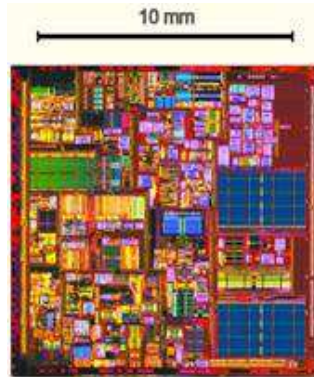


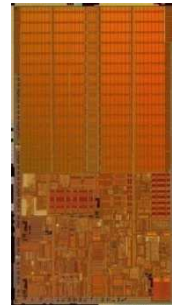
Architettura dei Calcolatori



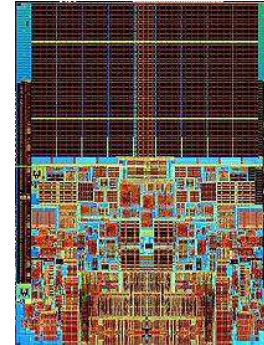
Williamette Core (0.18µm)
217 mm²/42Mtr
Intel-Pentium-4 (11/2000)



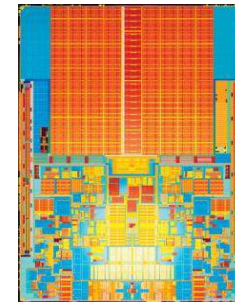
Northwood Core (0.13µm)
145 mm²/55Mtr
Intel-Pentium-4 (01/2002)



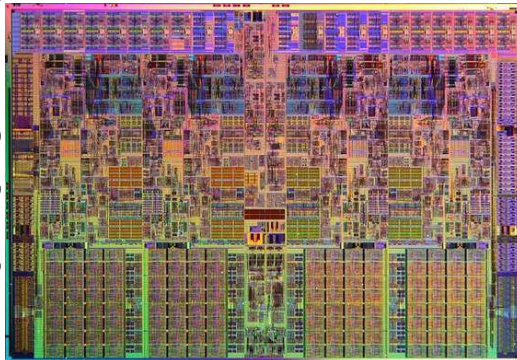
Dothan Core (0.09µm)
84 mm²/140Mtr
Intel-Pentium-M (05/2004)



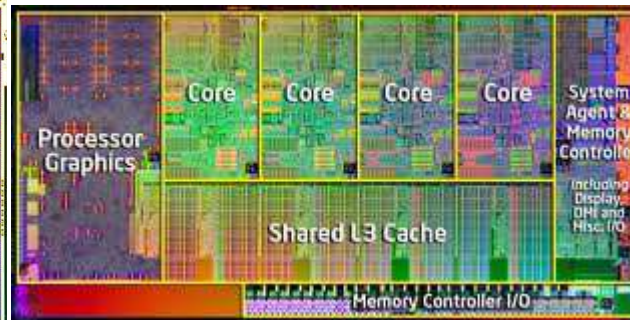
Conroe Core (0.065µm)
143 mm²/291Mtr
Intel-Core2-Duo (07/2006)



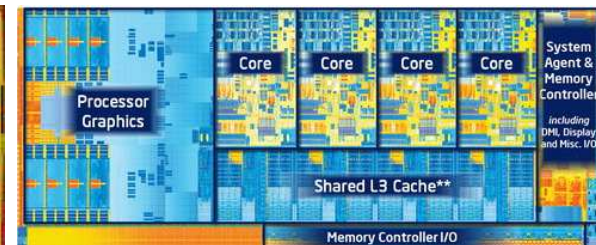
Penryn Core (0.045µm)
107 mm²/410Mtr
Intel-Core2-Duo (01/2008)



Bloomfield Core (0.045µm)
263 mm²/731Mtr
Intel-Core-i7 (11/2008)
Nehalem microarchitecture



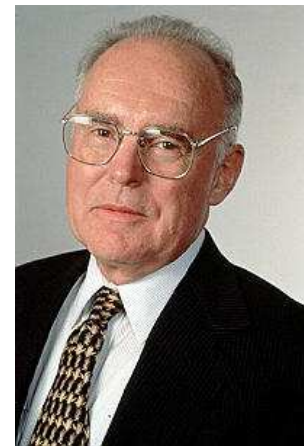
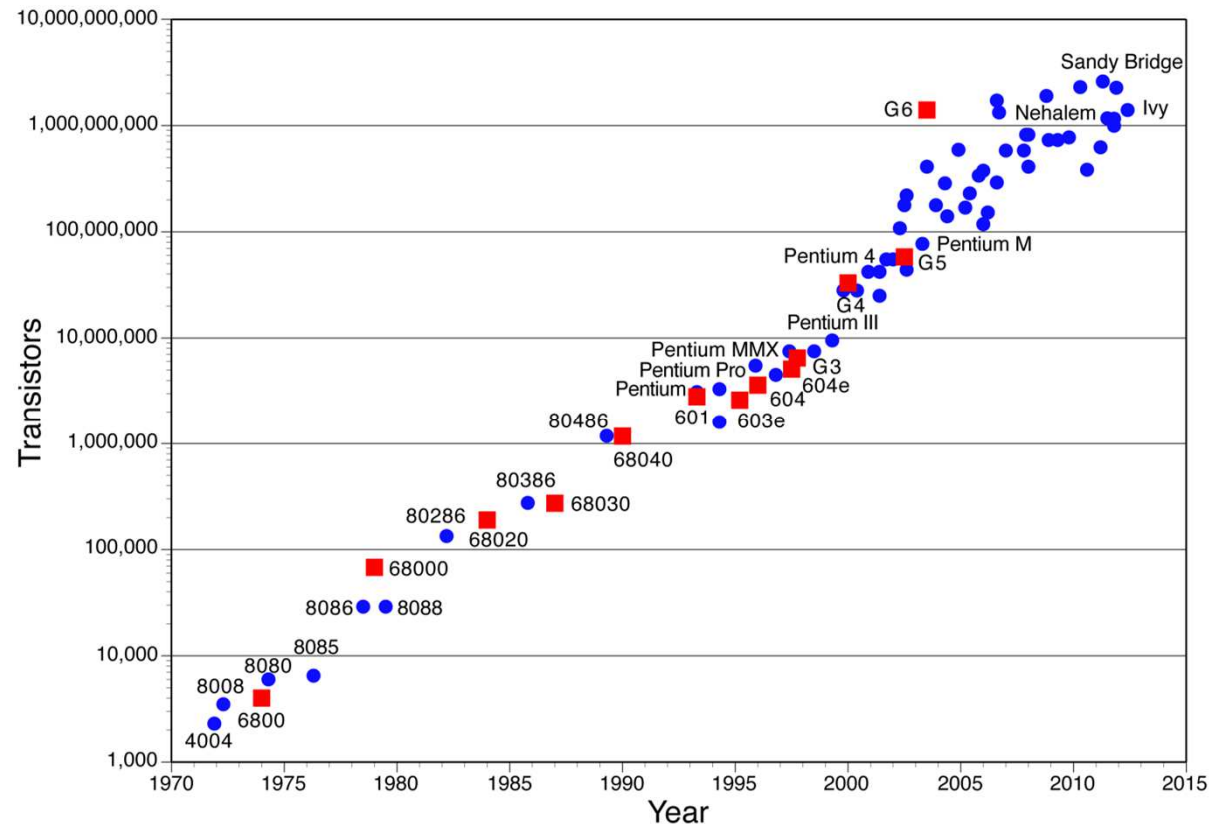
Sandy Bridge (0.032µm)
216 mm²/995Mtr
Intel-Core-i7-2920XM (01/2011)
Sandy-Bridge microarchitecture



Ivy Bridge (0.022µm) tri-gate
160 mm²/1400Mtr
Intel-Core-i73770 (04/2012)
Ivy-Bridge microarchitecture

Legge di Moore

- Il numero di transistor **RADDOPPIA** ogni 18 mesi (successivamente modificato in "24 mesi")
- Consentito sia da una maggiore densita' che da chip di maggiori dimensioni



Obiettivi del Corso e Come Raggiungerli

1) Saper scegliere un calcolatore esaminando i parametri che ne influenzano le prestazioni



Analisi quantitativa dei fattori che influenzano le prestazioni e discussione su come le architetture incidono su tali fattori

2) Capire l'architettura dei moderni calcolatori



Analisi dell'organizzazione interna dei principali elementi: processore, memoria, I/O, elementi di progettazione logica

3) Essere in grado di valutare l'efficacia dei meccanismi architetturali atti a migliorare le potenzialita' dei calcolatori



Analisi di soluzioni architetturali non necessarie ma ormai universalmente presenti quali cache, memoria virtuale, pipeline

1) Imparare a misurare e analizzare le prestazioni

- Definizione “software” della macchina
- Metriche e Benchmarks
- Influenza dei criteri prestazionali sulla definizione “software” della macchina

IPC

OOO

SPEC2000

multithreading

Tempo di esecuzione

ILP

multiprocessore

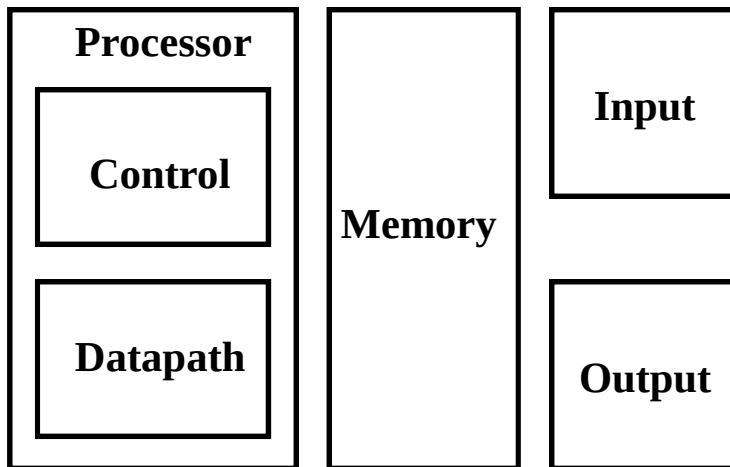
TPC

CMP

multicore

2) Analisi dell'architettura di un calcolatore

- Il processore: parte di controllo e parte datapath
- Le reti di interconnessione e interfacciamento
- Il sistema di Input/Output
- La Memoria



Dal 1946 tutti i computer hanno questi 5 componenti!

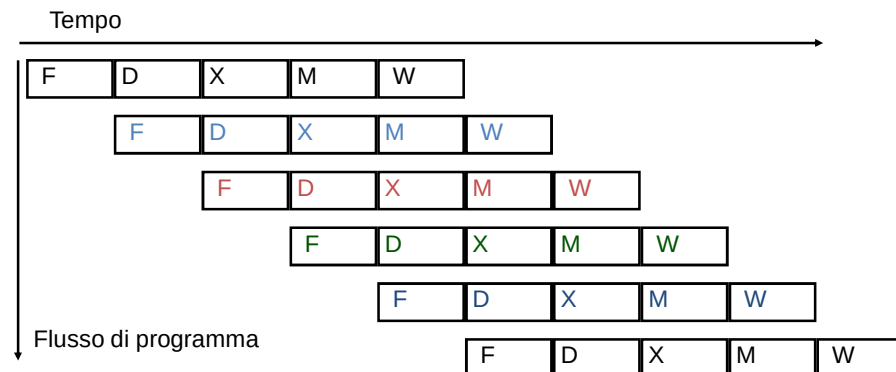
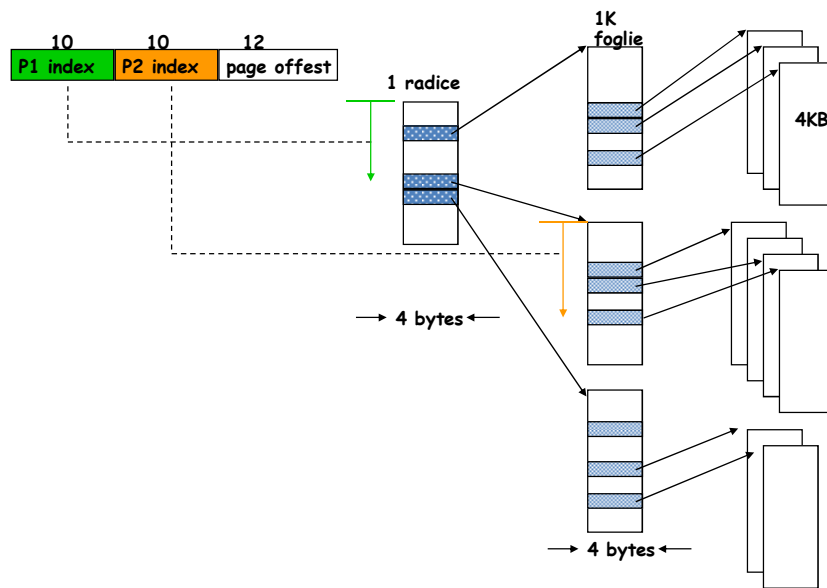
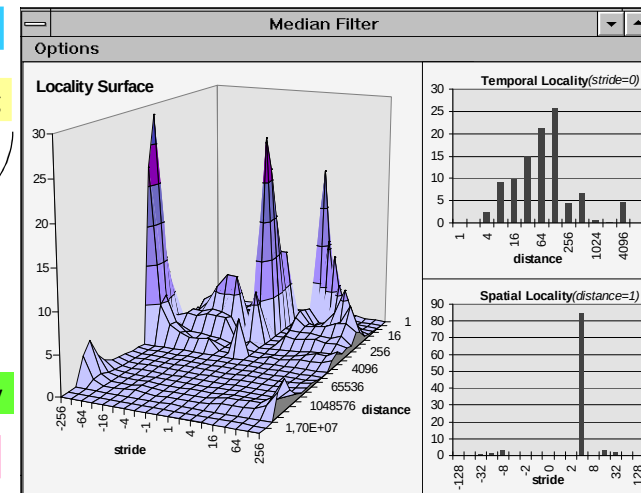
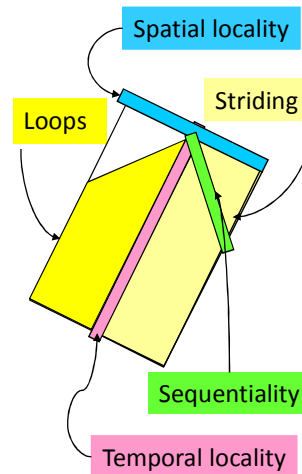
L'interno di ciascuno di questi componenti e' pero' cambiato in maniera impressionante!

- Progettazione Logica: come si passa dal dispositivo fisico, agli elementi logici ed architetturali
 - equazioni booleane, macchine a stati finiti, componenti logici

Understanding the TECHNOLOGY!!!

3) Architetture per Migliorare le Potenzialita'

- Memoria Cache
- Memoria Virtuale
- Pipeline



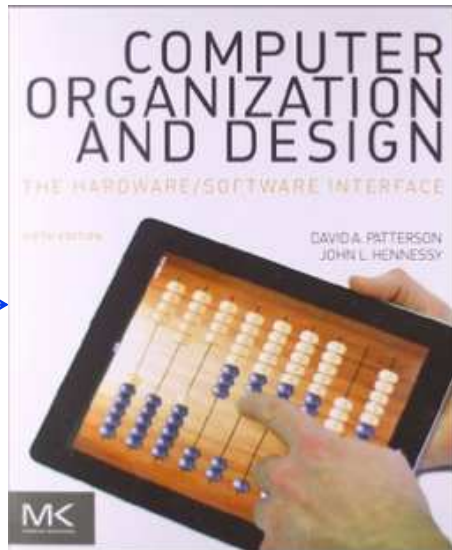
Conoscenze di base dai corsi precedenti

- Saper formalizzare un problema descritto in linguaggio naturale in modo che possa essere seguito dal calcolatore
- Saper leggere e saper scrivere semplici programmi C
- Struttura base della macchina
 - Modello di Esecuzione
ovvero principi di una macchina astratta in grado di eseguire programmi (ciclo fetch-execute, struttura di Von Neumann)
- Capire i passi che vengono compiuti per generare un programma:
 - compilazione, link, caricamento e esecuzione

Informazioni Logistiche

Amministrazione del Corso

- Docente: Roberto Giorgi (giorgi@unisi.it)
- Telefono: 0577-191-5182
- Ricevimento: Lunedì 16:30/19:00
- Dispense: <http://www.dii.unisi.it/~giorgi/didattica/arcad1>
- Testi di Riferimento:
 - Patterson and Hennessy, *Computer Organization and Design: The Hardware/Software Interface, Fifth Edition*, Morgan Kauffman, 2014
 - Quarta edizione in Italiano (basata sulla precedente ed. In inglese):
D. A. Patterson, J. L. Hennessy, *Struttura e Progetto dei Calcolatori*, Zanichelli, 2015



Formato dell'insegnamento

- **STUDIO**

- LEZIONI+ ESERCITAZIONI (totale 60 ore)
- Circa 40 ore di teoria, 20 ore di esercitazioni

- **VALUTAZIONE**

- 1 COMPITINO FINALE (65% valutazione) – 3 domande
 - MODALITA' RECUPERO COMPITINO:
 - Scritto ad un appello ordinario – 5 domande
- 1 PROGETTO (35% valutazione) – dettagli nei prossimi giorni

ALTRI INSEGNAMENTI CORRELATI

- High Performance Computer Architecture (Master Degree)

Compitino

- La data del compitino e' tentativamente il
 - Giovedì 18-12-2015 ore 14/16

QUESTE INFORMAZIONI SONO TUTTE
RIPORTATE ANCHE SUL SITO DEL CORSO
<http://www.ing.unisi.it/~giorgi/didattica/arcal1>

Organizzazione della lezione di oggi

- Tipologie di calcolatori
- Definizione di Architettura di un Calcolatore
- Introduzione al processore MIPS

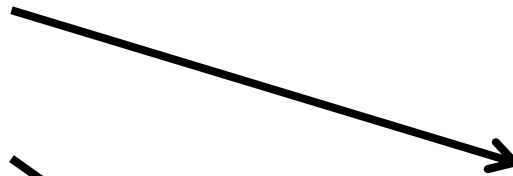
Tipologie di Calcolatori

I 3 tipi principali di calcolatori oggi

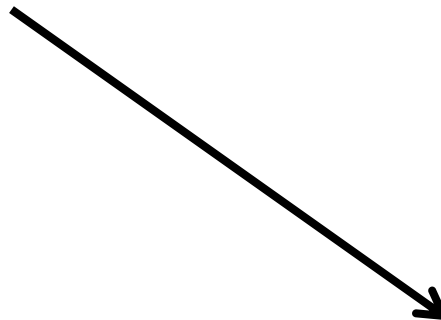
Embedded



Desktop

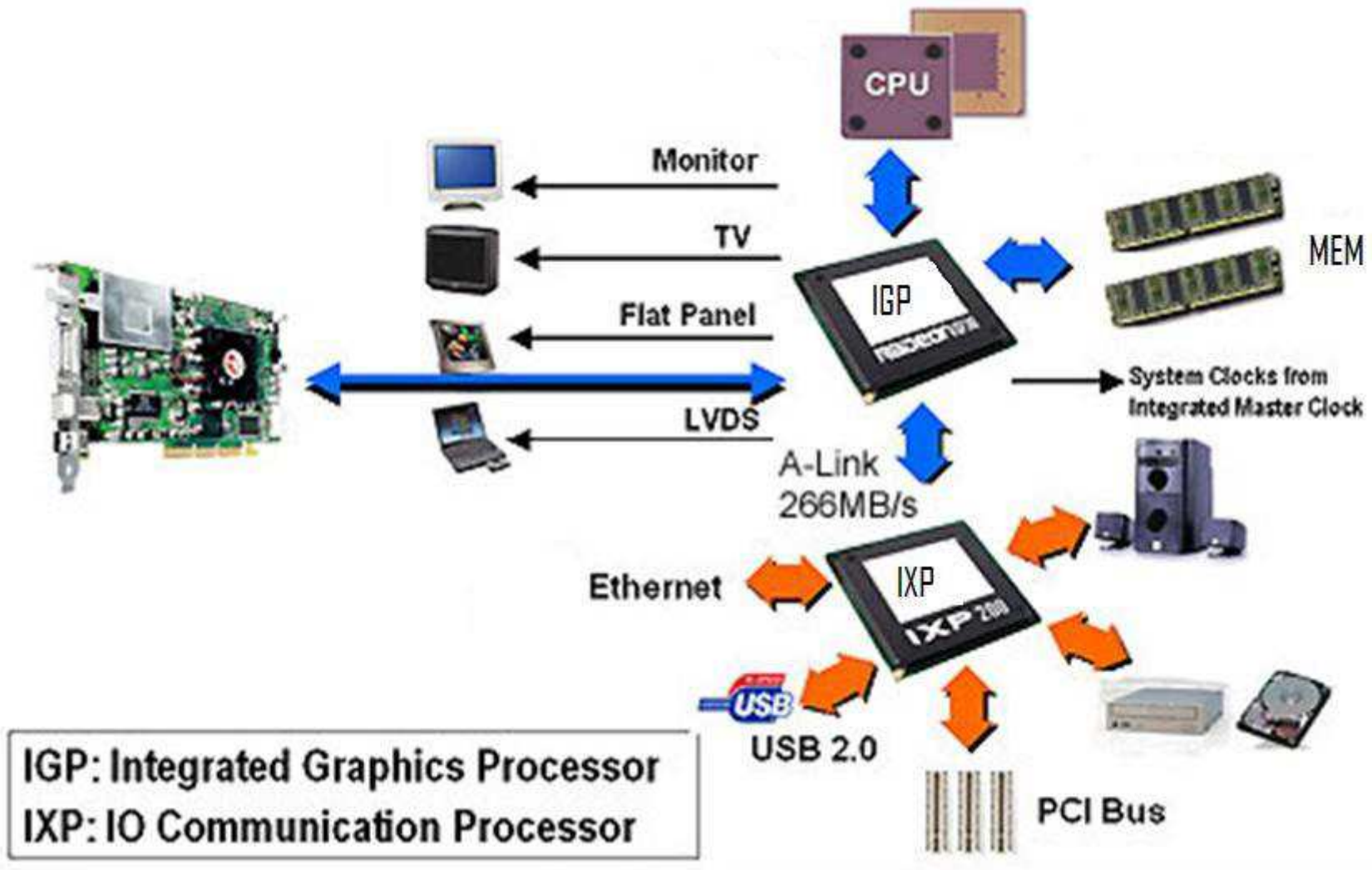


Server

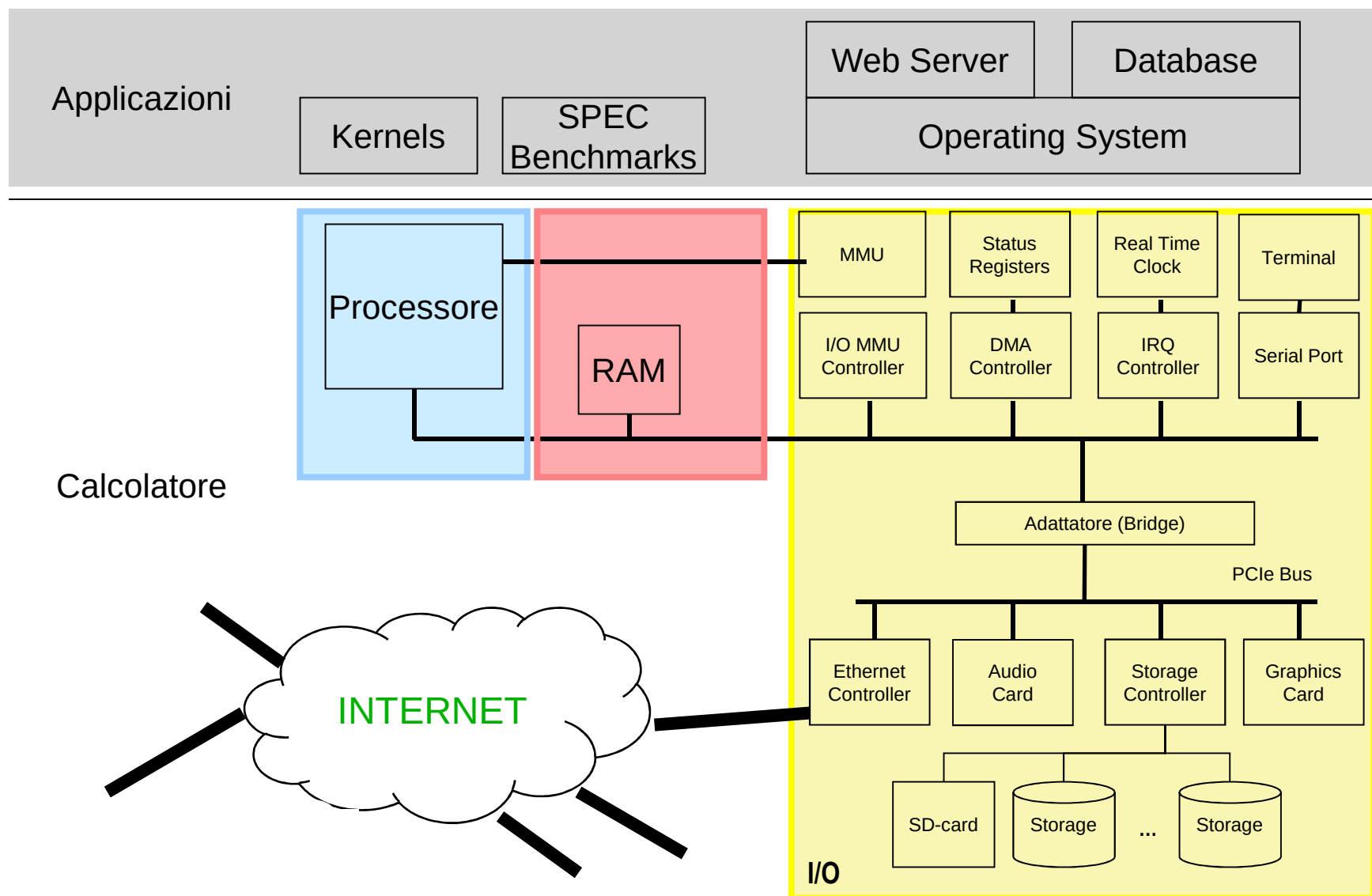


- Calcolatori 'Desktop'/'Embedded'
→ Architettura dei Calcolatori
- Calcolatori 'Server'/'Desktop'
→ High Performance
Computer Architecture

Parti principali di un calcolatore 'desktop'

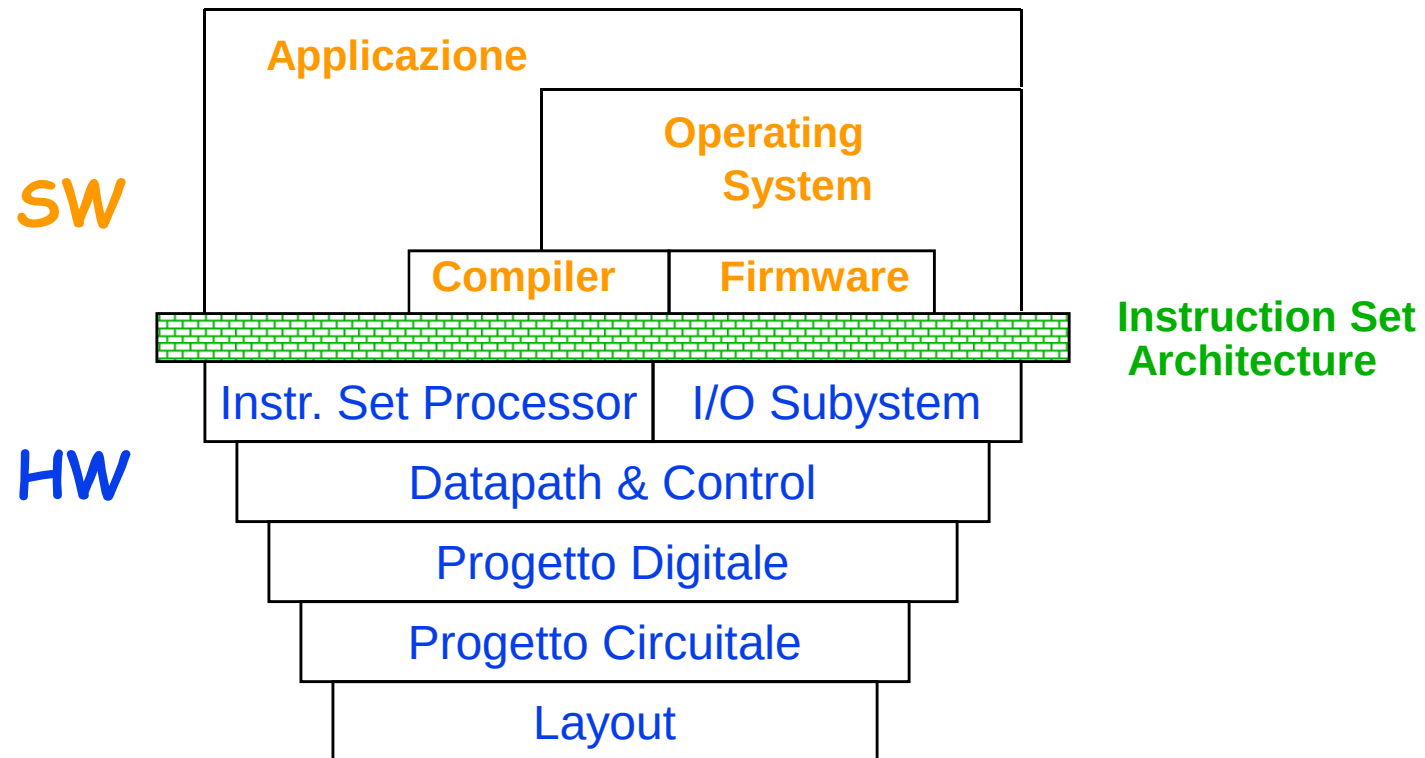


Contesto di lavoro di un Calcolatore



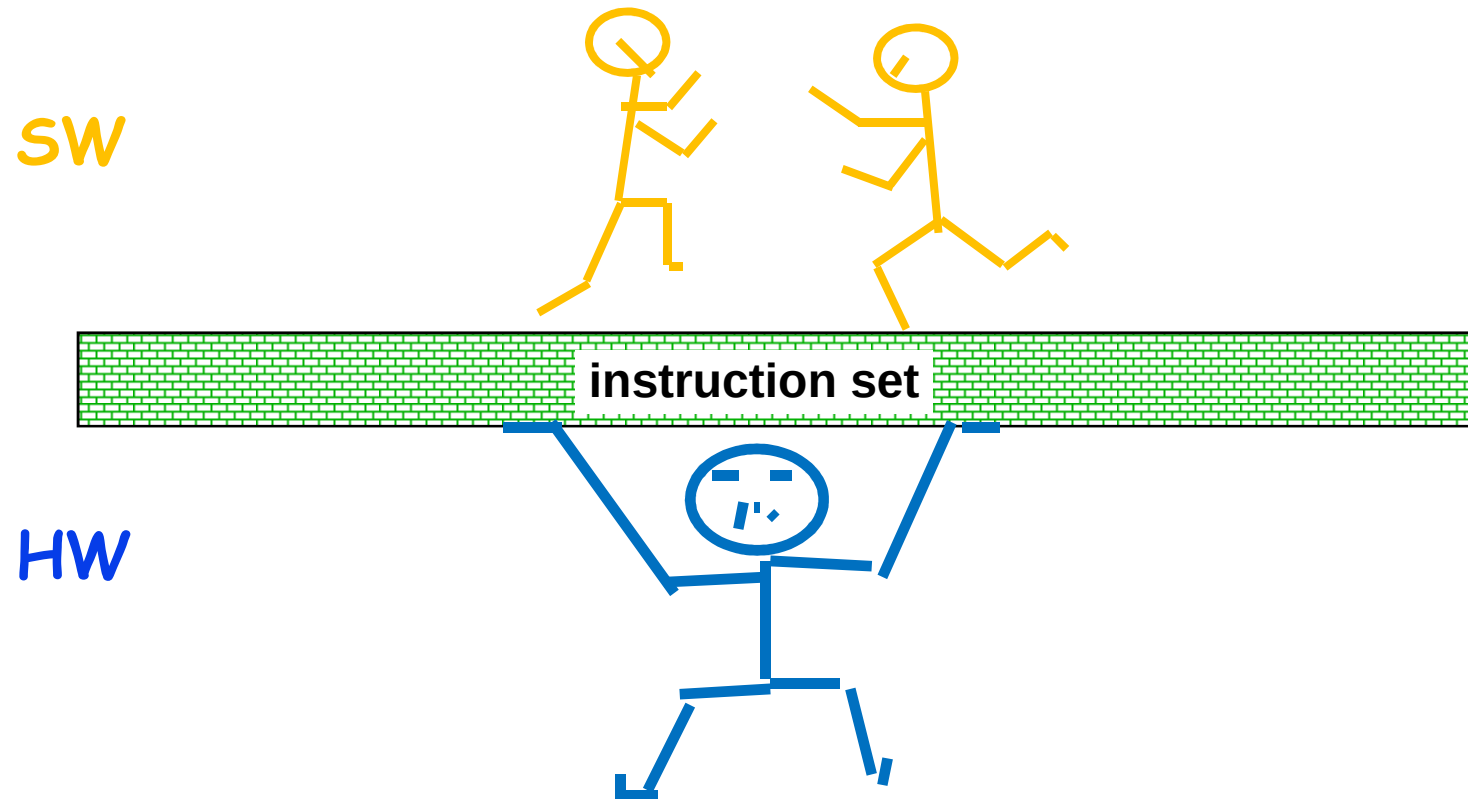
- **Tutti questi componenti hanno**
 - una propria interfaccia e
 - una propria organizzazione interna

Livelli di Astrazione di un Calcolatore

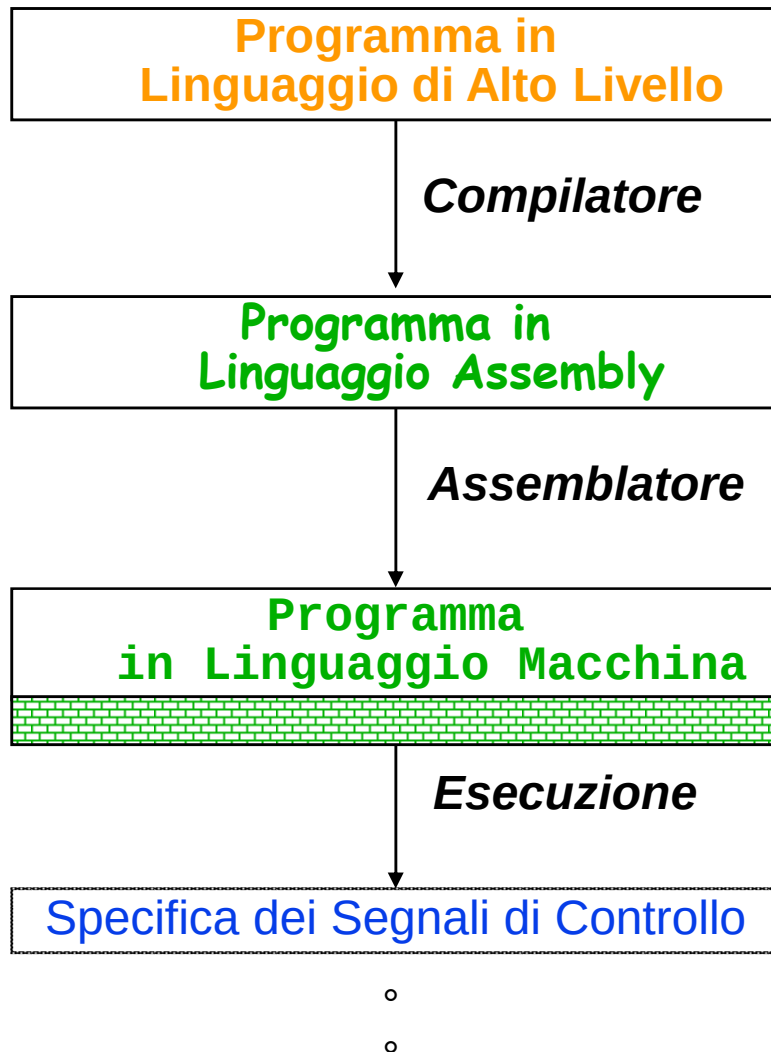


- Coordinamento fra molti *livelli di astrazione*
- Cambiamenti tecnologici che avvengono *ogni giorno*
- Ciclo virtuoso: progettazione→simulazione→valutazione

