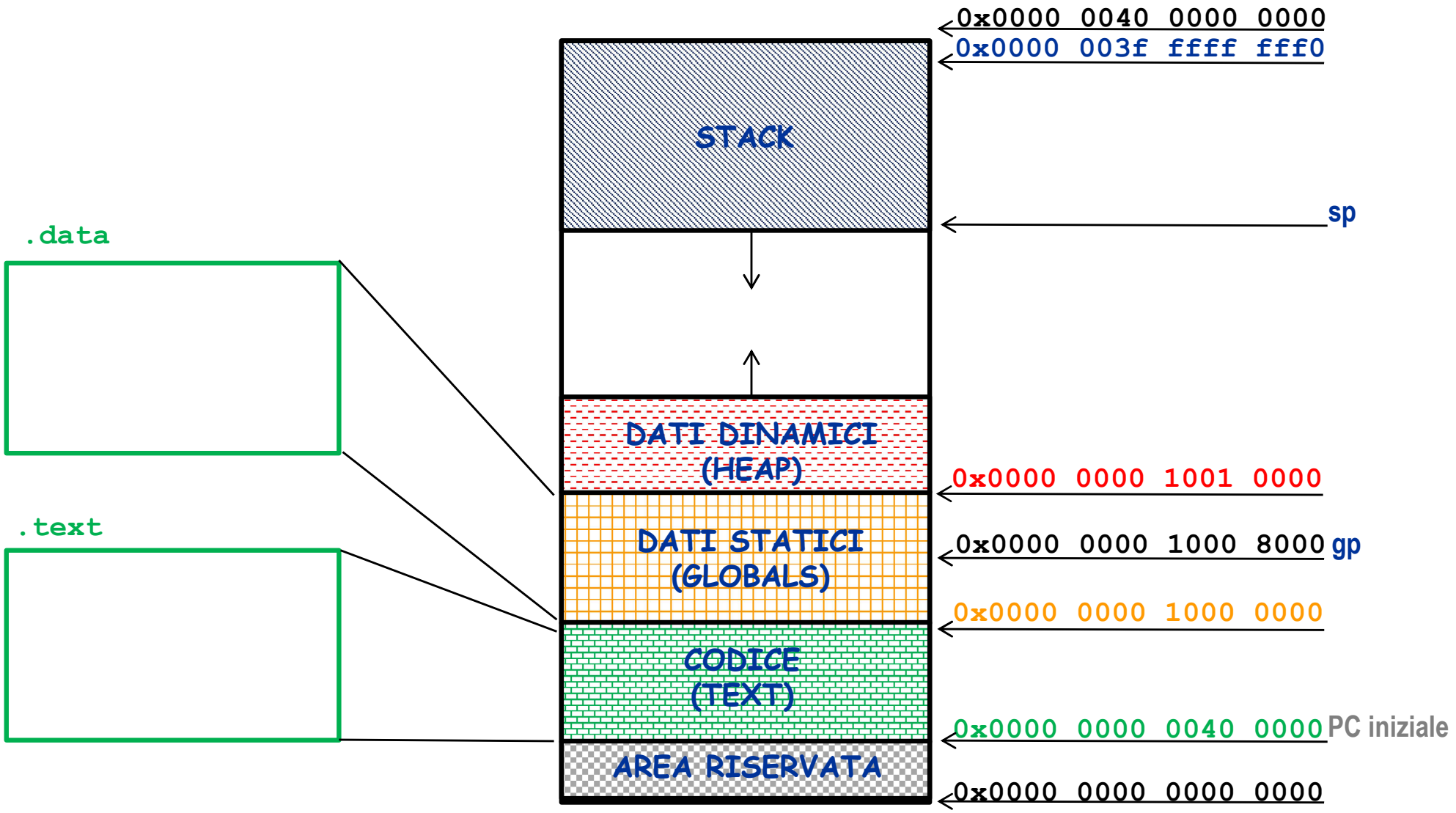


Nota1: FP è comodo per codificare nel corpo della funzione gli accessi a offset fissi all'interno del call-frame (non possibile con SP perché varia)

Nota2: I compilatori normalmente cercano di allocare le variabili locali prima di tutto nei registri, non in memoria (o stack)!

Mappa di memoria (Memory Layout)

- Dove si trova lo stack? il programma ? i dati globali ?



Nota: gli indirizzi sono stabiliti per convenzione

Principali direttive per l'assemblatore

.data [<addr>] marca l'inizio di una zona dati;
se <addr> è specificato, i dati sono memorizzati a partire da <addr>

.text [<addr>] marca l'inizio del codice assembly;
se <addr> è specificato, il codice è memorizzato a partire da <addr>

.globl <symb> dichiara che il simbolo <symb> visibile è dall'esterno (gli altri simboli sono locali per default) in modo che possa essere usato da altri file

.extern <symb> [<size>] dichiara che il dato memorizzato all'indirizzo <symb> è un simbolo globale ed occupa <size> byte.
Questo consente all'assemblatore di:

1) memorizzare il dato in una porzione del segmento dati (quello indirizzato tramite il registro gp)

2) dichiarare che i riferimenti al simbolo <symb> riguardano un oggetto esterno ad un modulo (torneremo su questo fra poco)

.asciz "str" mette in memoria la stringa "str",seguita da un byte '0'

.ascii "str" mette in memoria la stringa "str",SENZA lo '0' finale

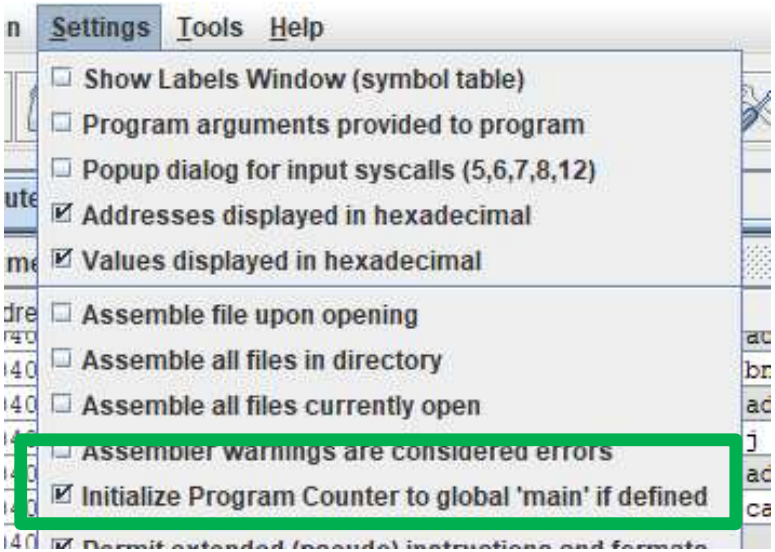
Programma fattoriale_riscv.s (1)

```
.data
    messaggio:      .asciz "Calcolo di n!\n"
    insdati:        .asciz "Inserisci n = "
    visris:         .asciz "n! = "
    RTN:            .asciz "\n"

.globl main
.text
fact:
# lo stack cresce verso il basso
    addi sp, sp, -12      # allocazione del call frame nello stack
    sw   a0, 8(sp)        # salvataggio di n nel call frame
    sw   ra, 4(sp)        # salvataggio dell'indirizzo di ritorno
    sw   s0, 0(sp)        # salvataggio del precedente frame pointer
    add  s0, sp, zero     # aggiornamento del frame pointer


# calcolo del fattoriale
    bne  a0, zero,  Ric   # test fine ricorsione n!=0
    addi a0, zero, 1      # 0! = 1
    j    Fine

Ric:                      # chiamata ricorsiva per il calcolo di (n-1)!
    addi a0, a0, -1       # a0 <- (n - 1)  passaggio del parametro in a0 per fact(n-1)
    jal  fact             # chiama fact(n-1) -> risultato in a0
    lw   t0, 8(s0)        # t0 <- n
    mul  a0, a0, t0       # n! = (n-1)! x n

# uscita dalla funzione
Fine:

    lw   s0, 0(sp)        # recupera il frame pointer
    lw   ra, 4(sp)        # recupera l'indirizzo di ritorno
    addi sp, sp, 12       # elimina il call frame dallo stack
    ret                   # ritorna al chiamante
```

Programma fattoriale_riscv.s (2)

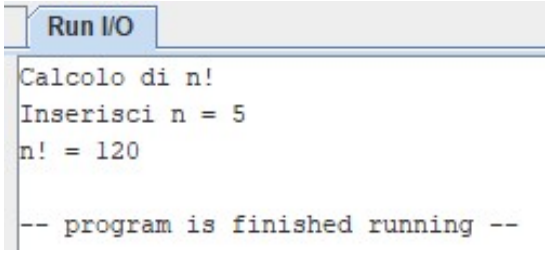
```
# Programma principale
#

main:

# Stampa intestazione
    la   a0, messaggio
    addi a7, zero, 4
    ecall

# Stampa richiesta
    la   a0, insdati
    addi a7, zero,  4
    ecall

# legge n (valore in a0)
    addi a7, zero,  5
    ecall

# chiama fact(n)
    call  fact          # parametro n in a0
    add  s0, a0, zero   # salva il risultato in s0

# stampa messaggio per il risultato
    la   a0, visris
    addi a7, zero,  4
    ecall

# stampa n!
    add  a0, s0, zero
    addi a7, zero,  1
    ecall

# stampa \n
    la   a0, RTN
    addi a7, zero,  4
    ecall

# exit
    addi a7, zero,  10
    ecall
```

Nota: NEL SIMULATORE RARS, selezionare Settings/Initialize_Program_Counter_to_global_'main'



NOTA: prerequisito: occorre istallare JAVA SE

Output del simulatore RARS | https://github.com/TheThirdOne/rars



Ulteriori direttive assemblatore

.byte <b1>, ..., <bn>

Mette in memoria gli n byte che sono specificati da <b1>, … , <bn>

.word <w1>, ..., <wn>

Mette in memoria le n word a 32-bit che sono specificate da <w1>, … , <wn>

.float <f1>, ..., <fn>

Mette in memoria gli n numeri floating point a singola precisione (32 bit) (in locazioni di memoria contigue)

.double <d1>, ..., <dn>

Mette in memoria gli n numeri floating point a doppia precisione (64 bit) (in locazioni di memoria contigue)

.half <h1>, ..., <hn>

Mette in memoria le n quantità a 16 bit (in locazioni di memoria contigue)

Per tipi di dati predefiniti

.space <n>

Mette in memoria n spazi (ovvero riserva n byte di memoria)

Per tipi di dati strutturati (es. Array, strutture, …)

.align <n>

Allinea il dato successivo a un indirizzo multiplo di 2^n

- es. «.align 2» allinea il valore successivo "alla word"
- «.align 0 disattiva l'allineamento generato dalle direttive .half, .word, .float, e .double fino alla successiva direttiva .data o .kdata

Servizi di sistema (system call) – istruzione 'ecall'

- 'ecall' serve per chiedere un servizio al sistema operativo (operazione nota come 'system call')
  - Il numero del servizio è indicato in a7*

    - 1: stampa un intero a video
      (a0* in ingresso, contiene l'intero in binario da stampare)

    - 4: stampa un messaggio a video
      (a0* in ingresso, indirizzo della stringa da stampare)

    - 5: leggi un intero dalla tastiera
      (a0* in uscita, conterrà l'intero letto in binario)

    - 10: termina il programma
      (il sistema operativo riprende il controllo del calcolatore)

Nota: si può assumere che la ecall si comporti come una chiamata a funzione speciale in cui non deve essere alterato alcun registro (salvo quelli utilizzati in ingresso o uscita naturalmente). Questo è possibile grazie al meccanismo di eccezione (che vedremo in dettagli più avanti nella lezione 9).

* Con riferimento al simulatore «RARS»; altri simulatori usano rispettivamente a0 e a1 anziché a7 e a0

Compilazione di un programma strutturato a moduli

- Nel precedente esempio non compaiono esplicitamente alcuni passi logici per arrivare ad eseguire un programma
  - Compilazione pura dei file
    → per controllare di aver scritto bene ciascun modulo
  - Collegamento di più moduli
    → per creare un unico file da essere eseguito (il programma)
  - Caricamento del programma in memoria e sua esecuzione
    → per consentire al processore di eseguire il nostro programma
- Compilazione separata

```
cc -c main1.c
cc -c queue.c
```

  - vengono così creati (salvo errori) i file `main1.o queue.o`
- Collegamento (Linking) → viene invocato un altro stadio di cc

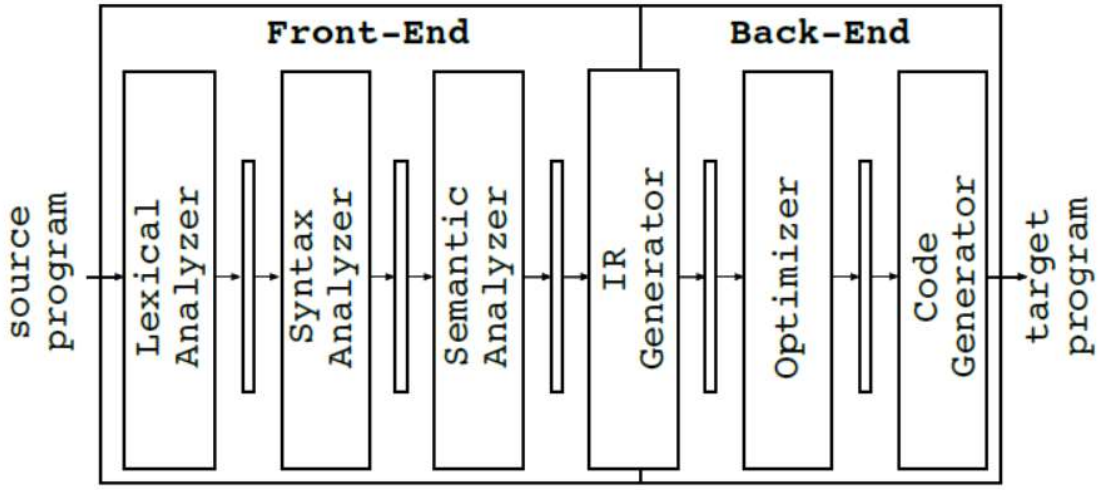
```
cc –o main main1.o queue.o
```

- Caricamento ed esecuzione

```
./main
```

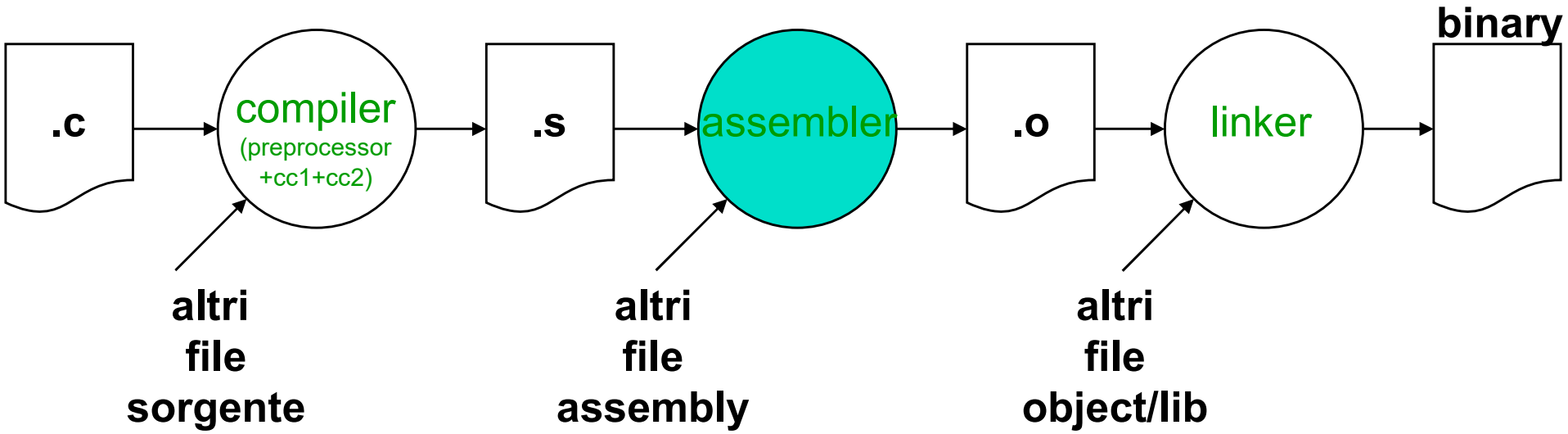
Passi principali della compilazione di un programma C

- **Preprocessore (stadio 'cpp')**
  - espande le macro e le costanti (#define)
  - inserisce nell'elaborato i file inclusi (#include)
  - processa le direttive condizionali (#ifdef, #ifndef, ...)
- **Compilazione vera e propria**
  - si svolge in uno o più stadi (cc1, cc2) fino ad ottenere il livello di ottimizzazione desiderato
  - il risultato intermedio è un programma in ASSEMBLY
  - il codice ASSEMBLY è legato all'architettura target
- **Creazione del file oggetto**
  - questo passo è eseguito dall'assemblatore
  - il file oggetto contiene
    - la codifica delle istruzioni assembly
    - i dati statici
    - informazioni di rilocazione
    - tabella dei simboli
    - altre informazioni di utilità



Visione semplificata di un compilatore
[https://www.researchgate.net/publication/332241231_Principles_Techniques_and_Tools_for_Explicit_and_Automatic_Parallelization]

Assemblatore



- È possibile produrre l'output in assembly del compilatore (UNIX/LINUX),

```
cc -S main.c
```

il risultato si trova nel file `main.s`

Assemblatore a due passate (1)

- Prima passata:
  determinazione delle locazioni di memoria etichettate (creazione della tabella dei simboli):

```
.text
        add   x18, x18, x0          LC=0
        addi  x19, x19, 0           LC=4
loop:   add    x4, x18, x0          LC=8
        add   x19, x19, x4          LC=12
        add   x18, x18, x19         LC=16
        bne   x18,  x5, loop        LC=20
        la    x18, myvar            LC=24
              ..........
        jal   fun1                  LC=3000
              ..........

fun1:         ..........            LC=20'000'000
              ..........
              ..........

.data
myvar:   .......                    LC=0
```

Symbol Table (provvisoria)

symbol	Symbol Location Counter (Symbol-LC)
loop	8
myvar	0
fun1	20'000'000

Assemblatore a due passate (2)

- Seconda passata:
  - traduzione delle istruzioni nei rispettivi codici operativi (`opcode`)
  - determinazione dei numeri dei registri (`rs1, rs2, rd`)
  - sostituzione delle etichette** nelle istruzioni con indirizzamento relativo al PC (`beq, bne`) e produzione symbol table finale
  - creazione della Tabella di Rilocazione

- ** Es. Per la `bne` a LC=20 che usa il simbolo 'loop'

```
offset = LC(SYMBOL)-[LC(BNE)]
```

```
loop:  add   x4, x18, x0        LC=8
       add  x19, x19, x4        LC=12
       add  x18, x18, x19       LC=16
       bne  x18,  x5, loop      LC=20
```

8-[20]= = -12 ⇨ `bne x18, x5, -12`

Nota: nella codifica si scarta il bit meno signficativo dell'offset:
-12 diventa -6
→ memorizzo nel campo «imm» dell'istruzione il valore -6

- Per ogni etichetta irrisolta, si controlla che non sia definita come "simbolo esterno" tramite `.globl` (ovvero un simbolo che deve essere visibile esternamente)
  - se non è un simbolo esterno, può essere un errore
  - se è un simbolo esterno o un riferimento ad un indirizzo assoluto (es. nelle istruzioni `jal, j, la`)
    → deve restare in symbol table

symbol	Symbol-LC
myvar	0
fun1	20'000'000

Creazione e uso della Tabella di Rilocazione

```
          -----                LC=0
          la x18, myvar        LC=24
          -----
          -----
          jal fun1             LC=3000
          -----
fun1:     -----                LC=20'000'000
```

Relocation Table

Referring Location Counter (Referring-LC)	TIPO ISTRUZIONE	SIMBOLO DIPENDENTE
24	la	myvar
3000	jal	fun1

• **USO DELLA RELOCATION TABLE: in fase di linking le informazioni in Symbol Table e Relocation Table vengono combinate con gli indirizzi effettivi: in questo caso il linker assume che il programma venga caricato ad un indirizzo fisso prestabilito (es. 0x0040'0000) mentre i dati risiedono ad un altro indirizzo fisso (es. 0x1000'0000»):**

```
0x0040'0000          add  x18, x18, x0
0x0040'0004          addi x19, x19, 0
0x0040'0008 (loop:)  add   x4, x18, x0
0x0040'000C          add  x19, x19, x4
0x0040'0010          add  x18, x18, x19
0x0040'0014          bne  x18, x5, -12      # già stabilito dall'assembler
0x0040'0018          la   x18, 0x1000'0000  # diviso in due parti da 20 bit e 12 bit
0x0040'001C          .....                  # la seconda parte è zero, per cui basta
                     .....                  # lui x18,0x10000
                     .....
```

0xBB8=3000

```
0x0040'0BB8          jal  0x0171'2D00       # non si può tradurre con un offset da 20 bit
0x1000'0000 (myvar:).......                 # lui  x5,0x01713 # 171'2D00=0x40000+20'000'000
                                            # jalr ra,0xD00(x5) # 0xD00 è negativo quindi
```

0x131'2D00=20'000'000

```
                                            # aumento di 1 l'immediato della lui
0x0171'2D00 (fun1:) .....
0x0171'2D04         .....                   # per myvar l'indirizzo LC=0 corrisponde all'inizio
                                            # della zona di dati globali 0x1000'0000
```

Simboli esterni

- Un simbolo è **esterno**, se viene esportato (es. da una libreria) verso altri moduli

  `.globl <symbol>` dichiara il simbolo <symbol> come un'etichetta visibile all'esterno del modulo

Modulo B

```
.globl cubic
cubic:
mul t0, a0, a0   # LC=0
mul a0, t0, a0
ret
```

Symbol Table modulo B

symbol	Symbol-LC
cubic	0

- In qualche altro modulo esisterà un riferimento alla funzione "cubic":

  `.extern <symbol>` dichiara che ogni riferimento a <symbol> (es. jal cubic) si riferisce ad un simbolo esterno a questo modulo

Modulo A

```
.extern cubic
----
jal cubic   # LC=1500
----
----
```

Symbol Table modulo A

symbol	Symbol-LC

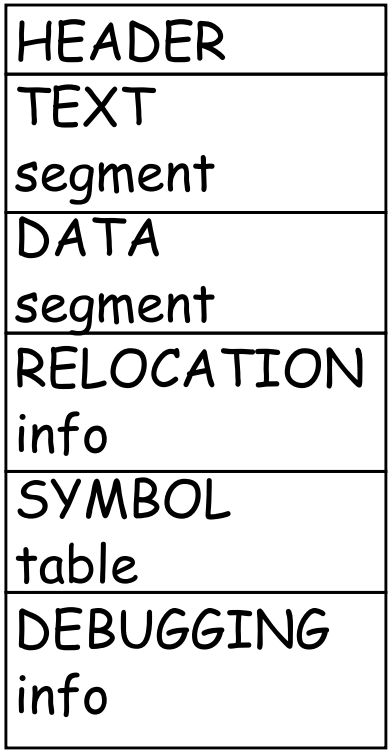
Relocation Table modulo A

Referring-LC	TIPO ISTRUZIONE	SIMBOLO DIPENDENTE
1500	jal	cubic

Nota: il simbolo "cubic" verrà quindi risolto in fase di linking

Nota2: nel simulatore RARS, si possono editare i due moduli separatamente (anche senza la dichiarazione .extern) ma con la dichiarazione .globl nel modulo «B» e abilitare l'assemblaggio simultaneo di tutti i moduli (il collegamento viene effettuato generando un'unica symble table)

Formato del file oggetto (.o)



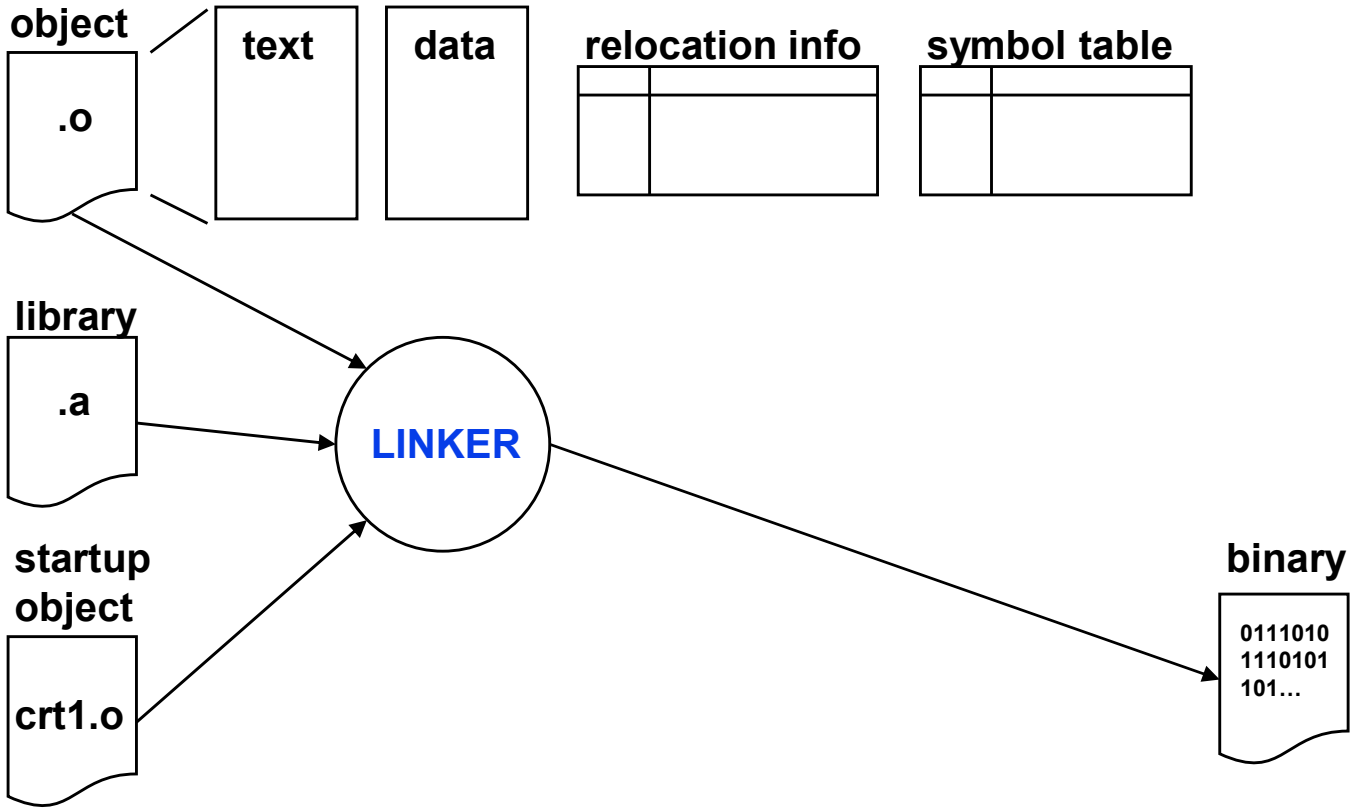
- Dimensione del file, delle parti (text, data) e CRC
- Codifica delle istruzioni (linguaggio macchina DA RILOCARE)
- Dati
- Lista delle istruzioni che fanno riferimento ad indirizzi assoluti
- Lista dei simboli locali (etichette, variabili) e etichette visibili all'esterno (assembly .globl)
- Parte opzionale per facilitare il debugging del programma

- UNIX/LINUX:

  il comando **nm** consente di produrre informazioni sul layout del file oggetto

File eseguibile (binary)

- è prodotto dal linker



- I riferimenti assoluti sono rilocati e i simboli sostituiti con indirizzi

Nota: un formato molto usato per gli eseguibili è detto ELF.
Per approfondimenti: http://www.caldera.com/developers/devspecs/mipsabi.pdf

Caricamento (loader)

- Il Sistema Operativo gestisce la richiesta fatta alla shell di mandare in esecuzione il mio programma

- Il sistema operativo effettua le seguenti operazioni
  - Lettura del file header
  - Creazione dello spazio di indirizzi allocato al programma
  - Copia di TEXT e DATA nello spazio indirizzi
  - Inizializzazione dei registri
  - Salto alla routine di startup

Nota:
è possibile generare il file binario in un colpo solo con

```
cc -o main main.c queue.c
```

Lezione 12 - Standard IEEE-754 per le operazioni floating-point

- Nelle operazioni floating point dobbiamo rappresentare
  - numeri con decimali, e.g., 3.1416
  - numeri molto piccoli, e.g., .000000001
  - numeri molto grandi, e.g., 3.15576×10^9

- Rappresentazione nel calcolatore:
  - Partendo dall'idea di memorizzare tali numeri usando cifre binarie, scriveremo il numero floating point x da rappresentare nella forma:

    $x = (-1)^{segno} \times mantissa \times 2^{esponente}$

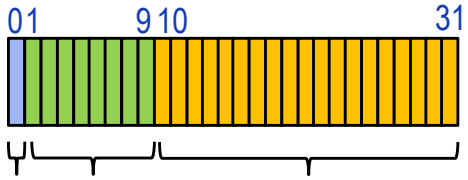
    - dove viene scelto 1 <= 'mantissa' < 2
    - più cifre binarie (bit) di 'mantissa' danno più accuratezza nelle operazioni
    - più cifre binarie (bit) di 'esponente' permettono numeri molto grandi e molto piccoli

IEEE-754

- **IEEE 754 floating point standard:**
  - **singola precisione ("float" del *C*) → uso 32 bit per memorizzare:**
    1 bit per S, 8 bit per E, 23 bit per M**
  - **doppia precisione ("double" del *C*) → uso 64 bit per memorizzare:**
    1 bit per S, 11 bit per E, 52 bit per M**
  - **quadrupla precisione ("__float128" del *C*) → uso 128 bit:**
    1 bit per S, 15 bit per E, 112 bit per M**

:= esize := msize

Esempio:



IEEE-754 Singola Precisione (32 bit totali)

Se x = 0.6, la sua rappresentazione floating point IEEE-754 è:
F = 00111111000110011001100110011010 (=0x3f19999a)

**Nota: utilizzando il "trucco" della normalizzazione della mantissa nella forma 1.xxxxx
le cifre (binarie) effettive per la mantissa sono 24 (singola prec.), 53 (doppia prec.), 113 (quad. prec.)

RELAZIONE FRA NUMERO e SUA RAPPRESENTAZIONE IEEE-754 <S,E,M>

- Sia

  $x = (-1)^{segno} \times mantissa \times 2^{esponente}$

- Nel formato IEEE-754 la rappresentazione F di x è:

  $(x)_{IEEE-754} = F = <S,E,M>$

  dove

  S = segno

  E = esponente + polarizzazione

  M = int[(mantissa-1) × 2^{msize}]

  RELAZIONE FRA NUMERO x
  E SUA RAPPRESENTAZIONE IEEE-754

  (S,E,M sono numeri interi rappresentati su un certo numero di bit ovvero
  S = 0 o 1, M < 2^{msize}, E < $2^{esize}-1$, polarizzazione=$2^{esize-1}-1$)

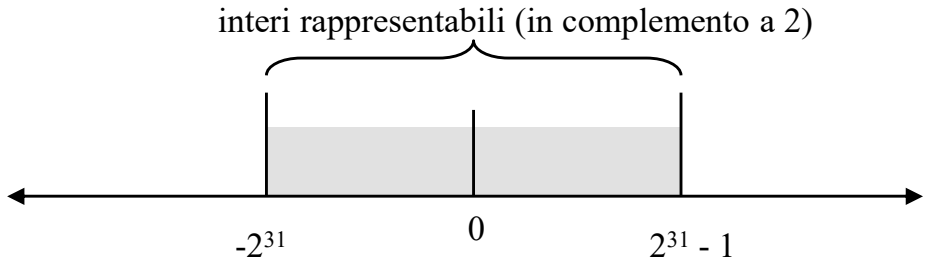
  Per es. in «IEEE-754 singola precisione» si ha: polarizzazione=127, msize=23, esize=8 quindi:

  $x=0.375 = (-1)^0 \times 1.5 \times 2^{-2}$
  → segno=0,esponente=-2,mantissa=1.5 → S=0, E=-2+127=125, M=(1.5-1)* $2^{23} = 2^{22}$
  → → $(x)_{IEEE-754} = F = <0,125,2^{22}>_{base10} = <0,0111\ 1101,1000\ 0000\ 0000\ 0000\ 0000\ 000>_{base2}$

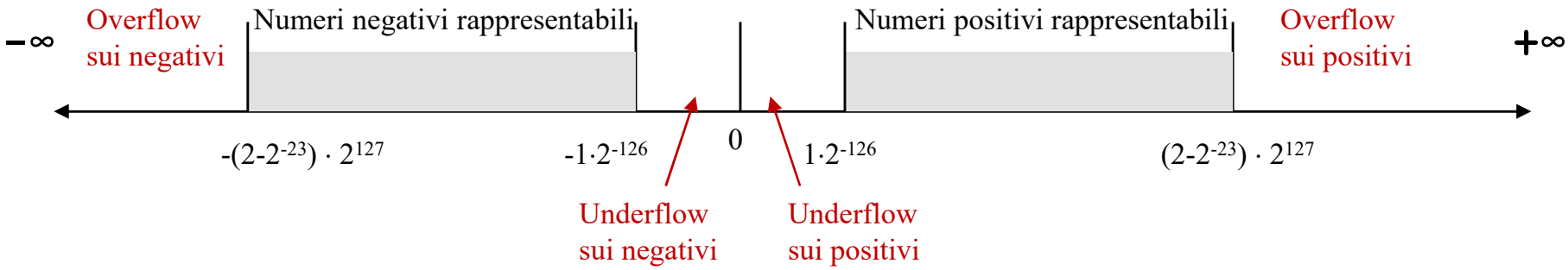
Nota: l'operazione int[.] tronca il numero di bit → quindi la rappresentazione IEEE-754 **spesso approssima** il numero x

Come utilizzare i bit

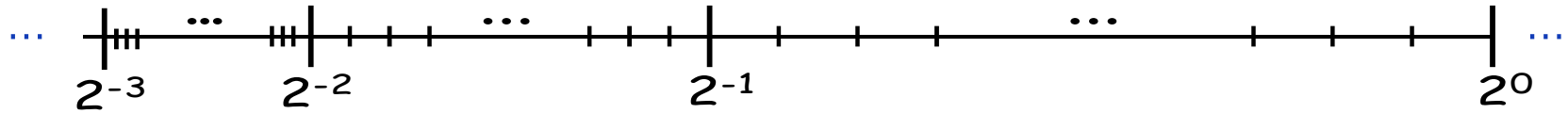
- Nel caso di interi a 32 bit:



- Nel caso di floating-point:



Nota: il numero di valori rappresentati con 32 bit è circa lo stesso nel caso INT e in quello FP... la densità dei punti non è costante, quindi.



IEEE 754 floating-point mantissa ed esponente

- **Mantissa**
  - Rappresentazione in base due con Normalizzazione (e modifica dell'esponente) in modo tale da riportare un 1 nella cifra più significativa
    - (tale unico) 1 prima della virgola non compare nella codifica
  - I bit dalla mantissa rappresentano un numero decimale (in base 2) >=1 e <2
    - M=1-mantissa= $m_1 \times 2^{-1} + m_2 \times 2^{-2} + \ldots + m_{23} \times 2^{-23}$

Nota: come detto, le cifre oltre la m23 le butto!

- **Esponente**
  - Rappresentazione "polarizzata" (biased) per rendere più facile il sort
    - polarizzazione=127 per la singola precisione, 1023 per la doppia precisione
    - i valori vanno da 1 a 2^e-2 (e=#bit per l'esponente) e codificano gli esponenti da $-2^{e-1}+2$ a $+2^{e-1}-1$ (e.g. -126..+127)
  - Codifiche riservate
    - esponente con tutti 1 e mantissa != 0
      codifica "Not a Number" (NAN) (risultato di 0/0)
    - esponente con tutti 1 e mantissa == 0
      codifica "Infinity" (INF) (risultato di un numero diverso da zero diviso 0)
    - esponente con tutti 0 e mantissa != 0
      codifica "underflow" (numero denormalizzato)

- Ricapitolando: per $(-1)^S \times (1+M) \times 2^{E-\text{polarizzazione}}$ → memorizzo <S,M,E>
  - S=0 o 1, M=mantissa_normalizzata senza "1." iniziale, E=esponente+polarizzazione

Esercizio

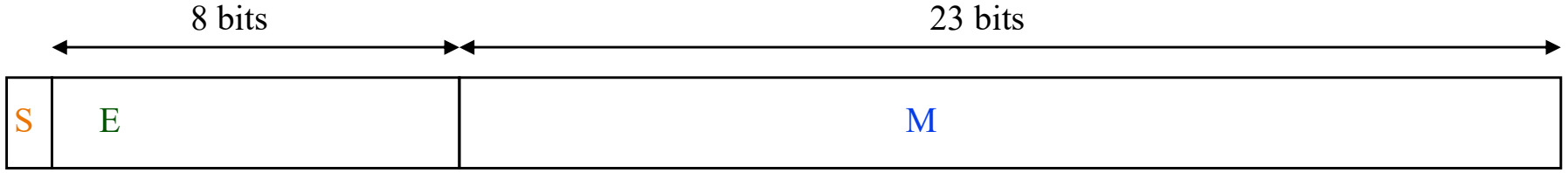
- Rappresentare -0.75 in formato IEEE single precision
  - rappresentazione decimale: $-0.75 = -3/4 = -3 \times 2^{-2}$
  - rappresentazione binaria: $-11 \times 2^{-2} = -1.1 \times 2^{-1}$
  - rappresentazione binaria normalizzata: -1.1×2^{-1}

  - segno=negativo
    → S=1
  - mantissa_normalizzata=1.1000 0000 0000 0000 0000 000
    → M = 1000 0000 0000 0000 0000 000
  - esponente=-1
    → E=esponente+polarizzazione = -1 + 127 =126 → 01111110

  - IEEE single precision: 10111111010000000000000000000000

  - Nota: la mantissa è stata normalizzata. Per questo tutti i bit più significativi risultano spostati verso SINISTRA.

Floating-Point: single precision (32 bits)



Esempi con mantissa ed esponente positivi/negativi:

$+1.1010001 \cdot 2^{+10100}$ → 0 10010011 10100010000000000000000

$-1.1010001 \cdot 2^{+10100}$ → 1 10010011 10100010000000000000000

$+1.1010001 \cdot 2^{-10100}$ → 0 01101011 10100010000000000000000

$-1.1010001 \cdot 2^{-10100}$ → 1 01101011 10100010000000000000000

Esempi di numeri "notevoli":

0 → 0 00000000 00000000000000000000000

1 → 0 01111111 00000000000000000000000

minore rappresentabile (2^{-126}) → 0 00000001 00000000000000000000000

maggiore rappres. $(2-2^{-23}) \cdot 2^{127}$ → 0 11111110 11111111111111111111111

∞ → 0 11111111 00000000000000000000000

NaN → 0 11111111 XXXXXXXXXXXXXXXXXXXXXXX

Nota: $2^{-126} \approx 1.18 \cdot 10^{-38}$, $(2-2^{-23}) \approx 3.4 \cdot 10^{38}$

Modalità di arrotondamento IEEE-754

- IEEE-754 prevede 4 modalità di arrotondamento
  - round to nearest: arrotonda al più vicino rappresentabile
  - round to+∞: arrotonda al primo numero rappresentabile NON INFERIORE al valore da arrotondare (eventualmente arrotonda a +inf)
  - round to-∞: arrotonda al primo numero rappresentabile NON SUPERIORE al valore da arrotondare (eventualmente arrotonda a -inf)
  - round to 0: arrotonda al più vicino rappresentabile con modulo NON SUPERIORE al valore da arrotondare
- Nel caso di "round to nearest" ci può essere ambiguità quando il numero da arrotondare sia esattamente equidistante dai due numeri più vicini rappresentabili
  - Lo standard adotta la regola "round ties to even", ovvero in caso di pareggio va scelto il più vicino rappresentabile PARI
  - Il vantaggio rispetto alla modalità "round ties to away (from 0)" (supportata nella revisione dello standard IEEE-754-2008) è che in media l'errore di arrotondamento si annulla**
  - L'implementazione di "round ties to even" è più complessa

**Goldberg, D. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Comp. Surveys* 23, 1 (Mar. 1991), 5-48. DOI= http://doi.acm.org/10.1145/103162.103163

Esempi di arrotondamento

numero	r.to 0	r.to nearest	r.to +inf	r.to -inf
1.00000003	1.0	1.0	1.00000012	1.0
1.00000007	1.0	1.00000012	1.00000012	1.0
-1.00000003	-1.0	-1.0	-1.0	-1.00000012
-1.00000007	-1.0	-1.00000012	-1.0	-1.00000012

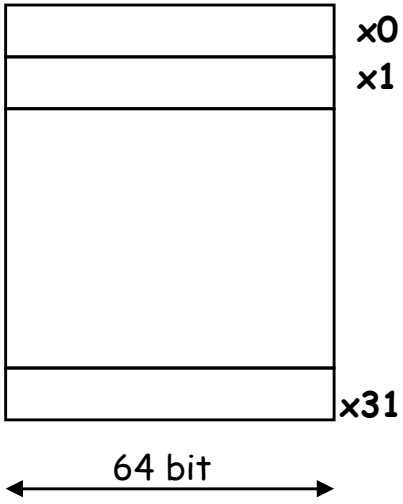
Ricapitolazione

- L'aritmetica del calcolatore è limitata da precisione finita
- I pattern di bit non hanno particolare significato ma esistono standard per:
  - complemento a due
  - IEEE 754 floating point
- Le istruzioni determinano il "significato" dei pattern di bit
- Le prestazioni e l'accuratezza sono importanti: ci sono diverse complessità nelle macchine reali
- Lo standard è stato rivisto nel Giugno 2008 come IEEE 754-2008: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4610935&isnumber=4610934
  - Fra le novità il supporto per floating point a 128-bit
  - Aritmetica floating point decimale (oltre che binaria)

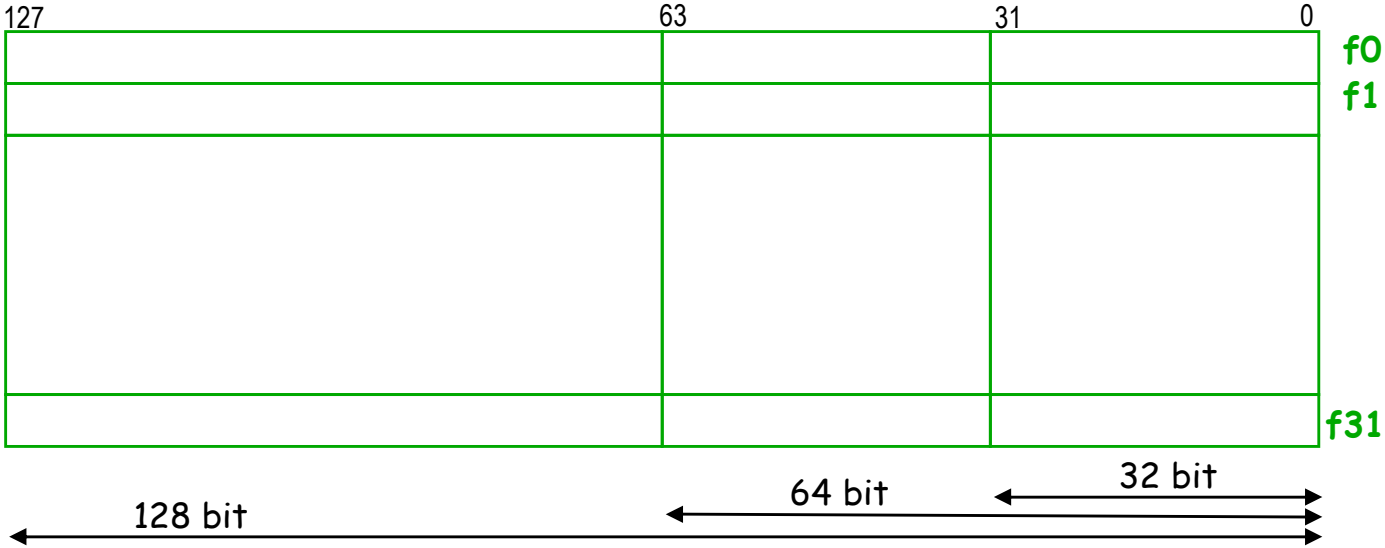
OPERAZIONI FLOATING-POINT NEL PROCESSORE RISC-V

Supporto architetturale per le op. floating-point nel RISC-V

REGISTRI PER
OPERAZIONI SU INTERI:



REGISTRI **AGGIUNTIVI**
PER OPERAZIONI SU FLOATING-POINT (s=32 bit, d=64bit, q=128bit):



- **fadd.s f0, f1, f2** **lavora sui registri fp a 32 bit (estensione 'F' del RISC-V)**
- **fadd.d f0, f2, f4** **lavora su registri fp a 64 bit (estensione 'D ' del RISC-V)**
- **fadd.q f0, f2, f4** **lavora su registri fp a 128 bit (estensione 'Q' del RISC-V)**

→ **Sta al programmatore di utilizzare correttamente i contenuti dei registri**

- **Ad es., se uso fadd.d f0,f1,f2 e subito dopo pretendo di usare fadd.s f4,f3,f0 sicuramente commetto un errore dato che i bit meno significativi di f0 codificano il valore a singola precisione in maniera diversa di quello a singola precisione→attenzione!**
- **Per usare correttamente operandi di precisione diversa ci sono istruzioni di conversione, es. fra double e single**

Istruzioni floating point nel processore RISC-V

- ARITMETICHE (dove 'z' può essere s per single, d per double, q per quad)

```
fadd.z  f1, f2, f3      # somma
fsub.z                  # sottrazione
fmul.z                  # moltiplicazione
fdiv.z                  # divisione
fsqrt.z  f1,f2          # radice quadrata
fmv.z   f1,f2           # spostamento fra reg. fp
fabs.z   f1,f2          # valore assoluto
fneg.z   f1,f2          # valore contrario (negato)
```

In blu-chiaro: pseudoistruzioni basate su FSIGNJ

- MEMORIA

```
flw /  fld / flq   f1, 100(x5)   # flw:32bit, fld:64bit, flq:128bit
fsw / fsd / fsq  f1, 100(x5)
```

- SPOSTAMENTO int←→fp

```
fmv.x.w   x5, f1
fmv.w.x   f1, x5
fmv.x.d   x5, f1
fmv.d.x   f1, x5
```

Nota: qua x signfiica registro intero

Nota2: lo spostamento di quantità 'q' non esiste

- SPOST. CON CONVERSIONE

```
fcvt.s.w  / fcvt.s.wu   f1, x5
fcvt.w.s  / fcvt.wu.s   x5, f1
fcvt.d.w  / fcvt.d.wu   f1, x5
fcvt.w.d  / fcvt.wu.d   x5, f1
```

W = intero con segno (word)
WU=intero senza segno (word-unsigned)

Istruzioni condizionali floating-point

- Sono nella forma

```
fcc.s  x5, f2, f4
```

dove 'cc' indica una condizione di confronto e può essere:

'lt' → less than

'le' → less than or equal

'eq' → equal

- Perché non ci sono i casi (gt, ne, ge)?
  Nel RISC-V si è scelto di non inserire istruzioni per tali casi in quanto basta… scambiare gli operandi* per trattarli!
  - Il risultato va su un registro intero
  - Esempio:

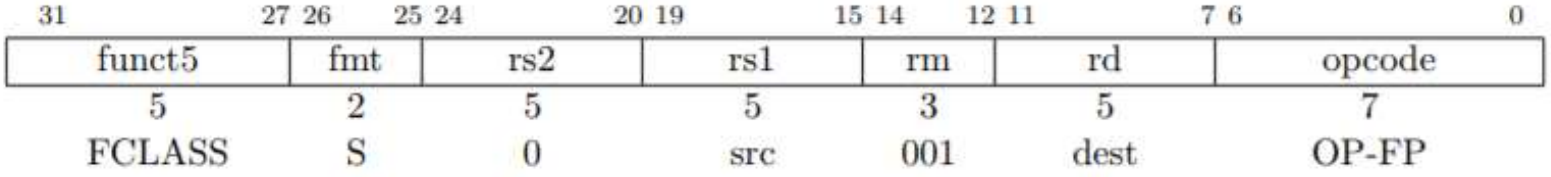
```
flt.s  x5, f2, f4
beq    x5, x0, L1
```

* oppure, es., usare bne invece beq nell'uso del valore booleano della condizione

Istruzione FCLASS.x

- FCLASS.z permette di sapere se il numero FP è normalizzato o meno, +/-INF, +/-0, NaN
  - dal manuale "RISC-V User-Level ISA V2.2"

The FCLASS.S instruction examines the value in floating-point register *rs1* and writes to integer register *rd* a 10-bit mask that indicates the class of the floating-point number. The format of the mask is described in Table 8.5. The corresponding bit in *rd* will be set if the property is true and clear otherwise. All other bits in *rd* are cleared. Note that exactly one bit in *rd* will be set.



rd bit	Meaning
0	*rs1* is $-\infty$.
1	*rs1* is a negative normal number.
2	*rs1* is a negative subnormal number.
3	*rs1* is -0.
4	*rs1* is $+0$.
5	*rs1* is a positive subnormal number.
6	*rs1* is a positive normal number.
7	*rs1* is $+\infty$.
8	*rs1* is a signaling NaN.
9	*rs1* is a quiet NaN.