Il Set di Istruzioni: un'interfaccia critica



Livelli di rappresentazione



```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
lw $15, 0($2)  
lw $16, 4($2)  
sw $16, 0($2)  
sw $15, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

Architettura dei Calcolatori

Architettura dei Calcolatori

- “Il termine ARCHITETTURA e' usato qui per descrivere l'insieme degli attributi di un sistema, cosi' come questo appare al programmatore*, ovvero la sua struttura concettuale e il suo funzionamento, con una distinzione dell'organizzazione delle reti che gestiscono il flusso dei dati e delle reti di controllo, rispetto alla progettazione logica e rispetto all'implementazione fisica”



-- Gene Amdahl, IBM Journal of R&D, Apr. 1964

Una definizione piu' “moderna” (da <http://www.cs.wisc.edu/~arch/www/>)

“Con Architettura dei Calcolatori si intende la scienza e l'arte di selezionare e collegare componenti hardware per creare calcolatori che soddisfano determinati requisiti funzionali, di prestazioni e di costo.

L'architettura dei calcolatori non ha niente a che vedere con l'uso dei calcolatori per progettare edifici.”

*programmatore ==
programmatore di sistema (operativo)
o ingegnere del compilatore

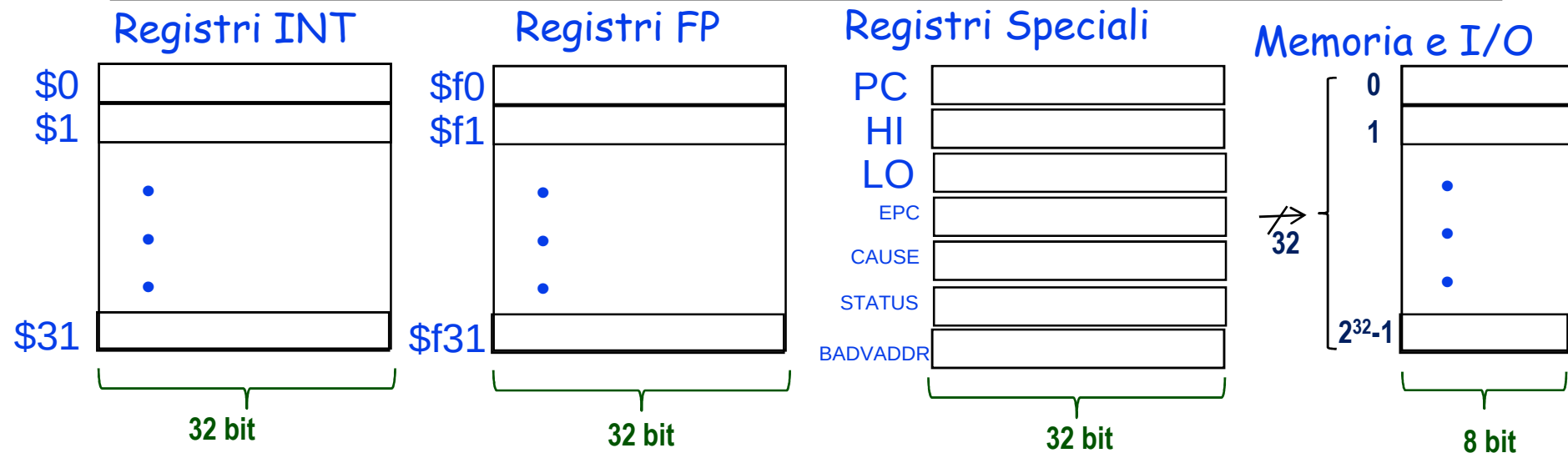
Cosa e' la "Architettura di un Calcolatore"

Computer Architecture =

**Instruction Set Architecture +
Machine Organization**

- **Instruction Set Architecture:**
 - Come l'Hardware viene visto dal Software
- **Machine Organization:**
 - Implementazione fisica
(in termini di blocchi funzionali e logici)

Instruction Set Architecture (es. MIPS 3000)



- **Categorie di Istruzioni**
 - Load/Store
 - Calcolo (per Interi e per Frazionari)
 - Jump/Branch
 - Istruzioni speciali
- **Formato delle Istruzioni:**
 - lunghezza FISSA a 32 bit → CPU a 32 bit

Organizzazione fisica del Calcolatore

- Caratteristiche e Prestazioni delle principali Unità Funzionali
 - (e.g., Registers, ALU, Shifters, Logic Units, ...)
- Metodi con cui tali componenti sono interconnessi
- Tipo di informazione che viene scambiata fra i componenti
- Logica e Metodi per controllare tale scambio di informazioni
- Insieme delle Unità Funzionali e loro interfacciamento
- Descrizione in termini di linguaggio Register Transfer Level (RTL), ove necessario

Lezione 90

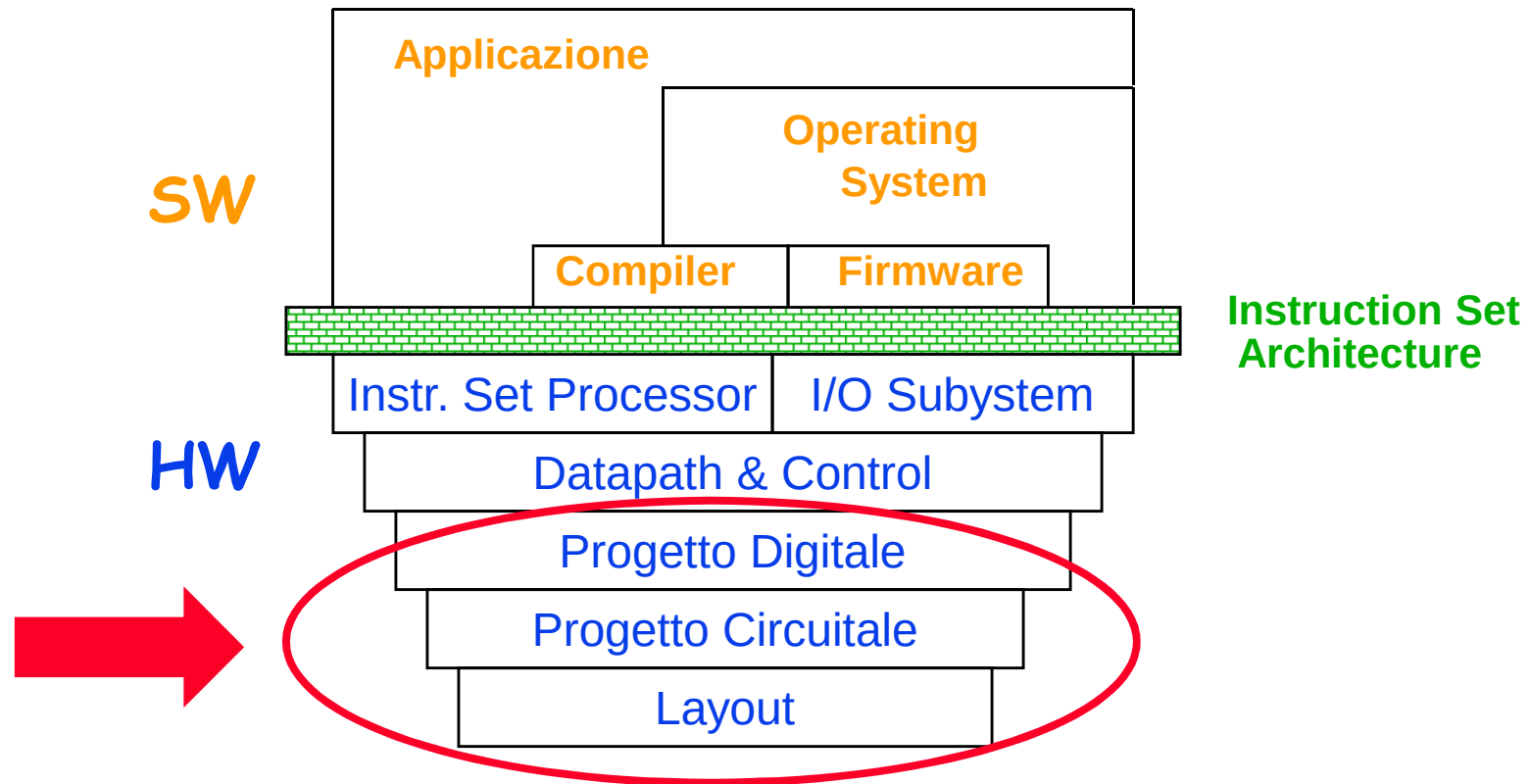
Elementi di Progettazione Digitale

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Progettazione Logica

Livelli di Astrazione di un Calcolatore



- Come si passa dal progetto digitale al layout ?
- Quali possibilita' offre l'attuale tecnologia ?

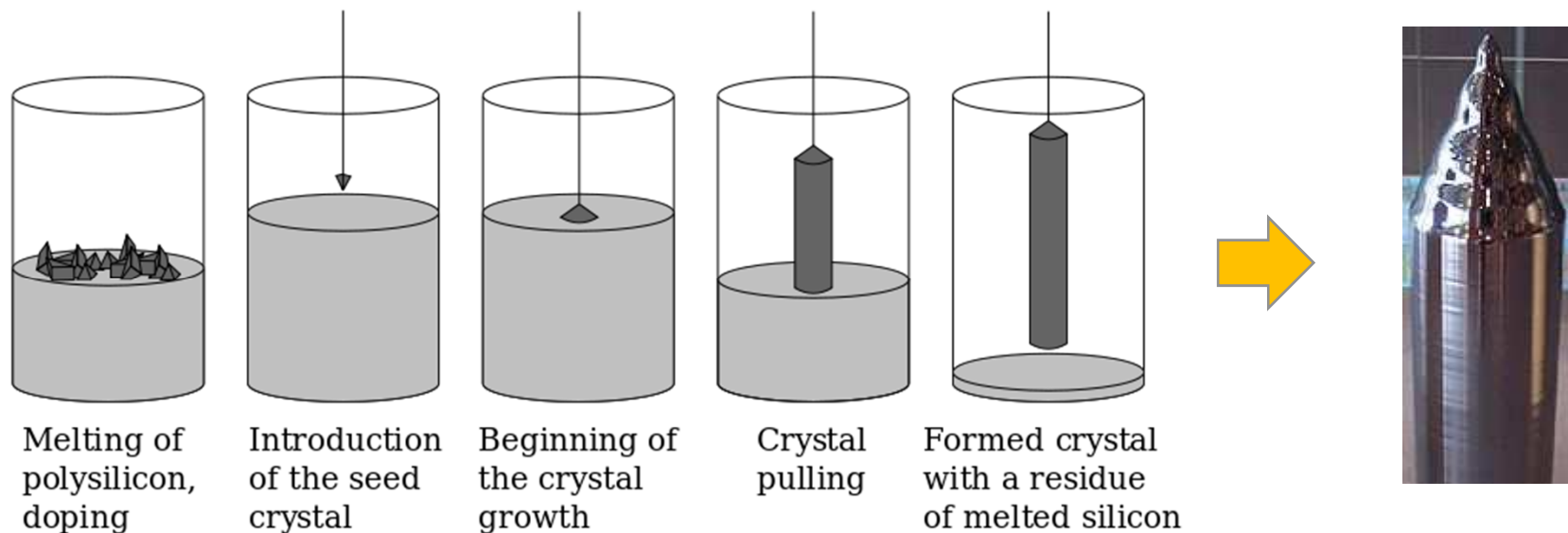
Approccio Bottom-Up: dal livello fisico all'architettura

- Preparazione del wafer di silicio
- Generazione dei dispositivi sul wafer
- Caratteristiche dell'inverter CMOS
- Altre porte: NAND, NOR, Transmission Gate

Il processo e' legato alla tecnologia:
nel periodo 1980-2015 questa e'
la metodologia consolidata

Preparazione del wafer di silicio

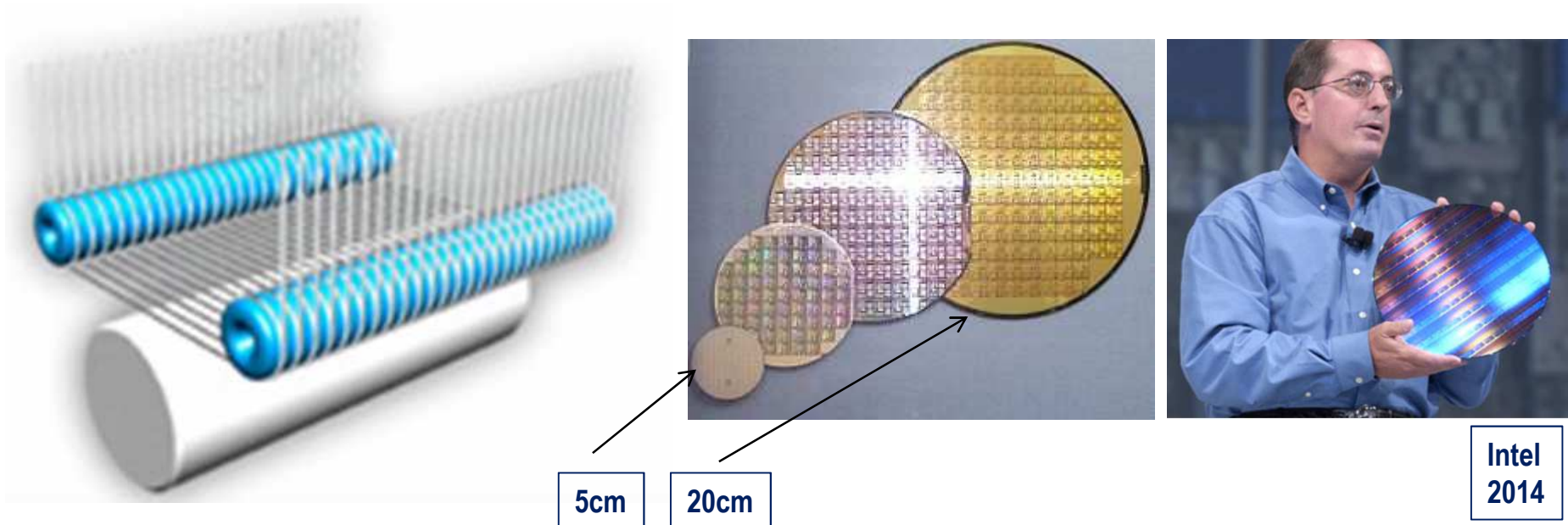
- Produzione di un lingotto di silicio monocristallino (es. Metodo di crescita di Czochralski) da silicio grezzo (quarzite)
- $\text{SiO}_2 + \text{C} \rightarrow \text{Si} + \text{CO}_2$ (in forno a 1500 C, purezza 98%)
- $\text{SiHCl}_3 + \text{H}_2 \rightarrow \text{Si} + 3\text{HCl}$ (silicio purissimo 99.9999999%)



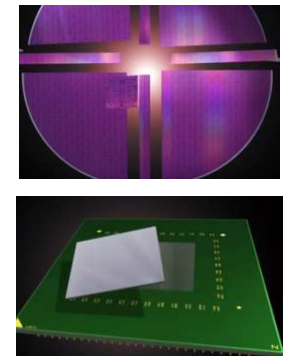
- Il diametro del cilindro così generato variano fra 10 cm e 45 cm circa

Dal lingotto al wafer

- Il lingotto viene sezionato in cosiddetti "wafer" ovvero dischi di spessore tra i 270 μm a 920 μm circa

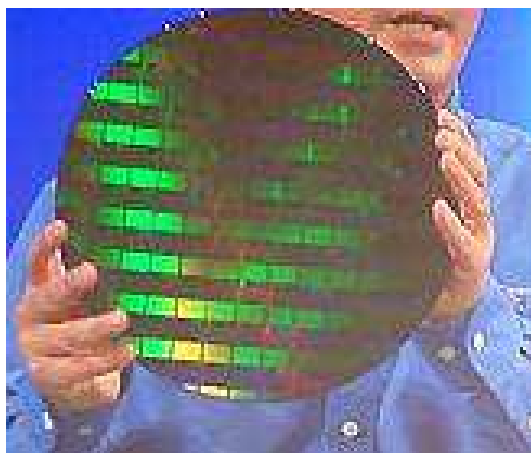


- Il wafer viene successivamente sottoposto a lappatura (etching), lucidatura e attacco con acido per ridurre imperfezioni e rugosità
- Il wafer viene successivamente sottoposto a lunghi processi per generare i dispositivi (transistor CMOS) e successivamente i chip vengono tagliati, verificati e inseriti nei package

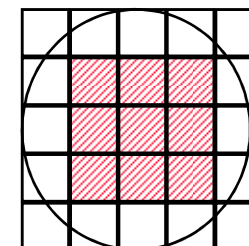
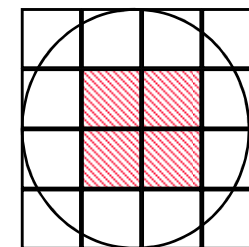


Perche' non si fanno chip "grossi"

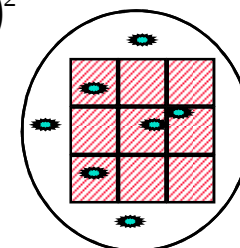
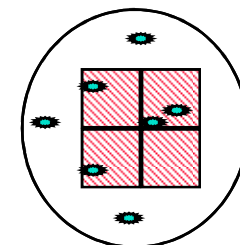
$$C_{DIE} = \frac{C_{WAFER}}{N_{DIE} \cdot Y_{WAFER}}$$



C_{DIE} = Costo del 'die' (del 'chip')
 C_{WAFER} = Costo del wafer
 N_{DIE} = Numero di 'die' in un wafer
 Y_{WAFER} = 'Yield' o Resa del wafer
 (numero 'die' per unita' di sup.)
 D_{WAFER} = Diametro del wafer
 A_{DIE} = Area del 'die'
 A_{WAFER} = Area del wafer = $\pi \cdot (D_{WAFER}/2)^2$
 N_{TEST} = Numero di 'die' usati per test
 F = Difetti per unita' di superficie



N. di chip "teorici" sul wafer
 N. di chip sul bordo del wafer
 N. di chip usati per test "distruttivi"



$$N_{DIE} = \frac{\pi \cdot \left(D_{WAFER}/2\right)^2}{A_{DIE}} - \frac{\pi \cdot D_{WAFER}}{\sqrt{2} \cdot A_{DIE}} - N_{TEST} \approx \frac{A_{WAFER}}{A_{DIE}}$$

Inoltre: $Y_{WAFER} = \frac{1}{1 + (F \cdot A_{DIE}/2)^2}$

$$\rightarrow C_{DIE} = \frac{C_{WAFER} \cdot A_{DIE} \cdot \left[1 + (F \cdot A_{DIE}/2)^2\right]}{A_{WAFER}} \propto \frac{A_{DIE}^3}{A_{WAFER}}$$

Il costo di un chip e' proporzionale a circa il cubo della sua area!

Generazione dei dispositivi sul wafer

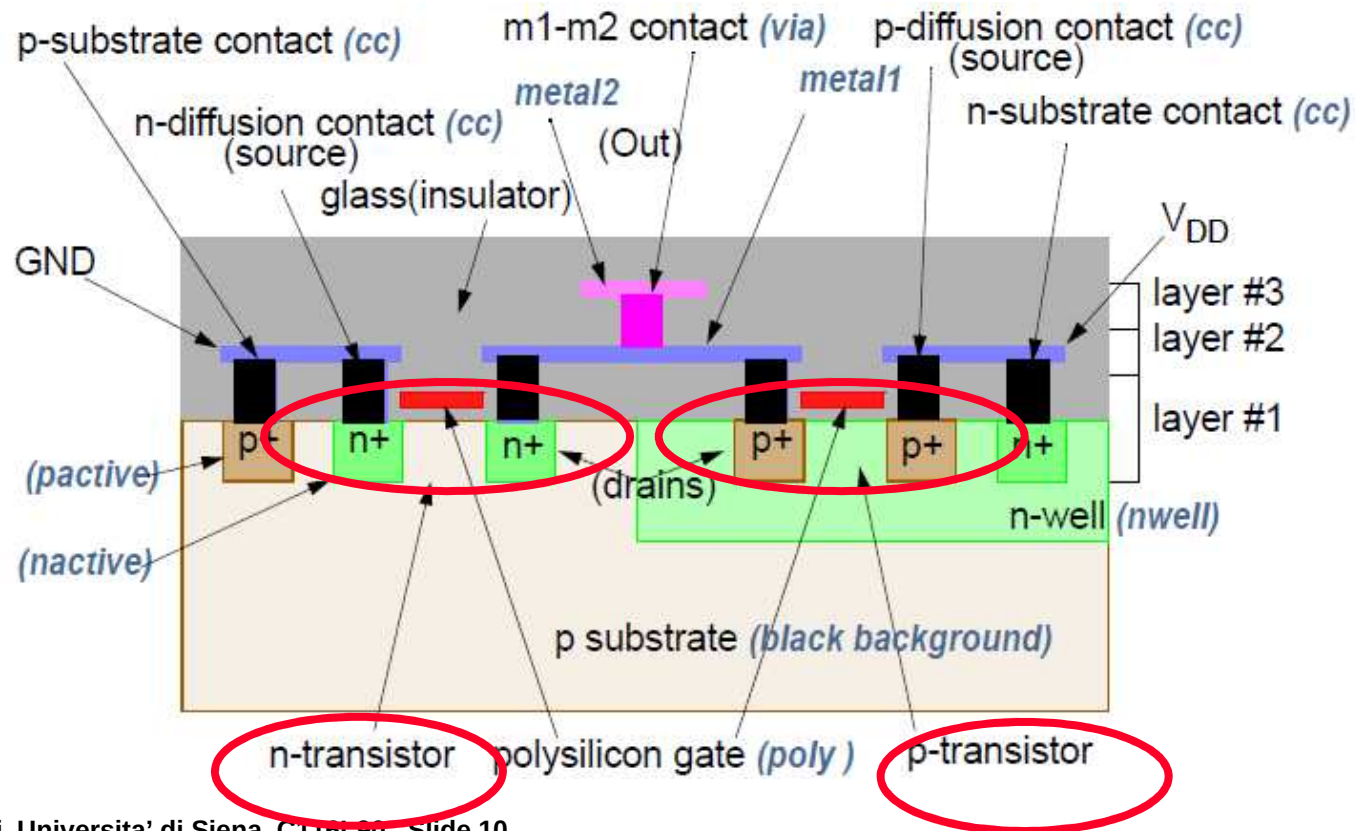
- Nel caso di chip usati per sistemi digitali (es. CPU), i dispositivi generati sul wafer consistono quasi esclusivamente in transistor e collegamenti
 - I transistor vengono realizzati con la tecnologia "MOS" (Metal Oxide Semiconductor)
 - I collegamenti possono essere realizzati sia con metallo che con polisilicio
- I transistor sono principalmente di due categorie:
 - Transistor di logica
 - Transistor di passo
- Inoltre per rendere il funzionamento piu' efficiente viene adottata la tecnologia CMOS (Complementary MOS) che sfrutta la possibilita' di abbinare transistor "di tipo N" (NMOS) a transistor "di tipo P" (PMOS)

MOS: Metal Oxide Semiconductor

- I transistor sono generati sul substrato di Silicio (semiconduttore)
- Il silicio puro non ha portatori di carica liberi ed e' un cattivo conduttore
- Il processo di drogaggio aumenta la conducibilita':
 - Con diffusioni di tipo 'n' (es. Fosforo) si dispone di elettroni in eccesso (n-well)
 - Con diffusioni di tipo 'p' (es. Boro) si dispone di lacune in eccesso (p-well)

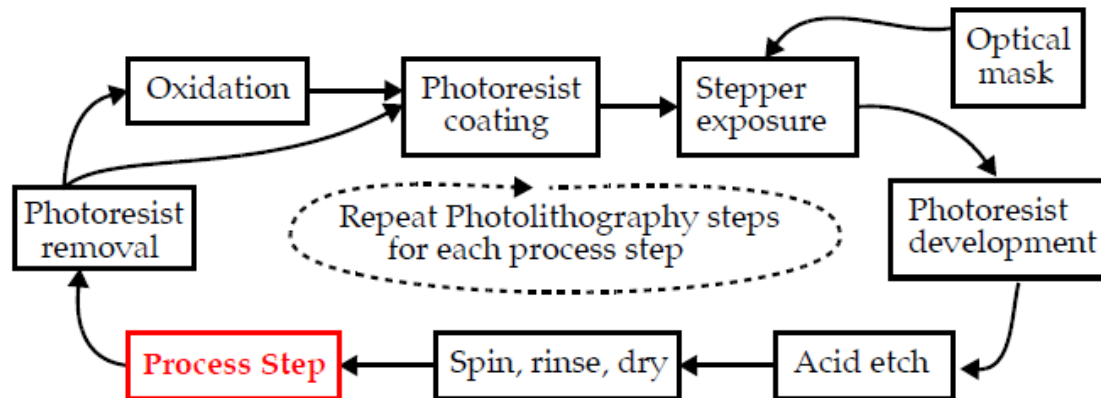
Tecnologia CMOS (brevetto Fairchild, 1963)

- La tecnologia CMOS (Complementary MOS) prevede l'utilizzazione di transistor sia di tipo N (NMOS) che di tipo P (PMOS) sullo stesso substrato, sfruttando la possibilità di poter effettuare diffusioni sia di tipo N che di tipo P per realizzare sia i contatti del transistor (source e drain) che il canale di collegamento fra di essi controllabile attraverso il segnale di gate



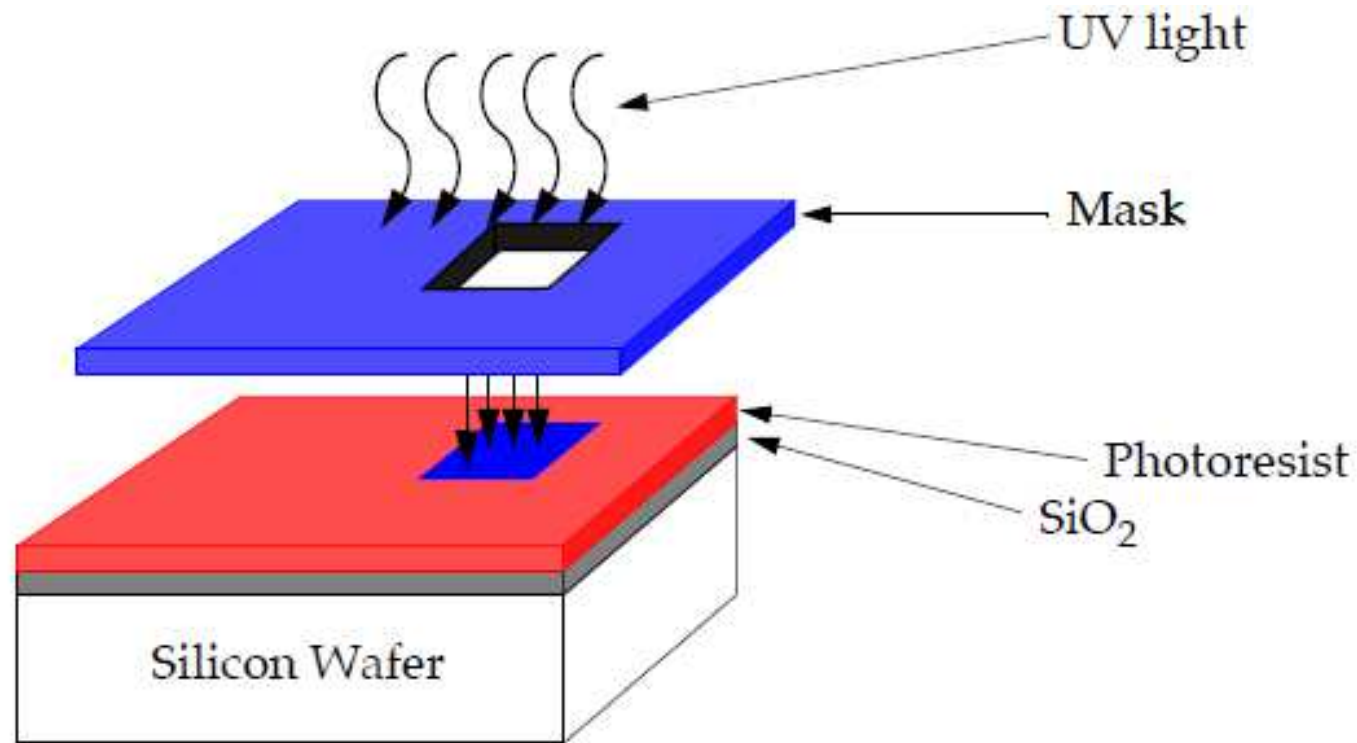
Come viene generata il transistor MOS

- Tipicamente si parte da un substrato (es.) di tipo p e si crea su di esso il dispositivo di attraverso una serie di passi che coinvolgono:
 - Crescita di ossido di Silicio (SiO_2)
 - Mascheratura
 - Attacco con acido per creare "fori"
 - Drogaggio o metallizzazione
 - Eventuale rimozione dell'ossido
- Questi passi vengono ripetuti piu' volte creando vari strati in cui possono essere cosi' create le diffusioni o le piste di metal/polisilicio (v. slide successive)

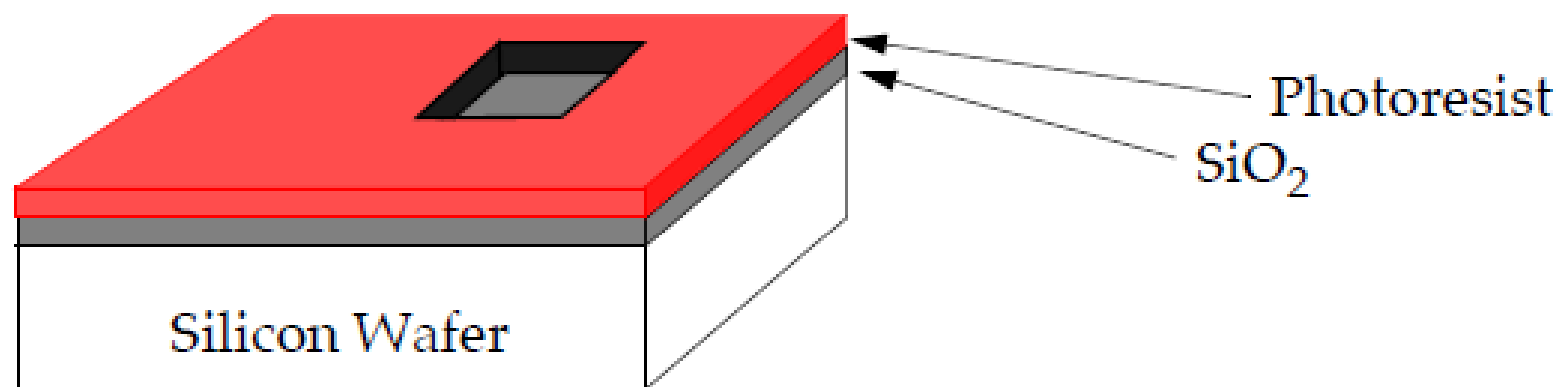


Generazione dei transistor MOS

- **Mascheratura.**
L'ossido e' gia' stato generato e ricoperto di photoresist.

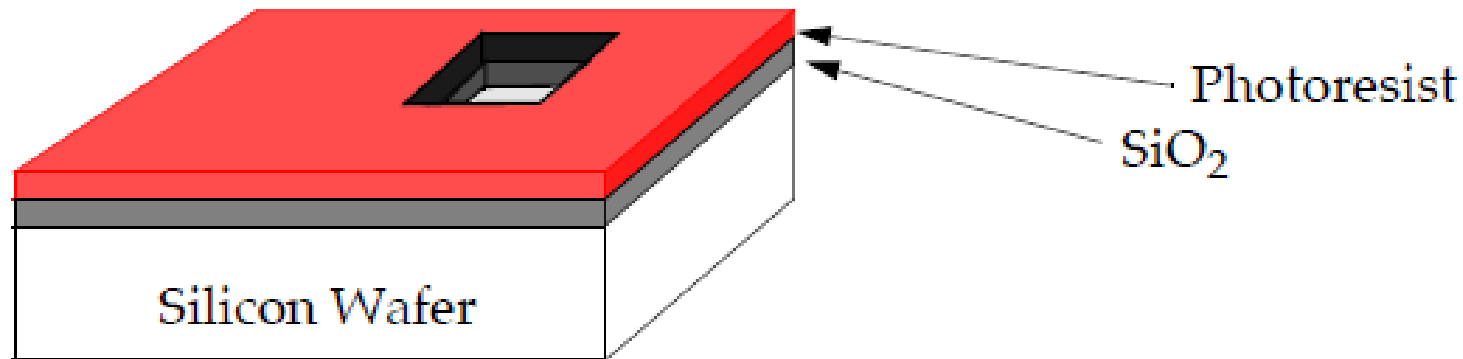


Rimozione photoresist polimerizzato dagli UV

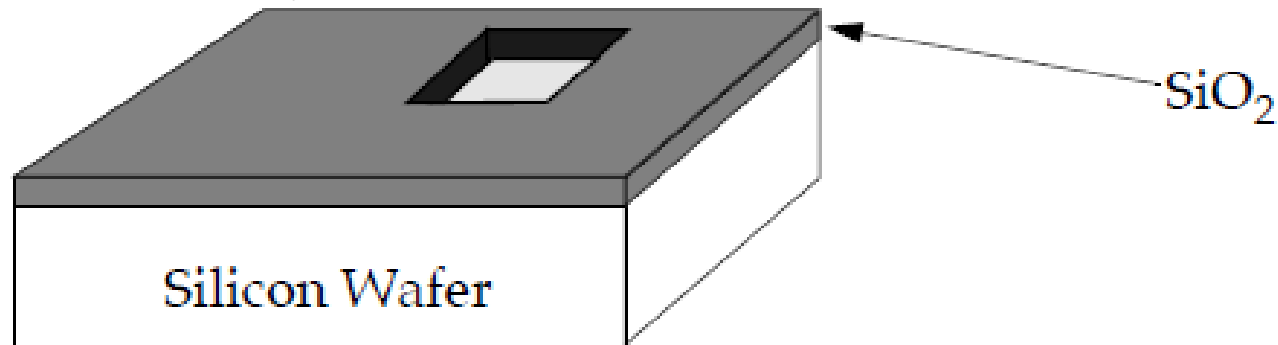


Attacco con l'acido e rimozione del photoresist

- Attacco es. con Acido Fluoridrico (HF) e creazione della finestra per la diffusione

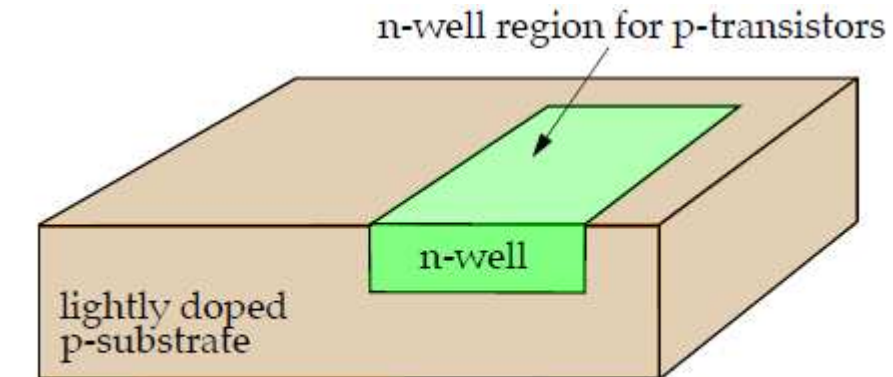


- Pulizia con acqua deionizzata e asciugatura con azoto
- Rimozione photoresist in eccesso



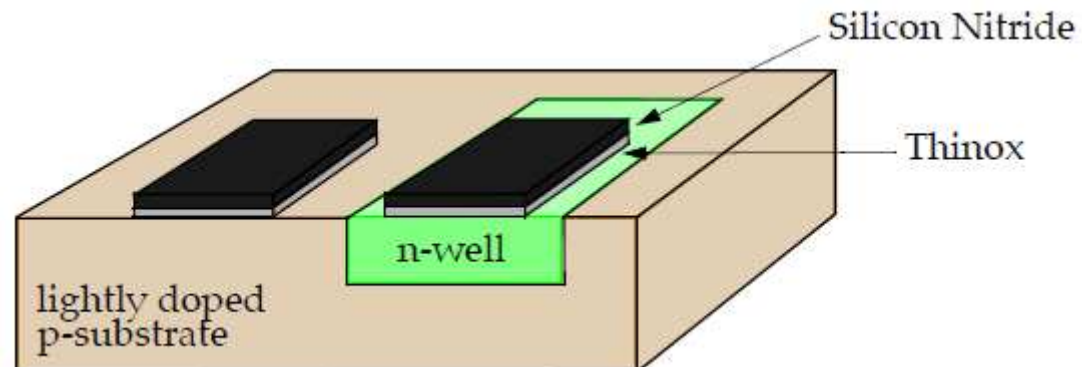
Processo CMOS semplificato

- Maschera N-WELL

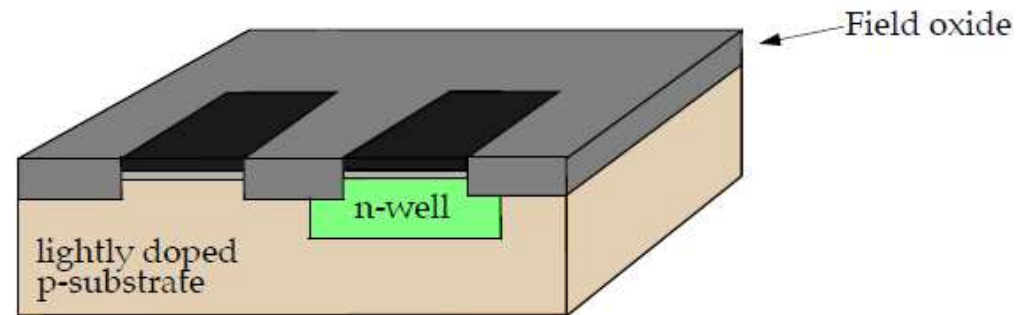


SiO_2 can be patterned as the mask for this step.

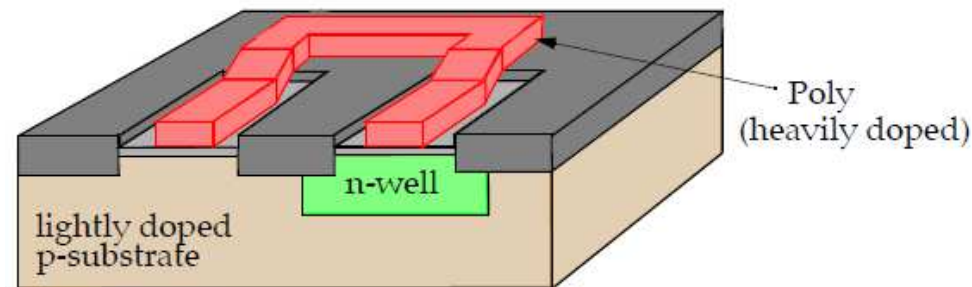
- Deposizione dell'OSSIDO di gate



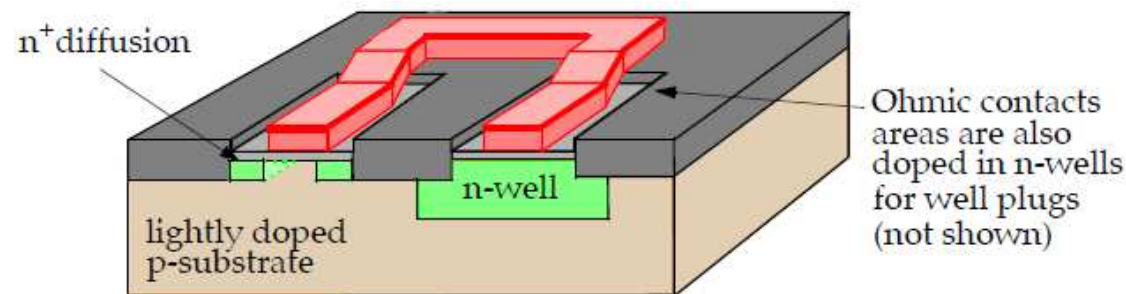
-
- Deposizione dell'ossido di isolamento



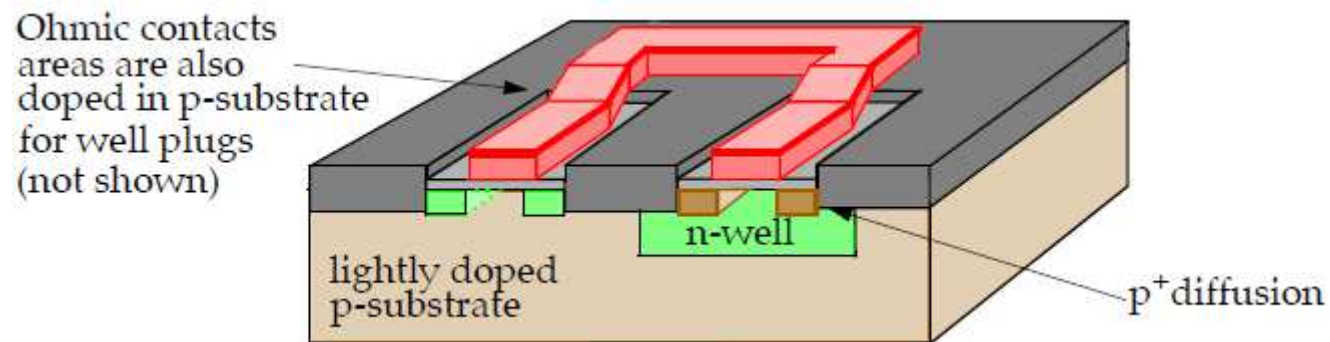
- Mascheratura per il polisilicio



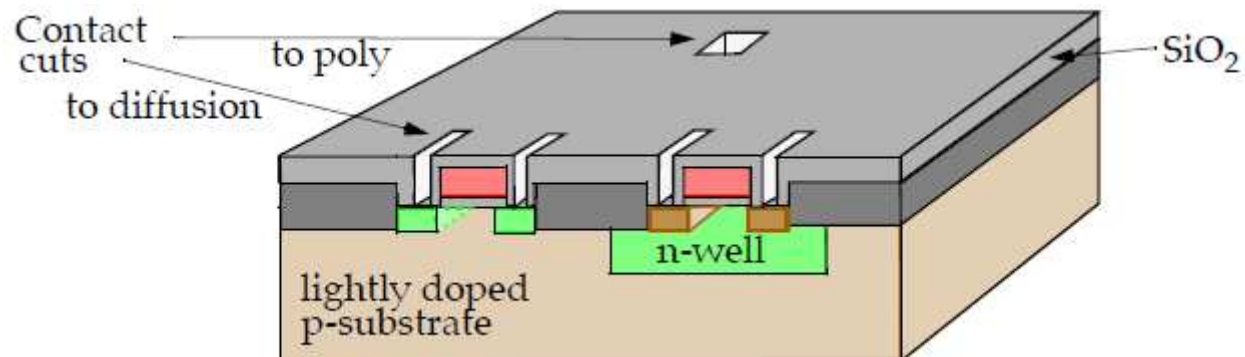
- Mascheratura per i contatti del transistor NMOS (sinistra)



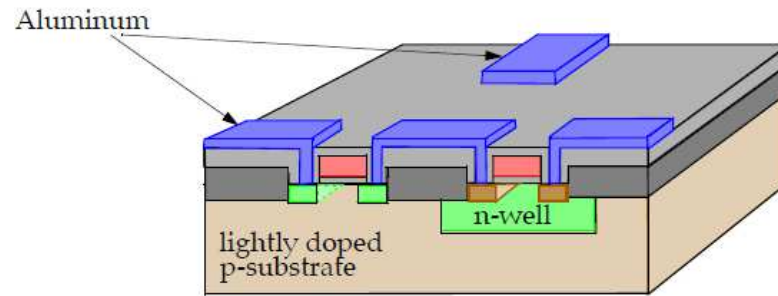
- Mascheratura per i contatti del transistor PMOS (destra)



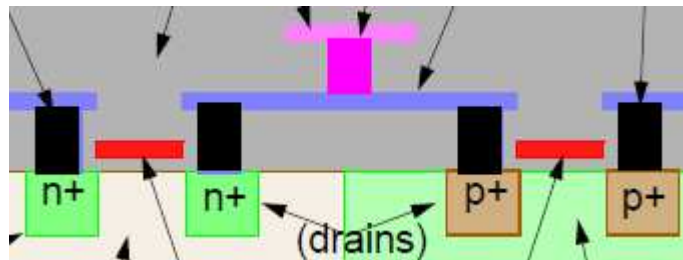
- Isolamento con ossido e fori per i contatti



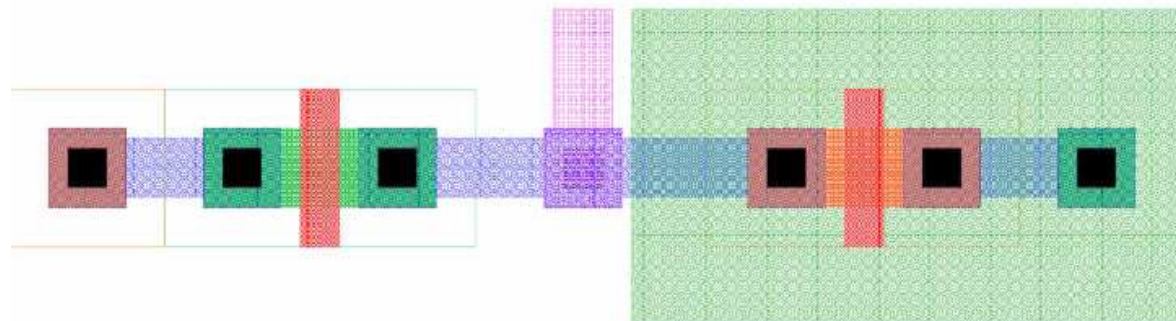
- Mascheratura per generare le piste di metal



- Con questo abbiamo generato la parte fondamentale dello schema iniziale

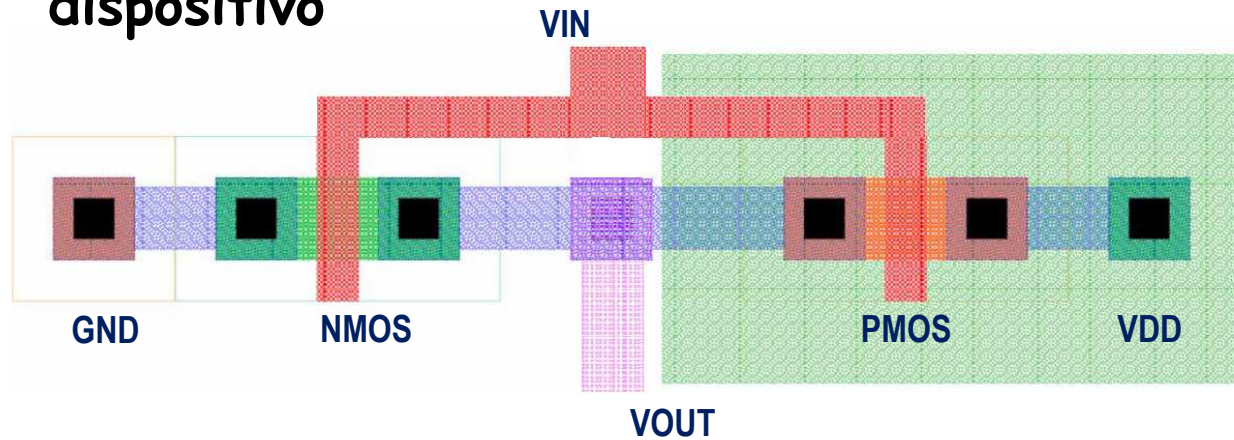


- Visto dall'alto il disegno puo' essere cosi' schematizzato (layout fisico)

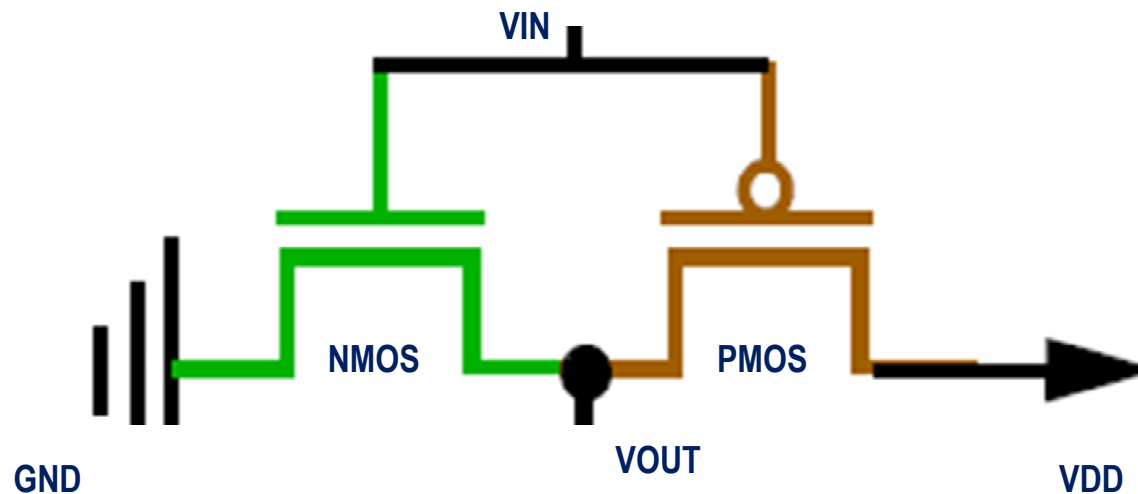


Layout fisico, elettrico e logico

- Questo "disegno" (layout fisico) fornisce quindi gli elementi essenziali per effettuare il progetto fisico del dispositivo



- Il corrispondente layout elettrico:



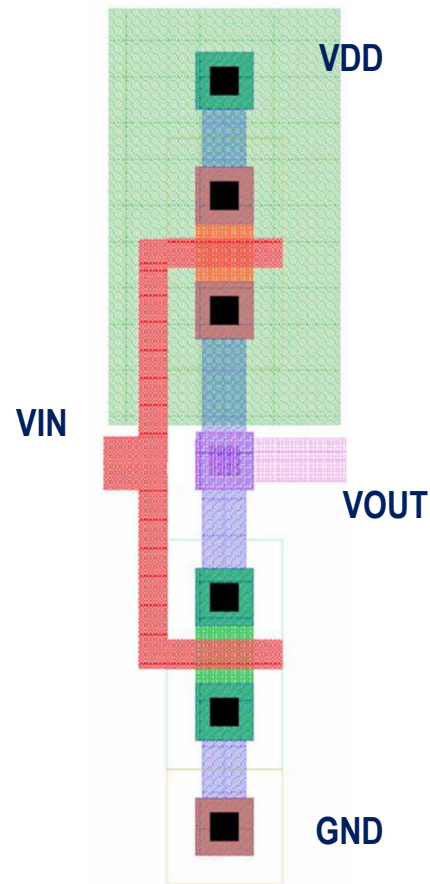
e logico:



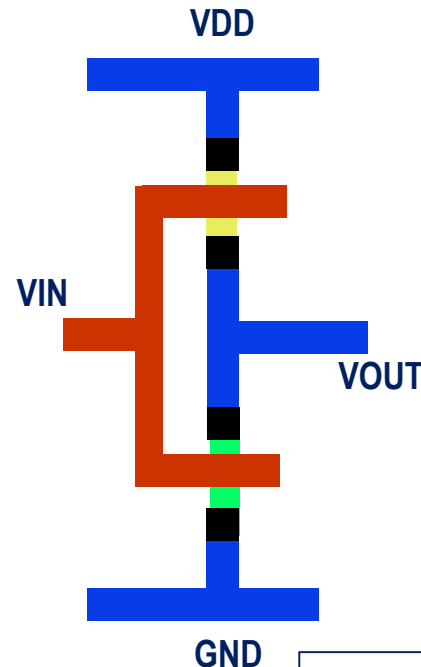
Layout fisico / Layout Simbolico / Stick diagram

- Il precedente layout fisico deve essere generato rispettando le caratteristiche fisiche della "tecnologia CMOS" specifica utilizzata (es. 14nm) → "layout rules"

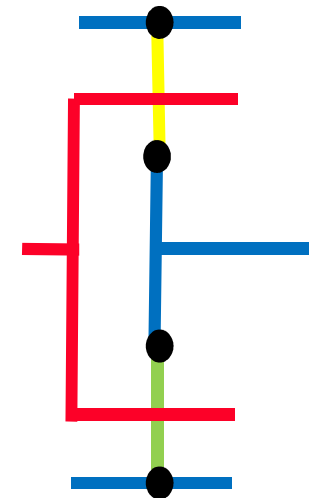
Layout fisico con design rules



Layout simbolico con design rules



Stick diagram



Le regole di layout impongono ad es. di rispettare certe distanze minime fra le piste, i contatti, etc

Nota: questa e' solo una delle possibili implementazioni fisiche di un inverter !

Vantaggi dell'inverter CMOS

- “Fully restored”: i livelli alto e basso (VDD e GND) sono completamente ristabiliti dopo il passaggio attraverso la porta
- “Ratioless”: I livelli logici non sono dipendenti dalle dimensioni relativi dei dispositivi
- Bassa impedenza di uscita: a regime l'uscita e' connessa o a VDD o a GND, aumentando la robustezza al rumore
- Alta impedenza di ingresso: il fanout (quanti dispositivi posso collegare sull'uscita) e' teoricamente illimitato in condizioni statiche, anche se il numero di dispositivi in uscita influenza il transitorio
- Bassa dissipazione di potenza statica: non c'e' mai un cammino diretto fra alimentazione e terra

Note importanti

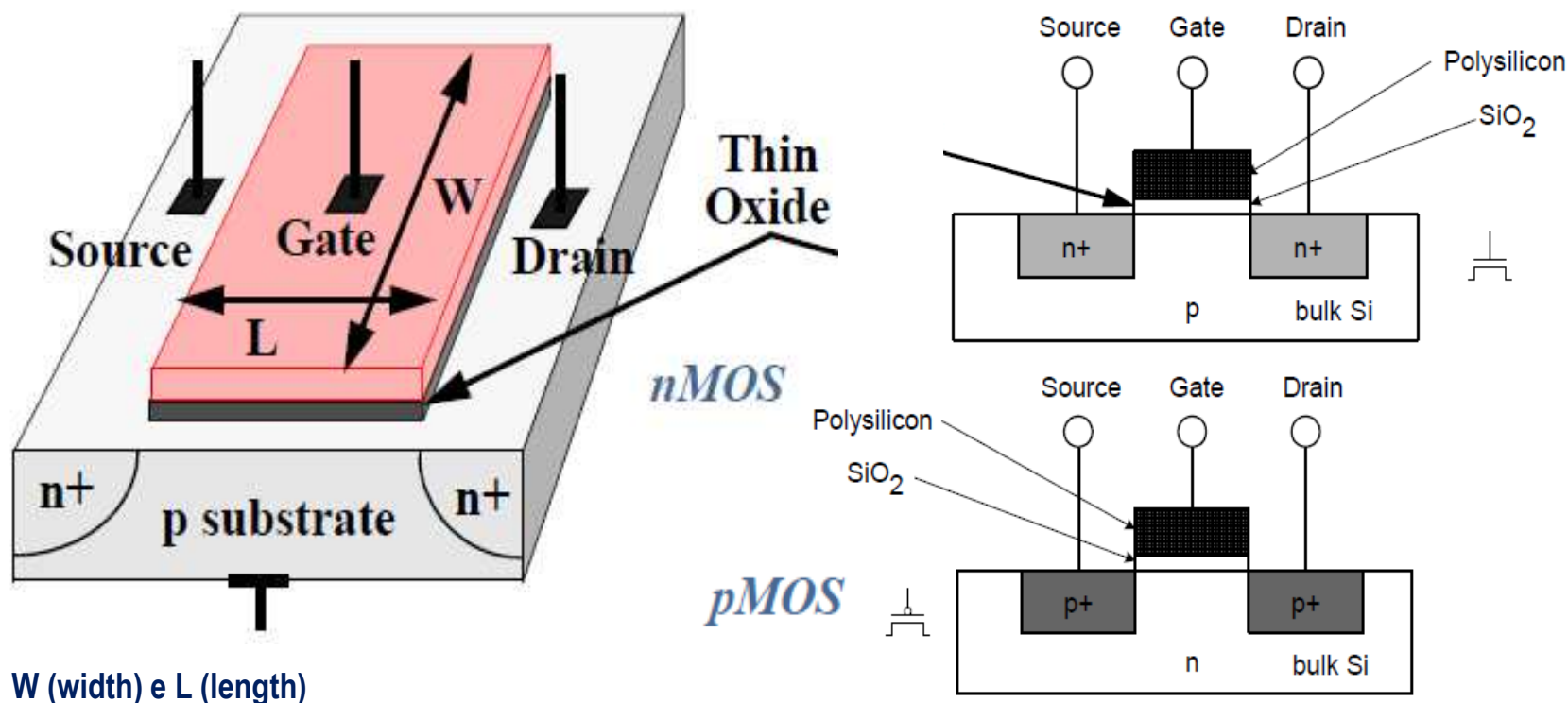
- La progettazione fisica descritta e' stata negli anni estremamente perfezionata e si e' recentemente (2011) passati alla produzione commerciale di dispositivi (es. FinFET) che si sviluppano non solo planarmente (2D) ma anche verticalmente (3D)
- La progettazione fisica permette di sfruttare al massimo ogni grado di liberta' (progettazione "full-custom") ma nel caso di chip contenenti vari miliardi di dispositivi non e' umanamente gestibile se non con l'aiuto di strumenti CAD (Computer Aided Design)
 - Tipicamente, si ricorre per la grande maggioranza dei casi a librerie gia' predisposte ("standard cells") e si usa il design full-custom solo in casi di necessita'
 - Il design fisico puo' quindi essere gestito con appositi strumenti in cui il progettista disegna "blocchi colorati" e automaticamente vengono verificate e imposte le regole di layout (es. L-Edit, Cadence tools, Synopsys tools, ...)
- Il passo puo' essere automatizzato (similmente a un compilatore per il software) generando il layout fisico a partire dal circuito elettrico
 - E' importante anche il passo inverso: estrarre il circuito elettrico (compreso gli effetti "parassiti" quali capacita', cross-talking) a partire dal layout fisico

Osservazioni importanti

- La fabbricazione dei chip e' appannaggio di pochi in quanto il costo di un "silicon foundry" supera oggi i 10 miliardi di euro (es. Intel, Samsung, Global Foundries)
- Viceversa la *progettazione* puo' essere effettuata anche con pochissime risorse
 - Grandi ditte come ARM, AMD, Xilinx hanno un business model "fabless" ovvero dopo aver generato il layout fisico lo passano a una silicon foundry che crea il chip per loro o addirittura vendono il design (es. ARM)

Caratteristiche elettriche del transistor MOS

- Abbiamo visto come si puo' realizzare un transistor MOS



- Dimensionando opportunamente W/L si possono variare i tempi di risposta (o di commutazione)

$$I_D = k_n' \frac{W}{L} \left[(V_{GS} - V_T) V_{DS} - \frac{V_{DS}^2}{2} \right]$$

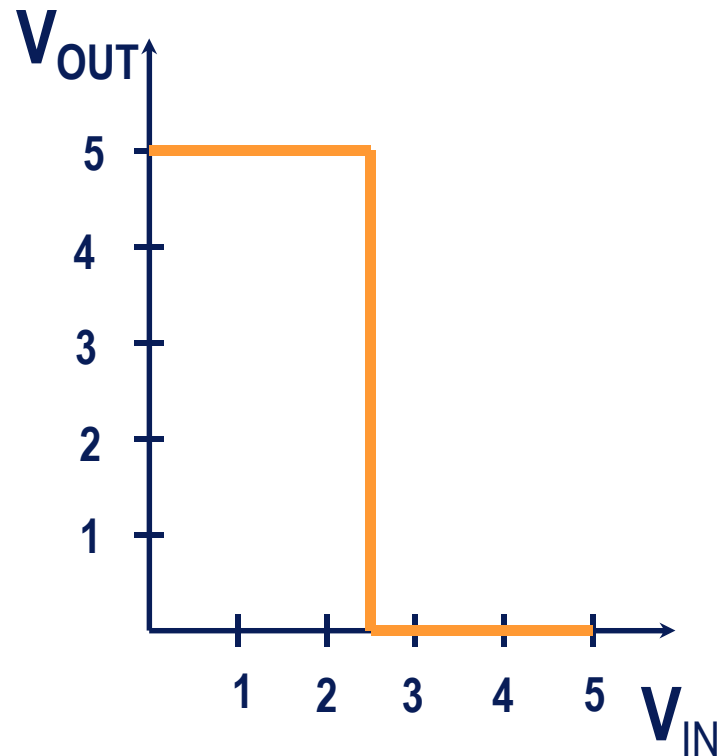
$$k_n' = \mu_n C_{ox} = \frac{\mu_n \epsilon_{ox}}{t_{ox}}$$

V_T Tensione di soglia

CARATTERISTICA DI TRASFERIMENTO dell'INVERTER

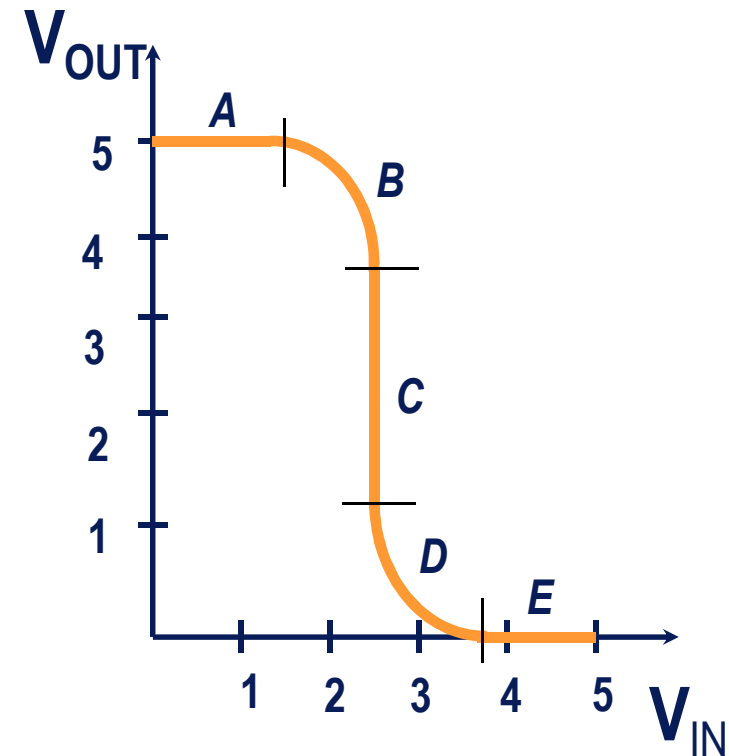
- Caratteristica

IDEALE



Caratteristica

REALE



Tratto di curva	Mn	Mp
A	interdizione	zona triodo
B	saturazione	zona triodo
C	saturazione	saturazione
D	zona triodo	saturazione
E	zona triodo	Interdizione

Osservazioni

- La tecnologia CMOS consente di realizzare la funzione logica piu' semplice (inverter o porta NOT) in modo tale che la sua caratteristica di trasferimento sia vicina alla caratteristica ideale
 - Altre tecnologie (es. NMOS, BJT) hanno alcune limitazioni (NMOS: livelli logici non pieni, BJT: alto consumo) ma possono offrire alcuni vantaggi (NMOS: maggiore compattezza, BJT: maggiore velocita')
 - Attualmente la tecnologia CMOS e' la tecnologia di gran lunga piu' utilizzata per realizzare sistemi digitali soprattutto per la sua efficienza energetica
- Per realizzare funzioni logiche piu' complesse ci viene in aiuto l'algebra booleana (v. lezioni successive)
 - In teoria oltre all'inverter ci servirebbero le funzioni elementari AND e OR
 - In pratica, le porte piu' semplici da realizzare a partire dall'inverter sono le porte NAND e NOR (v. slide successive)
 - Fortunatamente si puo' facilmente dimostrare che e' possibile costruire qualsiasi funzione logica a partire anche dalle sole porte NAND (o NOR)
 - E' immediato altresì vedere che NAND seguito da NOT equivale a AND e NOR seguito da NOR quivale a OR.

Porta NAND CMOS

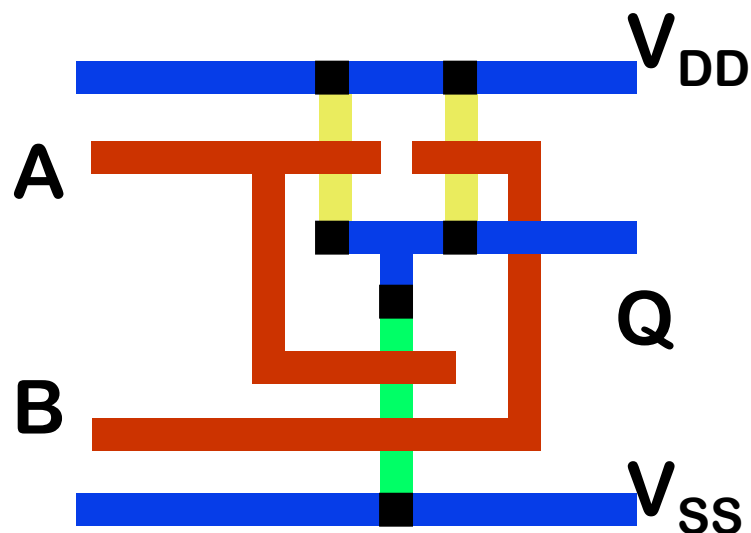
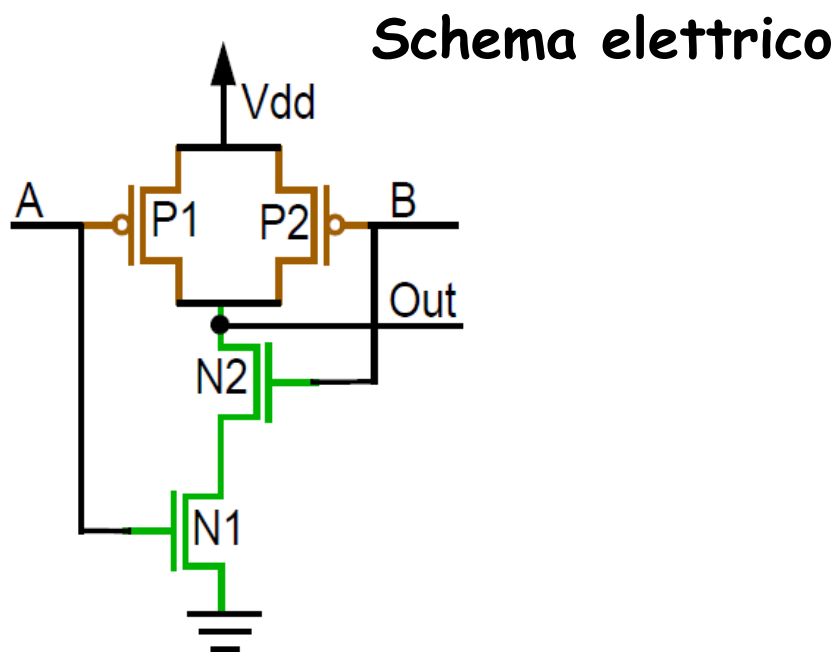
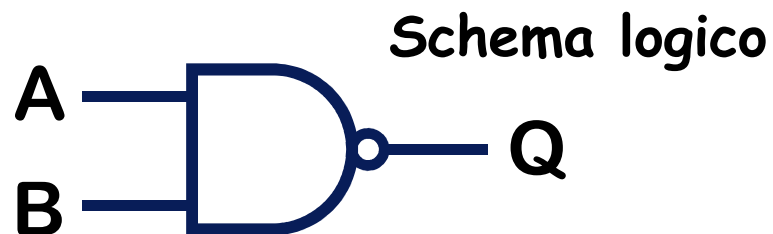


Tabella di Verità

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0



Porta NOR CMOS

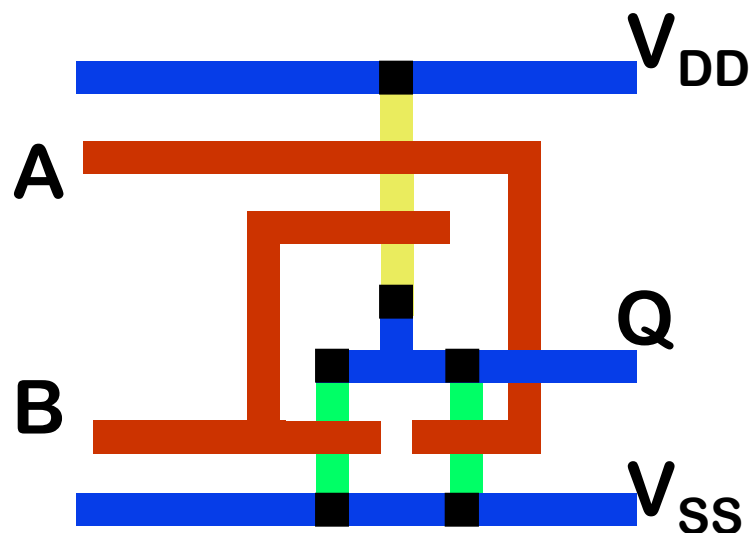
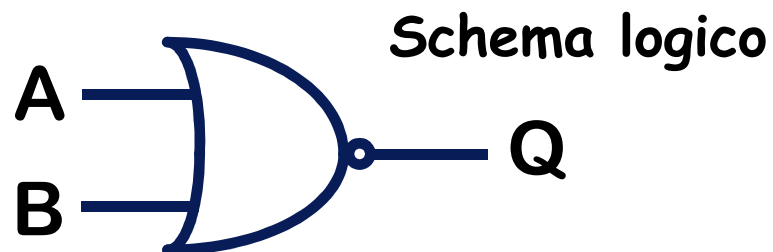
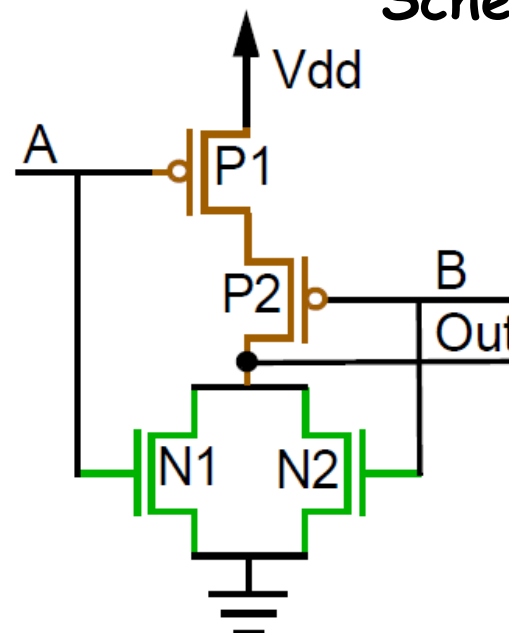


Tabella di Verità

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0



Schema elettrico



Porta di transito CMOS (transmission gate)

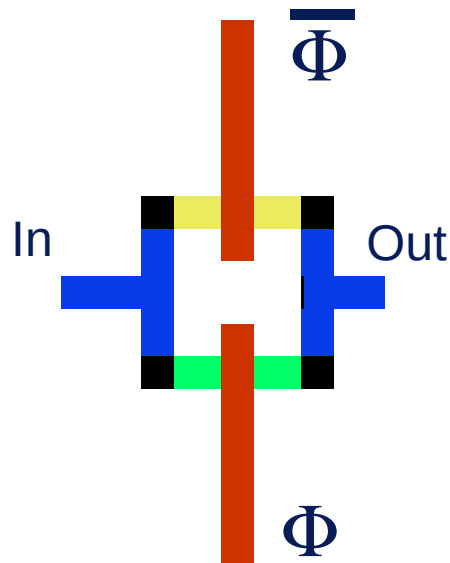
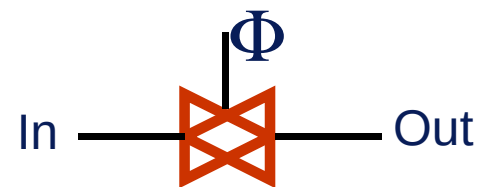


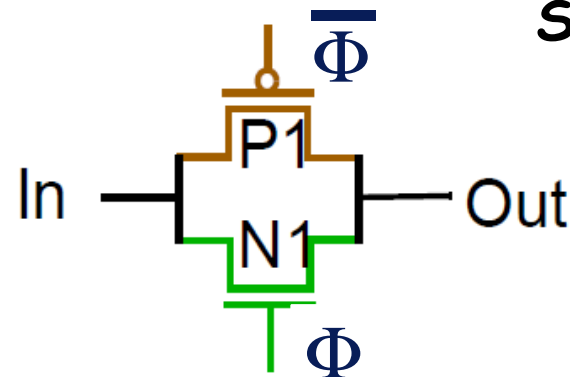
Tabella di Verità

Φ	In	Out
0	X	Z
1	0	0
1	1	1



Simbolo

Nota: $\overline{\Phi}$ e' sottinteso



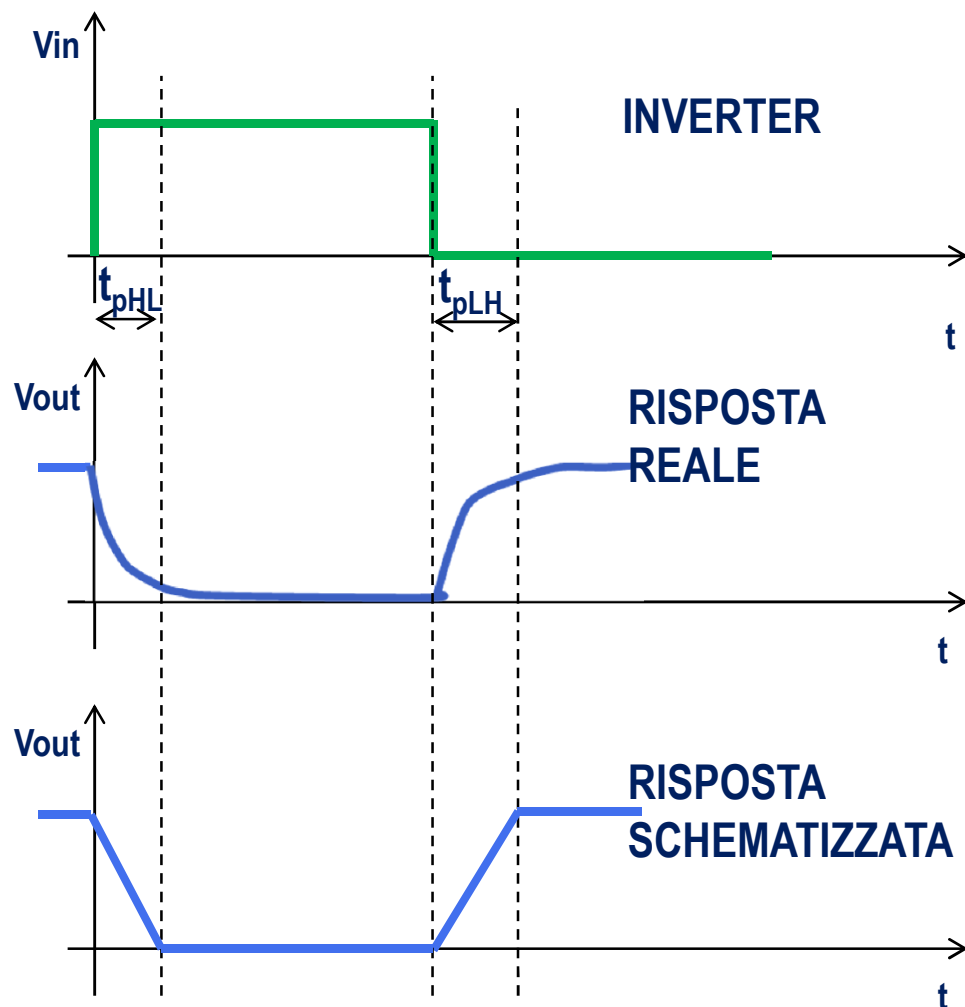
Schema elettrico

Nota: non e' un "tri-state" buffer in quanto quest'ultimo rigenera anche il segnale, mentre in questo caso il segnale si degrada un poco nel passaggio.

Nota2: viceversa si realizza un buffer mettendo due inverter in cascata

Ritardo di propagazione dell'inverter CMOS

- Le porte fondamentali e di trasmissione presentano un certo ritardo di propagazione t_p



Ad esempio:

applicando un segnale a gradino all'ingresso dell'inverter la forma d'onda di uscita raggiunge un livello accettabile (es. 90%) con un ritardo t_p

Il ritardo necessario per la transizione dal livello alto a livello basso t_{pHL} può essere diverso da quello della transizione opposta t_{pLH}

$$t_{pHL} = \ln(2) R_{eqn} C_L = 0.69 R_{eqn} C_L$$

$$t_{pLH} = \ln(2) R_{eqp} C_L = 0.69 R_{eqp} C_L$$

$$R_{eq} \approx \frac{3}{4} \frac{V_{DD}}{I_{DSAT}} \left(1 - \frac{7}{9} \lambda V_{DD} \right) \quad \begin{array}{l} \lambda: \text{empirical parameter} \\ \text{called } \textit{channel-length moduli} \end{array} \quad \lambda \propto \frac{1}{L}$$

$$I_{DSAT} = k \frac{W}{L} \left((V_{DD} - V_T) V_{DSAT} - \frac{V_{DSAT}^2}{2} \right)$$

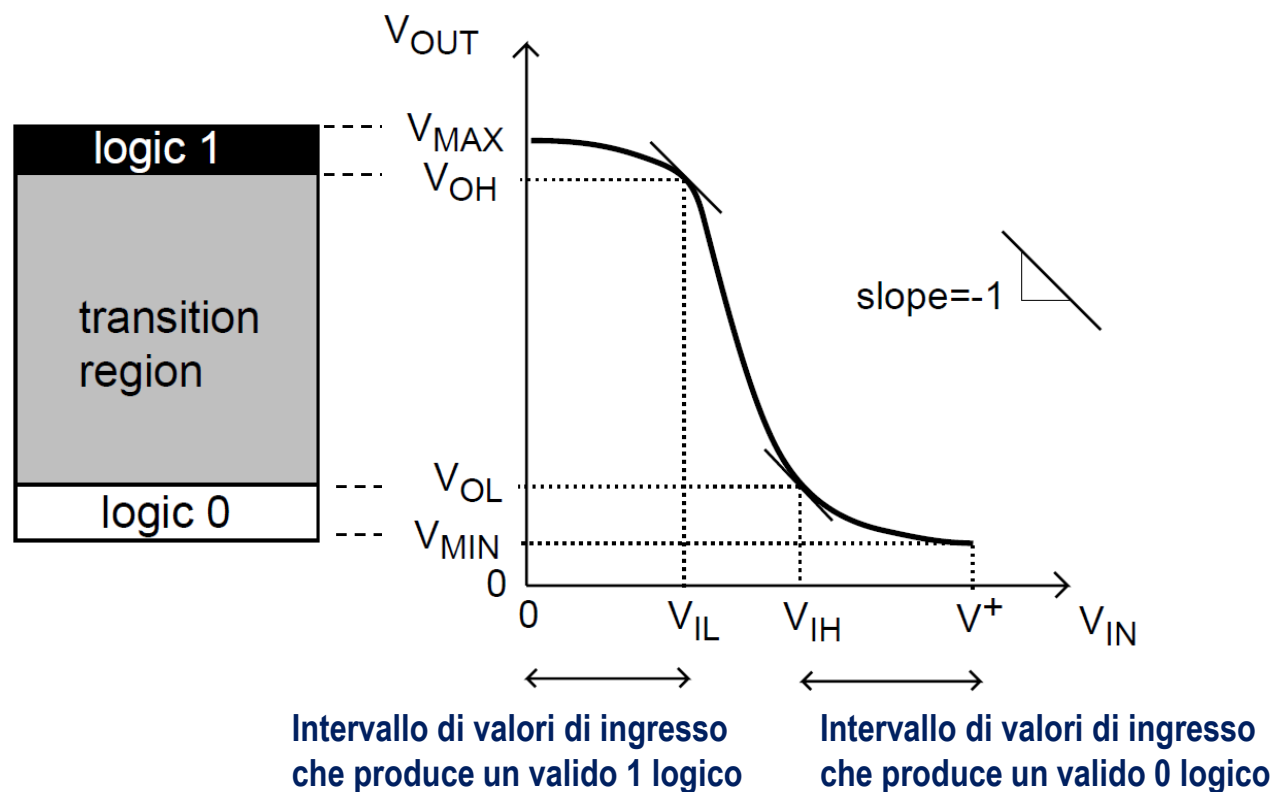
$$V_{DSAT} = \kappa (V_{GS} - V_T) (V_{GS} - V_T)$$

$$V_{DD} \uparrow \Rightarrow t_p \downarrow$$

$$L \downarrow \Rightarrow t_p \downarrow$$

Approfondimenti ulteriori possono essere fatti nei corsi di Elettronica. In questo contesto schematizzeremo il ritardo con una transizione lineare con durata finita

Definizione dei livelli logici 1 e 0



- **0 logico:**

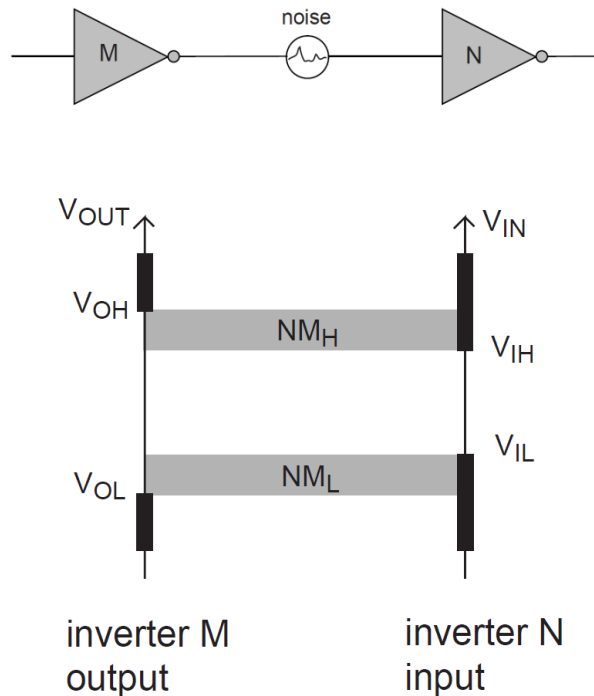
- V_{min} = tensione di uscita per la quale $V_{IN} = V^+$
- V_{OL} = la piu' piccola tensione di uscita per la quale la pendenza $e' = -1$

- **1 logico:**

- V_{OH} = la piu' grande tensione di uscita per la quale la pendenza $e' = -1$
- V_{MAX} = tensione di uscita per la quale $V_{IN} = 0$

Margine di rumore

- Se $(V_{OH}-V_{OL}) > (V_{IH}-V_{IL})$ l'inverter ha una certa immunità al rumore



- $NM_H = (V_{OH}-V_{IH})$
- $NM_L = (V_{IL}-V_{OL})$

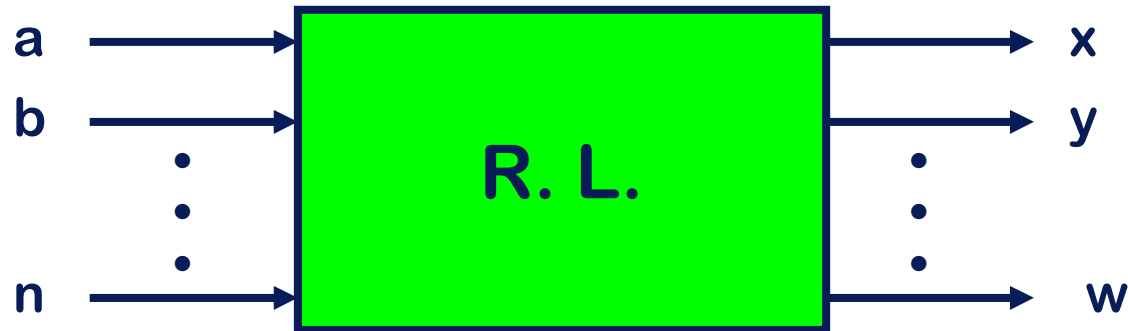
margine di rumore sul livello alto

margine di rumore sul livello basso

Dalle porte elementari
alle reti logiche

Reti Logiche

- Sistema elettronico che ha in ingresso segnali digitali e fornisce in uscita segnali digitali secondo leggi descrivibili con l'algebra Booleana



- R.L. è unidirezionale

Tipi di reti

- Reti COMBINATORIE

- In qualunque istante le uscite sono funzione del valore che gli ingressi hanno in quell'istante
- Il comportamento (uscite in funzione degli ingressi) è descritto da una tabella

- Reti SEQUENZIALI

- In un determinato istante le uscite sono funzione del valore che gli ingressi hanno in quell'istante e i valori che hanno assunto precedentemente
- La descrizione è più complessa
- Stati Interni
- Reti dotate di *MEMORIA*

Simboli

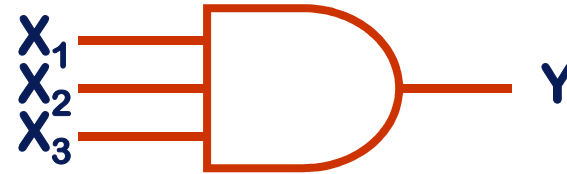
- Rete Logica => scomponibile in blocchi
- Blocchi base = simboli degli operatori elementari
- Rappresentazione delle funzioni logiche mediante schemi
- RAPPRESENTAZIONE SCHEMATICA

Porte Logiche Base

- Rappresentazione circuitale delle funzioni logiche

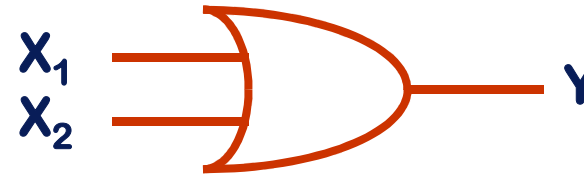
- AND

$$Y = X_1 \bullet X_2 \bullet X_3$$



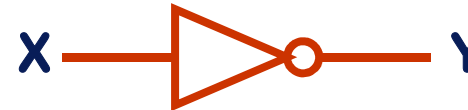
- OR

$$Y = X_1 + X_2$$



- NOT

$$Y = \overline{X}$$

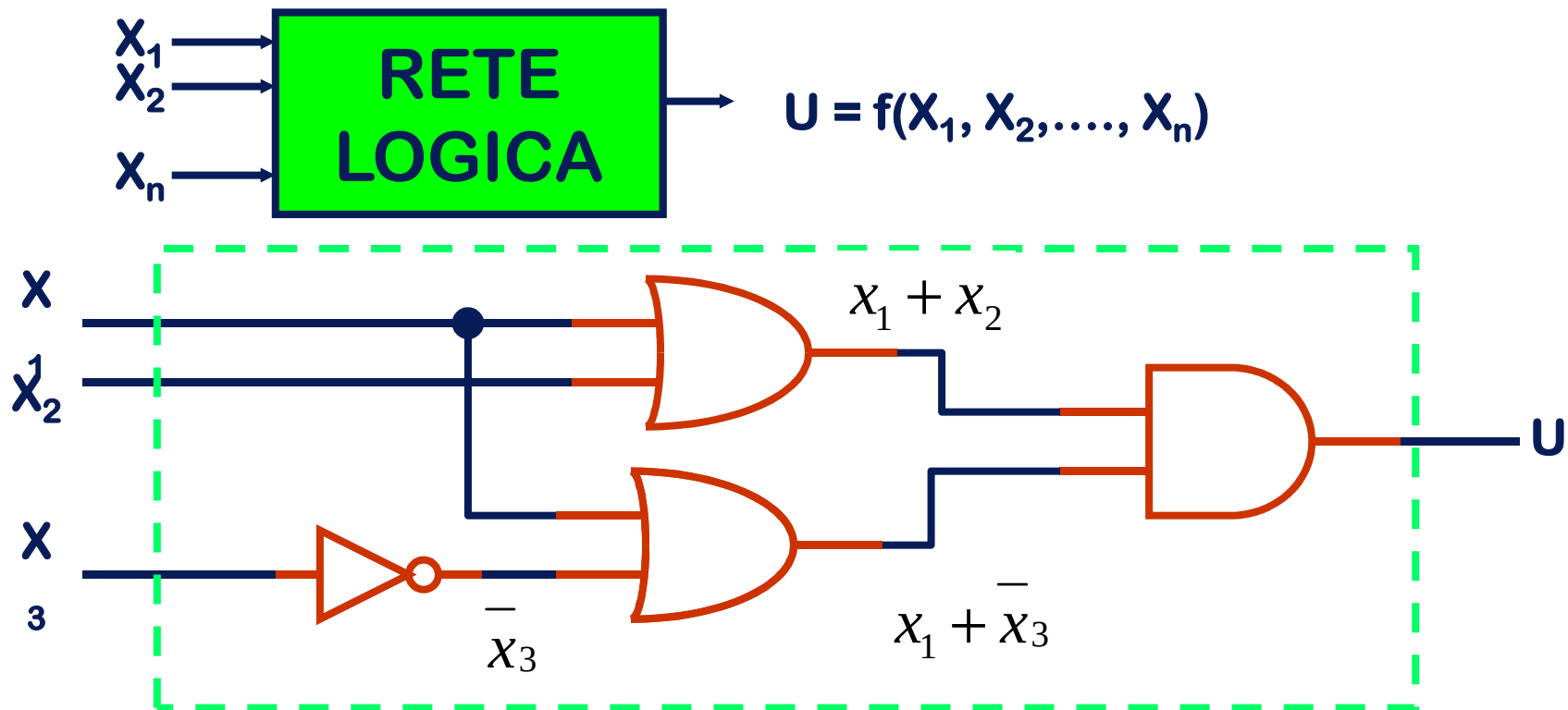


Esempio

- Schema simbolico della funzione

$$U = f(X_1, X_2, \dots, X_n) = (X_1 + X_2) \bullet (X_1 + \overline{X_3})$$

- RETE LOGICA

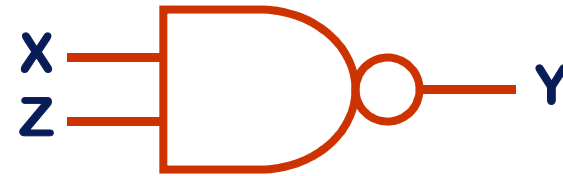


Altre porte logiche

- NAND

X	Z	Y
0	0	1
0	1	1
1	0	1
1	1	0

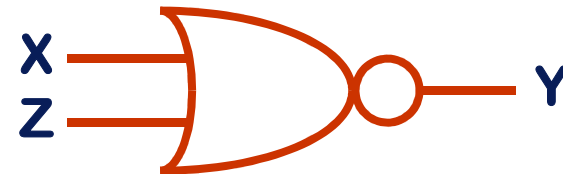
$$Y = \overline{X \bullet Z}$$



- NOR

X	Z	Y
0	0	1
0	1	0
1	0	0
1	1	0

$$Y = \overline{X + Z}$$



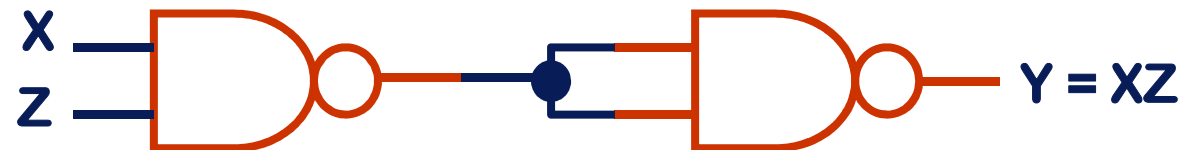
Proprietà della porta NAND (NOR)

- Utilizzando solamente porte NAND (NOR) è possibile realizzare qualunque rete logica

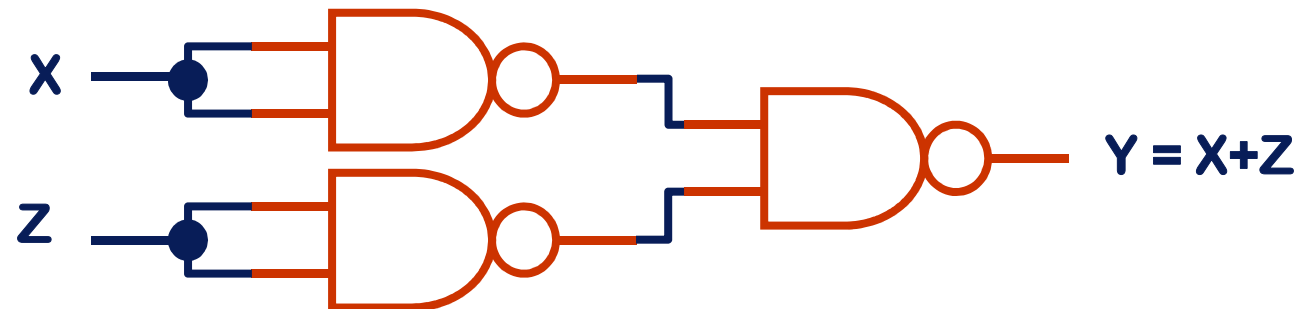
- NOT



- AND



- OR

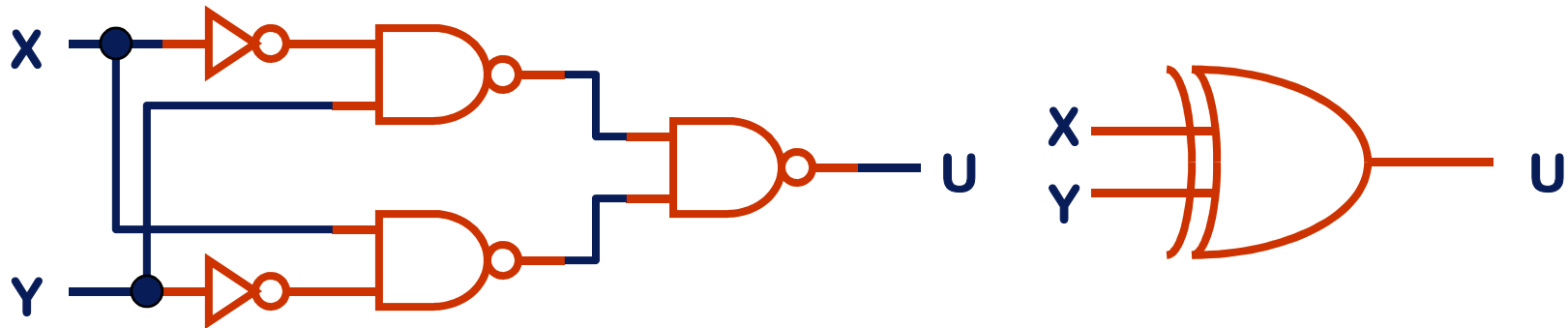


OR Esclusivo

- Realizzazione dell'OR Esclusivo

X	Y	U
0	0	0
0	1	1
1	0	1
1	1	0

$$U = X \oplus Y = \overline{X}Y + X\overline{Y}$$



Algebra Booleana

Algebra della Logica

- **George Boole**
 - Matematico inglese (1815 - 1864)
 - Algebra della Logica, Algebra di Boole, Algebra Booleana
 - Sistema matematico formale che descrive funzioni logiche
 - Funzioni che possono assumere al minimo (solo) due valori
 - Vero Falso
 - Le variabili di funzioni logiche possono assumere solo due valori
 - Sistema matematico formale
 - Insieme di elementi
 - insieme di operazioni
 - insieme di postulati
- » TEOREMI

Definizioni

- **Elementi** **(2) [Algebra delle commutazioni]**
 - 0 (logico) 1 (logico)
 - Falso Vero
 - Livello logico Baso Livello logico Alto
 - 0 V 5 V
- **Costanti** Possono assumere due valori
- **Variabili** Possono assumere due valori

$$x = 0 \quad \text{se} \quad x \neq 1$$

$$x = 1 \quad \text{se} \quad x \neq 0$$

Definizione di "AND"

- Operazione

- AND o PRODOTTO LOGICO

$$x \bullet y \qquad xy$$

- Postulato

- l'operazione AND è definita dalla tabella

x		y		$x \bullet y$
0	•	0	=	0
0	•	1	=	0
1	•	0	=	0
1	•	1	=	1

Osservazioni

1. $x \bullet y$ è uguale a "1" se e solo se x e y sono uguali a "1", altrimenti $x \bullet y$ è uguale a "0"
2. Si può estendere a "n" variabili:
 $x_1 \bullet x_2 \bullet \dots \bullet x_n$ è uguale "1" se e solo se $x_1, x_2, \dots, x_n$ sono uguali a "1"
- La funzione AND corrisponde al concetto:
un evento si verifica se e solo se tutte le condizioni sono verificate

Definizione di "OR"

- Operazione

- OR o SOMMA LOGICA

$$x + y$$

- Postulato

- l'operazione OR è definita dalla tabella

x		y		$x + y$
0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	1

Osservazioni

1. $x + y$ è uguale a "0" se e solo se x e y sono uguali a "0", altrimenti $x + y$ è uguale a "1"
2. Si può estendere a "n" variabili:
 $x_1 + x_2 + \dots + x_n$ è uguale "0" se e solo se $x_1, x_2, \dots, x_n$ sono uguali a "0"
- La funzione OR corrisponde al concetto:
perché un evento si verifica è sufficiente che una sola condizioni sia verificata

Definizione di "NOT"

- Operazione

- NOT o Complemento Logico , o Negazione, o Inversione

- Postulato

- l'operazione NOT è definita dalla tabella

x	$\overline{x}$
0	1
1	0

Osservazioni

1. se x è uguale a "0" allora x negato è uguale a "1", se x è uguale a "1" allora x negato è uguale a "0"

2. Ovvero $\overline{X} = 1$ se $X = 0$

$\overline{X} = 0$ se $X = 1$

$\overline{0} = 1$ e $\overline{1} = 0$

- La funzione NOT corrisponde al concetto:
negazione della condizione

Funzione logica (o Booleana)

- Una funzione

$$u = f(x_1, \dots, x_n)$$

è una legge che fa corrispondere un valore logico (0 o 1) di u ad ogni combinazione di valori $x_1, \dots, x_n$.

- La funzione f è costituita da variabili logiche, costanti e le tre operazioni logiche fondamentali

$$u = \overline{(x_1 + x_2)} \bullet (x_3 + x_1 \bullet \overline{x_2}) + \overline{x_3}$$

Osservazioni

- Nelle funzioni logiche le parentesi indicano una gerarchia di esecuzione uguale a quella comunemente usata nelle espressioni aritmetiche note
- Fra le operazioni logiche AND, OR e NOT esiste la gerarchia: 1) NOT, 2) AND, 3) OR
- La gerarchia prima descritta consente di ridurre l'uso di parentesi nelle funzioni logiche

Tabella di Verità (1)

- Una funzione logica può sempre essere espressa da una tabella che prende il nome di:

TABELLA DI VERITÀ (TRUTH TABLE)

- Osservazione
- Una funzione di “n” variabili ammette 2^n possibili configurazioni
- Una funzione di “n” variabili è completamente descritta da una tabella che ha sulla sinistra le 2^n possibili configurazioni degli ingressi e a destra i valori (0 o 1) a secondo del valore della funzione

Tabella di verità (2)

- Funzione di tre variabili

$$u = f(x, y, z)$$

x	y	z	u
0	0	0	$f(0,0,0)$
0	0	1	$f(0,0,1)$
0	1	0	$f(0,1,0)$
0	1	1	$f(0,1,1)$
1	0	0	$f(1,0,0)$
1	0	1	$f(1,0,1)$
1	1	0	$f(1,1,0)$
1	1	1	$f(1,1,1)$

Esempio

$$u = f(x, y, z) = (x + \bar{y}) \cdot (\bar{x} + z) + yz$$

$$f(0,1,1) = (0 + \bar{1}) \cdot (\bar{0} + 1) + 1 \cdot 1 = (0 + 0) \cdot (1 + 1) + 1 = 0 \cdot 1 + 1 = 0 + 1 = 1$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 1

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 2

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 3

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 4

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 5

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Passo 6

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Fine

$$u = f(x, y, z) = (x + \bar{y})(\bar{x} + z) + yz$$

x	y	z	$\bar{x}$	$\bar{y}$	$x + \bar{y}$	$\bar{x} + z$	$(x + \bar{y})(\bar{x} + z)$	yz	u
0	0	0	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1
0	1	0	1	0	0	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	1	1	0	0	0	0
1	0	1	0	1	1	1	1	0	1
1	1	0	0	0	1	0	0	0	0
1	1	1	0	0	1	1	1	1	1

Richiami

- Insieme di elementi
- Variabili, costanti
- Insieme di operazioni
- Insieme di postulati
- Espressioni algebriche
- Tabella di verità

Postulati di HUNTINGTON

Algebra Booleana

Almeno due elementi distinti

1a Somma logica (+)

1b Prodotto logico ($\bullet$)

Elementi Identità

2a $x + 0 = x$

2b $x \bullet 1 = x$

Proprietà Commutativa

3a $x + y = y + x$

3b $x \bullet y = y \bullet x$

Proprietà Distributiva

4a $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

4b $x + (y \bullet z) = (x + y) \bullet (x + z)$

Complementazione

5a $x + \bar{x} = 1$

5b $x \bullet \bar{x} = 0$

Osservazioni

- Alcune proprietà dell'algebra booleana sono vere anche nell'algebra normalmente usata:
 - Proprietà commutativa
 - Proprietà distributiva del prodotto logico
- Altre proprietà non sono vere :
 - Proprietà distributiva della somma logica
- L'operazione complemento logico esiste solo nell'algebra booleana
- La sottrazione e la divisione non esistono nell'algebra booleana

Principio di DUALITÀ

- Da un'osservazione dei postulati precedenti si osserva che quelli "b" si ottengono da "a"
 - Scambiando i due operatori binari fra loro, $(+)$ con $(\cdot)$ e $(\cdot)$ con $(+)$
 - Scambiando fra loro i due elementi identità, 1 con 0 e 0 con 1

TEOREMI FONDAMENTALI

- **Tecniche di dimostrazione dei teoremi**
 - Impiego dei postulati fondamentali
 - Uso di teoremi precedentemente dimostrati
 - Dimostrazione per assurdo
 - (si ipotizza verificata l'ipotesi opposta a quella desiderata e si conclude che non è possibile che sia vera)
 - Dimostrazione per induzione
 - (se una ipotesi è vera per k variabili e per $k+1$ variabili allora è vera per qualunque n)

Osservazione

- La tabella di verità consente di provare la veridicità di una relazione logica, poiché verifica se la relazione è vera per TUTTE le possibili combinazioni dei valori delle variabili