RISC-V Floating-Point ABI (Application Binary Interface)

- Definisce la convenzione di uso dei registri nel software
- Valido nel caso single e in quello double precision

  - f10,f11 (chiamati fa0,fa1) → argomenti delle funzioni o valori restituiti
  - f12-f17 (chiamati fa2,fa7) → argomenti ulteriori di funzioni

  - f8,f9,f18-f27 (chiamati fs0-fs11) → registri "saved"

  - f0-f7,f28-f31 (chiamati ft0-ft11) → registri temporanei

Alcune syscall che coinvolgono i numeri floating point

- **'ecall' serve per chiedere un servizio al sistema operativo (chiamata anche 'system call')**
  - **PARAMETRI DI INGRESSO/USCITA**
    - **a7 deve contenere il numero del servizio**
      - 2: stampa un single-precision float a video (fa0 contiene il float da stampare)
      - 6: leggi un single-precision float dalla tastiera (fa0 in uscita, contiene il float letto in binario)
      - 3: stampa un double-precision float a video (fa0 contiene il double da stampare)*
      - 7: leggi un double-precision float dalla tastiera (fa0 contiene il float letto in binario)*

*Nota: non disponibile in RARS 1.3

Lezione 13 - Valutazione delle Prestazioni

- Le prestazioni sono in unità di "oggetti-per-secondo"
  - "grande" è meglio
- Se siamo soprattutto interessati al tempo di risposta
  - $P_X = \frac{1}{E_X}$
  - P_X = prestazioni della macchina X
  - E_X = tempo di esecuzione della macchina X
- Speed up rispetto a una macchina di riferimento
  - $S_{XY} = \frac{P_X}{P_Y} = \frac{E_Y}{E_X}$
  - Lo speed up di X rispetto a Y dice "X è S_{XY} volte più veloce di Y"
  - In percentuali, $s_{xy}=(S_{XY}-1)*100$ dice "X è s_{xy}% più veloce di Y"
- Esempio:
  - E_X = 1sec, E_Y = 1.2sec $\Rightarrow$ S_{XY} = 1.2 $\Rightarrow$ X è 1.2 volte più veloce di Y
    s_{xy} = 20% $\Rightarrow$ X è il 20% più veloce di Y

Tempo Totale e Tempo di CPU

- T_J = Tempo Totale
  (o Tempo Assoluto o Tempo di Risposta o Tempo Trascorso (Elapsed Time))
  - tempo necessario per completare un lavoro; comprende una serie di tempi che sono "allungati" dalle interferenze di altri programmi che generano attività come:
    - accesso ai dischi
    - accesso alla memoria
    - attività di I/O
    - sovraccarico del Sistema Operativo

- T_{CPU} = Tempo di CPU (Tempo di Esecuzione del programma)
  - Tempo durante il quale un processore ha lavorato su un singolo programma
    - senza tempo speso in I/O (può includere il tempo "dell'uomo")
    - senza tempo per eseguire altri programmi
  - Può essere suddiviso in
    - T_U = tempo di CPU relativo all'utente
    - T_K = tempo di CPU relativo al Sistema Operativo (K=Kernel)

$$\Longrightarrow \quad T_U + T_K = T_{CPU}$$

Tempo di CPU misurato in UNIX/LINUX

- Dato il programma `myprogram`

```
time <myprogram>

81.3u   10.6s   2:19   66%
```

T_U

T_K

T_J

$$\frac{T_{CPU}}{T_J} \% = \frac{(T_U + T_K)}{T_J} * 100$$

Per il 34 % la macchina è usata
- per I/O
- per altri programmi
- per altra attività di sistema

- TK spesso non viene incluso
  - perché può nascondere inaccuratezze di misura
  - il calcolo può risultare sbilanciato se si usano sistemi operativi diversi
  - su alcune macchine il codice può essere considerato di kernel e su altre di utente
  - d'altra parte nessun programma può essere eseguito senza usare qualche parte di kernel

- Prestazioni di Sistema $\Rightarrow T_J$ Prestazioni CPU $\Rightarrow T_U$

Equazione delle prestazioni

$$T_{CPU} = C_{CPU} \cdot T_C = \frac{C_{CPU}}{fc} = N_{CPU} \cdot \overline{CPI} \cdot T_C$$

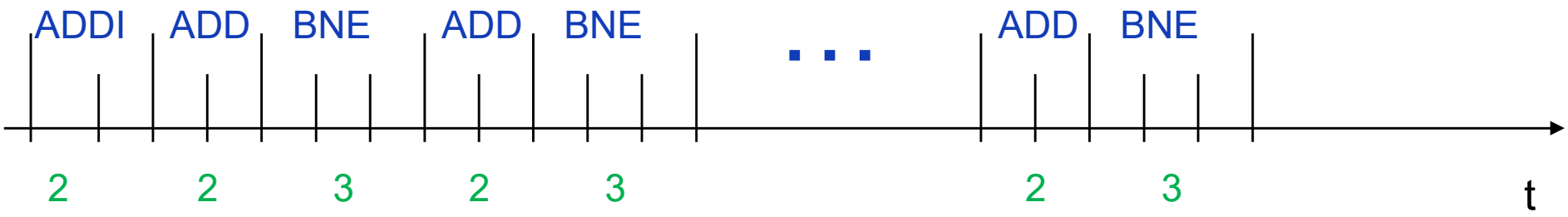
- T_C è il periodo di clock (f_C è la frequenza di clock)
- C_{CPU} sono i cicli di clock
- N_{CPU} è il Numero di istruzioni eseguite DINAMICAMENTE
- /CPI è il numero di Cicli Per Istruzione (medio)

Esempio:
bne → 2 cicli
add → 1 ciclo
2 istruzioni, 3 cicli → /CPI=3/2=1.5

Esempio 1

Assumiamo t0=0, t1=0 :

```
        addi t1, t1, 100
loop:   add  t0, t0, 1
        bne  t0, t1, loop
```



C_{CPU}=502, N_{CPU}=201

/ CPI= 502/201 ≈ 2.5

- Per ottenere maggiori prestazioni si può:
  - ridurre i cicli di clock per un programma
  - aumentare la frequenza di clock
- Spesso però l'abbassamento del numero di cicli comporta invece un abbassamento della frequenza di clock

Esempio 2

- $T_{CPU}(A) = 10s$ $f_C(A) = 400$ MHz ← MACCHINA A
- $T_{CPU}(B) = 6s$ $f_C(B) = ?$ ← MACCHINA B

Supponiamo che in B si riesca a ridurre il tempo ma con il risultato di avere

$$C_{CPU}(B) = 1.2 * C_{CPU}(A)$$

- Quanto deve essere la frequenza di clock della macchina B affinché il tempo di esecuzione del programma sia 6s?

- Soluzione:

$$C_{CPU}(A) = (f_C * T_{CPU})|_A = 4 * 10^9$$

$$f_C(B) = (C_{CPU} / T_{CPU})|_B = (1.2 * 4 * 10^9 / 6) = 800 \text{ MHz}$$

- Per ottenere una diminuzione del 40% del tempo di esecuzione è necessario raddoppiare la frequenza di clock

Esempio 3

- $f_C(A)$ = 1 GHz $f_C(B)$ = 500 MHz
- /CPI(A) = 2.5 /CPI(B) = 1.2
- Quale calcolatore è più veloce?
  (il set di istruzioni rimane lo stesso)

- $S_{AB} = \frac{T_{CPU}(B)}{T_{CPU}(A)} = \frac{N_{CPU}(B)*/CPI(B)*f_C(A)}{N_{CPU}(A)*/CPI(A)*f_C(B)} = \frac{1.2*1000}{2.5*500}$
- set istruzioni uguale → $N_{CPU}(A) = N_{CPU}(B)$
- S_{AB}= 0.9 → B è 1.1 volte più veloce di A

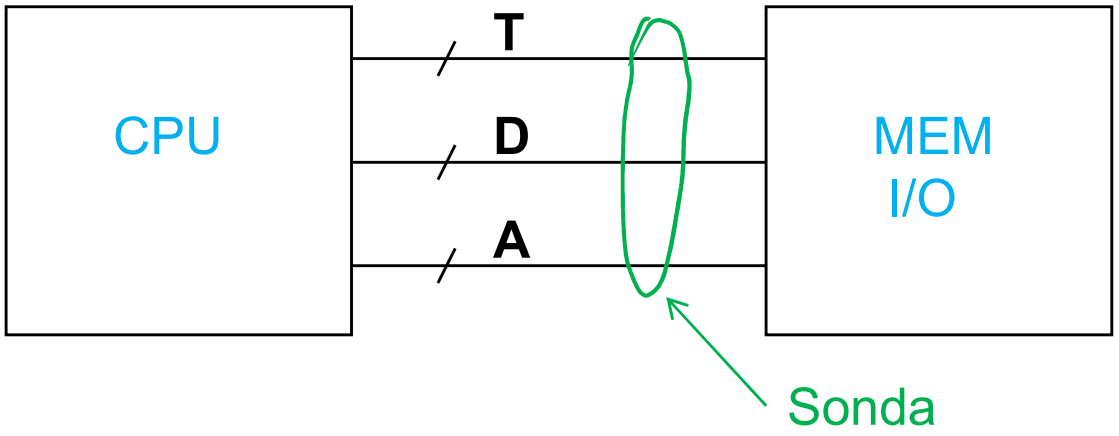
Motivi delle dipendenze dei parametri fondamentali

- **Programma**
  - Fibonacci-ricorsivo vs. Fibonacci-iterativo → N_{CPU}
- **Compilatore**
  - Opzioni –O1, -O2, -O3 → N_{CPU}, /CPI
- **Sistema Operativo**
  - Una syscall può essere implementata diversamente in Linux rispetto a Windows → N_{CPU}
- **Instruction Set (ISA)**
  - RISC-V vs. x86 vs. ARM, #cicli necessari per es. add → N_{CPU}, /CPI
- **Struttura CPU**
  - Una pipeline elabora più istruzioni per ciclo → /CPI
  - Stadi (logica combinatoria) con meno porte logiche → maggiore fc
- **Sottosistema di memoria**
  - Es. add → 1 ciclo, lw → 100 cicli
- **Tecnologia (elettronica)**
  - Es. 14 nm invece che 90nm

	N_{CPU}	/CPI	fc
Programma	X		
Compilatore	X	(X)	
Sistema Operativo	X		
ISA	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

TCPU	→ orologio
fc	→ dato di targa
NCPU, /CPI	→ difficili da misurare

Come valutare NCPU (1) – Sonda HW



T = TYPE={IF,DR,DW}
FETCH DATA-READ DATA-WRITE

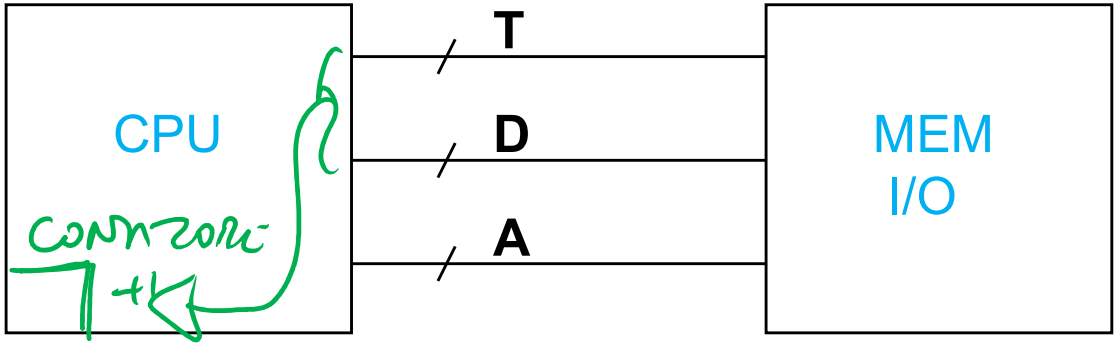
D = DATA

A = ADDRESS

- Strumentazione Hardware (sonda o probe) che legge le istruzioni che il processore preleva dalla memoria

- Richiede di poter aprire la macchina
- Richiede la disponibilità di una opportuna sonda
- Richiede la disponibilità di un analizzatore di stati logici

Come valutare NCPU (2) – Contatori HW



T = TYPE={IF,DR,DW}
FETCH DATA-READ DATA-WRITE

D = DATA

A = ADDRESS

- Contatori Hardware che contino il numero di istruzioni
  - Disponibili sui processori Intel (ed Alpha)

- Supporto da parte del processore e del sistema
  - Alpha: ATOM tools, Intel VTUNE

Come valutare NCPU (3) – Instrumentazione SW (Sonda-SW)

Codice Originale

```
add t0,t1,t2     #1ciclo
lw  t0,0(t1)     #100cicli
bne t3,t4,Loop   #3cicli
```

Codice Instrumentation

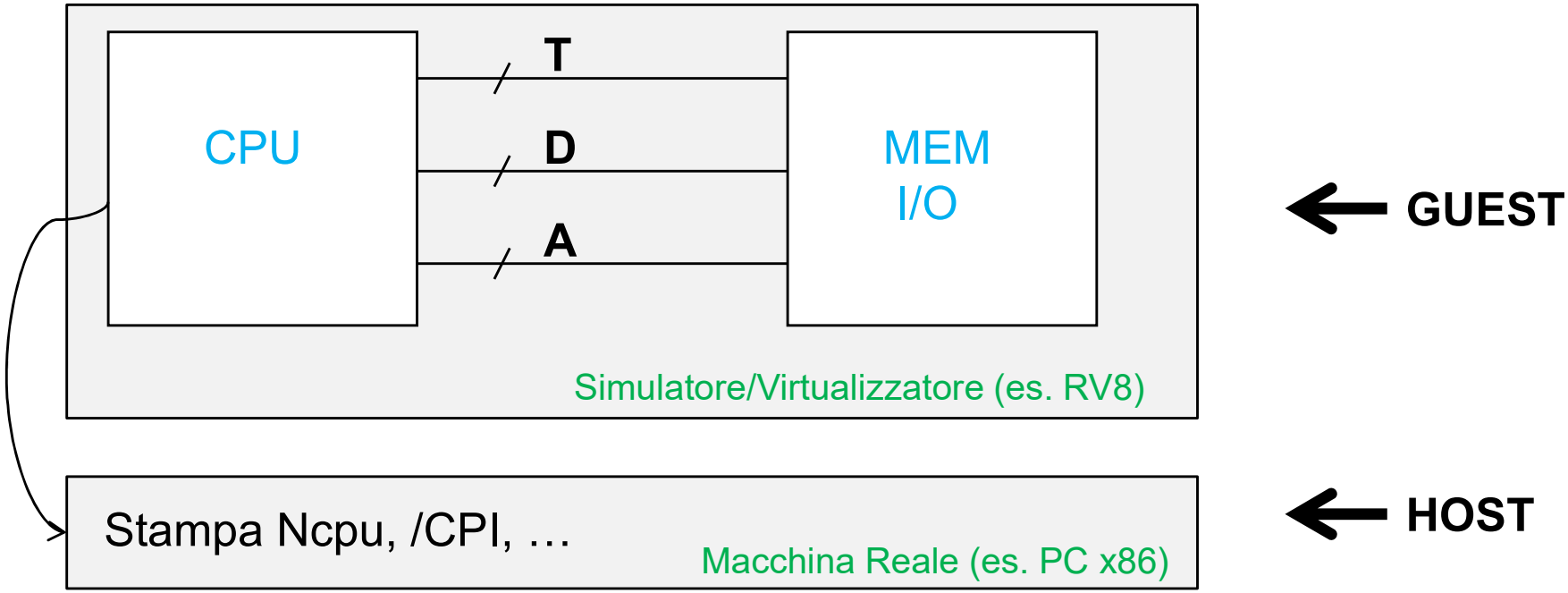
```
add  s8,zero,3
add  s9,zero,104
add  t0,t1,t2
lw   t0,0(t1)
bne  t3,t4,Loop
```

1) Riservo un registro (es. s8) per contare le istruzioni
2) Riservo un registro (es. s9) per contare i cicli di ogni basic-block*
3) Inserisco due istruzioni per tali conteggi per ogni basic-block* del programma (instrumentazioni)
4) Al termine del programma, s8 contiene Ncpu mentre s9/s8 è pari a /CPI

- Metodo usato in vari processori
  - MIPS R4400: Pixie Alpha: Atom Intel: VTUNE

*Basic-block=gruppo di istruzioni con un solo punto di ingresso e un solo punto di uscita

Come valutare NCPU (4) – Virtual-Machine SW



- Usare dei simulatori/virtualizzatori/emulatori
  - RV8 → simulatore di processore RISC-V
  - SPIM → simulatore di processore MIPS
  - QEMU → emulatore molto diffuso sotto Linux → virtual machine
  - Virtualbox, Vmware, VirtualPC → ulteriori virtual machines

Come calcolare /CPI

Si può calcolare da quante istruzioni di ogni tipo vengono eseguite ($N_{CPU,K}$)

$$CICLI = N_{CPU} \cdot \overline{CPI} \qquad CICLI = \sum_{K=1}^{T} N_{CPU,K} \cdot CPI_K$$

$$\overline{CPI} = \sum_{k=1}^{T} \overline{CPI}_k = \sum_{k=1}^{T} p_{CPU,k} \cdot CPI_k = \sum_{k=1}^{T} \frac{N_{CPU,k}}{N_{CPU}} \cdot CPI_k$$

T = numero tipi diversi di istruzioni
$N_{CPU,K}$ = numero di istruzioni di tipo K
$p_{CPU,k}$ = frequenza relativa di quel tipo di istruzione $\equiv N_{CPU,K}/N_{CPU}$
CPI_K = CPI delle istruzioni di tipo K
$/CPI_K$ = CPI **MEDIO** delle istruzioni di tipo K $\equiv p_{CPU,k} * CPI_K$

Op	$p_{CPU,K} \equiv N_{CPU,K}/N_{CPU}$	CPI_K	$/CPI_K \equiv p_{CPU,k} * CPI_K$	$\%T_{CPU,k} = CICLI_k/CICLI$ $/CPI_K : /CPI$
Arithmetic	0.50	1	0.5	33%
Load	0.20	2	0.4	27%
Store	0.10	2	0.2	13%
Branch	0.20	2	0.4	27%

/CPI = 0.5+0.4+0.2+0.4 = 1.5

Scalabilità

$$P := Prestazioni \qquad P \propto 1/T_{CPU} = \frac{1}{N_{CPU} \cdot \overline{CPI}} \cdot f_C$$

- Diciamo che un sistema ha prestazioni scalabili con la frequenza se, con $f_{200}/f_{100}=2$, ottengo $P_{200}/P_{100}=2$
  - Abbiamo visto però che, aumentando la frequenza, è verosimile che il CPI aumenti… es. $CPI_{200}=1.2*CPI_{100}$; in tal caso si otterrebbe $P_{200}/P_{100}=f_{200}/f_{100}*CPI_{100}/CPI_{200}=2/1.2\approx1.6$ quindi *poca scalabilità* (<2)
- Questo è ciò che si è verificato storicamente nei Pentium
  - Per es., se il sottosistema di memoria 'non tiene il passo del processore'
  - Risolto nei PentiumPro mantenendo uno stesso CPI all'aumentare di f_C
- La capacità di fornire maggiori prestazioni al variare di un parametro viene chiamata *scalabilità rispetto a quel parametro*
  - Per es., il processore PentiumPro risulta *più scalabile* del processore Pentium *con la frequenza* ($Scalabilita`(f)_{PentiumPro} > Scalabilita`(f)_{Pentium}$)
- Diciamo che un sistema ha *scalabilità ideale lineare* se

$$Scalabilita`(p) = k \cdot p$$

dove (k=costante, p=parametro)

Legge di Amdhal

- In generale può essere difficile stabilire l'esatta dipendenza da un dato parametro (anche quando questo compare esplicitamente nell'equazione!)

- Per questo motivo viene utilizzata la seguente formula, per mettere in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema:

$$S = \frac{P_{dopo}}{P_{prima}} = \frac{T_{prima}}{T_{dopo}} = \frac{T_{prima}}{\frac{T_{migliorabile}}{a} + T_{NON-migliorabile}} = \frac{1}{\frac{p}{a} + (1-p)}$$

speedup prestazioni tempo

dove
p := parte del tempo 'migliorabile'=Tmigliorabile/Tprima
a := speed-up della parte di sistema migliorabile

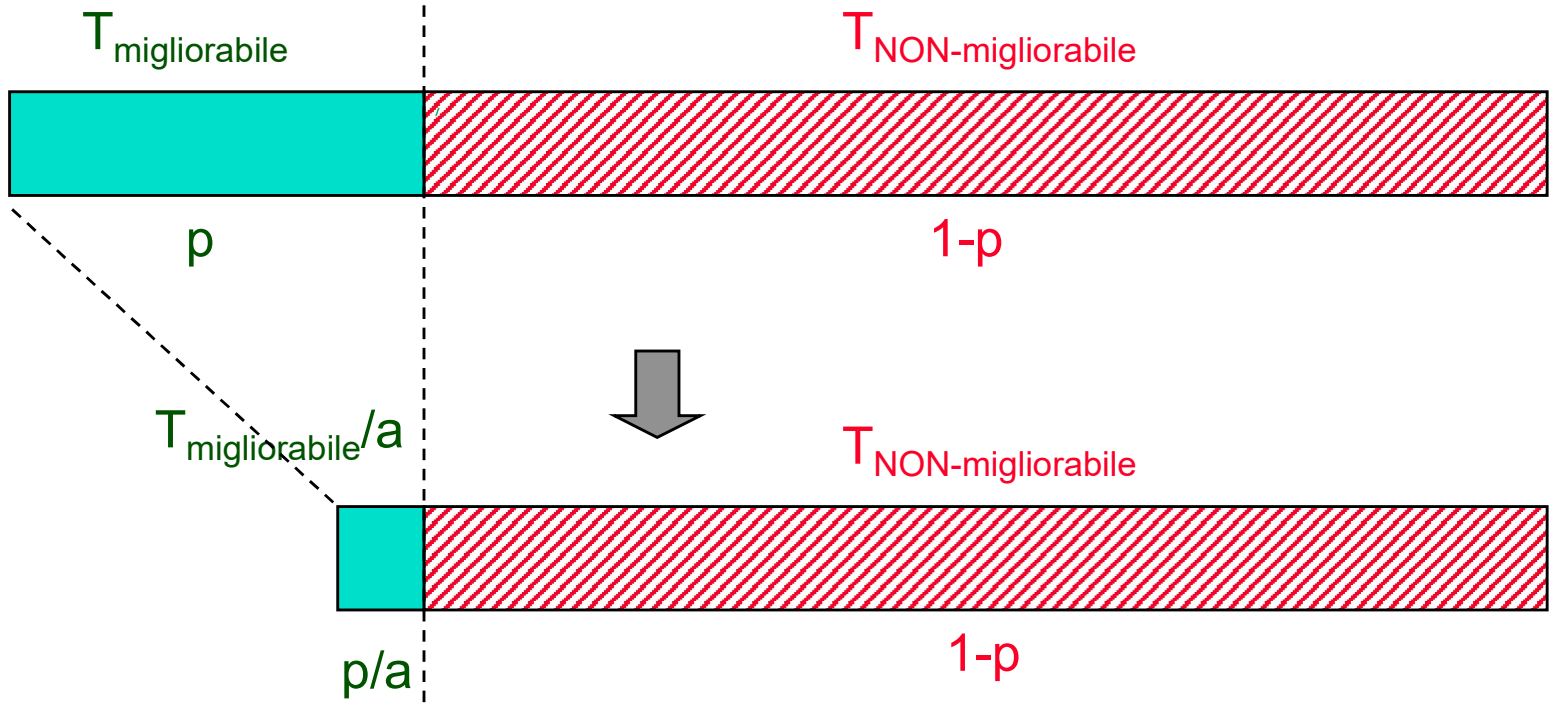
Quindi: $0 \leq p \leq 1$

Mentre: $a \geq 1$

Legge di Amdhal (2)

Dimostrazione "grafica" della legge di Amdhal

$$S = \frac{1}{\frac{p}{a} + (1 - p)}$$



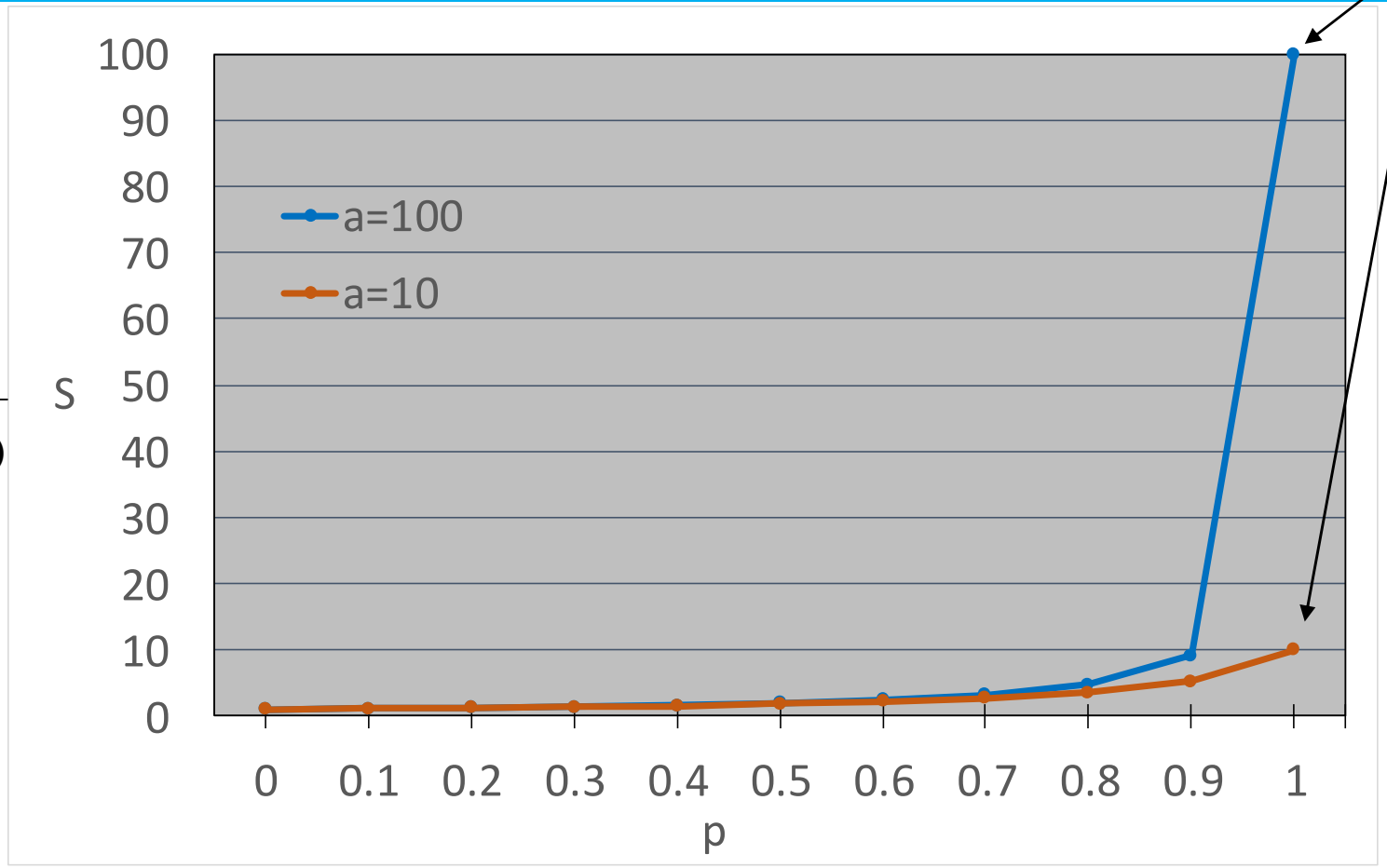
p=parte del tempo 'migliorabile'=$T_{migliorabile}/T_{prima}$

a=speed-up della parte di sistema migliorabile

Legge di Amdhal (3) – S al variare di p

Nota: questo punto (per p=1) vale 'a'

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$



→lo speed-up di prestazioni S assume valori significativamente alti solo quando la parte migliorata p è abbastanza grande (es. p>0.8)

→ è importante concentrare i miglioramenti su una parte del sistema che sia <u>consistente</u>

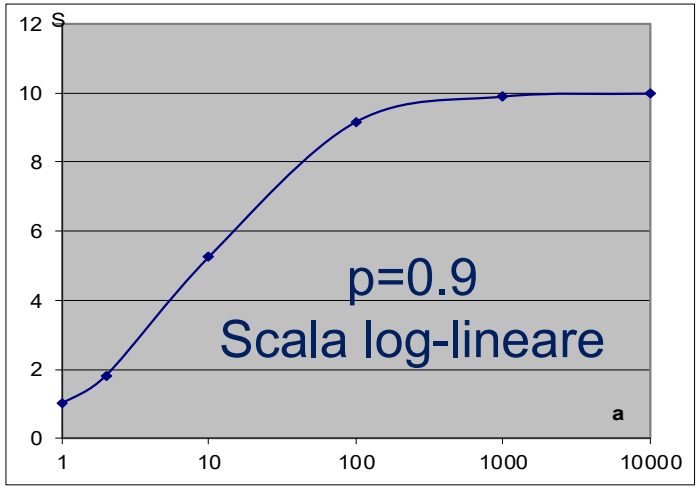
Legge di Amdhal (4) – S al variare di a

- Supponiamo che il 90% (→p=0.9) di un programma possa essere eseguita in parallelo (es. aggiungendo più processori)

$$S = \frac{1}{\frac{p}{a} + (1-p)}$$

$$\lim_{a \to \infty} S = \frac{1}{1-p}$$

a	S	p
1	1.000	0.9
2	1.818	0.9
10	5.263	0.9
100	9.174	0.9
1000	9.911	0.9
10000	9.991	0.9

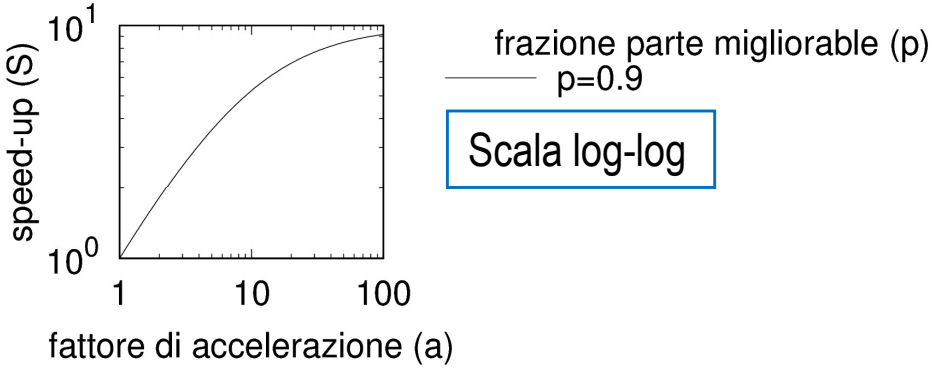


→lo speed-up di prestazioni S ottenibile migliorando una parte p è limitato da 1/(1-p)

Nell'esempio in figura, per a→ ∞ il massimo speed-up ottenibile è pari a 1/0.1=10. Quindi invece di aggiungere 9999 processori conviene parallelizzare anche una parte del restante 10%

Nota:

Se faccio lo stesso disegno in scala log-log:



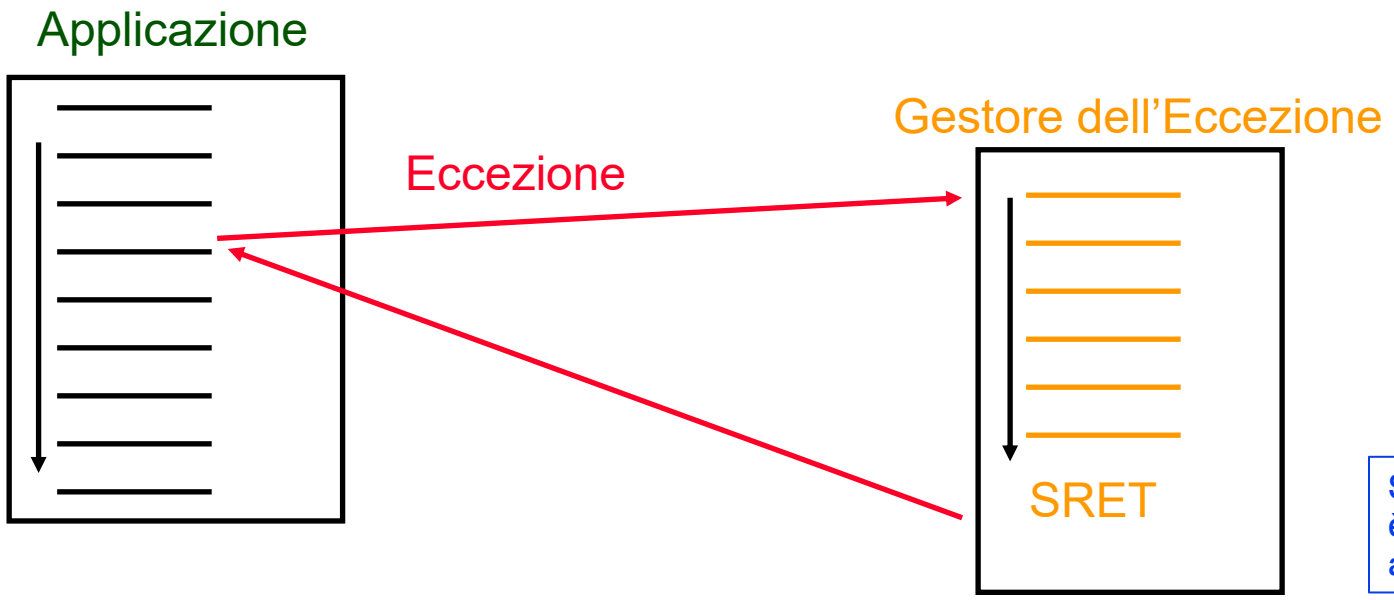
Conclusioni dalla legge di Amdhal

1) è molto importante migliorare la parte più utilizzata del sistema!

2) A un certo punto è importante anche migliorare
la parte meno utilizzata del sistema…
(ovvero spingere il più possibile p a valori molto alti)

Lezione 14
Il meccanismo di interruzione del processore

Patterson: 4.9



SRET=Supervisor Return
è una istruzione speciale
a conclusione del gestore

- **Eccezione** = trasferimento del flusso di controllo non legato strettamente al codice utente
  - Il sistema gestisce l'eccezione eseguendo una funzione speciale denominata "Gestore dell'eccezione"
- Esempio
  - l'utente preme un tasto e questo interrompe l'esecuzione del programma utente (qualunque esso sia) e la CPU esegue un "gestore della pressione di un tasto" che tipicamente legge un codice corrispondente al tasto e lo trasferisce in un apposito buffer relativo ai tasti premuti

Comportamento del Calcolatore durante un'eccezione

- Il programma in esecuzione deve essere sospeso e poi riattivato
  - Se l'eccezione è dovuta all'errore di una istruzione, l'istruzione "colpevole" può essere riprovata o simulata e il programma può o continuare o essere bloccato (abort)

  →è necessario che il Calcolatore salvi il punto del codice (program counter) in cui si è verificata l'eccezione (1)

- Il controllo passa quindi ad una "funzione speciale" denominata "Gestore dell'Eccezione" (Exception Handler) o "routine agganciata"
  - Il gestore dell'eccezione deve rimediare alla situazione
  - A seconda del tipo di eccezione, il gestore esegue azioni diverse

  →è necessario che il Calcolatore identifichi e salvi la causa dell'eccezione (2)

- Al momento dell'eccezione, nel processore RISC-V, queste informazioni vengono automaticamente salvate nei registri speciali SEPC e SCAUSE:

  (1) SEPC → il punto del codice in cui si verifica l'eccezione

  (2) SCAUSE → la causa dell'eccezione

Che ne è dell'istruzione che era in esecuzione?

- L'architettura RISC-V definisce l'istruzione che causa un'eccezione come una "istruzione senza effetto"
- La Routine di Gestione dell'eccezione deve evitare assolutamente di modificare lo stato della macchina

  → I contenuti di TUTTI i registri x1…x31 (e anche f0...f31) NON devono essere alterati

- Durante l'esecuzione dell'eccezione la CPU deve avere pieno controllo di tutte le risorse del Calcolatore

  → Il codice della routine di eccezione deve essere eseguito in una modalità protetta denominata "modo supervisor" (o modo sistema) – vedi slide successive

- Inoltre le eccezioni devono essere servite una alla volta

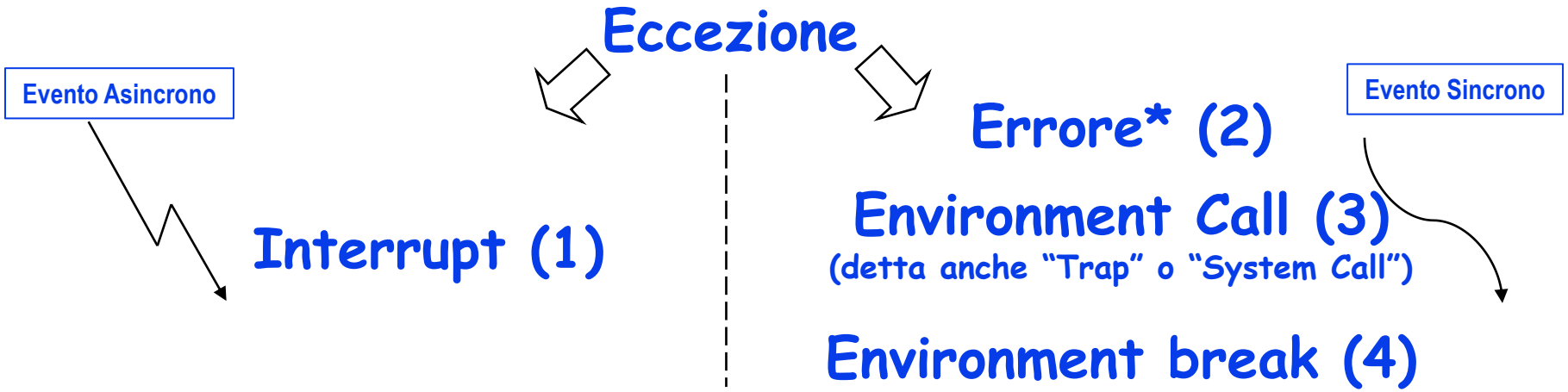
  →La prima cosa da fare è di disabilitare le eccezioni (ad HW); se la routine di gestione lo ritiene opportuno può ri-abilitarle

- Nel processore RISC-V è cura del Gestore dell'Eccezione di salvare i registri che usa; inoltre vengono automaticamente impostati i 2 bit che indicano modalità supervisor ed eccezioni disabilitate, rispettivamente

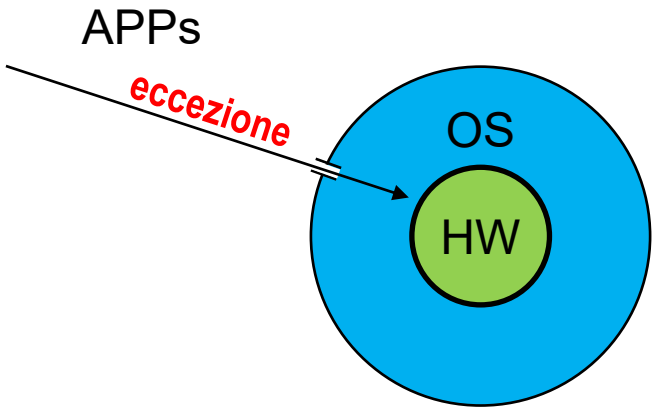
Modalità User/Supervisor

- Il calcolatore può gestire sé stesso utilizzando due modalità di esecuzione denominate classicamente
  - Modalità User
  - Modalità Supervisor
- Nella modalità Supervisor viene in particolare eseguito il Sistema Operativo, che può essere visto come un programma speciale in esecuzione sul calcolatore in una modalità privilegiata in cui:
  - Tutte le risorse del calcolatore possono essere utilizzate direttamente
  - Al software utente vengono presentate risorse "virtuali" che sono più potenti delle risorse fisiche, es.:
    - si forniscono i "file" anziché i "settori del disco"
    - si fornisce una memoria molto più grande (memoria virtuale)
  - Viene garantita la protezione fra i vari programmi utente o di più utenti
- Le eccezioni consentono al sistema di rispondere ad eventi che si verificano mentre un qualsiasi programma utente è in esecuzione
  - Il Sistema Operativo entra in azione dal momento in cui inizia ad essere eseguita la routine di gestione delle eccezioni

Classificazione Generale delle Eccezioni



- L'eccezione permette alle Applicazioni (APPs) di avere accesso al Calcolatore (HW) in modo controllato, attraverso codice che sta nel Sistema Operativo (OS/Kernel):

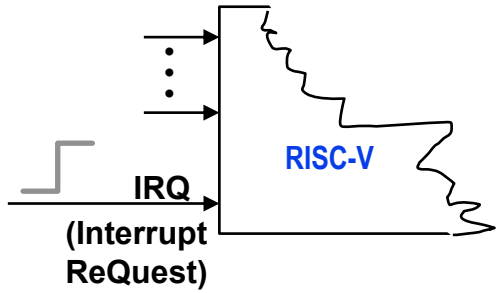


- Durante l'eccezione il Calcolatore esegue codice del Sistema Operativo e diremo quindi che si trova in modalità Supervisor (o modalità Kernel)
- Le APPs vengono invece normalmente eseguite in modalità User (senza poter fare accesso alle risorse privilegiate HW del Calcolatore)

*Nota: L' "Errore" è una condizione eccezionale, quale un overflow durante l'esecuzione di una istruzione e viene talvolta denominato esso stesso semplicemente col termine "Eccezione"

Interrupt, Errori, Environment Call, Debug

1) Interrupt (interruzione)



- Eccezione causata da eventi esterni
  - Pressione di un tasto
  - Movimento del mouse
  - …
- Asincrona rispetto all'esecuzione del programma
- La sua gestione "accade" fra istruzioni di un altro programma

2) Errore

- Eccezione causata da eventi interni
  - Condizioni eccezionali (overflow, divisione per zero)
  - Errori del sistema (parità)
  - Gestione della memoria virtuale (page fault)
- Sincrona rispetto all'esecuzione del programma

3) Environment Call (istruzione `ecall` del RISC-V)

- Eccezione (sincrona) causata da richiesta di un Servizio di Sistema
  - Stampa di un messaggio
  - Lettura di un intero
  - …

4) Environment Break (istruzione `ebreak` del RISC-V)

- Eccezione (sincrona) causata da motivi diagnostici o debugging

Routine di Gestione dell'Eccezione (Exception Handler)

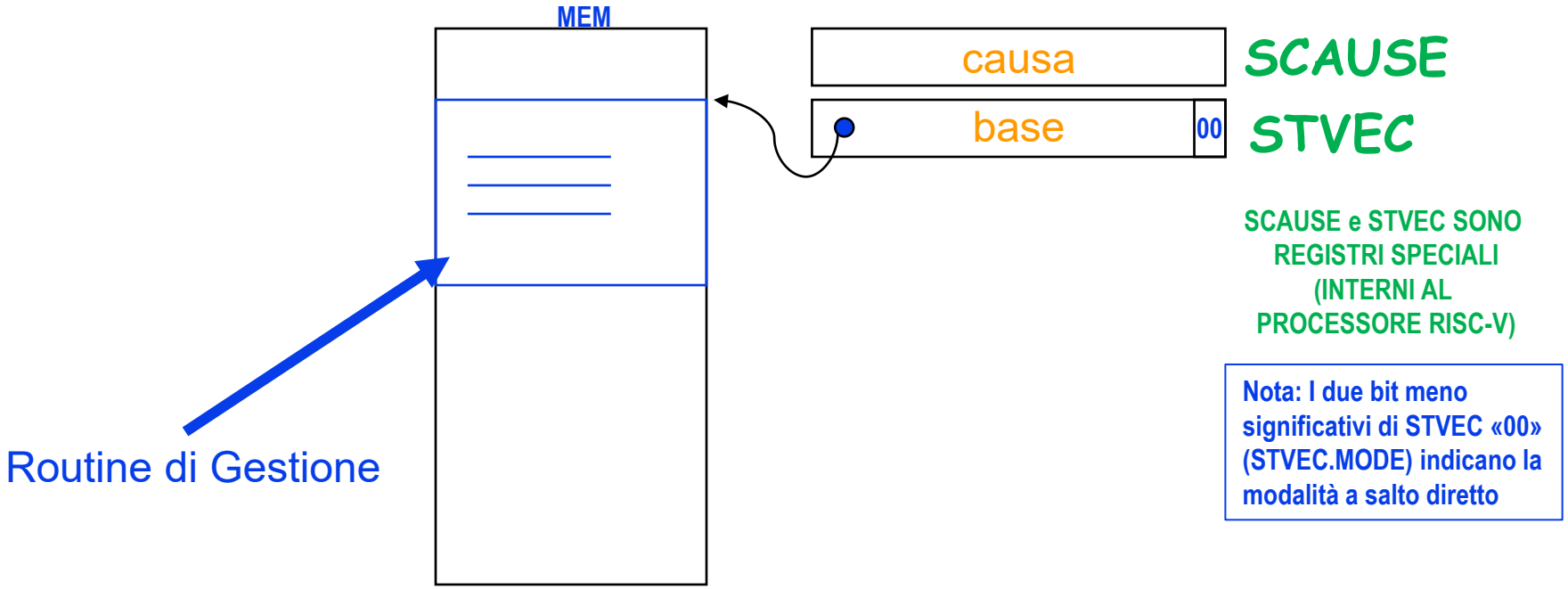
- Esistono vari metodi per implementarla, ad esempio:
  - Salto diretto all'indirizzo della routine di gestione
  - Vettore di Interruzione

- Lo stato della macchina deve essere preservato
  - Ad es. salvando i registri temporanei nello stack

Sia 'causa' il numero di identificazione dell'eccezione

Salto diretto ad indirizzo della routine di gestione

- Salto diretto ad un indirizzo specifico: PC ← base
  - Nel RISC-V è contenuto nel registro speciale STVEC
- Vantaggi:
  - Non è necessario fare un accesso in memoria per prelevare l'indirizzo della routine di gestione (più veloce)
- Svantaggi:
  - Nella routine di gestione occorre analizzare la causa dell'eccezione

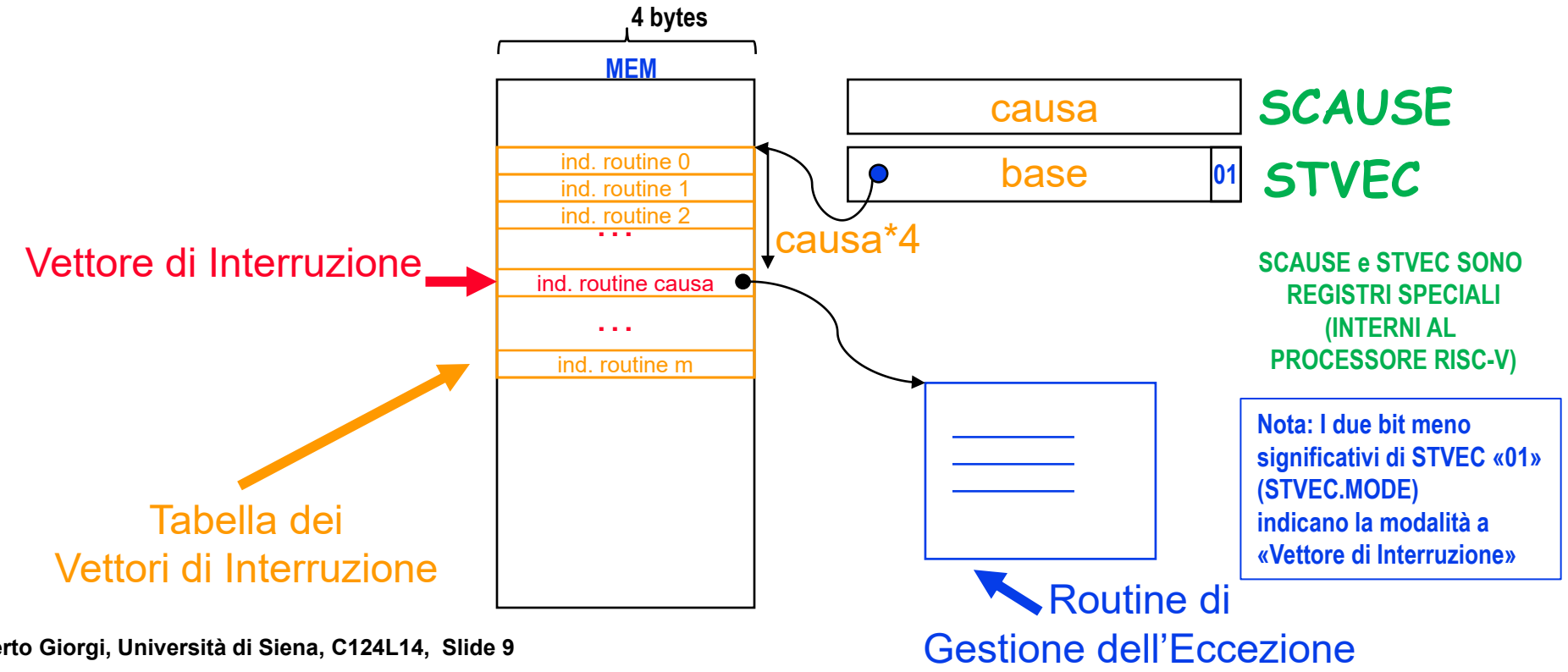


Vettore di Interruzione

- Memorizzo in una tabella gli indirizzi delle Routine di Gestione associata ad ogni possibile causa (usato da RISC-V e storicamente da 370, 68000, Vax, 80x86, …):

PC ← MEM[base+causa*4]

dove 'base' è l'indirizzo base di tale tabella



Salvataggio dello stato della macchina al verificarsi di un'eccezione

- Salvataggio sullo stack (Push)
  - Vax, 68k, 80x86 (intero set dei registri)
  - RISC-V, MIPS (solo i registri necessari, potenzialmente zero)

- Registri ausiliari (Shadow Registers)
  - M88k, ARM
  - Lo stato è salvato in registri ausiliari sia visibili che non visibili direttamento all'utente (es. registri interni della pipeline)

- Salvataggio in registri speciali
  - RISC-V, MIPS
  - Registri Speciali: EPC, CAUSE, STATUS, TVAL (o BadVaddr), ...

Supporto alle gestione delle eccezioni nel RISC-V

- In RV64 ci sono vari registri speciali (a 64 bit)
  - SEPC
    contiene l'indirizzo dell'istruzione "colpevole"
  - SSTATUS
    contiene i bit di abilitazione globale degli interrupt
  - SCAUSE
    i bit 63 e [3:0] codificano le possibili sorgenti di eccezione, ad es.:
    - illegal instruction={0,2}
    - breakpoint={0,3}
    - timer interrupt={1,5}
    - external interrupt={1,9}

    bits 3:0
    bit 63
  - STVAL (Supervisor Trap Value)
    contiene l'indirizzo di memoria al quale si è verificato un riferimento di memoria "sbagliato" (anche chiamato "BadVAddr")
  - SIP (Supervisor Pending Interrupts)
    monitoraggio degli interrupt in attesa
  - SIE (Supervisor Interrupt Enable)
    Abilitazioni più fini per gli interrupt
  - STVEC (Supervisor Trap Vector)
    indirizzo base della lista dei «vettori di interrupt»
  - SSCRATCH: registro per salvataggi temporanei

Nota: il prefisso "S" sta sempre ad indicare il modo "Supervisor"

Nota2: il bit 63 di SCAUSE specifica se si tratta di interrupt o eccezione

Dove si trovano tali registri

- STATUS registro CSR 0x100
- SEPC registro CSR 0x141
- SCAUSE registro CSR 0x142
- STVAL registro CSR 0x143
- SIP registro CSR 0x144
- SIE registro CSR 0x104
- STVEC registro CSR 0x105
- SSCRATCH registro CSR 0x140

I registri speciali fanno parte di un banco interno di registri chiamati 'CSR' o CONTROL-STATUS REGISTERS

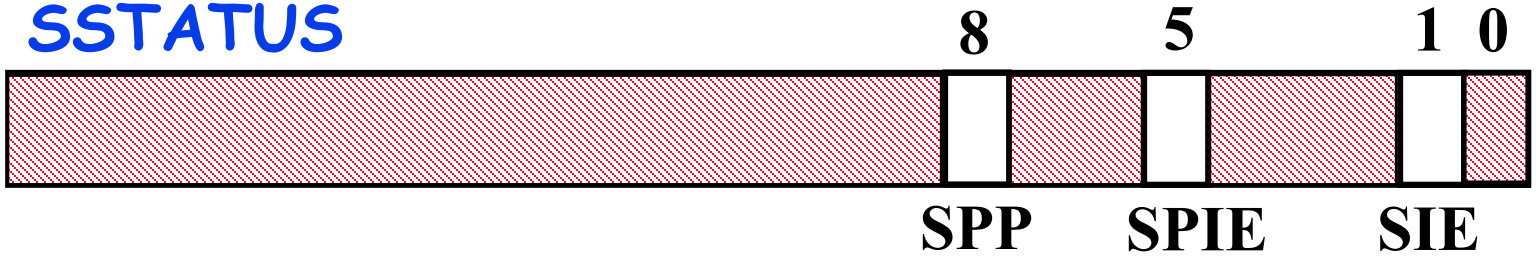
Tali registri vengono indirizzati con un numero su 12 bit (es. 0x105)

- Al verificarsi di un'eccezione, la parte di controllo della CPU modifica: SEPC, SSTATUS, SCAUSE, PC
  - Nota 1: in fase di fetch il PC (oldPC) è stato aggiornato a PC+4 (newPC). Per poter identificare l'istruzione "colpevole" è necessario far riferimento a oldPC anziché a newPC. Quindi SEPC= oldPC
  - Nota 2: in PC (newPC) viene (sovra-)scritto direttamente il nuovo valore (PC←STVEC)

Nota: nel simulatore RARS l'indirizzo della routine di gestione è modificabile

Status Register

Nota: il prefisso "S-" sta sempre ad indicare il modo "Supervisor"

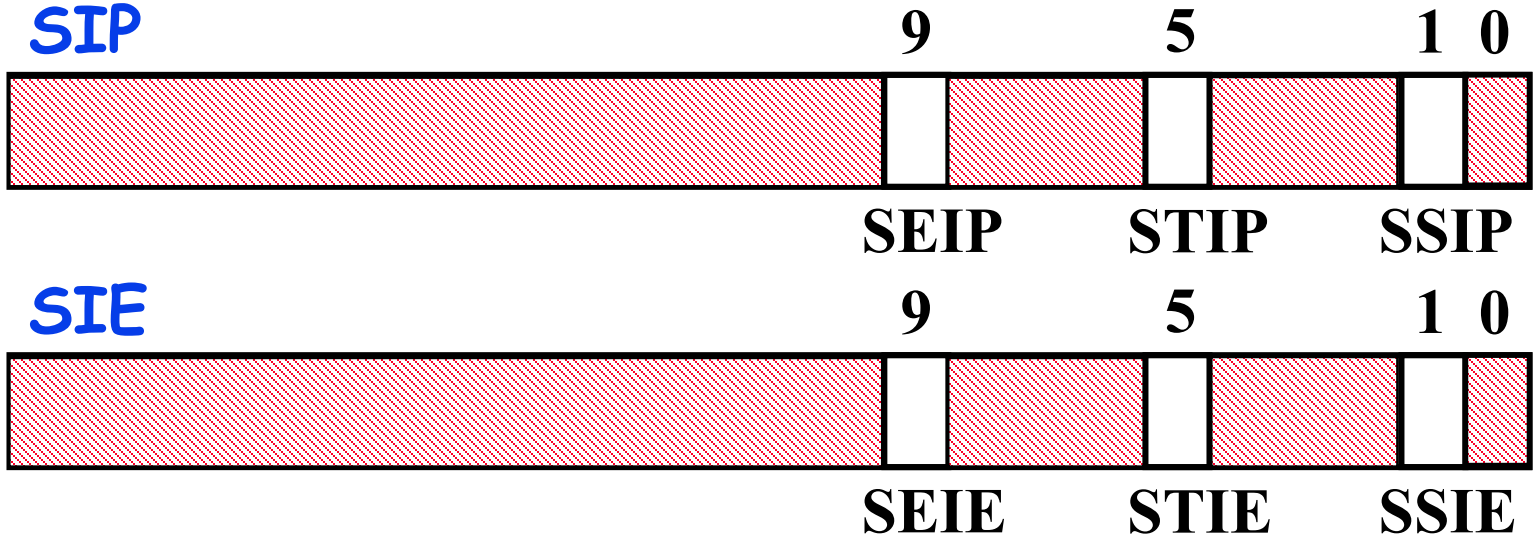


- Questi bit indicano il livello di privilegio PRECEDENTE (SPP) con il relativo valore del precedente interrupt-enable (SPIE) ed inoltre (SIE) permette l'abilitazione GLOBALE agli interrupt
- Il livello di privilegio può essere S- per Supervisor o U- per User
  - Questi sono rispettivamente codificati con i valori 1 e 0
  - Chi si trova ad un certo livello non può sapere se esistono livelli di privilegio superiori (tale informazione non è accessibile)
- SIE (S- Interrupt Enable) abilita globalmente gli interrupt a quel dato livello → per non «entrare in loop», viene subito disabilitato (0) al partire dell'eccezione
- SPIE (S- Previous Interrupt Enable) indica lo stato precedente del bit SIE al momento in cui si verifica l'eccezione
- SPP (S- Previous Privilege mode) indica il livello di privilegio precedente il momento in cui si verifica l'eccezione

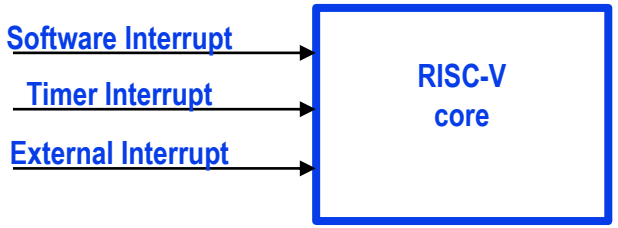
Nota: I livelli di privilegio del RISC-V sono in realtà più di questi – per un approfondimento vedere https://riscv.org/specifications/

Controllo più fine degli interrupt

Nota: il prefisso "S-" sta sempre ad indicare il modo "Supervisor"

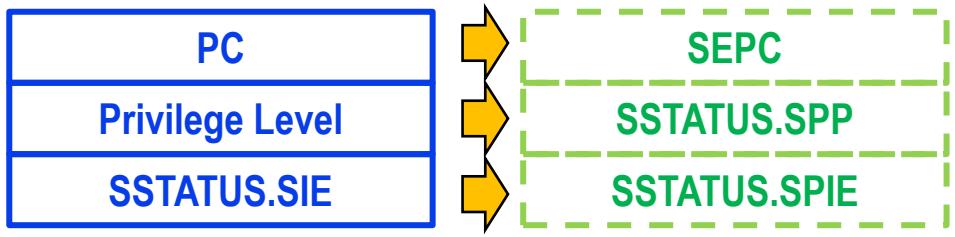


- SxIP sono bit di indicazione di interrupt pendenti (in attesa)
- SxIE sono bit di abilitazione più fine degli interrupt
  - x si riferisce alla possibile sorgente dell'eccezione
    - x=E → interrupt Esterno
    - x=T → interrupt dal Timer
    - X=S → interrupt Software (ovvero eccezione)

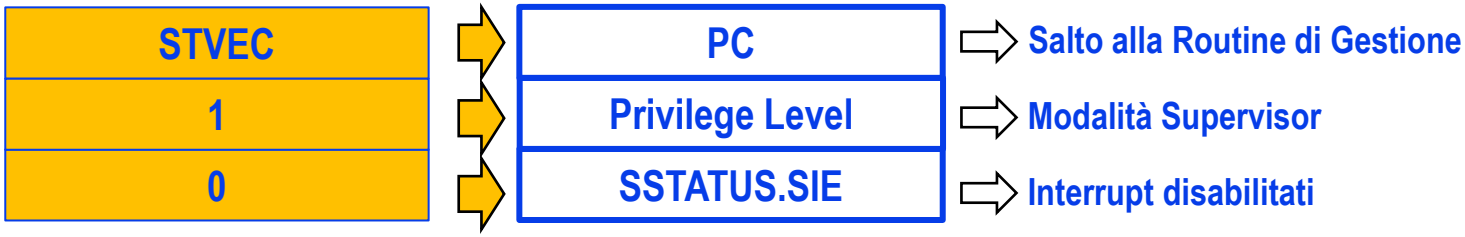


Ingresso in modalità Supervisor (e disabilitazione interrupt) e Uscita da modalità Supervisor (e ripristino)

- Al verificarsi di un'eccezione… ho le seguenti operazioni AD HARDWARE
  1) Salvataggio dello Stato della Macchina

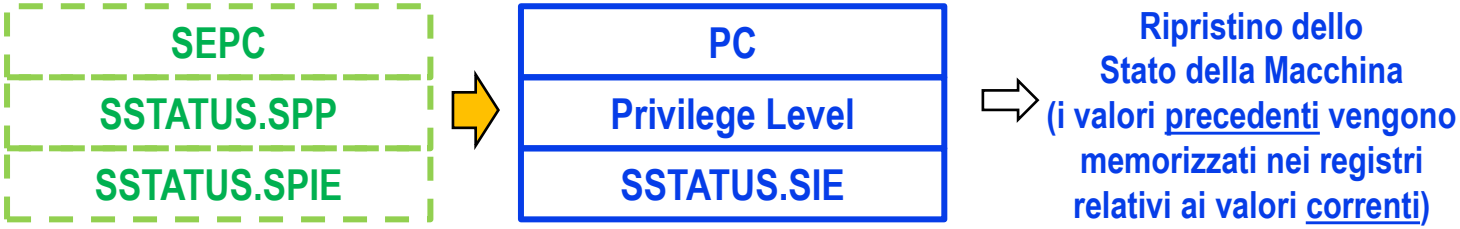


  2) Ingresso in modalità Supervisor (per poter eseguire la routine di gestione dell'eccezione)



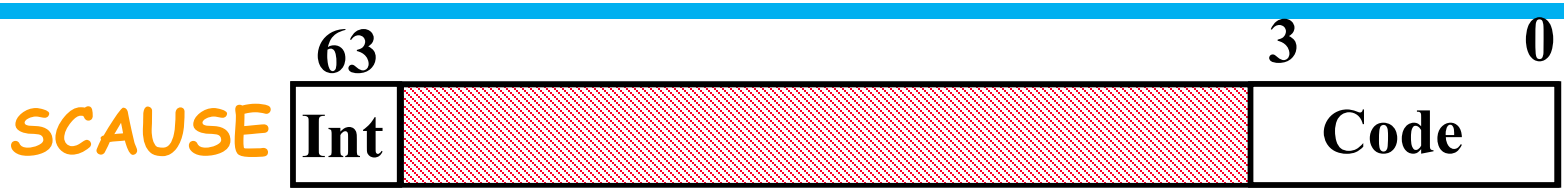
  3) Eseguo la routine di gestione dell'eccezione che DEVE terminare con l'istruzione SRET

- All'esecuzione della istruzione SRET… sempre AD HARDWARE



SCAUSE Register

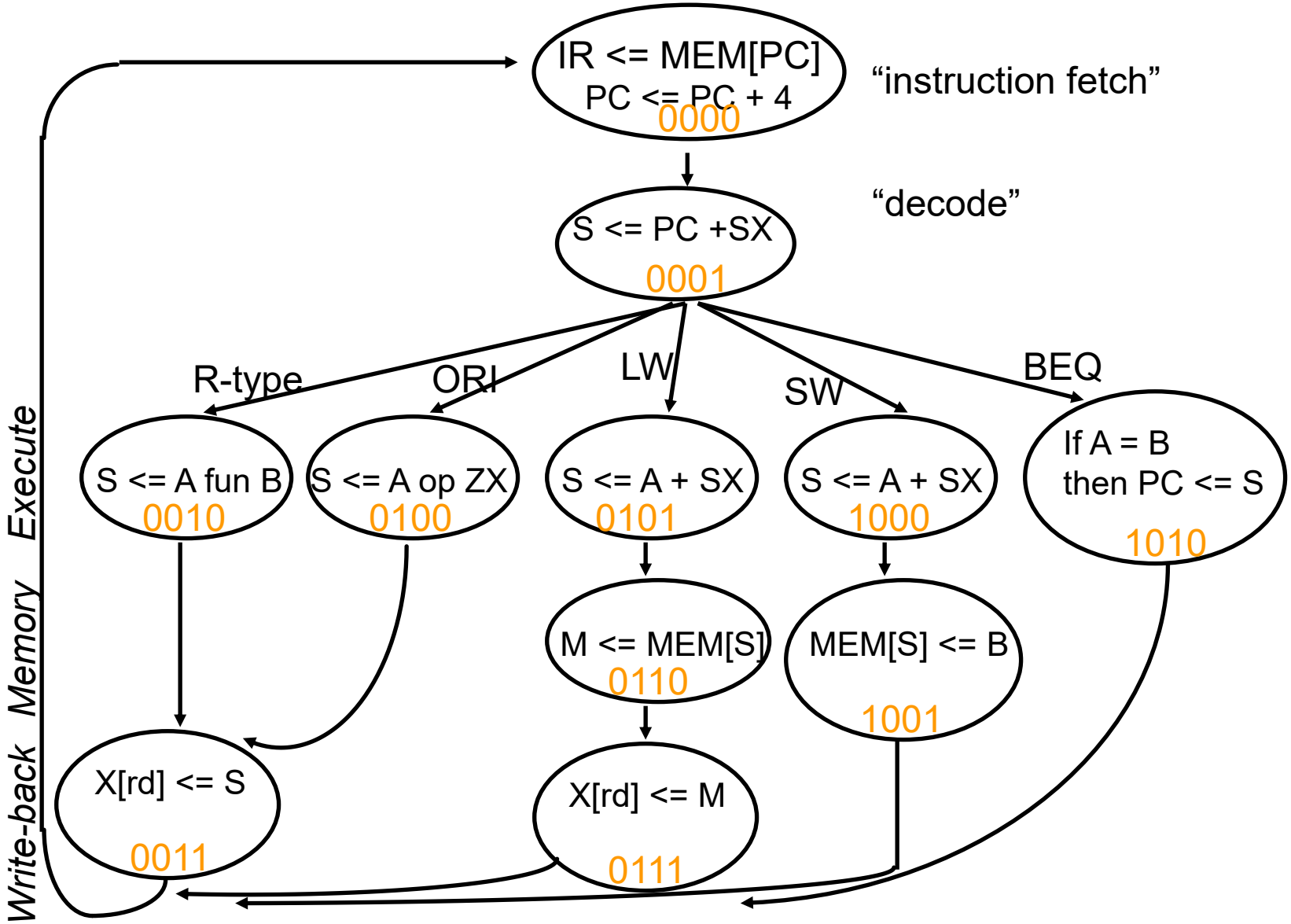
Nota: il prefisso "S-" sta sempre ad indicare il modo "Supervisor"



- Int (1 bit) se vale 1, la sorgente è un interrupt, se 0 la sorgente è un'eccezione
- Code (4 bits) codifica la ragione dell'eccezione
  - 0 → Instruction address misaligned
  - 2 → Illegal instruction
  - 3 → Breakpoint

  …altri esempi:
  - 4 → Load address misaligned
  - 5 → Load address fault
  - 6 → Store address misaligned
  - 7 → Store address fault
  - 8 → Environment call from U-mode
  - 9 → Environment call from S-mode
  - C → Instruction page fault
  - D → Load page fault
  - …

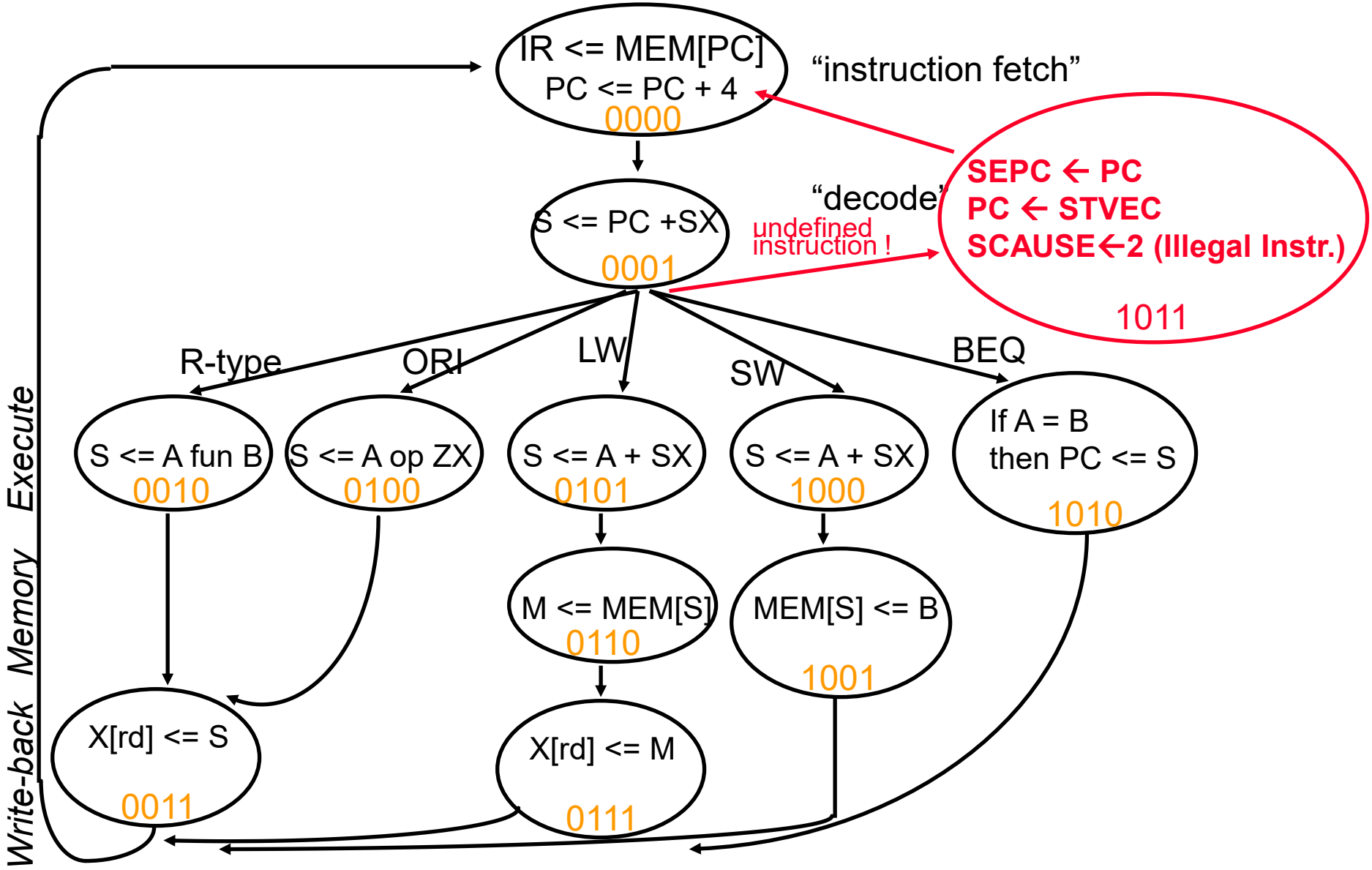
Macchina a Stati Finiti che descrive il nostro RISC-V



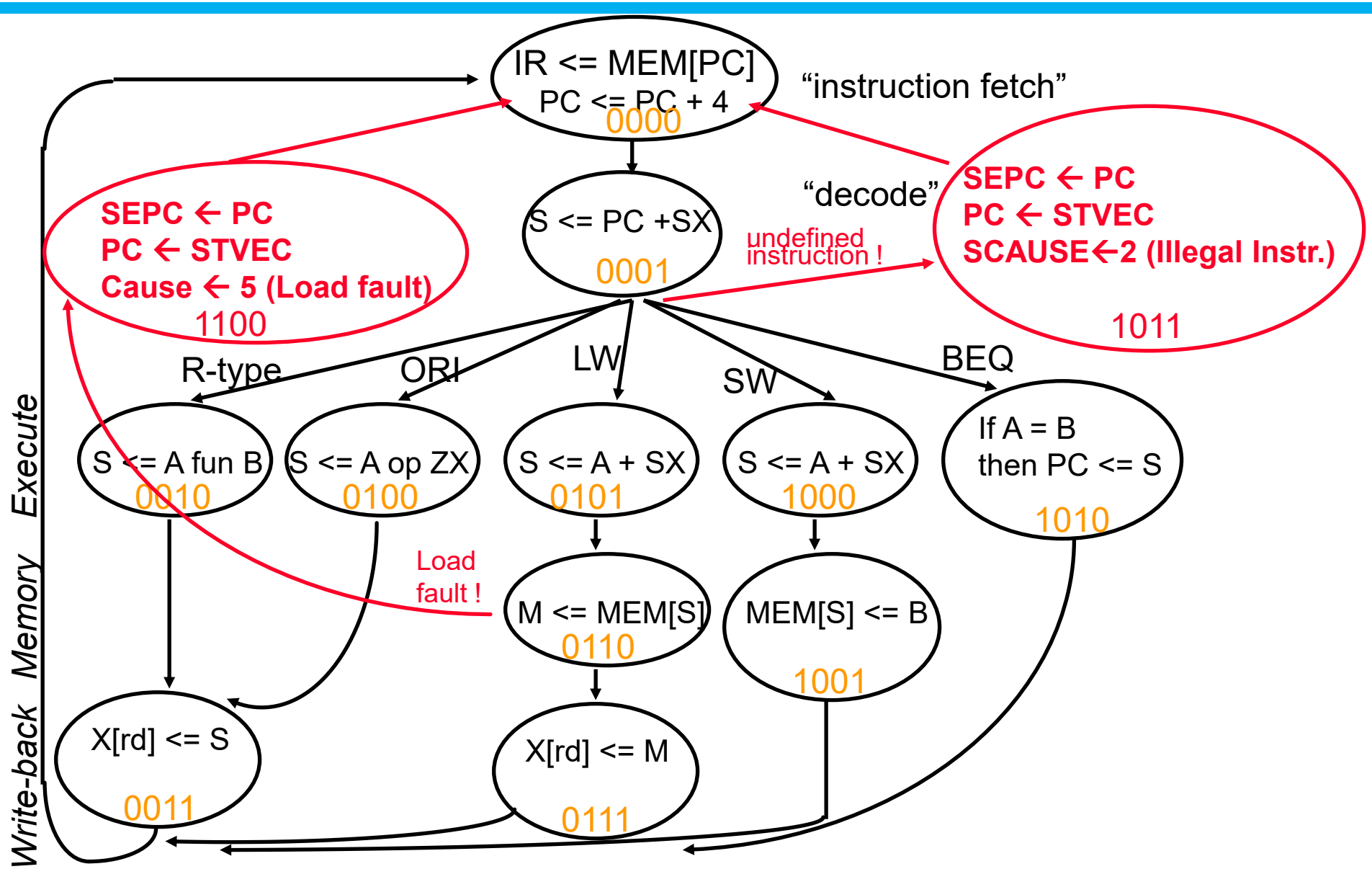
Modifica del RISC-V per poter gestire le eccezioni

- Supponiamo di partire dal processore "base" (che non gestisce eccezioni) e di voler gestire eccezioni: ad es. nel caso di «istruzione non definita»
- L'eccezione "Illegal Instruction" viene riconosciuta nel momento in cui, trovandomi nello stato 0001, non ho uno stato successivo in cui andare in quanto ho un op-code sconosciuto
  - Questo si può implementare definendo un nuovo stato (stato #11 o 1011) per tutti gli op-code diversi da lw, sw, (R-type), jmp, beq, e ori

Implementazione del meccanismo di eccezione per una istruzione non definita



Aggiunta di eccezione per «Load address fault»: stato#12

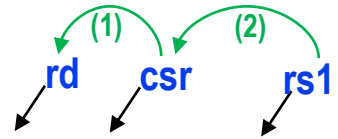


Lettura/Scrittura di SEPC, SCAUSE, STVEC, STVAL

- Per modificare i contenuti dei registri speciali CSR:
  - `CSRRW` CSR Read+Write
  - `CSRRS` CSR Read+Set
  - `CSRRC` CSR Read+Clear
  - `CSRRWI` CSRRW Immediate
  - `CSRRSI` CSRRW Immediate
  - `CSRRCI` CSRRW Immediate

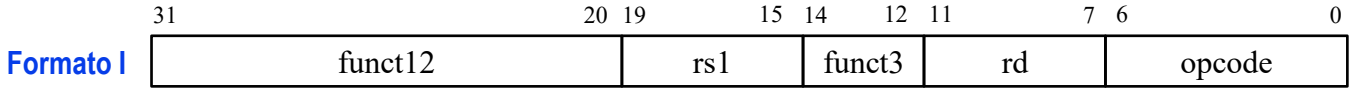
Nota: queste istruzioni sono Codificate con il formato I: i 12 bit del campo IMM12 sono usati per indicizzare un registro CSR (es. sstatus= 0x100); la variante immediata (es. CSRRWI) sfrutta quindi il campo RS1 per codificare un immediato a 5 bit.