- Tale metodo prende il nome di
- *Metodo dell'INDUZIONE PERFETTA*

TEOREMI

$$1a \quad x + 1 = 1$$

$$1b \quad x \bullet 0 = 0$$

$$2a \quad \overline{(\overline{x})} = x$$

$$3a \quad x + x = x$$

$$3b \quad x \bullet x = x$$

$$4a \quad x + xy = x$$

$$4b \quad x(x + y) = x$$

$$5a \quad x + \overline{x}y = x + y$$

$$5b \quad x(\overline{x} + y) = xy$$

$$6a \quad x + (y + z) = (x + y) + z = x + y + z$$

$$6b \quad x(yz) = (xy)z = xyz$$

$$7a \quad xy + \overline{x}z + yz = xy + \overline{x}z$$

$$7b \quad (x + y)(\overline{x} + z)(y + z) = (x + y)(\overline{x} + z)$$

$$8a \quad \overline{(x + y)} = \overline{x} \cdot \overline{y}$$

$$8b \quad \overline{(x \cdot y)} = \overline{x} + \overline{y}$$

Esempio di dimostrazione

- Teorema di De Morgan (8a e 8b)

$$8a \quad \overline{(x + y)} = \bar{x} \cdot \bar{y} \qquad 8b \quad \overline{(x \cdot y)} = \bar{x} + \bar{y}$$

x	y	$\bar{x}$	$\bar{y}$	$x + y$	$\overline{(x + y)}$	$\bar{x} \cdot \bar{y}$
0	0	1	1	0	1	1
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	1	0	0

c.v.d.

Osservazioni

1. I teoremi di destra si possono ottenere da quelli di sinistra scambiando OR con AND e "0" con "1"
2. Principio di dualità
3. Molti dei teoremi visti sono veri anche nell'algebra che conosciamo
4. Particolarmente significativi sono i teoremi di De Morgan e la proprietà distributiva
5. Molti teoremi, in particolare quelli di De Morgan, sono veri anche per "n" variabili

Esempio 1

- Semplificare la seguente espressione:

$$(x + z) \cdot (x + \bar{z}) \cdot (\bar{y} + z)$$

- In base ai teoremi visti si ha:

$$\begin{aligned} (x + z) \cdot (x + \bar{z}) \cdot (\bar{y} + z) &= (x + z \cdot \bar{z}) \cdot (\bar{y} + z) && \text{P 4b} \\ &= (x + 0) \cdot (\bar{y} + z) && \text{P 5b} \\ &= x \cdot (\bar{y} + z) && \text{P 2a} \end{aligned}$$

1a $x + 1 = 1$

1b $x \cdot 0 = 0$

2a $\overline{\overline{x}} = x$

3a $x + x = x$

3b $x \cdot x = x$

4a $x + xy = x$

4b $x(x + y) = x$

5a $x + \bar{x}y = x + y$

5b $x(\bar{x} + y) = xy$

6a $x + (y + z) = x + y + z$

6b $x(yz) = (xy)z = xyz$

7a $xy + \bar{x}z + yz = xy + \bar{x}z$

7b $(x + y)(\bar{x} + z)(y + z) = (x + y)(\bar{x} + z)$

8a $\overline{(x + y)} = \bar{x} \cdot \bar{y}$

8b $\overline{(x \cdot y)} = \bar{x} + \bar{y}$

1a Somma logica (+)

1b Prodotto logico ($\bullet$)

2a $x + 0 = x$

2b $x \bullet 1 = x$

3a $x + y = y + x$

3b $x \bullet y = y \bullet x$

4a $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

4b $x + (y \bullet z) = (x + y) \bullet (x + z)$

5a $x + \bar{x} = 1$

5b $x \bullet \bar{x} = 0$

Esempio 1'

- Per altra via; posto:
- si ha:

$$T = (x + \bar{z})$$

$$\begin{aligned}
 (x + z) \cdot (x + \bar{z}) \cdot (\bar{y} + z) &= (x + z) \cdot T \cdot (\bar{y} + z) \\
 &= (T \cdot x + T \cdot z) \cdot (\bar{y} + z) && \text{P 4b} \\
 &= (xx + xz + \bar{z}x + \bar{z}z) \cdot (\bar{y} + z) && \text{P 4a} \\
 &= (x + xz + \bar{z}x + \bar{z}z) \cdot (\bar{y} + z) && \text{P 3b} \\
 &= (x + 0) \cdot (\bar{y} + z) \\
 &= x \cdot (\bar{y} + z)
 \end{aligned}$$

1a $x + 1 = 1$

1b $x \bullet 0 = 0$

2a $\overline{(\bar{x})} = x$

3a $x + x = x$

3b $x \bullet x = x$

4a $x + x\bar{y} = x$

4b $x(x + y) = x$

5a $x + x\bar{y} = x + y$

5b $x(\bar{x} + y) = xy$

6a $x + (y + z) = x + y + z$

6b $x(yz) = (xy)z = xyz$

7a $xy + x\bar{z} + yz = xy + \bar{x}z$

7b $(x + y)(\bar{x} + z)(y + z) = (x + y)(\bar{x} + z)$

8a $\overline{(x + y)} = \bar{x} \cdot \bar{y}$

8b $\overline{(x \cdot y)} = \bar{x} + \bar{y}$

1a Somma logica (+)

1b Prodotto logico ($\bullet$)

2a $x + 0 = x$

2b $x \bullet 1 = x$

3a $x + y = y + x$

3b $x \bullet y = y \bullet x$

4a $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

4b $x + (y \bullet z) = (x + y) \bullet (x + z)$

5a $x + \bar{x} = 1$

5b $x \bullet \bar{x} = 0$

Premessa 1

- Osservazioni

- le funzioni AND, OR e NOT costituiscono un insieme *funzionalmente completo* di operatori logici
- In base al teorema di De Morgan si ha:

$$x + y = \overline{\overline{x} \cdot \overline{y}}$$

- ovvero la funzione OR si può realizzare con le funzioni AND e NOT quindi:
- le funzioni **AND e NOT** costituiscono un insieme *funzionalmente completo* di operatori logici

Premessa 2

- Osservazioni

- Sempre in base al teorema di De Morgan si ha:

$$x \cdot y = \overline{(\overline{x} + \overline{y})}$$

- ovvero la funzione AND si può realizzare con le funzioni OR e NOT quindi
 - le funzioni **OR e NOT** costituiscono un insieme *funzionalmente completo* di operatori logici
 - le funzioni OR e AND **non** costituiscono un insieme *funzionalmente completo* di operatori logici perché non è possibile realizzare la funzione NOT

Definizione

- Le funzioni NAND e NOR sono definite dalle seguenti tabelle di verità

x	y	u
0	0	1
0	1	1
1	0	1
1	1	0

$$\text{NAND} \quad u = \overline{(x \cdot y)}$$

x	y	u
0	0	1
0	1	0
1	0	0
1	1	0

$$\text{NOR} \quad u = \overline{(x + y)}$$

Osservazioni

- NAND e NOR sono contrazioni di NOT-AND e NOT-OR
- la funzione **NAND** costituisce un insieme *funzionalmente completo* di operatori logici

$$\overline{(x \cdot x)} = \bar{x} \quad \overline{(\overline{(x \cdot y)})} = x \cdot y$$

- la funzione **NOR** costituisce un insieme *funzionalmente completo* di operatori logici

$$\overline{(x + x)} = \bar{x} \quad \overline{(\overline{(x + y)})} = x + y$$

Funzioni "complesse" 1

- L'operatore "XOR", OR ESCLUSIVO è:

$$x \oplus y$$

- Definizione

x	y	u
0	0	0
0	1	1
1	0	1
1	1	0

$$x \oplus y = x \cdot \bar{y} + \bar{x} \cdot y = (x + y) \cdot (\bar{x} + \bar{y}) = (x + y) \cdot \overline{(x \cdot y)}$$

Funzioni "complesse" 2

- L'operatore "XNOR", NOR ESCLUSIVO è:

$$\overline{x \oplus y}$$

- Definizione

x	y	u
0	0	1
0	1	0
1	0	0
1	1	1

$$\overline{x \oplus y} = x \cdot y + \bar{x} \cdot \bar{y} = (x + \bar{y}) \cdot (\bar{x} + y)$$

Sintesi ad occhio di Reti Combinatorie

Ciclo

- **Definizione**

- Ciclo: Percorso chiuso che attraversa k blocchi ($k \geq 1$) tutti nella loro direzione di funzionamento

- **Osservazioni**

- Tutte le reti viste sono prive di cicli
- I blocchi base combinatori sono privi di cicli
- Le funzioni descrivibili dalle tabelle di verità sono tutte prive di cicli (le uscite sono funzione dei solo ingressi)

- **Conclusione**

- Tutte le reti logiche composte di blocchi combinatori e prive di cicli sono reti combinatorie

Sintesi di reti combinatorie

- **Sintesi**

- data la descrizione ai terminali di una rete combinatoria
- ottenere la struttura in blocchi logici e le relative interconnessioni

- **Osservazioni**

- il funzionamento della rete deve essere possibile descriverlo mediante una tabella di verità
- non esiste una sola realizzazione
- per poter scegliere fra le varie soluzioni è necessario definire il parametro da ottimizzare
- Funzione **COSTO**
- (numero di blocchi base, ritardo ingresso uscita, uso di particolari blocchi, ...)

- **VEDERE ESEMPI SUCCESSIVI**

Esempio di funzione

- Data la funzione definita dalla Tabella di Verità:

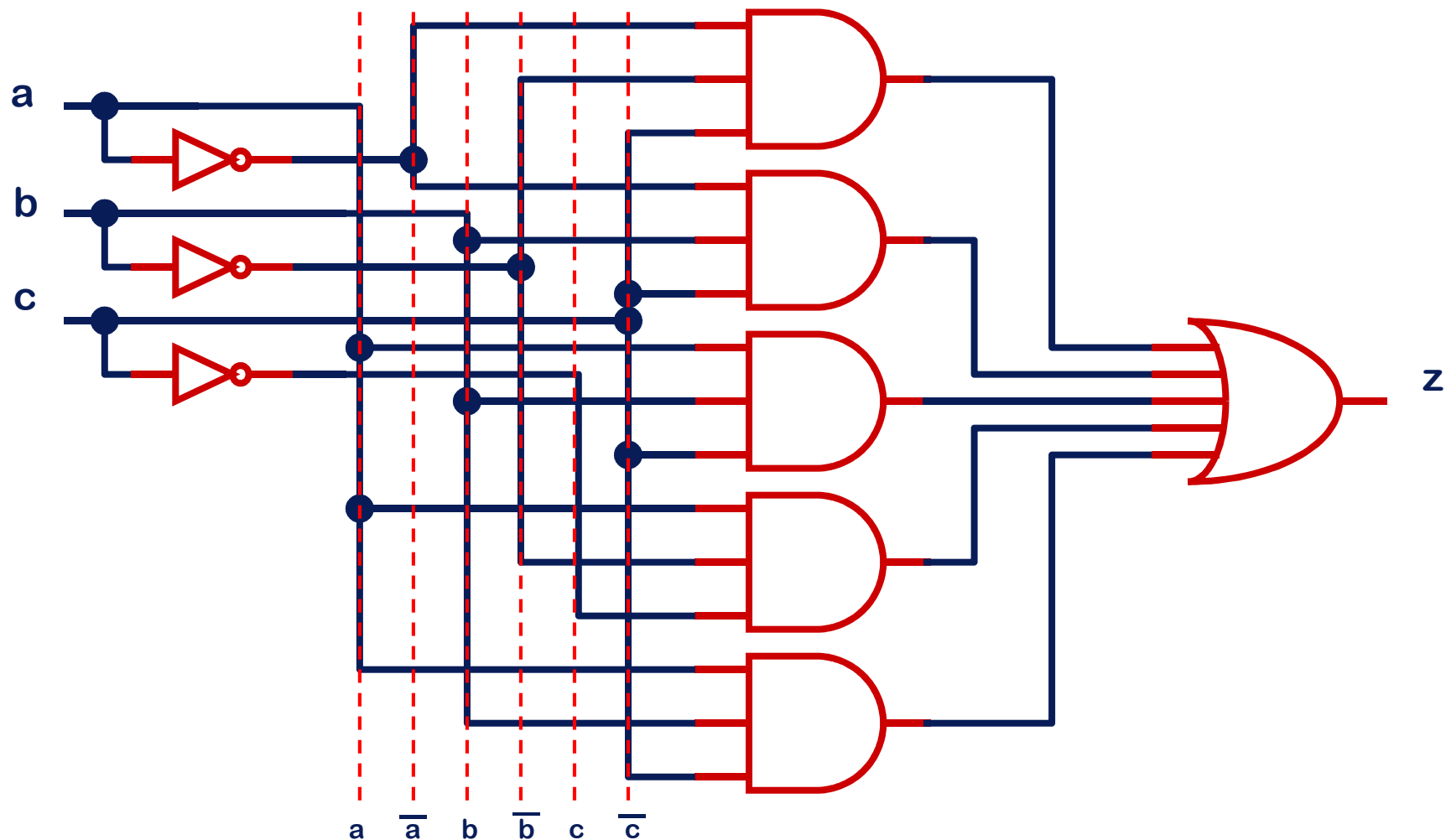
Si ha:

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$\begin{aligned}
 z &= \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c} \\
 &= \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot \bar{b} \cdot \bar{c} \\
 &= \bar{c} + a \cdot \bar{b} = \overline{c \cdot (a \cdot \bar{b})} = \overline{c \cdot (a + b)} = \overline{c \cdot a} + \overline{c \cdot b} \\
 &= \overline{(a + c)} + \overline{(c + b)} \\
 z &= (a + b + \bar{c}) \cdot (a + \bar{b} + \bar{c}) \cdot (\bar{a} + \bar{b} + \bar{c})
 \end{aligned}$$

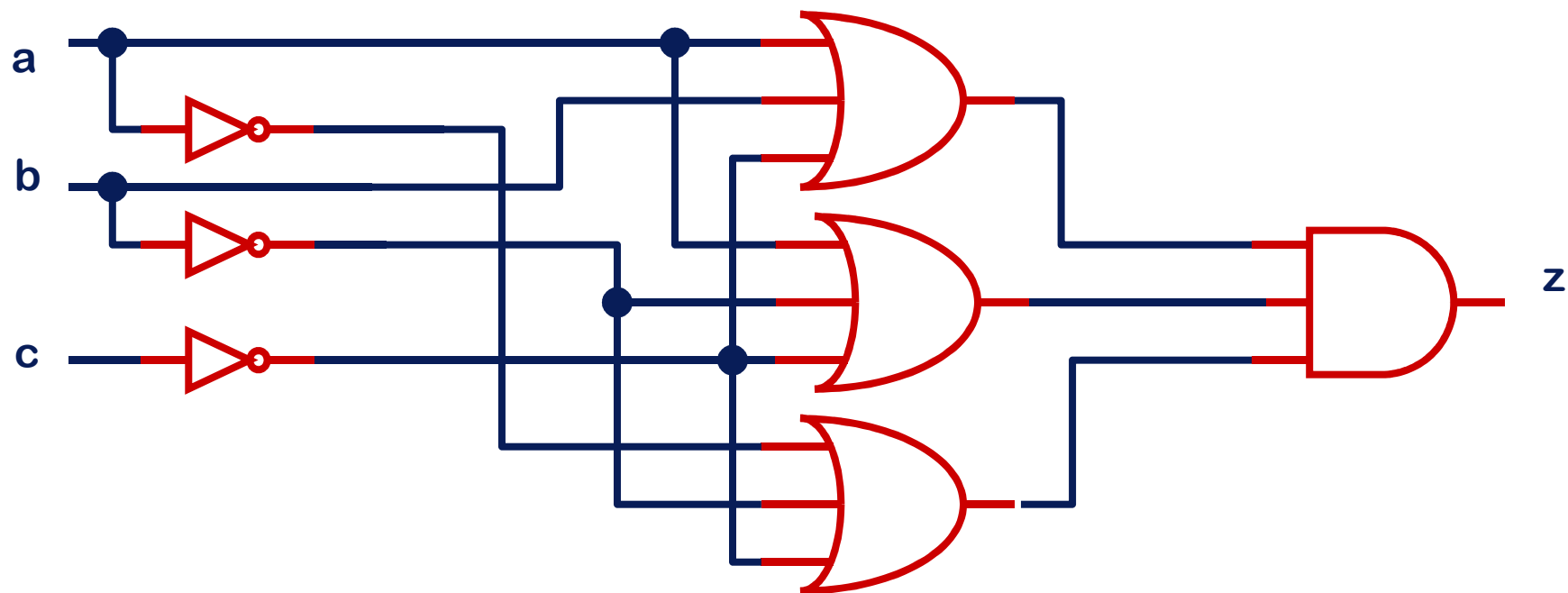
Schemi relativi 1

$$z = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c}$$



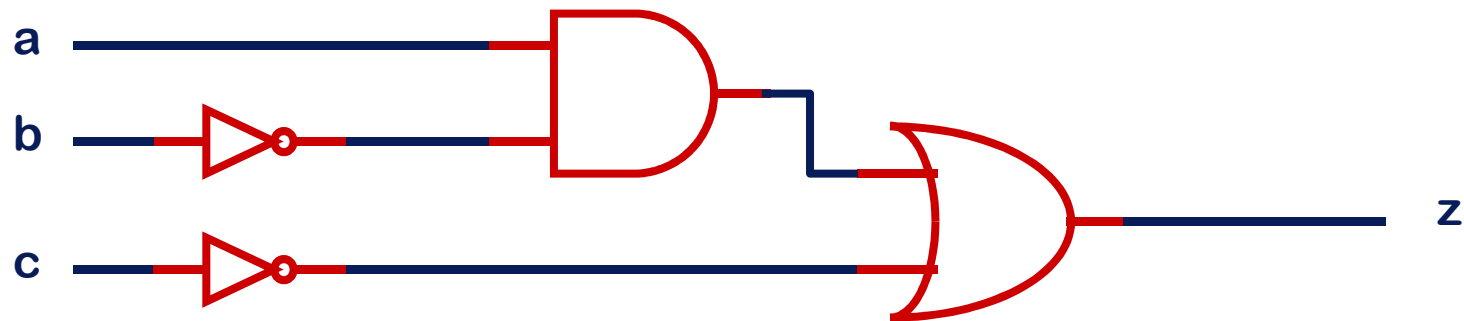
Schemi relativi 2

$$z = (a + b + \bar{c}) \cdot (a + \bar{b} + \bar{c}) \cdot (\bar{a} + \bar{b} + \bar{c})$$



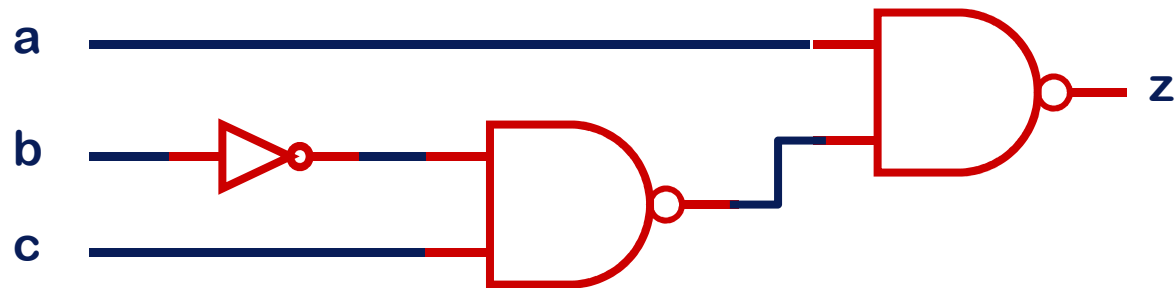
Schemi relativi 3

$$z = \bar{c} + a \cdot \bar{b}$$

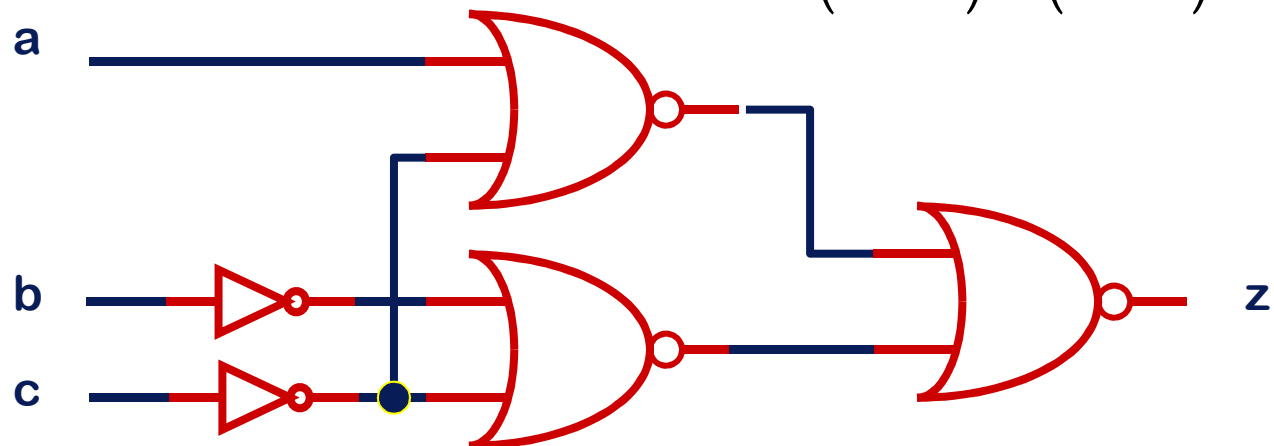


Schemi relativi 4

$$z = \overline{c \cdot (a \cdot \bar{b})}$$



$$z = \overline{(a + c) + (c + \bar{b})}$$



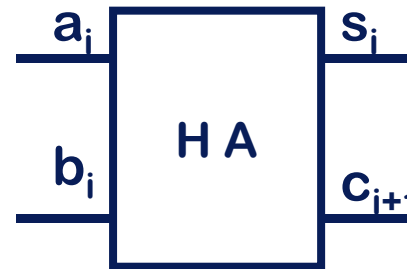
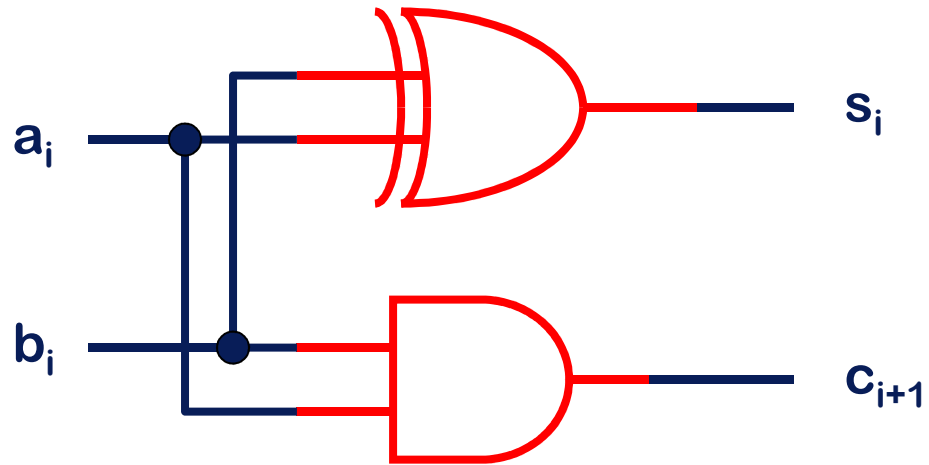
Half Adder

- Somma di due bit (senza riporto in ingresso)

a_i	b_i	s_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_i = a_i \oplus b_i$$

$$c_{i+1} = a_i \cdot b_i$$



Full Adder 1

- Somma di due bit compreso il Carry

c_i	a_i	b_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

s_i

$c_i \backslash a_i, b_i$	00	01	11	10
0		1		1
1	1		1	

c_{i+1}

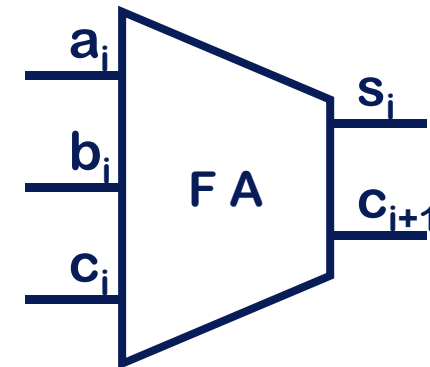
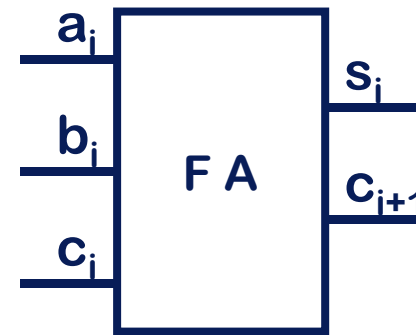
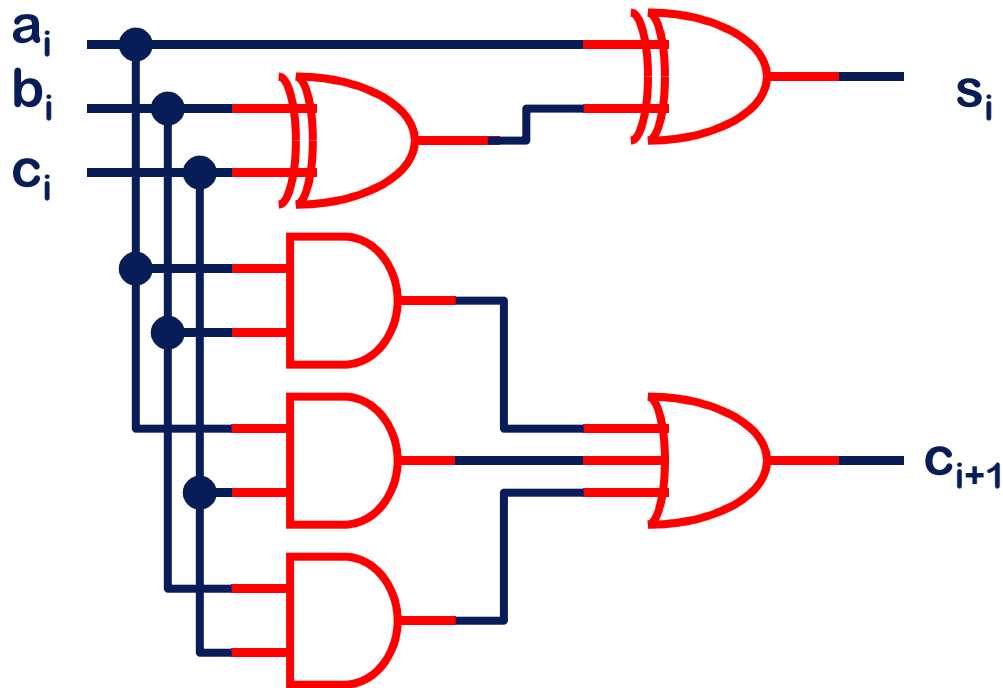
$c_i \backslash a_i, b_i$	00	01	11	10
0			1	
1		1	1	1

Full Adder 2

- Lo schema risulta

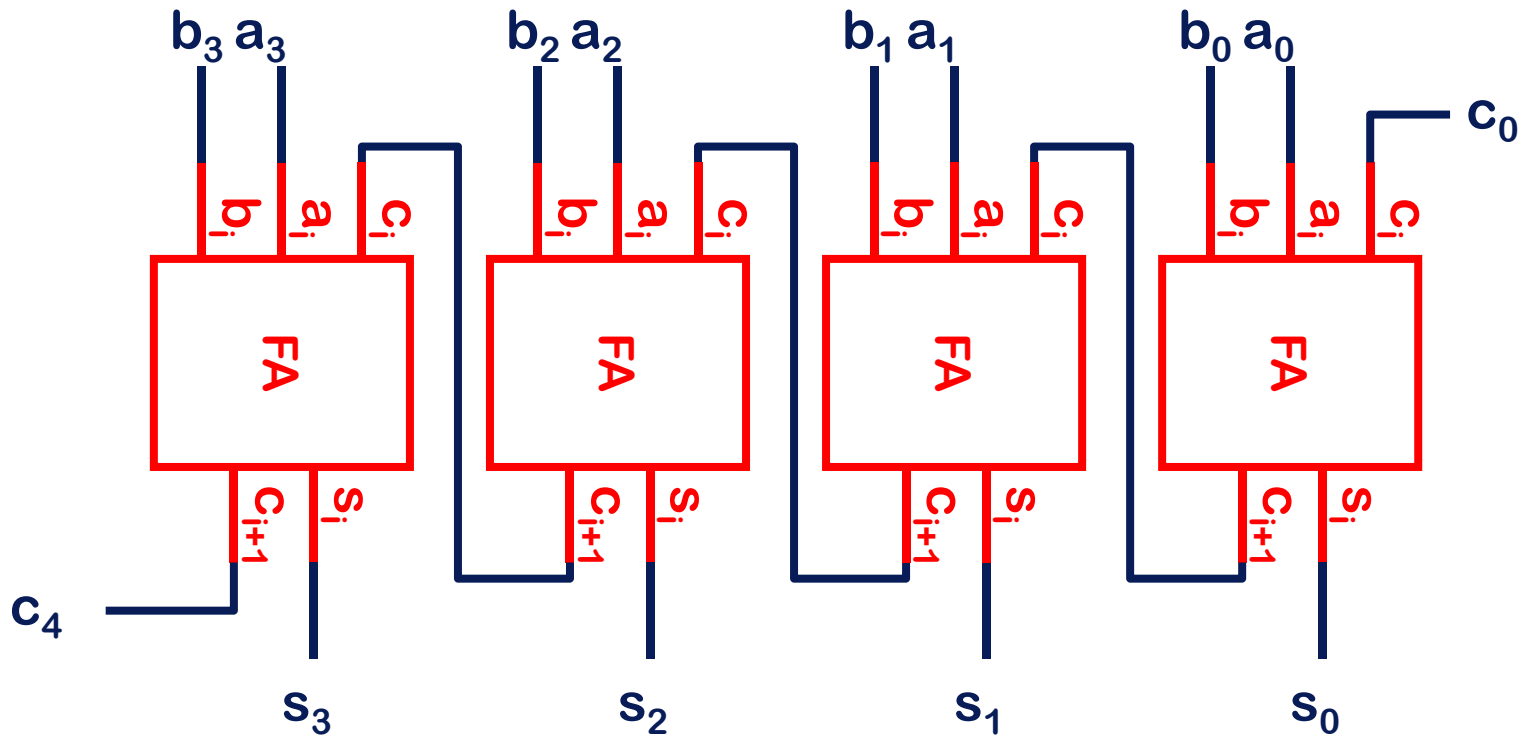
$$s_i = a_i \oplus b_i \oplus c_i = a_i \oplus (b_i \oplus c_i)$$

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$



Sommatore a riporto seriale (Ripple-Carry Adder)

- Somma di due parole di 4 bit in $C. 2$



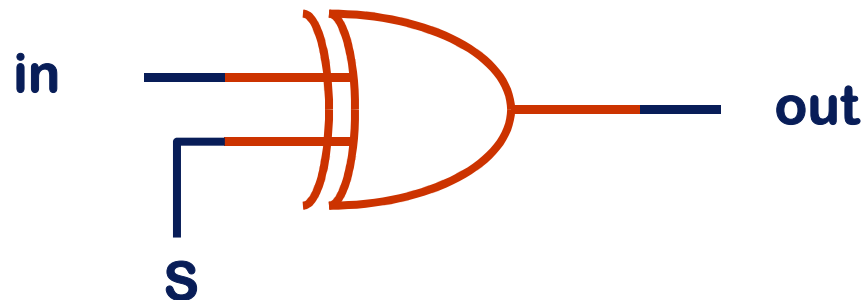
Proprietà dello XOR

- Lo XOR può essere visto come un inverter "programmabile"

S	in	out
0	0	0
0	1	1
1	0	1
1	1	0

per $S = 0$ è $out = in$

per $S = 1$ è $out = \overline{in}$



Considerazioni sulla sottrazione

- Si ricorda che

$$W = A - B = A + (-B)$$

- Operando in complemento a 2 si ha

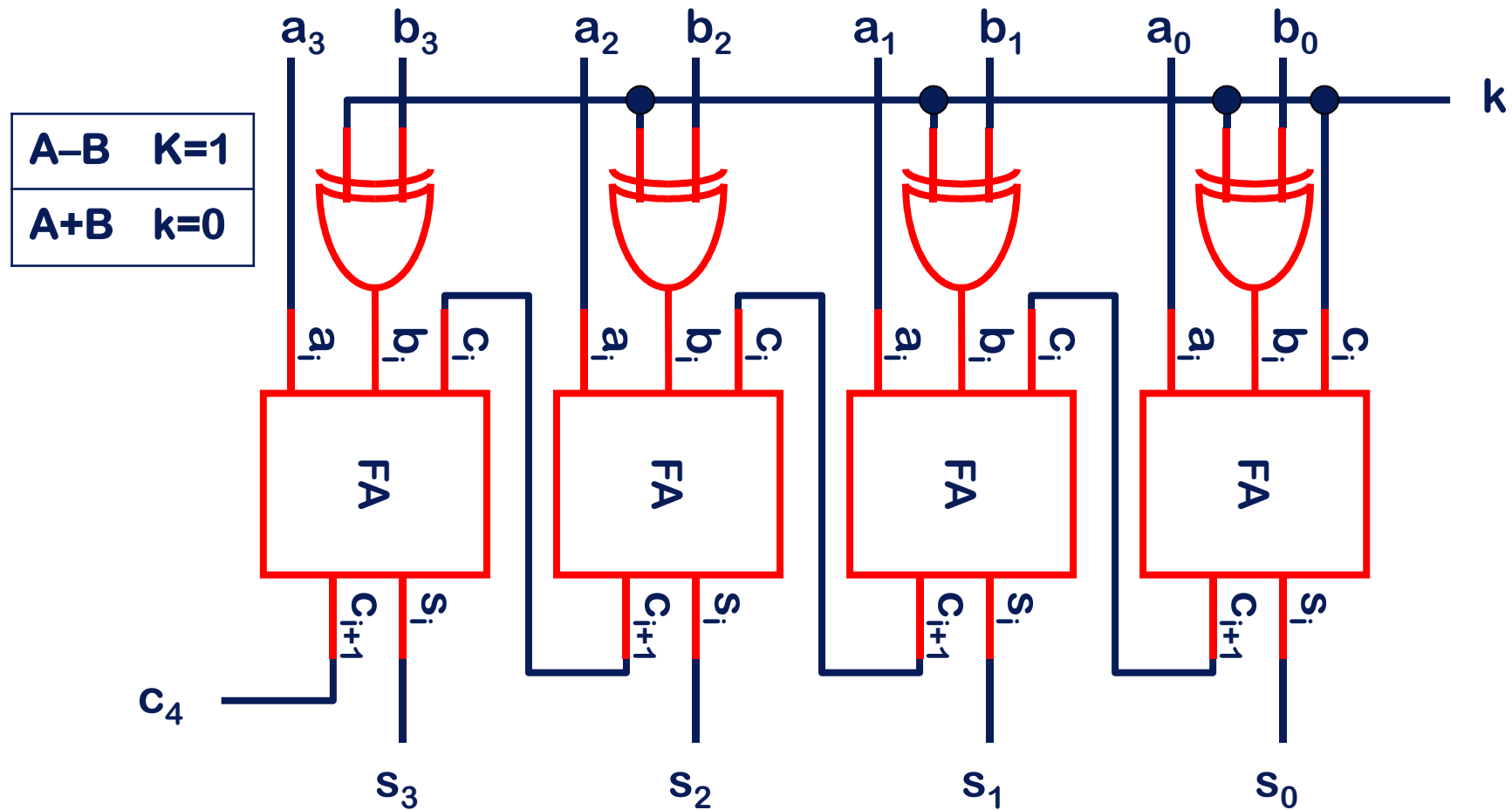
$$-B = (\overline{B}) + 1$$

- Quindi

$$W = A - B = A + (\overline{B}) + 1$$

Sommatore/Sottrattore

- In base alle proprietà dello XOR e come si può eseguire la differenza ($A - B$) in $C. 2$ si ha:



Conclusioni

- Postulati
- Principio di dualità
- Teoremi fondamentali
- insieme *funzionalmente completo* NAND e NOR
- Funzione XOR
- Reti logiche combinatorie e sequenziali
- Simboli
- Concetto di minimizzazione (funzione costo)
- Realizzazioni diverse della stessa funzione
- Half Adder e Full Adder
- Sommatore di due word di n bit

LEZIONE N° 91

Introduzione agli elementi architettonici principali

FORME STANDARD DI FUNZIONI BOOLEANE

Dalle tabelle di verita' alla sintesi della rete logica

- Una funzione booleana puo' essere definita tramite **tabella di verita'**

- Esempio

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

- E' possibile derivare una espressione algebrica a partire dalla tabella di verita' in (almeno) due modi:

- Identificando tutte le righe che producono un 1 e sommando (OR) i corrispondenti termini (prodotti ovvero AND di variabili) che producono tale 1. Tali «termini prodotto» prendono il nome di **mintermini**. L'espressione risultante e' detta **somma di prodotti**. Esempio:

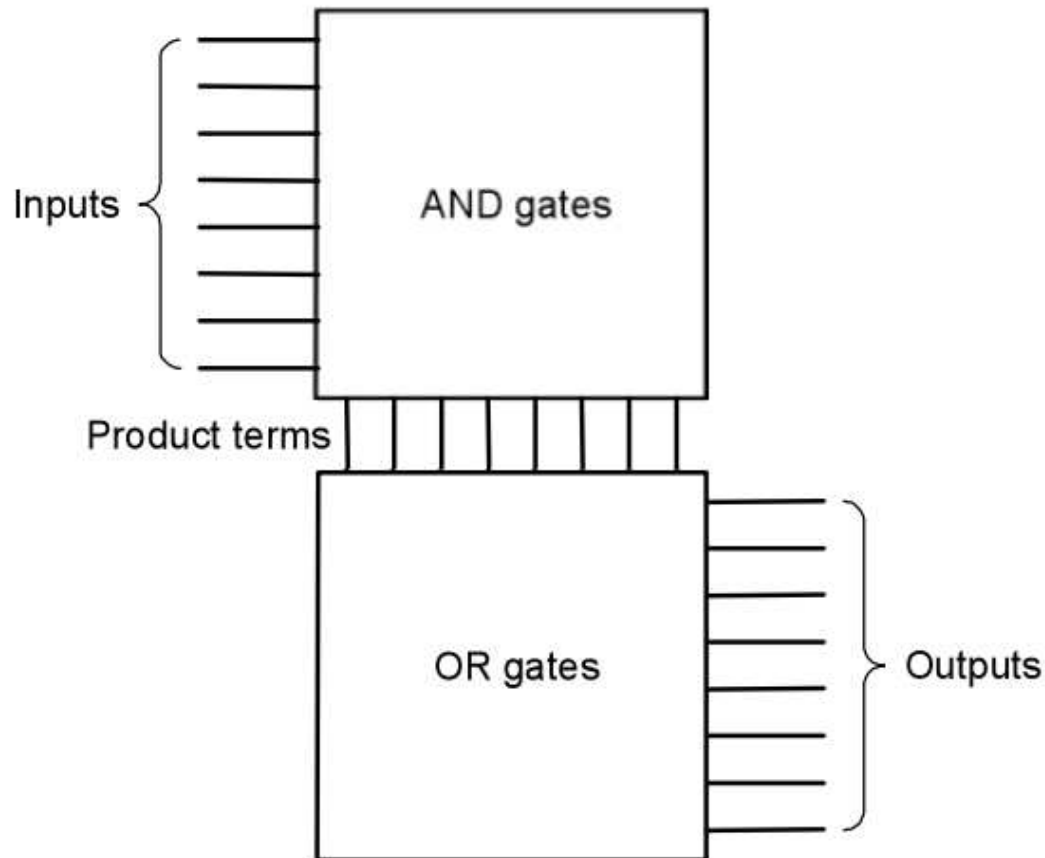
$$z = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot \bar{c} + a \cdot \bar{b} \cdot \bar{c} + a \cdot \bar{b} \cdot c + a \cdot b \cdot \bar{c}$$

- Identificando tutte le righe che producono uno 0 e facendo il prodotto (AND) dei corrispondenti termini (somme ovvero OR di variabili) che producono tale 0. Tali «termini somma» prendono il nome di **maxtermini**. L'espressione risultante e' detta **prodotto di somme**. Esempio:

$$z = (a + b + \bar{c}) \cdot (a + \bar{b} + \bar{c}) \cdot (\bar{a} + \bar{b} + \bar{c})$$

Programmable Logic Array - PLA

- Dato che si può sempre rappresentare una funzione logica come tabella di verità, si può sempre rappresentare anche come somma di prodotti (o prodotto di somme)
- Questo conduce ad una tecnica di implementazione universale nota come PLA



Le due matrici AND e OR possono essere anche rese "programmabili" (vedere lezioni successive)

Mappe di Karnaugh

- **Puo' essere comodo semplificare l'espressione booleana**
 - Per ottenere espressioni booleane piu' semplici e facili da verificare
 - Per sintetizzare una rete logica con un minor numero di porte (minor area)
- **La Mappa di Karnaugh e' una rappresentazione alternativa della tabella di verita' che, sfruttando la proprieta' di assorbimento*, permette rapidamente di identificare termini piu' semplici**
- **La Mappa di Karnaugh si ottiene:**
 - disponendo gli "indici" generati dalle variabili di ingresso in modo che due indici adiacenti differiscano per una sola cifra. Esempio:

a	b	c	z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

		a, b			
		00	01	11	10
z	c	1	1	1	1
	$\bar{c}$	0	0	0	1

$a \cdot \bar{b}$

Nota: la mappa si svolge in realta' su una superficie toroidale → I sottocubi possono continuare su celle adiacenti la parte opposta

	00	01	11	10
0				
1	1			1

Diagram illustrating the toroidal nature of the Karnaugh map. The map is a 2x4 grid. The top row (z=0) contains 0s in columns 00, 01, 11, and 10. The bottom row (z=1) contains 1s in columns 00 and 10, and 0s in columns 01 and 11. A green box highlights the 1s in the bottom row, columns 00 and 10, indicating they are adjacent in the toroidal sense.

- Localizzando i gruppi adiacenti di 1 con un numero di 1 pari a una potenza di 2 (detti sottocubi di ordine k se contengono 2^k uni)
- Un termine e' identificato dagli indici che rimangono "fissi" nel sottocubo. Esempio:

$$z = \bar{c} + a \cdot \bar{b}$$

Mappe di Karnaugh a 2, 3, 4, 5 variabili

	0	1
0	1	1
1		

2 variabili

	00	01	11	10
0				
1		1	1	

3 variabili

	00	01	11	10
00				
01		1	1	
11		1	1	
10				

4 variabili

	00	01	11	10
00	1			
01				
11				
10				

5 variabili (richiede una visione "multidimensionale")

Note sulle Mappe di Karnaugh

- Se la mappa contiene dei non-specificato (X o -)
 - Si puo' interpretare come e' piu' conveniente per ottenere sottocubi piu' grandi
- Se ci sono piu' possibilita' con sottocubi piu' piccoli e piu' grandi
 - Si preferiscono sempre coperture con sottocubi piu' grandi (hanno meno ingressi)
- Si prendono solo i termini strettamente necessari per ottenere una **lista di copertura "a costo minimo"**

Karnaugh, M. "A Map method for Synthesis of Combinational Logic",
Transactions of AIEE, Communication and Electronics, 72, part I, Nov. 1953, pp. 593-99

RETI COMBINATORIE NOTEVOLI

Decoder

Encoder

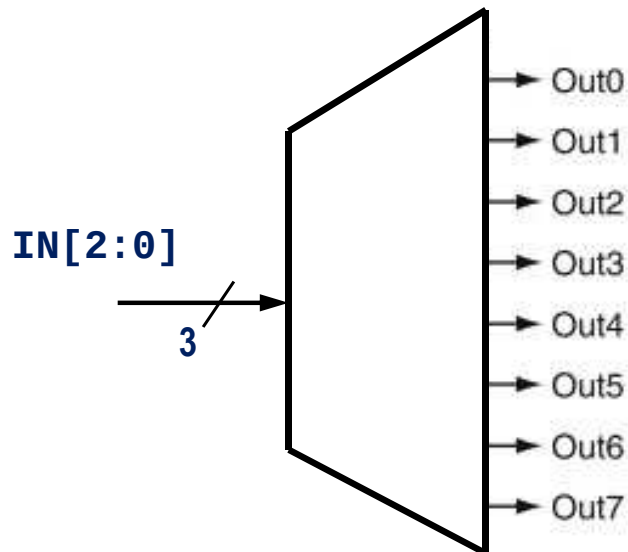
Encoder con priorit 

Multiplexer

Demultiplexer

Look-Up-Table (LUT)

Decoder



a. A 3-bit decoder

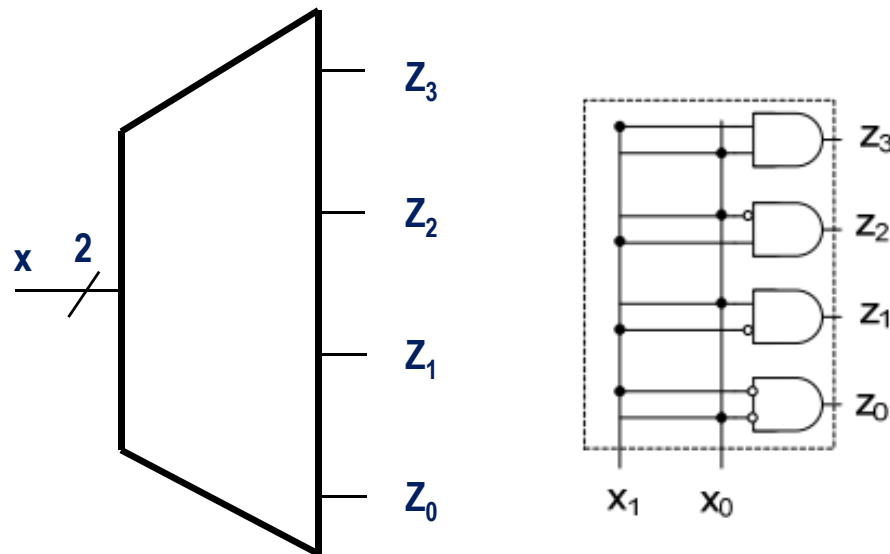
Inputs			Outputs							
IN2	IN1	IN0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

- Esempio di decoder a 3 bit o “da-3-a-8” (v. PH5, appendice B.3)
- Un solo filo di uscita vale 1 (gli altri valgono tutti 0): quello di indice pari all'intero “posto sui fili” di ingresso
- In generale, un decoder da-n-a- 2^n ha n ingressi e 2^n uscite

Realizzazione del decoder

- Si può sintetizzare in termini di porte logiche con 2^n porte AND ognuna delle quali riconosce uno dei 2^n mintermini



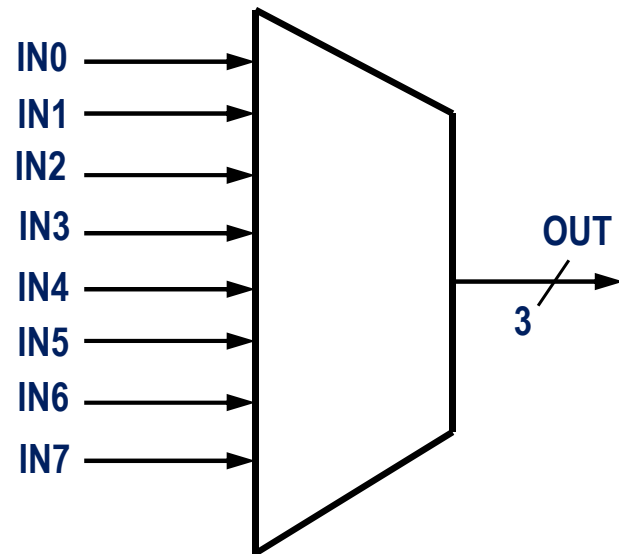
Es. decoder da-2-a-4

Nota: in realtà possono esistere implementazioni fisiche CMOS diverse e più efficienti dell'elemento architeturale "decoder"

Nota2: può anche avere un segnale di abilitazione; si realizza mettendo ogni uscita in AND con tale segnale di abilitazione

Encoder

- L'encoder effettua l'operazione inversa del decoder
- Posto un 1 sul k-esimo dei 2^n ingressi (gli altri sono posti a 0): le n uscite producono i bit corrispondenti alla rappresentazione in base due di k



IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0	OUT2	OUT1	OUT0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Nota: questa NON E' una tabella della verita'; la si puo' interpretare come tabella di verita' immaginando che le mancanti 56 righe le uscite sono indeterminate (X o "don't care")

Realizzazione dell'encoder

- $OUT0 = IN1 + IN3 + IN5 + IN7$
- $OUT1 = IN2 + IN3 + IN6 + IN7$
- $OUT2 = IN4 + IN5 + IN6 + IN7$
- Dunque tale encoder puo' essere realizzato con 3 porte OR a 4 ingressi

Esercizio per casa: ricavare queste equazioni dalle mappe di Karnaugh

Priority Encoder

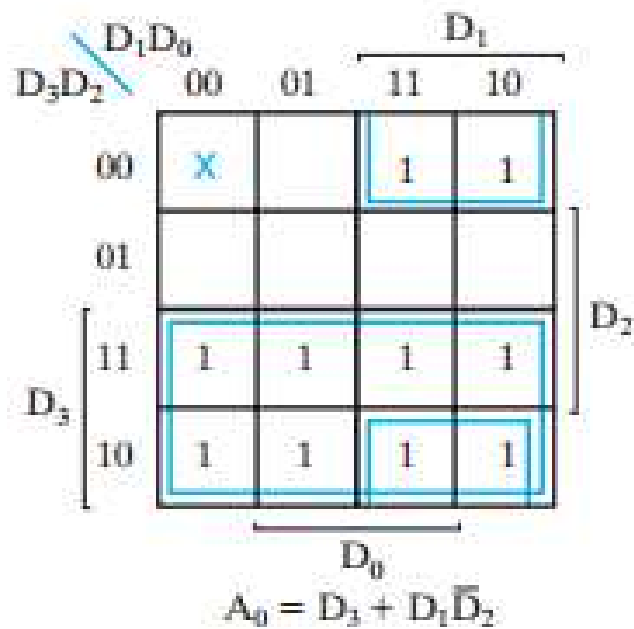
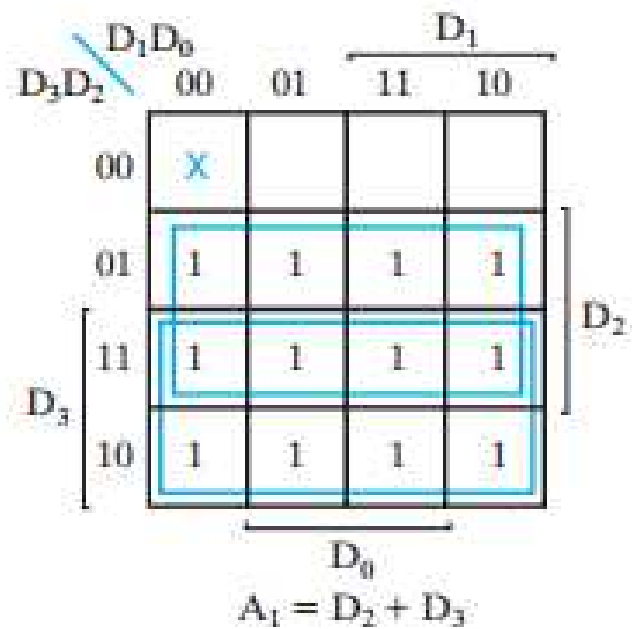
- Questa e' una rete diversa dall'encoder semplice

Inputs				Outputs		
D ₃	D ₂	D ₁	D ₀	A ₁	A ₀	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

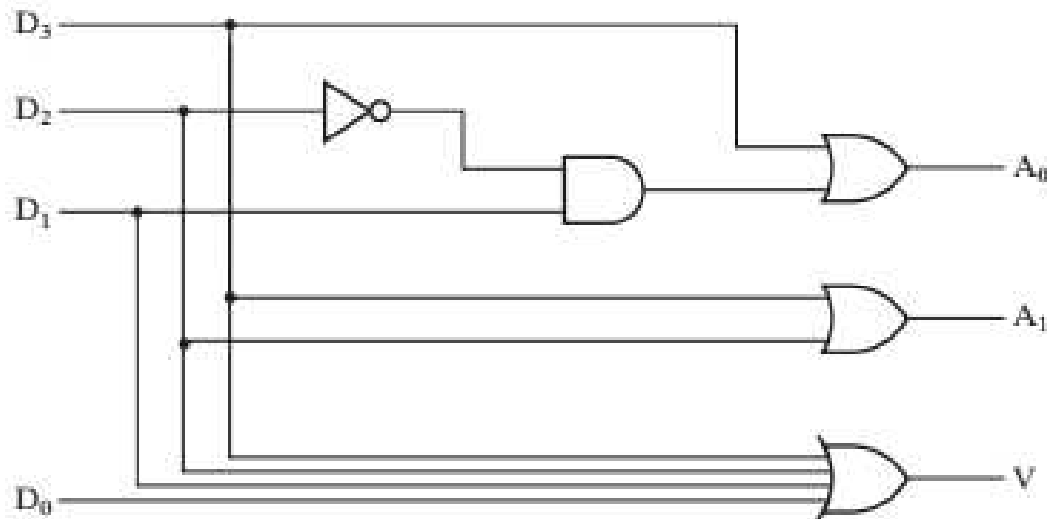
Tabella di verita' completa

- Quando D3 vale 1 tutti gli altri bit non contano e l'uscita indica la massima priorita' $(11)_2$ ovvero 3
- Quando D3 vale 0 ma D2 vale 1, gli altri bit a destra non contano e l'uscita indica priorita' 2 e cosi' via...
- Il bit V e' necessario per indicare se l'uscita e' valida ovvero se almeno un 1 e' stato posto in ingresso

Realizzazione del Priority Encoder

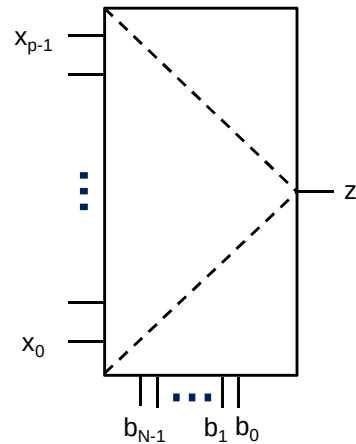


$$\begin{aligned} A_0 &= D_3 + D_1\bar{D}_2 \\ A_1 &= D_2 + D_3 \\ V &= D_0 + D_1 + D_2 + D_3 \end{aligned}$$



Multiplexer

- Il multiplexer serve per selezionare un'informazione



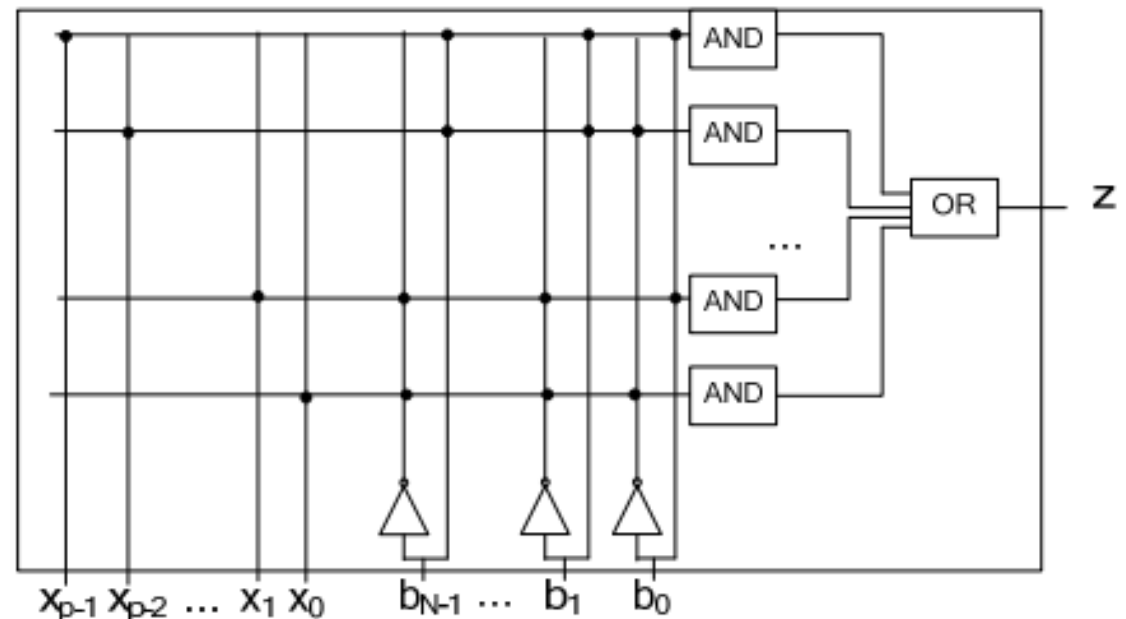
Multiplexer con 2^N ingressi

- Ha $2^N + N$ ingressi e 1 uscita
- Gli N ingressi b sono detti variabili di comando
- Specificando $B=i$ viene selezionato uno (quello con indice i) dei $2^N = p$ ingressi x

$$z = x_i \Leftrightarrow (b_{N-1} \dots b_1 b_0)_2 \equiv i$$

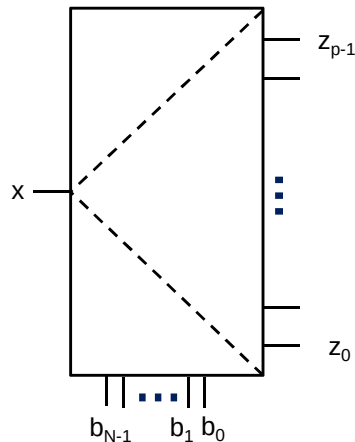
AND con $N+1$ ingressi

$$\begin{aligned}
 z = & x_0 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0} + \\
 & x_1 \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot b_0 + \\
 & \dots \\
 & x_{p-2} \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot \overline{b_0} + \\
 & x_{p-1} \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0
 \end{aligned}$$



Demultiplexer

- Realizza la funzione inversa del multiplexer, ovvero data un'informazione x , la distribuisce su una di 2^N uscite possibili



Demultiplexer con 2^N uscite

- Ha $1+N$ ingressi e 2^N uscite
- Gli N ingressi b sono detti variabili di comando
- Specificando $B=i$ viene inviato l'ingresso x su una (quella con indice i) delle $2^N=p$ uscite z
- Le altre uscite valgono 0

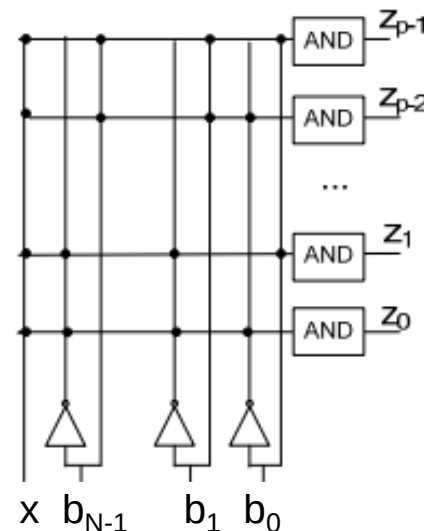
$$z_0 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot \overline{b_0}$$

$$z_1 = x \cdot \overline{b_{N-1}} \cdot \overline{b_{N-2}} \cdot \dots \cdot \overline{b_1} \cdot b_0$$

...

$$z_{p-2} = x \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot \overline{b_0}$$

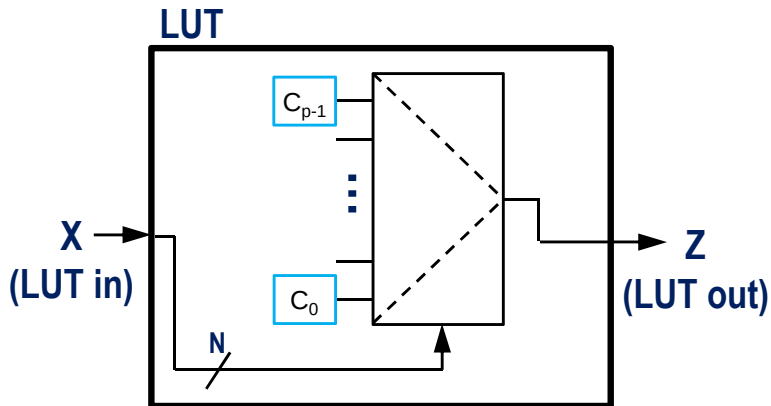
$$z_{p-1} = x \cdot b_{N-1} \cdot b_{N-2} \cdot \dots \cdot b_1 \cdot b_0$$



Coincide col decoder con abilitazione

Look Up Table (LUT)

- E' un Multiplexer in cui gli ingressi sono costanti



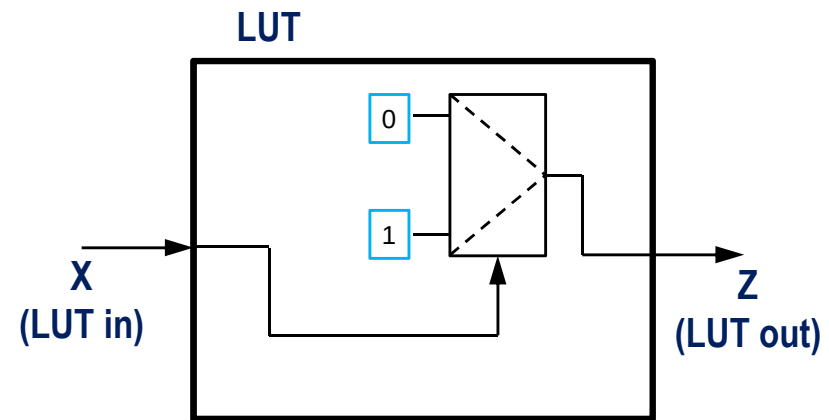
- Ha $2^N + N$ ingressi e 1 uscita
- Gli N ingressi X fungono da variabili di comando
- Specificando $X=i$ viene selezionato uno (quello con indice i) dei $2^N = p$ ingressi (FISSI)

- E' utile per realizzare una funzione booleana generica
- Esempio realizzazione della funzione "inverter" tramite LUT

X	Z
0	1
1	0

Profondita' (depth)

Ingressi Larghezza (width)



LUT larga 4 e profonda 8

- I segnali di controllo del MUX sono usati per selezionare le possibili uscite della funzione booleana

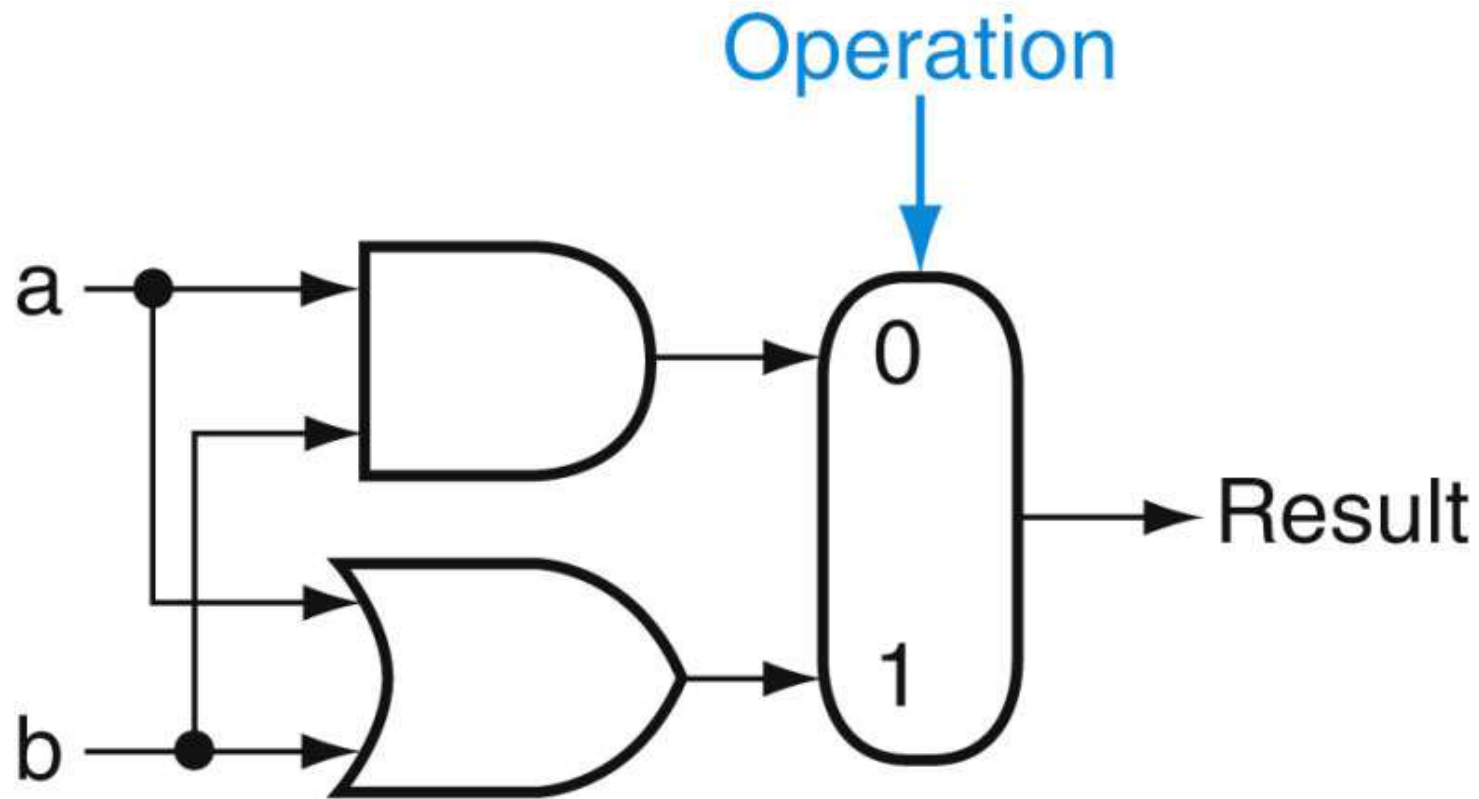
A0	A1	A2	Out0	Out1	Out2	Out3
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Diagram illustrating the structure of a 4-input, 8-deep LUT (Look-Up Table). The table has 8 rows and 7 columns. The first three columns are labeled A0, A1, and A2, representing the address inputs. The last four columns are labeled Out0, Out1, Out2, and Out3, representing the possible outputs of the boolean function. A bracket on the right side of the table indicates the depth (8 rows). A bracket at the bottom indicates the width (7 columns). The text "Stored Information" is written diagonally across the table, indicating the data stored in the LUT.

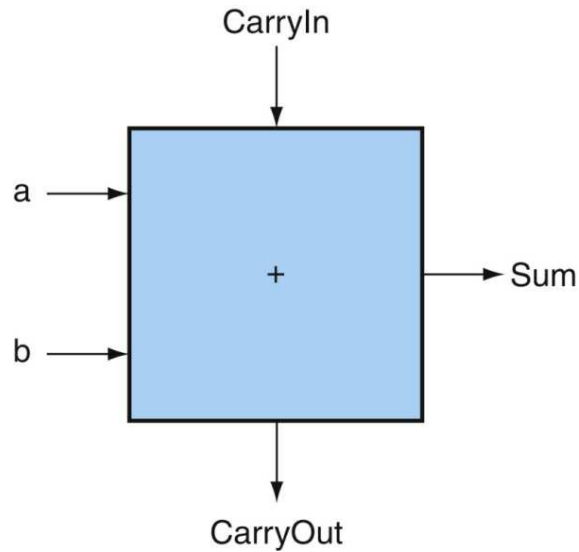
- Vedremo successivamente come rendere tali bit di ingresso "programmabili"
- Le FPGA forniscono un determinato numero di LUTs per realizzare funzioni piu' complesse (anche soft-processors !)

Relizzazione di ALU a 1-bit

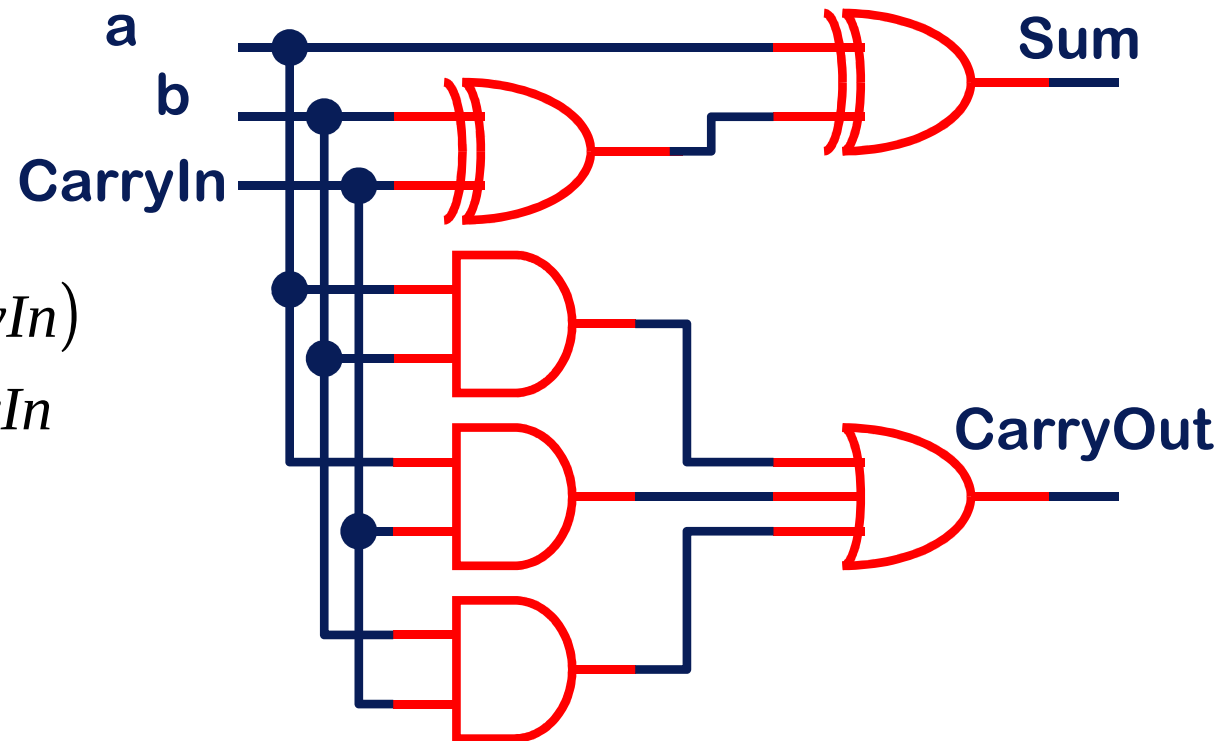
- AND e OR



Full Adder a 1 bit



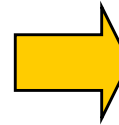
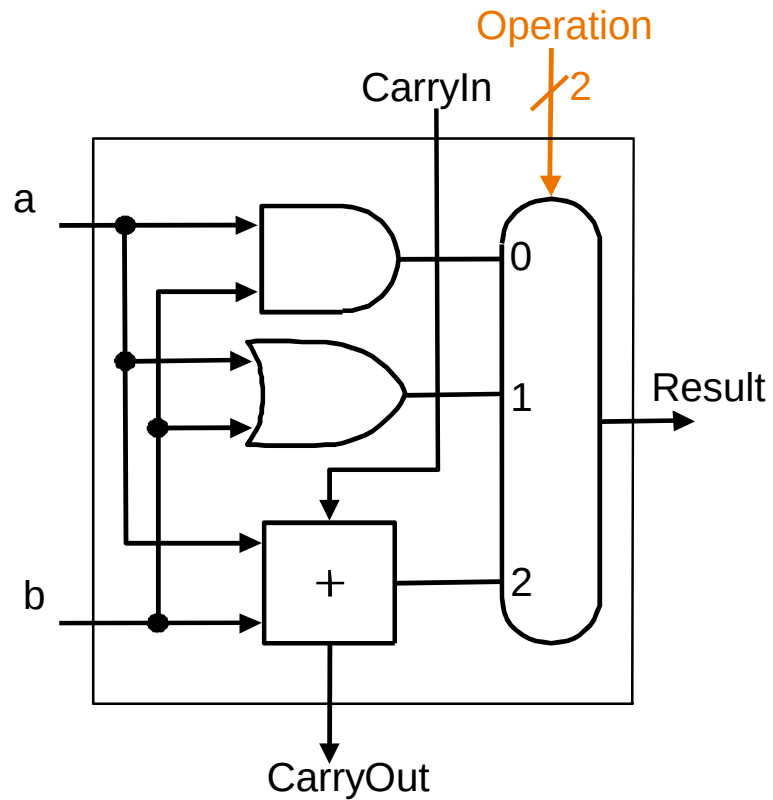
Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{\text{two}}$
0	0	1	0	1	$0 + 0 + 1 = 01_{\text{two}}$
0	1	0	0	1	$0 + 1 + 0 = 01_{\text{two}}$
0	1	1	1	0	$0 + 1 + 1 = 10_{\text{two}}$
1	0	0	0	1	$1 + 0 + 0 = 01_{\text{two}}$
1	0	1	1	0	$1 + 0 + 1 = 10_{\text{two}}$
1	1	0	1	0	$1 + 1 + 0 = 10_{\text{two}}$
1	1	1	1	1	$1 + 1 + 1 = 11_{\text{two}}$



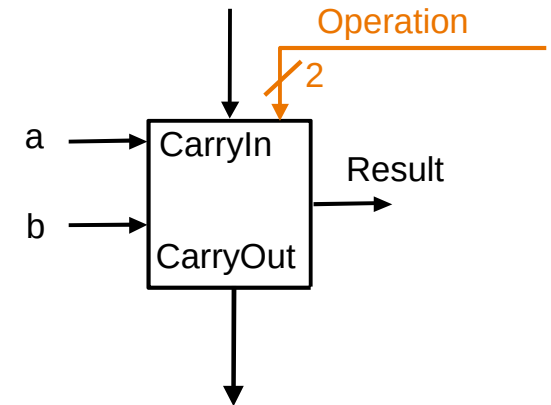
$$\text{sum} = a \oplus b \oplus \text{CarryIn} = a \oplus (b \oplus \text{CarryIn})$$

$$\text{CarryOut} = a \cdot b + a \cdot \text{CarryIn} + b \cdot \text{CarryIn}$$

Semplice ALU a 1-bit

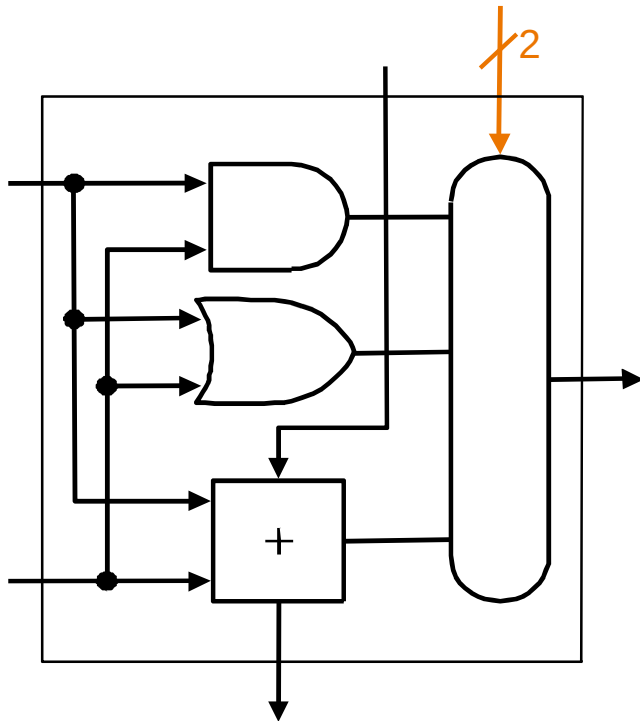


alu a 1-bit

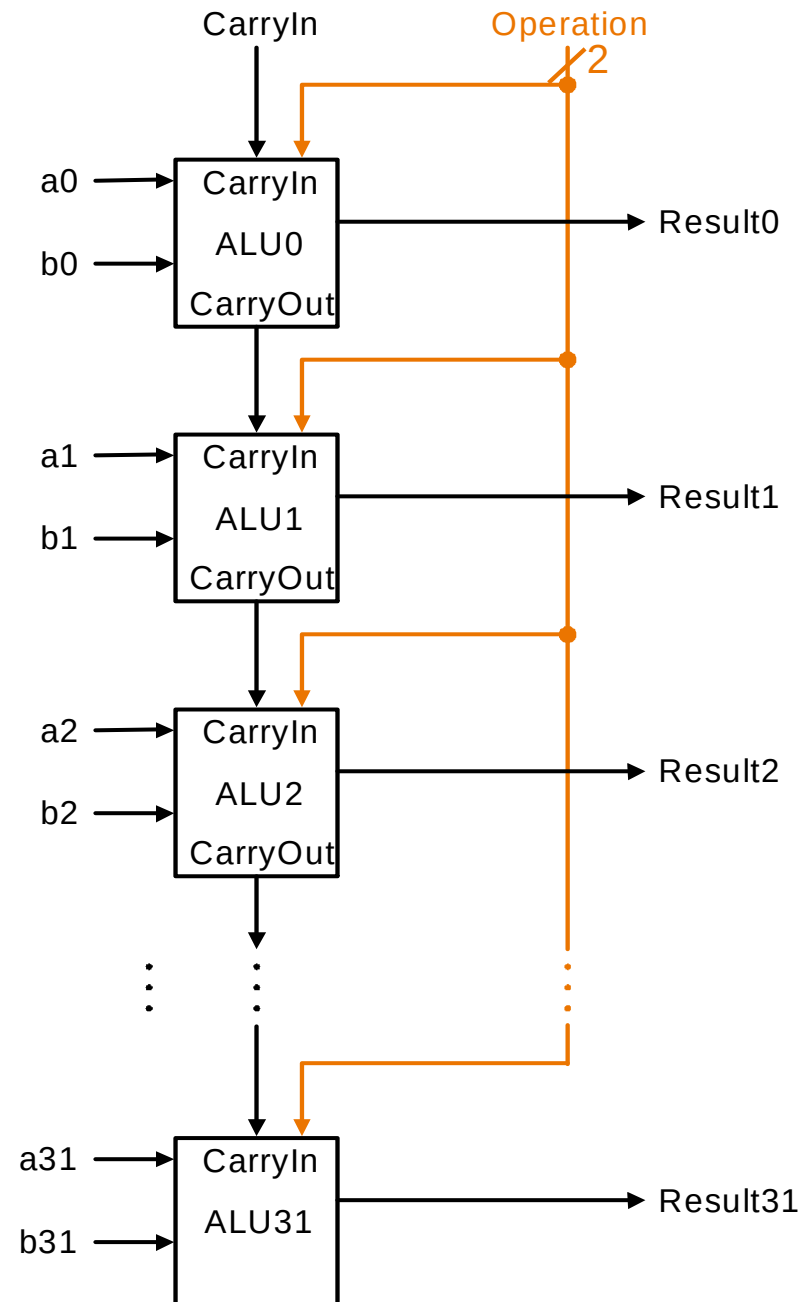


Operation	Function
0	AND
1	OR
2	ADD

Costruire una ALU a 32 bit



Problema:
il risultato e' disponibile
solo dopo che ALU31
ha ricevuto il carry

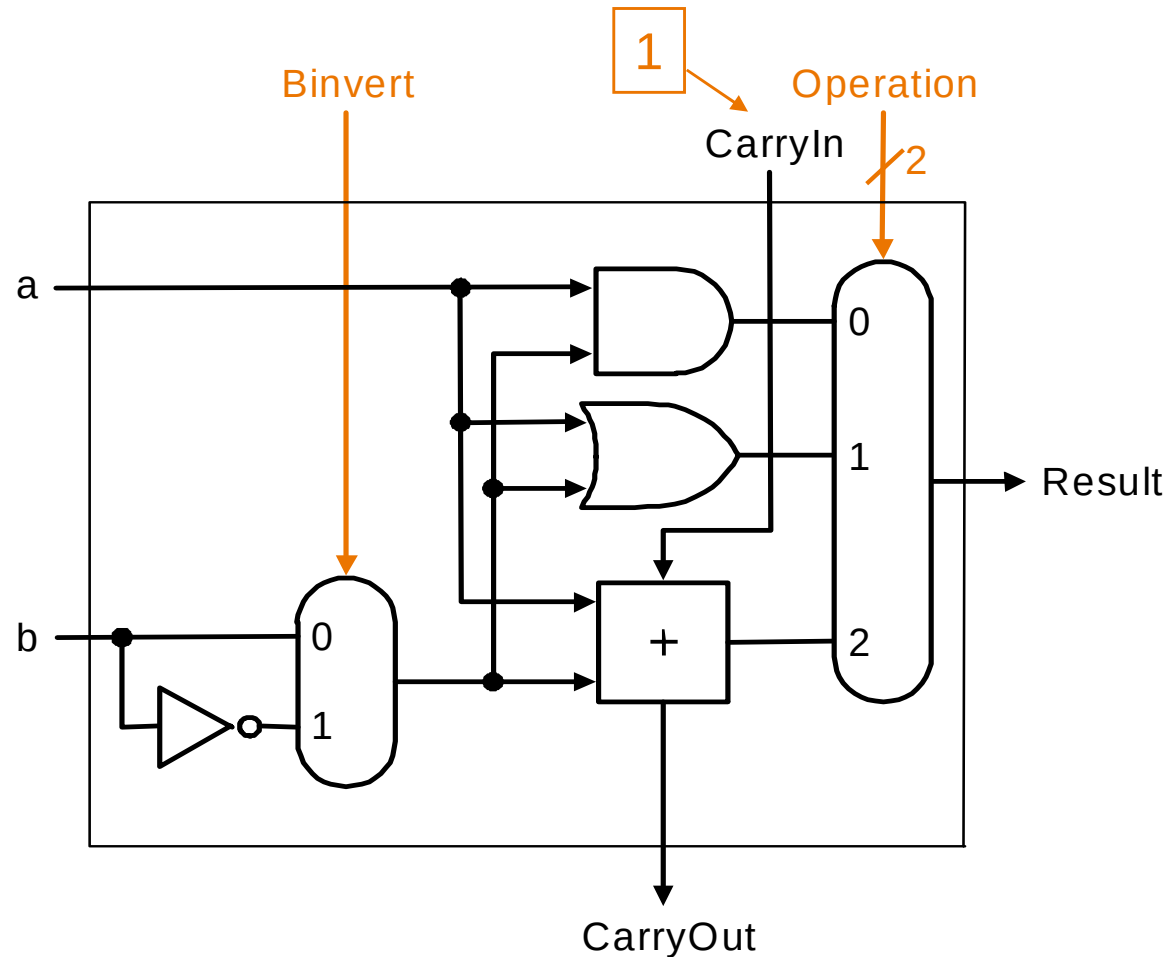


Differenti implementazioni del sommatore

- Vedere testo per diverse implementazioni di:
 - Sommatore
 - Sommatore con Look Ahead Carry
- Si puo' risolvere il problema della propagazione del carry utilizzando reti specializzate per la somma

Sottrazione $(a - b)$?

- **Approccio: fare $a + (-b)$**
- **Usare la rappresentazione in complemento a due
semplifica la logica necessaria per fare la negazione**



Supporto per la `slt` e per il test di uguaglianza

- Funzione set-on-less-than (`slt`)

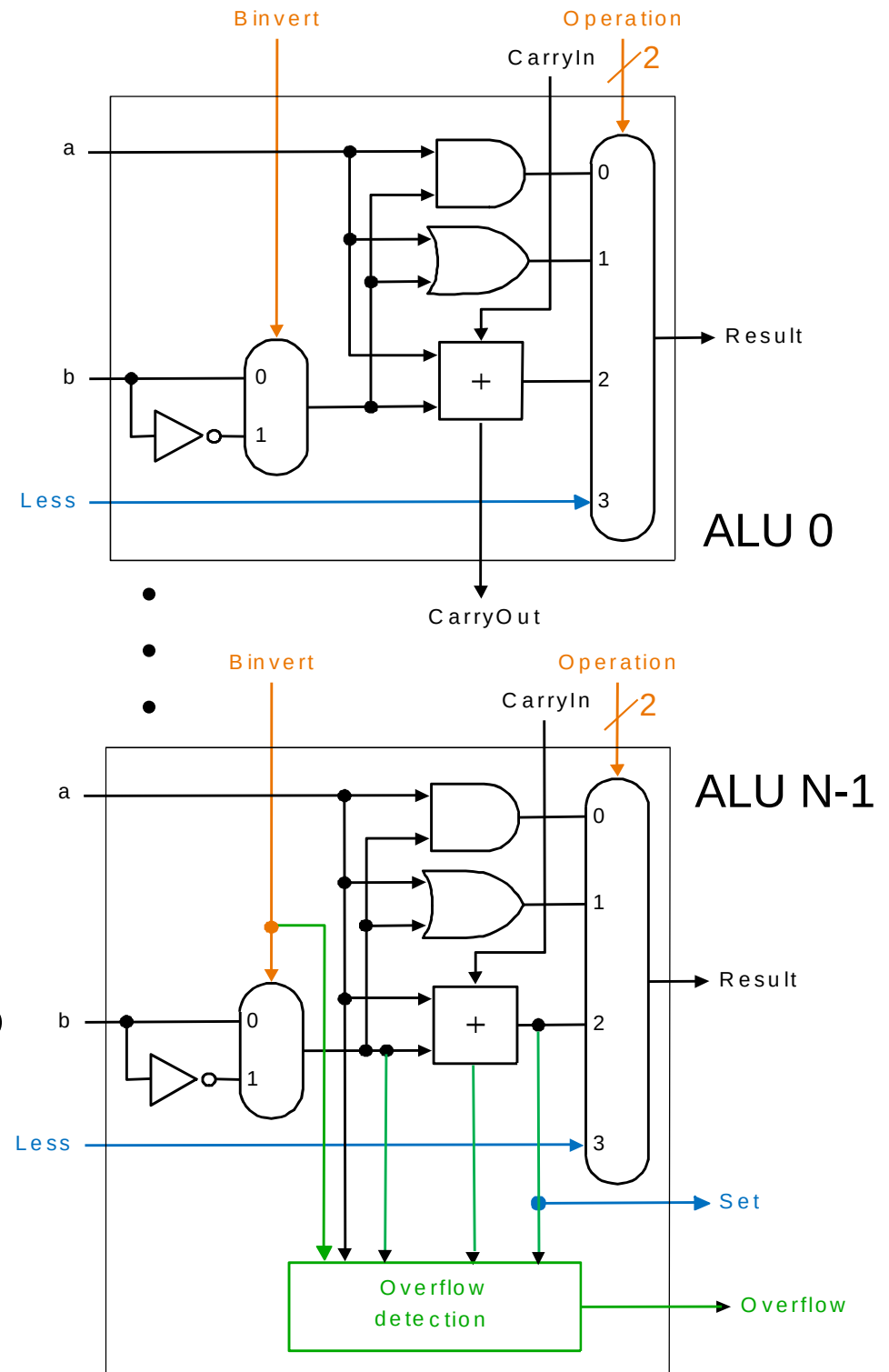
- Se $a < b$ produce 1, altrimenti 0
- Idea: usare la sottrazione
 $(a - b) < 0$ implica $a < b$

Supporto per il test di uguaglianza

- Se $a == b$ allora produco 1, altrimenti 0
- Idea: usare nuovamente la sottrazione
 $(a - b) = 0$ implica $a = b$

→ In entrambi i casi si può sfruttare la rete per fare la sottrazione

- Inserisco il caso "3" al multiplexer, corrispondente a $s \neq t$
- Nelle ALU 0, 1, ..., N-1 questo ingresso è collegato a un filo "Less" che costituisce un "bypass" ingresso-uscita
- Nella ALU N-1 inoltre (bit più significativo), controllo il valore di uscita 'set' che indica come è andato l'esito del confronto $a < b$
- Mentre, per gli interi in complemento a due è semplice sapere quando il risultato è negativo (segnale 'set'), in generale per rivelare l'overflow ho bisogno di una rete logica un poco più complessa

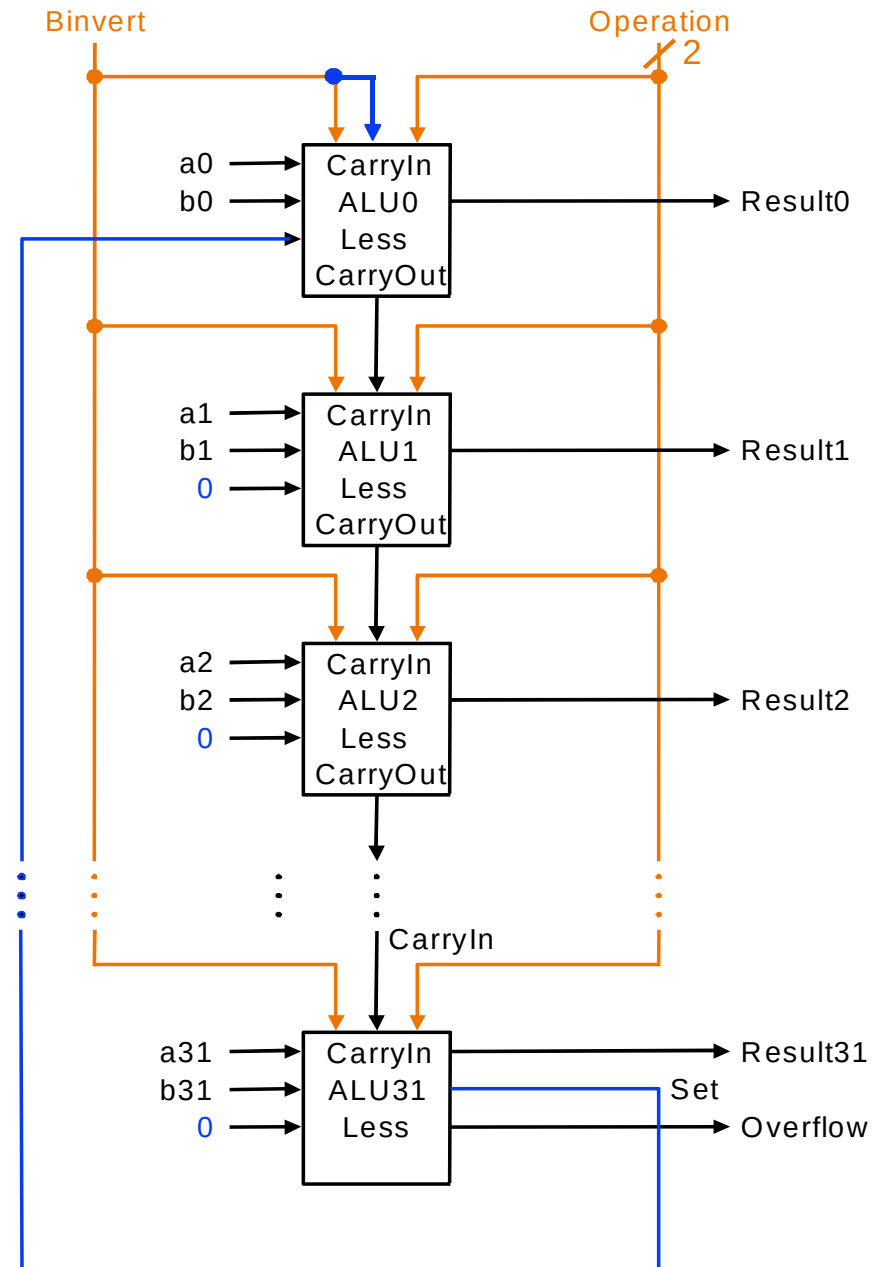


Supporto alla s1t *fuori* dalla ALU

- Se il confronto ha esito positivo il filo less, manda sul bit meno significativo il risultato (1)
- Aggiustiamo il CarryIn_0 :

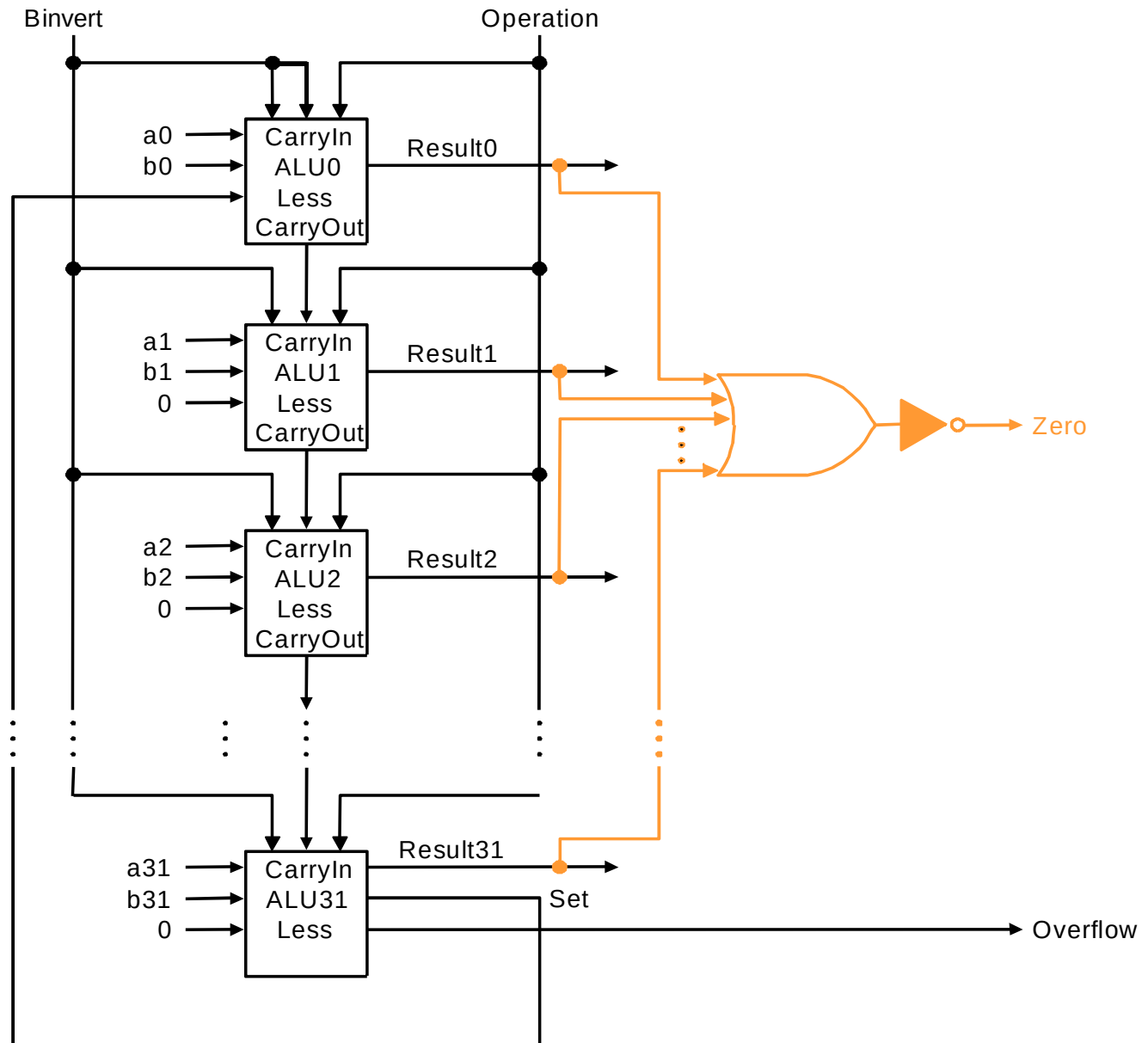
Operation	Function	CarryIn0	BInvert
0	AND	0	0
1	OR	0	0
2	ADD	0	0
2	SUB	1	1
3	SLT	1	1

→ $\text{CarryIn}_0 = \text{BInvert}$



Test per l'uguaglianza (beq)

- La rete fornisce il valore 1 quando tutti i bit del risultato sono a zero ($\rightarrow a=b$)
- Chiameremo tale segnale "Zero" e ci serve per decidere se l'esito del confronto e' positivo o meno

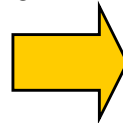


•Nota: "Zero" vale 1 quando il risultato e' zero!

ALUcontrol

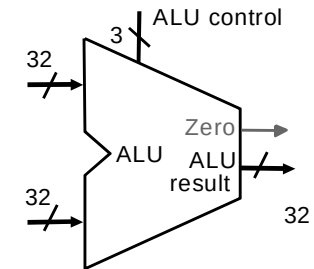
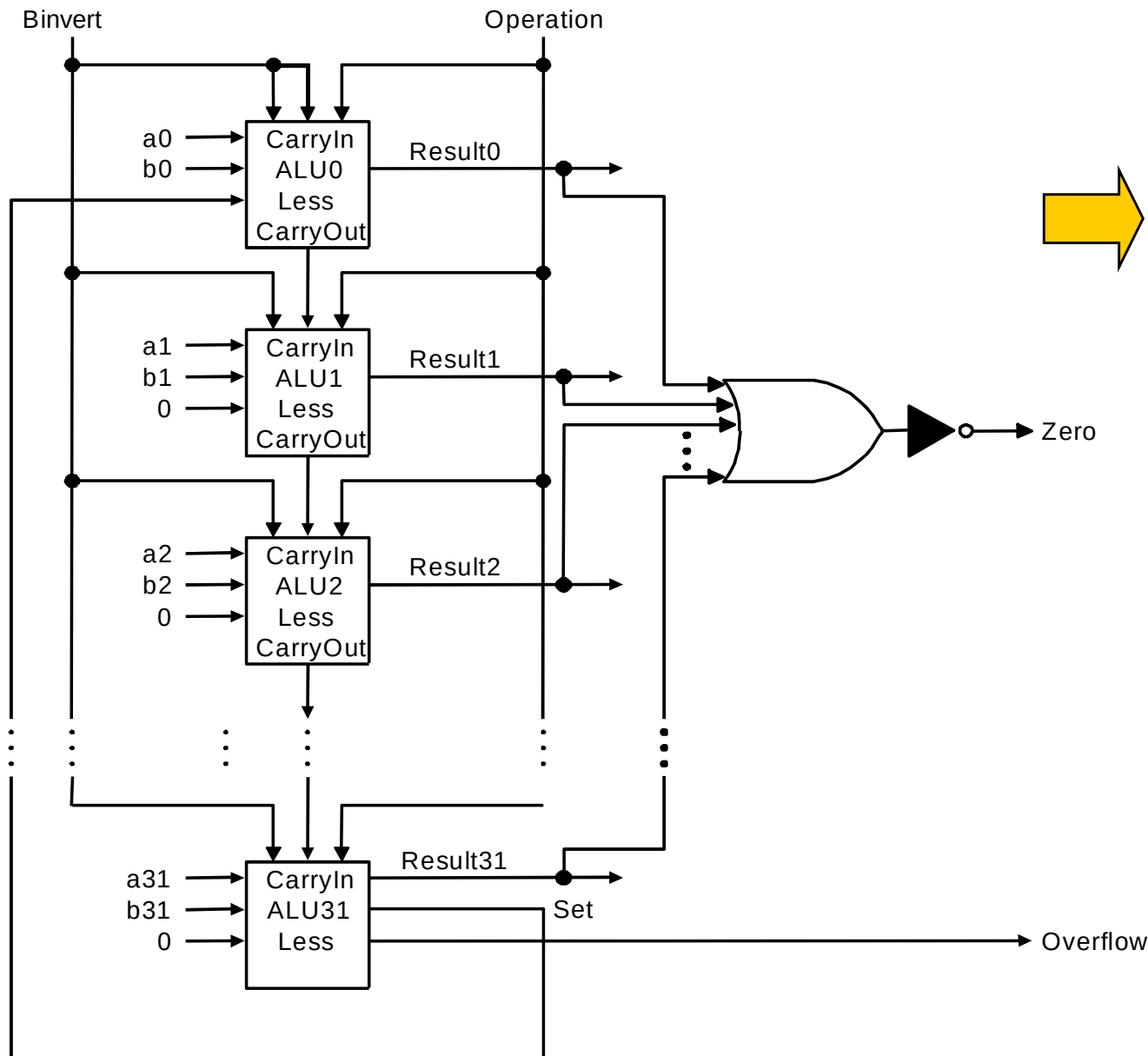
Operation	Function	Binvert
0	AND	0
1	OR	0
2	ADD	0
2	SUB	1
3	SLT	1
2	EQ	1

Binvert $\rightarrow$ ALUcontrol₂
Operation₁ $\rightarrow$ ALUcontrol₁
Operation₀ $\rightarrow$ ALUcontrol₀

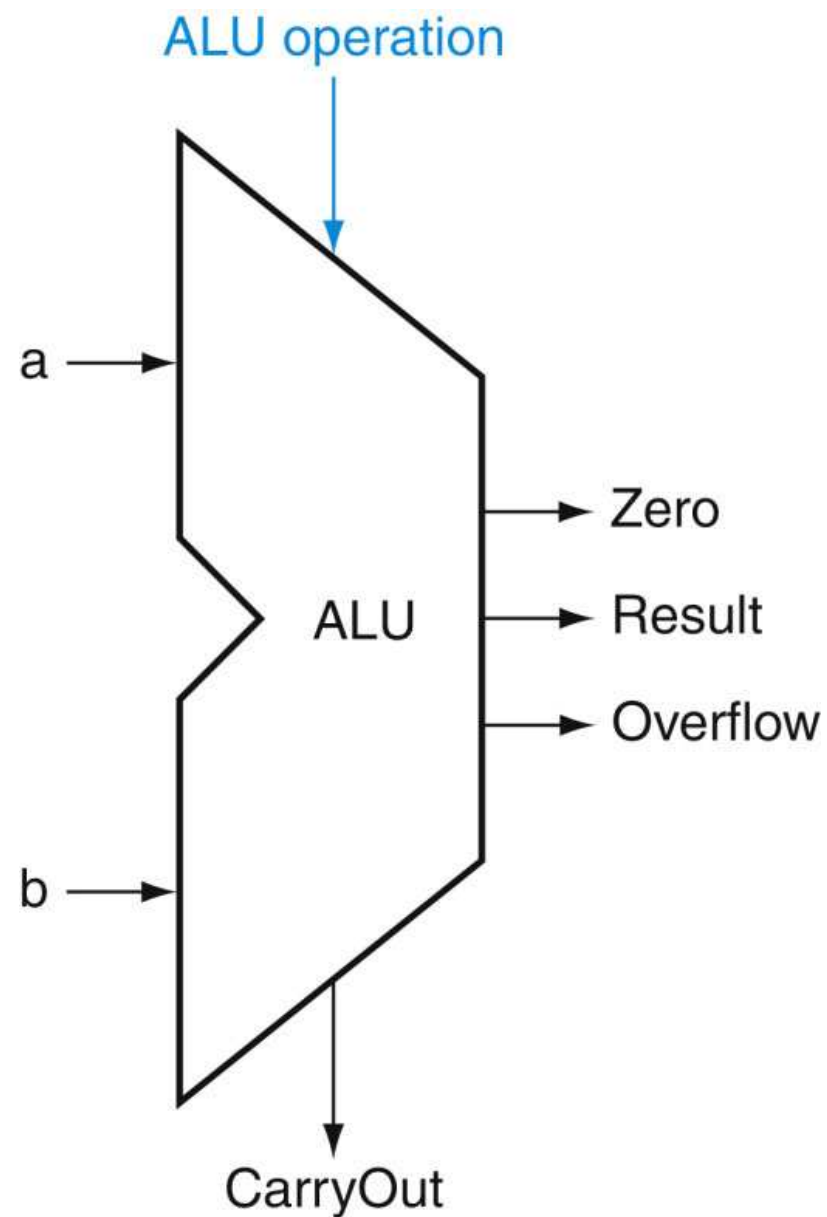


ALUcontrol	
0	00
0	01
0	10
1	10
1	11
1	10

Blocco "ALU a 32-bit"



Overflow lo possiamo dare
per implicito nel blocco
ALU



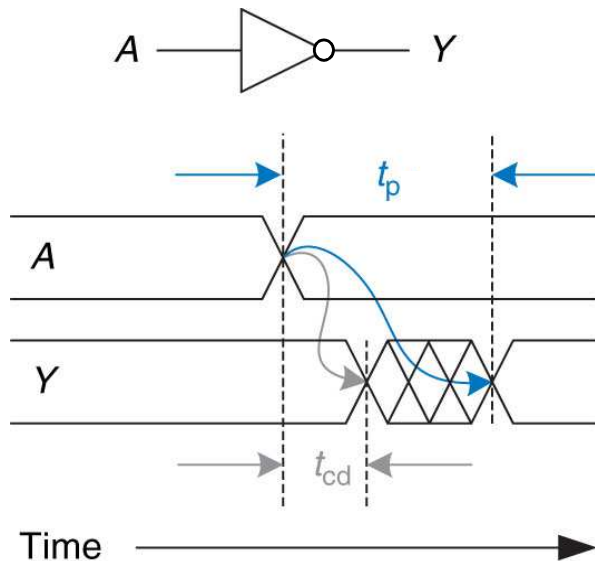
LEZIONE N° 92

Introduzione agli Hardware Description Languages (HDL)

TIMING

Timing

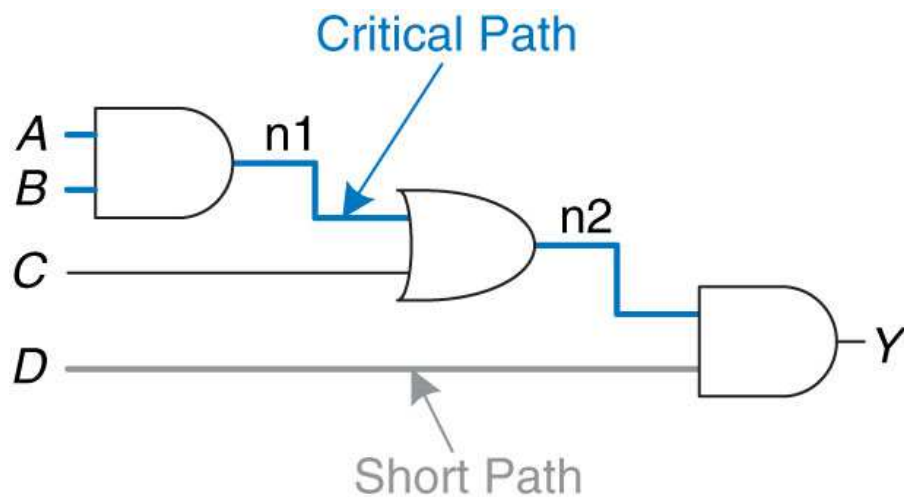
- Finora abbiamo analizzato solo l'aspetto funzionale della rete senza tenere conto dei tempi di propagazione del segnale fra ingresso e uscita (**behavior**)
- Abbiamo pero' visto come si puo' calcolare il tempo di propagazione per un semplice inverter CMOS



- Il **tempo di propagazione** t_p e' il tempo necessario affinché l'uscita sia stabile dal momento in cui ho variato l'ingresso
- Si definisce invece **tempo di contaminazione** t_{cd} il tempo che intercorre da quando ho variato l'ingresso a quando inizia a variare l'uscita

Critical path

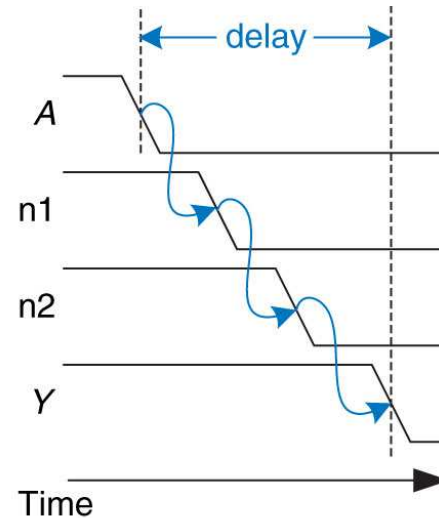
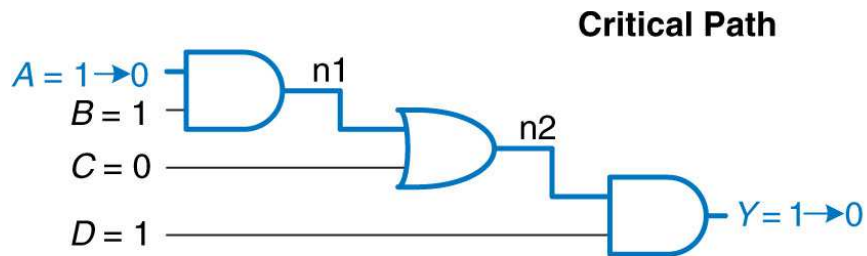
- Oltre ai ritardi dovuti alla singola porta logica, quando si interconnettono piu' porte in cascata puo' accadere che determinati segnali giungano alle varie porte intermedie con ritardi diversi. Esempio:



- In particolare il cammino piu' lungo, fra un ingresso e l'uscita e' chiamato **cammino critico (critical path)**

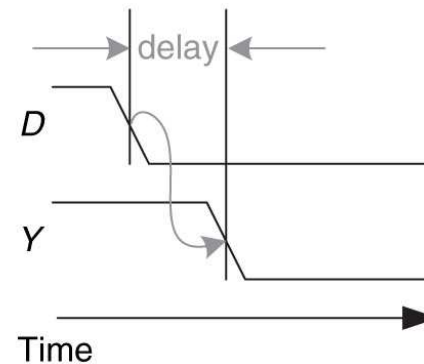
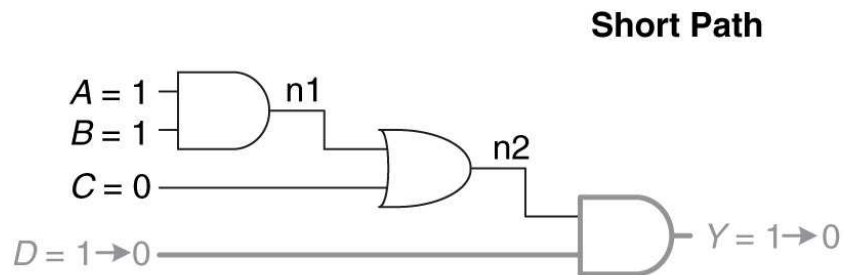
Calcolo dei tempi

- Cambiando l'ingresso A da 1 a 0



$t_p = 2t_{p\_AND} + t_{p\_OR}$

- Cambiando l'ingresso D da 1 a 0



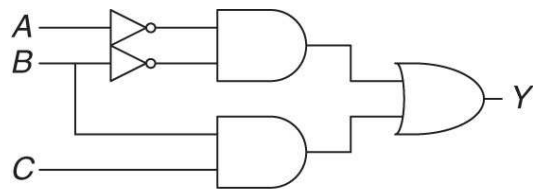
$t_{cd} = t_{cd\_AND}$

Tempi di propagazione tipici

Gate	t_p (ps)
NOT	30
2-input AND	60
3-input AND	80
4-input OR	90
tristate (A to Y)	50
tristate (enable to Y)	35

Alee (Glitches/Hazards)

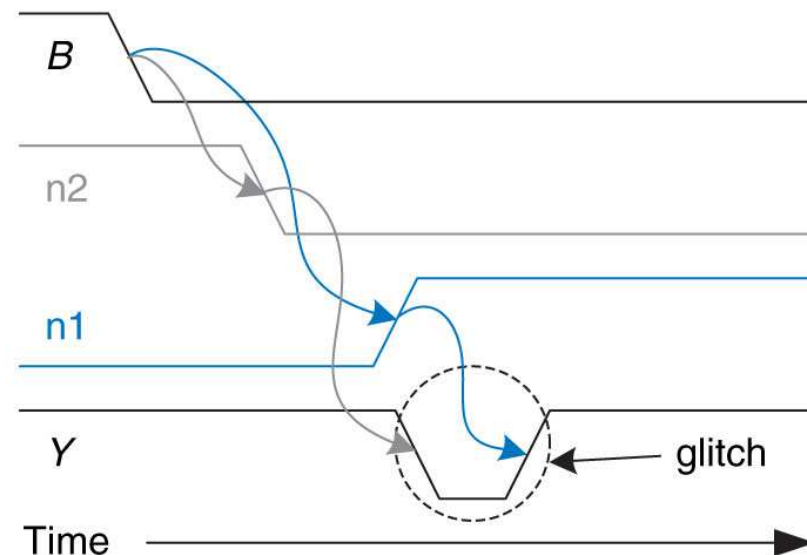
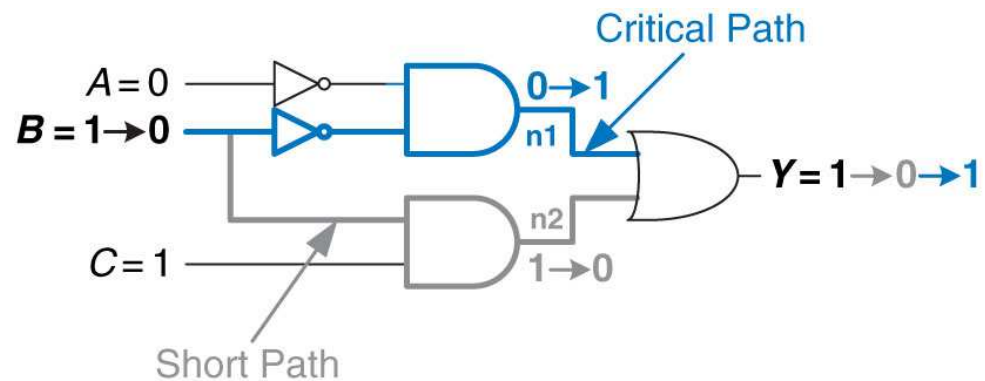
- Puo' accadere che variando un ingresso si creino variazioni multiple dell'uscita
- Esempio: rete con alee



Y	AB			
	00	01	11	10
0	1	0	0	0
1	1	1	1	0

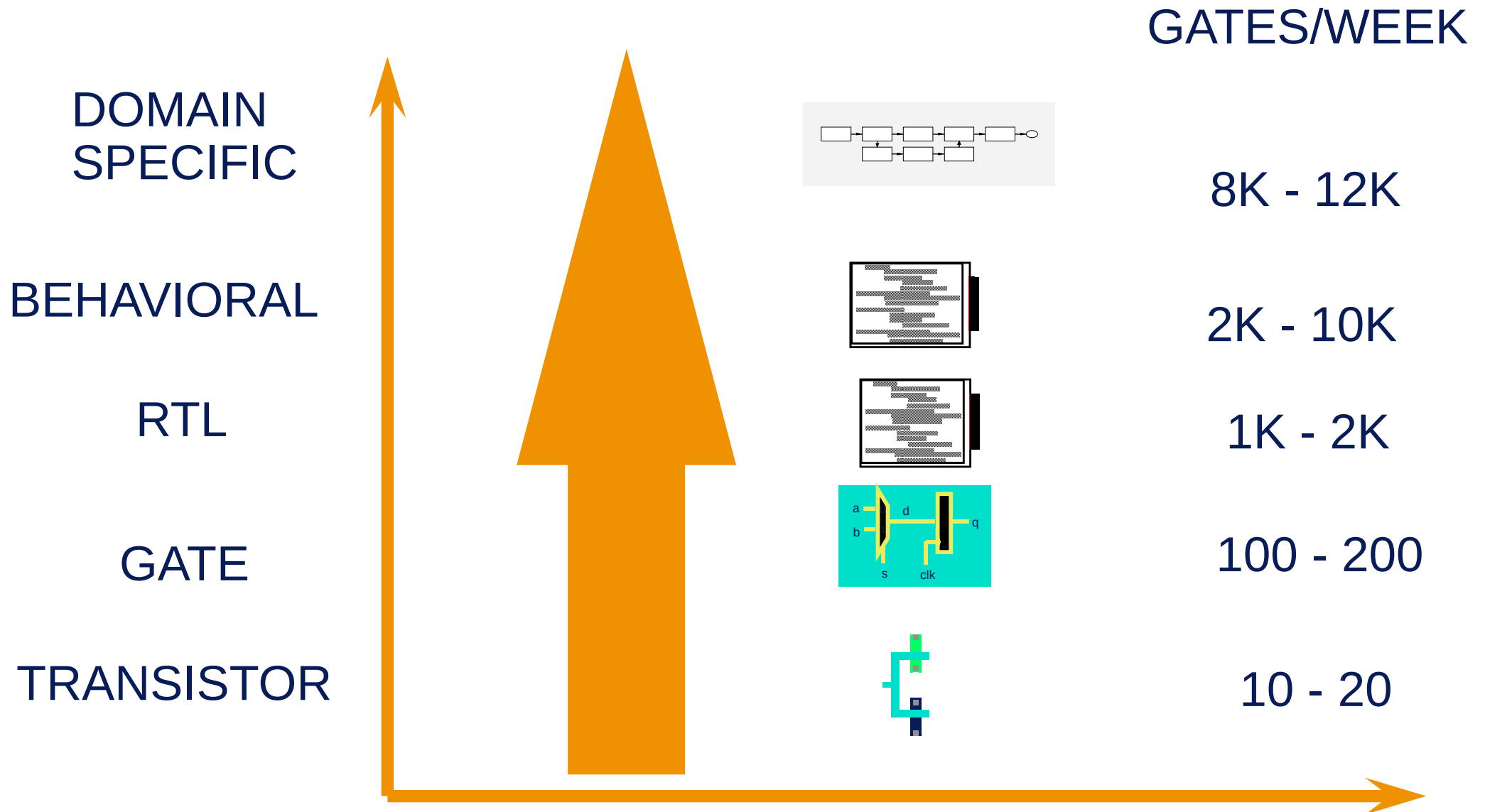
$$Y = \bar{A}\bar{B} + BC$$

- E' importante essere in grado di analizzare il comportamento temporale di una rete logica



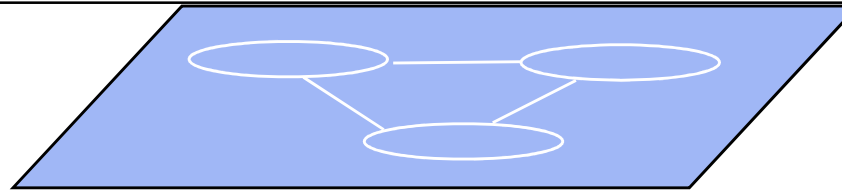
IMPORTANZA DEL CAD

Produttività vs. Approccio



Livello di astrazione

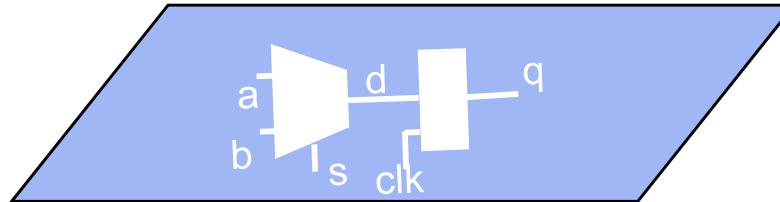
System



Behavioral