- Esempi




```
CSRRW   t0,stvec,t1     # carica in t0 stvec e lo aggiorna con t1
CSRRSI  t0,sie,2        # t0←sie e mette a 1 il bit-1 di sie (software int.)
CSRRSI  t0,sie,512      # NON possibile (l'imm. deve stare su 5 bit)
CSSRC   x0,sie,t1       # (t1=2^9+2^5=544) azzera i bit 9 e 5 di sie
                          (disabilita interrupt esterni e interrupt timer)
CSRRS   t0,sstatus,x0   # carica in t0 il contenuto di sstatus, sstatus non
                          viene modificato (caso particolare, essendo rs1=x0)
CSSRW   x0,sscratch,a0  # salva a0 nel reg. speciale sscratch
```

Ricapitolazione nuove istruzioni e loro codifica



CSRRW/CSSRWI	csr	rs1	1/5	rd	0x73
CSRRS/CSRRSI	csr	rs1	2/6	rd	0x73
CSRRC/CSRRCI	csr	rs1	3/7	rd	0x73

Nota: "csr" è l'indirizzo del registro CSR su cui si opera (es. 0x142 per SCAUSE)

- Per completezza:

ECALL	0x000	0	0	0	0x73
SRET	0x102	0	0	0	0x73
EBREAK	0x001	0	0	0	0x73

- Ed inoltre, si può richiedere al processore di fermarsi per attendere un interrupt (anziché effettuare un ciclo di attesa attiva) con l'istruzione `WFI` (Wait For Interrupt)

WFI	0x105	0	0	0	0x73

Nota: vedremo come la WFI sarà utile nella gestione dell'Input/Output del Calcolatore

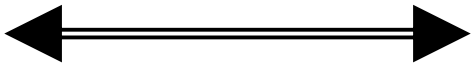
Esempio di Gestore delle Eccezioni

```
.text 0x80000080
CSRRW x0,sscratch,a0  # salva a0 nel reg. speciale sscratch
LA    a0,save         # salvo anche a1,ra -- ipotesi: gestore non rientrante
SD    a1,0(a0)        # necessario evitare uso stack (può essere la causa…)
SD    ra,8(a0)        # salvo ra perché dopo faccio una jal


CSRRW a1,scause,x0    # leggo la causa dell'eccezione
MV    a0,a1           # copio a1 in a0 (se il bit 63 è 0 → indica eccezione)
SRLI  a1,a1,63        # isolo il bit 63 (shift right logical)
BNE   a1,x0,fatto     # ignoro le interruzioni (ma non le eccezioni)

CSRRW a1,SEPC,x0      # carico il PC a cui si è verificata l'eccezione
JAL   gestione        # routine di gestione eccezione, a0=causa,a1=SEPC

fatto:
LA    a0,save         # ripristino i registri a0,a1,ra
LD    a1,0(a0)
LD    ra,8(a0)
CSRRW a0,sscratch,x0  # carico in a0 il valore salvato inizialmente in sscratch

SRET                  # PC←SEPC, ripristino modalità privilegiata,
                      # ripristino stato int.

.data 0x81000000
save: .space 16       # riservo una zona di 16 byte per salvare i registri
```

Nota: questo non è il gestore più efficiente possibile ma solo un esempio di come potrebbe essere strutturato il Gestore delle Eccezioni

Lezione 15: BUS

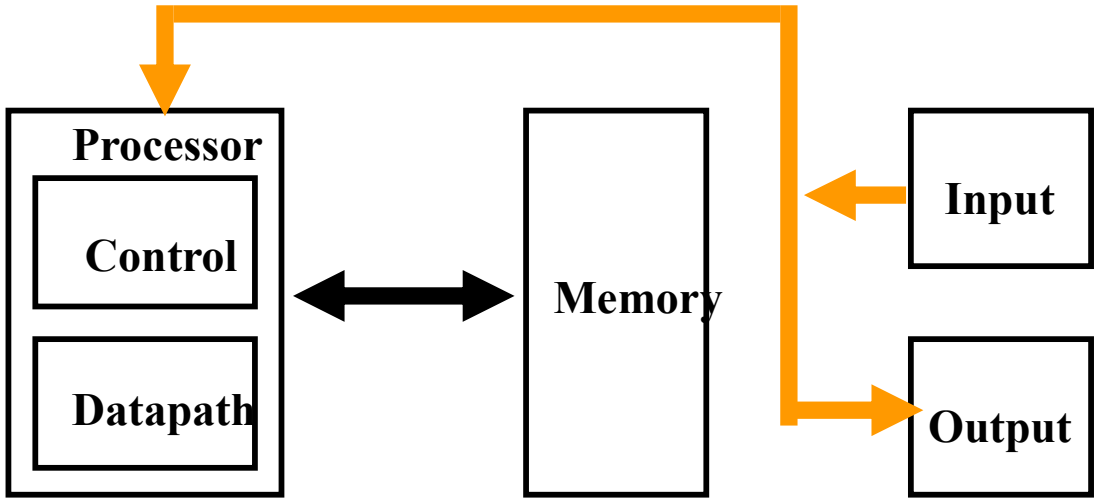
- BUS = veicolo (lento) nel quale molte persone viaggiano … ed inoltre…
- Un insieme di fili…



BUS

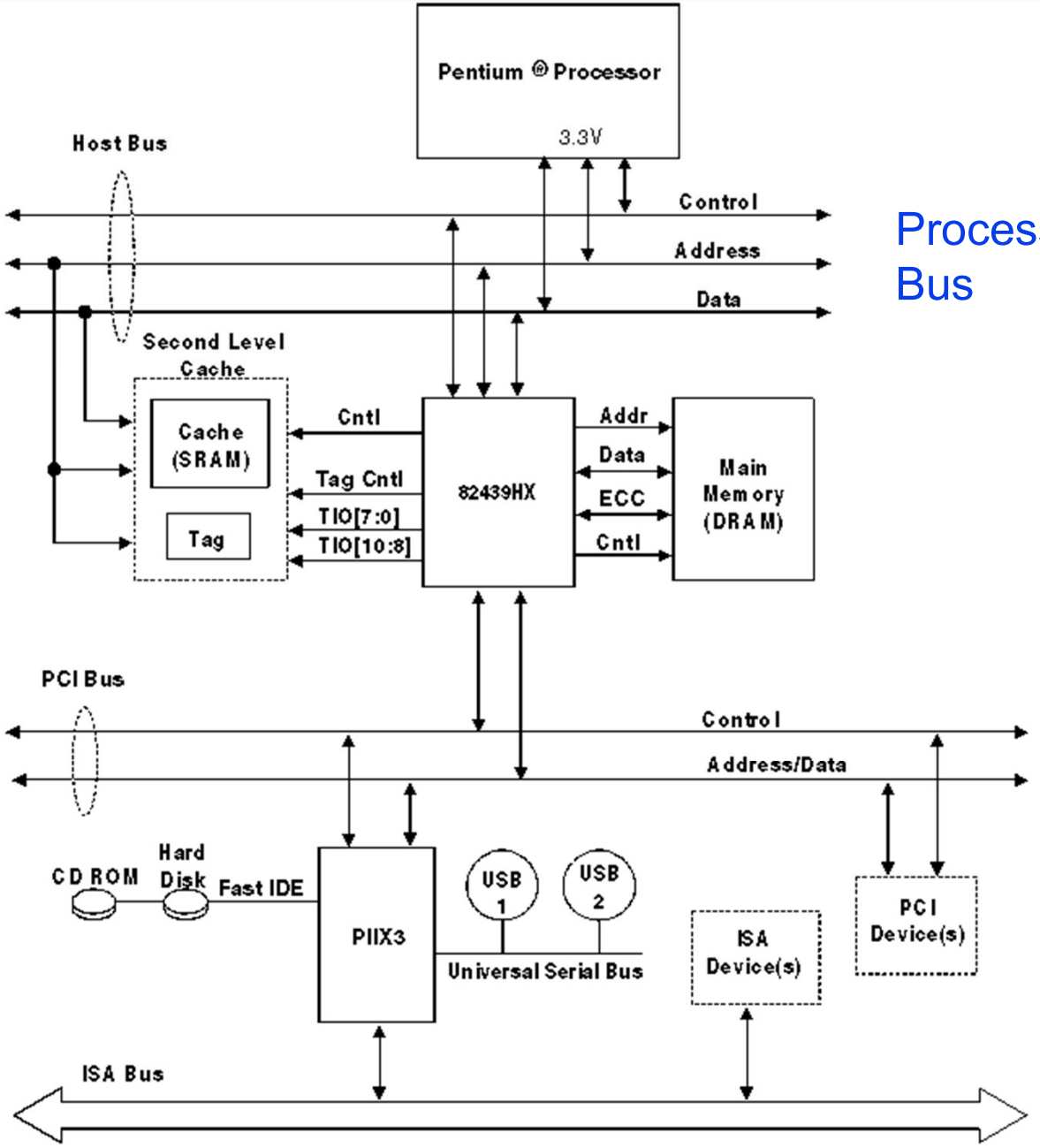
Un bus è:

- Un collegamento condiviso per comunicare
- Un insieme di fili usato per connettere vari sottosistemi

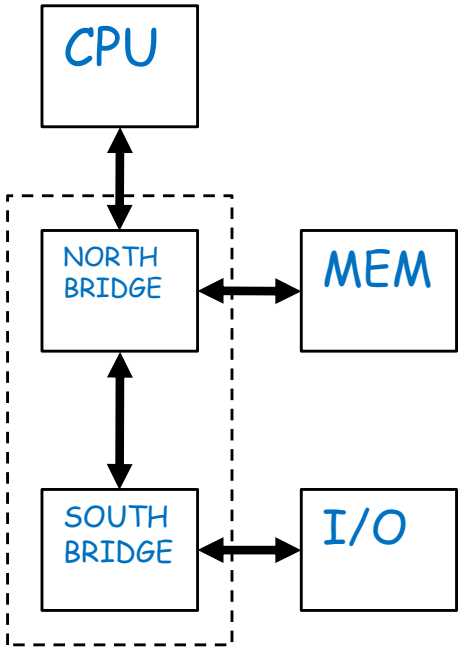


- Un bus è anche il mezzo fondamentale per costruire sistemi anche grandi e complessi
  - E' uno strumento di astrazione molto utilizzato

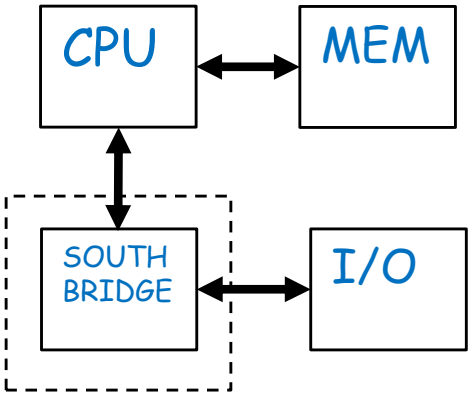
Esempio: Organizzazione di Sistema con CPU Pentium



Processor/Memory Bus

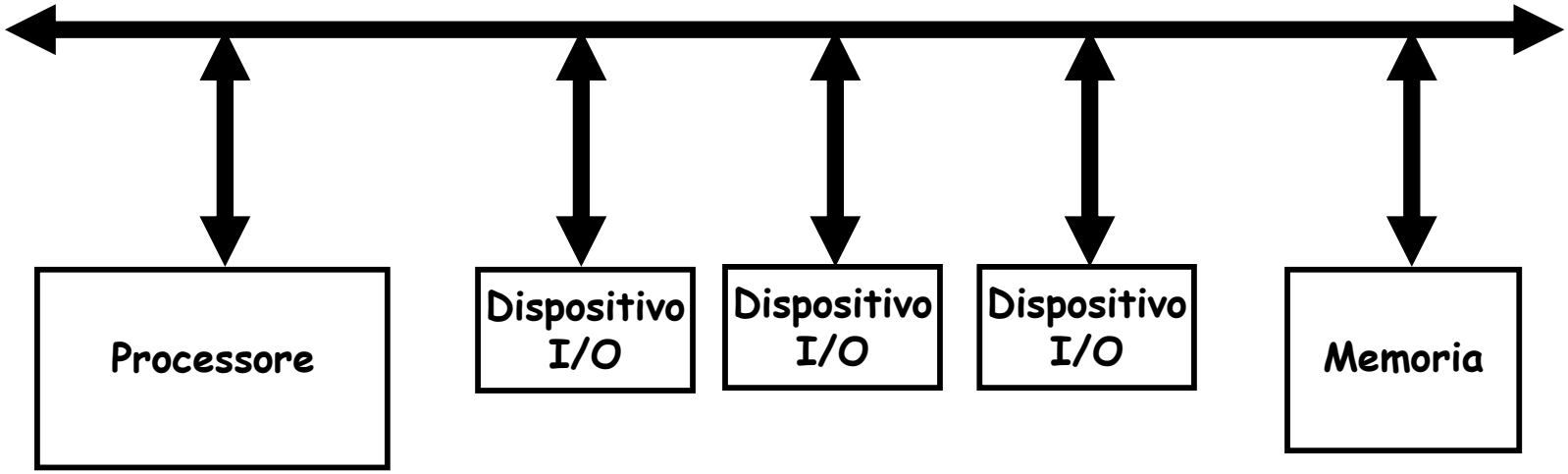


PCI Bus



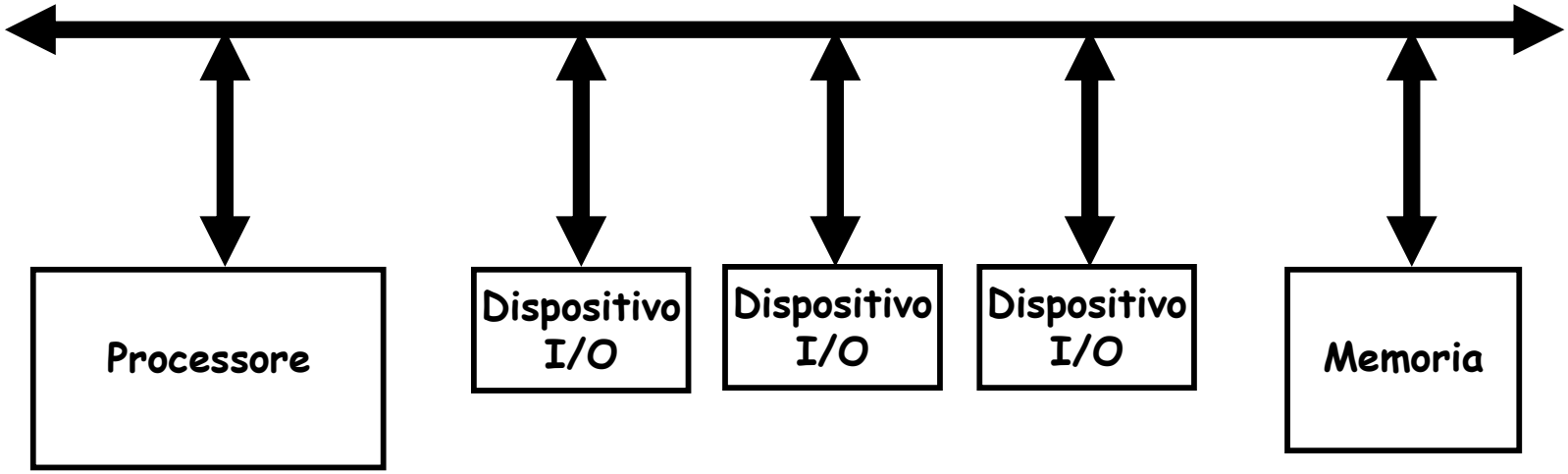
I/O Bus

Vantaggi dei bus



- **Versatilità**
  - Si possono aggiungere facilmente nuovi dispositivi
  - Una data periferica può essere usata su un calcolatore diverso che usa un bus che segue lo stesso standard
- **Basso costo**
  - Un unico insieme di fili può essere condiviso da dispositivi di vario tipo
- **Gestire la complessità** suddividendo il sistema

Svantaggi dei bus

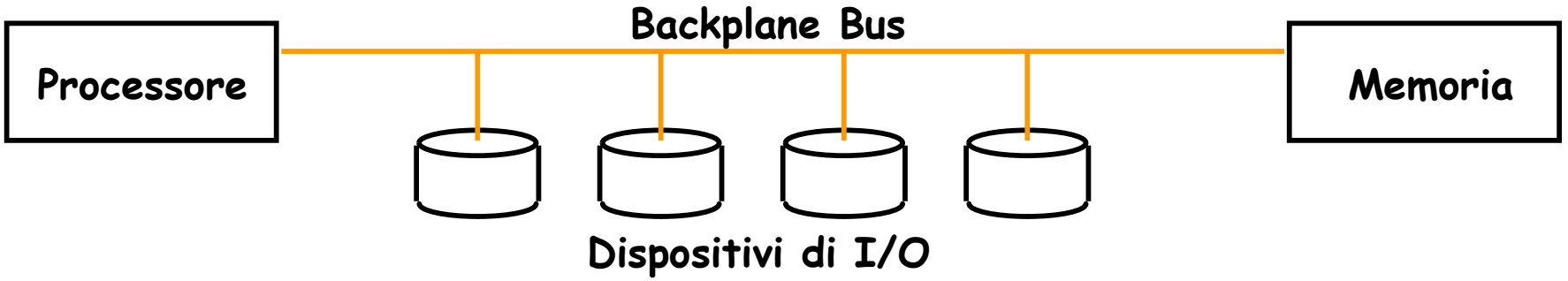


- Genera un "collo di bottiglia" nelle comunicazioni
  - La banda del bus limita il massimo throughput ottenibile per l'I/O
- La velocità massima del bus è fortemente limitata da
  - Lunghezza del bus
  - Numero di dispositivi collegati al bus
  - Necessità di supportare un ampio numero di dispositivi aventi
    - Tempi di latenza ampiamente variabili
    - Velocità di trasferimento dati ampiamente variabili

Tipi di Bus

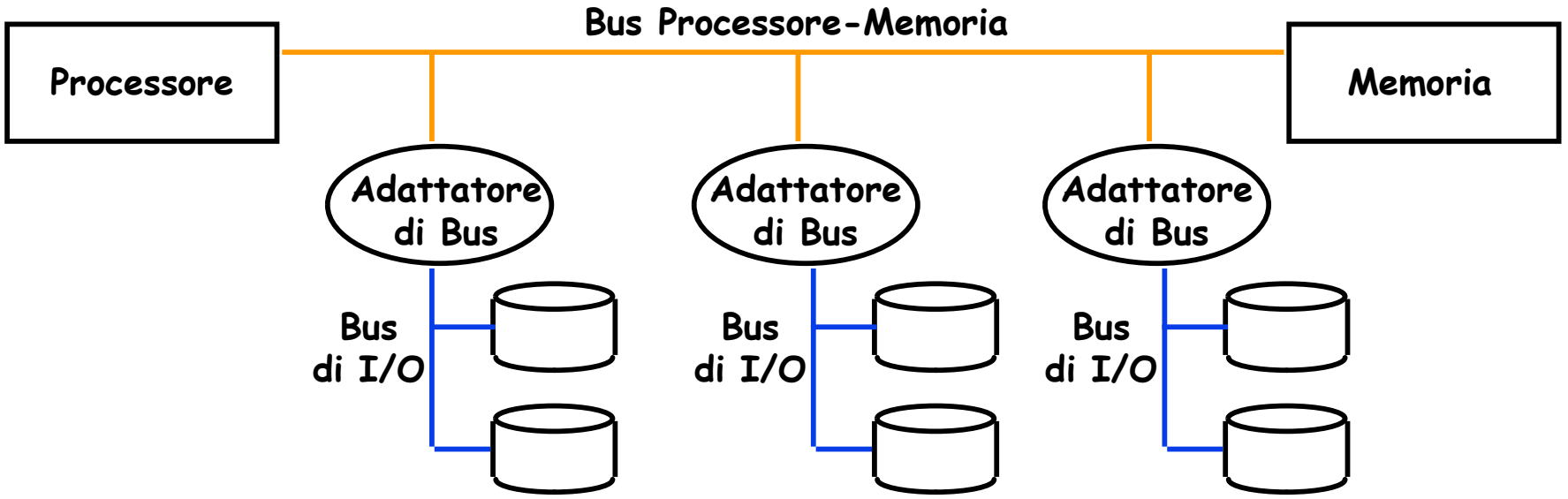
- Bus Processore-Memoria (specifico del sistema)
  - Corto e ad alta velocità
  - Deve solo essere adatto al sistema di memoria utilizzato
    - Tipicamente cerca di massimizzare la banda processore-memoria
  - Connette direttamente il processore
  - Ottimizzato per i trasferimenti di blocchi di cache
- Bus di I/O Bus (standard industriale)
  - Tipicamente abbastanza lungo e lento
  - Deve essere adatto a collegare dispositivi di I/O molto diversi fra loro
  - Si connette al bus processore-memoria o al backplane bus
- Backplane Bus (standard o proprietario)
  - Backplane: interconnessione che si svolge all'interno dello chassis
  - Consente al processore, alla memoria e ai dispositivi di I/O di coesistere e cooperare fra loro
  - Si può ottenere un risparmio se si usa un solo bus per tutti i componenti del sistema

Un Calcolatore con un solo Bus: Backplane Bus



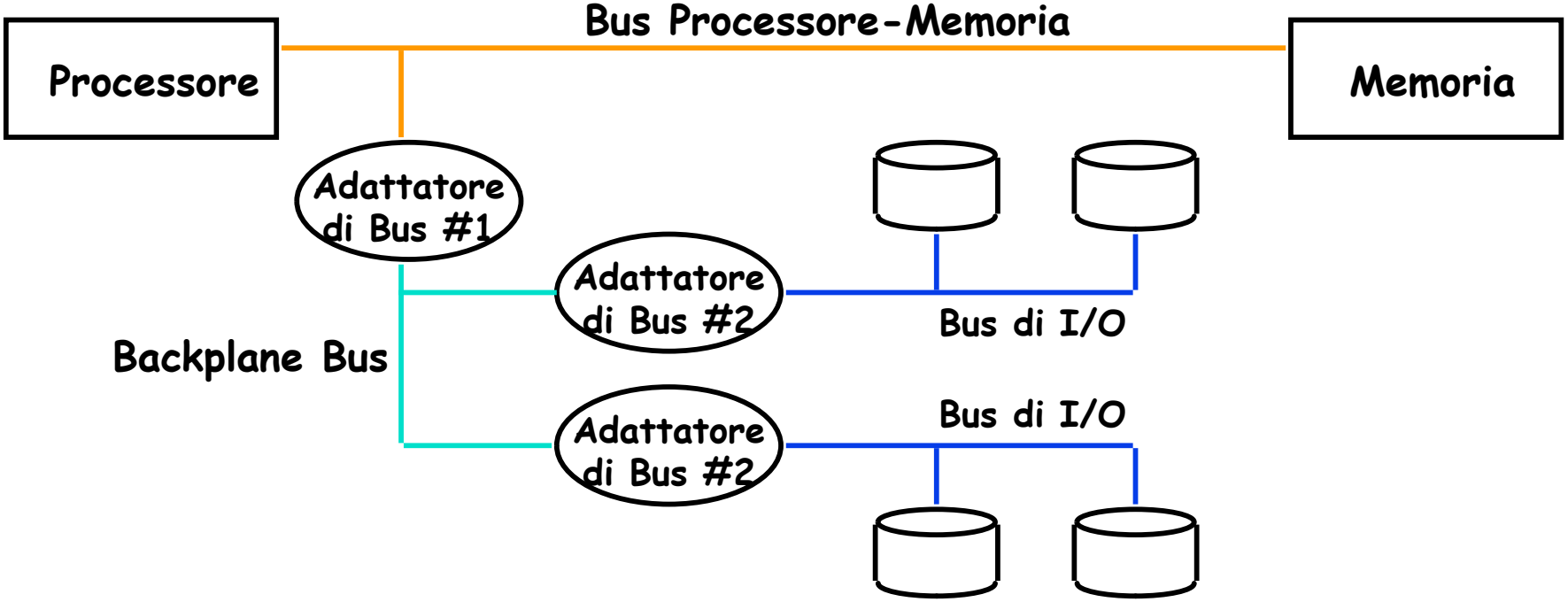
- E' basato su un singolo bus di sistema (backplane-bus)
  - Gestisce la comunicazione processore/memoria
  - Gestisce la comunicazione processore/dispositivi
  - Gestisce la comunicazione dispositivi/memoria
- Vantaggi: semplicità e a basso costo
- Svantaggi:
  - i) può facilmente andare in saturazione per troppo traffico diventando così un collo di bottiglia per l'intero sistema
  - ii) dovendo rispettare la velocità di ogni componente può essere lento
- Esempi: i primi PC della IBM usavano questo tipo di architettura

Sistema con due bus



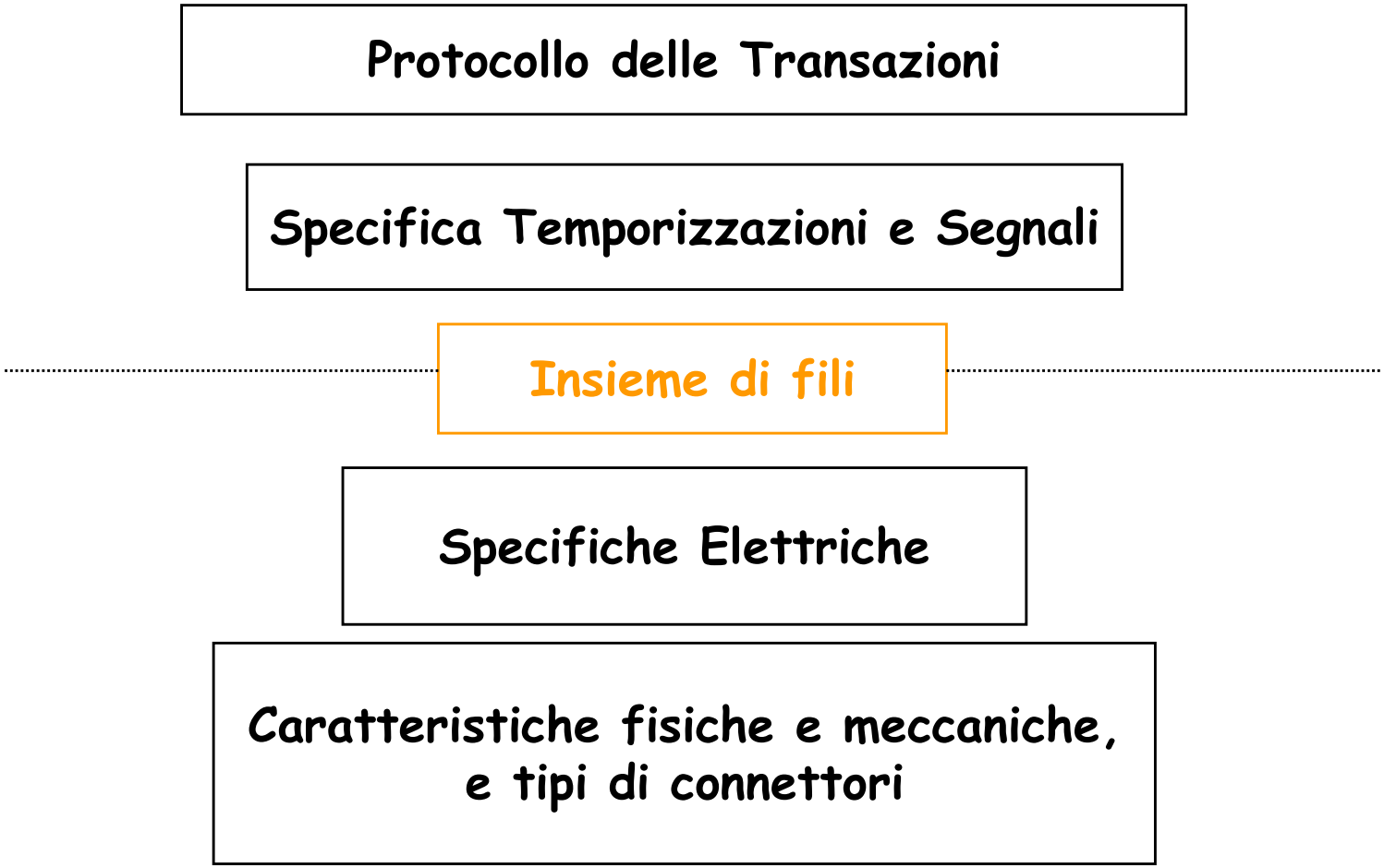
- In questa configurazione abbiamo due tipi di bus dedicati
  - La comunicazione fra i due tipi di bus avviene con dei componenti detti «adattatori di bus» o «bridge»
  - Il bus processore memoria gestisce traffico ad alta velocità
  - I bus di I/O gestiscono traffico specifico a seconda della tipologia di dispositivo e permettono l'espansione del sistema (es. PCI, SATA)
- Esempio commerciale/storico: il Macintosh-II della Apple
  - Come bus processore-memoria venne introdotto lo standard «NuBus»
  - Come bus generico di I/O venne utilizzato lo standard SCSI

Sistema con tre bus



- In questo caso il bus processore-memoria parla con un unico adattatore verso il backplane-bus restando così meno carico e più veloce
- Il backplane-bus serve per far parlare fra lor vari tipo di adattatori
- I bus di I/O sono disaccoppiate e dedicati come nel caso precedente (es. PCI, SATA, …)

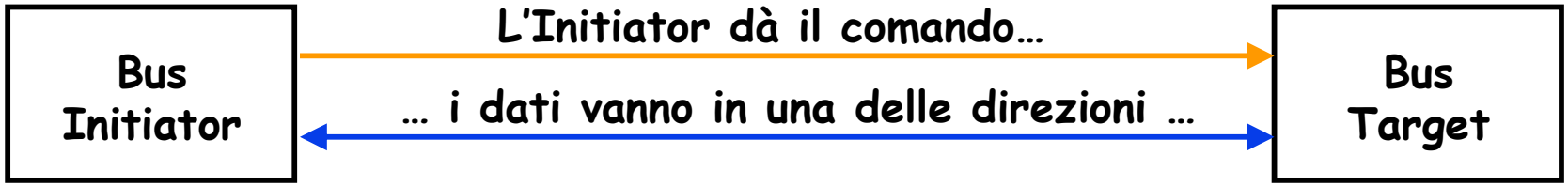
Come si definisce un bus?



Tipi di fili: le linee di controllo e le linee dati

- Linee di Controllo
  - Servono sia per indicare quali tipi di dato sono trasferiti…
  - …che per indicare il tipo di segnalazione (es. request/acknowledgment nel caso asincrono o clock nel caso sincrono)
  - … oppure la direzione (es. lettura oppure scrittura)
  - Tipicamente sono un numero basso di fili (es. 2-10)
- Linee Dati
  - Servono per trasportare le informazioni vere e proprie
    - Es. l'indirizzo a cui fare una scrittura/lettura
  - L'informazione può essere anche un valore (es. a 64 o 128 bit) da scrivere
  - L'informazione può essere anche l'identificatore della sorgente-dati (source) o del destinatario-dati (destination)
  - L'informazione può essere anche un comando complesso

Initiator/Target e Transazioni



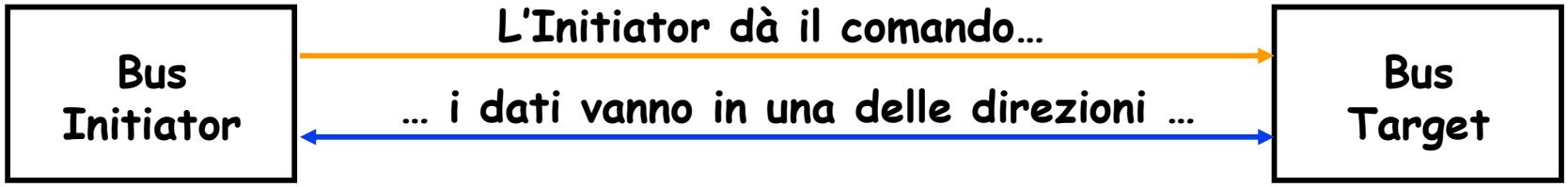
- Una transazione sul bus include tre fasi
  - Decidere chi è l'Initiator – fase di arbitraggio
  - Dare il comando (o l'indirizzo) – fase di richiesta
  - Trasferire i dati – fase di azione
- L'Initiator è l'unità che:
  - Ha il controllo del bus
  - Inizia la transazione dando un comando o specificando un'indirizzo
- Il Target è l'unità che:
  - Viene attivata dalla transazione ovvero risponde alla richiesta
  - Invia i dati all'Initiator, se l'Initiator richiede dati
  - Preleva i dati inviati dall'Initiator, se l'Initiator vuole mandare dati

Bus Sincroni e Bus Asincroni

- **Bus Sincrono**:
  - Fra le linee di controllo c'è il segnale di clock
  - Il protocollo di comunicazione è fisso ed è riferito al clock
  - Vantaggio: richiede pochissima logica e va molto veloce
  - Svantaggi:
    - Ogni dispositivo sul bus deve andare alla stessa frequenza di clock
    - Per evitare il "clock skew" per avere bus lunghi devo abbassare le velocità

- **Bus Asincrono**:
  - Sono sempre necessarie due linee di controllo (req, ack) per gestire il trasferimento (protocollo di handshaking)
  - Non ha bisogno del clock
  - Può collegarsi alla quasi totalità dei dispositivi
  - Può essere anche abbastanza lungo senza problemi di "clock skew"

Arbitraggio: ottenere accesso al Bus

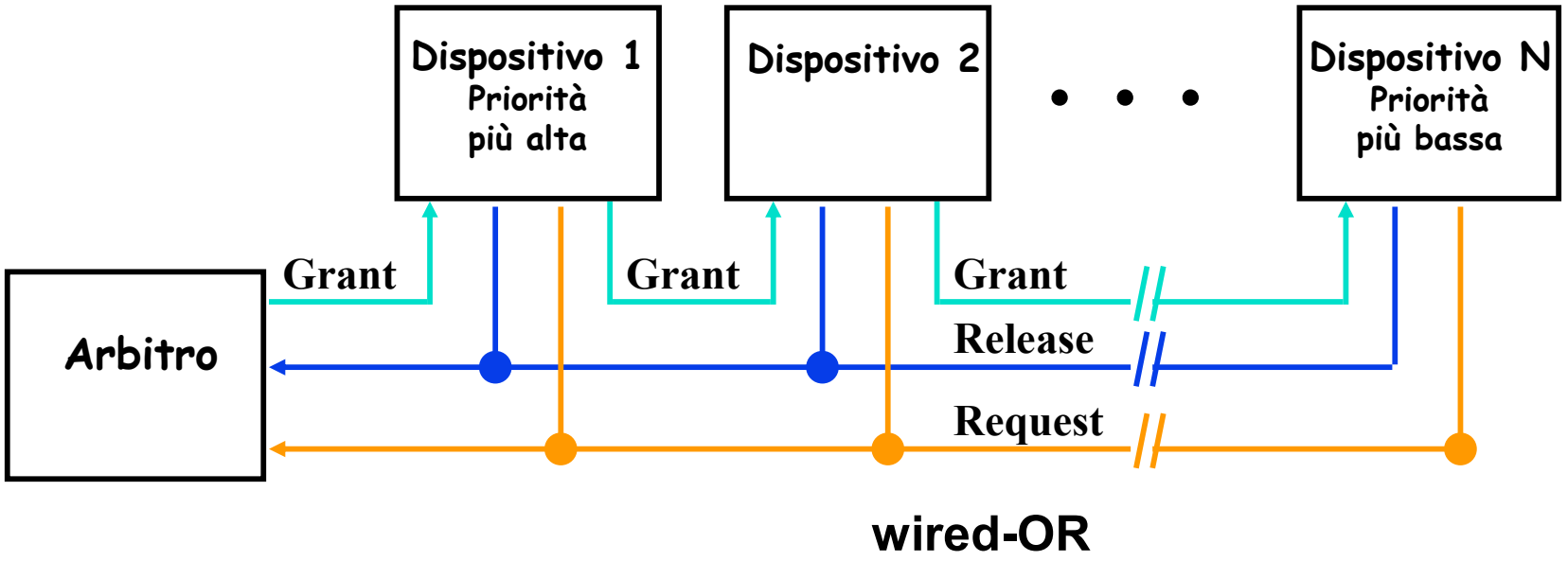


- E' uno dei problemi più importanti nella progettazione di un bus
  - Come viene prenotato il bus da un dispositivo che desidera usarlo?
- Innanzitutto per semplificare la situazione si deve arrivare alla definizione di chi è l'Initiator e chi è il Target
  - A quel punto solo il bus-Initiator potrà controllore l'accesso al bus
    - Tutte le successive richieste saranno iniziate e controllate da lui
  - Un Target risponderà alle richieste leggendo o scrivendo qualcosa
- Il sistema più semplice, che usa questo schema, potrebbe funzionare così:
  - Il processore è il solo bus-Initiator
  - Tutte le richieste saranno controllate dal processore
  - Principale svantaggio: il processore è coinvolto in ogni transazione

Bus-Initiator multipli: necessità di Arbitraggio

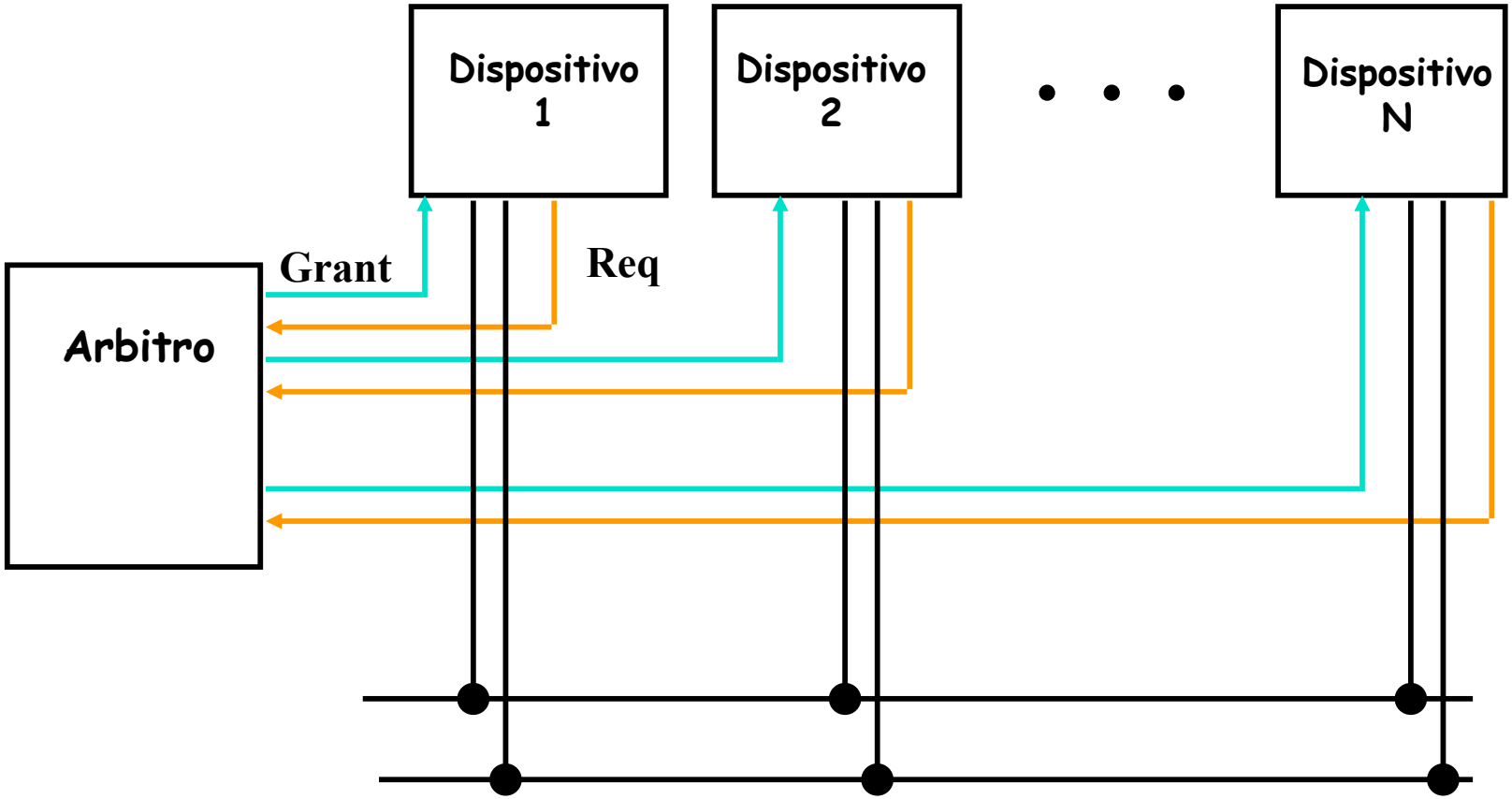
- Schema di arbitraggio del bus
  - Un bus-Initiator che voglia usare il bus segnala la richiesta ad un Arbitro
  - Il bus-Initiator non può usare subito il bus resta in attesa del permesso (grant)
  - Il bus-Initiator dovrà infine segnalare all'Arbitro che ha finito di usare il bus
- L'Arbitro cerca di bilanciare due fattori
  - Priorità della richiesta: il dispositivo a priorità maggiore sarà servito per primo
  - Fairness: anche se un dispositivo ha bassa priorità bisogna garantire che questo possa ottenere il controllo del bus entro un tempo ragionevole
- Categorie principali degli schemi di arbitraggio
  - Arbitraggio Centralizzato Daisy-Chain
  - Arbitraggio Centralizzato Parallelo
  - Arbitraggio Distribuito con Auto-Selezione
  - Arbitraggio Distribuito con Rilevamento delle Collisioni

Arbitraggio Centralizzato Daisy Chain



- Vantaggio: semplice
- Svantaggi:
  - Non è possibile garantire la fairness:
    - Un dispositivo a valle potrebbe rimanere bloccato indefinitamente
  - Il tempo di propagazione del segnale di grant può limitare la velocità del bus
- Es.: VMEbus

Arbitraggio Centralizzato Parallelo

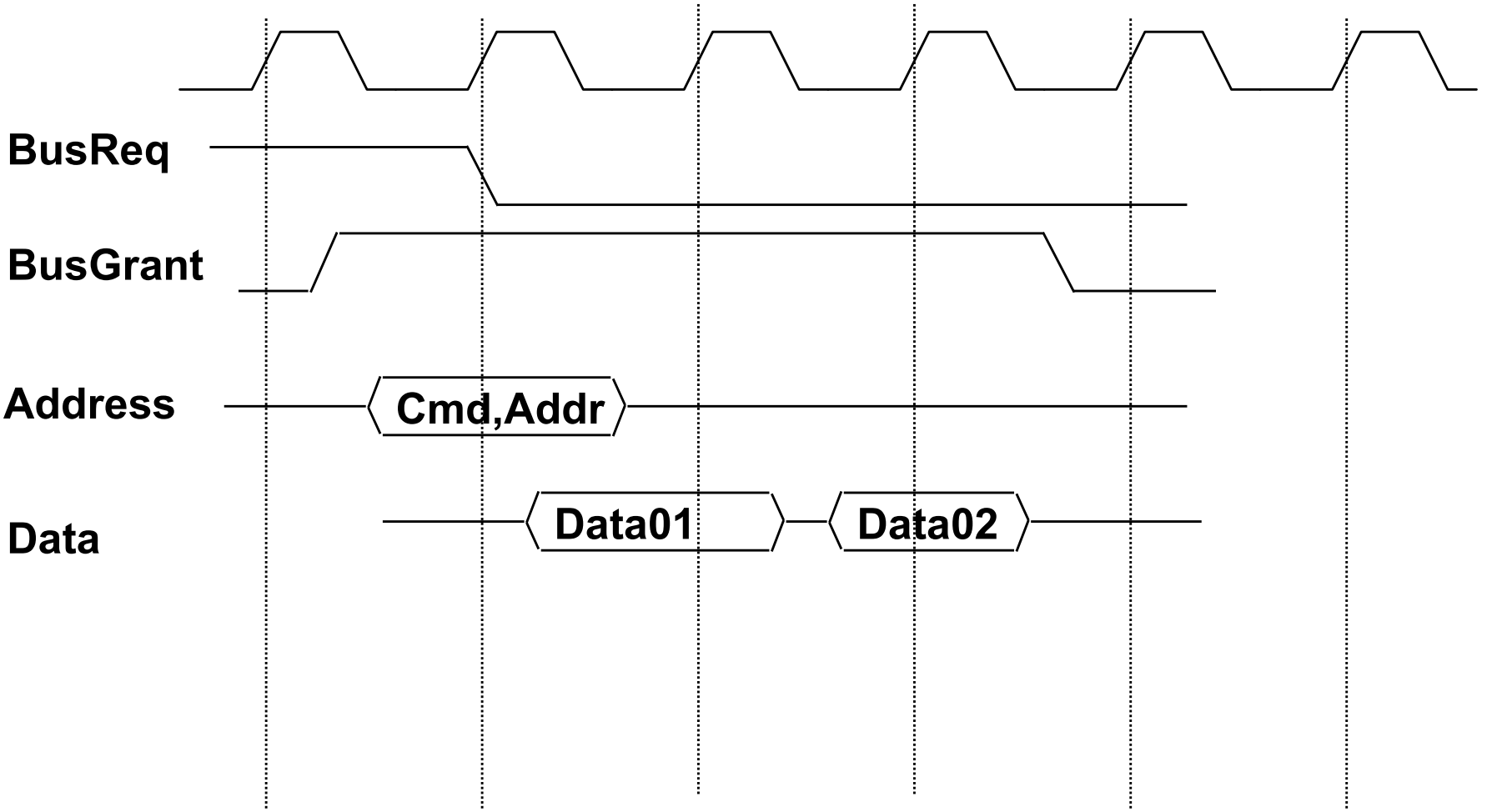


- Usato nella quasi totalità dei bus processore-memoria e nei bus di I/O ad alta velocità
- Inoltre è usato nei bus: Multibus I, EISA, PCI

Arbitraggio Distribuito

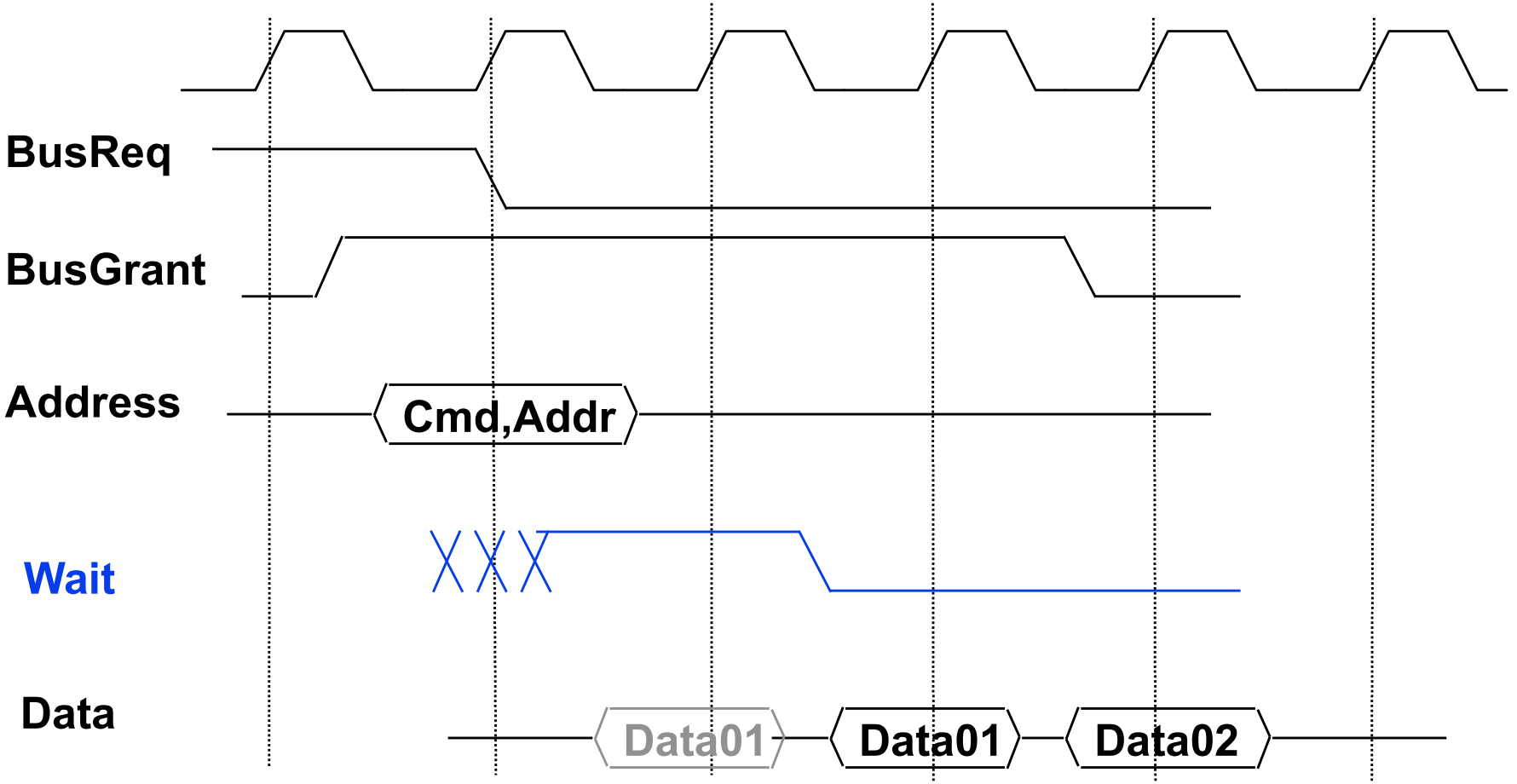
- La scelta dell'Initiator non viene fatta da una entità precisamente identificabile
- Esempi:
  - Multibus I: arbitraggio distribuito in daisy-chain
    - Inizialmente il bus è controllato da un'unità a priorità intermedia M
    - La richiesta da parte di un'unità A a priorità più alta attiva la logica per "convincere" M al rilascio del bus
  - Ethernet: arbitraggio distribuito con rilevamento delle collisioni
    - Inizialmente un'unità detiene il controllo del bus ma lo farà solo per un tempo limitato
    - Un potenziale Initiator prova a trasmettere: se il bus è libero bene, altrimenti riprova più tardi
    - Il tempo di attesa si calcola con una legge di decadimento casuale esponenziale, per evitare che si ripeta la stessa combinazione di unità che tentano l'accesso

Protocollo Sincrono (Caso Semplice)



- Anche un bus di memoria è più complesso di questo
  - La memoria (Target) può prendere più cicli per rispondere
  - Dovrà quindi segnalare che non è in grado di accettare indirizzi…

Protocollo Sincrono (Caso Tipico)



- Esempio: l'Initiator specifica un indirizzo (Address) da cui vuole leggere
- Con WAIT=1 il Target segnala che NON è pronto al trasferimento dati e che quindi i dati (es. Data01) verranno forniti successivamente
- Poi (con WAIT=0) il trasferimento ha luogo alla velocità del bus

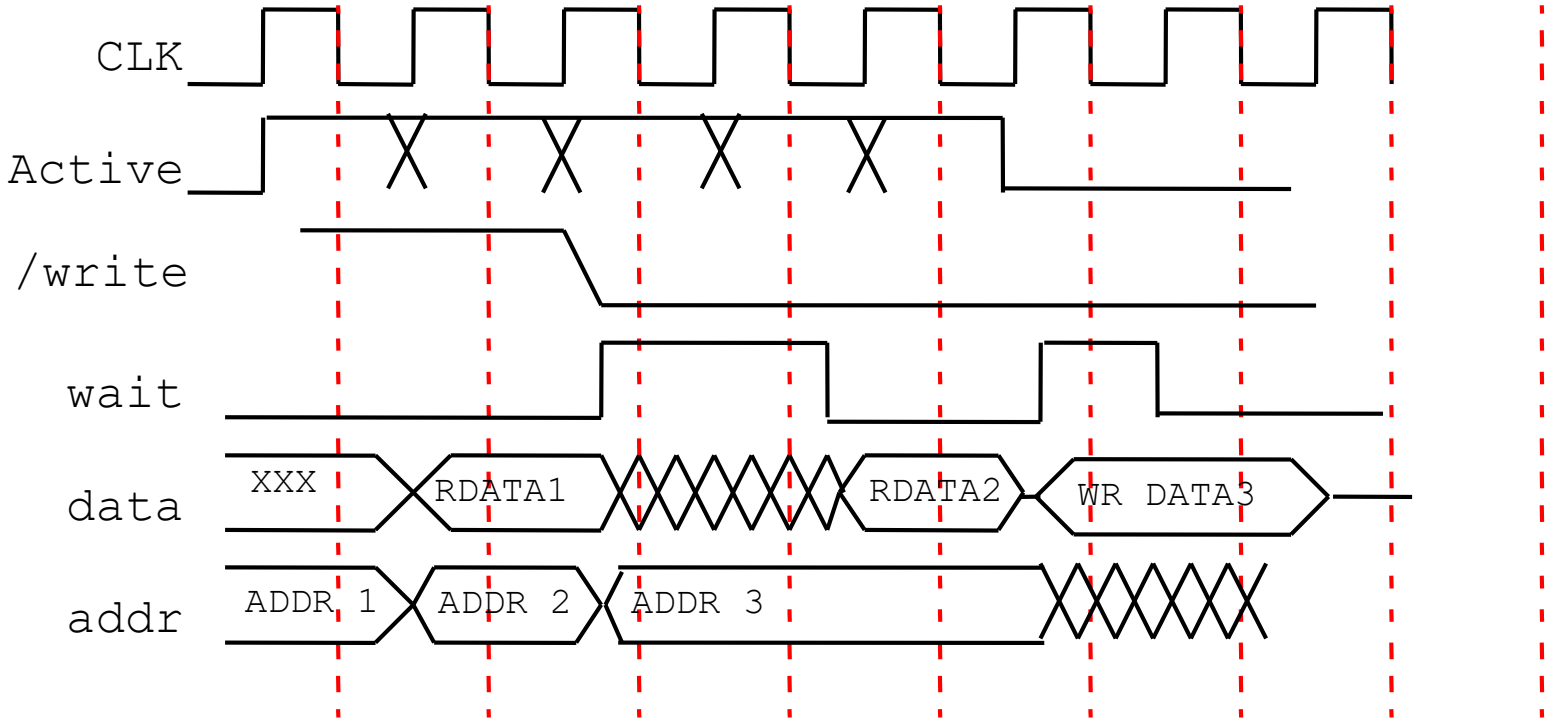
Come aumentare la banda del bus

- Uso di linee indirizzi/dati separate anziché multiplexate
  - posso trasmettere nello stesso ciclo indirizzo e dati
  - Svantaggi: (a) necessito di più fili, (b) aumenta la complessità
- Aumentare la larghezza del bus dati
  - Il trasferimento di più word richiederà meno cicli di bus
  - Esempio: nella SPARCstation 20 il bus di memoria è a 128 bit
  - Svantaggio: ancora necessito di più fili
- Trasferimento a blocchi (burst transfer)
  - Evito di ripetere l'indirizzo quando trasferisco K word successive
  - Specifico solo l'indirizzo della prima word
  - Il bus non viene rilasciato finché l'ultima word non è arrivata
  - Svantaggi: (a) maggiore complessità, (b) aumento del tempo di risposta nei casi di singola richiesta

Protocolli basati sul concetto di "Pipeline"

- Durante la fase di accesso ai dati inizio già a specificare l'indirizzo del trasferimento successivo

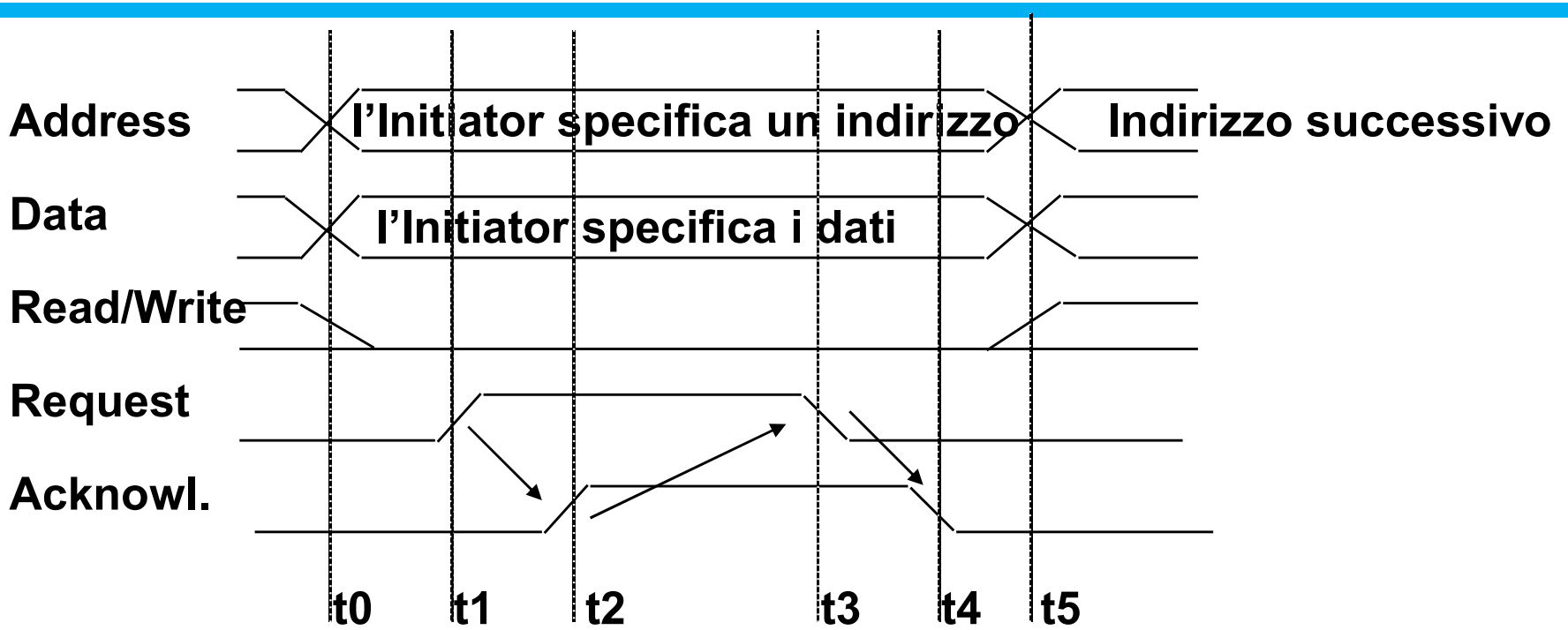
Esempio con singolo Initiator (processore-cache)



Aumento delle Transazioni su Bus MultiInitiator

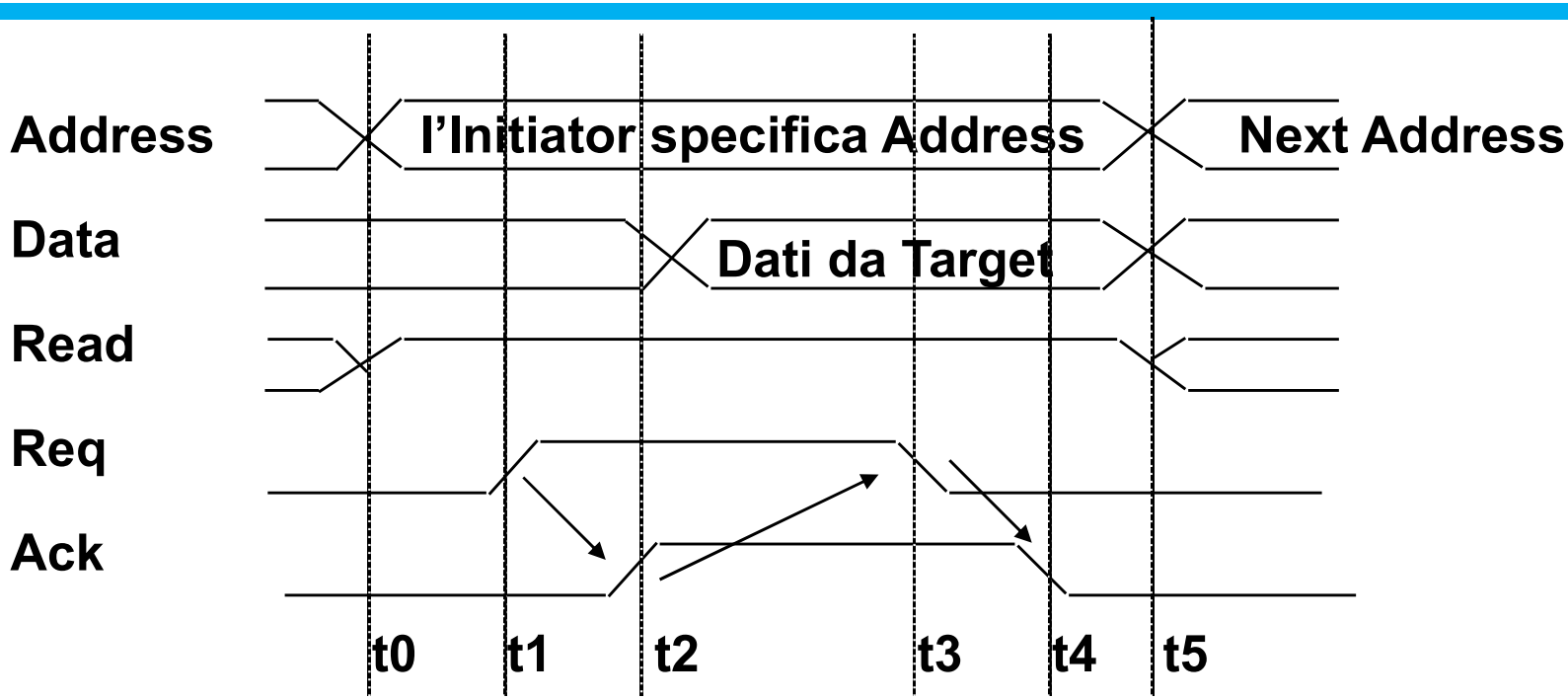
- Arbitraggio sovrapposto (overlapped)
  - Si guadagna tempo effettuando la fase di arbitraggio per la transazione successiva sovrapposta a quella in corso
- "Bus parking"
  - L'Initiator mantiene il bus ed effettua varie transazioni senza passare nuovamente alla fase di arbitraggio, fino a che qualche altro Initiator non si fa avanti
- Sovrapposizione delle fasi di indirizzamento e accesso
  - Visto nella pagina precedente
- "Split-phase" (o "packet switched") bus
  - Fasi di indirizzamento e accesso completamente separata
  - Ognuna necessita di un arbitraggio separato
  - Durante l'indirizzamento si produce anche un identificatore (tag) del pacchetto, che verrà associato ai dati in fase di risposta, per permettere l'abbinamento indirizzo-dato
- Nei bus moderni si usano tutte le tecniche precedenti combinate fra loro

Handshake Asincrono: Scrittura



- t0: L'Initiator ha ottenuto il controllo e specifica l'indirizzo, la direzione e dati; attende poi un certo intervallo di tempo affinché il Target capisca di essere coinvolto in quel trasferimento…
- t1: L'Initiator attiva il filo "Request"
- t2: Il Target attiva il filo "Ack", per indicare di aver ricevuto i dati → ha finito
- t3: L'Initiator rilascia "Req" → ha capito che il Target ha finito
- t4: Il Target rilascia "Ack" → ha capito che l'Initiator ha capito
- t5: la transazione è terminata e si può cambiare indirizzo/dati/r-w

Handshake Asincrono: Lettura



- t0: L'Initiator ha ottenuto il controllo e specifica l'indirizzo, la direzione e i dati; attende poi un certo intervallo di tempo affinché il Target capisca di essere coinvolto in quel trasferimento…
- t1: L'Initiator attiva il filo "Request"
- t2: Il Target attiva il filo "Ack", per dire che è pronto a trasmettere i dati
- t3: L'Initiator rilascia "Req" avendo ricevuto i dati → ha finito
- t4: Il Target rilascia "Ack" → ha capito che l'Initiator ha finito
- t5: la transazione è terminata e si può cambiare indirizzo/dati/r-w

Lezione 16
Tecniche di pilotaggio dei dispositivi di I/O

- Le periferiche si differenziano notevolmente fra loro perché
  - Trasferiscono differenti quantità di dati
  - Funzionano a velocità diverse
  - Utilizzano 'formati di dato' differenti

- Tipicamente funzionano a velocità più basse rispetto a CPU e Memoria

- Per interfacciarsi con il resto del sistema una periferica ha bisogno di un "controller" o "interfaccia"

Velocità di trasferimento tipiche per dispositivi di I/O

Dispositivo	Vel.Trasferimento (MB/s)
Tastiera	0.00001
Mouse	0.0001
Modem 56k	0.007
Canale telefonico	0.008
Doppia Linea ISDN	0.016
Stampante Laser	0.1
Scanner	0.4
Ethernet classica	1.25
USB (Universal Serial Bus)(2.5W)	1.5
Cinepresa Digitale	4
Disco IDE	5
CD-ROM 40x	6
Fast Ethernet	12.5
Bus ISA	16.7
Firewire (IEEE 1394) (15W)	50
Monitor XGA	60
Rete SONET OC-12	78
Disco SCSI Ultra2	80
Gigabit Ethernet	125
Conventional PCI 32-bit/33MHz	133
Nastro Ultrium	320
Bus PCI-X (1998)	533
10xGigabit Ethernet (2002)	1250
HyperTransport 1.0 (2001)	6400
100xGigabit Ethernet (2010)	12500
Bus PCI-E (2004)	16000
QuickPath (QPI)	25600
HyperTransport 3.1 (2008)	51200

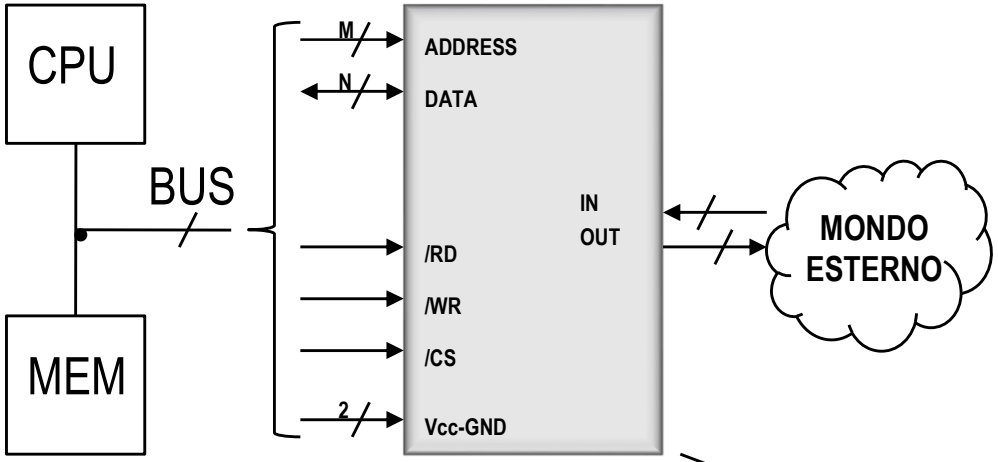
- **USB**
- v1 (low speed) 1.5Mb/s=187 KB/s
- v1.1 (full speed) 12Mb/s=1.5MB/s
- v2.0 (hi-speed,2001) 480Mb/s=60MB/s
- v3.0 (super speed,2009,4.5W) 5Gb/s=625MB/s
- v3.2gen1 (superspeed,2013) 10Gb/s=1.25GB/s
- V3.2gen3 (sperspeed,2019) 40Gb/s=5GB/s
- **SATA**
- 1.5Gb/s (2001) → 130MB/s(HDD),200MB/s(SSD)
- 3.0Gb/s (2001) → 200MB/s (reale)
- 6.0Gb/s (2008) → 400MB/s (reale)
- classica (1980) 10 MB/s = 1.25MB/s
- **ETHERNET**
- fast (1995) 100MB/s = 12.5MB/s
- Giga (1999) 1Gb/s = 125MB/s
- 10xGiga (2002) 10Gb/s = 1.25GB/s
- 100xGiga (2010) 100Gb/s = 12.5GB/s
- 400xGiga (2017) 400Gb/s = 50GB/s
- **PCI-EXPRESS (PCI-E o PCIe)**
- v1x (2004) 250MB/s(1lane),4GB/s(16-lane)
- v2.0 (2007) 500MB/s(1lane),8GB/s(16-lane)
- v3.0 (2010) 1GB/s(1lane),16GB/s(16-lane)
- v4.0 (2017) 2GB/s(1lane),32GB/s(16-lane)
- V5.0 (2019) 4GB/s(1lane),63GB/s(16-lane)
- V6.0 (2022) 8GB/s(1lane),121GB/s(16-lane)
- V7.0 (2025 – atteso) 16GB/s(1lane),242GB/s(16-lane)

Architettura del Sistema di I/O

- L'architettura del sistema di I/O è…
  l'interfaccia fra il dispositivo e il processore
- I dispositivi di I/O sono costituiti da
  - Componenti sensori/attuatori (e.g. tastiera, display, disco, …)
  - Componenti elettronici (controller del dispositivo)
  - Componenti software (il "device driver")
- Compiti del controller
  - Eventuale controllo di più dispositivi
  - Convertire il flusso seriale di bit in BLOCCHI di byte
  - Effettuare eventuali correzioni di errore
  - Rendere disponibili i dati alla memoria principale
    o prelevare i dati dalla memoria principale
- Compiti del device driver
  - Inizializzare il controller/dispositivo
  - Gestire le operazioni fra controller/dispositivo e processore

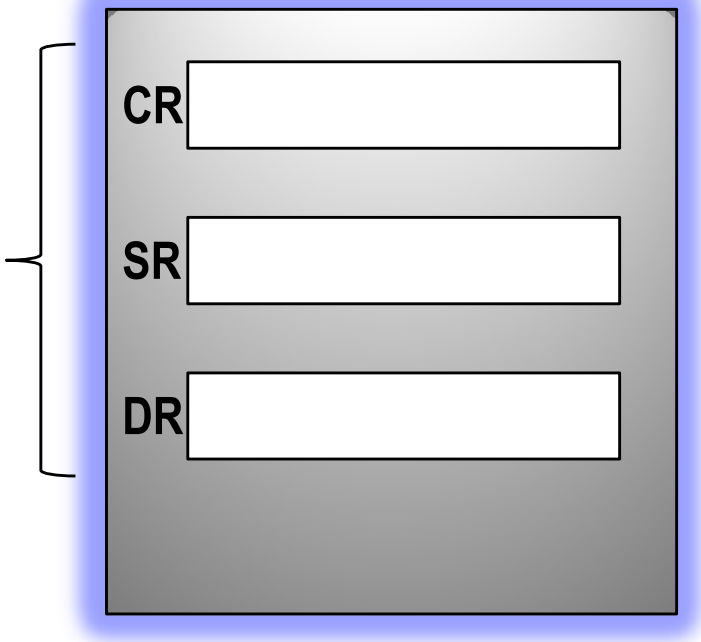
Nota: nel seguito si parlerà di "dispositivo" sottintendendo che ci si riferisce sempre al "controller del dispositivo"

Visione elettrica e logica del dispositivo



RD=READ
WR=WRITE
CS=CHIP SELECT

Il dispositivo viene visto dal processore come un insieme di locazioni dette "PORTE di I/O"



Porte di I/O tipicamente presenti in un dispositivo:

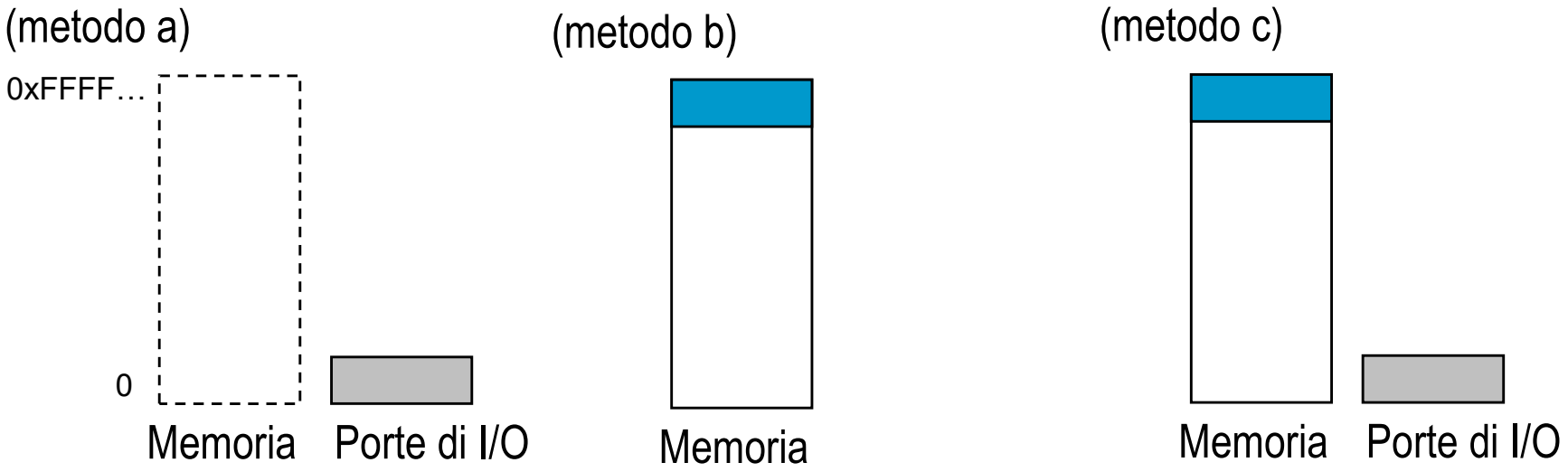
CR=CONTROL REGISTER
SR=STATUS REGISTER
DR=DATA REGISTER

Metodi di comunicazione fra processore e dispositivo

(a) Spazi separati di I/O e memoria

(b) Memory-mapped I/O

(c) Soluzione ibrida



Esempio (architettura x86 antiche):
- IN AL,[0x60] → legge un byte all'indirizzo dello spazio di I/O 0x60

Esempio (architettura RISC-V):
- LB t0,0(0x60) → legge un byte all'indirizzo dello spazio di I/O 0x60 **mappato in memoria**

Esempio (architettura x86 moderne):
- IN AL,0x60 → legge un byte all'indirizzo dello spazio di I/O 0x60
- MOV AL,[0x60] → legge un altro byte all'indirizzo di memoria 0x60 (es. da un altro dispositvo)

Nei casi a e b, i dispositivi si basano solo sulle porte o sulla memoria rispettivamente

Nel caso c, il processore può parlare con i dispositivi usando uno dei due metodi o entrambi

Istruzioni per la comunicazione fra processore e dispositivo

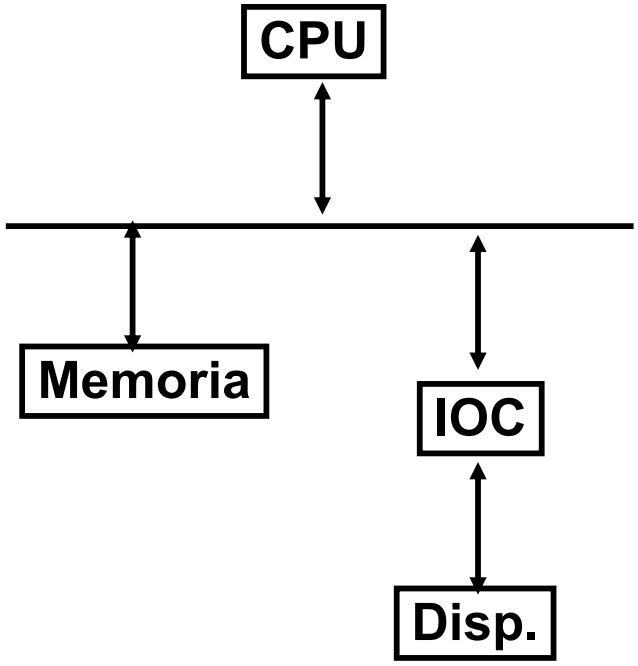
- Se si usano i metodi (a) o (c) per scambiare dati fra processore e memoria si rendono necessarie
  - Istruzioni speciali di I/O (es. Architetture x86)
  - L'istruzione speciale specifica il numero del dispositivo
    - Questo può essere trasmesso "come un indirizzo" sul bus di I/O
  - L'istruzione speciale specifica il comando da dare al dispositivo
    - Questo può essere trasmesso "come un dato" sul bus di I/O

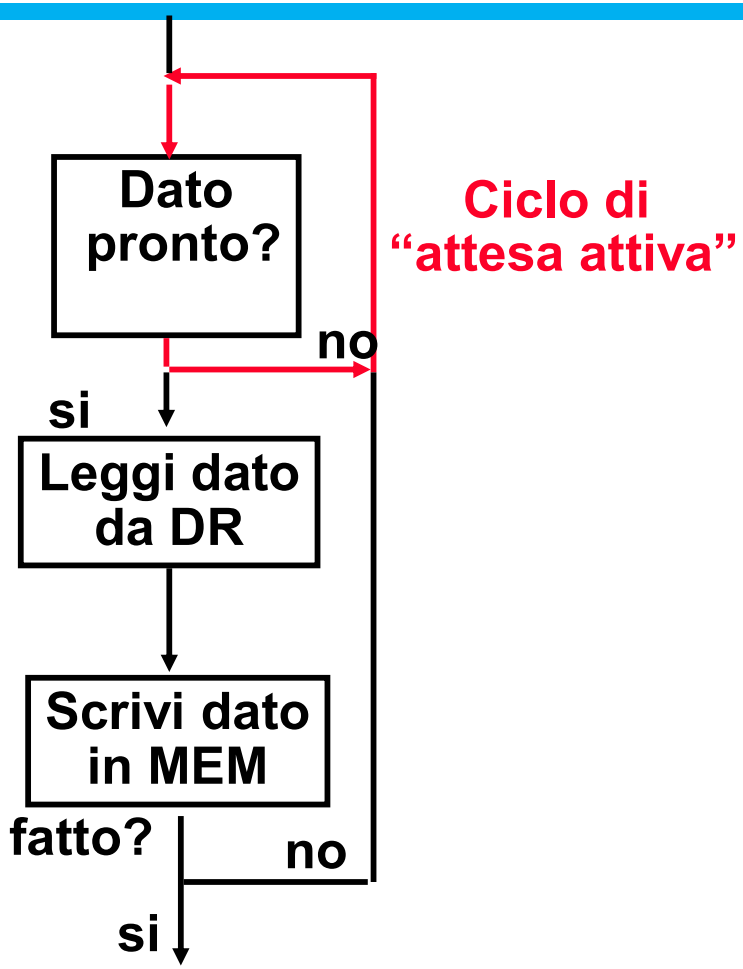
- Se si usa il metodo (b) non è necessario introdurre nuove istruzioni per comunicare con l'I/O
  - è sufficiente riservare certi indirizzi di memoria,
    non per mappare RAM ma per mappare indirizzi relativi al dispositivo
  - Letture e scritture a tali indirizzi
    sono interpretati dalla periferica come comandi
  - Si impedisce all'utente l'uso di questi indirizzi
    perché risiederanno in una zona riservata allo spazio di indirizzamento kernel
  - Questa tecnica è utilizzabile anche nel caso (c)

Protocolli di comunicazione fra processore e dispositivo

- Il processore ha bisogno di sapere quando
  - Il dispositivo ha completato un'operazione
  - Il dispositivo ha incontrato un errore
- Esistono tre protocolli fondamentali
  - Polling:
    - Il dispositivo mette il dato in un "data register" e segnala questo fatto tramite uno "status register"
    - Il processore controlla periodicamente lo "status register"
  - Interrupt:
    - Ogniqualvolta il dispositivo ha bisogno di attenzione dal processore, interrompe il processore tramite linea dedicata
  - DMA/IOP:
    - Il processore inizia l'operazione
    - Un processore ausiliario si occupa del trasferimento del dato
    - Al processore è notificato che l'operazione è finita con un interrupt dal DMA/IOP verso il processore

→ Protocollo di POLLING (o di I/O Programmato)



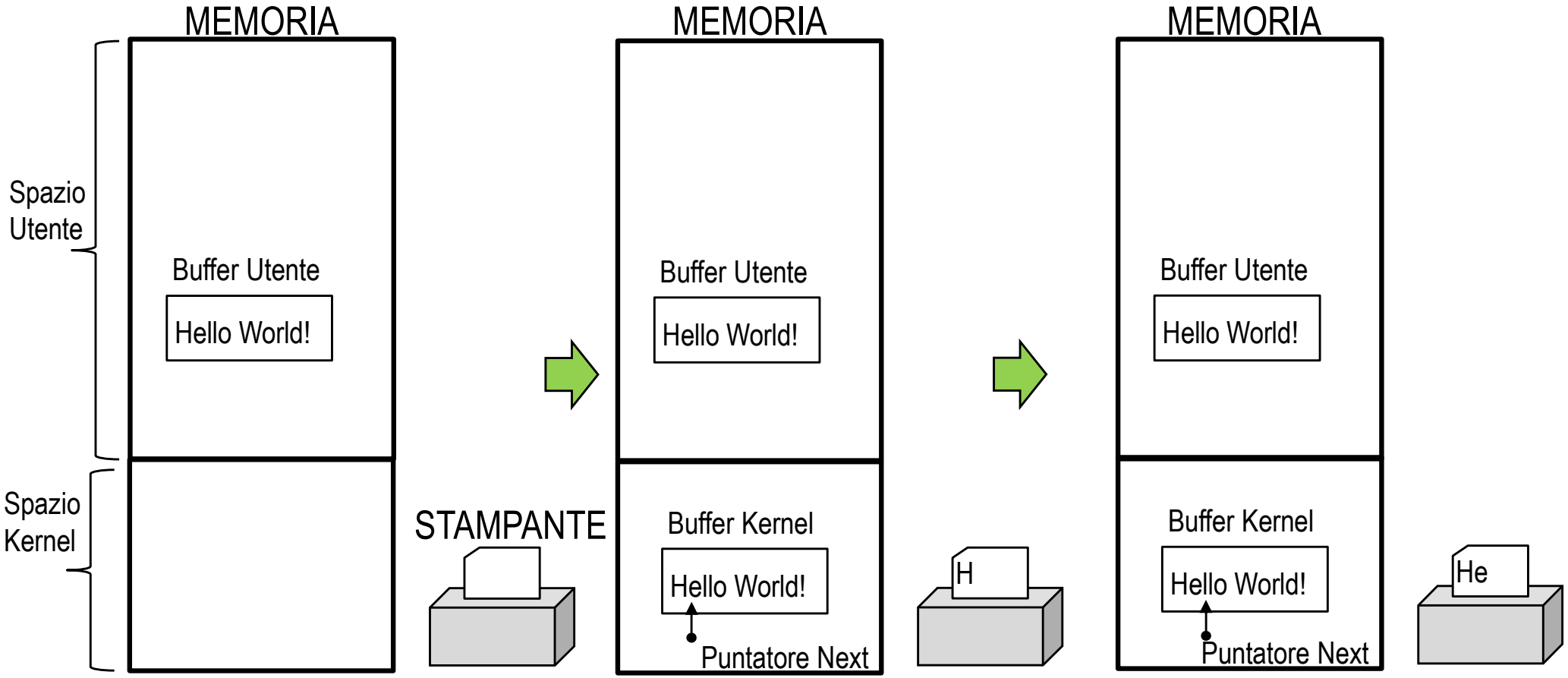


- Vantaggio
  - Semplice: il processore ha il controllo totale e fa tutto il lavoro
- Svantaggio
  - La gestione del polling può consumare molto tempo del processore
    - Trucco: distribuire i controlli di I/O fra codice che effettua altre operazioni

Polling - dettaglio

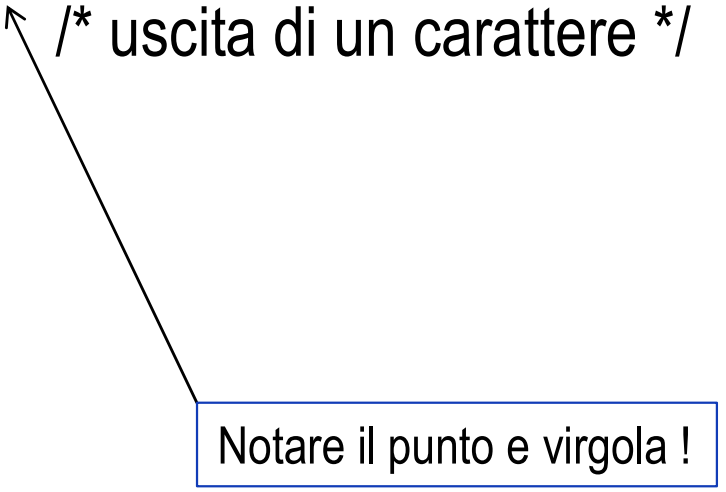
- La CPU richiede un'operazione di I/O
- Il controller di I/O effettua l'operazione di I/O
- Il controller di I/O attiva un bit nello "status register"
  - Il controller di I/O non informa direttamente la CPU
  - Il controller di I/O non interrompe la CPU
- La CPU guarda periodicamente il bit nello "status register"
  - La CPU può attendere oppure ritornare a vedere più tardi

Esempio di Polling: stampa di una stringa



Stampa di una stringa con Polling (Driver)

```
copy_from_user(buffer, p, count);          /* p è un buffer nel kernel */
for (k=0; k<count; ++k) {                  /* ripeti per ogni carattere */
    while (*printer_status_reg != READY); /* attendi il bit di READY */
    *printer_data_register = p[k];         /* uscita di un carattere */
}
return_to_user();
```

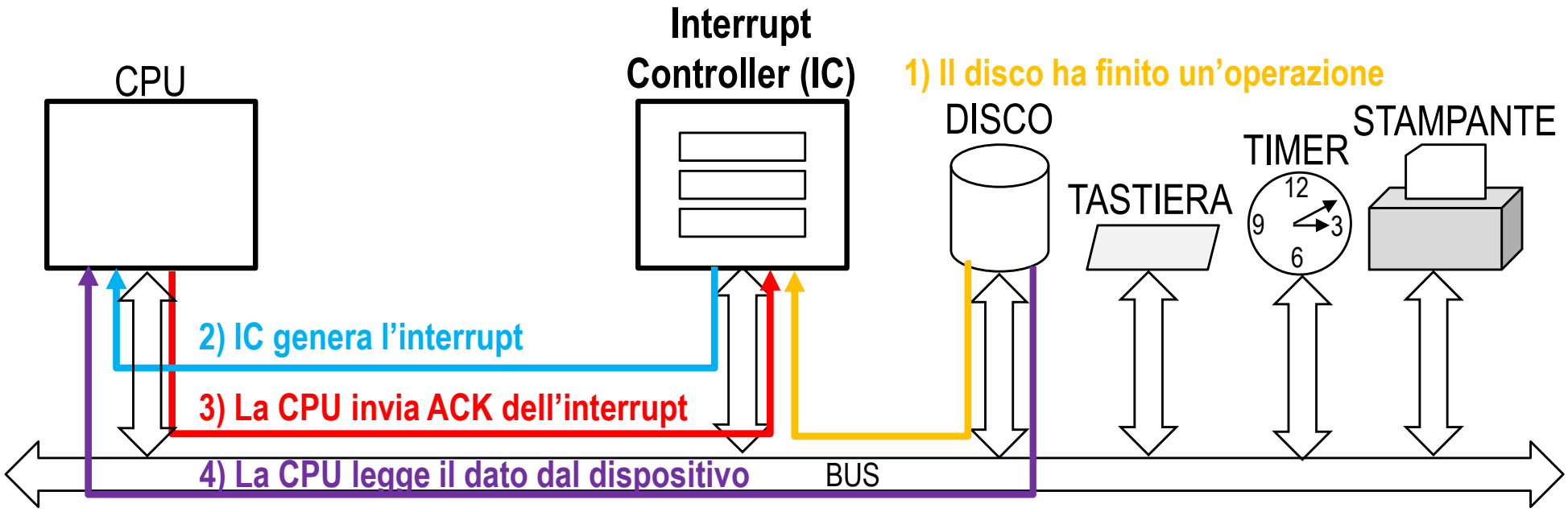


→ Protocollo ad INTERRUPT

- Evita l'attesa attiva del processore
  - Evita di fare numerosi controlli sullo stato del dispositivo
  - Il controller del dispositivo interrompe una sola volta quando ha bisogno

Protocollo ad Interrupt: funzionamento con più dispositivi

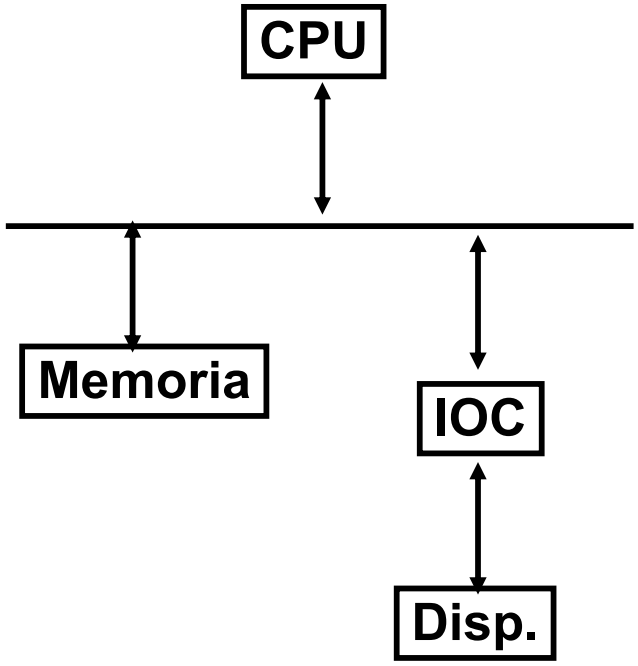
- La CPU dà un comando di lettura (es. dato da disco) e questo punto la CPU si occupa di altro… (trascorre tempo)…
  1) Quando il dato è pronto, il dispositivo invia un interrupt al controller di interrupt (IC), che quindi raccoglie gli eventuali interrupt dai vari dispositivi
  2) IC interrompe la CPU
  3) La CPU disabilita gli interrupt e notifica IC (ACK)
  4) La CPU legge il dato dal disco e riabilita gli interrupt

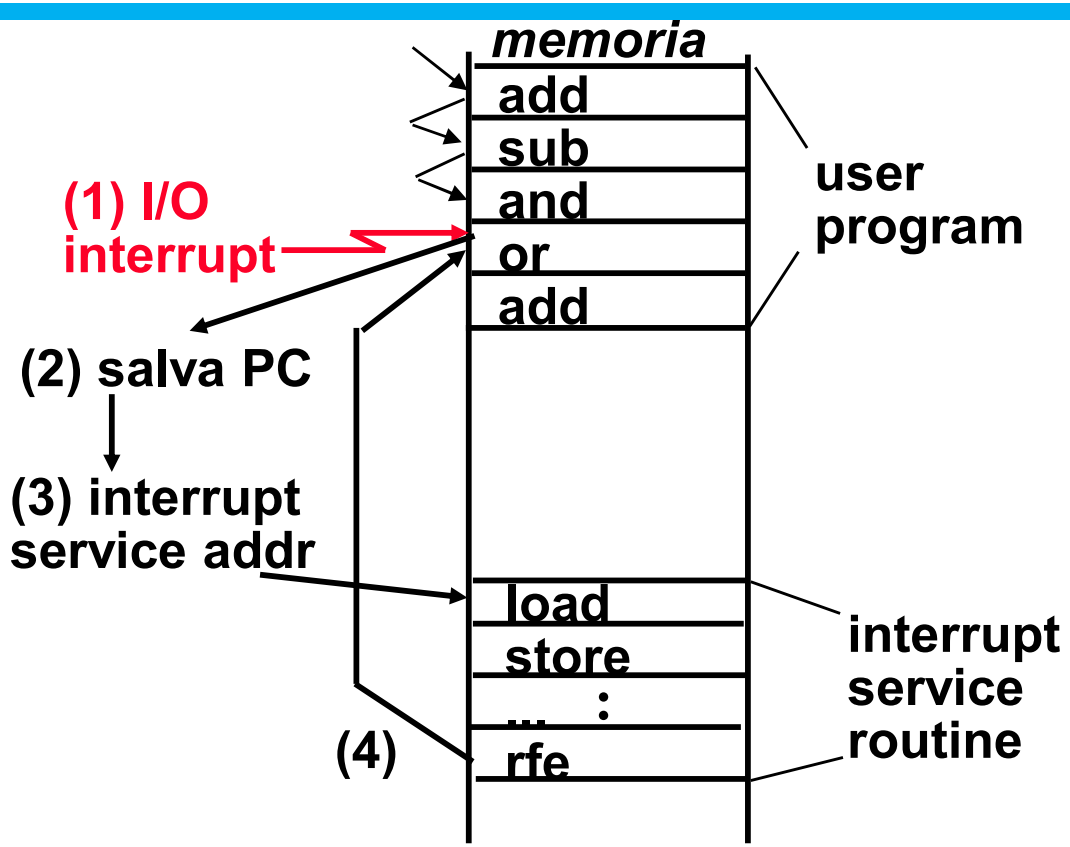


Dal punto di vista della CPU…

- La CPU dà un comando di lettura
- La CPU passa a fare altro lavoro
- Al termine di ogni istruzione guarda se c'è un interrupt pendente
- Se c'è un interrupt pendente
  - Salva il contesto
  - Serve l'interrupt: preleva il dato e lo mette in memoria

Trasferimento dati a Interrupt





- **Vantaggio:**
  - Il programma utente è bloccato solo durante il tempo effettivo per trasferire il dato dal dispositivo alla memoria
- **Svantaggio: è necessario hardware dedicato per**
  - Generare l'interrupt (lato controller di I/O)
  - Accorgersi dell'interrupt (lato CPU)
  - Salvare lo stato della CPU e ripristinare lo stato dopo l'interrupt