```
while (A<B)
loop
A:= A+K;
.....
end loop;
```

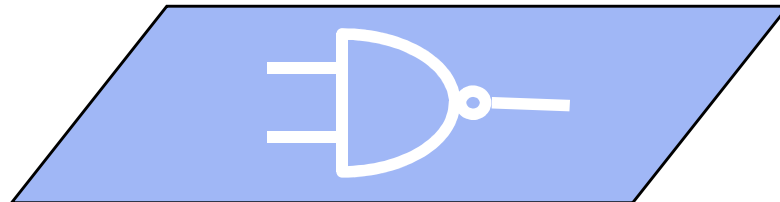
Register Transfer



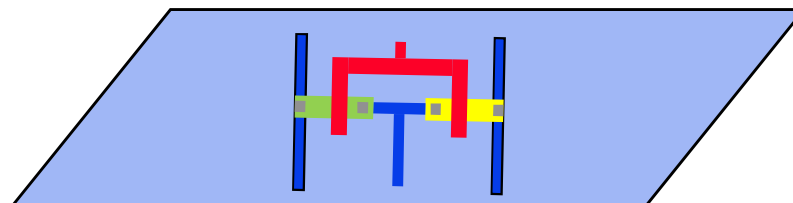
Logic

```
A:=not (B and C xor D)
F:= (A xor B)and (C or K);
```

Gate



Layout



sintesi

Verifica

Specifiche del linguaggio

- **Caratteristiche principali**

- Generare eseguibili (Simulazione)
- Permettere la stessa semantica per simulazione, sintesi e verifica
- Consentire diversi livelli di astrazione
- Descrizione dell'hardware attraverso software
- Indipendente da :
 - Tecnologia, stile metodologia di progetto, ...
- User friendly

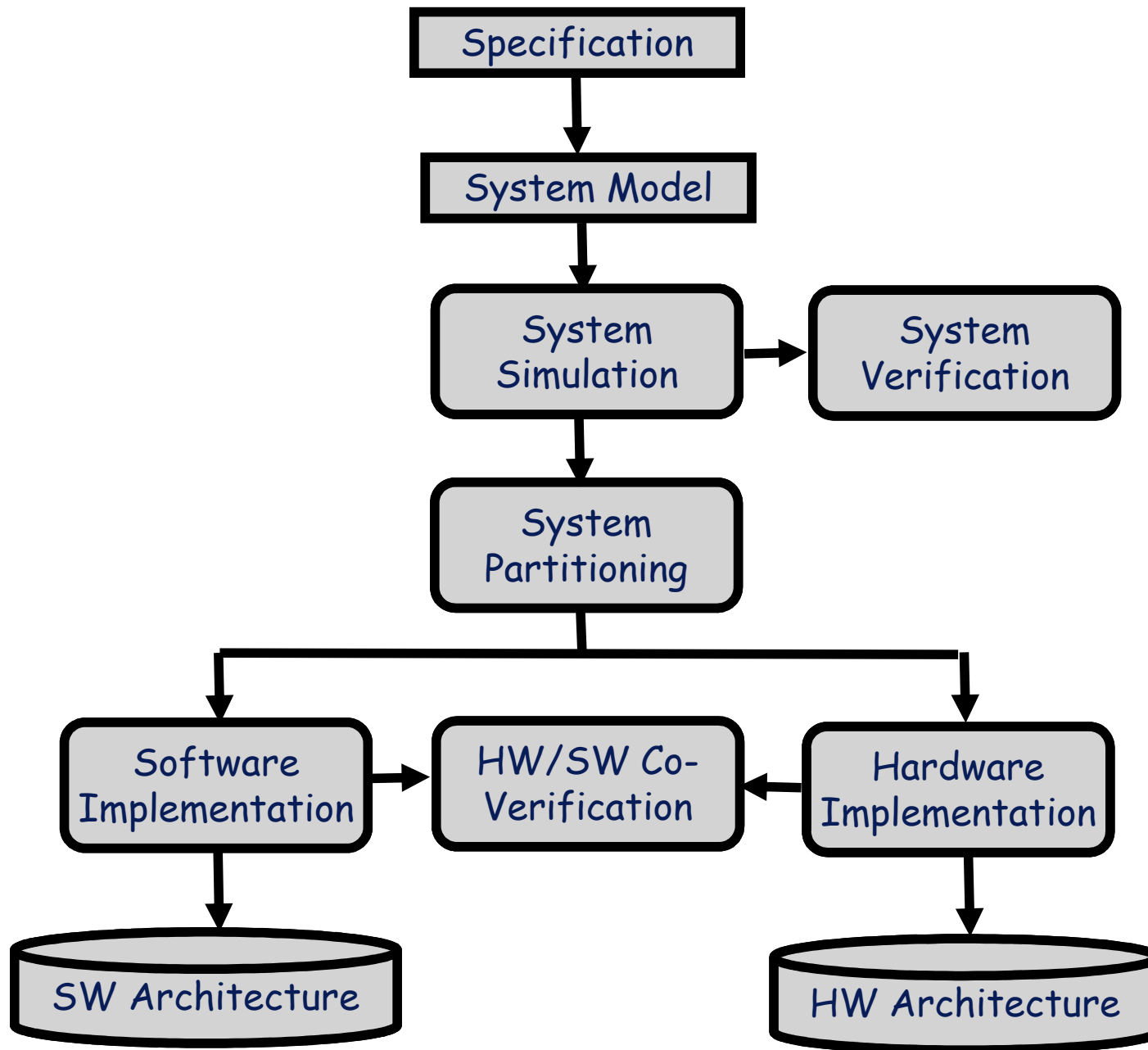
- **Linguaggi diversi per applicazioni diverse:**

- VHDL, Verilog, Esterel, Silage, etc.



STANDARD IEEE

HW/SW Co-Design



HW/SW Co-design tools

Commercial

- ✧ Cadence VCC
- ✧ Coware N2C
- ✧ Synopsys El-Greco
- ✧ Synopsys Co-Centric
- ✧ etc.

Public Domain/Research

- ✧ Polis - Berkeley Univ.
- ✧ Felix - Berkeley Univ.
- ✧ Cosmos - TIMA Grenoble
- ✧ Olympus - Stanford Univ.
- ✧ SystemC Initiative
- ✧ etc.

VERILOG

Verilog

- **Vantaggi di un linguaggio di descrizione dell'hardware (HDL)**
 - Permette di descrivere e simulare il comportamento di una rete logica e eliminarne i bug
 - La rete progettata puo' essere compilata generando automaticamente hardware (molto piu' veloce e con consumi energetici estremamente piu' bassi rispetto al solo software)
 - Nota: l'HDL e' ... software !
- **Il Verilog e' basato sul C (assomiglia al C)**
 - Permette di costruire dalle reti usando la programmazione strutturata
 - Il VHDL assomiglia al linguaggio Ada

Verilog: Tipi di dato e Operatori

- Tipi di dato
 - **wire** indica un segnale
 - **reg** (register) indica una variabile che memorizza un valore
 - Un reg non necessariamente corrisponde ad un registro fisico, anche se spesso e' cosi'
- Un registro o wire a 32 bit si dichiara con
 - **wire[31:0]** Y
 - **reg[31:0]** X
 - Si puo' fare accesso ad un sottoinsieme di tali bit indicandolo con la notazione [starting_bit : ending_bit]
- Si possono combinare piu' registri per formare una struttura "register file" ovvero una "memoria" *
 - **reg[31:0] registerfile[0:31]**
 - Si fa accesso al singolo registro con la classica notazione registerfile[num_registro]

Verilog: operatori

Relational Operators

Operator	Application
<	a < b // is a less than b? // return 1-bit true/false
>	a > b // is a greater than b?
>=	a >= b // is a greater than or // equal to b
<=	a <= b // is a less than or // equal to b

Arithmetic Operators

Operator	Application
*	c = a * b ; // multiply a with b
/	c = a / b ; // int divide a by b
+	sum = a + b ; // add a and b
-	diff = a - b ; // subtract b // from a
%	amodb = a % b ; // a mod(b)

Logical Operators

Operator	Application
&&	a && b ; // is a and b true? // returns 1-bit true/false
	a b ; // is a or b true? // returns 1-bit true/false
!	if (!a) ; // if a is not true c = b ; // assign b to c

Equality and Identity Operators

Operator	Application
=	c = a ; // assign a to c
==	c == a ; /* is c equal to a returns 1-bit true/false applies for 1 or 0, logic equality, using X or Z oper- ands returns always false 'hx == 'h5 returns 0 */
!=	c != a ; // is c not equal to // a, returns 1-bit true/ // false logic equality
===	a === b ; // is a identical to // b (includes 0, 1, x, z) / // 'hx === 'h5 returns 0
!==	a !== b ; // is a not // identical to b returns 1- // bit true/false

Unary, Bitwise and Reduction Operators

Operator	Application
+	Unary plus & arithmetic(binary) addition
-	Unary negation & arithmetic (binary) subtraction
&	b = &a ; // AND all bits of a
	b = a ; // OR all bits
^	b = ^a ; // Exclusive or all bits of a
-&, - , -^	NAND, NOR, EX-NOR all bits to-gether c = -& b ; d = - a ; e = -^ c ;
~, &, , ^	bit-wise NOT, AND, OR, EX-OR b = ~a ; // invert a c = b & a ; // bitwise AND a,b e = b a ; // bitwise OR f = b ^ a ; // bitwise EX-OR
-&, - , -^	bit-wise NAND, NOR, EX-NOR c = a -& b ; d = a - b ; e = a -^ b ;

Shift Operators and other Operators

Operator	Application
<<	a << 1 ; // shift left a by // 1-bit
>>	a >> 1 ; // shift right a by 1
?:	c = sel ? a : b ; /* if sel is true c = a, else c = b , ?: ternary operator */
{}	{co, sum } = a + b + ci ; /* add a, b, ci assign the overflow to co and the re- sult to sum: operator is called concatenation */
{ {} }	b = {3{a}} /* replicate a 3 times, equivalent to {a, a, a} */

Verilog: valori e costanti

- I possibili valori di un registro o un filo sono:
 - 0 o 1 per rappresentare i valori logici falso o vero rispettivamente
 - X per rappresentare un "don't care" (non-specificato) o un valore non noto
 - Z per rappresentare lo stato di alta impedenza (filo "scollegato" o uscita "libera")
- I nomi simbolici delle costanti si dichiarano con parameter
 - parameter A=1
- I valori delle costanti possono essere specificati in varie basi (2,8,10,16 tipicamente)
 - 4'b0101 indica una costante binaria a 4 bit che vale 5
 - 8'h4 indica una costante binaria a 8 bit che vale 4
 - {x{ bitfield }} replica i bit indicati x volte, es. {3{2'b01}} crea 010101
 - {bitfield1, bitfield2} concatena i due bitfield, es. {A[7:4],B[5:2] }

Verilog: Indicazione del timing

- Per indicare il tempo di propagazione di un'operazione:

`assign #2 n=~b`

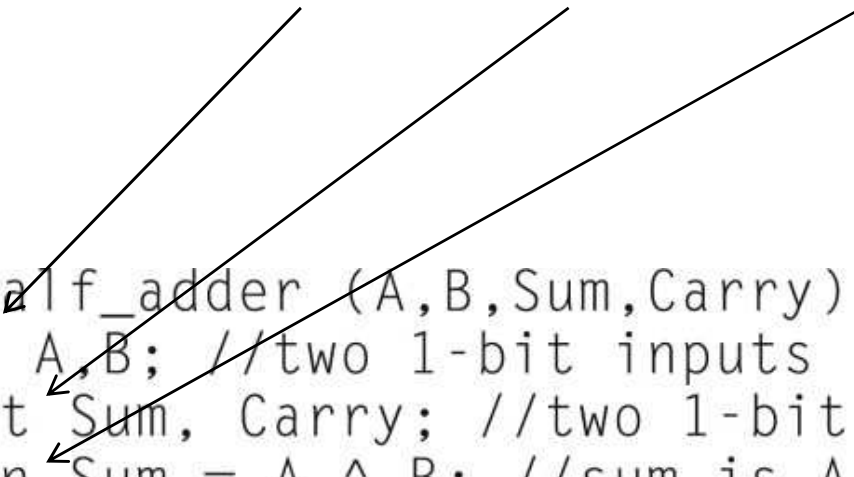
- Indica un ritardo di 2 ns (default - ma si puo' cambiare la scala)

Esempio: half adder in Verilog

- Dichiaro innanzitutto un modulo con determinati nomi per gli ingressi e le uscite:

ingressi uscite comportamento