Stampa di una stringa ad Interrupt

Codice di preparazione alla stampa (es. Implementazione di un system call):

```
copy_from_user(buffer, p, count);

while (*printer_status_reg != READY);
*printer_data_register = p[0];
count = count - 1;
enable_interrupts();
scheduler();
```

Trasferisco il primo carattere

La chiamata allo scheduler comporta il blocco (cioè la sua deschedulazione dalla CPU) del processo che ha invocato questa system call

Routine di servizio dell'interrupt

```
if (count == 0) {
   unblock_user();
} else {
   *printer_data_register = p[k];
   count = count - 1;
   k = k + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

Inizialmente avrò k=1

Interrupt

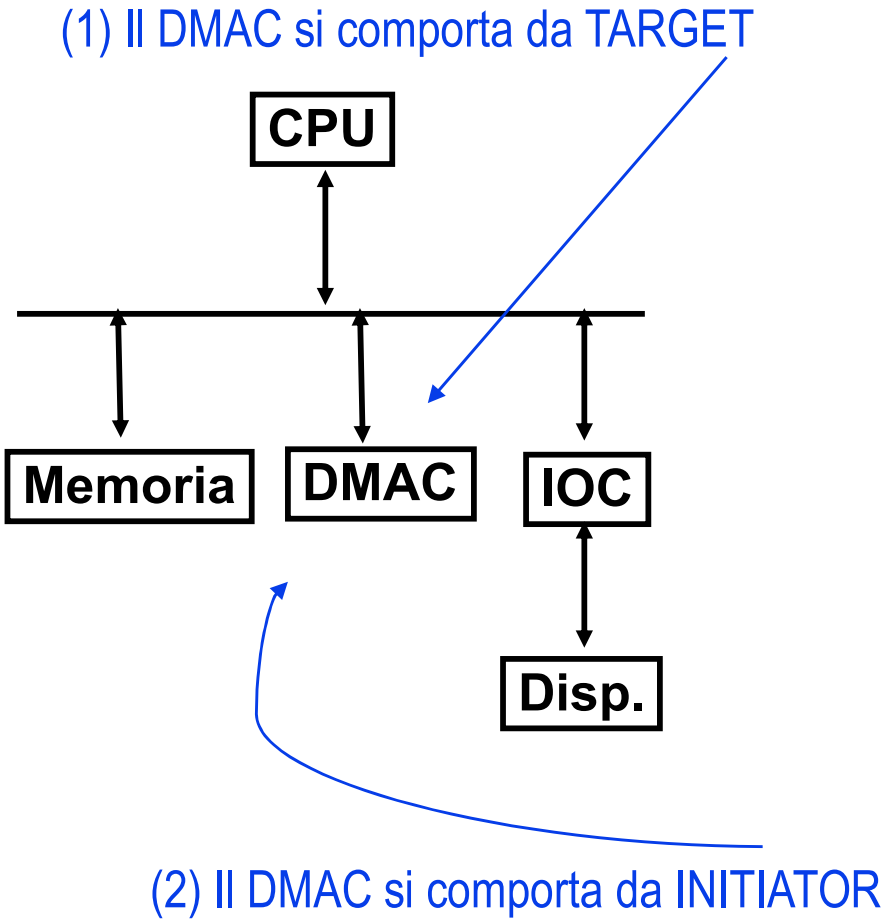
- Per la CPU, l'interrupt è esattamente come un'eccezione salvo che
  - L'interrupt è un evento asincrono
    - Non è provocato da istruzioni
    - Non impedisce il completamento di alcuna istruzione
  - Ad esso fa seguito un trasferimento di informazioni

- L'implementazione dell'interrupt è più complicata di quella di un'eccezione, perché
  - Deve consentire di identificare il dispositivo che ha generato l'interrupt
  - Le richieste di interruzione possono avere priorità diversa
  - Le richieste di interruzione provenienti da interrupt diversi possono sovrapporsi (interrupt multipli)

Identificare il dispositivo che ha generato l'interrupt

- Differenti linee per ogni controller
  - Usato nei PC
  - Svantaggio: limita il numero dei dispositivi
- Software poll
  - La CPU chiede ad ognuno dei controller se ha generato un interrupt
  - Lento
- Daisy Chain (Hardware poll)
  - Il segnale di Interrupt Acknowledge viene inviato in daisy-chain
  - Il controller è responsabile di mettere sul bus un "vettore"
  - La CPU usa il vettore per identificare la routine di servizio
- Bus Mastering
  - Il controller deve richiedere il bus prima di fare interrupt
  - e.g. PCI, SCSI

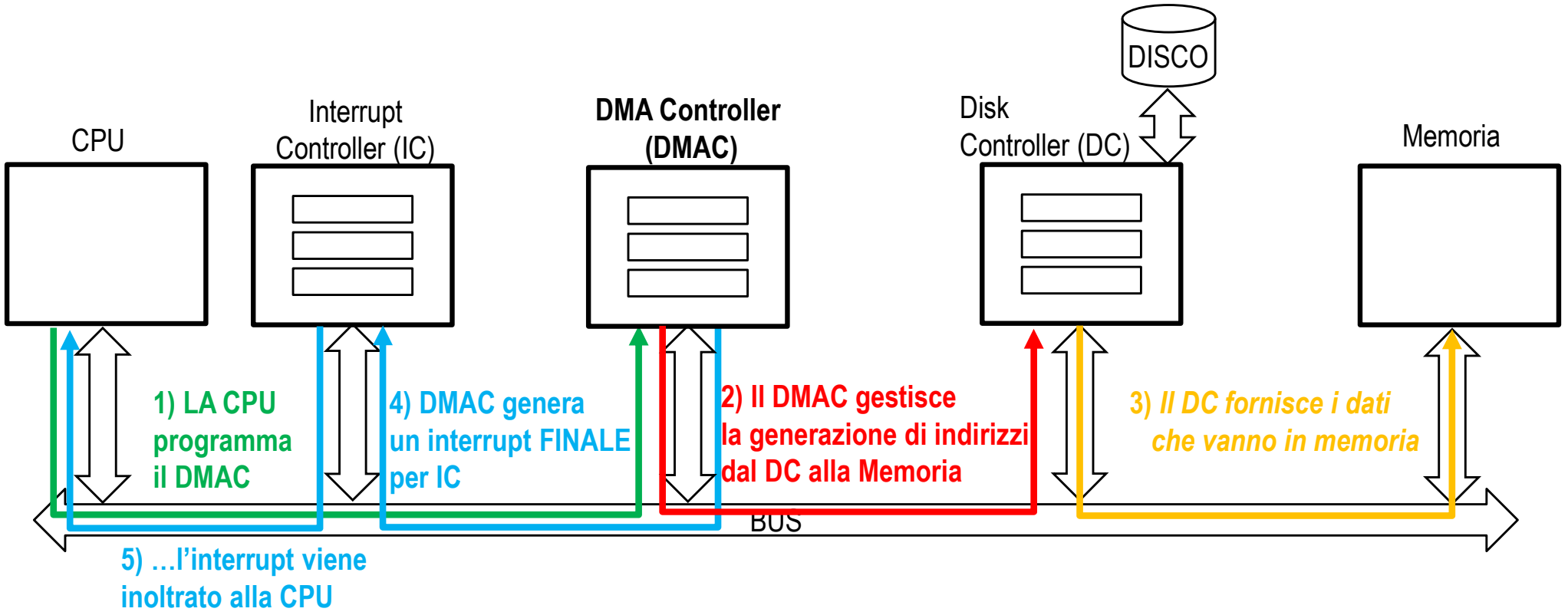
→ Direct Memory Access (DMA)



1) La CPU dice al DMA Controller:
   - Indirizzo iniziale della zona di memoria coinvolta
   - Indirizzo del controller del dispositivo
   - Direzione (Read/Write)
   - Quantità di dati da trasferire
   - Infine la CPU da lo "start"
- La CPU può svolgere altro lavoro

2) Il DMA Controller si occupa del trasferimento
   - genera tutti i necessari segnali per indirizzare la periferica e la memoria e per gestire l'handshake
- Il DMA Controller manda un interrupt quando ha finito

DMA -dettaglio



Stampa di una stringa con DMA

Codice di preparazione alla stampa (es. Implementazione di un system call):

```
copy_from_user(buffer, p, count);
setup_DMA_controller();
scheduler();
```

La chiamata allo scheduler comporta il blocco (cioè la sua deschedulazione dalla CPU) del processo che ha invocato questa system call→ il controllo viene quindi passato al sistema operativo che schedula un altro processo

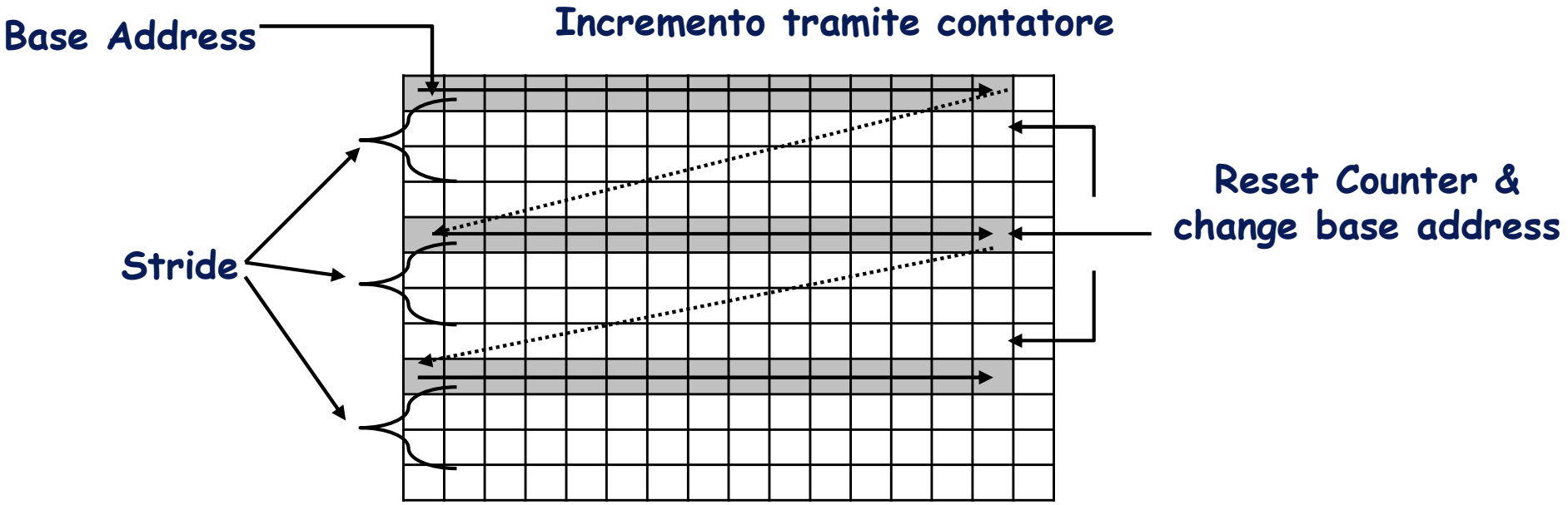
Routine di servizio dell'interrupt