```
module half_adder (A,B,Sum,Carry);  
  input A,B; //two 1-bit inputs  
  output Sum, Carry; //two 1-bit outputs  
  assign Sum = A ^ B; //sum is A xor B  
  assign Carry = A & B; //Carry is A and B  
endmodule
```



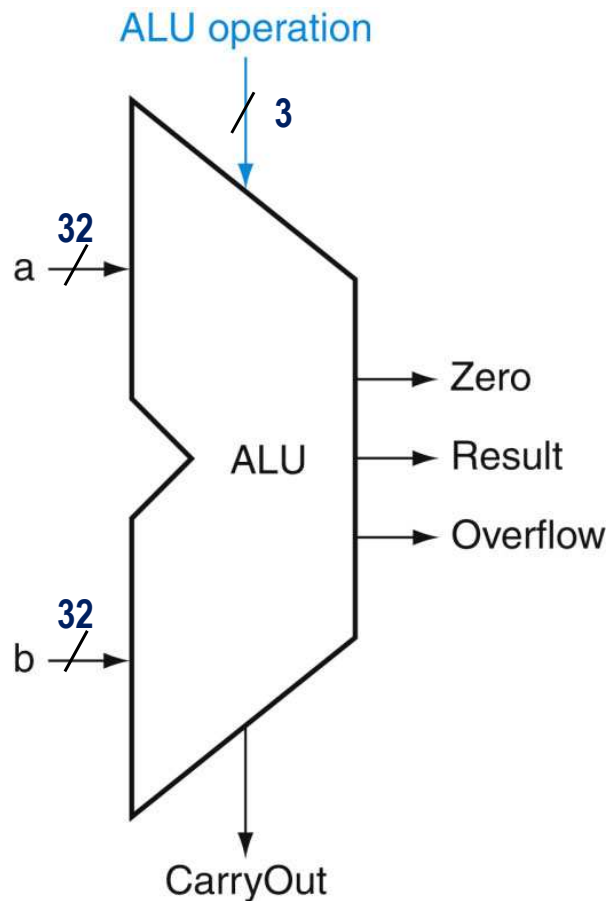
Esempio: multiplexer a 4 ingressi di 32 bit

Memoria interna (registro)

Qualcosa accade quando vengono variati i seguenti segnali

```
module Mult4to1 (In1,In2,In3,In4,Sel,Out);  
  input [31:0] In1, In2, In3, In4; //four 32-bit inputs  
  input [1:0] Sel; //selector signal  
  output reg [31:0] Out; // 32-bit output  
  always @(In1, In2, In3, In4, Sel)  
  case (Sel) //a 4->1 multiplexor  
    0: Out <= In1;  
    1: Out <= In2;  
    2: Out <= In3;  
    default: Out <= In4;  
  endcase  
endmodule
```

Esempio: ALU



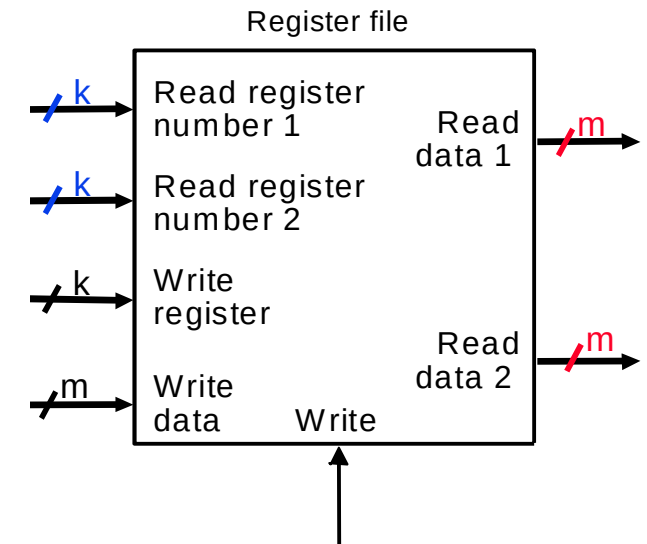
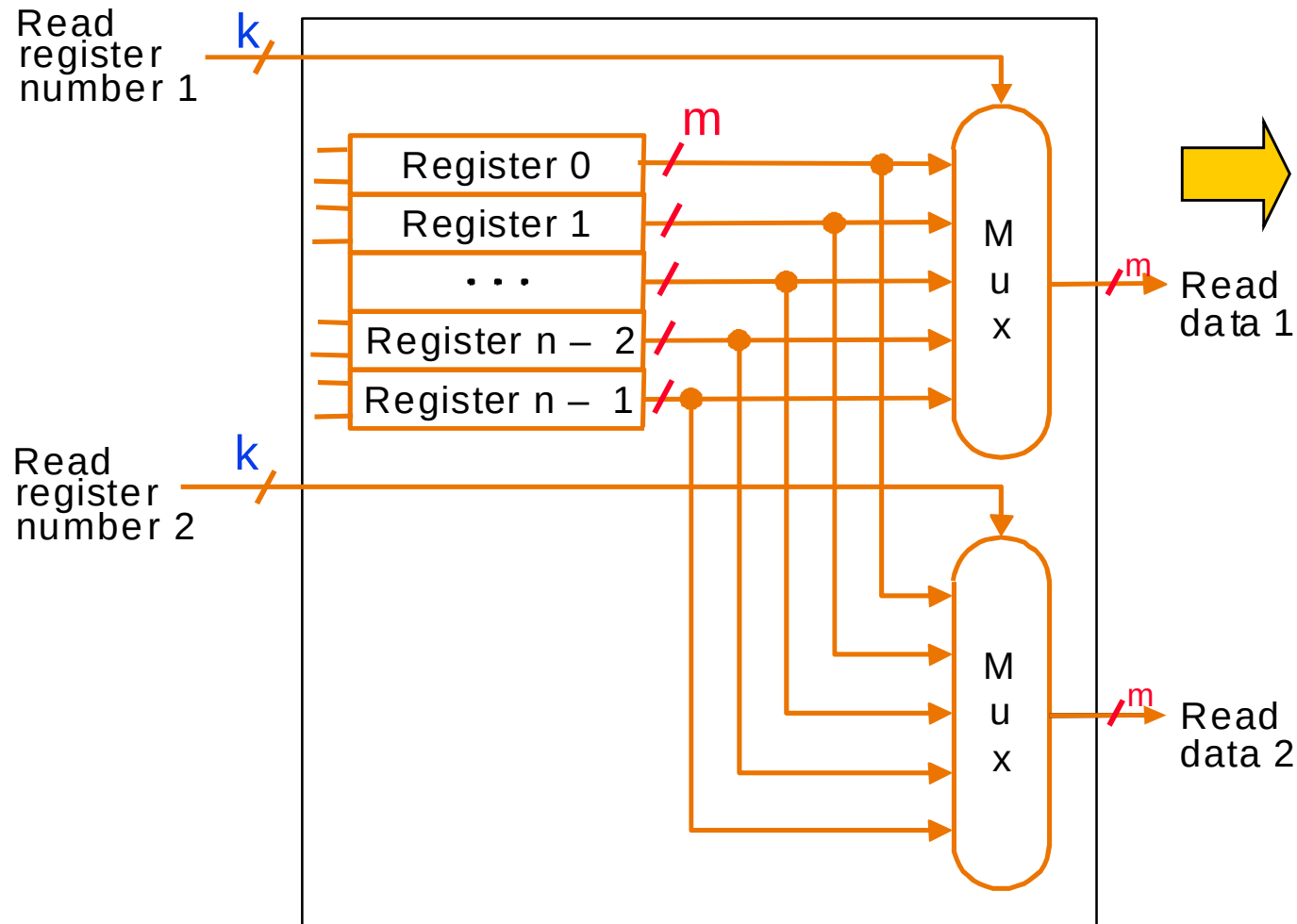
```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
    input [3:0] ALUctl;
    input [31:0] A,B;
    output reg [31:0] ALUOut;
    output Zero;
    assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0; goes anywhere
    always @(ALUctl, A, B) //reevaluate if these change
        case (ALUctl)
            0: ALUOut <= A & B;
            1: ALUOut <= A | B;
            2: ALUOut <= A + B;
            6: ALUOut <= A - B;
            7: ALUOut <= A < B ? 1:0;
            12: ALUOut <= ~(A | B); // result is nor
            default: ALUOut <= 0; //default to 0, should not happen;
        endcase
    endmodule
```

Aggiungo anche
la funzione NOR
→ 4 bit su ALUctl

REALIZZAZIONE DEL PROCESSORE MIPS

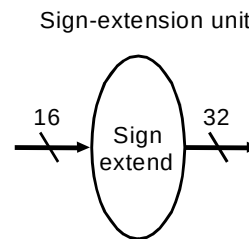
Register File: logica di lettura

- Si usano i flip-flop di tipo D

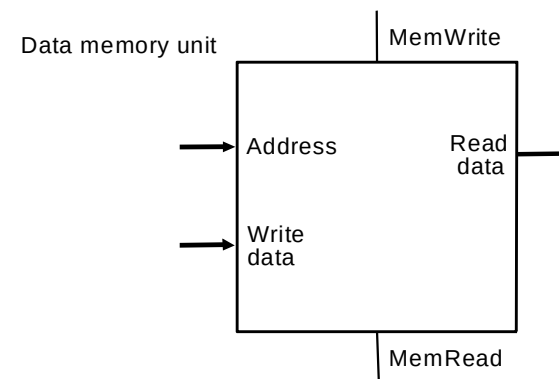
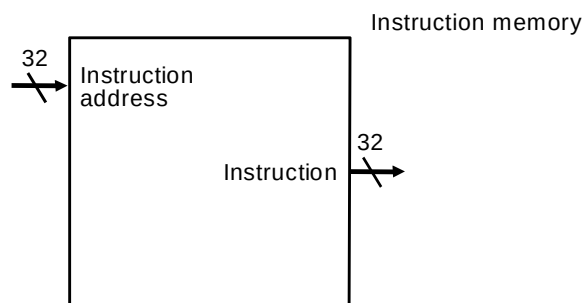


Altra logica sparsa

- Nel caso della beq, l'offset di salto puo' essere sia positivo che negativo e specificato su 16 bit: occorre una rete di estensione dell'operando con segno da 16 a 32 bit
- Questo e' anche il caso della lw, sw



- La memoria istruzioni e la memoria dati possono essere viste così':



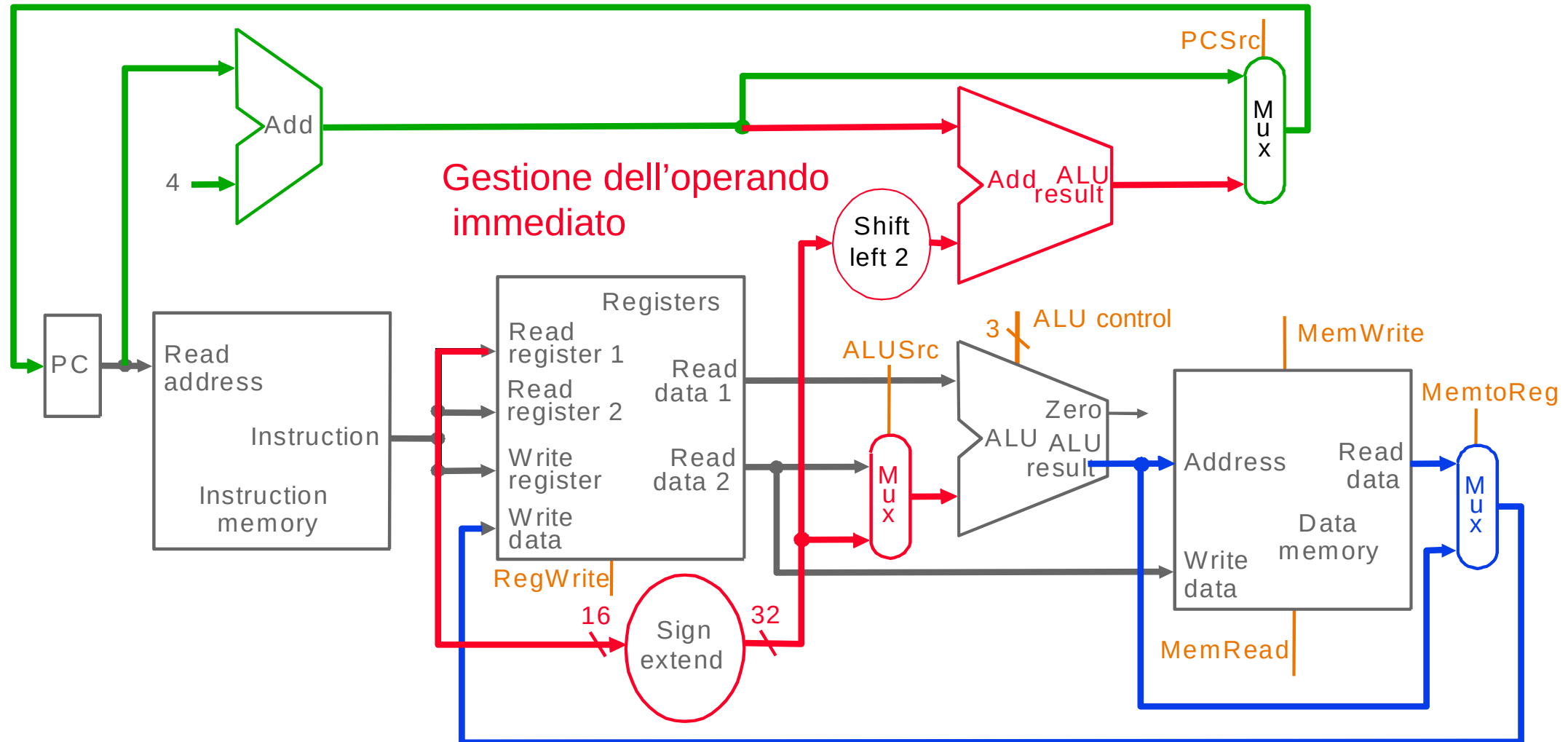
Riassumendo

- **ALU per supportare il set di istruzioni MIPS**
 - idea chiave: usare un multiplexer per selezione l'output desiderato
 - possiamo effettuare efficientemente la sottrazione usando la rappresentazione in complemento a 2
 - il confronto "minore di" si fa guardando il bit più significativo
 - il confronto "uguale a" si fa con una differenza e rilevando "0"
 - possiamo replicare una ALU a 1-bit per produrre una ALU a 32-bit
- **Hardware:**
 - tutte le porte logiche dovrebbero sempre lavorare
 - la velocità di una porta è influenzata dal numero di ingressi a tale porta
 - la velocità di un circuito è influenzata dal numero di porte in cascata (e in particolare sul "cammino critico" o sul più profondo livello di logica)

Costruzione del Datapath

Gestione indirizzo istruzione successiva

I segnali di controllo vengono pilotati in base a "op-code" e "funct" dell'istruzione:



Gestione del risultato della ALU o dato dalla memoria

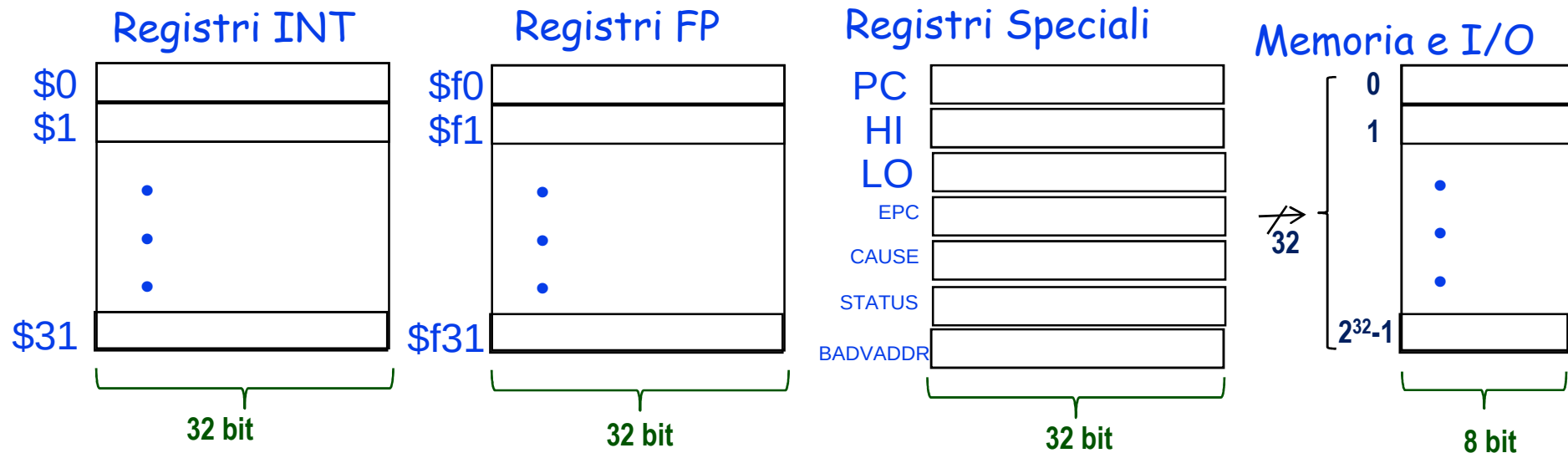
Lezione 2

Assembly MIPS (2)

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Instruction Set Architecture (es. MIPS 3000)



- **Categorie di Istruzioni**

- Load/Store
- Calcolo (per Interi e per Frazionari)
- Jump/Branch
- Istruzioni speciali

- **Formato delle Istruzioni:**

- lunghezza FISSA a 32 bit → CPU a 32 bit

Ricapitolazione:

- Istruzione

- Significato

add \$1,\$2,\$3

\$1 = \$2 + \$3

sub \$1,\$2,\$3

\$1 = \$2 - \$3

slt \$1,\$2,\$3

\$1 = (\$2 < \$3) ? 1 : 0

lw \$1,100(\$2)

\$1 = Memory[\$2+100]

sw \$1,100(\$2)

Memory[\$2+100] = \$1

bne \$4,\$5,L

Next inst. is at Label if \$4!=\$5

beq \$4,\$5,L

Next inst. is at Label if \$4==\$5

j Label

Next inst. is at Label

- Formati:

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

Aritmetica con segno e senza segno

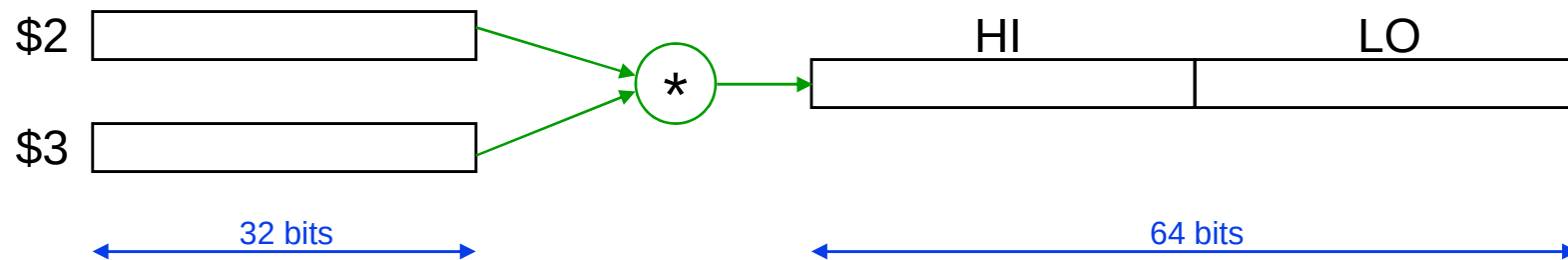
- Le istruzioni aritmetiche fin qui esaminate operano su **interi con segno**
 - Gli operandi (a 32 bit, nei registri) sono rappresentati in **complemento a due: i valori vanno da -2^{31} a $2^{31}-1$**
 - Anche i byte-offset di lw e sw e il numero di istruzioni di cui saltare nella bne/beq sono interi con segno da -2^{15} a $2^{15}-1$
- E' possibile utilizzare anche operandi **senza segno**
 - In tal caso i valori vanno da 0 a $2^{32}-1$
 - Invece di add, sub, slt si useranno addu, subu, sltu
 - slt e sltu forniranno un risultato diverso se un operando e' negativo (es. $-4 < 3 \rightarrow 1$ con slt, ma 0 con sltu)
 - addu e subu manipolano gli operandi esattamente nello stesso modo di add e sub, ma NON segnalano alcuna situazione di overflow al processore**

** il processore gestisce tali situazioni con il meccanismo della "eccezione", che sara' esaminato piu' avanti nel corso

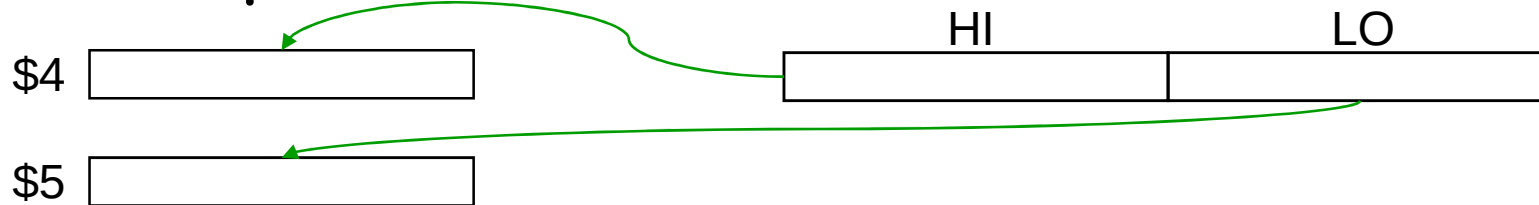
Moltiplicazione

`mult $2, $3`

$(HI, LO) = \$2 * \$3$



- Per recuperare il risultato



`mfhi $4`

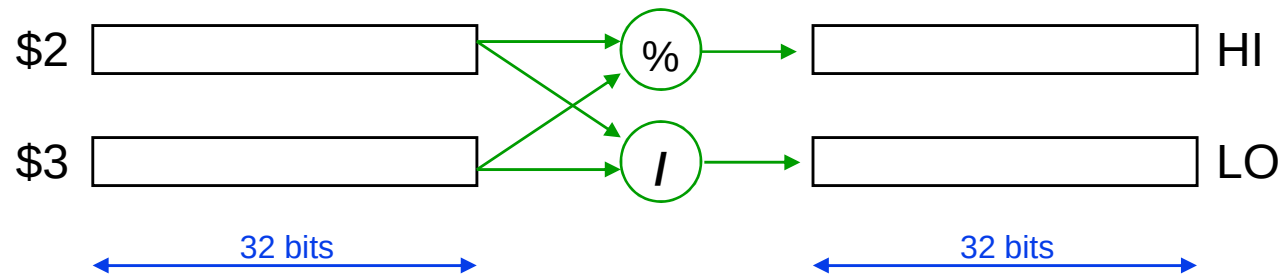
`mflo $5`

Nota1: `mfhi`="move from HI", `mflo`="move from LO"
Nota2: si possono usare HI e LO anche come registri di appoggio: `mt hi`="move to HI", `mt lo`="move to LO"

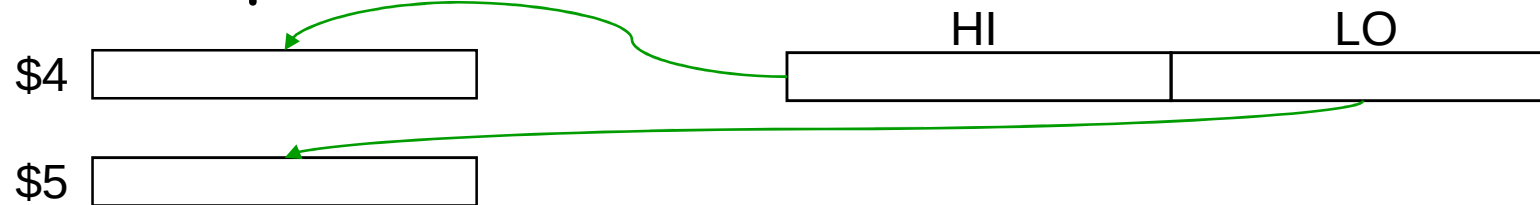
Divisione

div \$2, \$3

LO = \$2 / \$3
HI = \$2 % \$3



- **Per recuperare il risultato**



mfhi \$4

mflo \$5

Costanti

- Le costanti “piccole” sono le piu' usate (50% dei casi)

e.g., $A = A + 5;$
 $B = B + 1;$
 $C = C - 18;$

- Approccio RISC:
 - Mettere le costanti “piccole” nell'istruzione
 - Mettere le costanti meno frequenti in memoria
 - Creare un registro hard-wired (come \$zero) per la costante 0
- Istruzioni MIPS :

```
addi $29, $29, 4
slti $8, $18, 10
```

- per codificare queste istruzioni si usa ancora il **formato I**
(I sta per 'immediato': uno degli operandi e' dentro l'istruzione stessa)
QUINDI la costante piccola e' un numero da -32768 a 32767

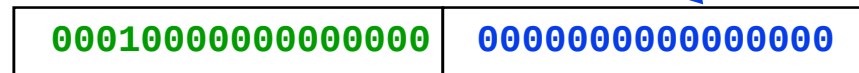
Che si fa con le costanti "grandi"?

- Vorremo caricare costanti a 32 bit nei registri
 - Es. 0x1000'2222
 - La parte bassa (0x2222) la potrei caricare con `addiu $t0, $0, 0x2222`
 - Come carico pero' la parte alta ?

1) Si introduce una nuova istruzione ("load upper immediate" = lui)

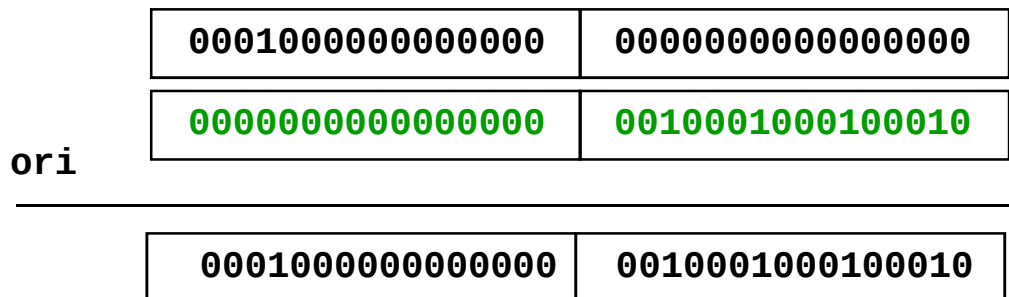
`lui $t0, 0x1000`

Riempita con zeri



2) Si usa successivamente una `addiu` (o una `ori`) per caricare la parte bassa

`ori $t0, $t0, 0x2222`



Pseudoistruzioni

- La coppia di istruzioni

- lui \$1, 16bit_alti
- ori \$1, \$1, 16bit_bassi

E' talmente frequente che a livello di assembler si puo' creare una **pseudoistruzione** la = "load address"

- la \$1, costante_a_32_bit

- dove $\text{costante\_a\_32\_bit} := (\text{16bit\_alti} | \text{16bit\_bassi})$
- viene automaticamente tradotta nella coppia di istruzioni native lui+ori

- Esistono altre pseudoistruzioni, ad es.

- muli \$1, \$2, costante_a_16_bit

- L'istruzione nativa e' pero' la mult

- Noi cercheremo di usare sempre istruzioni native

- **Unica pseudoistruzione ammessa: la**

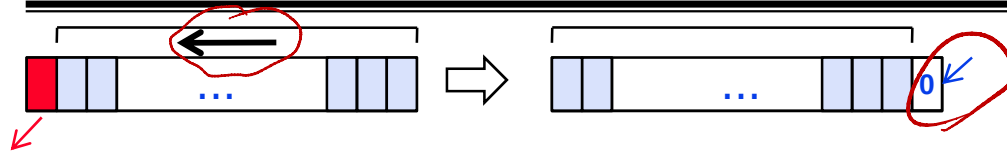
MIPS: istruzioni logiche

Instruzione	Esempio	Significato	Commento
<u>and</u>	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	3 reg. operands; Logical AND
<u>or</u>	or \$1,\$2,\$3	$\$1 = \$2 \$3$	3 reg. operands; Logical OR
<u>xor</u>	xor \$1,\$2,\$3	$\$1 = \$2 \oplus \$3$	3 reg. operands; Logical XOR
<u>nor</u>	nor \$1,\$2,\$3	$\$1 = \sim(\$2 \$3)$	3 reg. operands; Logical NOR
<u>and immediate</u>	andi \$1,\$2,10	$\$1 = \$2 \& 10$	Logical AND reg, constant
<u>or immediate</u>	ori \$1,\$2,10	$\$1 = \$2 10$	Logical OR reg, constant
<u>xor immediate</u>	xori \$1, \$2,10	$\$1 = \$2 \oplus 10$	Logical XOR reg, constant
<u>shift left logical</u>	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant
<u>shift right logical</u>	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant
<u>shift right arithm.</u>	sra \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right (sign extend)
shift left logical	sllv \$1,\$2,\$3	$\$1 = \$2 \ll \$3$	Shift left by variable
shift right logical	srlv \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right by variable
shift right arithm.	srav \$1,\$2, \$3	$\$1 = \$2 \gg \$3$	Shift right arith. by variable

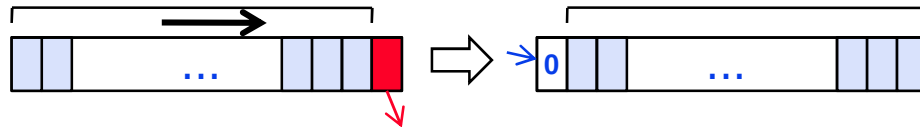
Nota1: "not" non esiste... si fa con nor e un operando \$0

Nota2: "nor i" non esiste... si fa con la andi e negando gli operandi

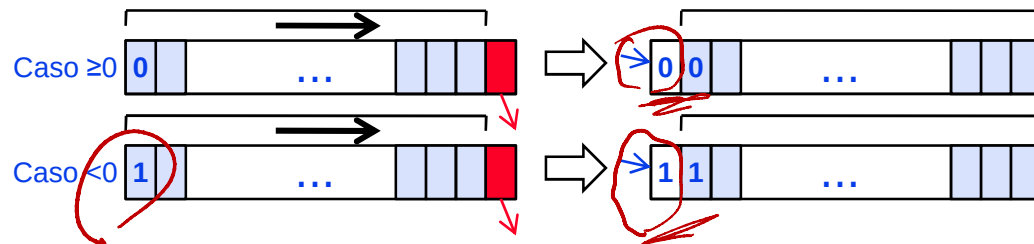
TRASLAZIONE (SHIFT)



Shift Left Logical (SLL) (~~FSL~~ Δ)



Shift Right Logical (SRL)



Shift Right Arithmetic (SRA)

Nota: SLA non esiste ...=SLL

Istruzioni per la manipolazione di stringhe

- Fino a questo punto abbiamo visto istruzioni che operano su quantità a 32 bit

- Per accedere al singolo byte sono a disposizione

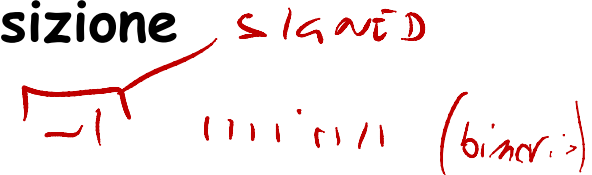
- Utile per le stringhe di caratteri ASCII

lb \$5, 0(\$4)

"load byte"

sb \$5, 0(\$4)

"store byte"



- Per accedere alla half-word (16 bit) ci sono

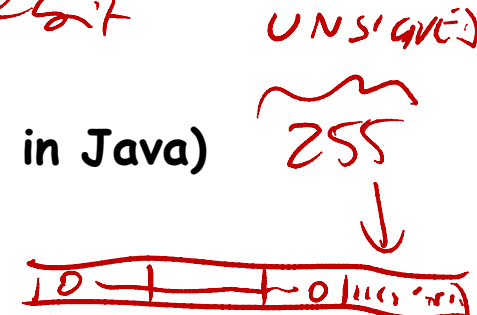
- Utile per le stringhe di caratteri UNICODE (es. in Java)

lh \$5, 0(\$4)

"load half-word"

sh \$5, 0(\$4)

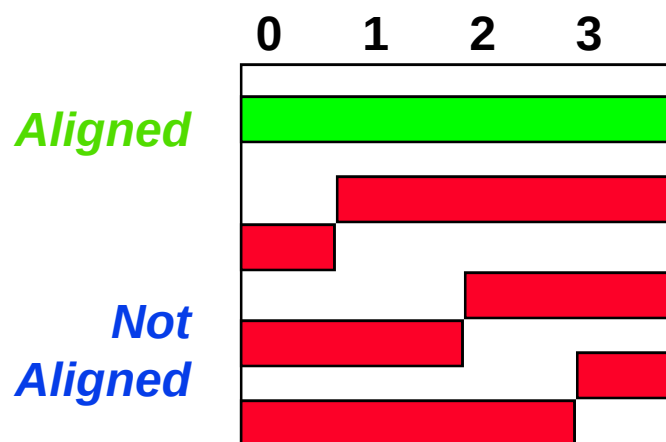
"store half-word"



Nota: in fase di caricamento (load), dovendo porre la quantità da 8 o 16 bit in 32 bit, viene automaticamente effettuata l'estensione del segno. Se ciò non si vuole si devono usare **lbu** (al posto di **lb**) e **lhu** (al posto di **lh**) → estensione con 0

Restrizioni sull'allineamento degli indirizzi

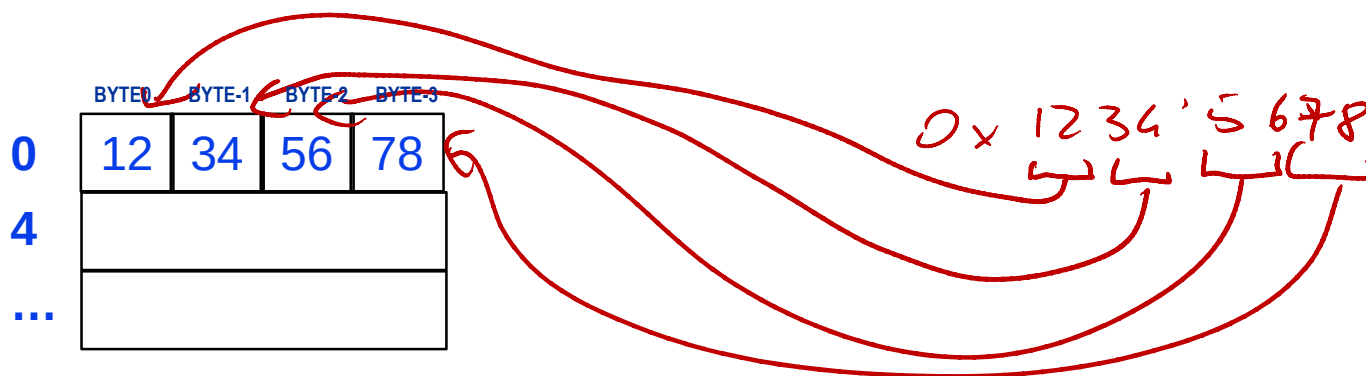
- La memoria e' classicamente indirizzata "al byte"
 - Le istruzioni di load e store usano indirizzi al byte, pero'
 - lw e sw trasferiscono 32 bit
 - lh, lhu e sh trasferiscono 16 bit
 - solo lb, lbu, lb trasferiscono 8 bit
- Nel MIPS, l'indirizzo non puo' essere qualunque...
 - per lw e sw deve essere allineato ad un multiplo di 4
 - Per lh, lhu, sh deve essere allineato ad un multiplo di 2
- Esempi di dati ALLINEATI e NON ALLINEATI "alla word"



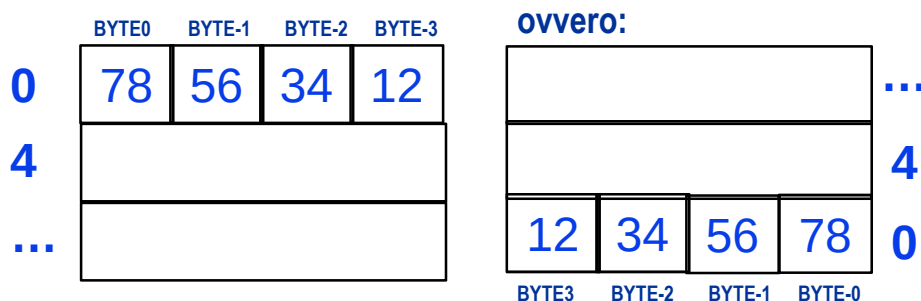
Nota: se si specifica un indirizzo non allineato rispetto a quanto l'istruzione desidera, viene generata una eccezione

Ordinamento dei byte (Endianess)

- **Big Endian:** si inizia con la parte "piu' grossa" ovvero a indirizzi piu' bassi i byte piu' significativi:
IBM 360/370, Motorola 68k, **MIPS**, Sparc(pre-v9)
 - Se ad esempio devo memorizzare 0x12345678 in B.E.



- **Little Endian:** si inizia con la parte "piu' piccola" ovvero a indirizzi piu' bassi i byte meno significativi:
Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



Nota: e' importante sapere l'endianess di una macchina quando si scandiscono stringhe o dati byte-a-byte
SPECIALMENTE dopo un TRASFERIMENTO DA RETE
in cui la macchina remota puo' avere una endianess diversa

Nota2: MIPS piu' recenti - es. quello prodotto da LSI-Logic (LR-3000) - permettono di scegliere la endianess preferita !
Così sono «bi-endian» anche SPARCv9, HP-PA, Alpha, ARM

Esempio: traduzione dei puntatori del C in assembly

```
typedef struct tag {  
    int i;  
    char *cp, c;  
} table;
```

32 bit

DEF.

8 bit

32 bit

```
table z[20];
```

```
table *p;
```

```
char c;
```

Ipotesi:

~~&z ≡ \$7~~
~~&p ≡ \$8~~
~~&c ≡ \$9~~

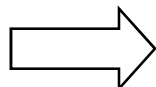
```
p=&z[4];
```



```
addi $1, $7, 36
```

```
sw $1, 0($8)
```

```
c=p->c;
```



```
(addi $1, $7, 36)
```

```
lb $2, 8($1)
```

```
sb $2, 0($9)
```

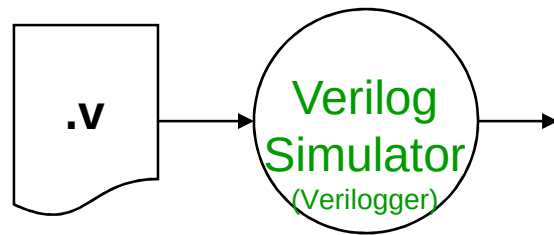
Esercitazione 1

Verilog e MIPS

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Verifica Logica



File Verilog

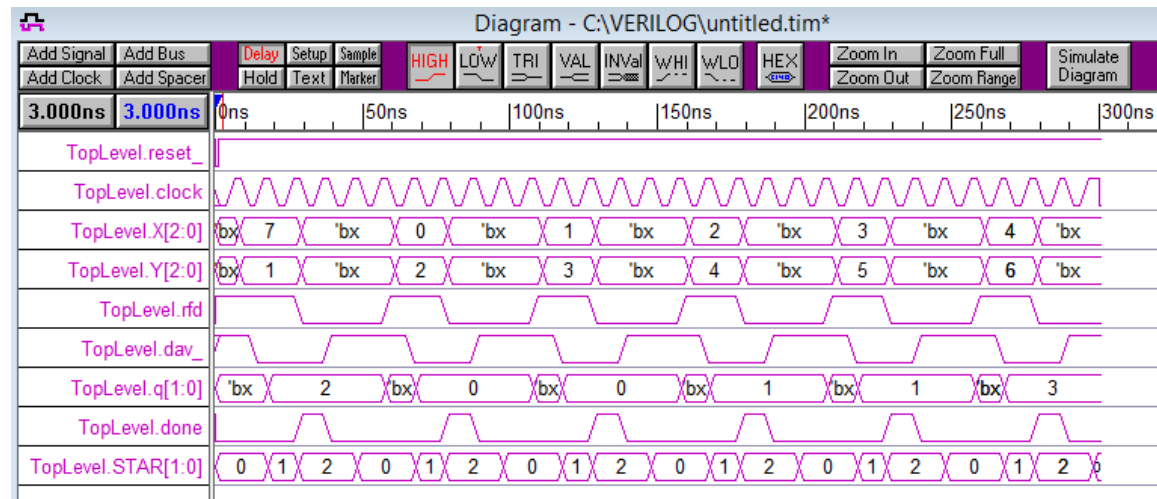


Diagramma Temporale

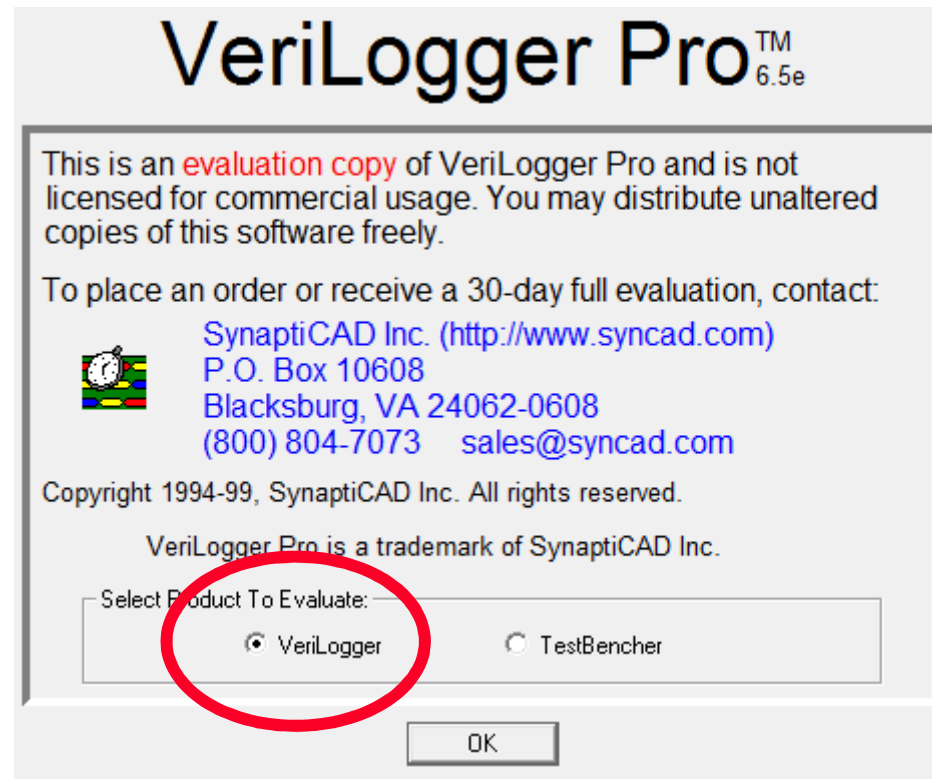
Istallazione di Verilogger

- Distribuibile gratuitamente
- Istruzioni per l'istallazione sul proprio PC (Windows)
 - <http://www.dii.unisi.it/~giorgi/didattica/tools/verilog.exe>
 - Eseguire (scompatta in C:\VERILOG)
 - Crearsi un link (es. dal Desktop) al file C:\VERILOG\vlogev.exe
- Nel Laboratorio
 - Individuare l'icona relativa

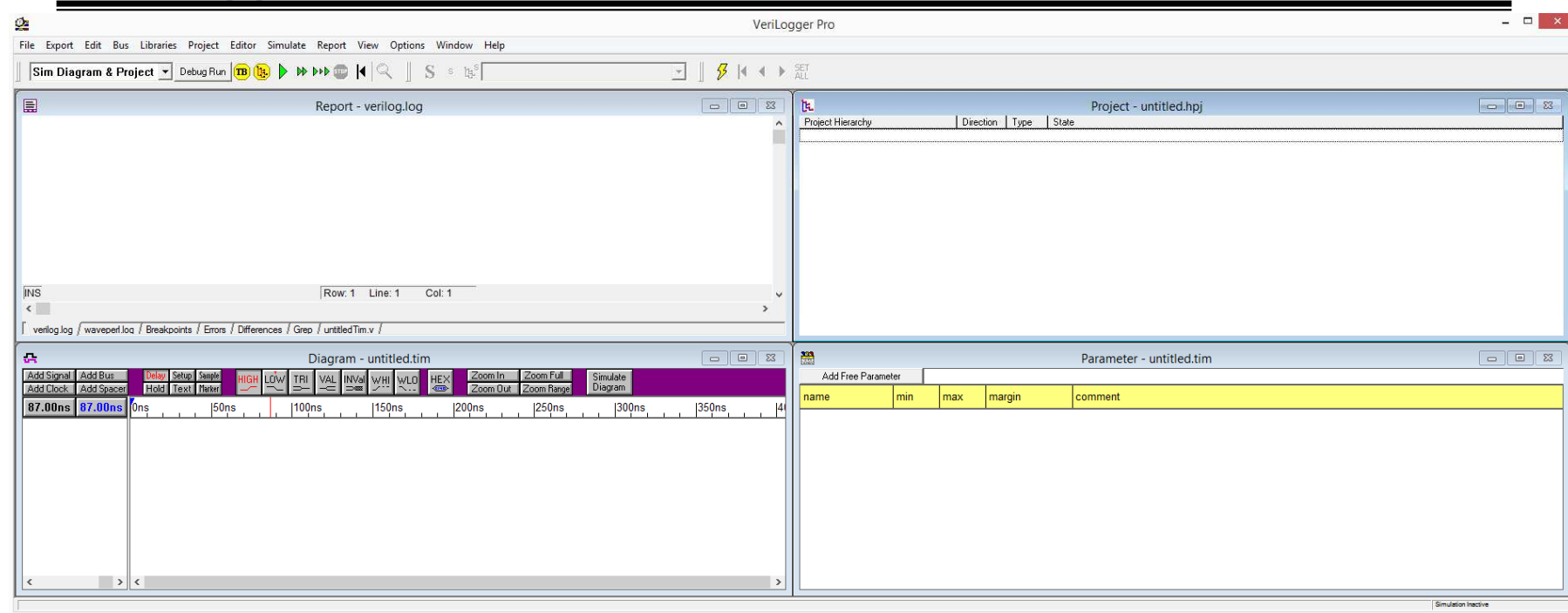


Uso di Verilogger: un primo esempio

- Lanciare il programma Verilogger (vlogev.exe)
 - Selezionare nella prima finestra "VeriLogger"

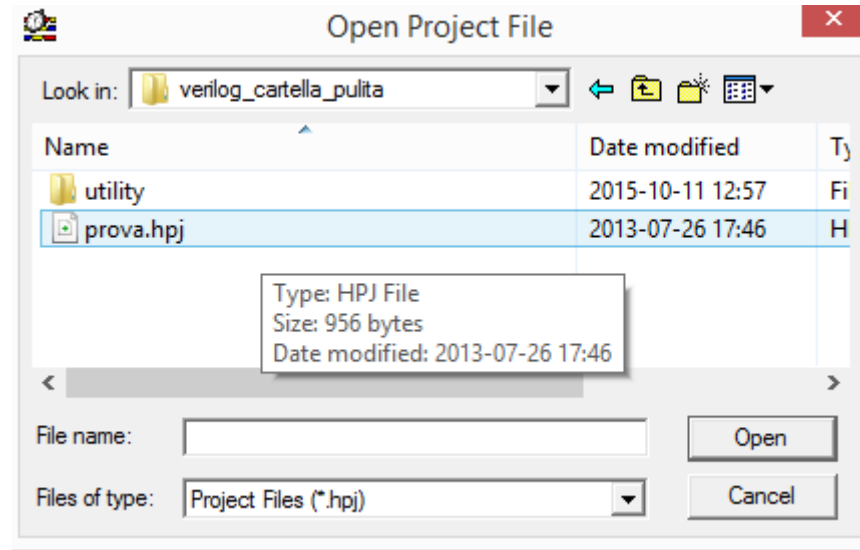
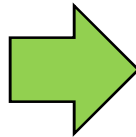
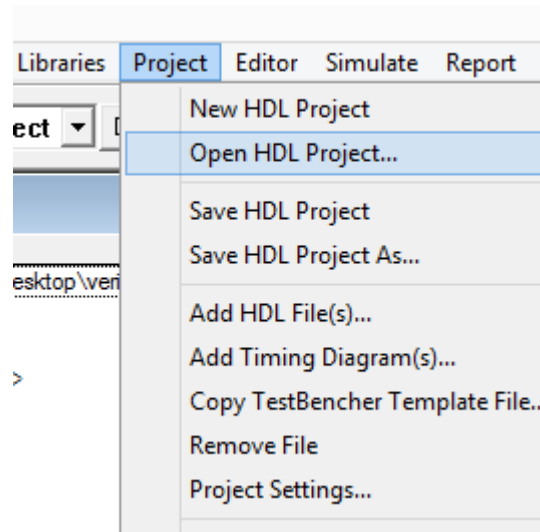


Verillogger: schermata iniziale

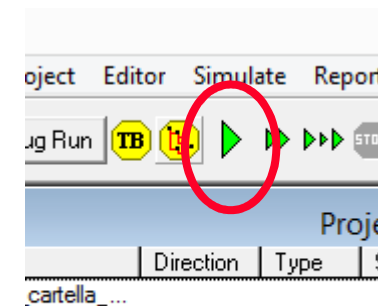
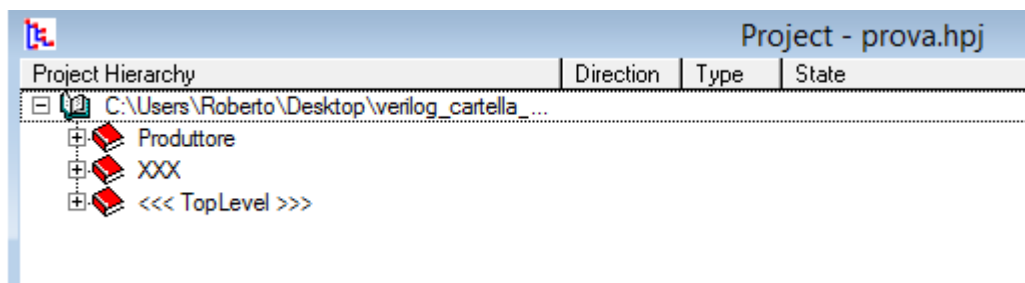
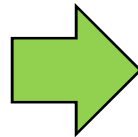
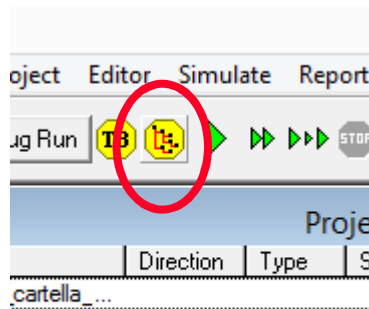


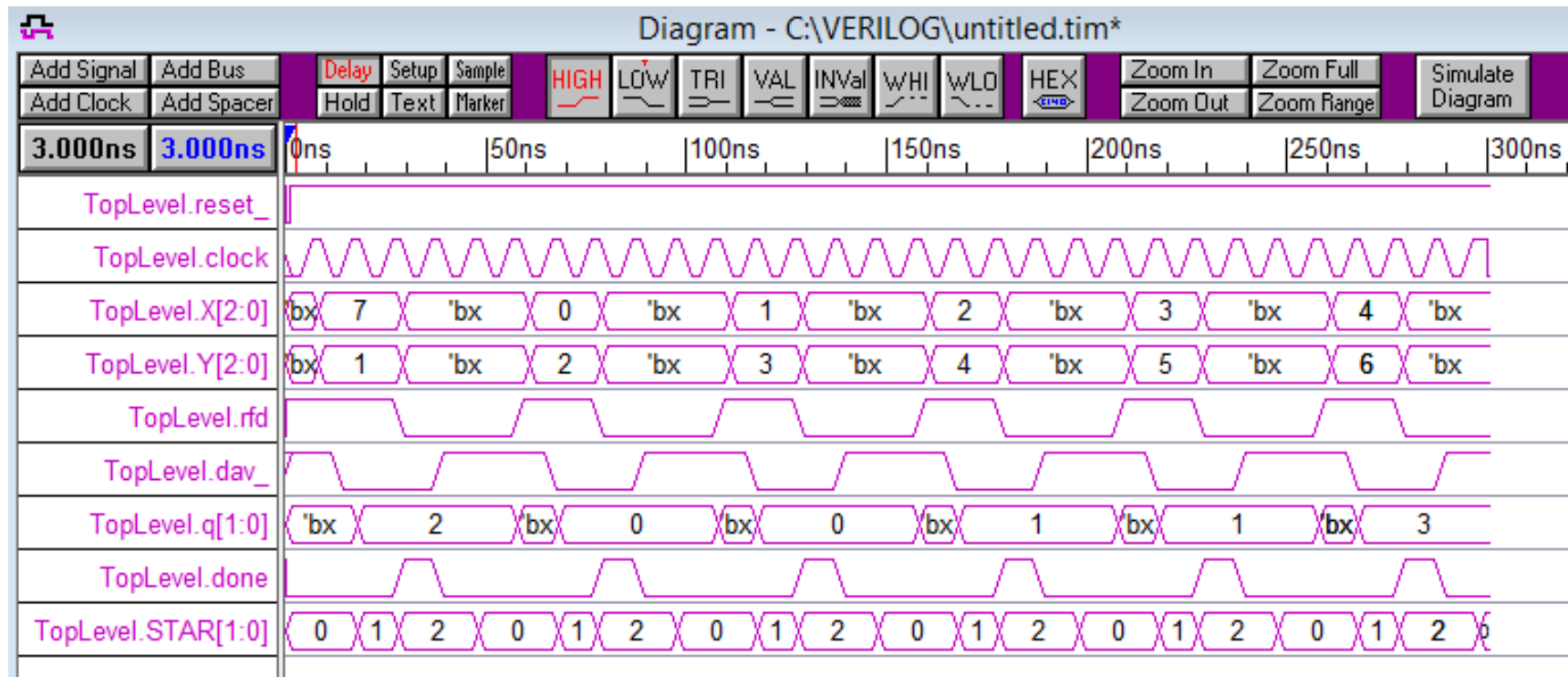
- Sono presenti 4 finestre
 - Report
 - Project
 - Parameter
 - Diagram

Selezione/Creazione di un progetto



Compilazione e generazione del diagramma temporale





Porta nand

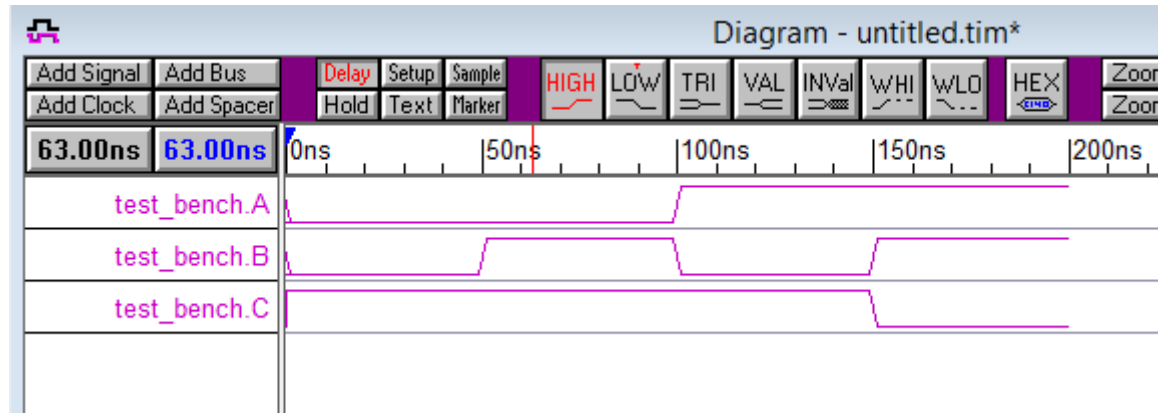
```
`timescale 1ns / 1ps
//create a NAND gate out of an AND and an Inverter
module some_logic_component (c, a, b);
    // declare port signals
    output c;
    input a, b;
    // declare internal wire
    wire d;
    //instantiate structural logic gates
    and a1(d, a, b); //d is output, a and b are inputs
    not n1(c, d);    //c is output, d is input
endmodule

//test the NAND gate
module test_bench; //module with no ports
    reg A, B;
    wire C;

    //instantiate your circuit
    some_logic_component S1(C, A, B);

    //Behavioral code block generates stimulus to test circuit
    initial
    begin
        A = 1'b0; B = 1'b0;
        #50; $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
        A = 1'b0; B = 1'b1;
        #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
        A = 1'b1; B = 1'b0;
        #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
        A = 1'b1; B = 1'b1;
        #50 $display("A = %b, B = %b, NAND output C = %b \n", A, B, C);
    end
endmodule
```

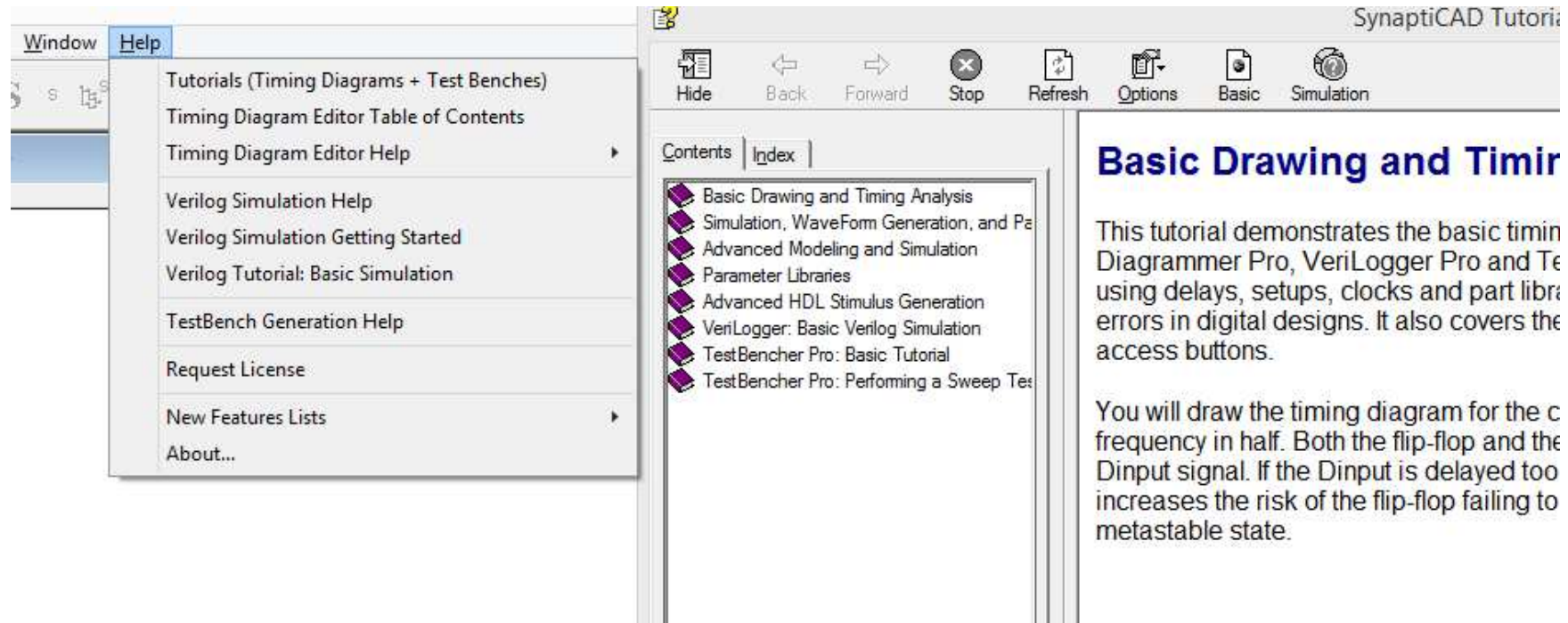
NAND: diagramma e report



Report - C:\VERILOG\verilog.log

Finding handle to syncad_top.test_bench.C
Compile Complete
.
Running...
A = 0, B = 0, Nand output C = 1
A = 0, B = 1, Nand output C = 1
A = 1, B = 0, Nand output C = 1
A = 1, B = 1, Nand output C = 0
0 Errors, 0 Warnings
Compile time = 0.02000, Load time = 0.28400, Execution time = 0.02400

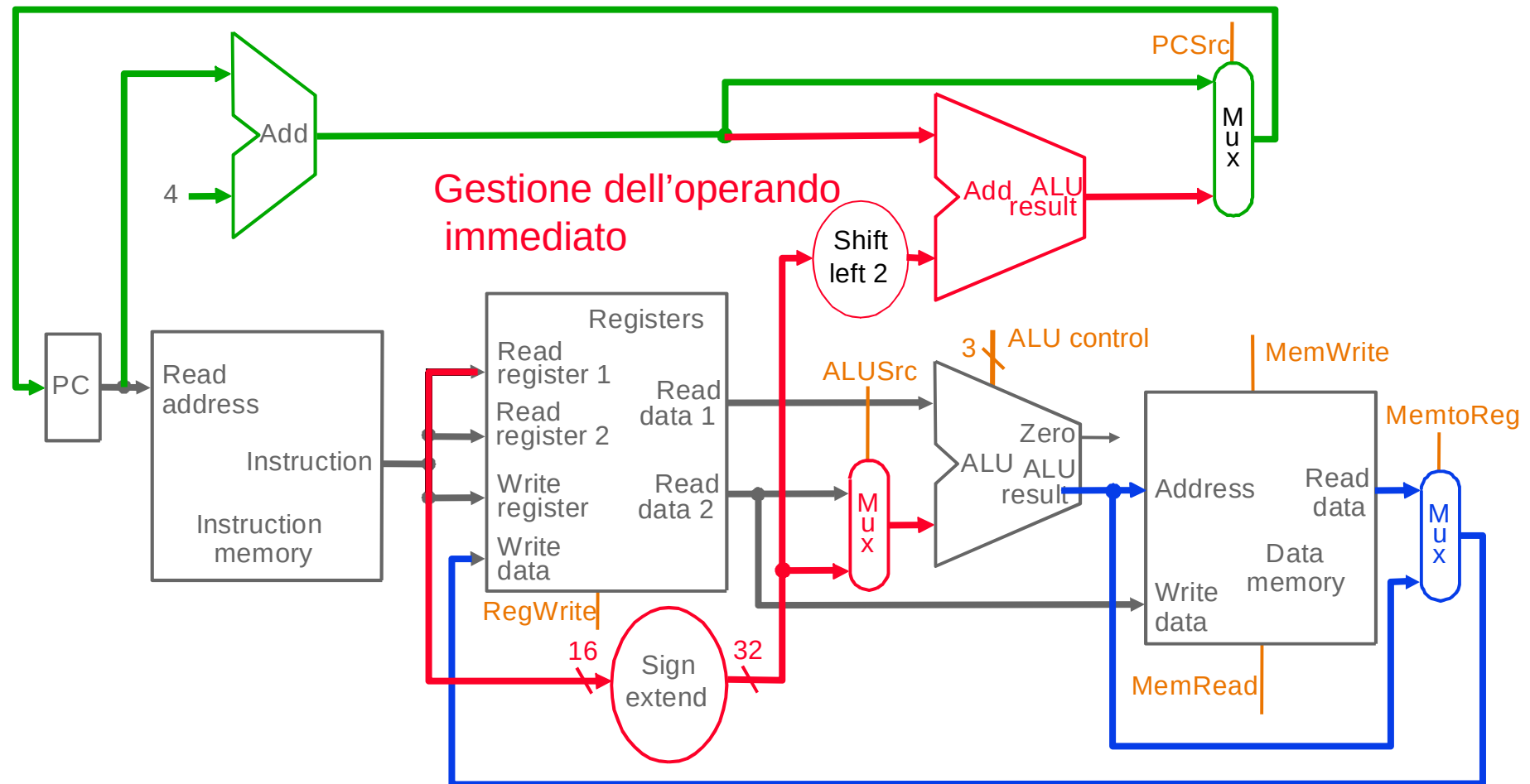
Documentazione



ESEMPIO: COSTRUZIONE DEL DATAPATH DEL MIPS

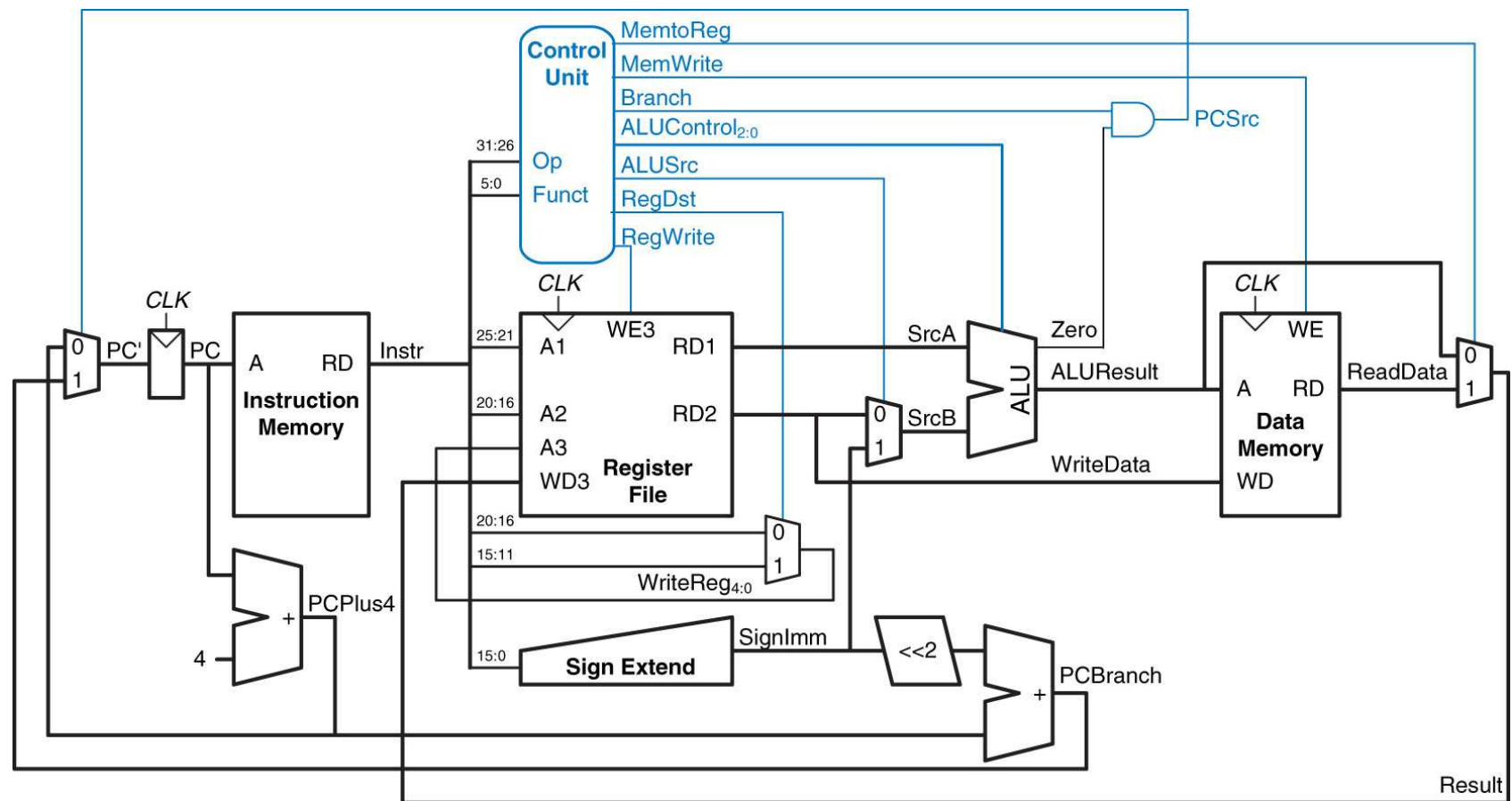
Costruzione del Datapath

Gestione indirizzo istruzione successiva I segnali di controllo vengono pilotati in base a "op-code" e "funct" dell'istruzione:



Gestione del risultato della ALU o dato dalla memoria

Datapath piu' preciso



RegWrite
PCSrc
ALUSrc
AluControl[2:0]
MemWrite
MemtoReg

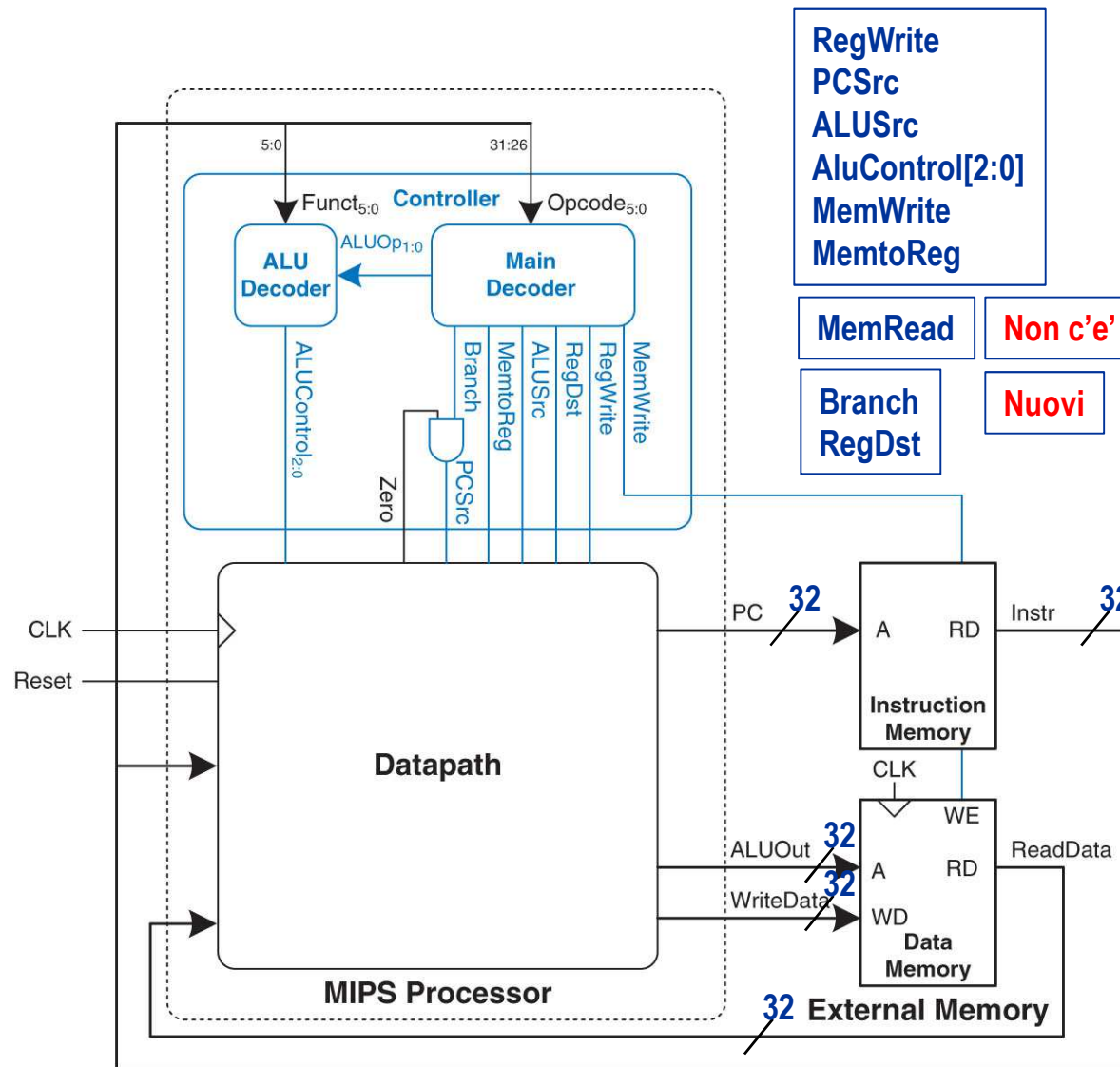
MemRead

Non c'e'

Branch
RegDst

Nuovi

MIPS processor



```
module mips(clk, reset, readdata, instr
    pc, memwrite, aluout, writedata);
input clk, reset;
input [31:0] readdata;
input [31:0] instr;
output [31:0] pc;
output memwrite,
output [31:0] aluout, writedata;
```

```
wire memtoreg, alusrc, regdst,
    regwrite, pcsrc, zero;
wire [2:0] alucontrol;
```

```
controller c(instr[31:26], instr[5:0], zero,
    memtoreg, memwrite, pcsrc,
    alusrc, regdst, regwrite,
    alucontrol);
```

```
datapath dp(clk, reset, memtoreg, pcsrc,
    alusrc, regdst, regwrite,
    alucontrol,
    zero, pc, instr,
    aluout, writedata, readdata);
```

```
endmodule
```

MIPS Controller

```
module controller(opcode, funct, zero,
  zero, memtoreg, memwrite, pcsrc, alusrc,
  regdst, regwrite, alucontrol);

input [5:0] opcode, funct;
input zero;
output memtoreg, memwrite;
output pcsrc, alusrc;
output regdst, regwrite;
output [2:0] alucontrol;

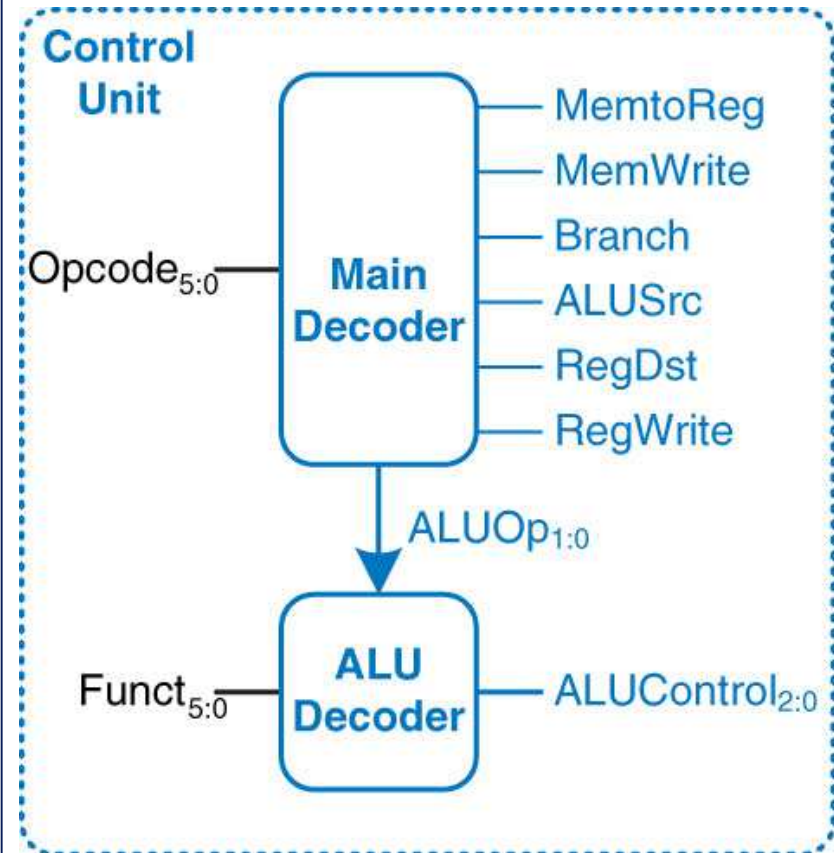
wire [1:0] aluop;
wire branch;

maindec md(opcode, memtoreg, memwrite, branch,
  alusrc, regdst, regwrite, aluop);

aludec ad(funct, aluop, alucontrol);

assign pcsrc = branch & zero;

endmodule
```



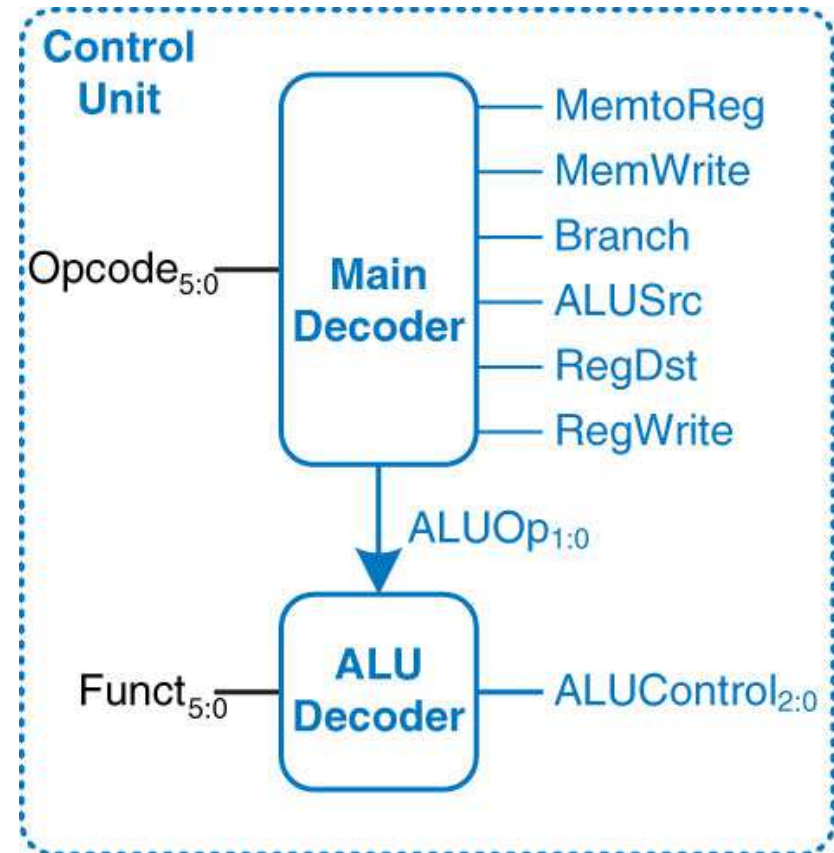
Main Decoder

```
module maindec(opcode, memtoreg, memwrite,
  branch, alusrc, regdst, regwrite, aluop);

input [5:0] opcode;
output memtoreg, memwrite;
output branch, alusrc;
output regdst, regwrite;
output [1:0] aluop;

wire [7:0] controls;
assign {regwrite, regdst, alusrc, branch, memwrite,
  memtoreg, aluop} = controls;

always @(opcode)
  case(opcode)
    6'b000000: controls <= 8'b11000000; // RTYPE
    6'b100011: controls <= 8'b10100100; // LW
    6'b101011: controls <= 8'b00101000; // SW
    6'b000100: controls <= 8'b00010001; // BEQ
    6'b001000: controls <= 8'b10100000; // ADDI
    default: controls <= 8'bxxxxxxxx; // illegal op
  endcase
endmodule
```

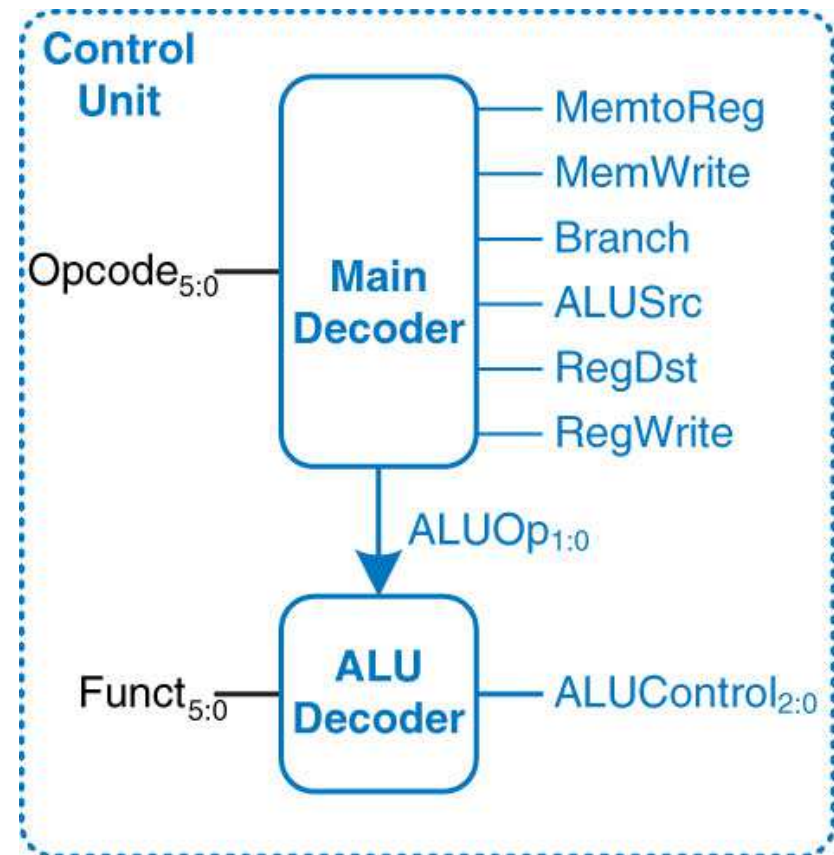


ALU Decoder

```
module aludec(funct, aluop,  alucontrol);

input [5:0] funct,
input [1:0] aluop,
output [2:0] alucontrol;

always @(aluop)
case(aluop)
  2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi)
  2'b01: alucontrol <= 3'b110; // sub (for beq)
  default: case(funct) // R-type instructions
    6'b100000: alucontrol <= 3'b010; // add
    6'b100010: alucontrol <= 3'b110; // sub
    6'b100100: alucontrol <= 3'b000; // and
    6'b100101: alucontrol <= 3'b001; // or
    6'b101010: alucontrol <= 3'b111; // slt
    default: alucontrol <= 3'bxxx; // ???
  endcase
endcase
endmodule
```



MIPS Datapath

Nota: I seguenti esempi vanno adattati alla sintassi Verilog !

```
module datapath(input clk, reset, input memtoreg, pcsrc, input alusrc, regdst, input regwrite, jump,
               input [2:0] alucontrol, output zero, output [31:0] pc, input [31:0] instr, output [31:0] aluout,
               writedata, input [31:0] readdata);

wire [4:0] writereg; wire [31:0] pcnext, pcnextbr, pcplus4, pcbranch; wire [31:0] signimm, signimmsh;
wire [31:0] srca, srcb; wire [31:0] result;

// next PC logic
flopr #(32)  pcreg(clk, reset, pcnext, pc);
adder       pcadd1(pc, 32'b100, pcplus4);
sl2         immsh(signimm, signimmsh);
adder       pcadd2(pcplus4, signimmsh, pcbranch);
mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, jump, pcnext);

// register file logic
Regfile     rf(clk, regwrite, instr[25:21], instr[20:16], writereg, result, srca, writedata);
mux2 #(5)   wrmux(instr[20:16], instr[15:11], regdst, writereg);
mux2 #(32)  resmux(aluout, readdata, memtoreg, result);
signext     se(instr[15:0], signimm);

// ALU logic
mux2 #(32)  srcbmux(writedata, signimm, alusrc, srcb);
alu         alu(srca, srcb, alucontrol, aluout, zero);
endmodule
```

MIPS Registerfile and Adder

```
module regfile(input clk, input we3, input [4:0] ra1, ra2, wa3, input [31:0] wd3, output [31:0] rd1, rd2);

    reg [31:0] rf[31:0];
    // three ported register file
    // read two ports combinatorially
    // write third port on rising edge of clk
    // register 0 hardwired to 0
    // note: for pipelined processor, write third port
    // on falling edge of clk

    always @(posedge clk)
        if (we3) rf[wa3] <= wd3;

    assign rd1 = (ra1 != 0) ? rf[ra1] : 0;
    assign rd2 = (ra2 != 0) ? rf[ra2] : 0;
endmodule
```

```
module adder(input [31:0] a, b, output [31:0] y);
    assign y = a + b;
endmodule
```

Shift-Left-2, Sign-Extension, Resettable Flip-flop, MUX2

```
module sl2(input [31:0] a, output [31:0] y);  
    // shift left by 2  
    assign y = {a[29:0], 2'b00};  
endmodule
```

```
module signext(input [15:0] a, output [31:0] y);  
    assign y = {{16{a[15]}}, a};  
endmodule
```

```
module flopr #(parameter WIDTH = 8)(input clk, reset,  
    input [WIDTH-1:0] d,  
    output [WIDTH-1:0] q);  
    always @(posedge clk, posedge reset)  
        if (reset) q <= 0;  
        else q <= d;  
endmodule
```

```
module mux2 #(parameter WIDTH = 8) (input [WIDTH-1:0] d0, d1,  
    input s, output [WIDTH-1:0] y);  
    assign y = s ? d1 : d0;  
endmodule
```

Instruction Memory and Data Memory

```
module imem(input [5:0] a, output [31:0] rd);  
  reg [31:0] RAM[63:0];  
  
  // initial $readmemh("memfile.dat", RAM);  
  
  assign rd = RAM[a]; // word aligned  
endmodule
```

```
module dmem(input clk, we, input [31:0] a, wd, output [31:0] rd);  
  reg [31:0] RAM[63:0];  
  
  assign rd = RAM[a[31:2]]; // word aligned  
  always @(posedge clk)  
    if (we) RAM[a[31:2]] <= wd;  
endmodule
```

Top module

```
module top(input clk, reset, output [31:0] writedata, dataadr, output memwrite);
  reg [31:0] pc, instr, readdata;

  // instantiate processor and memories
  mips mips(clk, reset, pc, instr, memwrite, dataadr, writedata, readdata);
  imem imem(pc[7:2], instr);
  dmem dmem(clk, memwrite, dataadr, writedata, readdata);

endmodule
```

Esercizi

- Provare a descrivere in Verilog le reti combinatorie notevoli che abbiamo visto a lezione

Lezione 3

Assembly MIPS (3)

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Indirizzamento nelle istruzioni Branch e Jump

- Per variare il flusso sequenziale delle istruzioni si usa:

bne \$t4,\$t5,Label

l'istruzione successiva e' a Label se \$t4 != \$t5

beq \$t4,\$t5,Label

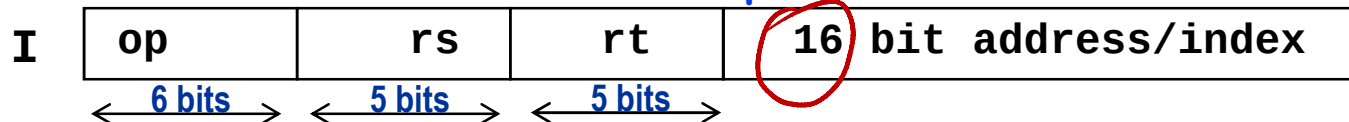
l'istruzione successiva e' a Label se \$t4 == \$t5

j Label

l'istruzione successiva e' a Label

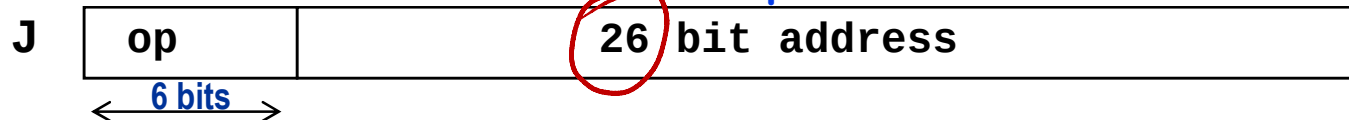
- Per la codifica dell'istruzione, nel caso di bne, beq dovendo inserire anche gli indici di due registri si usa

- Formato I → restano 16 bit per codificare DOVE saltare



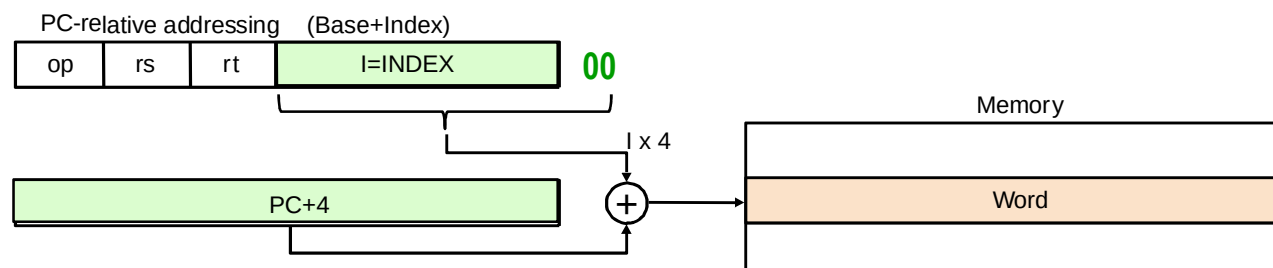
- Nel caso di j (jump) si usa

- Formato J → restano 26 bit per codificare DOVE saltare



- Come si possono sfruttare al massimo questi bit?

Indirizzamento relativo al PC (bne, beq)



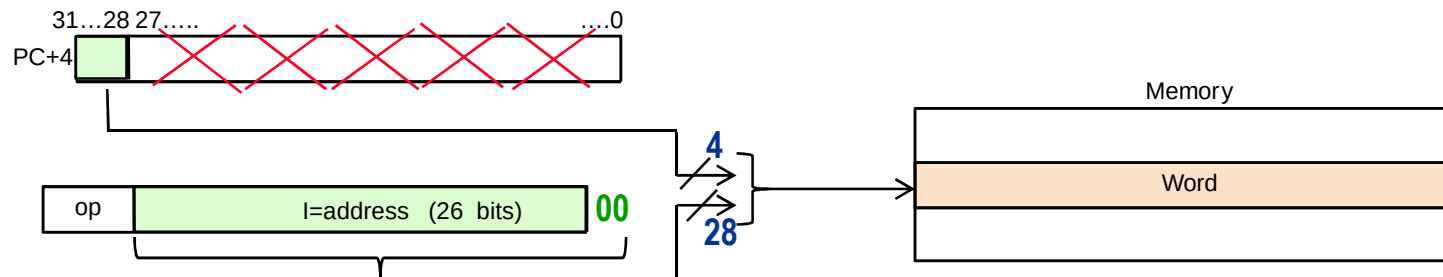
- E' anche noto come indirizzamento 'Base+Indice'
- I branch tipicamente saltano vicino (principio di localita': la distanza e' quasi sempre <20 istr.)

$$A = (PC+4) + 4 \cdot I$$

Nota: "PC+4" tiene conto del valore assunto da PC nel momento del calcolo dell'indirizzo effettivo A

- A= indirizzo effettivo utilizzato
 - PC=Program Counter DELL'ISTRUZIONE
 - I= indice (campo 'index' dell'istruzione), puo' essere negativo
- Poiche' l'indirizzo di una istruzione e' sempre multiplo di 4 il campo effettivo in cui posso spaziare e' 2^{18} (256KB)

Indirizzamento Pseudoindiretto



$A = \{ (PC+4)[31:28], I*4 \}$ ← Notazione Verilog: significa «concatenamento»

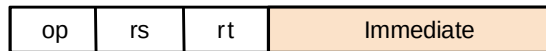
- A = indirizzo effettivo utilizzato
- PC = Program Counter
- I = indice (campo 'address' dell'istruzione)

- Così l'ampiezza massima di salto è 2^{28} (256 MB)
 - La memoria è quindi suddivisa in 16 «fette» ampie 256MB
 - Il loader e il linker controllano che non si scavalchino "confini" di 256MB

Nota: per saltare ad un indirizzo qualunque a 32 bit si usa l'istruzione `jr` (jump relative) ovvero la coppia `la $x, addr; jr $x`

Tabella riassuntiva modi indirizzamento MIPS:

1. Immediate addressing



addi, andi, ori
slti, lui
addiu, sltiu

2. Register addressing: (direct addressing)

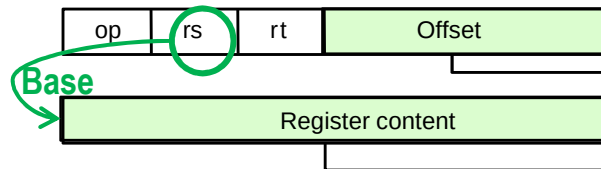


Registers

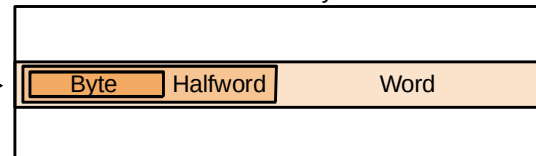
Register content

add, sub, mult, div, slt
addu, subu, multu, divu, sltu
and, or, xor, nor,
sll, srl, sra
mfhi, mflo, mthi, mtlo
jr

3. Base addressing (Base+Offset)

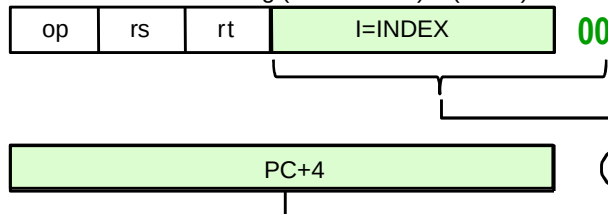


Memory

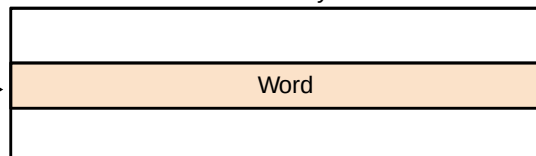


lw, lh, lb
lhu, lbu
sw, sh, sb

4. PC-relative addressing (Base+Index) $(PC+4) + 4 \times I$

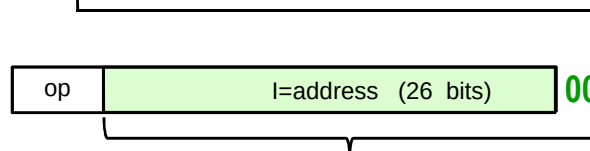
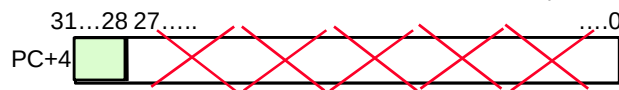


Memory

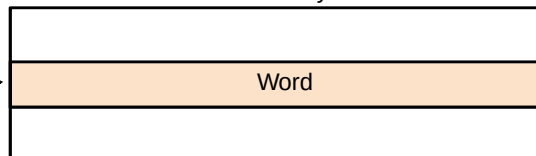


bne, beq

5. Pseudodirect addressing: $PC = (PC+4) [31..28] \mid \text{address} \times 4$



Memory



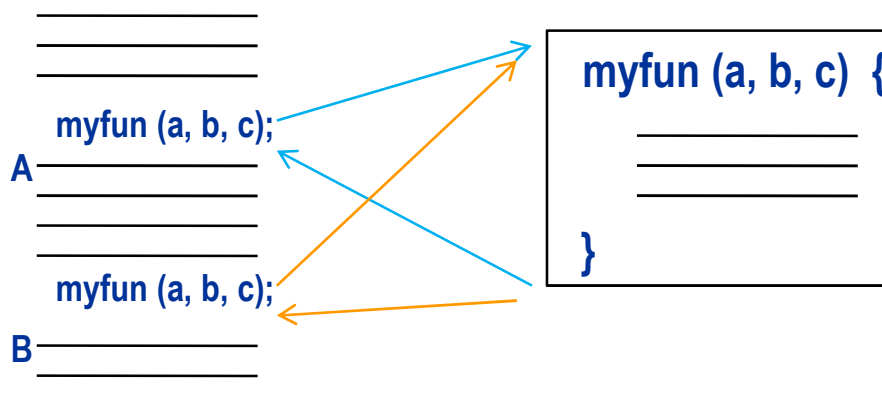
j, jal

jal=jump-and-link

Chiamata a funzione (1)

Patterson 2.7

- 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" e' 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 **NON PUO'** essere implementata con una semplice `j` o `bne/beq` → OCCORRE:

- i) memorizzare da qualche parte il punto di ritorno (es. "A");
- ii) saltare all'inizio della funzione "myfun"

- In Assembly MIPS questi due passi sono effettuati dalla istruzione `jal` → `jal myfun` salta a myfun mettendo l'indirizzo di ritorno nel registro `$ra`



Operazioni connesse alla chiamata di funzione

1) eventuale **salvataggio** registri da preservare
(es. \$t..., \$a..., nello stack param. oltre il quarto)

pre-chiamata
(lato chiamante)

2) preparazione dei parametri di ingresso (nuovi \$a...)

3) chiamata della funzione

jal MYFUN

4) allocazione del call-frame nello stack

5) eventuali **salvataggi** vari
(es. \$a0-\$a3, \$ra, \$fp, \$s0-\$s7)

prologo (lato chiamato)

6) eventuale inizializzazione nuovo \$fp

7) esecuzione del codice della funzione

corpo (lato chiamato)

8) preparazione dei parametri di uscita (\$v0-\$v1)

9) ripristino dei parametri salvati (salvati al punto 5)

epilogo (lato chiamato)

10) ritorno al codice originario

jr \$ra

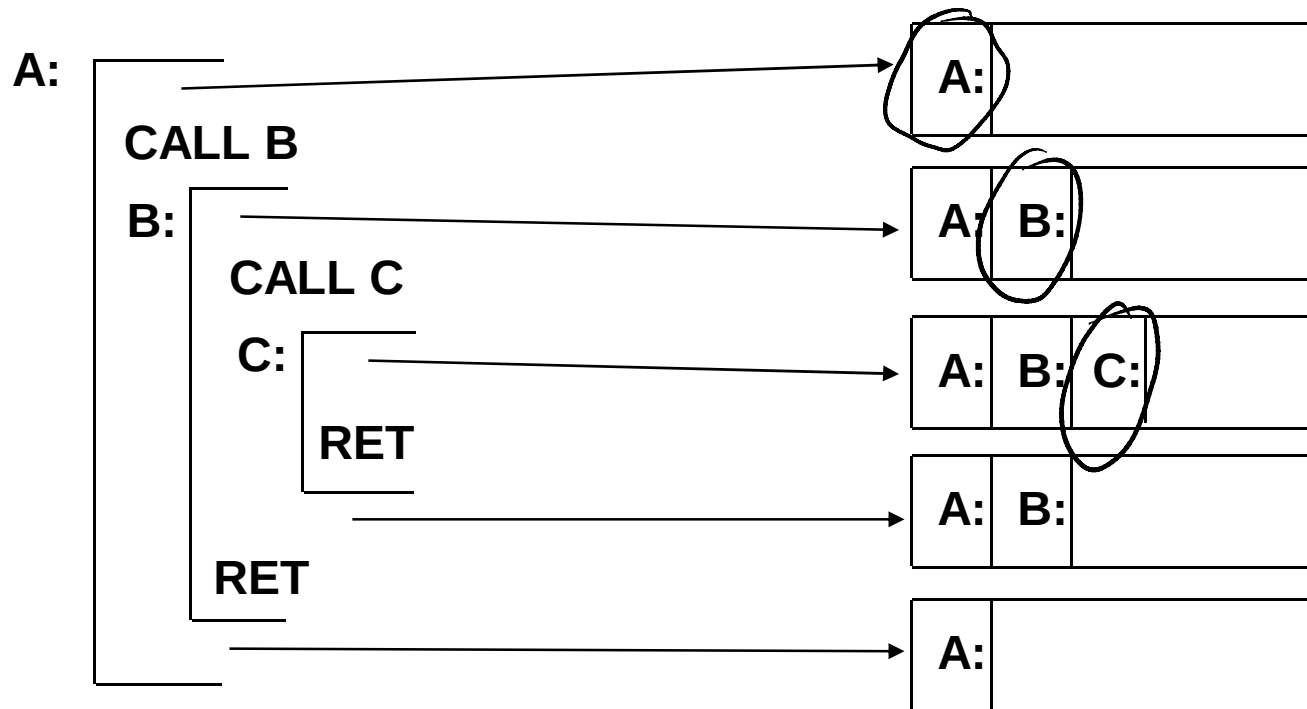
11) eventuale ripristino dei vecchi valori (es. \$t... \$a...)

12) uso dei valori di uscita della funzione

post-chiamata
(lato chiamante)

- I vari "**salvataggi**" vengono fatti in una zona di memoria chiamata "record di attivazione" (o "call frame")
- I call-frame sono gestiti con politica **LIFO** (Last-In, First-Out), e quindi conviene gestirli con uno **STACK**

Uso dello stack nelle chiamate a funzione

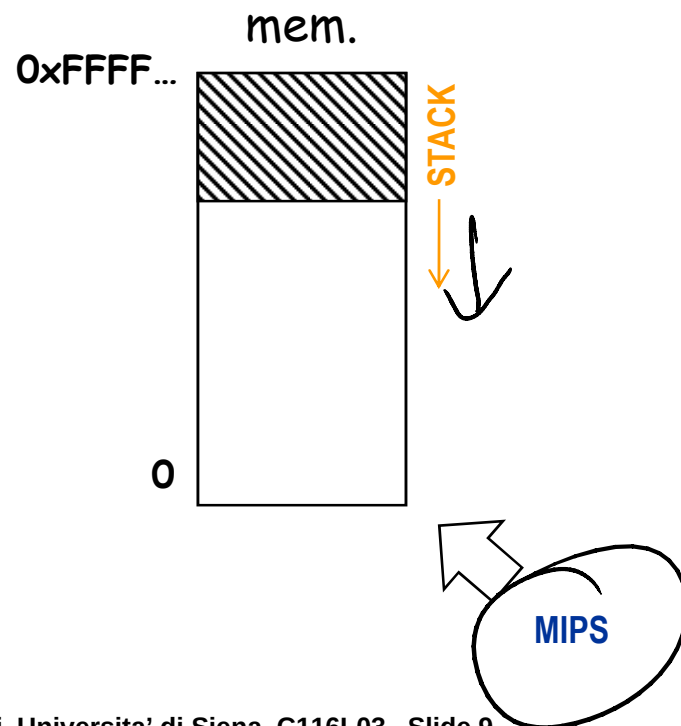


- Alcune macchine forniscono la gestione di stack come parte della architettura stessa (e.g. VAX, processori Intel)
 - Istruzioni PUSH e POP
- Gli stack possono comunque anche essere implementati via software (e.g. MIPS)

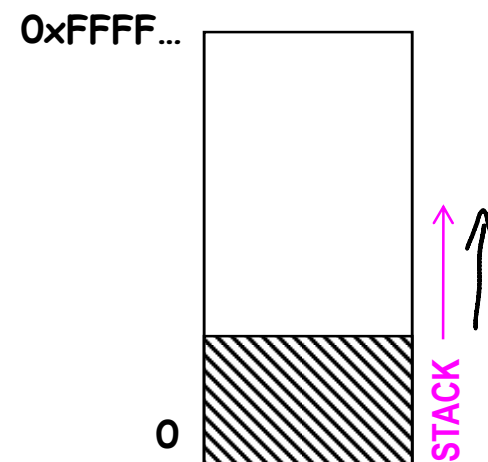
Convenzione sugli stack (1): “verso di crescita”

- Gli stack possono crescere verso l'alto oppure verso il basso
- Sto assumendo che gli indirizzi di memoria crescano verso l'alto
 - Con '0xFFFF...' indico gli indirizzi alti in memoria
 - Con '0' indico gli indirizzi bassi in memoria

Convenzione grow down

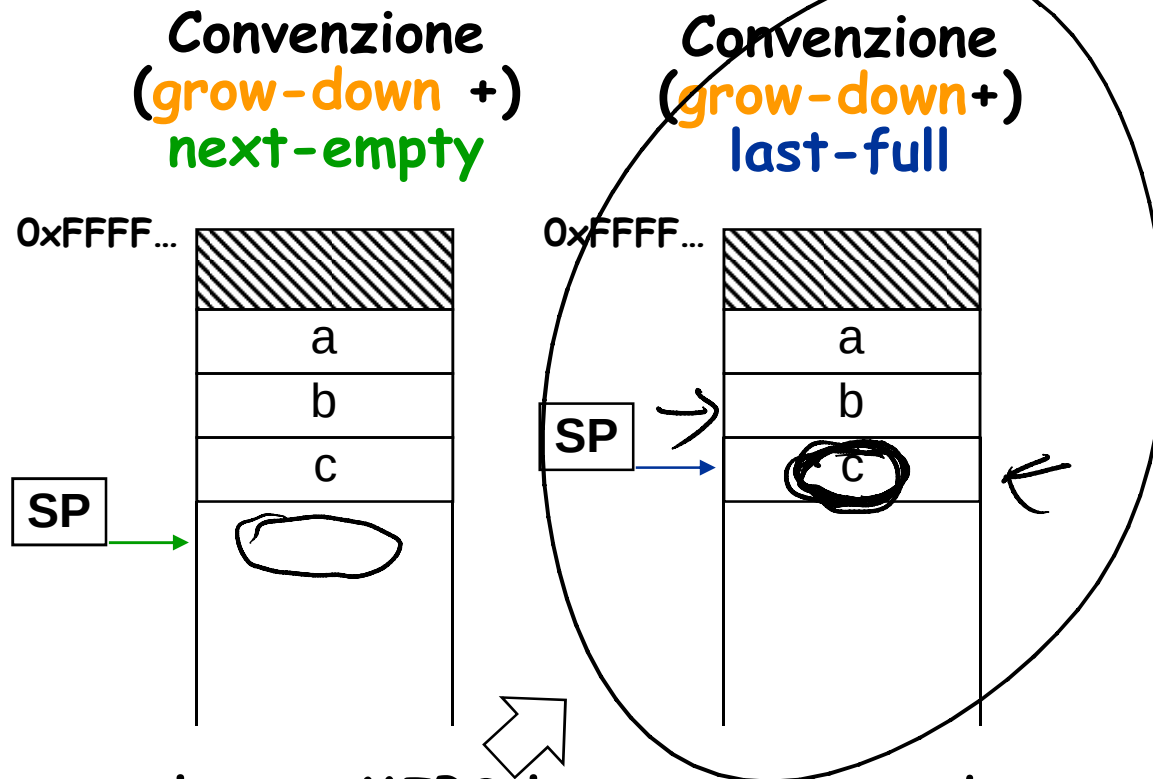


Convenzione grow up



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

- Convenzione su 'a cosa punta SP': **Next-Empty** vs. **Last-Full**



- nel caso MIPS la convenzione e':
 - Grow-down + Last-Full**
 - SP e' nel registro \$sp**

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]

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 MIPS per le chiamate a funzioni (1)

PRE-CHIAMATA (LATO CHIAMANTE)

1) Eventuale salvataggio registri da preservare nel chiamante

- Si assume che \$a0-\$a3 e \$t0-\$t9 possano essere sovrascritti
 - se li si vuole preservare vanno salvati nello stack (dal chiamante)
 - in particolare (i vecchi valori di) \$a0-\$a3

2) Preparazione degli argomenti della funzione

- I primi 4 argomenti vengono posti in \$a0-\$a3 (nuovi valori)
- Gli eventuali altri argomenti oltre il quarto 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 MIPS 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-a3 solo se la funzione ha necessita' di riusarli nel corpo di questa funzione, successivamente a ulteriori chiamate a funzione che usino tali registri, (nota: negli altri casi a0-a3 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 salvare \$s0-\$s7 se intendo sovrascrivere tali registri (il chiamante si aspetta di trovarli intatti)

6) Eventuale inizializzazione di \$fp al nuovo call-frame

CORPO DELLA FUNZIONE

7) CODICE EFFETTIVO DELLA FUNZIONE

Convenzione MIPS per le chiamate a funzioni (3)

EPILOGO (LATO CHIAMATO)

8) Se deve essere restituito un valore dalla funzione

- Tale valore viene posto in \$v0 (e \$v1)

9) I registri (se salvati) devono essere ripristinati

- \$s0-\$s7
- \$ra
- \$fp

Notare che \$sp deve solo essere aumentato di opportuno offset (lo stesso sottratto nel punto 4)

RITORNO AL CHIAMANTE

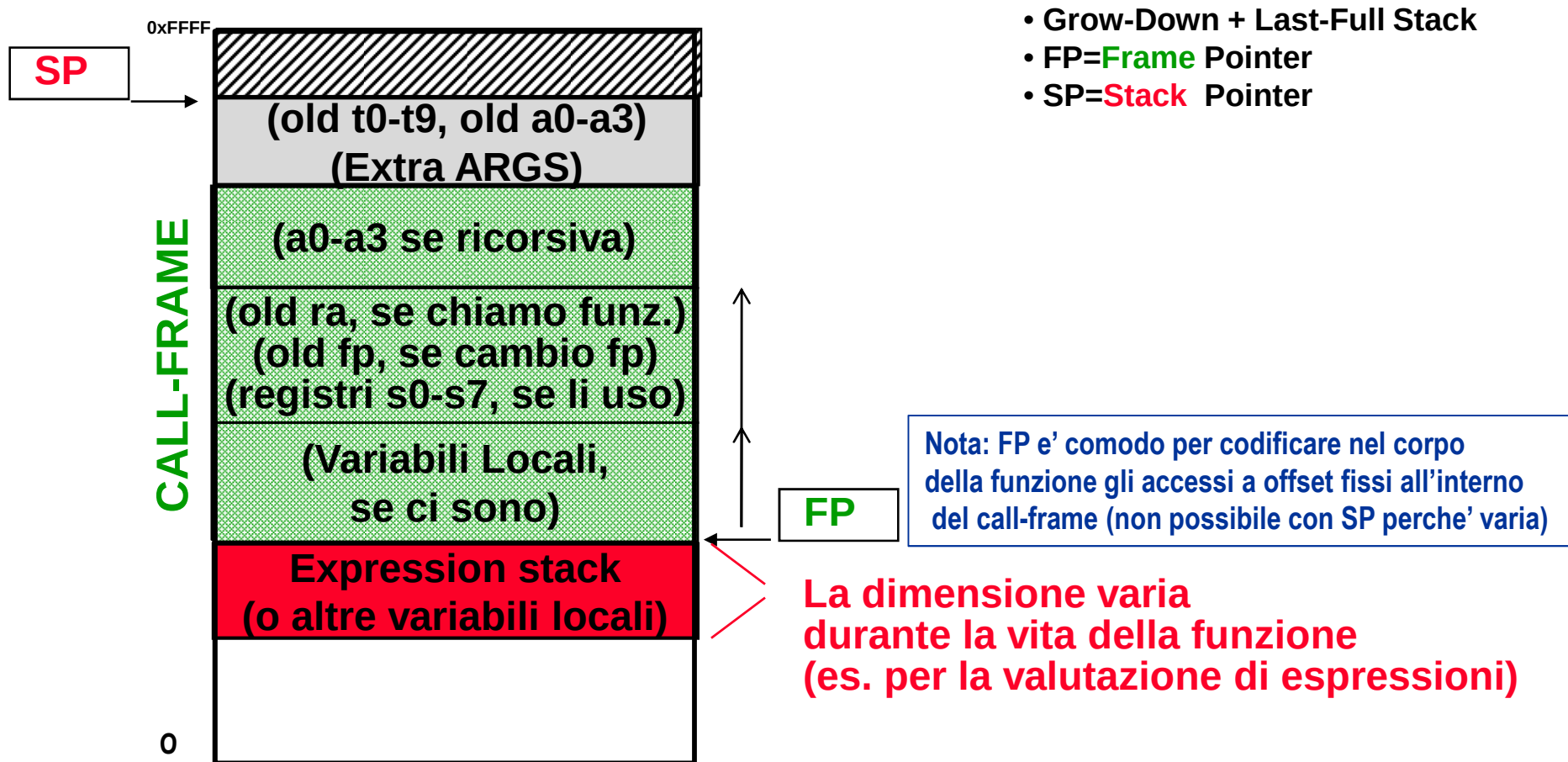
10) jr \$ra

POST-CHIAMATA (LATO CHIAMANTE)

11) Ripristino dei valori \$t... \$a... (vecchi) eventualmente salvati

12) Eventuale uso del risultato della funzione (in \$v0 (e \$v1))

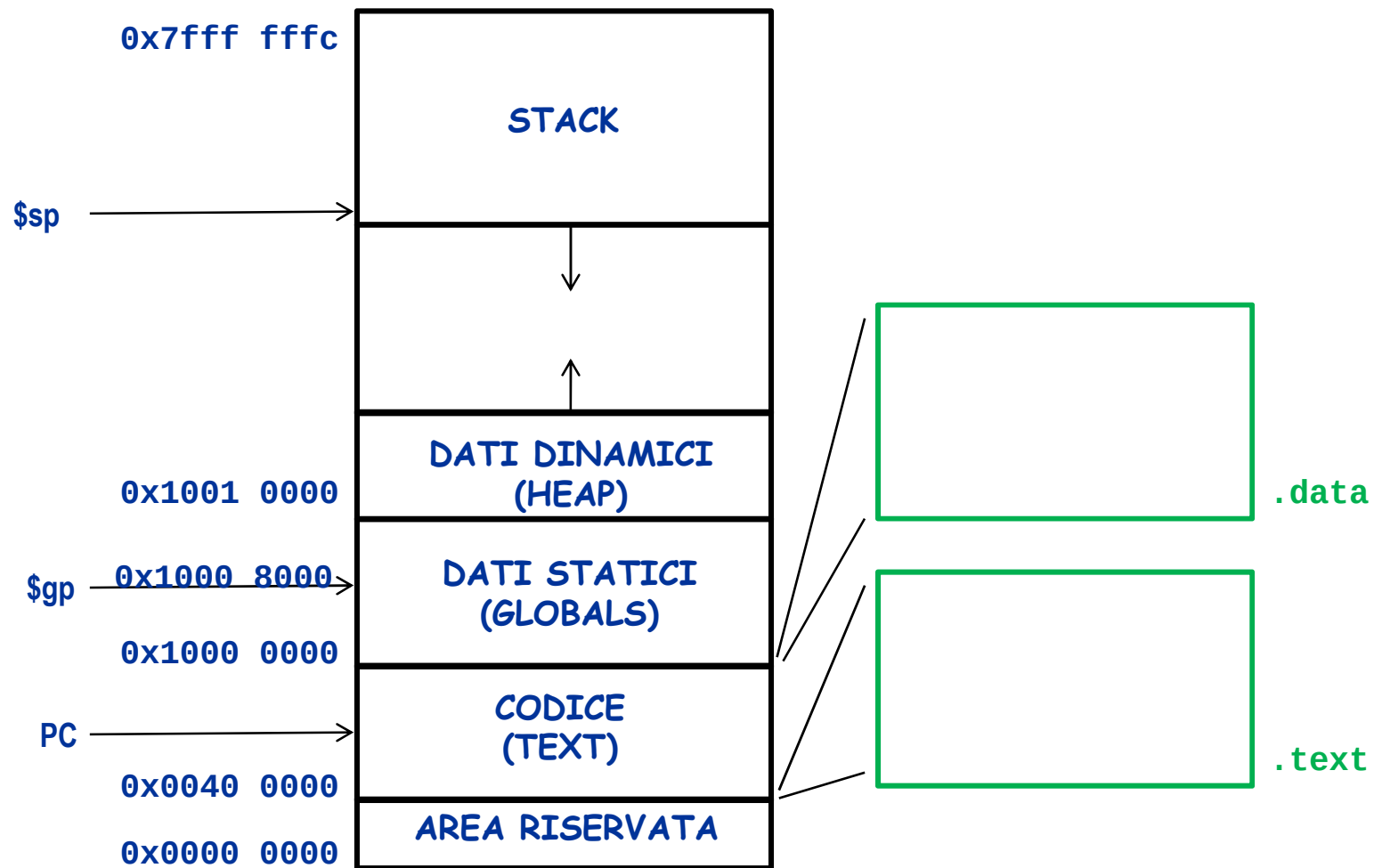
Riepilogo: struttura del Call-Frame



- I linguaggi strutturati inseriscono un'ulteriore informazione: (e.g. Pascal) un link al frame di attivazione 'padre'
- I compilatori normalmente cercano di allocare le variabili locali prima di tutto nei registri, non in memoria (o stack)!

Memory Layout

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



Programma fattoriale.s (1)

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

.text
.globl main

# Funzione per il calcolo del fattoriale
# Parametri:    n : $a0 ($4)
# Risultato:    n! : $v0 ($2)

fact:

# crea il call frame sullo stack (12 byte)
# 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    $fp, 0($sp)       # salvataggio del precedente frame pointer
    add   $fp, $sp, $0       # aggiornamento del frame pointer

# calcolo del fattoriale
    bne   $a0, $0, Ric      # test fine ricorsione n!=0
    addi  $v0, $0, 1         # 0! = 1
    j     Fine

Ric:
    addi  $a0, $a0, -1       # chiamata ricorsiva per il calcolo di (n-1)!
    jal   fact              # $a0 ← (n - 1) passaggio del parametro in $a0 per fact(n-1)
    lw    $t0, 8($fp)       # chiama fact(n-1) → risultato in $v0
    mult  $v0, $t0          # $t0 ← n
    mflo  $v0              # n! = (n-1)! x n

# uscita dalla funzione
Fine:
    lw    $fp, 0($sp)       # recupera il frame pointer
    lw    $ra, 4($sp)       # recupera l'indirizzo di ritorno
    addi  $sp, $sp, 12      # elimina il call frame dallo stack
    jr    $ra              # ritorna al chiamante
```


Programma fattoriale.s (2)

```
# Programma principale
#
main:
# Stampa intestazione
    la    $a0, messaggio
    addi $v0, $0, 4
    syscall

# Stampa richiesta
    la    $a0, insdati
    addi $v0, $0, 4
    syscall

# legge n (valore in $v0)
    addi $v0, $0, 5
    syscall

# chiama fact(n)
    add $a0, $v0, $0      # parametro n in a0
    jal fact
    add $s0, $v0, $0      # salva il risultato in s0

# stampa messaggio per il risultato
    la    $a0, visris
    addi $v0, $0, 4
    syscall

# stampa n!
    add $a0, $s0, $0
    addi $v0, $0, 1
    syscall

# stampa \n
    la    $a0, RTN
    addi $v0, $0, 4
    syscall

# exit
    addi $v0, $0, 10
    syscall
```

Ulteriore materiale di riferimento:
Appendice A, Patterson/Hennessy

Principali direttive per l'assemblatore

<code>.data [<addr>]</code>	marca l'inizio di una zona dati; se <addr> viene specificato i dati sono memorizzati a partire da tale indirizzo
<code>.text [<addr>]</code>	marca l'inizio del codice assembly; se <addr> viene specificato i dati sono memorizzati a partire da tale indirizzo
<code>.globl <symb></code>	dichiara un simbolo <symb> visibile dall'esterno (gli altri sono locali per default) in modo che possa essere usato da altri file
<code>.ascii <str></code>	mette in memoria la stringa <str>, seguita da un byte '0'

Ulteriori direttive assembler (1)

.ascii <str>

Mette in memoria la stringa <str> e non inserisce lo '0' finale

.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 quantita' a 16 bit (in locazioni di memoria contigue)

.space <n>

Mette in memoria n spazi

.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`

Ulteriori direttive assemblatore (2)

.extern <symbol> [<size>]

Dichiara che il dato memorizzato all'indirizzo <symbol> e' un simbolo globale ed occupa <size> byte di memoria. 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 <symbol> riguardano un oggetto esterno ad un modulo (torneremo su questo fra poco)

.kdata [<addr>]

Marca l'inizio di una zona dati pertinente al kernel. Se e' presente il parametro opzionale <addr>, tali dati sono memorizzati a partire da tale indirizzo.

.ktext [<addr>]

Marca l'inizio di una zona contenente codice del kernel. Se e' presente il parametro opzionale <addr>, tali dati sono memorizzati a partire da tale indirizzo.

Alcune syscall

- 'syscall' serve per chiedere un servizio al sistema operativo (letteralmente 'system call')
 - Il numero del servizio e' indicato in \$v0
 - 1: stampa un intero a video
(\$a0 contiene l'intero in binario da stampare)
 - 4: stampa un messaggio a video
(\$a0 indirizzo della stringa da stampare)
 - 5: leggi un intero dalla tastiera
(\$v0 conterra' l'intero letto in binario)
 - 10: termina il programma
(il sistema operativo riprende il controllo del calcolatore)

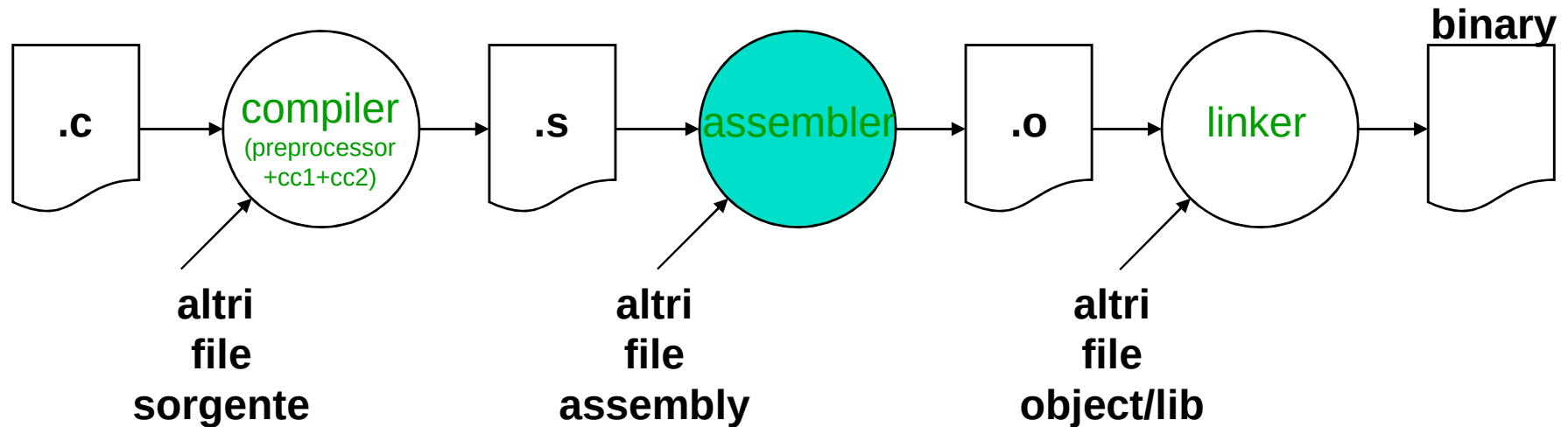
Compilazione di un programma strutturato a moduli

- 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 piu' 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 cosi' 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 piu' stadi (cc1, cc2) fino ad ottenere il livello di ottimizzazione desiderato
 - il risultato intermedio e' un programma in **ASSEMBLY**
 - il codice **ASSEMBLY** e' legato all'architettura target
- **Creazione del file oggetto**
 - questo passo e' eseguito dall'assemblatore
 - il file oggetto contiene
 - la codifica delle istruzioni assembly
 - i dati statici
 - informazioni di rilocazione
 - tabella dei simboli
 - altre informazioni di utilita'

Assemblatore



- **UNIX:**
E' possibile produrre l'output in assembly del compilatore,

`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):

```
      add  $18, $18, $0      LC=0
      addi $19, $19, 0       LC=4
loop: add  $4, $18, $0       LC=8
      add  $19, $19, $4      LC=12
      add  $18, $18, $19     LC=16
      bne  $18, $5, loop     LC=20
      la   $18, myvar        LC=24
      .....
      .....
etic1: .....               LC=1000
      .....
      .....
      j    etic1            LC=3000
      .....
myvar: .....               LC=7000
```

Symbol Table (provvisoria)

symbol	Value of Location Counter (LC)
loop	8
etic1	1000
myvar	7000

Assemblatore a due passate (2)

- Seconda passata:

- traduzione delle istruzioni nei rispettivi codici operativi (opcode)
- determinazione dei numeri dei registri (rs, rt, rd)
- sostituzione delle etichette** nelle istruzioni con indirizzamento relativo al PC (beq, bne) e produzione symbol table finale
- creazione della **Tabella di Rilocalizzazione**

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

$$offset = \{LC(SYMBOL) - [LC(BNE) + 4]\} / 4$$

$$\{8 - [20 + 4]\} / 4 = -4 \Rightarrow \text{bne } \$18, \$5, -4$$

Nota: il "+4" tiene conto dalla posizione del PC a tempo di esecuzione

- Per ogni etichetta irrisolta, si controlla che non sia definita come simbolo esterno (direttiva .globl)
 - se non e' un simbolo esterno, puo' essere un errore
 - se e' un simbolo esterno o un riferimento ad un indirizzo assoluto (es. nelle istruzioni jal, j, la) **deve restare in symbol table**

symbol	LC
etic1	1000
myvar	7000

Creazione della Tabella di Rilocalizzazione

<pre> ----- la \$18, myvar ----- etic1: ----- ----- ----- j etic1 ----- </pre>		LC=0 LC=24 LC=1000 LC=3000	Relocation Table		
LC=LOCAZIONE ISTRUZIONE		TIPO ISTRUZIONE	SIMBOLO DIPENDENTE		
24		la	myvar		
3000		j	etic1		

- 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  $18, $18, $0
0x0040'0004      addi $19, $19, 0
0x0040'0008 (loop:) add  $4, $18, $0
0x0040'000C      add  $19, $19, $4
0x0040'0010      add  $18, $18, $19
0x0040'0014      bne  $18, $5, -4      # gia' stabilito dall'assembler
0x0040'0018      la   $18, 0x1000'0000 # diviso in due parti da 16 bit
0x0040'001C      .....
0x0040'001C      .....
0x0040'03E8 (etic1:).....
0x0040'03EC      .....
0x0040'03EC      .....
0x0040'0BB8      j     0x010'00FA      # (26 bit) 1000FA*4=4003E8

0x1000'0000 (myvar: ).....

```

Simboli esterni

- Un simbolo e' esterno se viene esportato (es. da una libreria) verso altri moduli

```
.globl cubic
```

```
cubic:
```

```
mult $4, $4      LC=0
```

```
mflo $5
```

```
mult $4, $5
```

```
mflo $5
```

```
jr $31
```

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

Symbol Table modulo B

symbol	LC
cubic	0

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