```
acknowledge_interrupt();
unblock_user();
return_from_interrupt();
```

DMA Controller (DMAC)

- Il DMA può supportare diverse modalità di indirizzamento
  - 1D, ovvero un singolo registro per gli indirizzi
  - 2D, ovvero due registri per l'indirizzamento
  - 3D, ovvero tre o più registri per l'indirizzamento

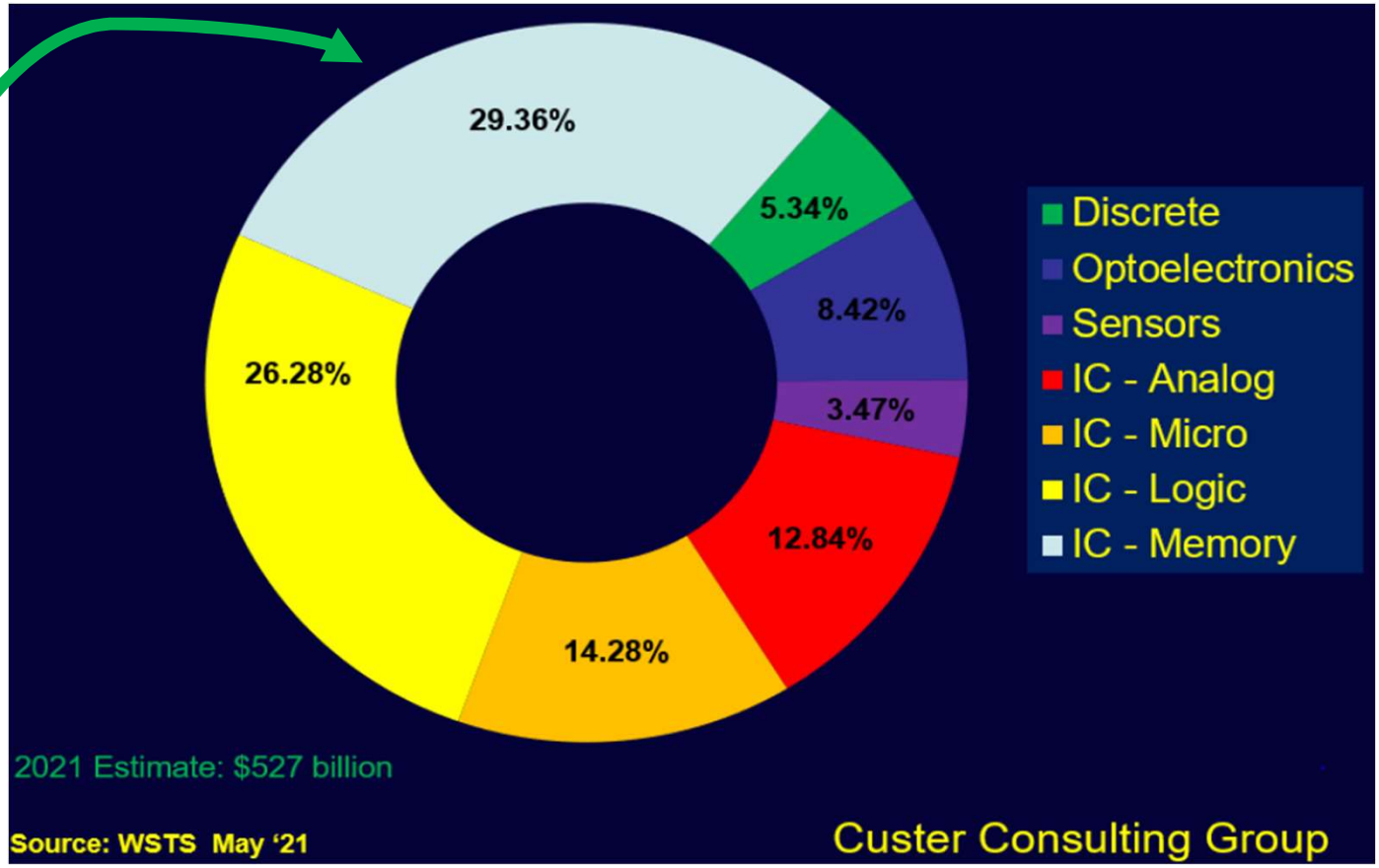
- Esempio di indirizzamento 2D
  - Utile ad es. per prelevare i payload dai pacchetti Ethernet



Lezione 17
Introduzione al sottosistema di memoria

Patterson: 5.1,5.2

Market Share dei Principali Componenti a Semiconduttore



Le memorie rappresentano il 29% del mercato dei semiconduttori

Tecnologie usate per realizzare le memorie

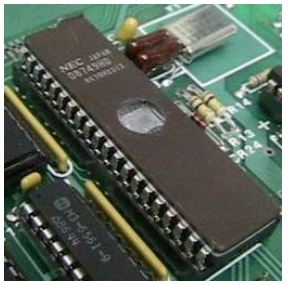
- Tecnologie ad accesso sequenziale
  - Il tempo di accesso dipende in maniera lineare dalla locazione fisica
    - Nastri

- Tecnologie ad accesso semicasuale
  - Il tempo di accesso dipende sia dalla locazione fisica dell'informazione che dal particolare istante in cui faccio accesso
    - Dischi: dischi-fissi, CDROM, DVD, ...

- Tecnologie ad accesso casuale (Random Access)
  - Il tempo di accesso è indipendente dalla locazione fisica dell'informazione
    - Memoria principale
    - Due tipi: Non-Volatili e Volatili

Memorie accesso casuale di tipo Non-Volatile (NVM)

- «Non-Volatile» → conservano i dati anche se tolgo l'alimentazione
- ROM (Read-Only Memory)
  - Non scrivibile (sia vantaggio che svantaggio)
- EPROM (Erasable Programmable Read-Only Memory)
  - ✓ Scrivibile (ma una sola volta)
  - ✓ Cancellabile a livello di chip (raggi UV)
  - ☹ Necessario intervenire con apposito "kit di cancellazione"


[1]

- EEPROM (Electrically-Erasable Programmable Read-Only Memory)
  - ✓ Scrivibile (1 sola volta)
  - ✓ Cancellabile a livello di byte (elettricamente)
  - ☹ Dovendo scrivere molti byte si perde molto tempo nelle cancellazioni
- Memoria "Flash": Evoluzione della EEPROM
  - ✓ Scrivibile (1 sola volta)
  - ✓ Cancellabile a livello di blocco (elettricamente)

  → Per rendere possibile «l'illusione di leggere/scrivere» occorre aggiungere un apposito controller che effettua in modo opportuno scritture e cancellazioni

[1] https://en.wikipedia.org/wiki/EPROM#/media/File:NEC_D8749HD.png

Memorie accesso casuale di tipo Volatile

- «Volatile» → conservano i dati solo se alimentate
- Principali tipi di implementazioni:
  - DRAM (Dynamic Random Access Memory)
    - ✓ Dense (molti bit per unità di superficie), basso consumo, basso costo
    - ☹ Lente, dinamiche → Necessità di essere "rinfrescate" periodicamente
  - SRAM (Static Random Access Memory)
    - ✓ Veloci, statiche → basta alimentare il chip per non perdere i dati
    - ☹ Bassa densita', alto consumo, alto costo

- Nuove implementazioni di DRAM:
  - DDRx, HBM, …

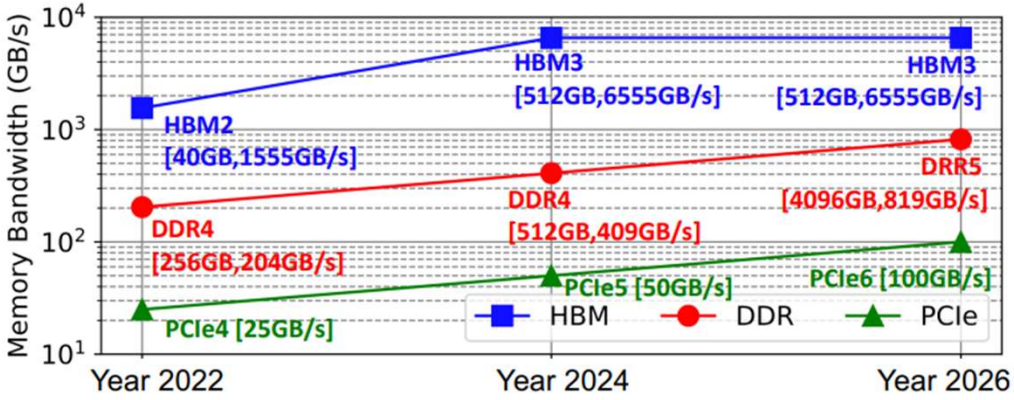


Fig. 1: Memory bandwidth trends for HBM, DDR and PCIe from years 2022 to 2026. Relative bandwidth improvements remain constant. The PCIe is the performance bottleneck for disaggregated systems.

Joël Lindegger and others, Scrooge: a fast and memory-frugal genomic sequence aligner for CPUs, GPUs, and ASICs, Bioinformatics, Volume 39, Issue 5, May 2023

Parametri delle Prestazioni della Memoria

t_a=tempo di accesso (Access Time)

- Intervallo fra l'inizio di una lettura (indirizzo su A-bus) e l'arrivo del PRIMO dato (su D-bus)

t_c=tempo di ciclo (Cycle Time)

- Intervallo fra l'inizio di una lettura e l'inizio della lettura successiva

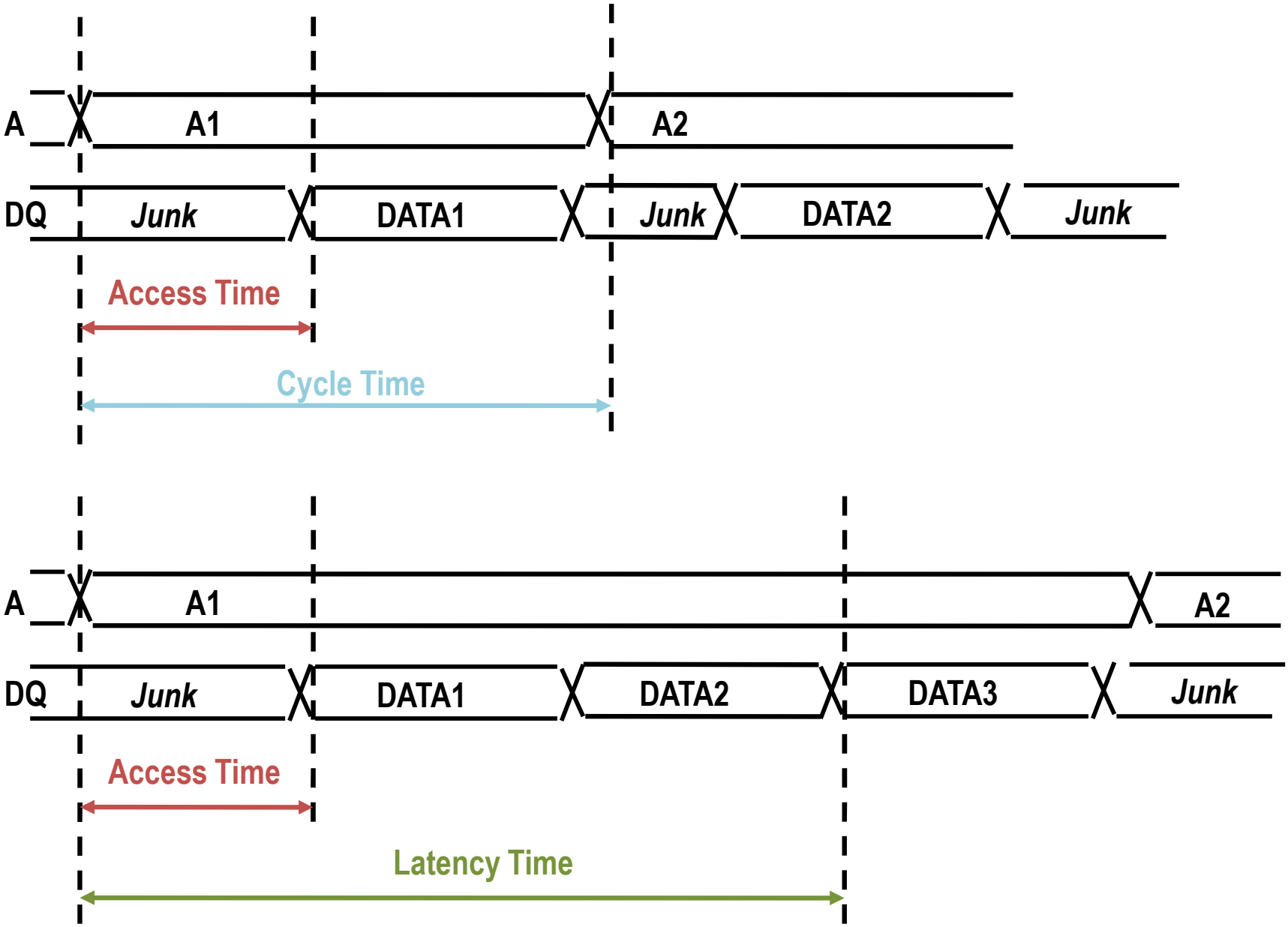
t_l=tempo di latenza (Latency Time)

- Tempo di accesso al DATO COMPLETO ovvero alla k-esima word nel caso in cui la memoria supporti trasferimenti a gruppi di k word (se k=1 $t_l=t_a$). Es. dischi, ma anche memorie RAM

ω=Banda (Bandwidth)

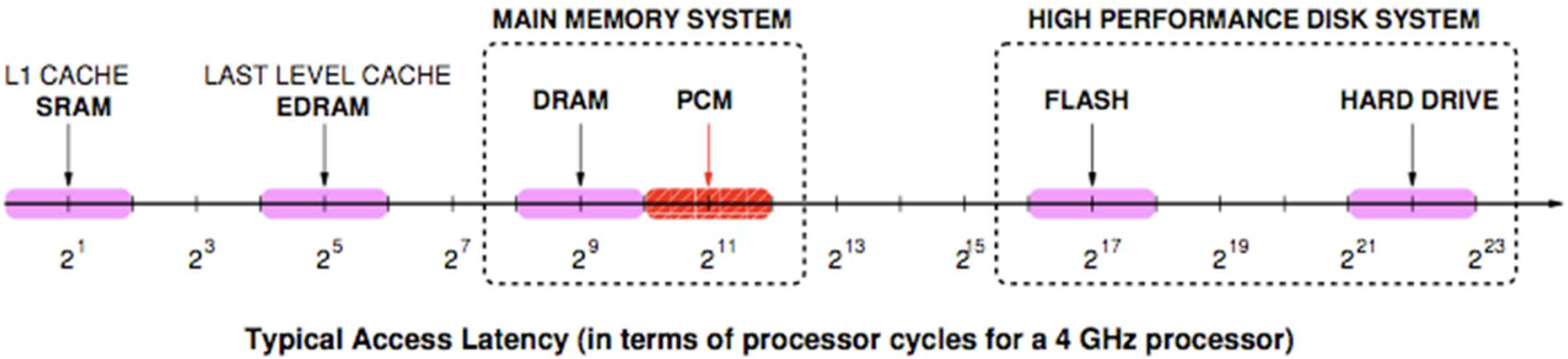
- Tasso di trasmissione dei byte (si misura in byte/s)
- A parità di tempo di ciclo posso trasmettere più byte in parallelo

Tempi caratteristici delle memorie



DRAM versus SRAM

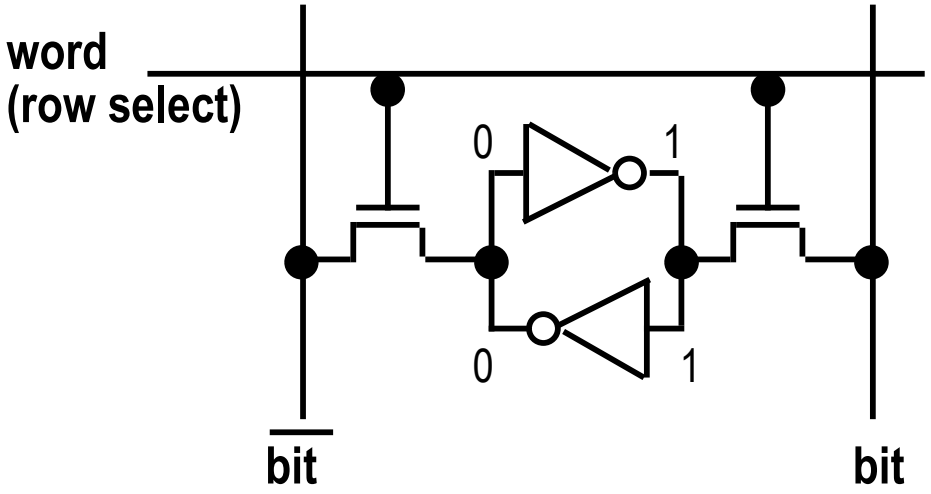
- Per massimizzare il parametro prestazioni/costo
  - Si usa poca memoria SRAM (costosa e veloce) all'interno del chip (es. registri)
  - Si usa una ragionevole quantità di DRAM (meno costosa e relativamente veloce) all'esterno del chip



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Cella di RAM Statica (SRAM)

Cella a 6 Transistor



Scrittura

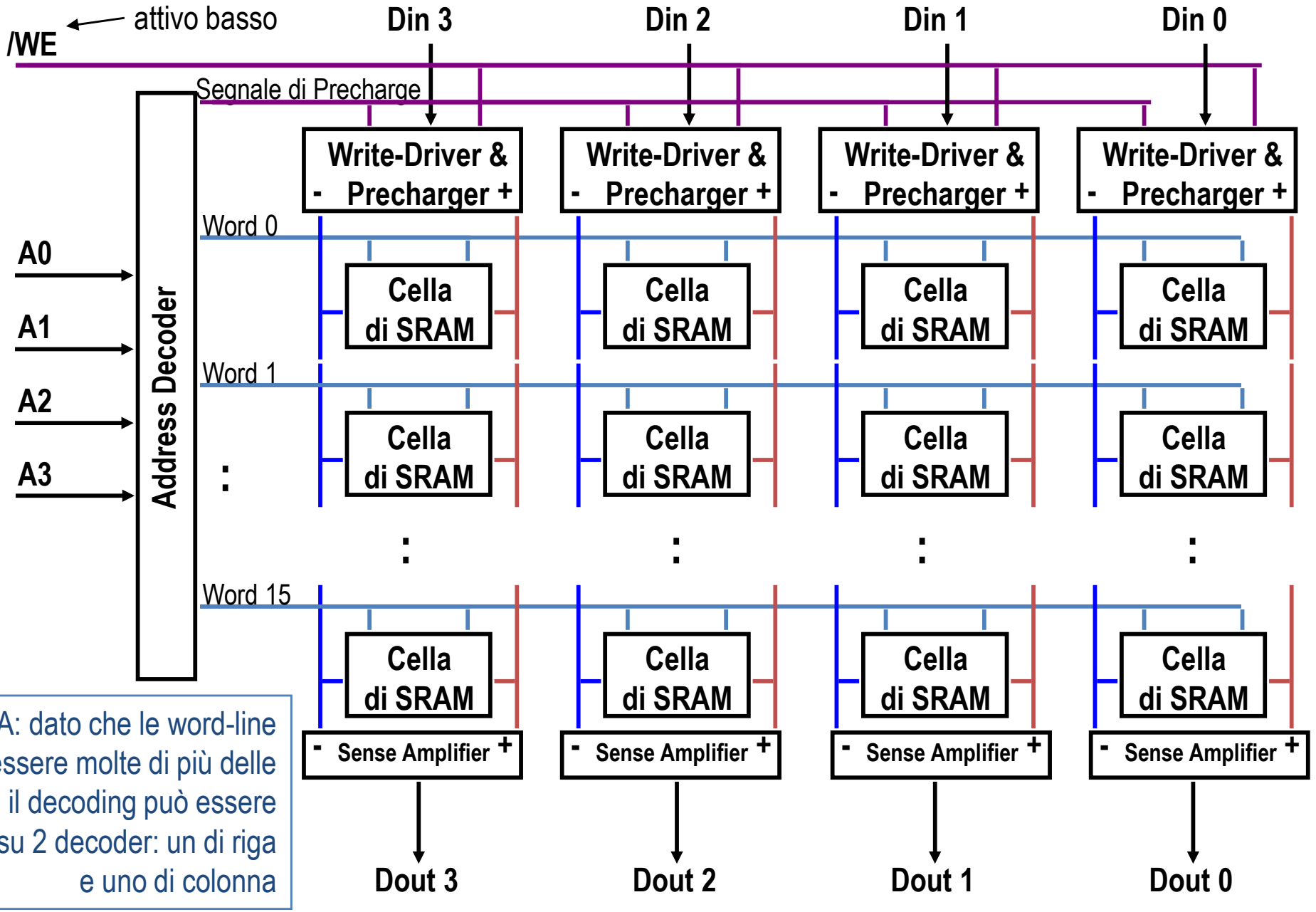
1. Preparare il dato sulle bit-line
2. Selezionare la riga (word-line)

Lettura

1. Precaricare bit e /bit a Vdd *
2. Selezionare la riga (word-line)
3. La cella preleva carica da bit o /bit
4. Il Sense-Amplifier in fondo alla colonna amplifica la differenza fra bit e /bit

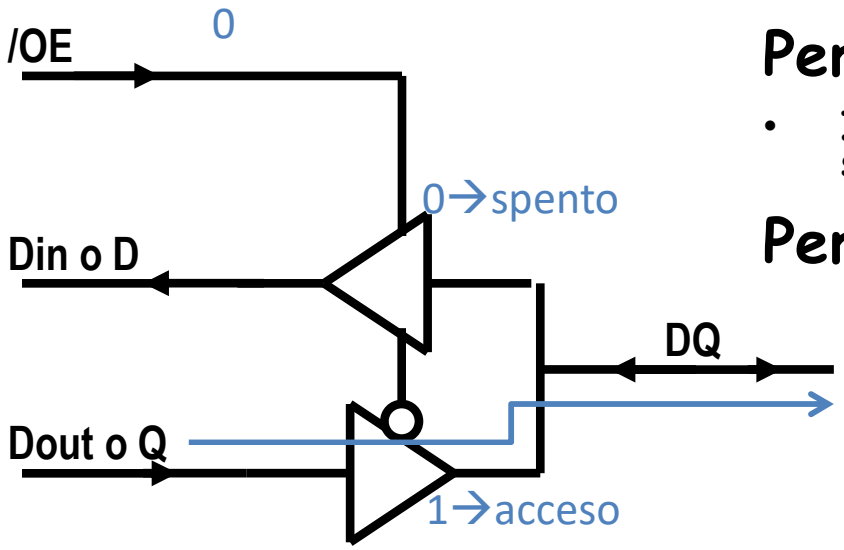
*in realtà la linee bit e /bit si caricheranno ad un valore un poco <Vdd

Organizzazione di SRAM con 16 word da 4 bit (16x4)



NOTA: dato che le word-line possono essere molte di più delle data-line, il decoding può essere suddiviso su 2 decoder: un di riga e uno di colonna

SRAM: Bus-drivers ovvero interfaccia verso il bus dati



Per Din e Dout si usa un unico filo

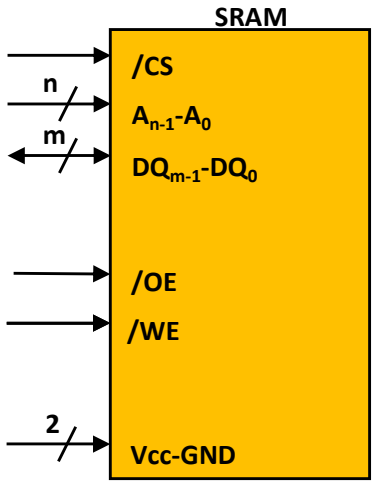
- Il motivo principale è perché sul bus dati posso avere scambio in entrambe le direzioni (lettura o scrittura)

Per decidere la direzione si usa /OE:

- /OE → Output Enable (attivo basso), per controllare i buffer thristate:
  1 sul filo tristate → buffer attivo
  0 sul file tristate → buffer disattivo

- Le possibili combinazioni sono:
  - Scrittura:
    - /WE attivo (basso), /OE disattivo (alto) → DQ è un ingresso
  - Lettura:
    - /WE disattivo (alto), /OE attivo (basso) → DQ è un'uscita
  - Evitare di attivare contemporaneamente i segnali /WE e /OE
    - Il risultato risulta indeterminato
- Quando il chip non deve caricare il bus dati, si disabilita con /CS
  - /CS disattivo → alta impedenza

SRAM: Diagramma Logico



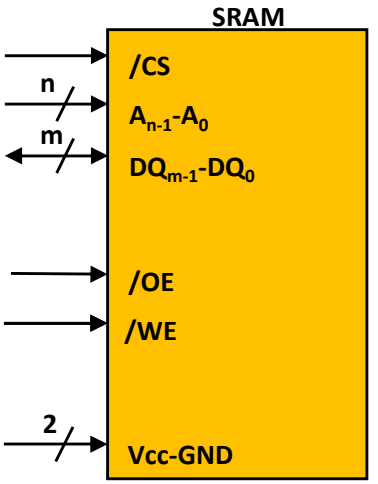
Chip di SRAM da
2^n x m bit

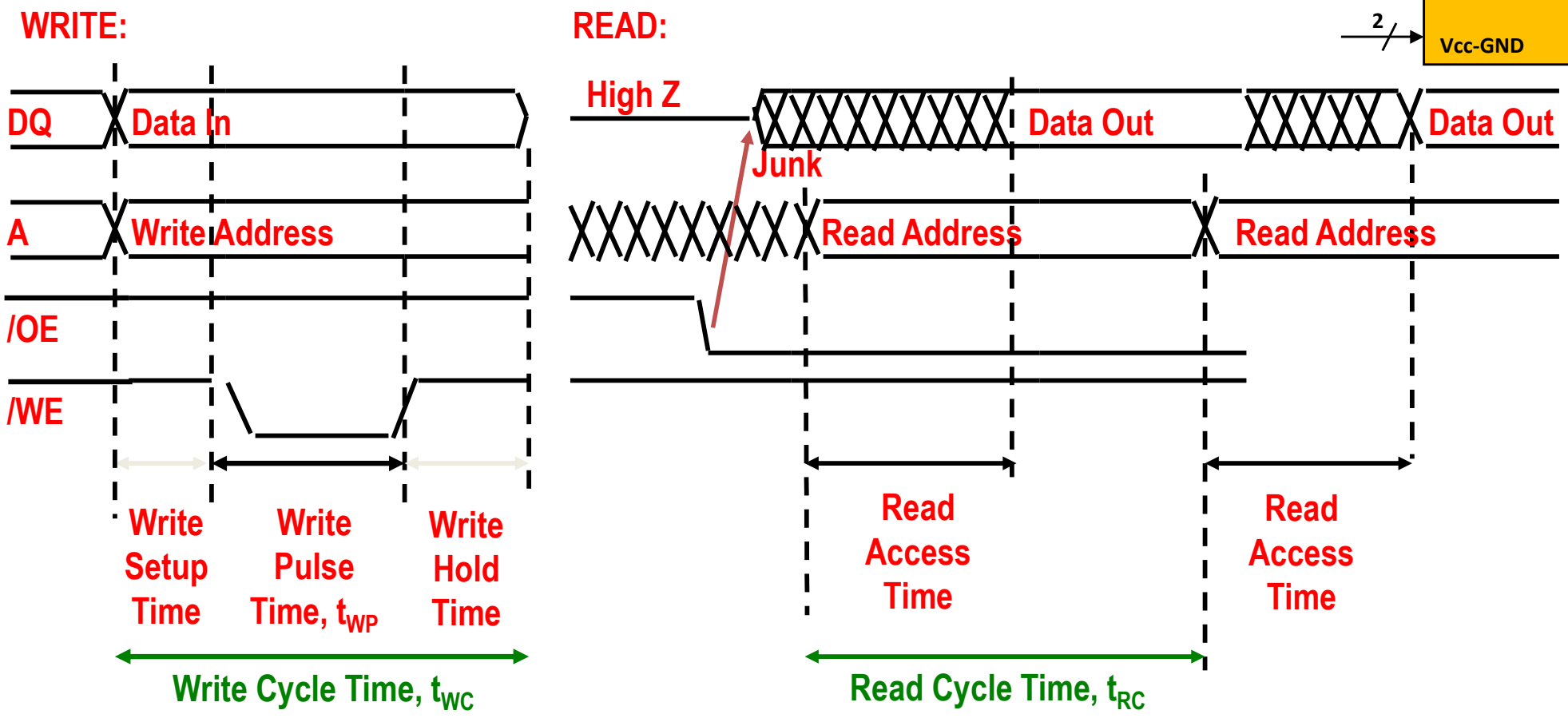
- La memoria ha una capacità di 2^n word ad m-bit

- Altri segnali sempre presenti
  - /CS (o /CE) → Chip Select o Chip Enable
  - Vcc, GND → Alimentazione

SRAM: Temporizzazioni

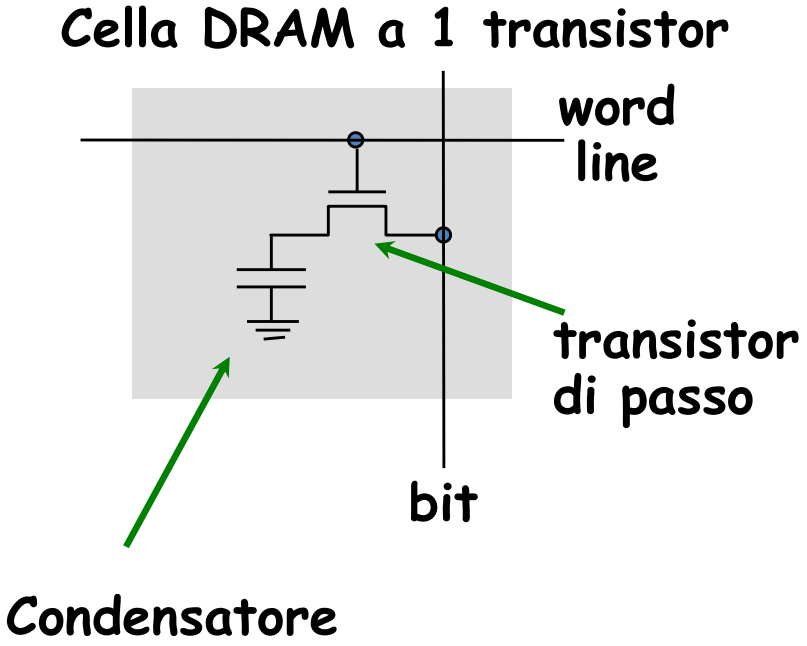
Ricordare che la CPU agisce come Initiator
Mentre la memoria agisce come Target





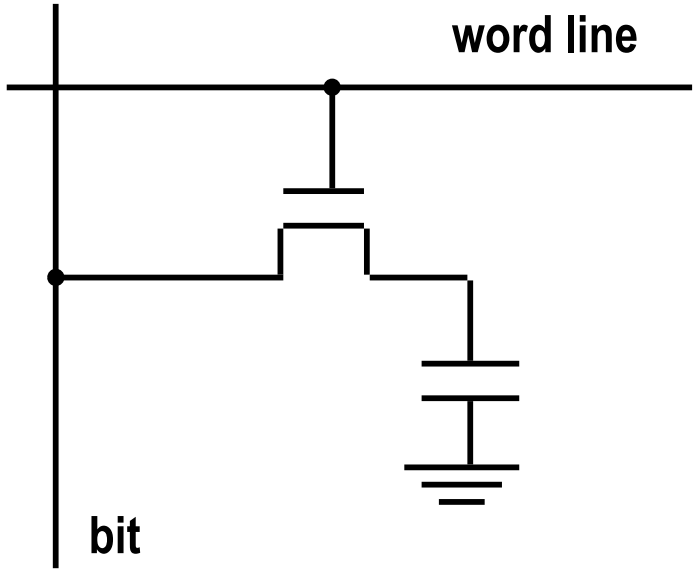
Cella di RAM dinamica (DRAM)

- La cella a 6 transistor per quanto possa sembrare piccola, ha un peso notevole sull'area totale di una RAM
- Qual è il numero minimo di transistor che può essere usato per memorizzare un bit?
- Un solo transistor → l'elemento di memorizzazione può essere realizzato con un condensatore
  - Condensatore carico → 1 logico
  - Condensatore scarico → 0 logico
  - Il transistor consente la lettura e la scrittura



DRAM: Funzionamento della cella a 1 transistor

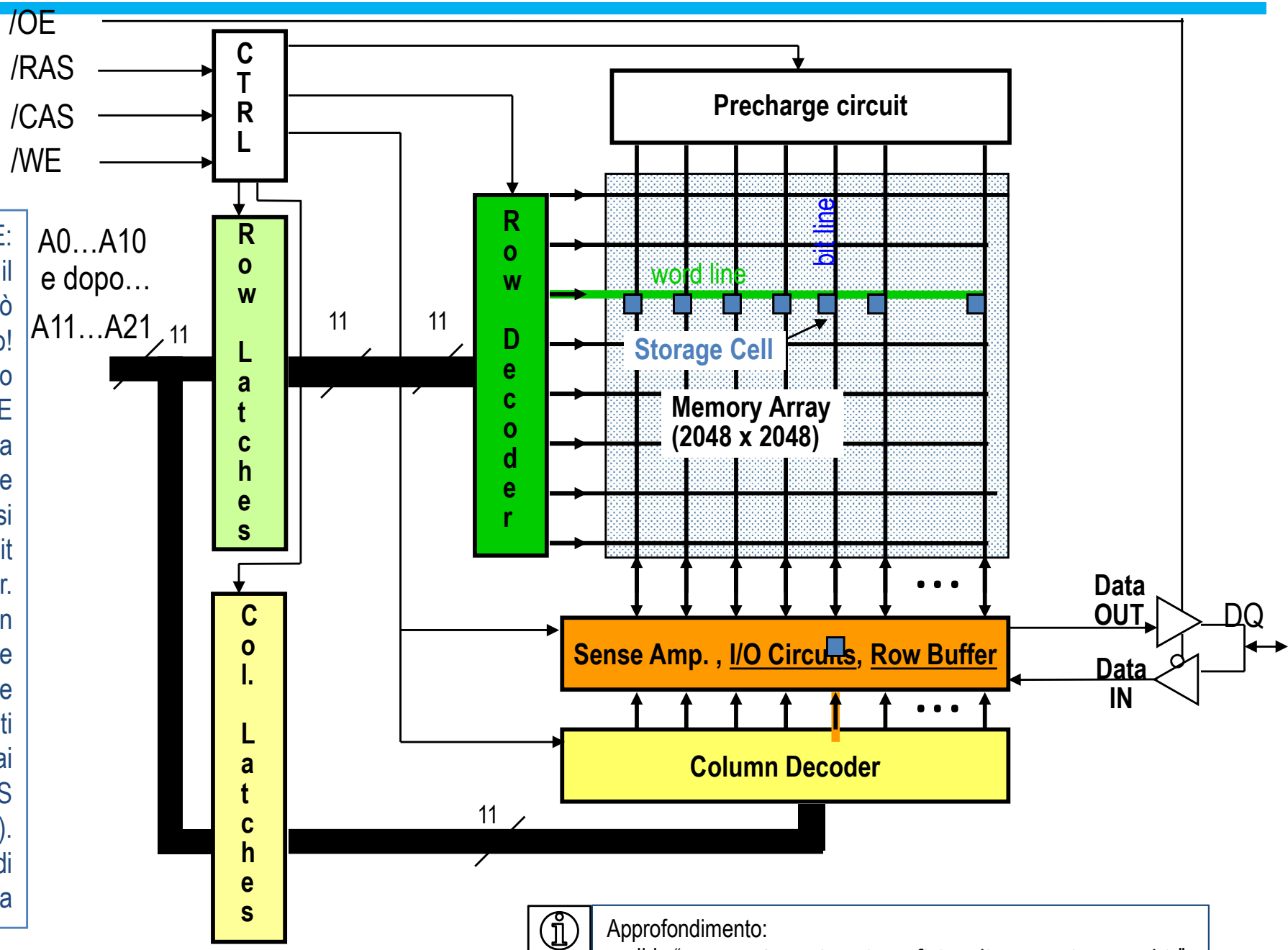
- **Scrittura:**
  1. Caricare la bit-line col dato
  2. Selezionare la riga (word-line)
- **Lettura:**
  1. Precaricare la bit-line a Vdd/2
  2. Selezionare la riga (word-line)
  3. Scambio di carica fra cella e bit-line
     - lievissima variazione di tensione sulla bit-line
  4. Rilevazione della variazione (Sense-Amplifier molto sofisticato)
     - l'amplificatore deve essere in grado di rilevare una variazione di carica di circa 1 milione di elettroni
  5. Caricare la bit-line col dato letto (scrittura !)
     - Ripristino il contenuto della cella
- **Refresh**
  1. Effettuo una "finta" lettura di <u>tutte</u> le celle
     - Ripristino il contenuto di <u>ogni</u> cella



N.B. La cella ha <u>un solo filo</u> di accesso (word-line)
→ separazione ROW/COL

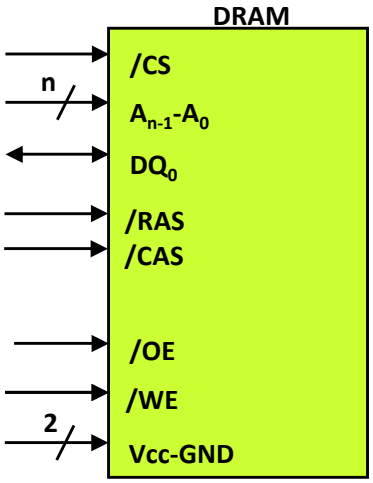
DRAM: Organizzazione fisica di memoria a 4 Mbit

NOTA BENE:
differentemente dalle SRAM il precharge-circuit non può convogliare il dato!
E` quindi necessario effettuare la scrittura in DUE FASI: 1) si legge una intera riga di 2048 bit e la si scrive in un buffer di riga; 2) si seleziona 1 singolo bit all'interno del buffer.
Perciò l'indirizzo è spezzato in due parti (indirizzo di riga e indirizzo di colonna, che vengono inviati in due istanti successivi (marcati dai segnali RAS e CAS rispettivamente).
NON si inviano quindi 22 bit di indirizzo ma solo 11 per volta



Approfondimento:
v. slide "DRAM: Organizzazione fisica di memoria a 4 Mbit"

DRAM: Visione a livello di chip



Chip di DRAM da 2^{2n} x 1 bit

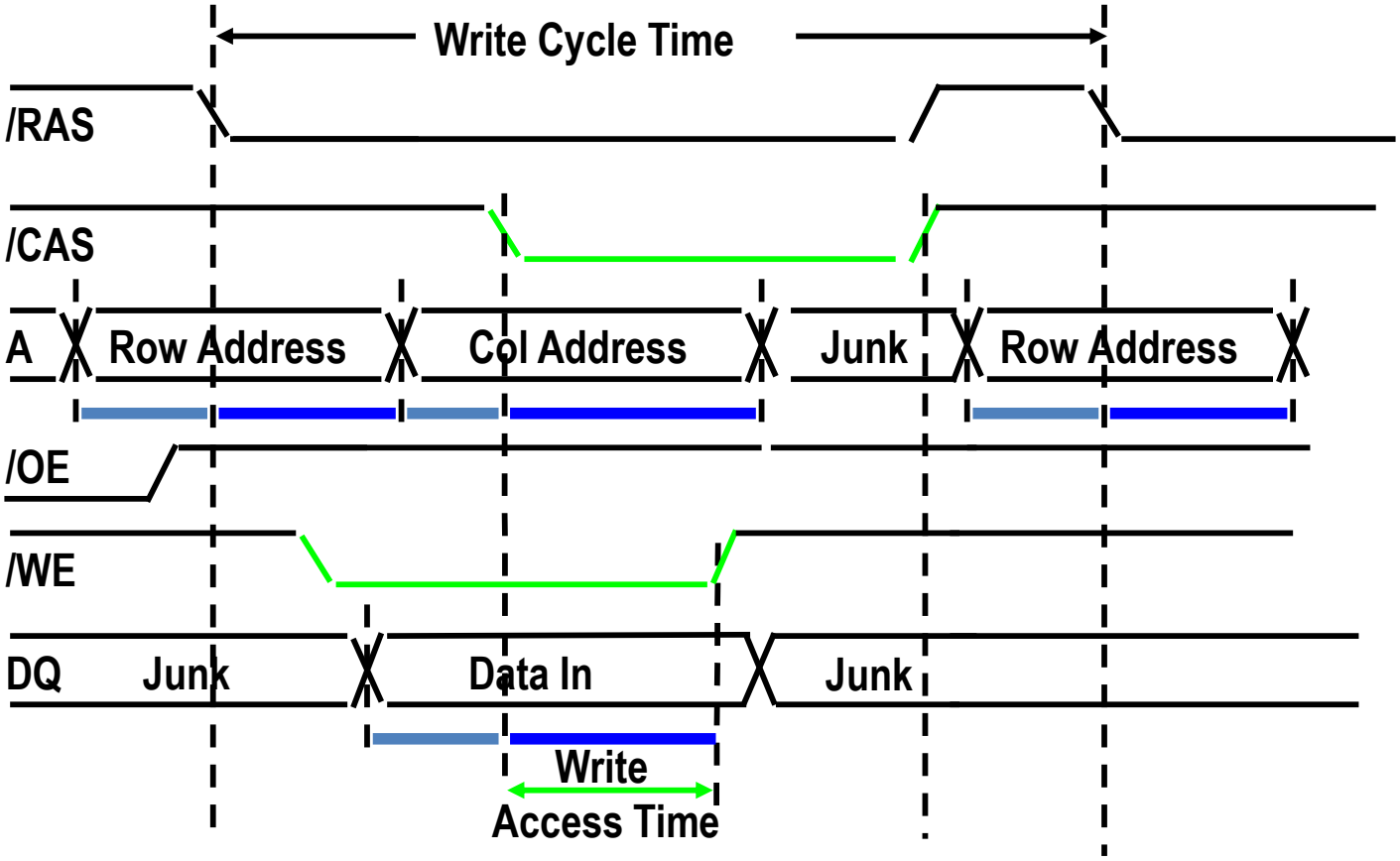
(nella slide precedente n=11 quindi 2^{22} x 1 bit ovvero 4Mbit x1)

- Din e Dout sono multiplexati su DQ:
  - Scrittura:
    - /WE attivo (basso), /OE disattivo (alto) → DQ è un ingresso
  - Lettura:
    - /WE disattivo (alto), /OE attivo (basso) → DQ è un'uscita
- Gli indirizzi di Riga e di Colonna condividono i pin A
  - /RAS va basso → i latch di riga memorizzano i pin A
  - /CAS va basso → i latch di colonna memorizzano i pin A
  - Nota: RAS/CAS sono sensibili al fronte in discesa

Nota: In alcuni chip di DRAM il segnale OE non c'è e invece di DQ ci sono Din e Dout

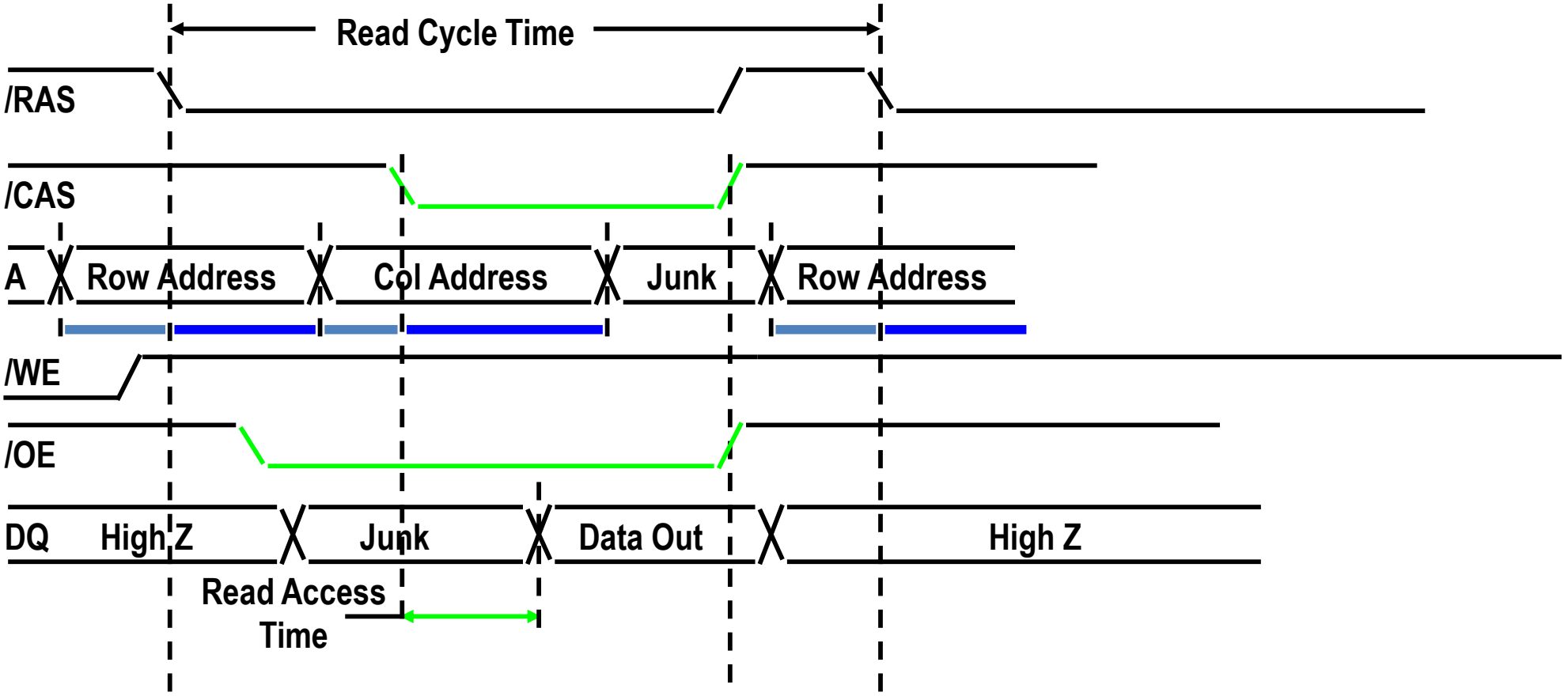
DRAM: Ciclo di scrittura

- L'accesso inizia con l'attivazione di /RAS

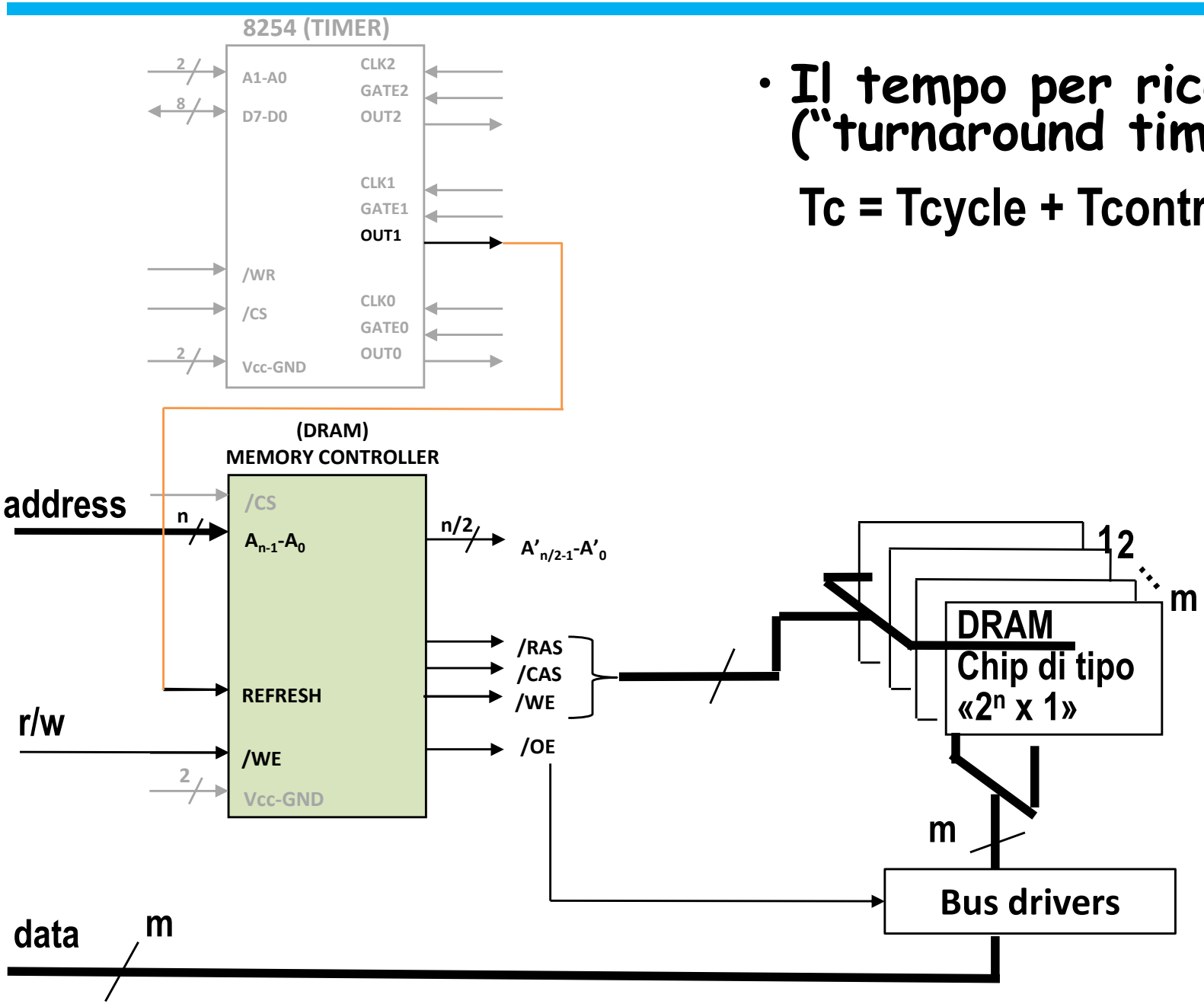


DRAM: Ciclo di lettura

- L'accesso inizia con l'attivazione di /RAS



DRAM Controller (Memory Controller)



- Il tempo per ricevere un dato ("turnaround time") è dato da

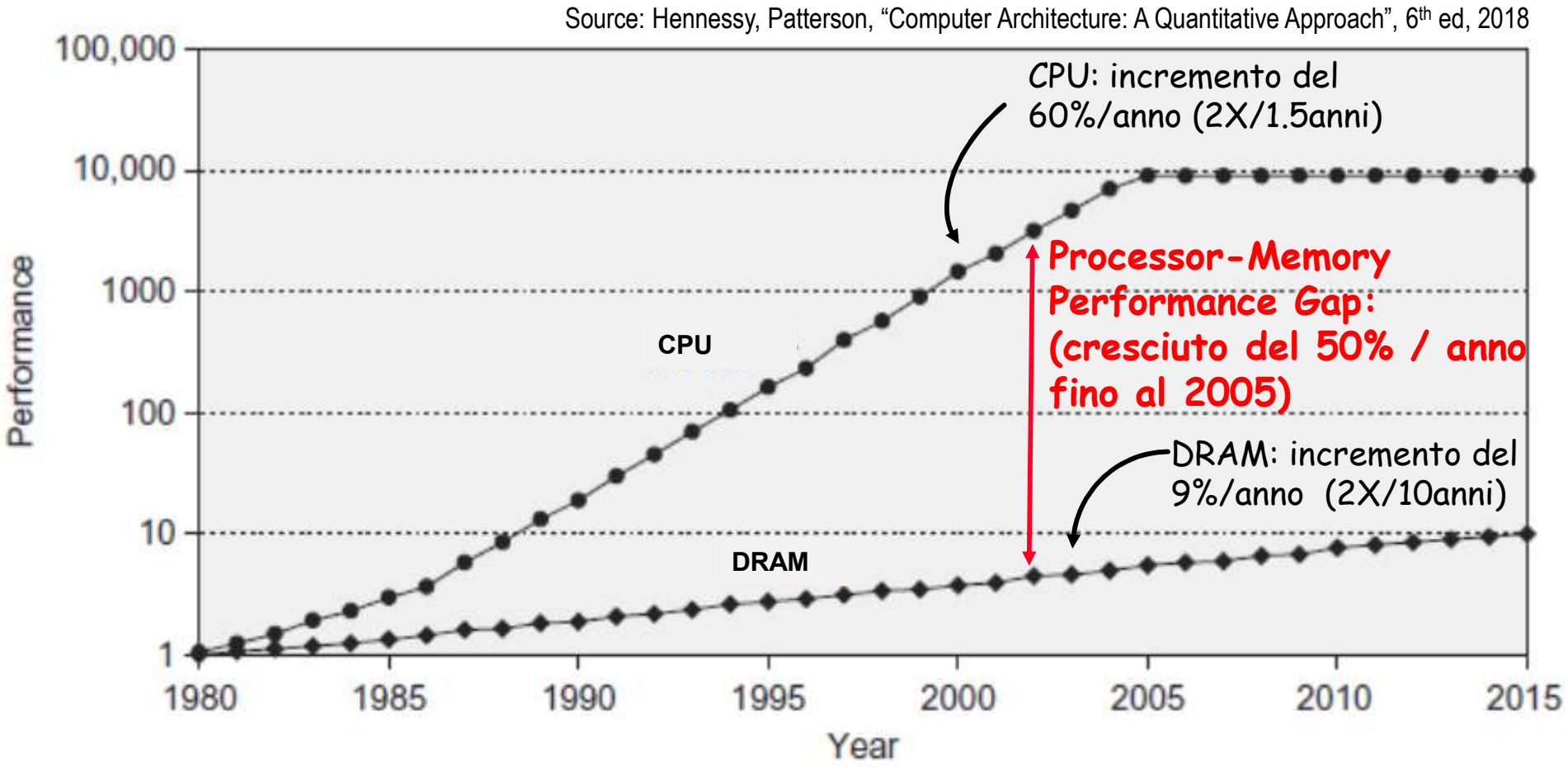
$$Tc = Tcycle + Tcontroller + Tdriver$$

N.B. è il DRAM controller che avrà cura di inviare (in due passi) metà degli indirizzi ai chip di DRAM sincronizzandoli sui segnali RAS e CAS

Lezione 18
Introduzione alla memoria cache

Patterson: 5.3

- Gap Prestazioni Processore-Memoria



→ Necessità di nuove architetture per le memorie

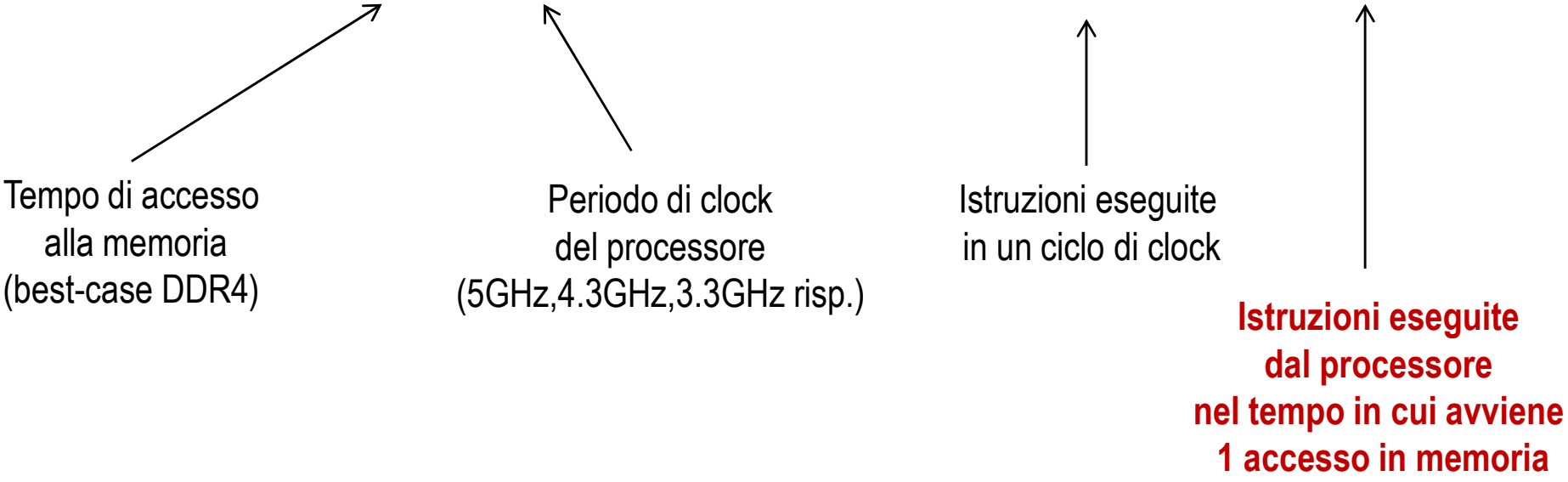
Gap Processore-Memoria: un esempio

Valutiamo il gap Processore-Memoria in termini di istruzioni per eseguite dal processore durante 1 accesso in memoria

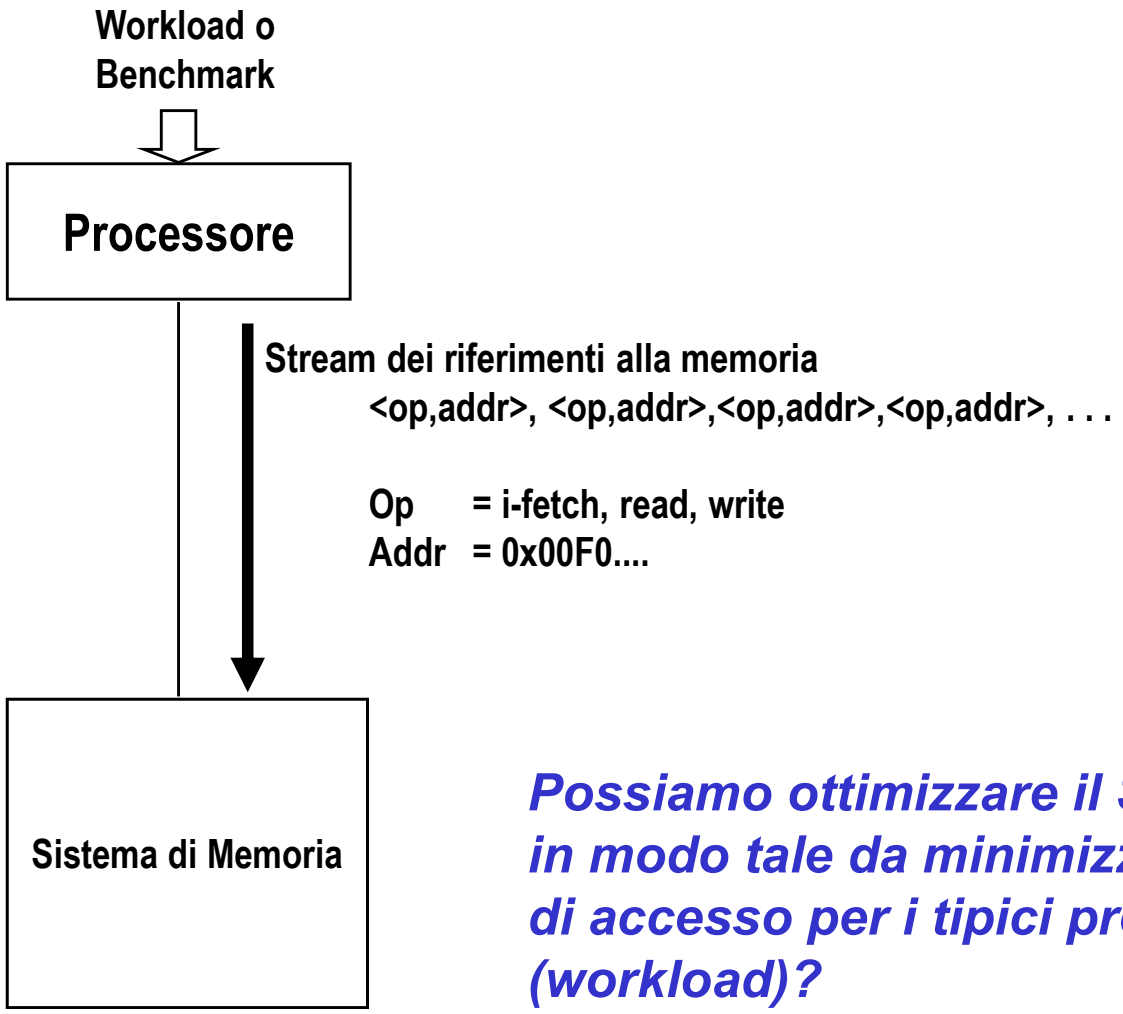
Intel Core i7-9900k: 26 ns/0.2 ns = 130 clk x 4istr/clk →520 istr.

AMD Ryzen 2700X: 26ns/0.23ns = 113 clk x 4istr/clk →452 istr.

IBM Power8: 26ns/0.3ns = 86.7 clk x 8ist/clk →693 istr.



"L'Arte della Progettazione del Sistema di Memoria"



Possiamo ottimizzare il Sistema di Memoria in modo tale da minimizzare il tempo medio di accesso per i tipici programmi che utilizziamo (workload)?

Principio di Località dei Programmi

- Un riferimento in memoria tende ad essere ripetuto dopo poco tempo

  → LOCALITA' TEMPORALE

  - Esempio:
    - 0008
    - 1000
    - 0008
    - 2000
    - 0008

- Un riferimento in memoria tende ad essere a locazioni vicine a quelle usate recentemente

  → LOCALITA' SPAZIALE

  - Esempio:
    - 0004
    - 0008
    - 000C
    - 0010
    - 0014

Implicazioni del Principio di Località

1) Posso utilizzare piccole memorie veloci per mantenere i riferimenti in memoria più utilizzati dal processore

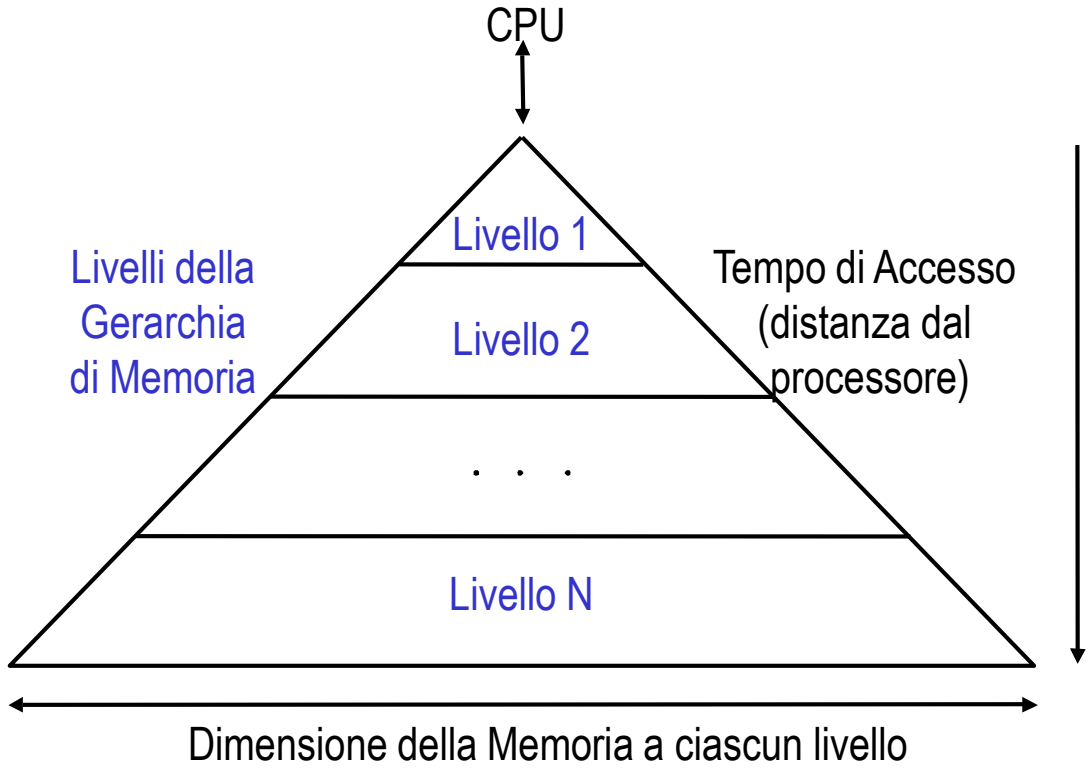
→ MEMORIA CACHE

2) Se i riferimenti utilizzati non possono essere soddisfatti nella memoria veloce

→ creo un secondo livello di memoria un po' più grande e un po' meno veloce

3) Posso reiterare questo ragionamento per N livelli di memoria

→ GERARCHIA DI MEMORIA



Gerarchia di Memoria

- Presenta al processore una quantità di memoria pari a tutta quella disponibile all'ultimo livello (realizzata con la tecnologia più economica)
- Fornisce un tempo di accesso che si avvicina a quello del primo livello (in cui posso usare la tecnologia più veloce possibile). Es. con 2 soli livelli:
  - Primo Livello: Cache, tecnologia SRAM
  - Secondo Livello: Memoria Principale, tecnologia DRAM
- Matematicamente:
  - h= probabilità di trovare un dato nel primo livello - Maggiore LOCALITA' comporta $h \rightarrow 1$
  - m= probabilità di trovare un dato nel livello successivo (ovviamente h+m=1)
  - t_{hit}= tempo di accesso al dato dal primo livello
  - t_{miss}= tempo di accesso al dato nel livello successivo

→Average Memory Access Time (AMAT):

$$AMAT = h \times t_{hit} + m \times t_{miss}$$ se $h \rightarrow 1$ allora $AMAT \rightarrow t_{hit}$

Se scomponiamo MissTime = HitTime + PenaltyTime

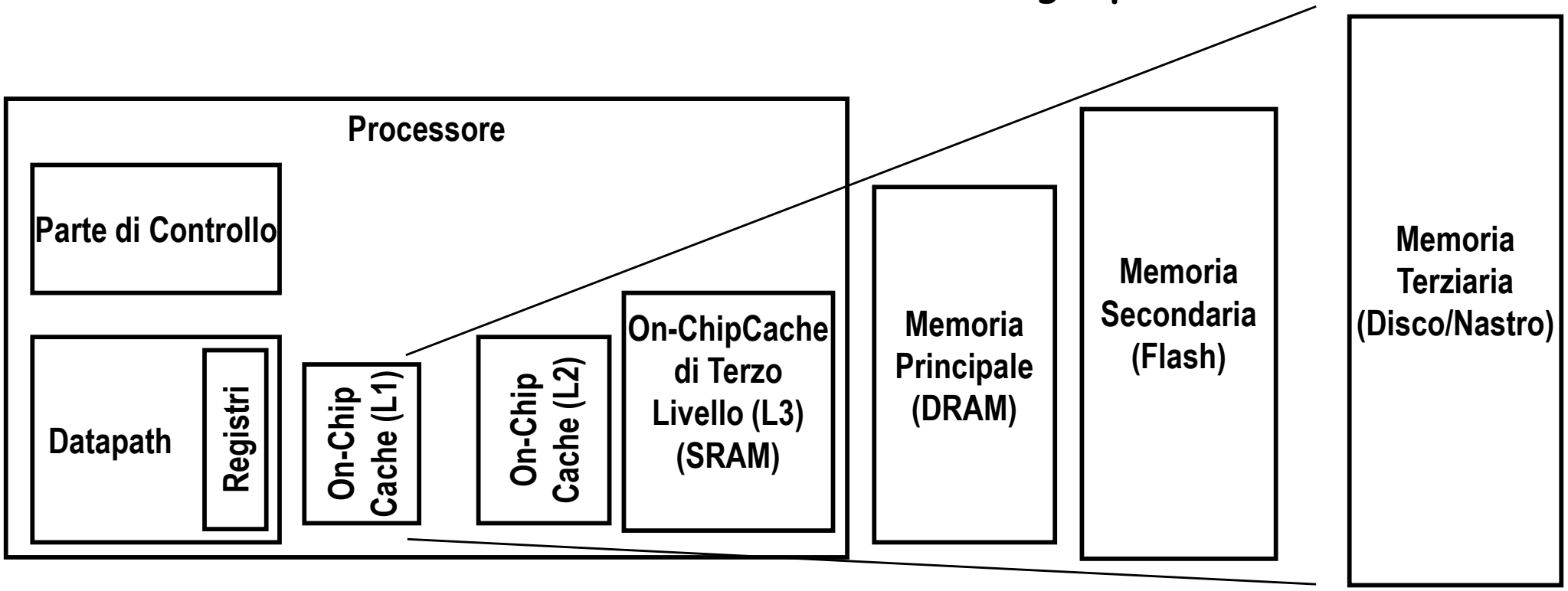
$$= h \times t_{hit} + m \times (t_{hit} + t_{penalty}) = t_{hit} + m \times t_{penalty}$$

Terminologia

blocco: insieme di byte a locazioni contigue ed indirizzi multipli della dimensione del blocco
hit: il riferimento può essere soddisfatto nel livello di memoria immediatamente successivo
miss: il riferimento richiede un accesso a livelli di memoria oltre il successivo (più lenti)

Organizzazione gerarchica della memoria

- Sfruttando il principio di località è possibile:
  - Presentare all'utente tutta la memoria presente al livello della tecnologia meno costosa (es. disco o nastro)
  - Fornire la velocità di accesso offerta dalla tecnologia più veloce



Velocità (ns):	0.3-1	3-10	10-20	100	10,000,000 (10 ms)	10,000,000,000 (10 s)
Capacità (bytes):	1-64K	64K-256K	0.5M-32M	16G-128G	1-4T	4-16T
Costo (EURO/GB):	40000	40000	40000	10-20	0.5	0.0001

(Anno 2020)

Memoria Cache

- Scopo della cache è di memorizzare (per un rapido accesso) copie di blocchi di memoria principale
  - Funzionamento: ogni volta che faccio accesso ad una locazione di memoria, lascio in cache una copia del corrispondente blocco
  - Conseguenza: per il principio di località la cache tende a memorizzare i blocchi più usati dal processore (working set)
- Normalmente si ha:
  - B = dimensione del Blocco di cache in byte = 2^b
  - C = dimensione della Cache in byte = 2^c
  - M = dimensione della Memoria in byte = 2^m

M >> C

ovvero

$N_M >> N_C$

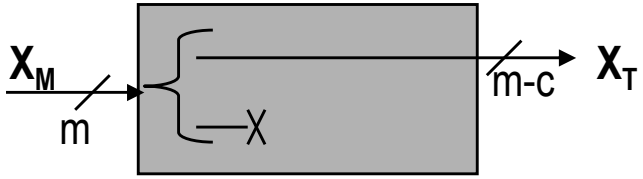
N_M = Numero di blocchi della Memoria = M/B
N_C = Numero di blocchi della Cache = C/B

Inoltre, ad un certo istante per lo working set:
$W_C \subset W_M$

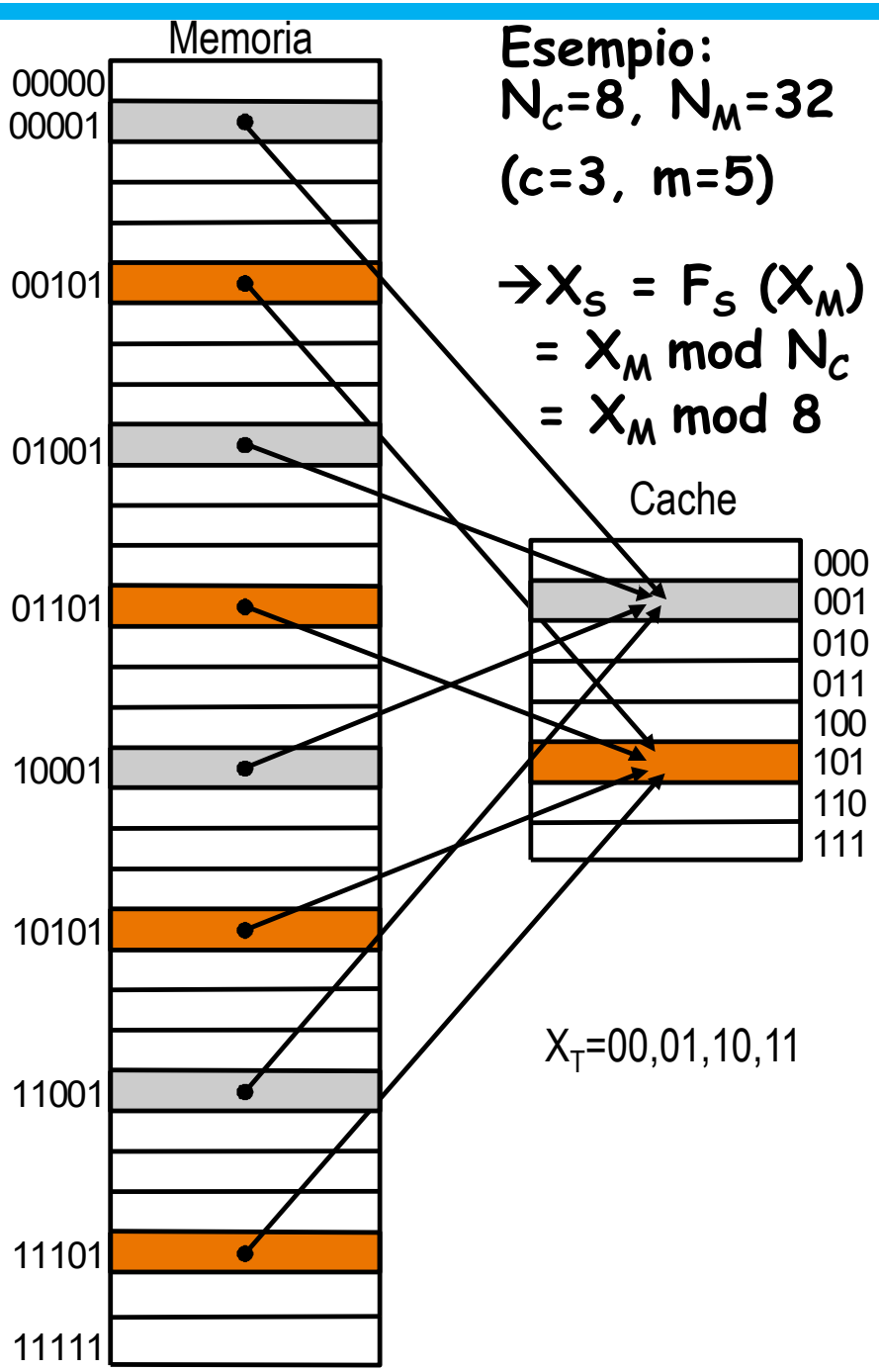
W_C = insieme dei blocchi del mio programma presenti in cache
W_M = insieme dei blocchi (totali) del mio programma in memoria

Cache ad ACCESSO DIRETTO

Equivale a fare $X_S = X_M \bmod N_C$



Equivale a fare $X_T = X_M \operatorname{div} N_C$



Schema logico di una cache ad accesso diretto

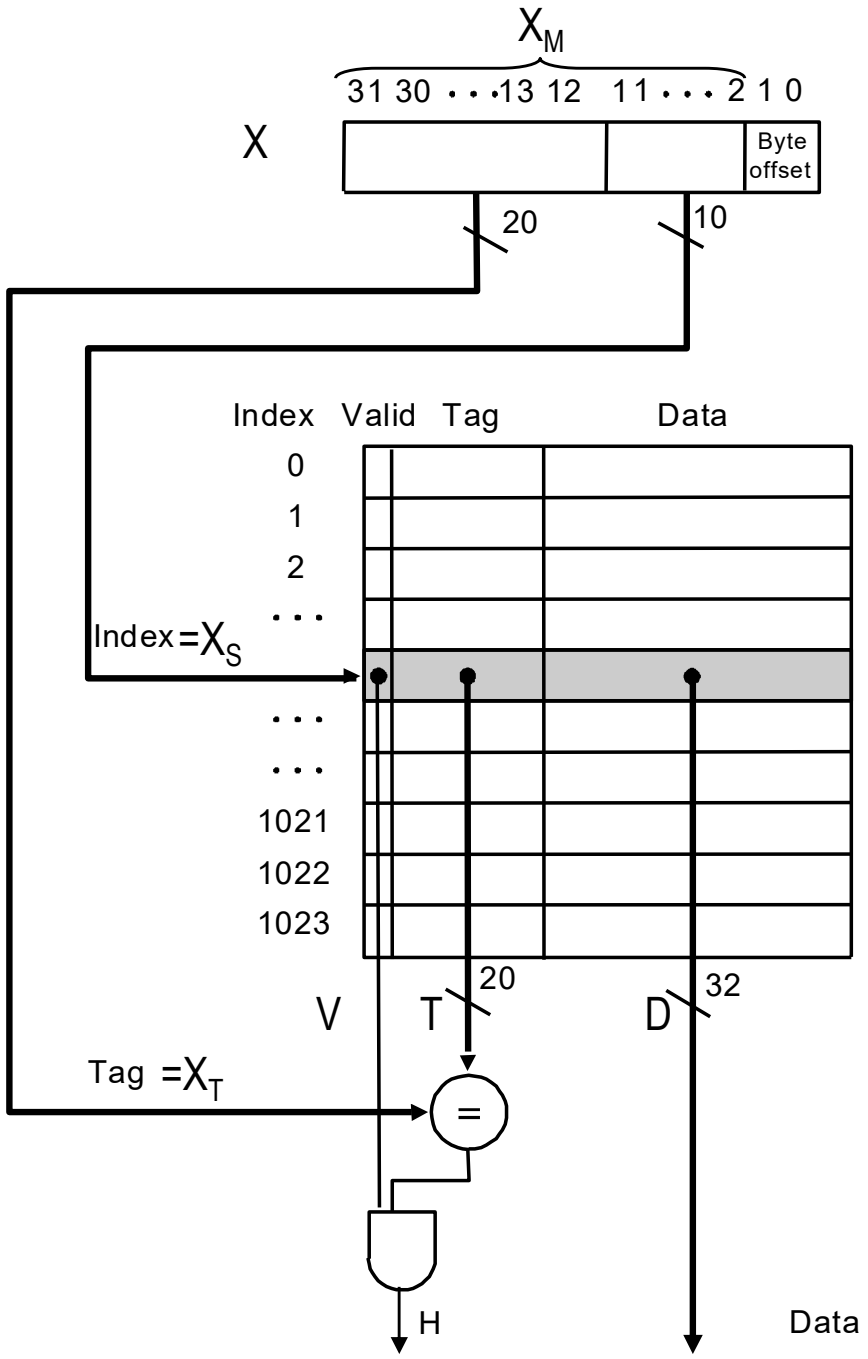
- M = 4 Gbyte
- C = 4 Kbyte
- B = 4 byte

X indirizzo al byte

$X_M = X / B$
$= X / 2^2$

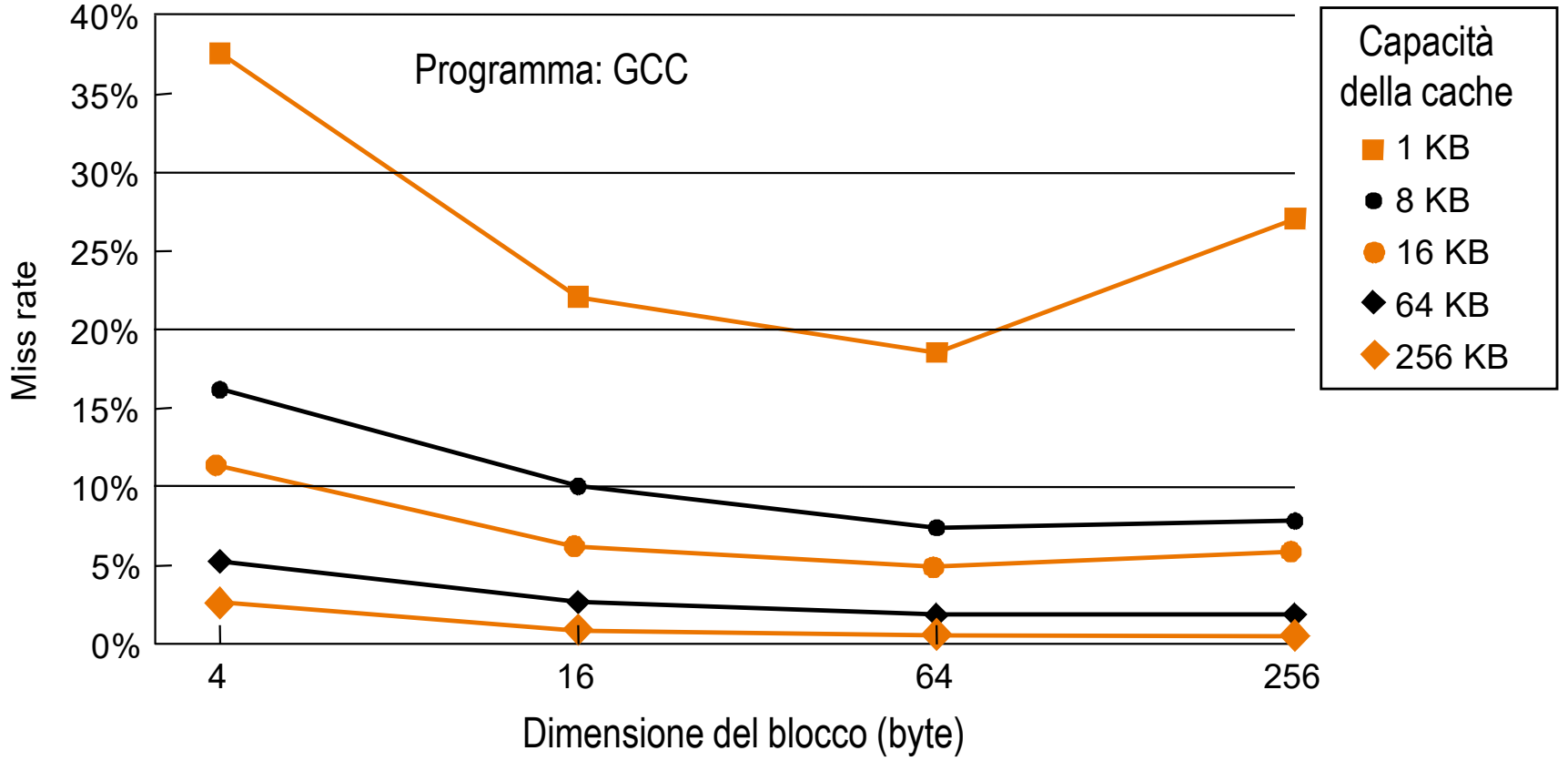
$X_S = X_M \bmod N_C$
$= X_M \bmod C/B$
$= X_M \bmod 2^{10}$

$X_T = X_M / N_C$
$= X_M / (C/B)$
$= X_M / 2^{10}$



Dipendenza del Miss Rate dalla dimensione del blocco

- **Aumentando la dimensione del blocco il miss rate tende a diminuire**



Program	Block size	Miss rate
gcc	4	5.4%
	16	1.9%
spice	4	1.2%
	16	0.4%

Cache a blocchi più grandi: sfrutto la località spaziale

- M = 4 Gbyte
- C = 64 Kbyte
- B = 16 byte

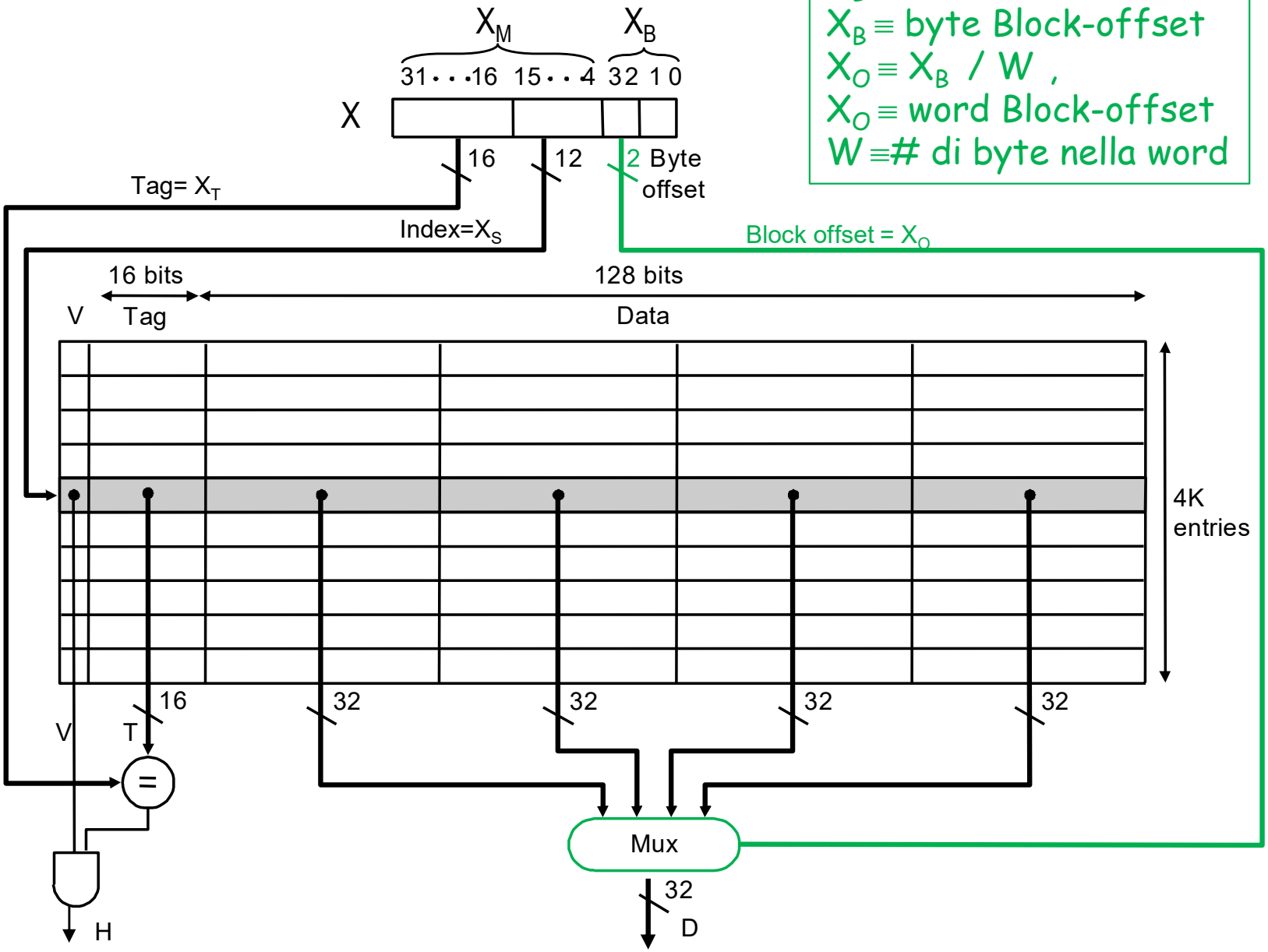
X indirizzo al byte

$X_M = X / B$
$= X / 2^4$

$X_S = X_M \bmod N_C$
$= X_M \bmod C/B$
$= X_M \bmod 2^{12}$

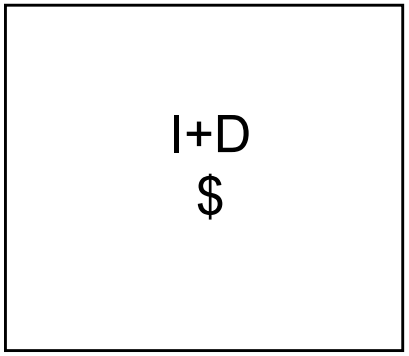
$X_T = X_M / N_C$
$= X_M / (C/B)$
$= X_M / 2^{12}$

$X_B \equiv X \bmod B$
$X_B \equiv$ byte Block-offset
$X_O \equiv X_B / W$,
$X_O \equiv$ word Block-offset
$W \equiv$ # di byte nella word

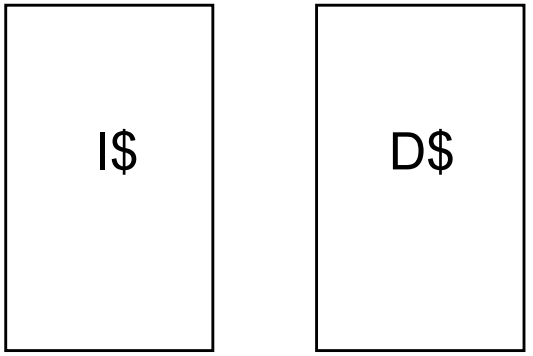


Split Cache

CACHE UNIFICATA



SPLIT CACHE



- La split cache consente di sfruttare maggiormente la località spaziale soprattutto nell'accesso ai dati

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Numero medio di riferimenti per istruzione

- OGNI ISTRUZIONE FA ACCESSO ALLA MEMORIA
  es. ADD R1, R2, R3 genera 1 riferimento (alla memoria istruzioni)
  LW R1, 0(R2) genera 2 riferimenti (1 alla memoria dati + 1 alla mem. istruzioni)

- Indichiamo quindi con
  $/R_{INST}$ = Average References per Instruction $\quad \overline{R}_{INST} \stackrel{\Delta}{=} \frac{N_{REF}}{N_{CPU}}$

- $N_{REF,i}$=numero di riferimenti generati dalla istruzione i-esima
  - $N_{REF,i1}$ =1 (es. istruz. ADD)
  - $N_{REF,i2}$ =2 (es. istruz. LW)

$$\overline{R_{INST}} = \frac{N_{REF}}{N_{CPU}} = \frac{1}{N_{CPU}} \sum_{i=1}^{N_{CPU}} N_{REF,i}$$

- Nell'esempio precedente:
  - N_{REF} = numero totale di riferimenti = 3
  - N_{CPU} = 2

  →$/R_{INST}$=1.5

Nota: la cache vede al suo ingresso RIFERIMENTI, non istruzioni !

Equazione delle prestazioni MODIFICATA

- Equazione delle prestazioni:

$$T_{CPU} = C_{CPU} \cdot T_C = N_{CPU} \cdot \overline{CPI} \cdot T_C$$

- /CPI stavolta è un "CPI-efficace" = "CPI ideale" +

  + numero cicli medi di stallo (penalty) per istruzione

$$T_{CPU} = N_{CPU} \cdot \left(\overline{CPI}_{ideal} + \frac{N_{REF}}{N_{CPU}} \cdot \overbrace{\bar{R}_{INST}}^{} \cdot m \cdot C_{pen} \right) \cdot T_C$$

$$= T_{CPU,ideal} + N_{REF} \cdot m \cdot C_{pen} \cdot T_C$$

→ Ho due modi per aumentare le prestazioni:

- Diminuire il miss rate
- Diminuire i cicli di penalty

Esempio: impatto della cache sulle prestazioni

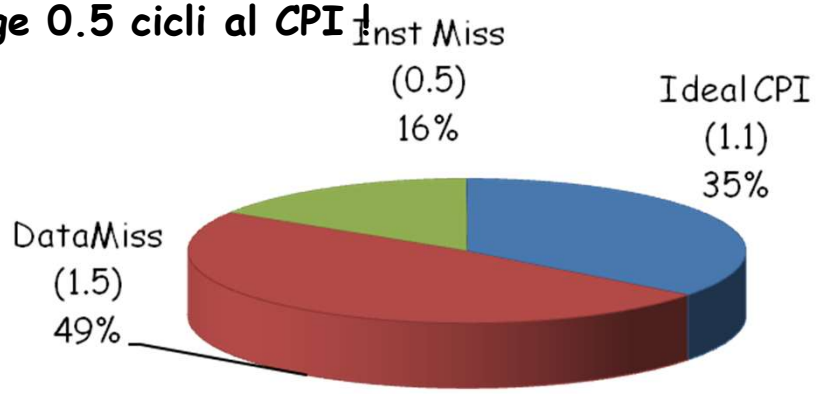
- Supponiamo di avere un processore a 200 MHz (5 ns per ciclo)
  - CPI_{ideal} = 1.1
  - 50% arit/log, 30% ld/st, 20% control
- Supponiamo che il miss-rate per le istruzioni sia l'1%
- Supponiamo che il 10% delle operazioni in memoria subisca miss con una penalty pari a a 50 cicli (di clock)
- Da (**) si ha per il CPI: $CPI = CPI_{ideal} + (m_I + N_{CPU,L/S}/N_{CPU} \times m_D) \times C_{pen}$

$$= 1.1 \text{ (cycles)} + 0.01 \text{ (miss/inst)} \times 50 \text{ (cycles/miss)} + 0.30 \text{ (dmop/inst)} \times 0.10 \text{ (miss/dmop)} \times 50 \text{ (cycles/miss)}$$

$$= 1.1 \text{ cycles} + 0.5 \text{ cycles} + 1.5 \text{ cycle} = 3.1$$

→Per il 65% (2.0/3.1) del tempo il processore attende la mem.!

→Un 1% di miss-rate per le istruzioni aggiunge 0.5 cicli al CPI!



dmop=data memory operation

Cache Associative

Patterson: 5.4

- Sullo stesso indirizzo di cache insistono più blocchi.
  → posso fare in modo di «catturarne» più di uno

Il numero di blocchi associati ad un dato indirizzo di cache è fisso per una data cache ed è chiamato:

A = "grado di Associatività" della cache o "numero di vie"

- Detti inoltre
  - *B = dimensione del Blocco di cache in byte*
  - *C = Capacità della cache in byte*

  dove:

  $A \in \{1,2,\ldots,A_{MAX}\}$

  $B = 2^b$ con $b \in \{0,1,2,\ldots,b_{MAX}\}$

  $C = 2^c$ con $c \in \{0,1,2,\ldots,c_{MAX}\}$

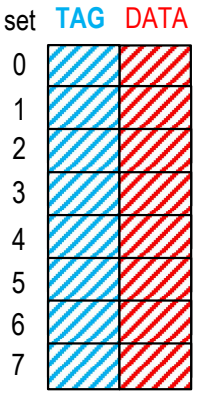
- Sussistono le seguenti relazioni

  $N_C = C / B = 2^{c-b}$ → Numero di blocchi della Cache

  $N_M = M / B = 2^{m-b}$ → Numero di blocchi della Memoria (essendo M la dimensione della memoria in byte e $m=\log_2(M)$)

Cache Set Associative

Cache ad 1 via
(direct mapped o direct access)



Cache Set-Associativa a 2 vie (A=2)

Cache Set-Associativa a 4 vie (A=4)

- Data una cache ad 1 via, con 8 blocchi, posso organizzare gli stessi blocchi in modo diverso
- Raggruppo i blocchi in insiemi o "set": il numero di blocchi in uno stesso set è detto A="numero di vie" o associatività della cache

$$N_S = C / (B \cdot A)$$

→ Numero di Set della cache

→ L'indirizzo del set sarà

$$X_S = X_M \bmod N_S$$

...e il tag

$$X_T = X_M \operatorname{div} N_S$$

Nota: come caso particolare queste relazioni valgono anche per le cache ad accesso diretto (A=1→N_S=N_C)

Cache Full Associative

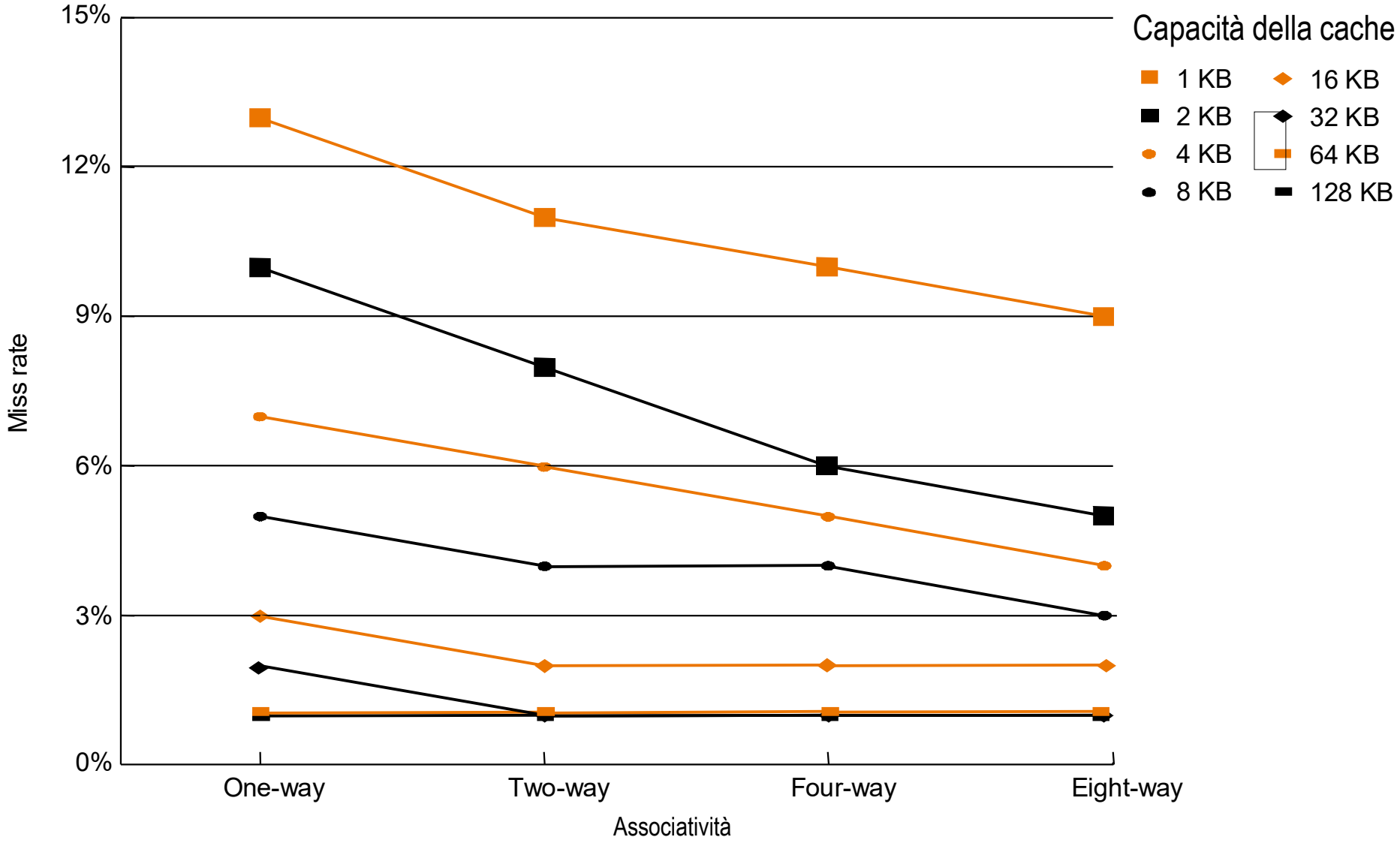
Cache Set-Associativa a 8 vie (A=8) → Completamente Associativa (o "Full Associative")

$N_S=1 \rightarrow X_T = X_M$ e $X_S = 0$ (cioè non occorre indirizzare il set)

- Come individuo il blocco all'interno dello stesso set?
  - Il problema si ha in generale in tutte le cache associative
  - Devo confrontare in parallelo il tag X_T coi tag memorizzati T_k
  - La funzione di hit sarà più complessa
    - $\forall k=1...A,\ X_T == T_k \ \&\& \ V_k == 1 \ ? \ 1 : 0$

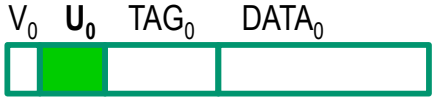
Prestazioni: Direct-access vs. Set-associative cache

- E' utile ?
  Sì: l'associatività tipicamente riduce il miss rate



Politica di Rimpiazzamento (bits U)

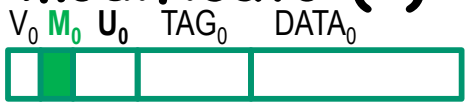
- Come si procede quando un set è pieno?
  1) Scelgo il blocco da rimpiazzare secondo una certa politica
  2) Carico il nuovo blocco nella locazione liberata
  3) Leggo o scrivo in quel blocco
- Le politiche di rimpiazzamento possono essere di svariato tipo; noi consideriamo le seguenti tre:
  - random: scelgo un blocco a caso
    - Veloce, economica, prestazioni scarse...
  - LRU (Least Recently Used): scelgo il blocco usato meno recentemente
    - Ottime prestazioni (sfrutta la località temporale), costosa da implementare (con due vie è facile tenere traccia, ma per più vie l'hardware è più complesso)
    - Spesso viene usato una versione semplificata (es. Intel usa pseudo-LRU) che ha prestazioni un poco inferiori
  - FIFO: scelgo il blocco caricato meno recentemente
    - Non molto usata
    - Ha prestazioni intermedie fra random e LRU
- Per gestire la politica di rimpiazzamento si aggiungono ulteriori bits (bits U) a fianco di ogni set



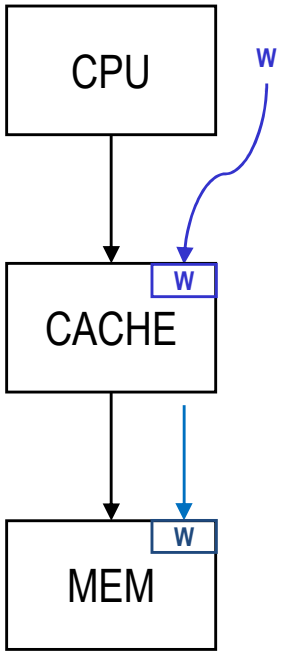
es. Per LRU occorrono $\lceil \log_2(A) \rceil$ bits U

Politica di gestione delle scritture su hit (bit M)

- Un'ottimizzazione della cache consiste nell'evitare di ricopiare in memoria un blocco rimpiazzato, se NON è stato modificato (*)
- Ho due tecniche:



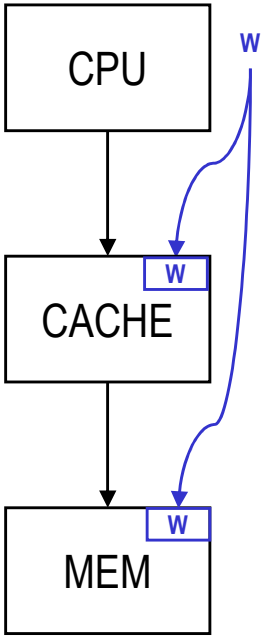
- Write-Back (WB)



(WB1) hit → scrivo SOLO in cache e metto a 1 il bit M di quel blocco (inizialmente M=0)

(WB2) In caso di rimpiazzamento, il blocco va ricopiato in memoria SOLO se M=1

- Write-Through (WT)



(WT1) hit → scrittura in cache e anche in memoria (in caso di rimpiazzamento non occorre ricopiare il blocco in memoria)

- Confronto Write-Back (WB) / Write-Through (WT):

- WB: occorre un bit M per ogni blocco (in WT il bit M non occorre)

+ WB: riduco il traffico sul bus di memoria (write), rispetto a WT

- WB: si introduce un po' di traffico di aggiornamento della memoria

(*)Nota: questa politica può essere usata anche in cache a 1 via

Schema architetturale di una cache 4-way set associative con politica di rimpiazzamento LRU/FIFO e write-back

- M = 4 Gbyte
- A = 4
- B = 64 byte
- C = 32 Kbyte

X indirizzo al byte

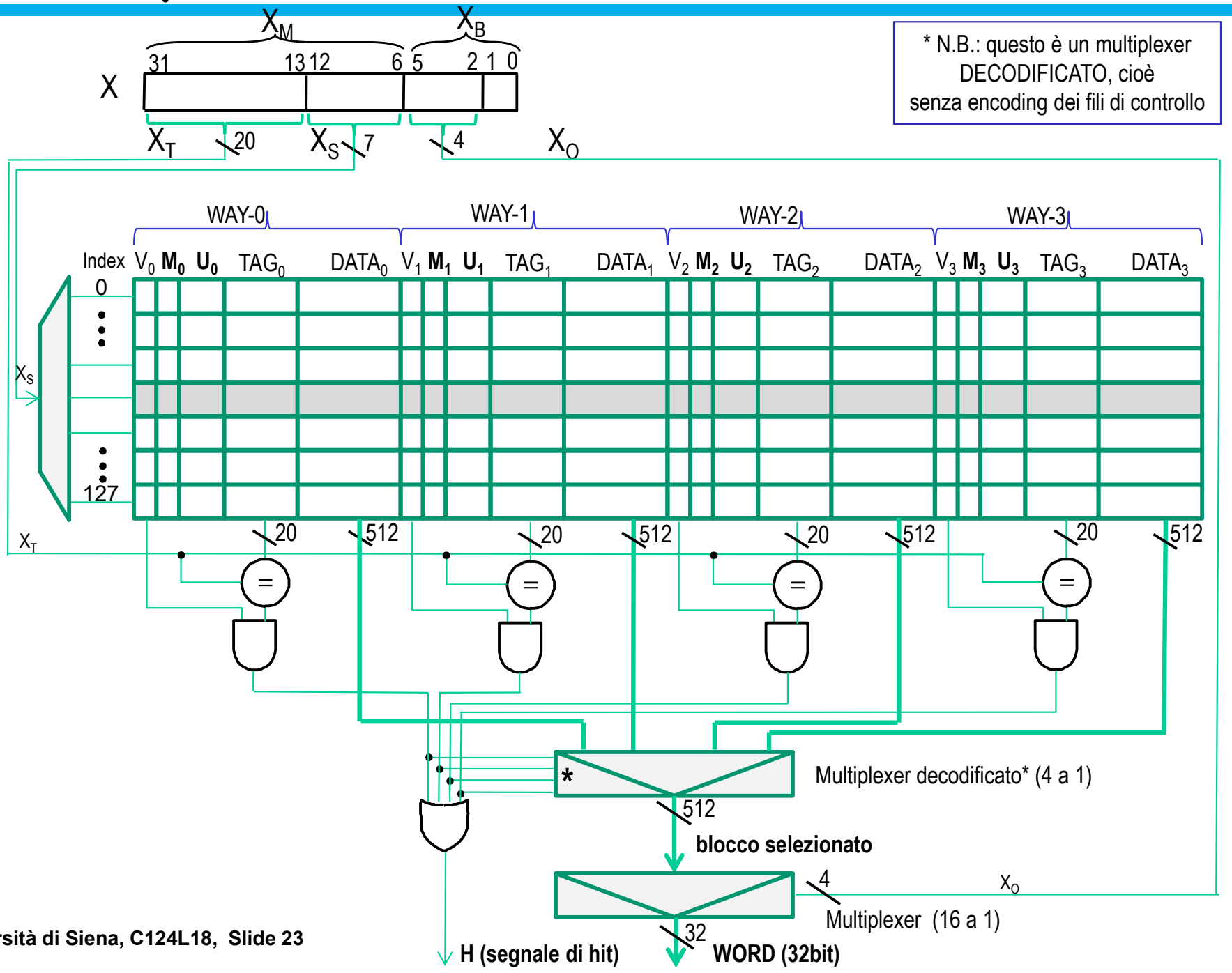
$X_M = X / B$
$= X / 64$

$N_S = C/B/A$
$= 2^{15}/2^6/2^2$
$= 2^7$

$X_S = X_M \bmod N_S$
$= X_M \bmod (C/B/A)$
$= X_M \bmod 2^7$

$X_T = X_M / N_S$
$= X_M / (C/B/A)$
$= X_M / 2^7$

X_O = indirizzo word nel blocco

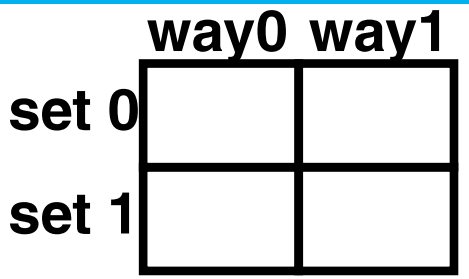


Esercizio con politica di rimpiazzamento LRU (1)

- Consideriamo una cache set-associative a due vie, con una capacità totale di 4 word e blocchi da 1 word:

  A=2
  B=4
  C=16

  $\Rightarrow N_S=2$



- Supponiamo che i riferimenti X_M (al blocco) generati dal processore siano:

  R 0
  W 6
  R 0
  R 1
  W 4
  R 0
  R 2
  R 3
  W 5
  W 4

- Quanti hit e quanti miss saranno generati se la politica di rimpiazzamento è LRU?

Esercizio con politica di rimpiazzamento LRU (2)

- Indirizzi 0, 6, 0, 1, 4, 0, ...

  X_M %2 /2 (N_S=2)

  0: (X_S=0,X_T=0) → miss, carico nel set 0 (way 0)

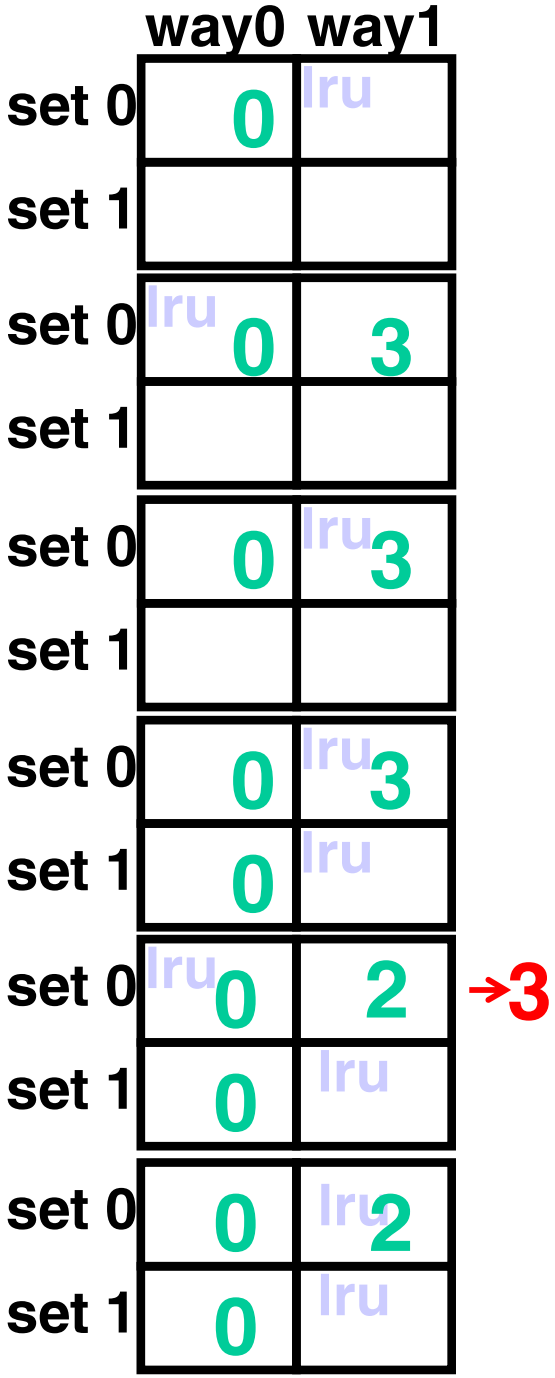
  6: (X_S=0,X_T=3) → miss, carico nel set 0 (way 1)

  0: (X_S=0,X_T=0) → hit

  1: (X_S=1,X_T=0) → miss, carico nel set 1 (way 0)

  4: (X_S=0,X_T=2) → miss, carico nel set 0 (way 1, rimpiazzo il 3)

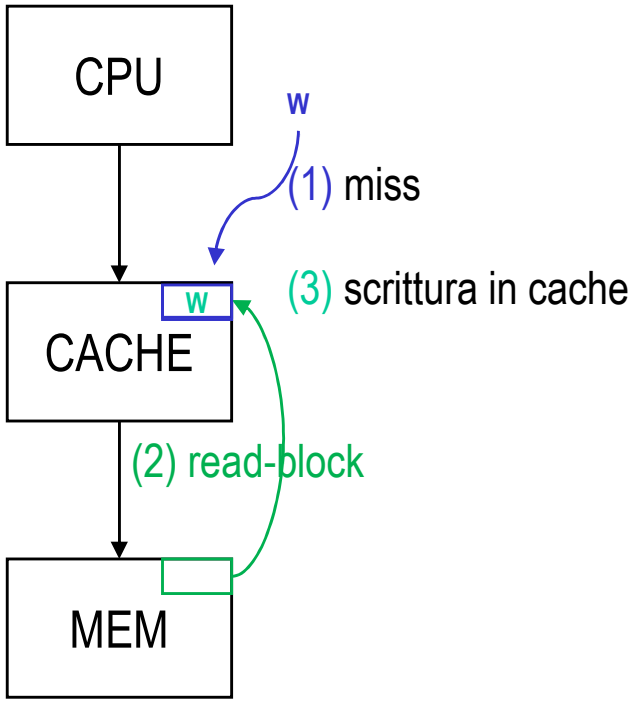
  0: (X_S=0,X_T=0) → hit



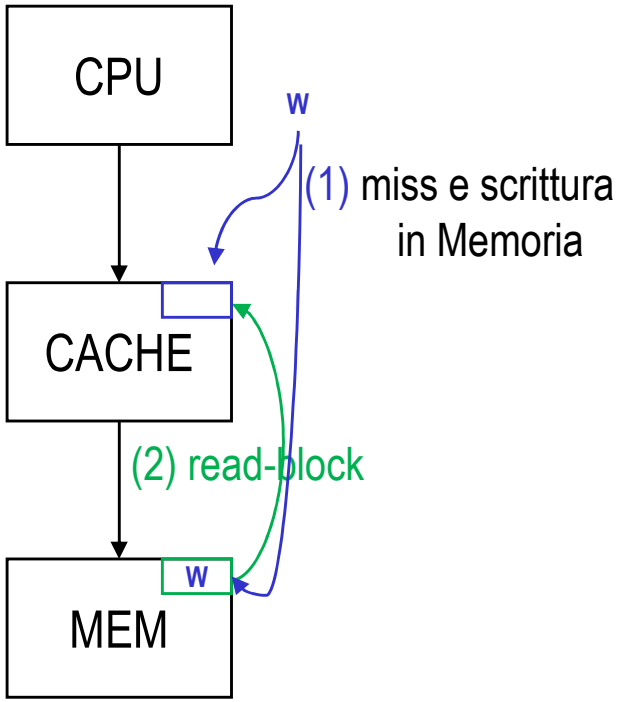
Politica di gestione delle scritture su miss: WA/WNA

- **Cosa faccio se durante una scrittura si verifica miss?**
- **Esistono due tecniche:**

- **Write Allocate (WA)**



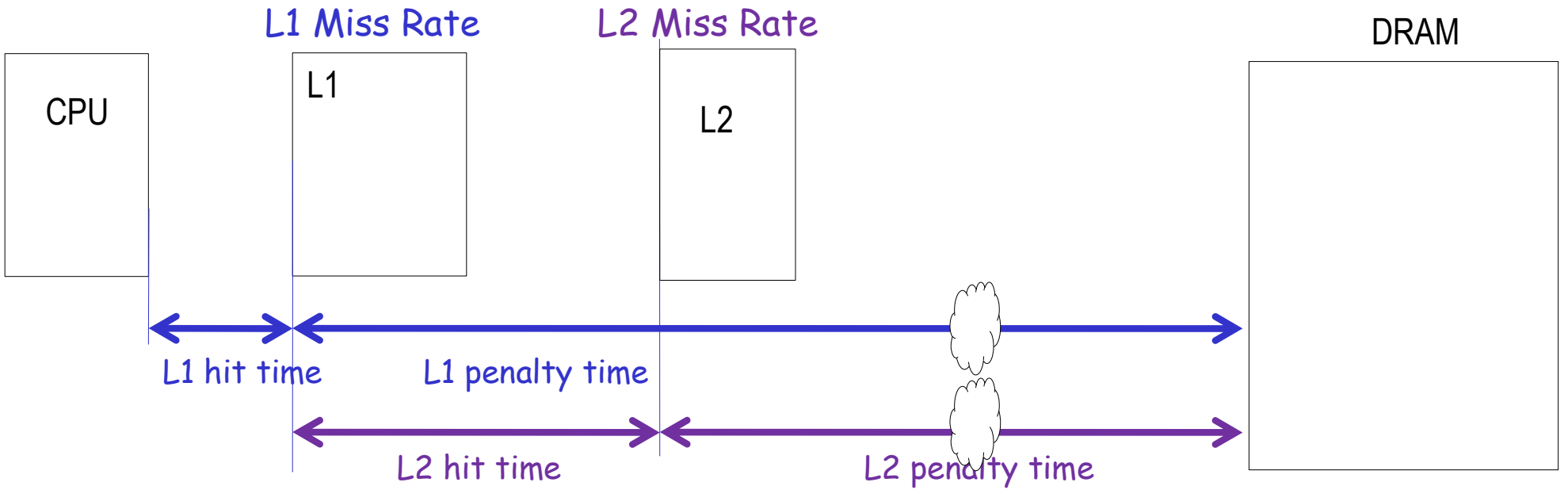
- **Write Not-Allocate (WNA)**



WA, NOTA: Le operazioni 2 e 3 vengono compiute come una singola azione
(si scrive nel blocco durante il suo caricamento)

WNA, NOTA: L'operazione 2, per scelta progettuale, può non essere implementata (ovvero NON si carica il blocco in cache): il beneficio effettivo di prelevare il blocco o meno può dipendere fortemente dal tipo di applicazione; noi assumeremo che venga compiuta.

Cache a 2 livelli



AMAT = L1 Hit Time + L1 Miss Rate * L1 Penalty Time

L1 Penalty Time = L2 Hit Time + L2 Miss Rate * L2 Penalty Time

AMAT = L1 Hit Time + L1 Miss Rate * (L2 Hit Time + L2 Miss Rate * L2 Penalty Time)

Valori tipici

- L1
  - Capacità: decine di KB
  - Hit-time: uno (o due) cicli di clock
  - Miss-rate tipici: 1-5%
- L2:
  - Capacità: centinaia di KB
  - Hit-time: qualche ciclo di clock
  - Miss-rate tipici: 10-20%

- I miss di L2 sono una frazione dei miss di L1
  - Perché allora il miss rate di L2 è così alto?

Cache a più livelli: esempio (confronto con 1 solo livello)

- Supponiamo di avere L1 e L2 con i seguenti dati:
  - L1 Hit-Time = 1 cc
  - L1 Miss-rate = 5%
  - L2 Hit-Time = 5 cc
  - L2 Miss-rate = 15% (% dei miss di L1 che fa miss in L2)
  - L2 Penalty-Time = 100 cicli

- Con 2 LIVELLI:
  - L1 Penalty Time = L2 Hit Time + L2 Miss Rate * L2 Penalty Time = 5 + 0.15 x 100 = 20 cc
  - AMAT = L1 Hit-Time + L1 Miss-Rate * L1 Penalty-Time = 1 + 0.05 x 20 = 2 cc

---

- Con 1 solo LIVELLO:
  - L1 Hit-Time = 1 cc
  - L1 Miss-rate = 5%
  - L1 Penalty-Time = 100 cc
  - AMAT = 1 + 0.05 x 100 = 6 cc

→ La cache L2 migliora il tempo medio di accesso di un fattore 3

cc= clock cycles

Contributi separati delle istruzioni generiche e L/S

$$\mathrm{T_{CPU}} = N_{\mathrm{CPU}} \cdot \left(\overline{CPI}_{ideal} + \left(\frac{N_{\mathrm{REF,I}}}{N_{\mathrm{CPU}}} \cdot m_I + \frac{N_{\mathrm{REF,D}}}{N_{\mathrm{CPU}}} \cdot m_D \right) \cdot C_{pen} \right) \cdot \mathrm{T_C}$$

$$m_I = N_{MISS\_ISTRUZIONI} \,/\, N_{REF,I}$$
$$m_D = N_{MISS\_DATI} \,/\, N_{REF,D}$$

$$(**) = N_{\mathrm{CPU}} \cdot \left(\overline{CPI}_{ideal} + \left(m_I + \frac{N_{\mathrm{CPU,L/S}}}{N_{\mathrm{CPU}}} \cdot m_D \right) \cdot C_{pen} \right) \cdot \mathrm{T_C}$$

$$N_{\mathrm{REF,I}} = N_{\mathrm{CPU}}$$
$$N_{\mathrm{REF,D}} = N_{\mathrm{CPU,L/S}}$$

$$= \mathrm{T_{CPU,ideal}} + \left(N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D \right) \cdot C_{pen} \cdot \mathrm{T_C}$$

m= miss combinato dati+istruzioni

$$= \mathrm{T_{CPU,ideal}} + N_{\mathrm{REF}} \cdot \left(\frac{N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D}{\underbrace{N_{\mathrm{REF}}}_{= m}} \right) \cdot C_{pen} \cdot \mathrm{T_C}$$

$$m = \frac{N_{\mathrm{CPU}} \cdot m_I + N_{\mathrm{CPU,L/S}} \cdot m_D}{N_{\mathrm{REF}}}$$

$$= \mathrm{T_{CPU,ideal}} + N_{REF} \cdot m \cdot C_{pen} \cdot \mathrm{T_C} = \mathrm{T_{CPU,ideal}} + N_{CPU} \cdot m \cdot \overline{R}_{INST} \cdot C_{pen} \cdot \mathrm{T_C}$$

Nota: per una split cache gli accessi avvengono in parallelo. Quindi: $m_{I,SPLIT} \neq m_I$, $m_{D,SPLIT} \neq m_D$, $m_{SPLIT} \neq m$