```
.extern cubic
```

```
----
```

```
jal cubic      LC=1500
```

```
-----
```

```
-----
```

```
-----
```

`.extern <symbol>` dichiara che i riferimenti a <symbol> riguardano un oggetto esterno al modulo

Symbol Table modulo A

symbol	LC

Relocation Table
modulo A

LC=LOCAZIONE ISTRUZIONE	TIPO ISTRUZIONE	SIMBOLO DIPENDENTE
1500	jal	cubic

Nota: il simbolo "cubic" verra' risolto in fase di linking

Formato del file oggetto (.o)

HEADER
TEXT segment
DATA segment
RELOCATION info
SYMBOL table
DEBUGGING info

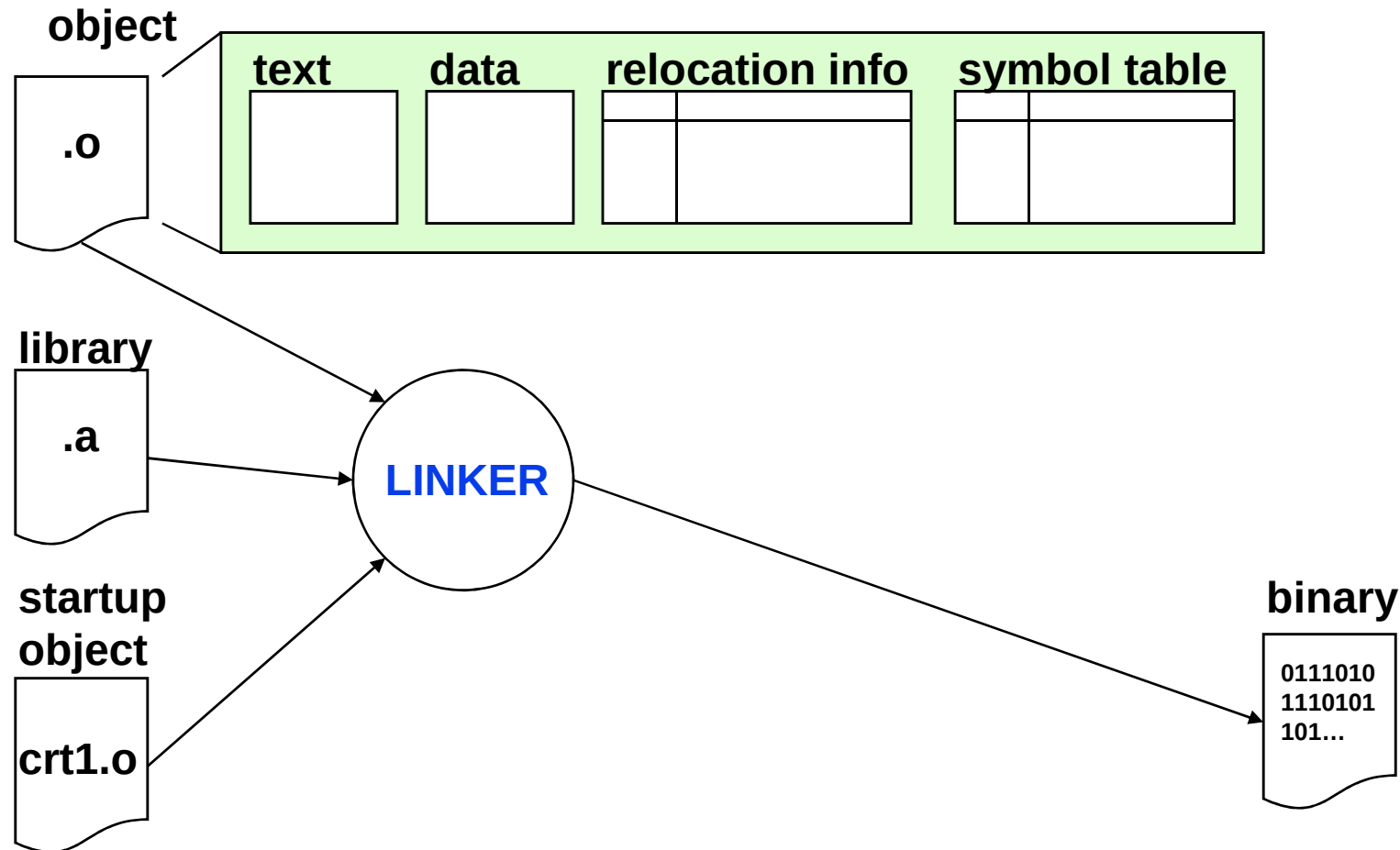
- Dimensione del file, delle parti (text, data) e CRC
- Codifica delle istruzioni (linguaggio macchina DA RILOCARE)
- Dati
- Lista delle istruzioni che fanno riferimento ad indirizzi assoluti
- Lista dei simboli locali (etichette, variabili) e etichette visibili all'esterno (assembly .globl)
- Parte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

- **UNIX:**

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

File eseguibile (binary)

- E' prodotto dal linker



I riferimenti assoluti sono rilocati e i simboli sostituiti con indirizzi

Nota: un formato molto usato per gli eseguibili e' 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:

e' possibile generare il file binario in un colpo solo con

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

in questo caso vengono automaticamente fatte tutte le scelte di default

SLIDE AGGIUNTIVE

Regole di allocazione dei registri

Num. registro	Scopo	Nome	invariato su "call"?
0	la costante 0	\$zero	n.a.
1	riservato per l'assemblatore	\$at	n.a.
2-3	risultati di funzioni ed espressioni	\$v0-\$v1	no
4-7	argomenti della funzione	\$a0-\$a3	no
8-15	temporanei (il chiamato li puo' sovrascrivere)	\$t0-\$t7	no
16-23	temporanei salvati (il chiamato DEVE salvarli**)	\$s0-\$s7	SI
24-25	ulteriori temporanei (come sopra)	\$t8-\$t9	no
26-27	riservati per il kernel	\$k0-\$k1	n.a.
28	global pointer (il chiamato DEVE salvarlo**)	\$gp	SI
29	stack pointer (il chiamato DEVE salvarlo**)	\$sp	SI
30	frame pointer (il chiamato DEVE salvarlo**)	\$fp	SI
31	return address (il chiamato DEVE salvarlo**)	\$ra	SI

** nel caso in cui tale registro sia utilizzato



Lezione 5

Valutazione delle Prestazioni

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Prestazioni - Introduzione

- Buone prestazioni hardware $\Rightarrow$ efficacia dell'intero sistema
- Difficolta' a misurare le prestazioni
 - complessita' dei moderni sistemi software
 - peculiarita' hardware non sempre ideali in qualsiasi situazione
- Non basta il manuale del Set di Istruzioni e un insieme significativo di applicazioni
 - le metriche possono variare a seconda del tipo di applicazione
- Le prestazioni sono un parametro decisionale per l'acquisto di un calcolatore
 - i produttori e i venditori hanno interesse a mettere in luce gli aspetti piu' convenienti per loro
- Quali elementi di un calcolatore influiscono sulla metrica di interesse
 - CPU, Memoria, I/O
- **Nostro obiettivo: capire le implicazioni su costi e prestazioni di determinate scelte architetturali**

Definizioni

- Le prestazioni sono in unita' di "oggetti-per-secondo"
 - "grande" e' 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 e' S_{xy} volte piu' veloce di Y"
 - In percentuali, $s_{xy} = (S_{xy} - 1) * 100$ dice "X e' $s_{xy}\%$ piu' veloce di Y"
- Esempio:
 - $E_x = 1\text{sec}, E_y = 1.2\text{sec} \Rightarrow S_{xy} = 1.2 \Rightarrow X \text{ e' } 1.2 \text{ volte piu' veloce di Y}$
 $s_{xy} = 20\% \Rightarrow X \text{ e' il } 20\% \text{ piu' veloce di Y}$

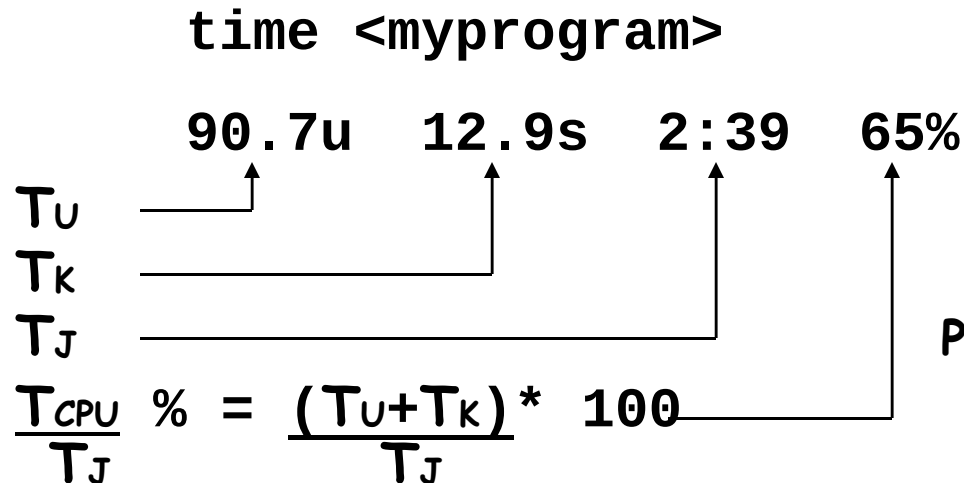
Tempo Totale e Tempo di CPU

- 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 (puo' includere il tempo "dell'uomo")
 - senza tempo per eseguire altri programmi
 - Puo' essere suddiviso in
 - T_U = tempo di CPU relativo all'utente
 - T_K = tempo di CPU relativo al Sistema Operativo (K=Kernel)

$$T_U + T_K = T_{CPU}$$

Tempo di CPU misurato in UNIX/LINUX

- Dato il programma myprogram



Per il 35 % la macchina e' usata

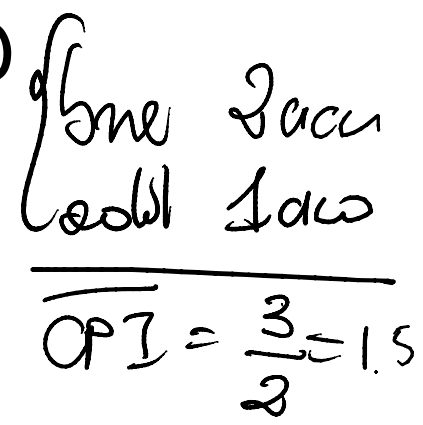
- T_K spesso non viene incluso
 - perche' puo' nascondere inaccuratezze di misura
 - il calcolo puo' risultare sbilanciato se si usano sistemi operativi diversi
 - su alcune macchine il codice puo' essere considerato di kernel e su altre di utente
 - d'altra parte nessun programma puo' essere eseguito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

• Prestazioni di Sistema $\Rightarrow T_J$

Prestazioni CPU $\Rightarrow T_U$

Equazione delle prestazioni

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

- T_C e' il periodo di clock (f_C e' la frequenza di clock)
 - C_{CPU} sono i cicli di clock
 - N_{CPU} e' il Numero di istruzioni eseguite **DINAMICAMENTE**
 - $\overline{CPI}$ e' il numero di Cicli Per Istruzione (medio)
- 

Handwritten calculation showing the average CPI (Cicli Per Istruzione) as the sum of 3 and 1, divided by 2, resulting in 1.5.

$$\overline{CPI} = \frac{3 + 1}{2} = 1.5$$

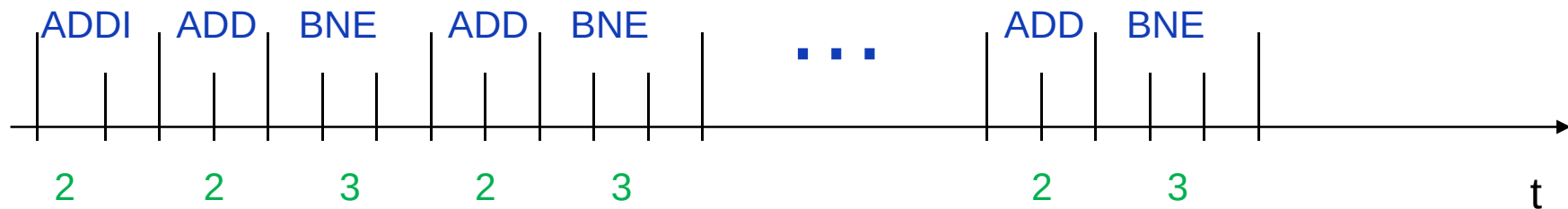
Esempio

Assumiamo $\$18=0$, $\$19=0$:

addi $\$19$, $\$19$, 100

loop: add $\$18$, $\$18$, 1

bne $\$18$, $\$19$, loop



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

/ $CPI= 502/201 \approx 2.5$

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

Esempio 1

- $T_{CPU}(A) = 10s$ $f_c(A) = 400 \text{ MHz}$
- $T_{CPU}(B) = 6s$ $f_c(B) = ?$

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:

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

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

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

Esempio 2

- $f_c(A) = 1 \text{ GHz}$ $f_c(B) = 500 \text{ MHz}$
 - $/CPI(A) = 2.5$ $/CPI(B) = 1.2$
 - Quale calcolatore e' piu' 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 $\rightarrow N_{CPU}(A) = N_{CPU}(B)$
- $S_{AB} = 0.9 \rightarrow B$ e' 1.1 volte piu' veloce di A

Come misurare i parametri fondamentali dell'equazione

- T_{CPU} → orologio
- f_c → dato di targa
- N_{CPU} , $/CPI$ → difficili da misurare

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

Motivi delle dipendenze dei parametri fondamentali

- Programma

- Fibonacci-ricorsivo
vs. Fibonacci-iterativo $\rightarrow N_{CPU}$

- Compilatore

- Opzioni -O1, -O2, -O3 $\rightarrow N_{CPU}, /CPI$

- Sistema Operativo

- Una syscall puo' essere implementata diversamente in Linux rispetto a Windows $\rightarrow N_{CPU}$

- Instruction Set (ISA)

- MIPS vs. x86 vs. ARM, #cicli necessari per es. add $\rightarrow N_{CPU}, /CPI$

- Struttura CPU

- Una pipeline elabora piu' istruzioni per ciclo $\rightarrow /CPI$
- Stadi (logica combinatoria) con meno porte logiche $\rightarrow$ maggiore f_c

- Sottosistema di memoria

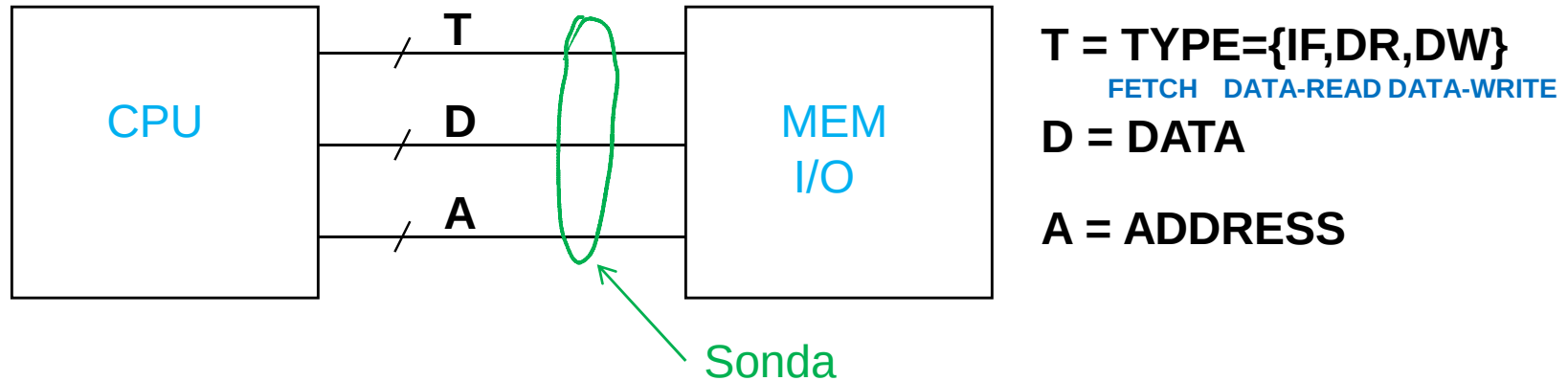
- Es. add $\rightarrow$ 1 ciclo, lw $\rightarrow$ 100 cicli

- Tecnologia (elettronica)

- Es. 14 nm invece che 90nm

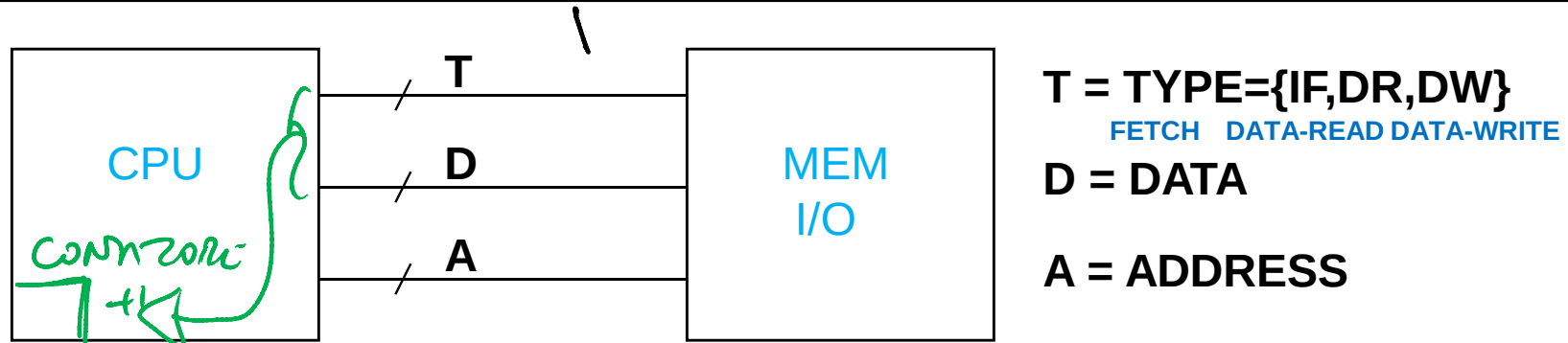
	N_{CPU}	$/CPI$	f_c
Programma	X		
Compilatore	X	(X)	
Sistema Operativo	X		
ISA	X	X	
Struttura CPU		X	X
Sottosistema di Memoria		X	
Tecnologia			X

Come valutare Ncpu (1)



- Strumentazione Hardware (sonda o probe) che legge le istruzioni che il processore preleva dalla memoria
- Richiede di poter aprire la macchina
- Richiede la disponibilita' di una opportuna sonda
- Richiede la disponibilita' di un analizzatore di stati logici

Come valutare Ncpu (2)



- 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
- Inserimento di "sonde software" (program instrumentation)
 - MIPS R4400: Pixie
- Simulatori dell'architettura (tecnica Software)
 - Es. SPIM

Come valutare Ncpu (3)

Codice Originale

```
add $1 $2 $3    #1ciclo  
lw  $1, 0($2)   #100cicli  
bne $5,$6,Loop  #3cicli
```

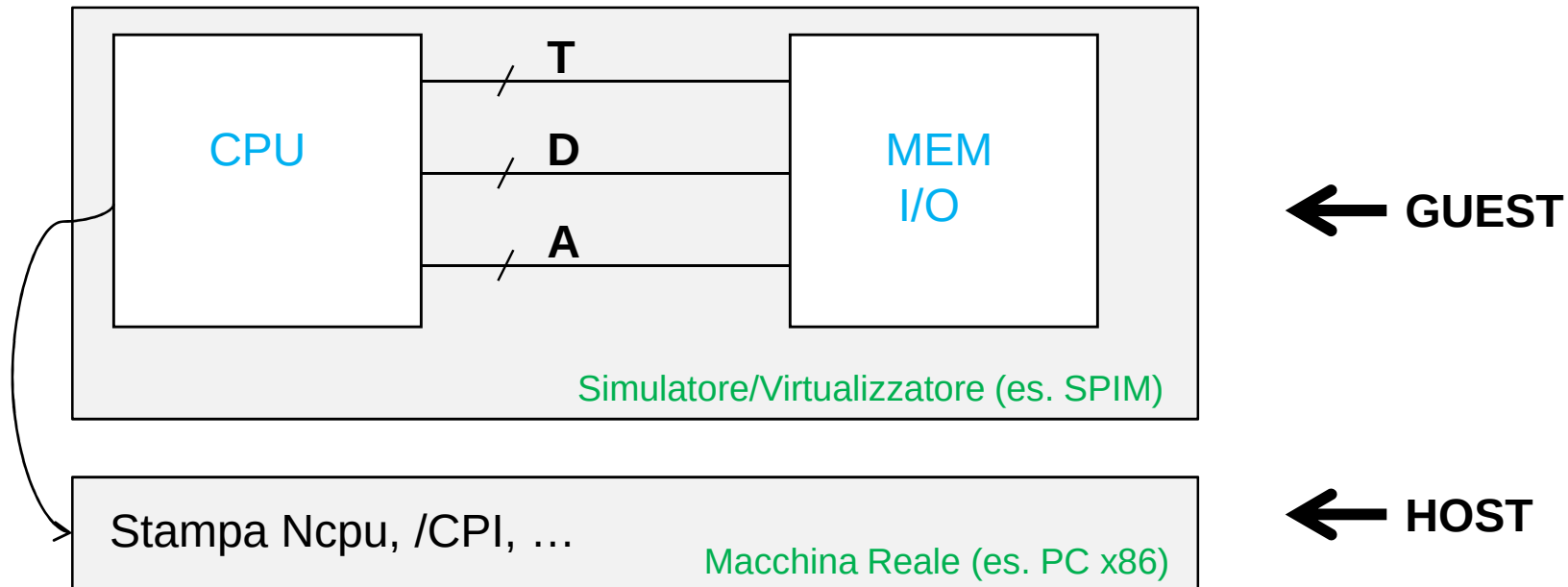
Codice Instrumentato

```
add  $25,3  
add  $26,104  
add  $1 $2 $3  
lw   $1, 0($2)  
bne  $5,$6,Loop
```

- 1) Riservo un registro (es. **\$25**) per contare le istruzioni
 - 2) Riservo un registro (es. **\$26**) 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, **\$25** contiene Ncpu mentre **\$26/\$25** e' 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)



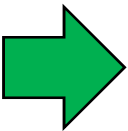
- Usare dei simulatori/virtualizzatori/emulatori
 - SPIM → simulatore di processore MIPS
 - QEMU → emulatore molto diffuso sotto Linux → virtual machine
 - Virtualbox, Vmware, VirtualPC → ulteriori virtual machines

Come calcolare /CPI

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

$$\overline{CPI} = \sum_{K=1}^T \overline{CPI}_K = \sum_{K=1}^T p_{CPU,K} \cdot CPI_K = \frac{1}{N_{CPU}} \cdot \sum_{K=1}^T N_{CPU,K} \cdot CPI_K$$

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} $CPI_K : /CPI$
Aritmethic	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%
  $/CPI = 0.5+0.4+0.2+0.4 = 1.5$				

Esempio: scelta di compilatori per una macchina

- Per le istruzioni di tipo A, B, C: $/CPI_A=1$ $/CPI_B=2$ $/CPI_C=3$

COMPILATORE	Istr. Tipo A	Istr. Tipo B	Istr. Tipo C
CC1	2	1	2
CC2	4	1	1

- Quale compilatore produce il minor numero di istruzioni?

- $N_{CPU}(1) = 5$ $N_{CPU}(2) = 6$
- il compilatore 1 produce il codice piu' corto

- Quale compilatore produce il codice piu' veloce?

- $C_{CPU}(1) = 2*1 + 1*2 + 2*3 = 10$ cicli
- $C_{CPU}(2) = 4*1 + 1*2 + 1*3 = 9$ cicli
- il compilatore 2 produce, pero', il codice piu' veloce!

- Qual e' il $/CPI$ di ciascuno dei due compilatori?

- $/CPI(1) = C_{CPU}(1)/N_{CPU}(1) = 10/5 = 2$
- $/CPI(2) = C_{CPU}(1)/N_{CPU}(1) = 9/6 = 1.5$
- il compilatore 2 produce il codice con il miglior $/CPI$

Scalabilita'

$$P \propto 1/T_{\text{CPU}} = \frac{1}{N_{\text{CPU}} \cdot \overline{CPI}} \cdot f_c$$

- Ad un raddoppio di f_c ci aspettiamo un raddoppio di P , ma cio' non avviene... (cfr. SPEC Pentium)
 - Il sottosistema di memoria 'non tiene il passo del processore'
- La capacita' di fornire maggiori prestazioni al variare di un parametro viene chiamata *scalabilita' rispetto a quel parametro*
 - Il processore PentiumPro risulta *piu' scalabile* del processore Pentium *con la frequenza*
 - In particolare:
$$P_{200}/P_{100} = f_{200}/f_{100} * CPI_{100}/CPI_{200} \quad \rightarrow \quad CPI_{200} \neq 2 * CPI_{100}$$

Legge di Amdhal

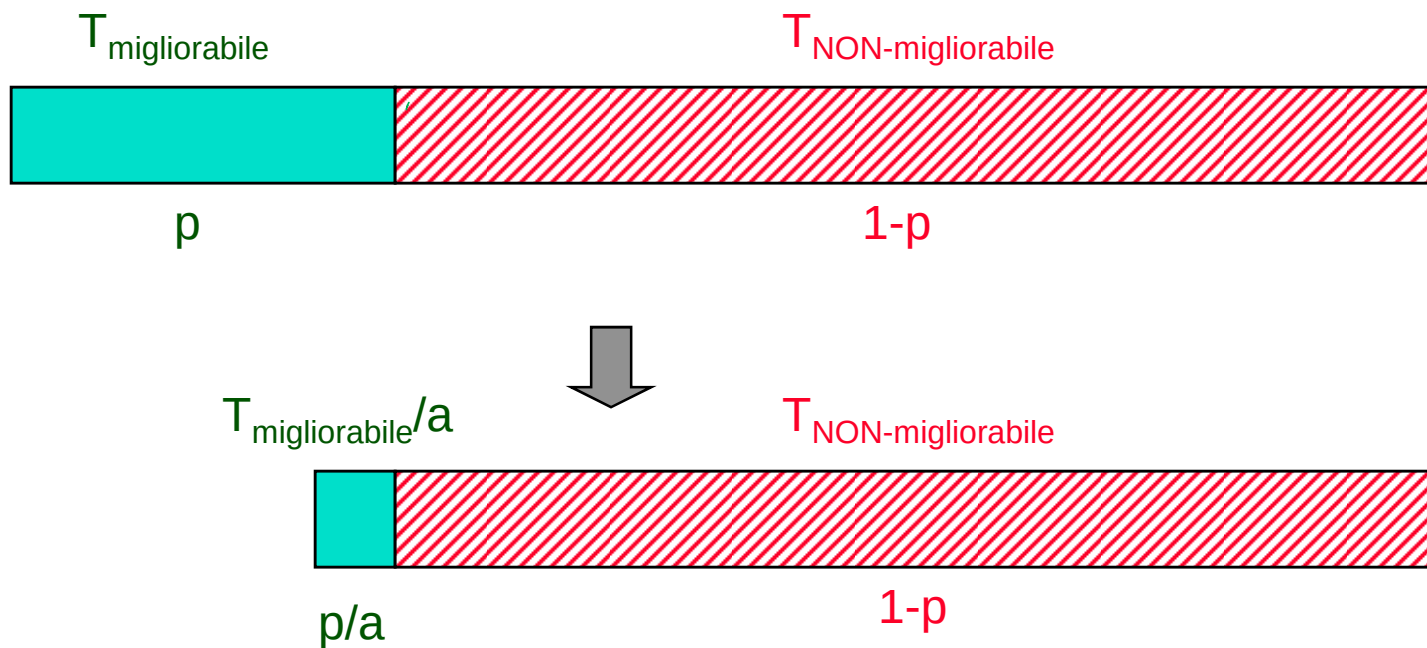
- In generale puo' essere difficile stabilire l'esatta dipendenza da un dato parametro (anche quando questo compare esplicitamente nell'equazione!)
- Per questo motivo viene utilizzata la seguente formula, per mettere in relazione lo speedup di una parte del sistema con lo speedup dell'intero sistema:

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

- Dove $p = \% \text{ del tempo 'migliorabile'} = T_{migliorabile} / T_{prima}$
 $a = \text{speed-up della parte di sistema migliorabile}$

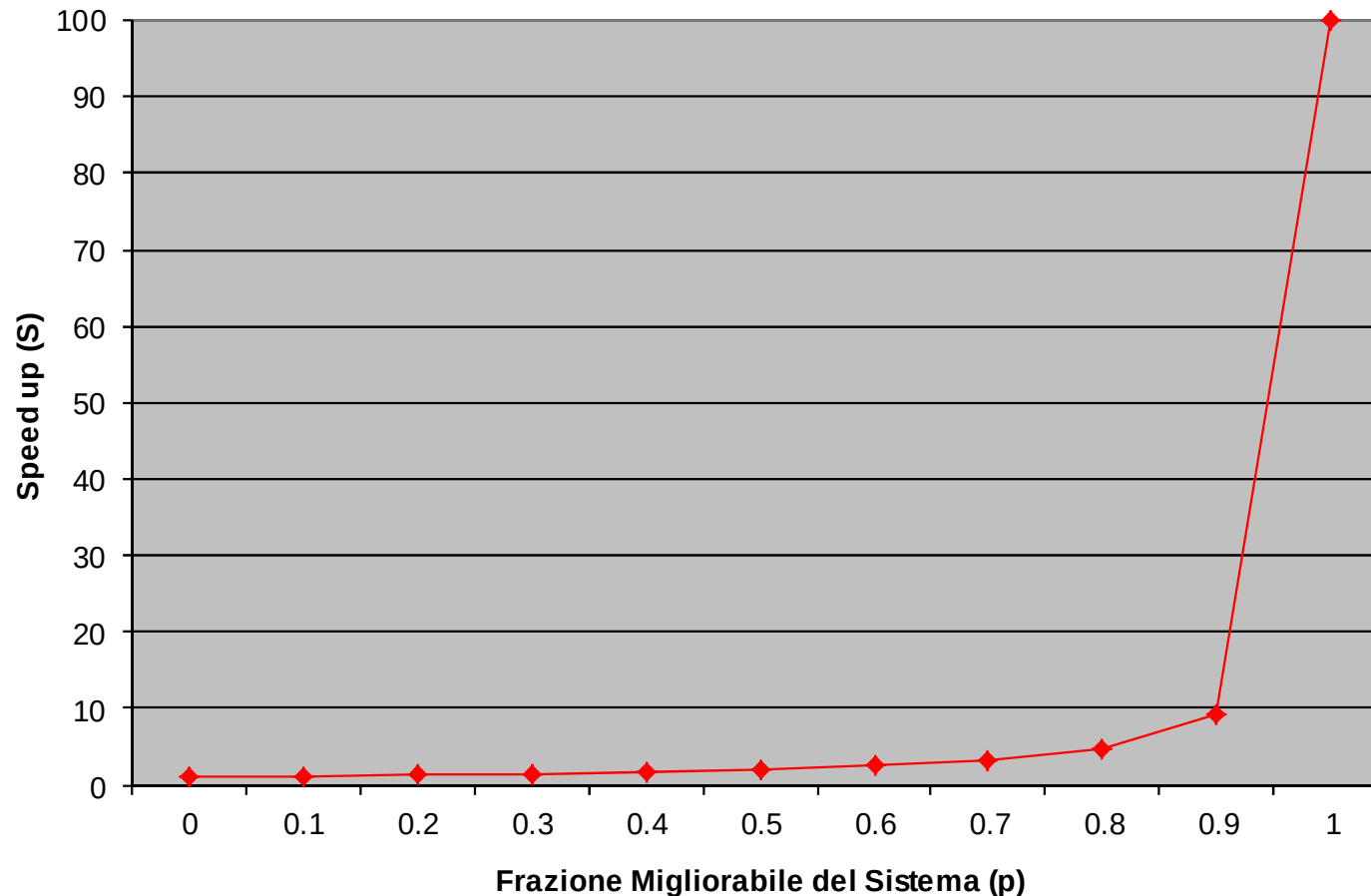
Legge di Amdhal (2)

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



$p = \% \text{ del tempo 'migliorabile'} = T_{\text{migliorabile}} / T_{\text{prima}}$
 $a = \text{speed-up della parte di sistema migliorabile}$

Legge di Amdhal (3)

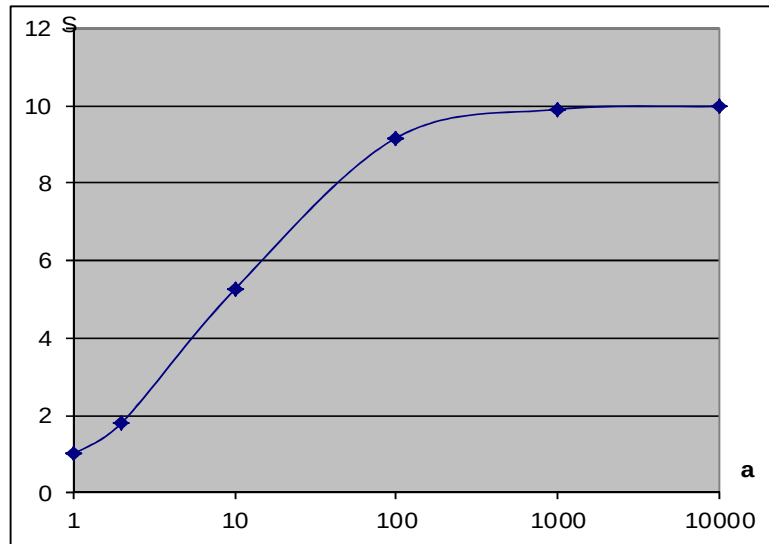


- Ipotesi: la frazione migliorabile ha uno speed-up (α) pari a 100
→ E' importante migliorare una parte che e' consistente (>0.8)

Esempio

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

a	S	p
1	1.000	0.90
2	1.818	0.90
10	5.263	0.90
100	9.174	0.90
1000	9.911	0.90
10000	9.991	0.90



Per $a \rightarrow \infty$ il massimo speed-up ottenibile
è pari a $1/0.1=10$

→ Invece di aggiungere 9999 processori conviene
parallelizzare anche una parte del restante 10%

Conclusioni dalla legge di Amdhal

- 1) E' molto importante migliorare la parte piu' utilizzata del sistema!

- 2) A un certo punto e' importante anche migliorare la parte meno utilizzata del sistema...

Definizioni

- **WORKLOAD**
insieme di programmi
abitualmente usati su un dato calcolatore
- **BENCHMARK**
programma campione che gli utenti sperano abbia
un comportamento simile al proprio workload (e.g. SPEC2000)
- **SMALL-KERNEL**
programma "giocattolo" usato per le fasi di testing
di prototipi (e.g. Lawrence Livermore Loops, Linpack)
- **MICRO-BENCHMARK**
sequenze di istruzioni ritenute significative (e.g. REPS MOVE)
- I migliori benchmark sono i programmi reali
 - campo ingegneristico: programmi scientifici/ingegneristici
 - software developers: compilatori, sistemi di documentazione
- I moderni compilatori possono ottimizzare il codice
in base al tipo di programma (scientifico, office, ...)
 - nota: ottimizzazioni troppo spinte possono produrre codice
assembly **NON CORRETTO**

Tipi di Benchmark

Vantaggi

- rappresentativo

Carico effettivo
(workload)

- portabilita'
- ampiamente usati
- catturano i miglioramenti

Set di Benchmark
(SPEC2000,TPC)

- facili da seguire, usati nelle prime fasi del ciclo di progetto

Small "Kernel"
(e.g. matrix loop)

- identificare prestazioni di picco e potenziali colli di bottiglia

Micro-benchmarks
(e.g REP MOVs)

Svantaggi

- molto specifici
- non portabili
- difficili da eseguire, o da misurare
- difficile identificare cause
- meno rappresentativo
- facili da fuorviare
- il "picco" puo' essere distante dalle prestazioni delle reali appliacazioni

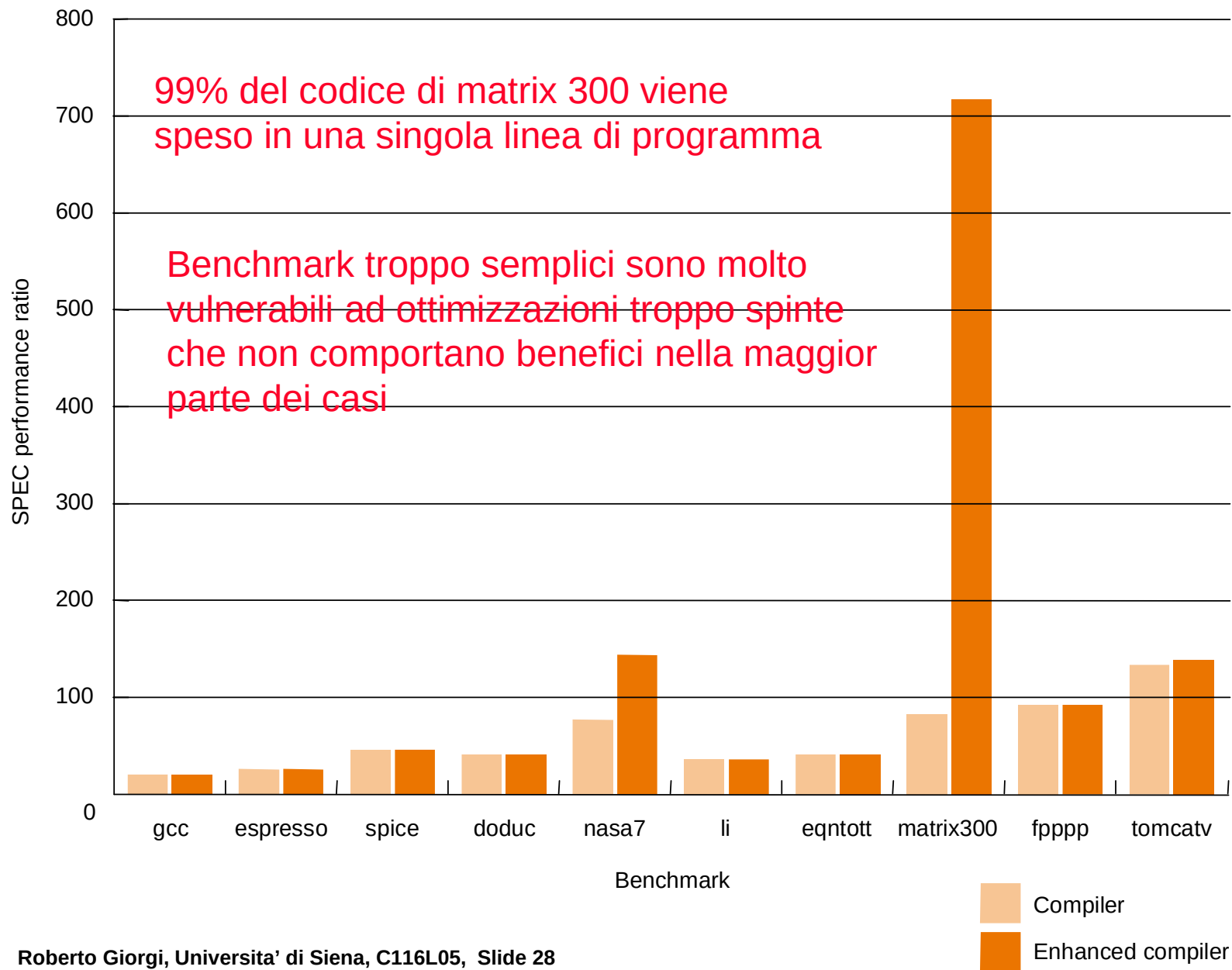
Benchmark Suites

- Insieme di programmi usati per valutare un computer
- Sviluppati e gestiti da organizzazioni non-profit (e.g. Standard Performance Evaluation Corp=SPEC, PC Magazine, ...) → www.spec.org www.tpc.org
- Mettono sotto test ben definiti aspetti del sistema:
 - performance delle applicazioni da ufficio
 - performance delle applicazioni database
 - prestazioni CPU+memoria
 - Performance grafiche
- Esempi
 - SPEC95, SPEC2000, SPEC2006
 - TPC-A, TPC-B (carichi transazionali puri)
 - TPC-C (On-Line Transaction Processing o OLTP = Database + WEB server) → TPC-H, TPC-S, TPC-E
 - TPC-D (Decision Support Systems o DSS)
 - TPC-W (Applicazioni per commercio elettronico e Internet)

SPEC Benchmarks

- **Prima versione: SPEC89**
 - 10 programmi forniscono un singolo numero (SPECmark)
 - Macchina di riferimento: VAX 11/780
- **Seconda versione: SPEC92**
 - SPECint92 (6 programmi con operazioni su interi)
 - SPECfp92 (14 programmi con operazioni su floating point)
 - Le opzioni del compilatore NON possono essere qualsiasi
 - spice:
unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)=memcpy(b,a,c)")
 - wave5:
/ali=(all,dcom=nat) /ag=a /ur=4 /ur=200
 - nasa7:
/norecu /ag=a /ur=4 /ur2=200 /lc=blas
- **Terza versione: SPEC95**
 - Macchina di riferimento SPARCStation 10/40
- **Quarta versione: SPEC2000**
- **Quinta versione: SPEC2006**

Risultati SPEC89: IBM PowerStation550, 2 compilatori



SPEC 95

- 18 applicazioni (benchmark) con i relativi dati di ingresso che riflettono uno workload di carattere tecnico
- Otto benchmark con prevalenti operazioni su interi
 - go, m88ksim, gcc, compress, li, jpeg, perl, vortex
- Dieci benchmark con prevalenti operazioni su floating point
 - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
- Devono essere eseguiti con le opzioni standard dei compilatori

Caratteristiche da definire

- Codice del benchmark
- Dati di ingresso
- Dati attesi in uscita
- Opzioni del compilatore
- Caratteristiche della macchina
- Tipo di sistema operativo

Esempio caratteristiche prova

- **Hardware**

- numero del modello Powerstation550
- CPU 41.67MHz POWER 614
- FPU Integrata
- Numero di CPU 1
- Dimensione Cache 64Kbyte dati, 8Kbyte istruzioni
- Memoria 64MB
- Sottosistema dischi 2.4 Gbyte SCSI

- **Software**

- tipo e versione del Sistema Operativo AIX 3.1.5
- versione compilatore AIX XL/C 1.1.5, AIX XL Fortran 2.2
- Altro software Nessuno
- Tipo di filesystem AIX
- Livello di firmware -

- **Sistema**

- Parametri Sis. Op. Nessuno
- carico background Nessuno
- Stato del sistema Multiuser

Ricapitolazione

- **Metriche a tempo di progetto del sistema**
 - Il sistema puo' essere implementato, in quanto tempo e a che costo?
 - Il sistema puo' essere programmato, facilita' di compilazione
- **Metriche statiche**
 - Quanti byte occupa il programma?
- **Metriche dinamiche**
 - Quante istruzioni vengono eseguite?
 - Quanti byte il processore preleva per eseguire il programma?
 - Quanti cicli servono mediamente per una istruzione?
 - Quanto e' efficace "tirare" sul clock?
- **Miglior metrica:**
 - Tempo di esecuzione del mio programma!

Esercitazione 2

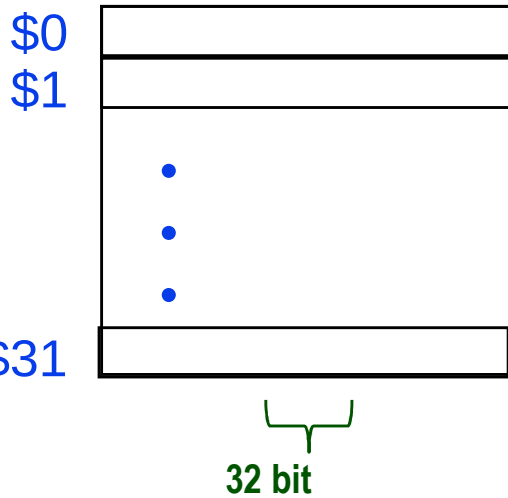
Assembly MIPS (1)

<http://www.dii.unisi.it/~giorgi/didattica/arcad1>

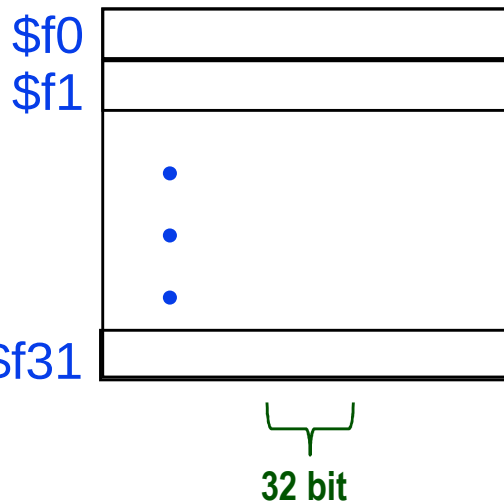
All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHES, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Instruction Set Architecture (es. MIPS 3000)

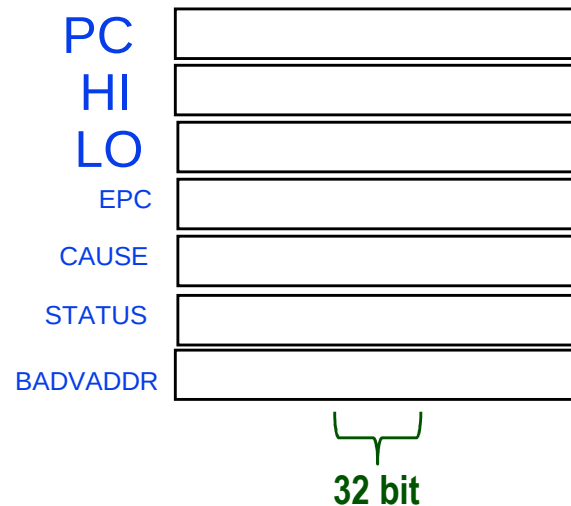
Registri INT



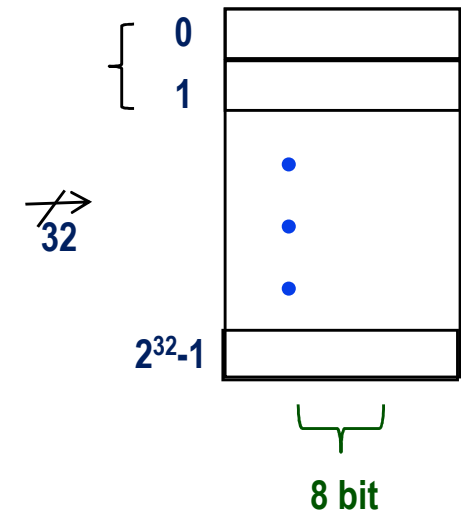
Registri FP



Registri Speciali



Memoria e I/O



- **Categorie di Istruzioni**

- Load/Store
- Calcolo (per Interi e per Frazionari)
- Jump/Branch
- Istruzioni speciali

- **Formato delle Istruzioni:**

- lunghezza FISSA a 32 bit → CPU a 32 bit

Il Set di Istruzioni MIPS

Istruzioni Aritmetiche MIPS (1)

- Tutte le istruzioni aritmetiche hanno 3 operandi
- L'ordine degli operandi e' fisso
(il primo e' sempre l'operando destinazione)

Es:

C code: $A = B + C$

MIPS code: `add $s0, $s1, $s2`

Istruzioni Aritmetiche MIPS (2)

- Principio di progetto:
“la semplicità favorisce la regolarità”.
- Naturalmente questo complica qualcosa...

C code: $A = B + C + D;$
 $E = F - A;$

→ $\$s0 \equiv A, \$s1 \equiv B, \$s2 \equiv C, \$s3 \equiv D, \$s4 \equiv E, \$s5 \equiv F$

MIPS code: $\text{add } \$t0, \$s1, \$s2$
 $\text{add } \$s0, \$t0, \$s3$
 $\text{sub } \$s4, \$s5, \$s0$

- **Gli operandi delle istruzioni aritmetiche devono SEMPRE essere registri**
- Si hanno solo 32 registri a disposizione

Registri o Memoria

- L'associazione fra variabili e registri viene fatta dal *compilatore*
- Cosa accade se il programma ha moltissime variabili o fa accesso a strutture dati complesse (es. vettori)?

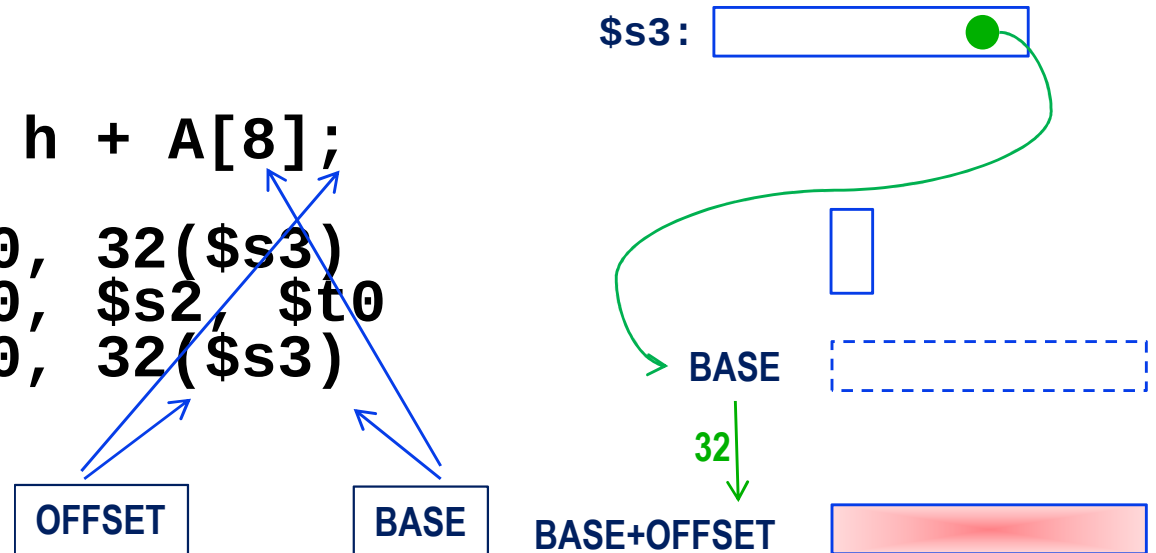
Istruzioni di accesso alla memoria

- Istruzioni Load e Store

Es.: $\$s3 \equiv A$, $\$s2 \equiv h$

C code: $A[8] = h + A[8];$

MIPS code: $\text{lw } \$t0, 32(\$s3)$
 $\text{add } \$t0, \$s2, \$t0$
 $\text{sw } \$t0, 32(\$s3)$



- NOTE:

- Il dato caricato e' **32 bit** (1 word=4 bytes)
- Nella 'sw' (al contrario delle altre istruzioni con operandi), **la destinazione (in memoria) e' l'ultimo operando**
- gli operandi di istruzioni Aritmetiche sono **SEMPRE REGISTRI**, mai celle di memoria! → **IN MIPS, NON ESISTE $\text{add } \$t0, \$t1, 32(\$s3)$**

Primo Esempio

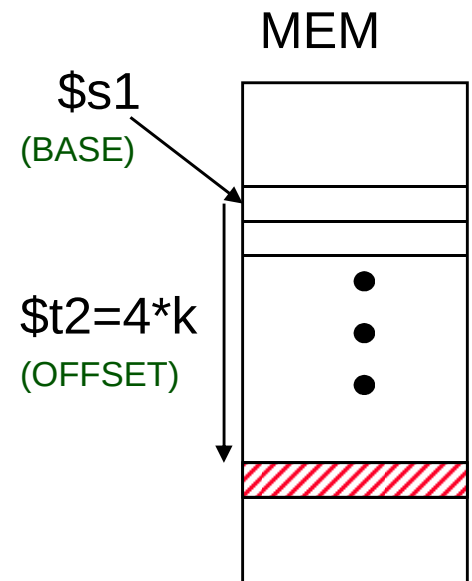
- Quali sono le istruzioni eseguite dalla macchina per il seguente frammento di codice C?

```
swap(int v[], int k)
{ int temp;
  temp = v[k]
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



$\$s0 \equiv k$
 $\$s1 \equiv v$

```
swap:
add  $t2, $s0, $s0
add  $t2, $t2, $t2
add  $t3, $s1, $t2
lw   $t4, 0($t3)
lw   $t5, 4($t3)
sw   $t5, 0($t3)
sw   $t4, 4($t3)
jr   $ra
```



- jr \$ra** conclude sempre una subroutine (vedremo poi perche'...)

Nota: v e' un vettore, quindi quando (in C) scrivo " v " intendo implicitamente "l'indirizzo base del vettore v "

Regole di allocazione dei registri

Num. registro	Scopo	Nome	invariato su "call"?
0	la costante 0	\$zero	n.a.
1	riservato per l'assemblatore	\$at	n.a.
2-3	risultati di funzioni ed espressioni	\$v0-\$v1	no
4-7	argomenti della funzione	\$a0-\$a3	no
8-15	temporanei (il chiamato li puo' sovrascrivere)	\$t0-\$t7	no
16-23	temporanei salvati (il chiamato DEVE salvarli**)	\$s0-\$s7	SI
24-25	ulteriori temporanei (come sopra)	\$t8-\$t9	no
26-27	riservati per il kernel	\$k0-\$k1	n.a.
28	global pointer (il chiamato DEVE salvarlo**)	\$gp	SI
29	stack pointer (il chiamato DEVE salvarlo**)	\$sp	SI
30	frame pointer (il chiamato DEVE salvarlo**)	\$fp	SI
31	return address (il chiamato DEVE salvarlo**)	\$ra	SI

**** nel caso in cui tale registro sia utilizzato**

Linguaggio Macchina

- Le istruzioni, come i registri e le word sono a 32 bit

Es.: add \$t0, \$s1, \$s2

i registri simbolici corrispondono ai registri numerici:

\$t0≡\$8, \$s1≡\$17, \$s2≡\$18

- Formato delle istruzioni:

000000	10001	10010	01000	00000	100000
op	rs	rt	rd	shamt	funct

- *Che cosa significano i nomi dei campi?*

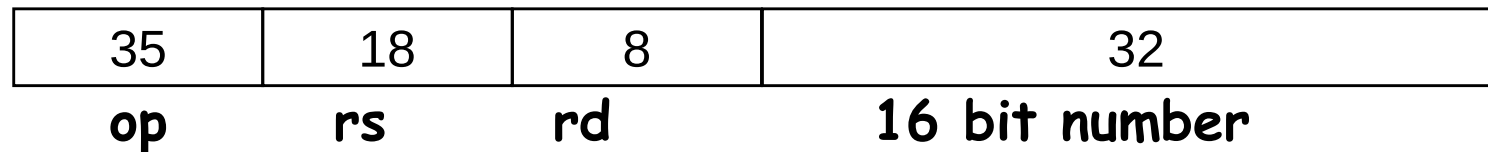
- *rs = source register*
- *rt = second source register (“t”=“s”+1)*
- *rd = destination register*
- *op = (opcode) codice operativo*
- *shamt = shift amount*
- *funct = codice funzione (estende op)*

QUESTO FORMATO E' CHIAMATO “R”

Linguaggio Macchina

- Consideriamo le istruzioni load-word e store-word
 - Cosa dovremmo fare in base al “principio di regolarità”?
 - Nuovo principio: “un progetto ottimo richiede compromessi”
- Introduciamo un nuovo tipo di formato per le istruzioni
 - i registri simbolici corrispondono ai registri: \$t0≡\$8, \$s2≡\$18

Es.: lw \$t0, 32(\$s2)

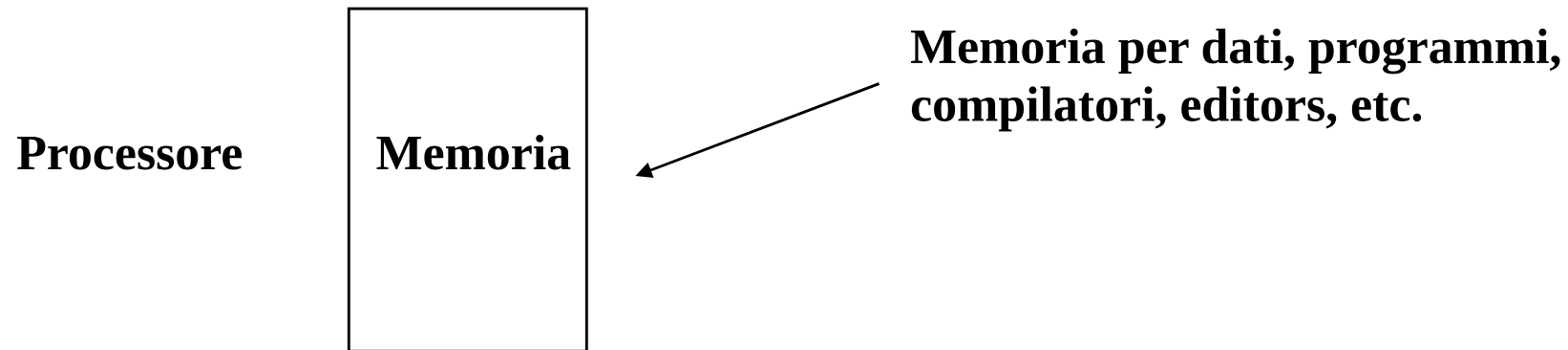


QUESTO FORMATO E' CHIAMATO “I”

Che tipo di compromesso abbiamo fatto?

Caricamento del programma e sua esecuzione

- Anche le istruzioni sono sequenze di bit
- I programmi sono caricati in memoria
 - devono essere letti così come vanno letti i dati



- Fetch & Execute Cycle
 - Le istruzioni sono prelevate (fetched) e messe in un registro interno (non visibile al programmatore)
 - I bit in tale registro "controllano" le azioni successive del processore
 - Si preleva l'istruzione successiva e si continua

Controllo del flusso del programma

- Istruzioni che regolano le decisioni
 - alterano il flusso principale del programma
 - ovvero: cambiano la “prossima istruzione” da eseguire
- Istruzioni di salto condizionale del MIPS

```
bne $t0, $t1, Label  
beq $t0, $t1, Label
```

Es.: if (b==c) a = b + c;



$\$s3 \equiv a$, $\$s4 \equiv b$, $\$s5 \equiv c$

```
bne $s4, $s5, Label  
add $s3, $s4, $s5
```

Label:

Controllo del flusso del programma

- Istruzioni di salto incondizionato del MIPS

j label

Utile per implementare ad es.: $\$s3 \Leftarrow a$, $\$s4 \Leftarrow b$, $\$s5 \Leftarrow c$

if (b!=c)		beq \$s4, \$s5, Lab1
a=b+c;		add \$s3, \$s4, \$s5
else	⇒	j Lab2
a=b-c;		Lab1: sub \$s3, \$s4, \$s5
		Lab2: ...

PER SUPPORTARE UN INDIRIZZO SUFFICIENTEMENTE AMPIO PER “LABEL”
VIENE INTRODOTTO UN ULTERIORE FORMATO DI ISTRUZIONE DETTO “J”

op	26 bit address

- Come si puo' costruire un ciclo “for”, un ciclo “while” e uno “do..while”

for(k=0;k<100;++k) {...} add \$s0,\$zero,\$zero ini_for: slt \$t0,\$s0,\$s1 beq \$t0,\$zero,fin_for add \$s0,\$s0,\$s2 j ini_for fin_for:	while(k<100){...} ini_wh: slt \$t0,\$s0,\$s1 beq \$t0,\$zero,fin_wh j ini_wh fin_wh:	do {...} while(k<100); ini_do: slt \$t0,\$s0,\$s1 bne \$t0,\$zero,ini_do

Controllo del flusso di programma

- Abbiamo: `beq, bne` $\Rightarrow$ come si fa un salto su “minore”?
- Nuova istruzione (Set on Less Than):

		<code>if \$s1 < \$s2 then</code>
		<code> \$t0 = 1</code>
<code>slt \$t0, \$s1, \$s2</code>	$\Leftrightarrow$	<code>else</code>
		<code> \$t0 = 0</code>

- Possiamo usare questa istruzione insieme a
`beq $t0, $zero, Label`
 - possiamo così costruire strutture di controllo con condizioni di salto generiche

Ricapitolazione:

• Istruzione

add \$s1,\$s2,\$s3

sub \$s1,\$s2,\$s3

slt \$t0,\$s1,\$s2

lw \$s1,100(\$s2)

sw \$s1,100(\$s2)

bne \$s4,\$s5,L

beq \$s4,\$s5,L

j Label

Significato

$\$s1 = \$s2 + \$s3$

$\$s1 = \$s2 - \$s3$

$\$t0 = (\$s1 < \$s2) ? 1 : 0$

$\$s1 = \text{Memory}[\$s2+100]$

$\text{Memory}[\$s2+100] = \$s1$

Next inst. is at Label if $\$s4 \neq \$s5$

Next inst. is at Label if $\$s4 == \$s5$

Next inst. is at Label

• Formati:

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

Linguaggio Assembly e Linguaggio Macchina

- Il linguaggio Assembly fornisce una conveniente rappresentazione simbolica
 - e' piu' comprensibile (a noi) che sequenze di numeri (binari)
- Il **linguaggio macchina** e' la "realta' sottostante"
 - es. La destinazione non e' il primo operando...
- Il linguaggio Assembly fornisce "**pseudoistruzioni**"
 - es., "move \$t0, \$t1" esiste solo in Assembly
 - l'assemblatore traduce usando "add \$t0,\$t1,\$zero"
- Quando le prestazioni diventano un fattore critico e' necessario ricordare quali sono le istruzioni reali

SIMULATORE SPIM



Lezione 7

Standard IEEE-754

per le operazioni floating-point

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

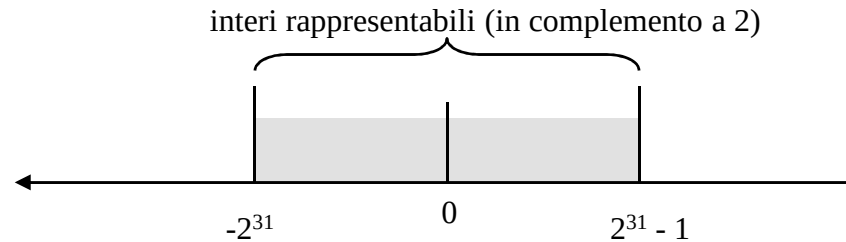
Floating-Point (breve schema)

- Nelle operazioni floating point dobbiamo rappresentare
 - numero con decimali, e.g., 3.1416
 - numeri molto piccoli, e.g., .0000000001
 - numeri molto grandi, e.g., 3.15576×10^9
- Rappresentazione:
 - segno, mantissa, esponente: $(-1)^{\text{segno}} \times \text{mantissa} \times 2^{\text{esponente}}$
 - piu' bit di mantissa danno piu' accuratezza
 - piu' bit di esponente danno un intervallo piu' ampio
- IEEE 754 floating point standard:
 - singola precisione ("float" del C) → uso 32 bit per memorizzare:
1 bit segno, **8 bit esponente**, **23 bit mantissa****
 - doppia precisione ("double" del C) → uso 64 bit per memorizzare:
1 bit segno, **11 bit esponente**, **52 bit mantissa****

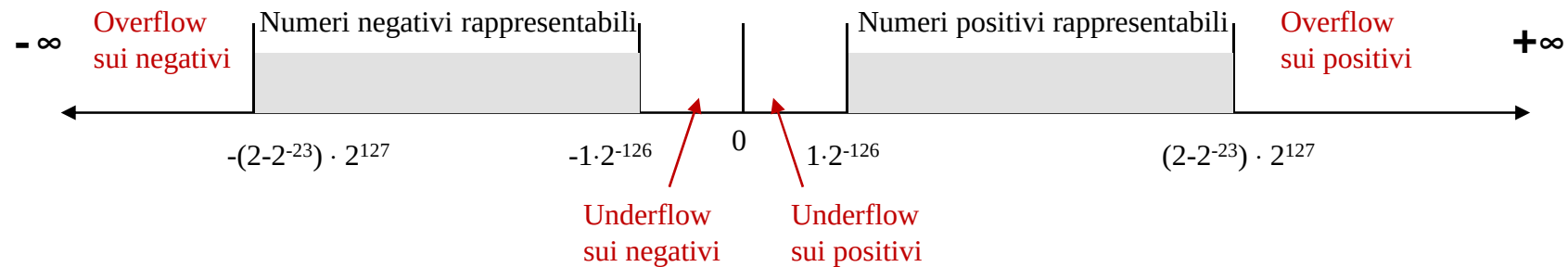
**Nota: utilizzando il "trucco" della normalizzazione della mantissa nella forma 1.xxxxx le cifre (binarie) effettivamente rappresentate sono rispettivamente 24 e 53

Intervalli rappresentabili

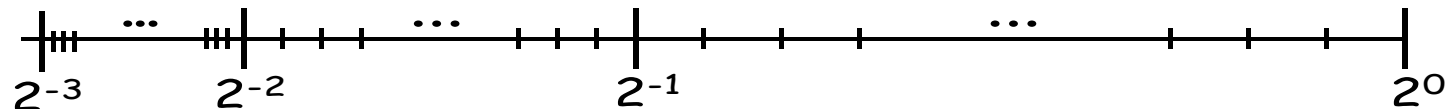
- Nel caso di interi a 32 bit:



- Nel caso di floating-point a 32 bit (IEEE-754)



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



IEEE 754 floating-point standard

• Mantissa

- Rappresentazione in base due con Normalizzazione (e modifica dell'esponente) in modo tale da riportare un 1 nella cifra piu' 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
 - $M = 1 - \text{mantissa} = m_1 \times 2^{-1} + m_2 \times 2^{-2} + \dots + m_{23} \times 2^{-23}$

m_1	m_2		m_{23}

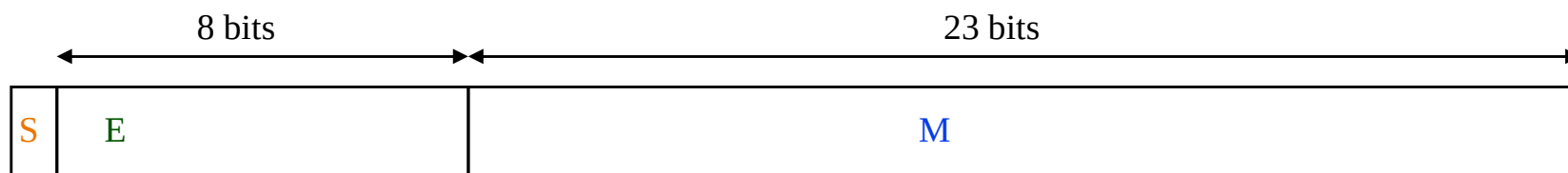
• Esponente

- Rappresentazione "polarizzata" (biased) per rendere piu' facile il sort
 - polarizzazione=127 per la singola precisione, 1023 per la doppia precisione
 - i valori vanno da 1 a $2^e - 2$ ($e = \# \text{bit per l'esponente}$) 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 $\neq 0$
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 $\neq 0$
codifica "underflow" (numero denormalizzato)
- Ricapitolando: per $(-1)^S \times (1 + M) \times 2^{E - \text{polarizzazione}}$ $\rightarrow$ memorizzo $\langle S, M, E \rangle$
 - $S = 0$ o 1 , $M = \text{mantissa\_normalizzata senza "1." iniziale}$, $E = \text{esponente} + \text{polarizzazione}$

Esempio

- 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
→ $E=\text{esponente}+\text{polarizzazione} = -1 + 127 = 126 \rightarrow 01111110$
- IEEE single precision: $10111111010000000000000000000000$
- Nota: la mantissa e' stata normalizzata. Per questo tutti i bit piu' significativi risultano spostati verso SINISTRA.

Floating-Point: single precision (32 bits)



Esempi con mantissa ed esponente positivi/negativi:

$+1.1010001 \cdot 2^{+10100} \rightarrow$ 0 10010011 101000100000000000000000
 $-1.1010001 \cdot 2^{+10100} \rightarrow$ 1 10010011 101000100000000000000000
 $+1.1010001 \cdot 2^{-10100} \rightarrow$ 0 01101011 101000100000000000000000
 $-1.1010001 \cdot 2^{-10100} \rightarrow$ 1 01101011 101000100000000000000000

Esempi di numeri "notevoli":

0 $\rightarrow$ 0 00000000 000000000000000000000000
 1 $\rightarrow$ 0 01111111 000000000000000000000000
 minore rappresentabile (2^{-126}) $\rightarrow$ 0 00000001 000000000000000000000000
 maggiore rappres. ($(2-2^{-23}) \cdot 2^{127}$) $\rightarrow$ 0 11111110 111111111111111111111111
 $\infty \rightarrow$ 0 11111111 000000000000000000000000
 NaN $\rightarrow$ 0 11111111 XXXXXXXXXXXXXXXXXXXXXXXX

Tabella di riepilogo codifica FP nei processori INTEL

Da: Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture, Sep. 2008.

URL: <http://download.intel.com/design/processor/manuals/253665.pdf>

Class		Sign	Biased Exponent	Significand	
				Integer	Fraction
Positive	+ ∞	0	11..11	1	00..00
	+ Normals	0	11..10	1	11..11
		.	.	.	.
		.	.	.	.
		0	00..01	1	00..00
	+ Denormals	0	00..00	0	11..11
Negative		.	.	.	.
		.	.	.	.
		0	00..00	0	00..01
	+ Zero	0	00..00	0	00..00
	- Zero	1	00..00	0	00..00
	- Denormals	1	00..00	0	00..01
NaNs		.	.	.	.
		.	.	.	.
		1	00..00	0	11..11
	- Normals	1	00..01	1	00..00
		.	.	.	.
		.	.	.	.
NaNs		1	11..10	1	11..11
	- ∞	1	11..11	1	00..00
	SNaN	X	11..11	1	0X..XX ²
NaNs	QNaN	X	11..11	1	1X..XX
	QNaN	1	11..11	1	10..00
	Floating-Point Indefinite				

Single-Precision

← 8 Bits →

← 23 Bits →

Double-Precision

← 11 Bits →

← 52 Bits →

Double Extended-Precision

← 15 Bits →

← 63 Bits →

Note:

- (1) il bit della parte intera e' implicito e non viene quindi memorizzato
- (2) La parte frazionaria di SNaN deve essere diversa da zero con il bit piu' significativo pari a 0

SNaN= Signaling NaN (solleva eccezione)

QNaN=Quiet NaN (non solleva eccezione)

QNaN-FP= "INDETERMINATO" → nessuno dei due operandi era un NaN: lo ha generato l'operazione

I numeri denormalizzati possono essere generati ma non vengono memorizzati

E' prevista una modalita' DAZ
=Denormal-Are-Zero

E' supportata una modalita' a 80 bit (non IEEE-754)

Complessita' del Floating Point

- Le operazioni sono leggermente piu' complesse (v. testo)
- Oltre alla condizione di overflow si puo' avere "underflow"
 - (codifica: esponente con bit tutti a zero) → numeri denormalizzati
- L'accuratezza puo' essere un grosso problema
 - IEEE 754 prevede 2 ulteriori bit: 1 *guard* bit e 1 *round* bit
 - 4 modalita' per effettuare l'arrotondamento
(*round to $+\infty$* , *round to $-\infty$* , *round to 0*, *round to nearest representable*)
 - altre complessita' (es. numeri denormalizzati)
- Implementare lo standard puo' essere "ingannevole"
 - Non usare lo standard puo' essere anche peggio
 - vedere il testo per la descrizione dell'80x86 e del Pentium bug!

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 e' 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 complessita' nelle macchine reali
- Lo standard e' stato rivisto e il documento pubblicato nel Giugno 2008 come IEEE 754-2008
<http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4610935&isnumber=4610934>
 - Fra le novita' il supporto per floating point a 128-bit
 - Aritmetica floating point decimale (oltre che binaria)

EPSILON di macchina (MACHEPS = ϵ_{mach})

- E' la differenza fra 1 e "1+"
(1+ $\equiv$ il piu' piccolo numero rappresentabile maggiore di 1)
 - E' anche chiamato "precisione di macchina"
 - Da' una maggiorazione dell'errore di arrotondamento

- Esempio, IEEE-754 Single-Precision

$$\begin{aligned} 1 &\equiv \text{esponente}=0 \quad \text{sottointeso } 1. \\ &\equiv (0 \ 01111111 \ 000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000)_{\text{IEEE-754-SP}} \\ &= (1.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000)_2 \\ &= (1)_{10} \end{aligned}$$

$$\begin{aligned} 1+ &\equiv \text{esponente}=0 \quad \text{sottointeso } 1. \\ &\equiv (0 \ 01111111 \ 000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001)_{\text{IEEE-754-SP}} \\ &= (1.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001)_2 \\ &\approx (1.000000119209289 \dots)_{10} \end{aligned}$$

$$\begin{aligned} \rightarrow \epsilon_{mach} &\equiv \text{esponente}=-23 \quad \text{sottointeso } 1. \\ &= (0 \ 01101000 \ 000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000)_{\text{IEEE-754-SP}} \\ &= (0.000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001)_2 \\ &= (2^{-23})_{10} \approx (0.000000119209289 \dots)_{10} \approx (1.19209289 \dots \cdot 10^{-7})_{10} \end{aligned}$$

- In generale, per un tipo floating-point rappresentato in base b con p cifre di mantissa (normalizzato con una cifra $\neq 0$ prima della virgola), ovvero con $p-1$ cifre dopo la virgola

$$\rightarrow \epsilon_{mach} = b^{-(p-1)} = b^{1-p} \quad (\text{es. nel caso IEEE-754 single-precision } p=24)$$

Precisazioni su MACHEPS

- Alcuni usano come definizione: “il piu’ piccolo numero positivo rappresentabile, tale che aggiunto ad 1 lo rende diverso da 1”
 - tale definizione e’ diversa dalla precedente perche’ il valore viene a dipendere dalla modalita’ di arrotondamento usata
- Nel caso di IEEE-754 Single-Precision:
 - Per arrotondamenti al successivo (round to even) questa e’ $2^{-24} + 2^{-47}$ (poco piu’ di meta’ rispetto alla precedente definizione)
 - Per arrotondamenti verso $+\infty$ questo e’ il piu’ piccolo numero rappresentabile, ovvero 2^{-149} (un numero denormalizzato)
 - Per arrotondamenti verso $-\infty$ questo coincide con la precedente definizione
- Il seguente programma C non determina MACHEPS bensì un numero distante non piu’ di un fattore 2 da MACHEPS, usando una ricerca lineare

Calcolo di MACHEPS

```
#include <stdio.h>

int main( int argc, char **argv )
{
    float machEps = 1.0f;

    printf( "current Epsilon, 1 + current Epsilon\n" );
    do {
        printf( "%G\t%.20f\n", machEps, (1.0f + machEps) );
        machEps /= 2.0f;
        /* Se sommando l'epsilon successivo ho 1 --> ho trovato epsilon */
    }
    while ((float)(1.0 + (machEps/2.0)) != 1.0);

    printf( "\nCalculated Machine epsilon: %G\n", machEps );
    return 0;
}
```

MACHEPS Output

current Epsilon, 1 + current Epsilon

1	2.00000000000000000000
0.5	1.50000000000000000000
0.25	1.25000000000000000000
0.125	1.12500000000000000000
0.0625	1.06250000000000000000
0.03125	1.03125000000000000000
0.015625	1.01562500000000000000
0.0078125	1.00781250000000000000
0.00390625	1.00390625000000000000
0.00195312	1.00195312500000000000
0.000976562	1.00097656250000000000
0.000488281	1.00048828125000000000
0.000244141	1.00024414062500000000
0.00012207	1.00012207031250000000
6.10352E-05	1.00006103515625000000
3.05176E-05	1.00003051757812500000
1.52588E-05	1.00001525878906250000
7.62939E-06	1.00000762939453125000
3.81470E-06	1.00000381469726562500
1.90735E-06	1.00000190734863281250
9.53674E-07	1.00000095367431640625
4.76837E-07	1.00000047683715820312
2.38419E-07	1.00000023841857910156

Calculated Machine epsilon: 1.19209E-07

Nota: in MATLAB

“eps” o “eps(‘double’)” restituisce 2^{-52}

“eps(‘single’)” restituisce 2^{-23}

MACHEPS di alcune macchine

Da: Kalbasi, K., "Can you trust your computer? [digital arithmetic]," *Potentials, IEEE* , vol.9, no.2, pp.15-18, Apr 1990.

URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=52995&isnumber=1912>

Computer	Rounding or Chopping	base	cifre mantissa	E_{min}	E_{max}	MACHEPS
CDC CYBER 170	R	2	48	-976	1'071	3.55×10^{-15}
CDC CYBER 205	C	2	47	-28'626	28'718	1.42×10^{-14}
Cray-1	C	2	48	-8'192	8'191	7.11×10^{-15}
DEC VAX (single)	R	2	24	-127	127	5.96×10^{-8}
DEC VAX (double)	R	2	56	-1'023	1'023	1.11×10^{-16}
HP-11C, 15C	R	10	10	-99	99	5.00×10^{-10}
IBM 3033 (single)	C	16	6	-64	63	9.54×10^{-7}
IBM 3033 (double)	C	16	14	-64	63	2.22×10^{-16}
IBM/PC (single)	R	2	24	-126	127	5.96×10^{-8}
IBM/PC (double)	R	2	53	-1'022	1'023	1.11×10^{-16}
PRIME 850 (single)	C	2	23	-128	127	2.38×10^{-7}
PRIME 850 (double)	C	2	47	-32'896	32'639	1.42×10^{-14}

L'epsilon di macchina dipende da diversi fattori, incluso la modalita' di arrotondamento

FORMATO IEEE-754

Esercizio

- **Compito 2/11/07 domanda 1**

Lezione 8

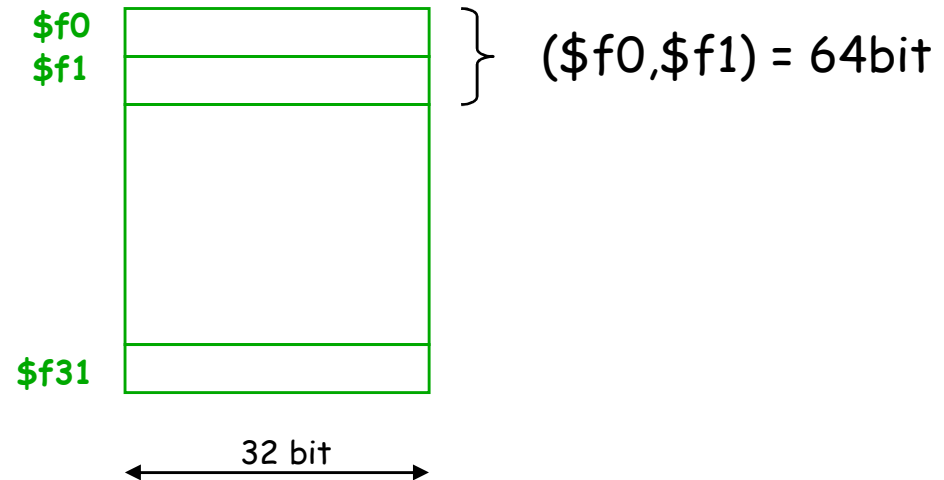
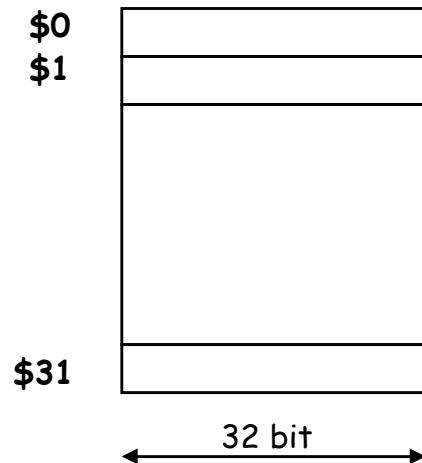
Operazioni floating-point nel processore MIPS

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

Modalita' di arrotondamento IEEE-754

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

Supporto architetturale per le op. floating-point



- `add.s $f0, $f1, $f2` lavora sui registri fp a 32 bit
- `add.d $f0, $f2, $f4` lavora su registri fp a 64 bit

Istruzioni floating point nel processore MIPS

- ARITMETICHE

add.s \$f1, \$f2, \$f3

sub.s

mul.s

div.s

abs.s \$f1

neg.s \$f1

add.d \$f2, \$f4, \$f6

sub.d

mul.d

div.d

abs.d \$f2

neg.d \$f2

- SPOSTAMENTO

mov.s \$f1, \$f2

mfc1 \$a0, \$f1

mtc1 \$f1, \$a0

mov.d \$f2, \$f4

- MEMORIA

lwc1 \$f1, 100(\$a0)

swc1 \$f1, 100(\$a0)

- SALTI

bc1t LABEL

bc1f LABEL

- CONVERSIONE

cvt.d.s \$f4, \$f1

cvt.s.d \$f1, \$f4

cvt.w.s \$f1, \$f3

cvt.s.w \$f3, \$f1

Istruzioni condizionali floating-point

- Sono nella forma

`c.xx.s $f2, $f4`

dove 'xx' puo' essere:

'lt' → less than

'le' → less than or equal

'gt' → greater than

'ge' → greater than or equal

'eq' → equal

'ne' → not equal

- Il risultato non va su un registro ma in un flag interno 'cond'
- Esempio:

`c.lt.s $f2, $f4`

`bc1t L1`

MIPS Floating-Point ABI (Application Binary Interface)

- Definisce la convenzione di uso dei registri nel software
- Valido nel caso single e in quello double precision
 - \$f0, \$f2 → valori di ritorno delle funzioni
 - \$f12,\$f14 → argomenti delle funzioni
 - \$f20,\$f22,\$f24,\$f26,\$f28,\$f30 → registri "saved"
 - \$f4,\$f6,\$f8,\$f10,\$f16,\$f18 → registri temporanei
- I registri dispari sono tipicamente usati per le load/store e move

Alcune syscall che coinvolgono i numeri floating point

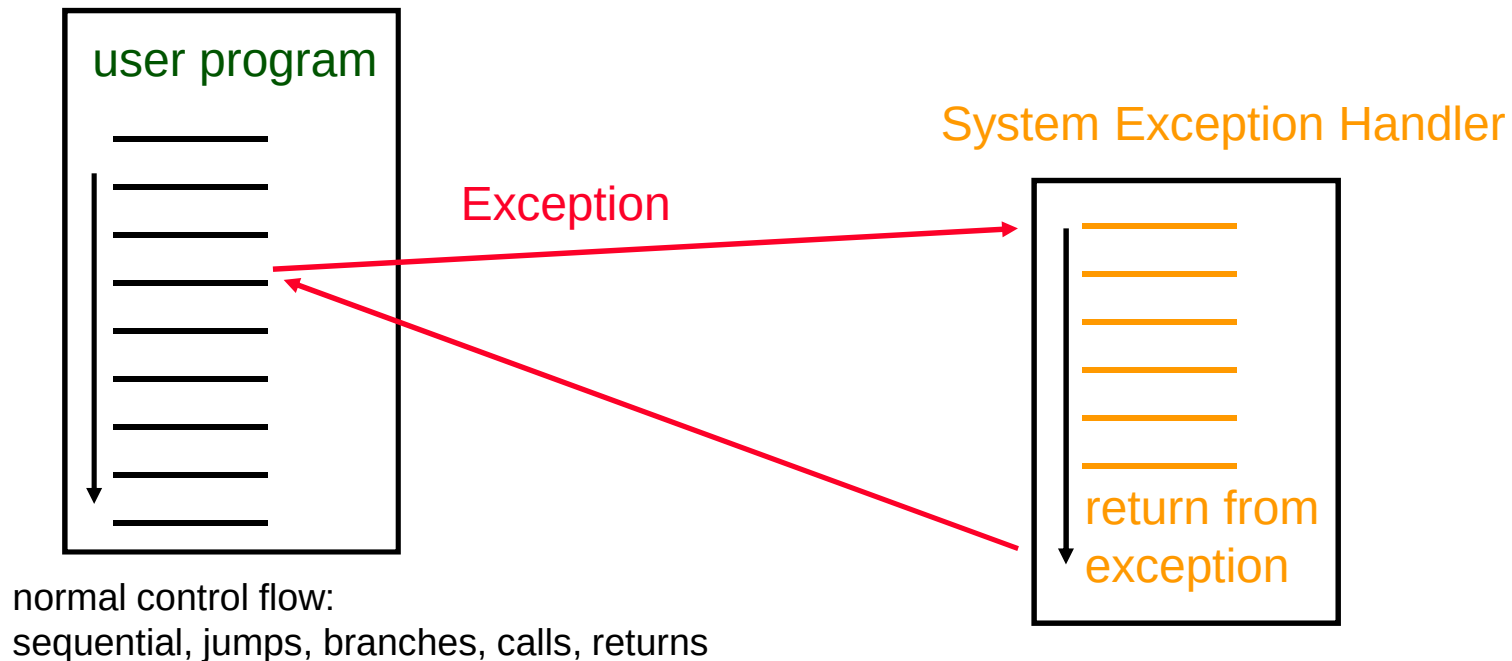
- 'syscall' serve per chiedere un servizio al sistema operativo (letteralmente 'system call')
 - PARAMETRI DI INGRESSO
 - \$v0 del contenere il numero del servizio
 - 2: stampa un single-precision float a video (\$f12 contiene il float da stampare)
 - 6: leggi un single-precision float dalla tastiera (\$f0 contiene il float letto in binario)
 - 3: stampa un double-precision float a video (\$f12(-f13) contiene il double da stampare)
 - 7: leggi un double-precision float dalla tastiera (\$f0(-f1) contiene il float letto in binario)

Lezione 9

Eccezioni e interrupt

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

Eccezioni

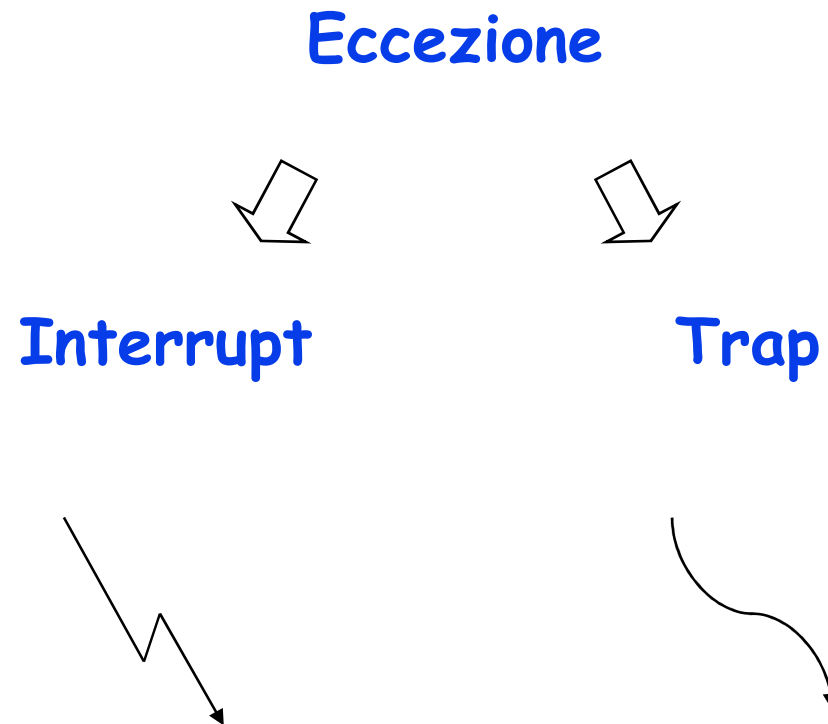


- **Eccezione** = trasferimento non-previsto del flusso di controllo
 - Il sistema gestisce l'eccezione eseguendo una routine "agganciata" ad una eccezione di quel tipo
 - E' necessario memorizzare l'indirizzo dell'istruzione che ha generato l'eccezione (es. divisione per zero)
 - Successivamente il controllo deve tornare al programma utente
 - E' necessario salvare e ripristinare lo stato del programma utente

Che ne e' dell'istruzione che era in esecuzione?

- L'architettura MIPS definisce l'istruzione che causa un'eccezione come una "istruzione senza effetto"
- Le eccezionidevono evitare assolutamente di modificare lo stato della macchina
 - Es. i contenuti dei registri \$s... NON devono essere alterati
- La gestione delle eccezioni e' un aspetto complesso del microprocessore spesso trascurato
 - Potenzialmente puo' anche limitare le prestazioni

Classificazione Generale delle Eccezioni



Classificazione delle Eccezioni (2)

- Interrupt (interruzione)
 - Eccezione causata da eventi esterni
 - Asincrona rispetto all'esecuzione del programma
 - La sua gestione "accade" fra istruzioni di un altro programma
 - Il programma in esecuzione deve essere **sospeso** e poi **riattivato**
- Trap
 - Eccezione causata da eventi interni
 - Condizioni eccezionali (overflow, divisione per zero)
 - Errori del sistema (parita')
 - Gestione della memoria virtuale (page fault)
 - Sincrona rispetto all'esecuzione del programma
 - Il gestore dell'eccezione deve rimediare alla situazione
 - L'istruzione "colpevole" puo' essere riprovata o simulata e il programma puo' o continuare o essere bloccato (abort)

Classificazione usata nel MIPS

- Ogni cambiamento inaspettato del flusso di controllo viene denominato "eccezione"
 - Non distinguo fra sorgenti esterne e interne del microprocessore

<i>Tipo di evento</i>	<i>Da dove?</i>	<i>Terminologia MIPS</i>
Richiesta di I/O da dispositivo	Esterno	Interrupt
Chiamata a Sistema Operativo dal programma utente (System Call)	Interno	Eccezione
Overflow aritmetico	Interno	Eccezione
Uso di istruzione non definita	Interno	Eccezione
Malfunzionamento hardware	Interno	Eccezione
Malfunzionamento hardware	Esterno	Interrupt

Routine di Gestione dell'Eccezione (Exception Handler)

- Esistono vari metodi
 - Vettore di Interruzione (approccio tradizionale)
 - Tabella delle Routine di Gestione (approccio RISC)
 - Salto ad indirizzo fisso (approccio MIPS)

Sia '**causa**' il numero di identificazione dell'eccezione

Vettore di Interruzione

- Memorizzo in una tabella gli indirizzi delle Routine di Gestione per ogni possibile causa
 - 370, 68000, Vax, 80x86, . . .
- Sia 'base' l'indirizzo base di tale tabella
- Lo stato viene salvato facendo ricorso allo stack
- Il valore del PC sara': $PC \leftarrow MEM[base+causa*4]$

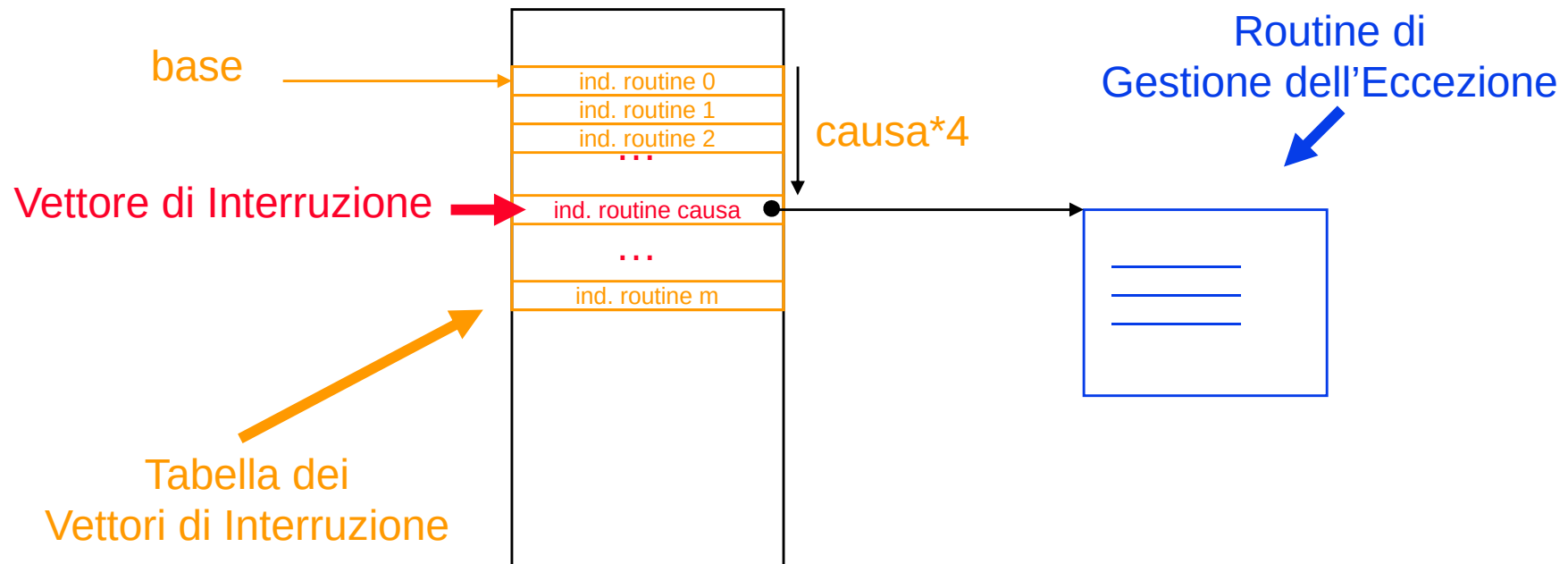
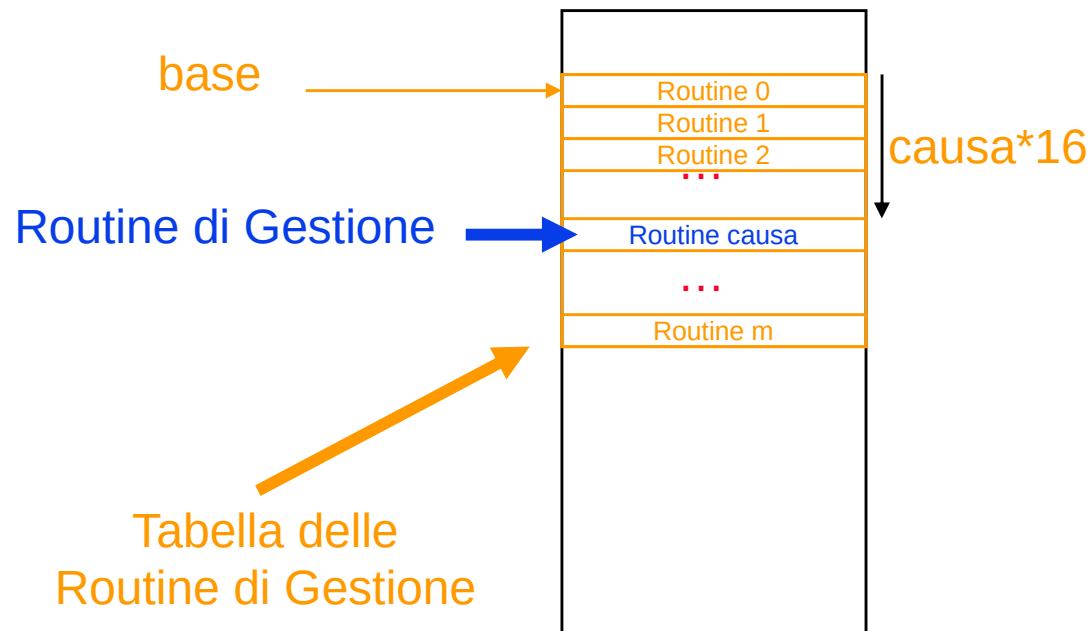


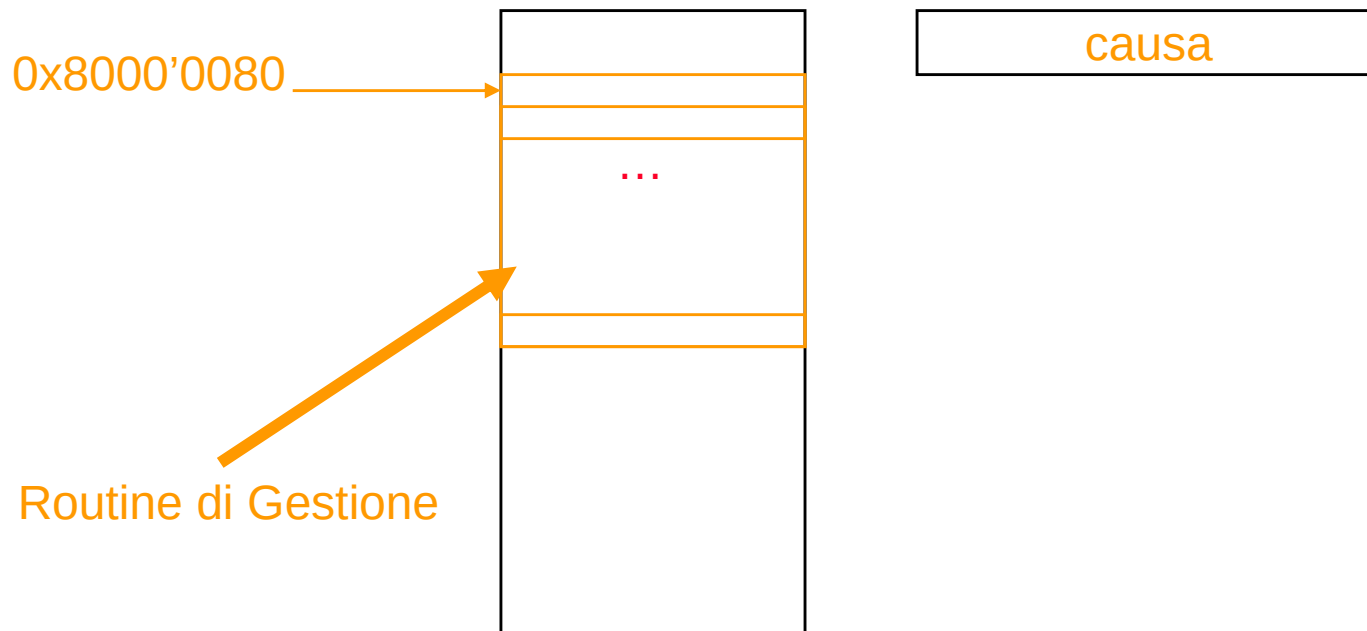
Tabella delle Routine di Gestione

- Memorizzo in una tabella direttamente le Routine di Gestione per ogni possibile causa
 - Sparc, PA, M88K, . . .
- Sia 'base' l'indirizzo base di tale tabella
- Il valore del PC sara': $PC \leftarrow \text{base} + \text{causa} * 16$
- Salto senza far uso dello stack; nel codice iniziale ho piu' flessibilita'



Salto ad indirizzo fisso

- Salto sempre ad un indirizzo fisso che nel MIPS vale 0x8000'0080
- Il valore del PC sara': $PC \leftarrow 0x8000'0080$
- Salto senza far uso dello stack; nel codice della routine di gestione analizzo i vari casi
- In pratica ho una piccola tabella per gestire gli eventi:
 - RESET, Overflow, Istruzione Sconosciuta, TLB, ...



Salvataggio dello stato durante un'eccezione

- **Salvataggio sullo stack (Push)**
 - Vax, 68k, 80x86
- **Registri ausiliari (Shadow Registers)**
 - M88k, ARM
 - Lo stato e' salvato in registri ausiliari sia visibili che non visibili direttamente all'utente (es. registri interni della pipeline)
- **Salvataggio in registri speciali**
 - MIPS
 - 4 Registri: EPC, BadVaddr, Status, Cause

Supporto alla gestione delle eccezioni nel MIPS ISA

- Vari registri a 32 bit:
- **EPC**
contiene l'indirizzo dell'istruzione "colpevole"
- **Status**
contiene i bit di mascheramento e i bit di abilitazione
- **Cause**
i bit 5,4,3,2 di questo registro codificano le possibili sorgenti di eccezione, ad es.:
 - undefined instruction=0
 - arithmetic overflow=1
- **BadVAddr** (Bad Virtual Address)
contiene l'indirizzo di memoria al quale si è verificato un riferimento di memoria "sbagliato"

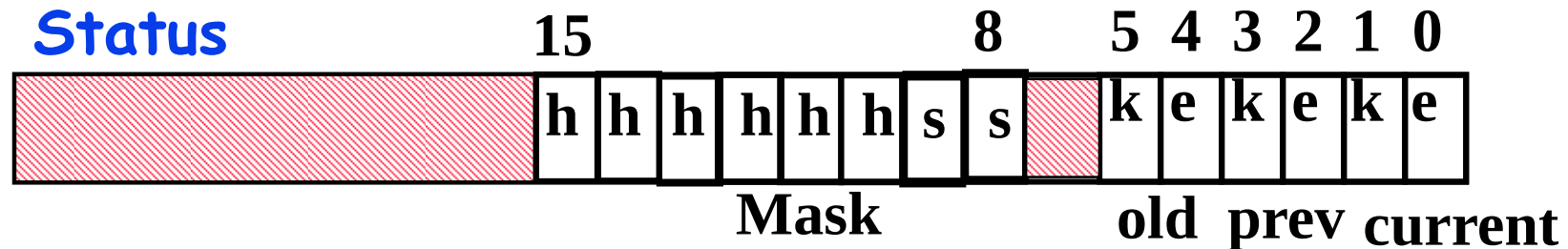
Dove si trovano tali registri

- **EPC** registro 14 coprocessore 0
- **Status** registro 12 coprocessore 0
- **Cause** registro 13 coprocessore 0
- **BadVAddr** registro 8 coprocessore 0

'coprocessore 0'
non e' altro che
una parte della CPU

- Al verificarsi di un'eccezione, la parte di controllo del microprocessore modifica: EPC, Status, Cause, BadVAddr, PC
 - Nota 1:
 - In fase di fetch il PC (oldPC) e' stato aggiornato PC+4 (newPC). Per poter identificare l'istruzione "colpevole" e' necessario far riferimento a oldPC anziche' a newPC. Quindi in EPC viene scritto oldPC.
 - Nota 2:
 - PC (newPC) viene (sovra-)scritto direttamente il nuovo valore (PC ← 0x8000'0080)

Status Register (1)

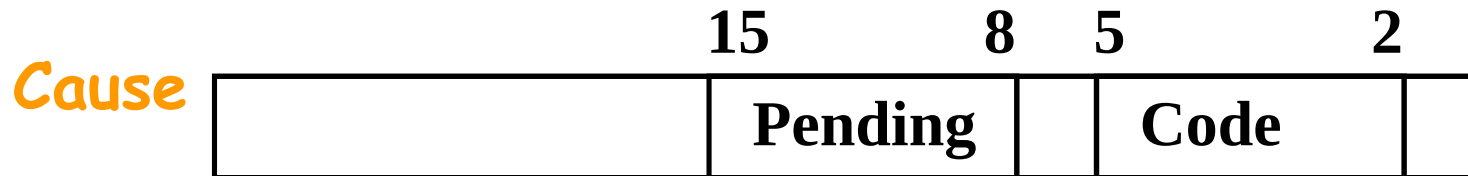


- **Mask (8 bits)** = 1 bit per ognuna di 6 sorgenti di eccezione hardware e 2 sorgenti di eccezione software
 - 1 => abilita quel tipo di eccezione
 - 0 => disabilita quel tipo di eccezione
- **k** = kernel/user bit (v. slide successiva)
 - 0 => ero nel kernel quando si e' verificata l'eccezione
 - 1 => ero nella modalita' utente
- **e** = interrupt enable bit
 - 0 => interrupt sono (erano) disabilitati
 - 1 => interrupt sono (erano) abilitati
- **Al verificarsi di un'eccezione**
 - i 4 bit LSB sono fatti scorrere verso sinistra di 2 posizioni
 - i nuovi 2 bit LSB sono azzerati; questo significa
 - esecuzione in modalita' kernel e con gli interrupt disabilitati

Status Register (2): Modalita' User/Kernel

- Il calcolatore puo' gestire se stesso utilizzando due modalita' di esecuzione denominate classicamente
 - Modalita' User
 - Modalita' Kernel
- Il Sistema Operativo puo' essere visto come un programma speciale in esecuzione sul calcolatore in una modalita' privilegiata (modalita' kernel) in cui:
 - Tutte le risorse del calcolatore possono essere utilizzate direttamente
 - Al software utente vengono presentate risorse "virtuali" che sono piu' potenti delle risorse fisiche, es.:
 - si forniscono i "file" anziche' i "settori del disco"
 - si fornisce una memoria molto piu' grande (memoria virtuale)
 - Viene garantita la protezione fra i vari programmi utente o di piu' utenti
- Le eccezioni consentono al sistema di rispondere ad eventi che si verificano mentre un qualsiasi programma utente e' in esecuzione
 - Il Sistema Operativo entra in azione dal momento in cui inizia ad essere eseguita la routine di gestione delle eccezioni

Cause Register

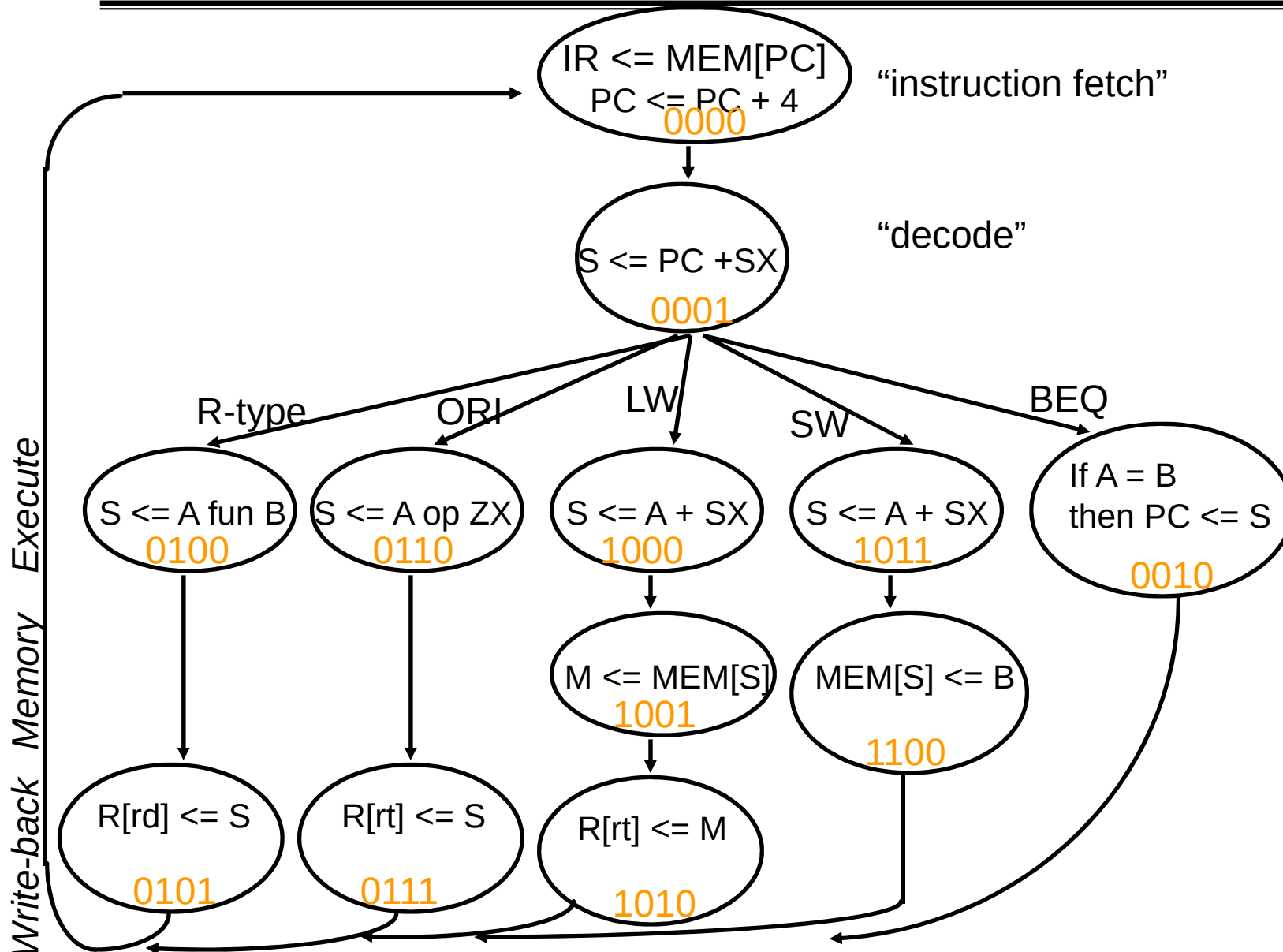


- **Pending (8 bits)** assegna 1 bit ad ognuna delle 6 sorgenti hardware e delle 2 sorgenti software: il bit vale 1 se l'eccezione si e' gia' verificata ma non e' stata ancora servita
 - Serve per gestire i casi in cui si puo' verificare piu' di un'eccezione contemporaneamente (eccezioni annidate o "nested exceptions")
 - Utile per controllare se ci sono eccezioni nonostante si sia disabilitata la partenza della routine di eccezione per quella sorgente specifica (nello status Register)
- **Code (4 bits)** codifica la regione dell'eccezione
 - 00 (INT) → external interrupt
 - 04 (ADDRL) → address error exception (load or instr fetch)
 - 05 (ADDRS) → address error exception (store)
 - 06 (IBUS) → bus error on instruction fetch
 - 07 (DBUS) → bus error on data fetch
 - 08 (Syscall) → system call exception
 - 09 (BKPT) → breakpoint exception
 - 10 (RI) → reserved instruction exception
 - 12 (OVF) → arithmetic overflow exception

Interruzioni Precise/Imprecise

- **Precisa** → lo stato della macchina viene conservato, come se il programma avesse eseguito fino all'istruzione "colpevole"
 - Posizione adottata nei calcolatori IBM
 - Vantaggio: il codice di sistema potrà essere trasportato senza problemi su una implementazione diversa della stessa architettura
 - Svantaggio: difficile da implementare nel caso di pipeline, out-of-order execution, ...
- **Imprecisa** → è il software di sistema che deve occuparsi di scoprire in che stato il sistema si trova e quindi prendere le opportune decisioni
 - Posizione adottata nei primi processori MIPS (i più recenti usano le interruzioni PRECISE)
 - Svantaggio: le routine di gestione delle interruzioni debbono essere probabilmente riscritte se si cambia microprocessore MIPS
 - Vantaggio: snellisce l'hardware del microprocessore nei casi più frequenti
- Il motivo principale per cui in alcuni processori si evita l'uso degli interrupt precisi è legato essenzialmente a obiettivi di prestazioni elevate
 - Chiaramente questo può mettere in crisi: sviluppatori del software di sistema, utenti, il mercato...

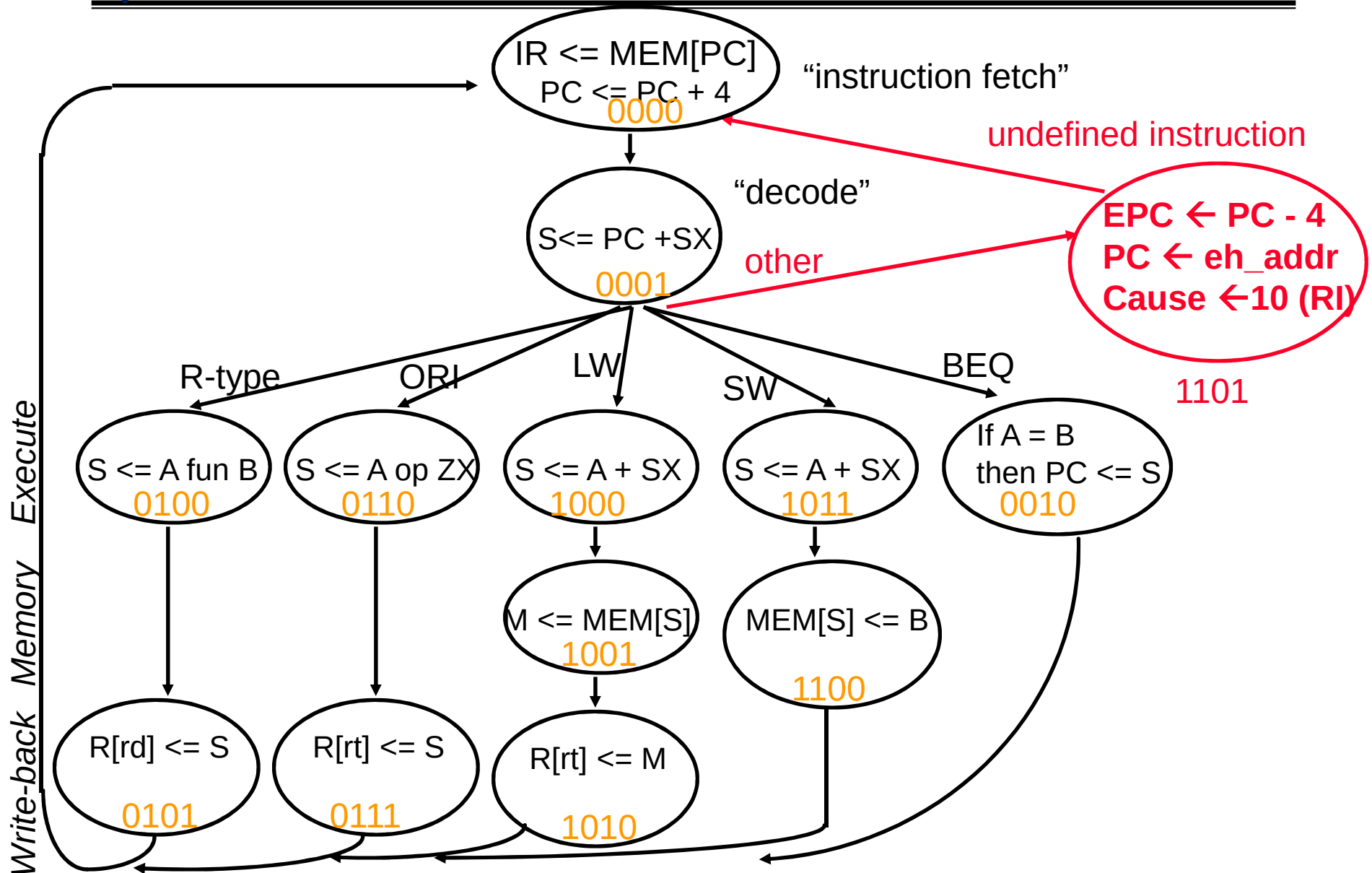
Macchina a Stati Finiti che descrive il nostro MIPS



Modifica del MIPS per poter gestire le eccezioni

- Supponiamo di partire dal processore "base" (che non gestisce eccezioni) e di voler gestire eccezioni: es. nel caso di istruzione indefinita.
- L'eccezione "Undefined Instruction" viene riconosciuta nel momento in cui, trovandomi nello stato 0001, non ho uno stato successivo in cui andare quando ho un op-code sconosciuto
 - Questo si puo' implementare definendo un nuovo stato (stato #13 o 1101) per tutti gli op-code diversi da lw, sw, 0 (R-type), jmp, beq, e ori
 - Nella figura e' mostrato simbolicamente con "other" per indicare che l'op-code non e' nessuno di quelli noti uscenti dallo stato #1

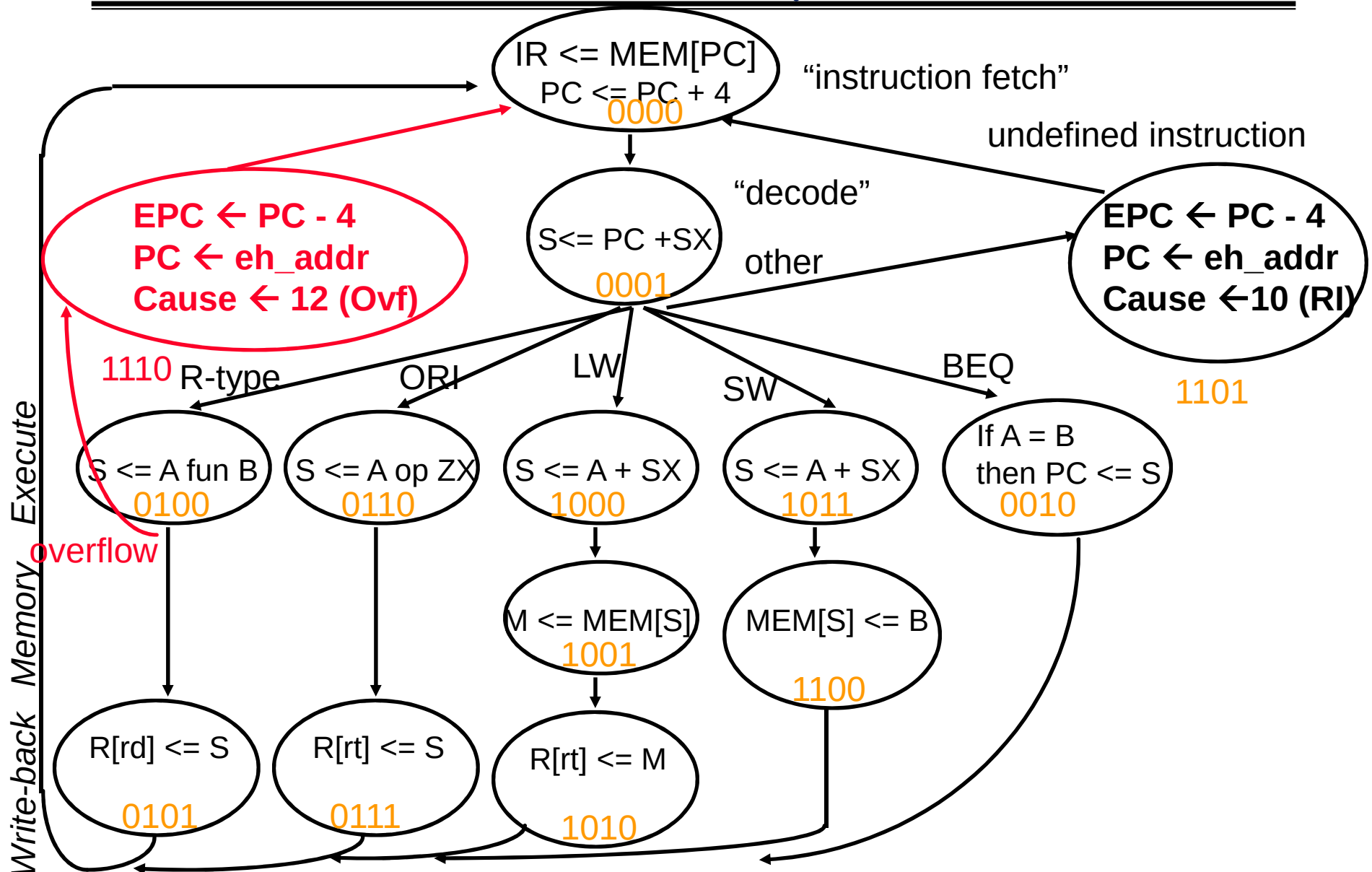
Implementazione del meccanismo di eccezione per una istruzione non definita



Implementazione del meccanismo di eccezione per overflow aritmetico

- L'eccezione "Arithmetic overflow"
 - Può essere implementata con un ulteriore stato #14
- L'Unità di Controllo di un microprocessore reale deve inoltre
 - Gestire tutti gli eventi che si possono generare a causa di eccezioni e le varie interazioni fra istruzioni in modo tale che la logica di controllo rimanga semplice e veloce

Modification to the Control Specification



Lettura/Scrittura di EPC, Cause, Status, BadVAddr

- `mtc0 xx, $2` $\rightarrow$ $xx \leftarrow \$2$
move to coprocessor 0 from register
- `mfc0 $2, xx` $\rightarrow$ $\$2 \leftarrow xx$
move from coprocessor 0 into register
- `lwc0 xx, 100($2)` $\rightarrow$ $xx \leftarrow \text{MEM}[\$2+100]$
load word into coprocessor 0 from memory
- `swc0 xx, 100($2)` $\rightarrow$ $\text{MEM}[\$2+100] \leftarrow xx$
store word from coprocessor 0 into memory
- `xx` indica uno dei registri speciali `CAUSE`, `EPC`, ...

Gestore delle Interruzioni

```
.ktext 0x80000080
mfc0 $k0, $13      # $k0 ← Cause (Cop0 reg #13)
mfc0 $k1, $14      # $k1 ← EPC (Cop0 reg #14)

andi $k0, $k0, 0x3C # prelevo il tipo dell'eccezione (bit 5..2)
beq  $k0, $0, fatto # ignora le interruzioni (ma non le eccezioni)

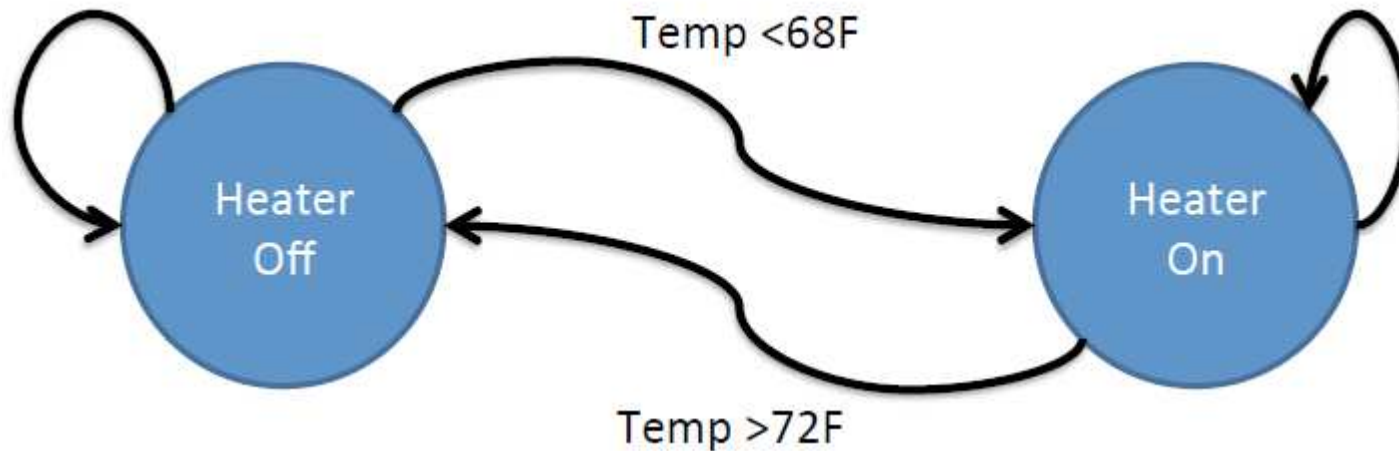
sw $a0, save0      # ipotesi: gestore non rientrante
sw $a1, save1      # necessario evitare uso stack (puo' essere la causa...)
add $a0, $k0, $0
add $a1, $k1, $0
jal gestione       # routine di gestione eccezione, param. input $a0,$a1
lw $a0, save0
lw $a1, save1

fatto:
addiu $k1, $k1, 4   # istr. succ. a quella che ha causato l'eccez.
rfe                # restore from exc.: ripristina mask e (k,e) in Status
jr $k1             # SALTA all'istr.succ. a quella che ha causato l'eccez

.kdata
save0: .word 0
save1: .word 0
```

Macchine a Stati Finiti

- Un gruppo di stati e di transizioni

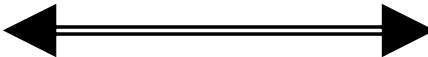


- Ci si sposta da uno stato all'altro seguendo una linea di transazione SE la condizione dalla transizione e' soddisfatta

Macchine a Stati Finiti (2)

- Nel campo dell'architettura dei calcolatori per le FSM:
 - Le transizioni tipicamente avvengono su fronti del clock
 - Sono complete
 - Tutti gli stati definiscono transizioni per tutti gli input
 - Sono deterministiche
- Possono essere trasformate in reti logiche dette Reti Sequenziali
 - Lo vedremo prossimamente.

Lezione 11: BUS

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

_____ filo

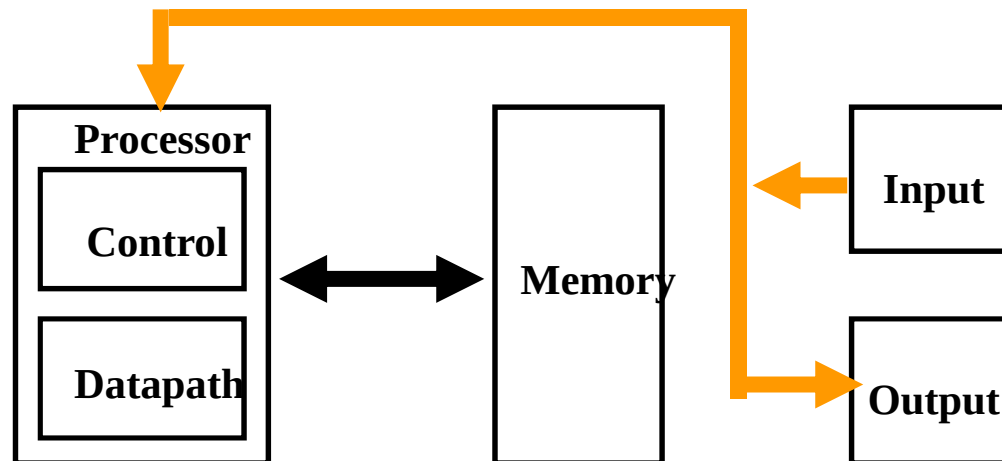
_____ / n
n fili

_____> filo che trasmette in una direzione

_____<—> filo che trasmette in entrambe le direzioni

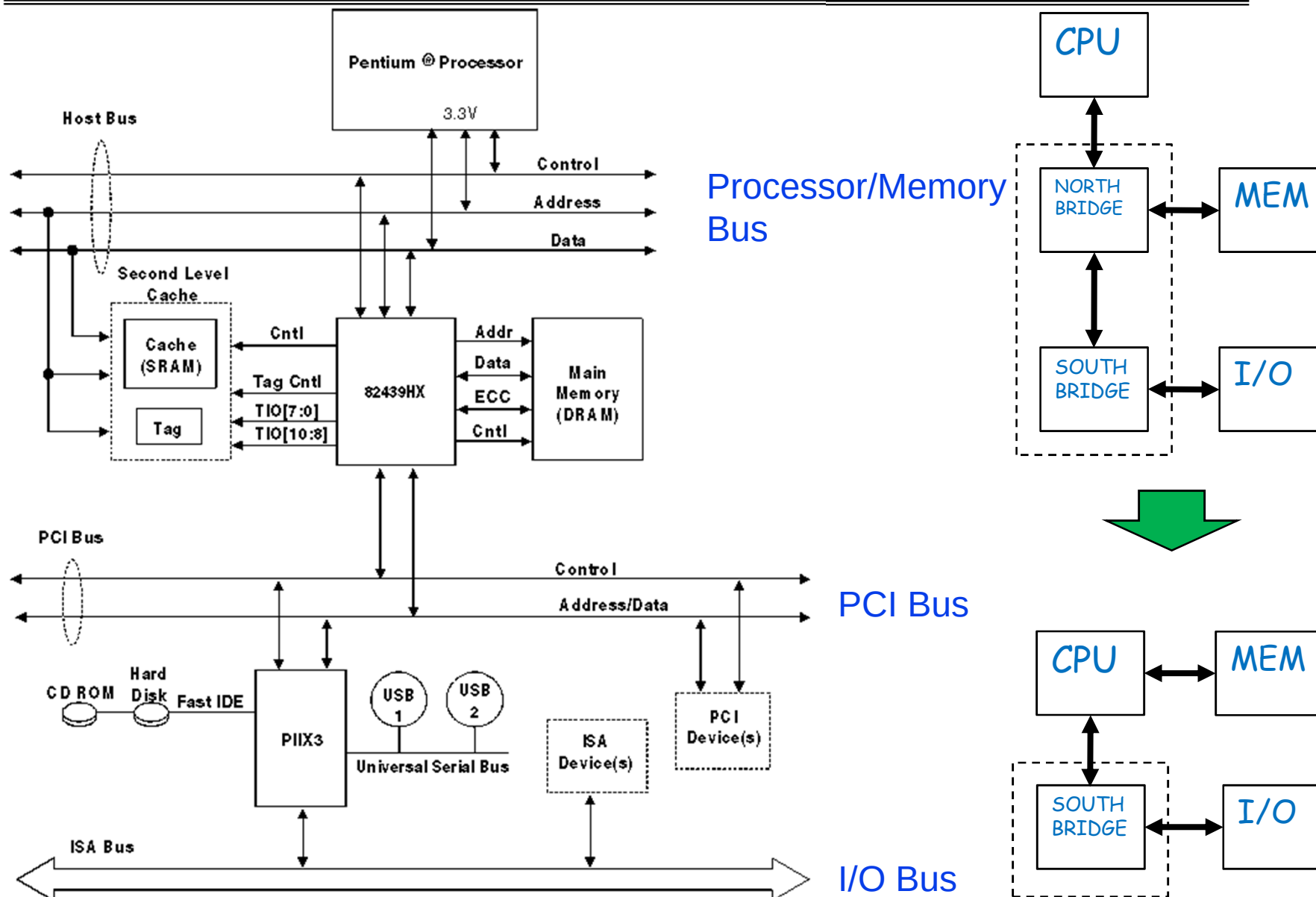
Un bus e':

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

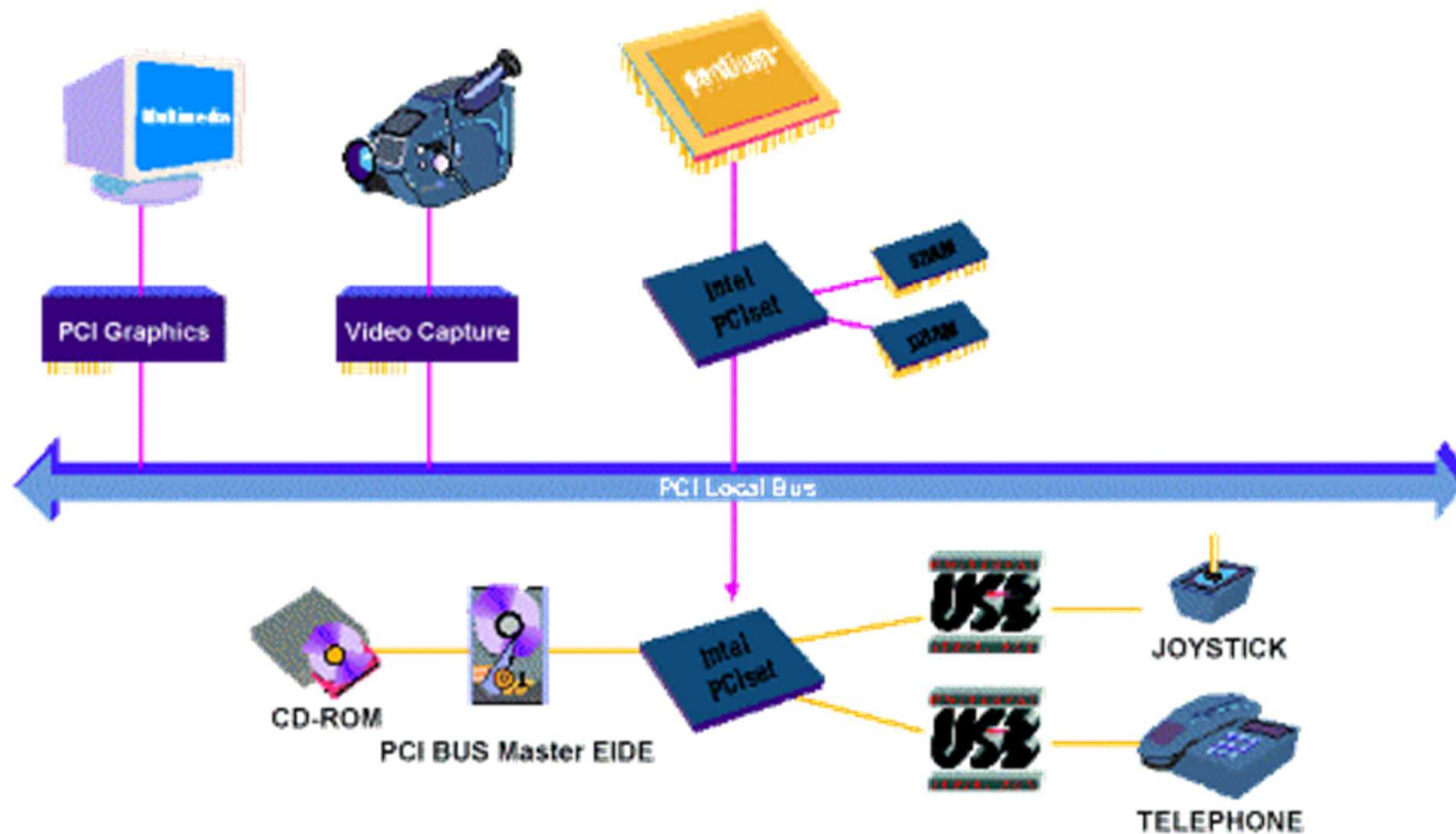


- Un bus e' anche il mezzo fondamentale per costruire sistemi anche grandi e complessi
 - E' uno strumento di astrazione molto utilizzato

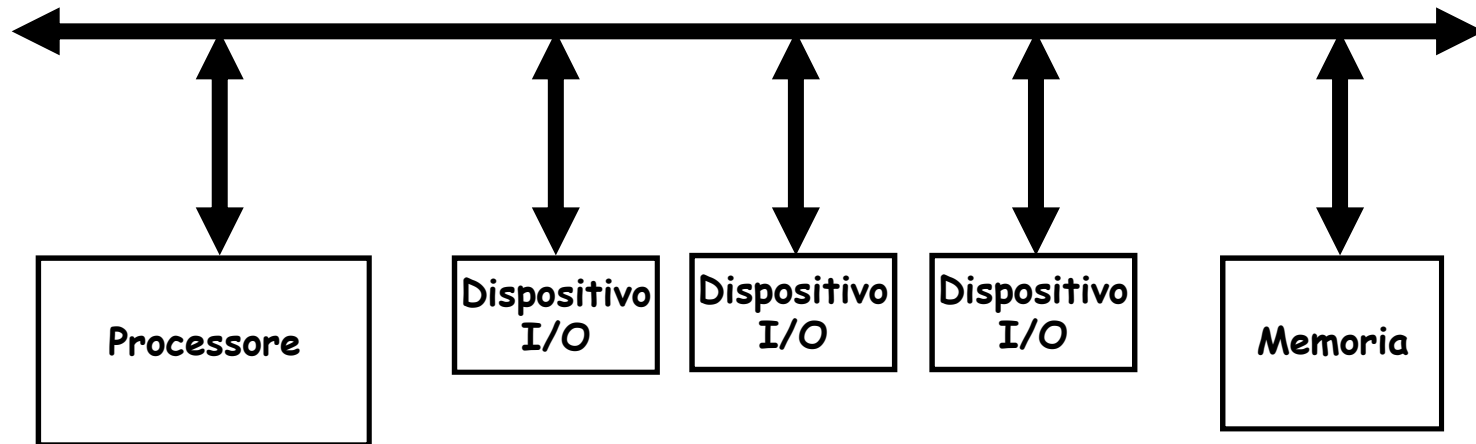
Esempio: Organizzazione di Sistema con CPU Pentium



Esempi di dispositivi collegabili tramite bus

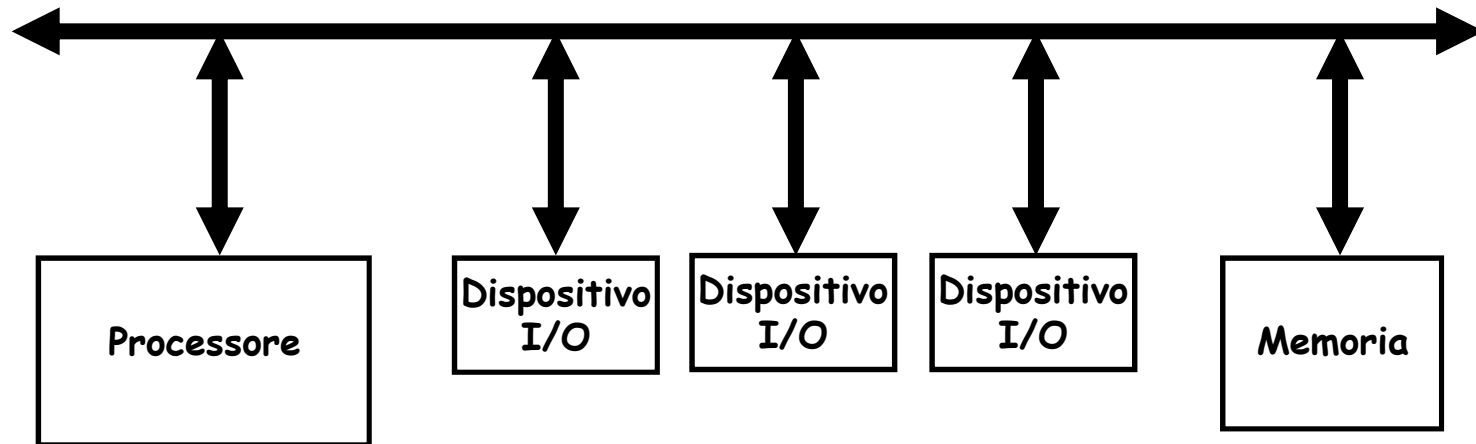


Vantaggi dei bus



- **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

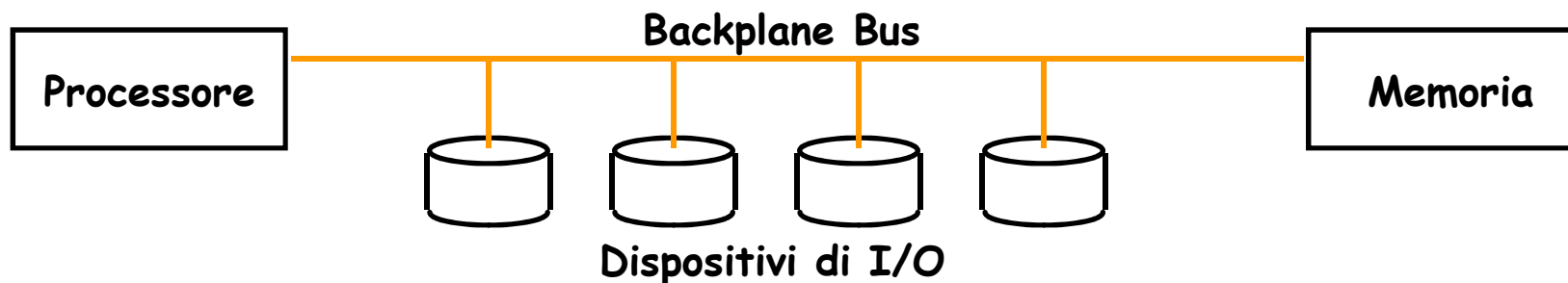


- Genera un “collo di bottiglia” nelle comunicazioni
 - La banda del bus limita il massimo throughput ottenibile per l'I/O
- La velocità massima del bus è fortemente limitata da
 - **Lunghezza** del bus
 - **Numero** di dispositivi collegati al bus
 - **Necessità** di supportare un ampio numero di dispositivi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

Tipi di Bus

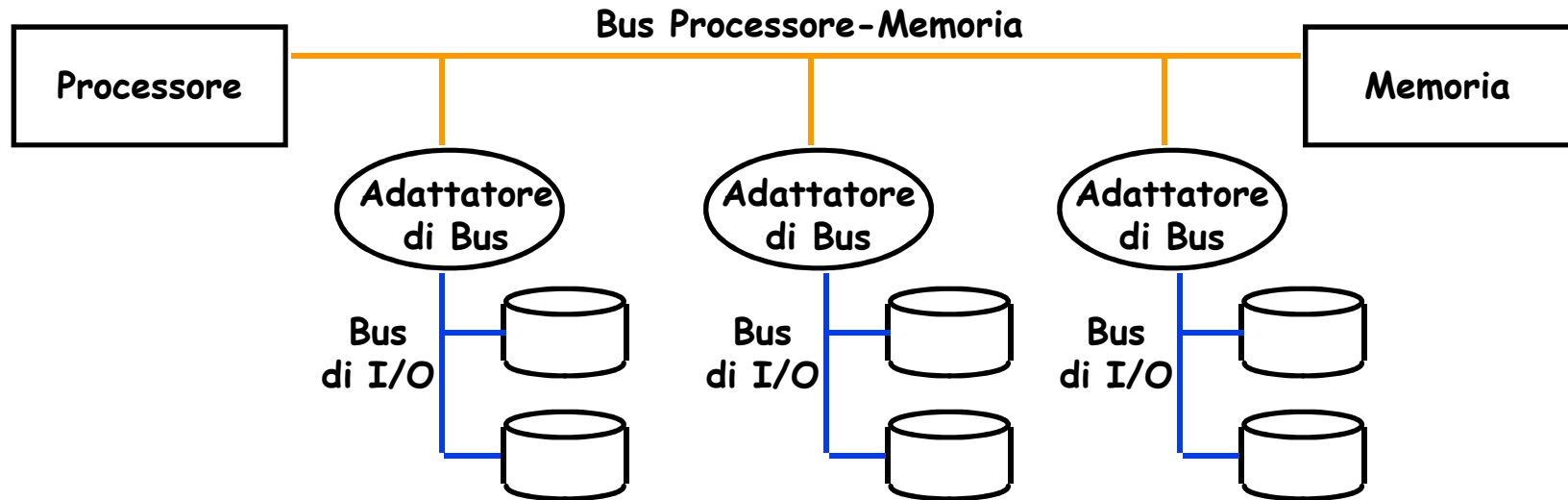
- **Bus Processore-Memoria** (specifico del sistema)
 - Corto e ad alta velocità
 - Deve solo essere adatto al sistema di memoria utilizzato
 - Tipicamente cerca di massimizzare la banda processore-memoria
 - Connette direttamente il processore
 - Ottimizzato per i trasferimenti di blocchi di cache
- **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



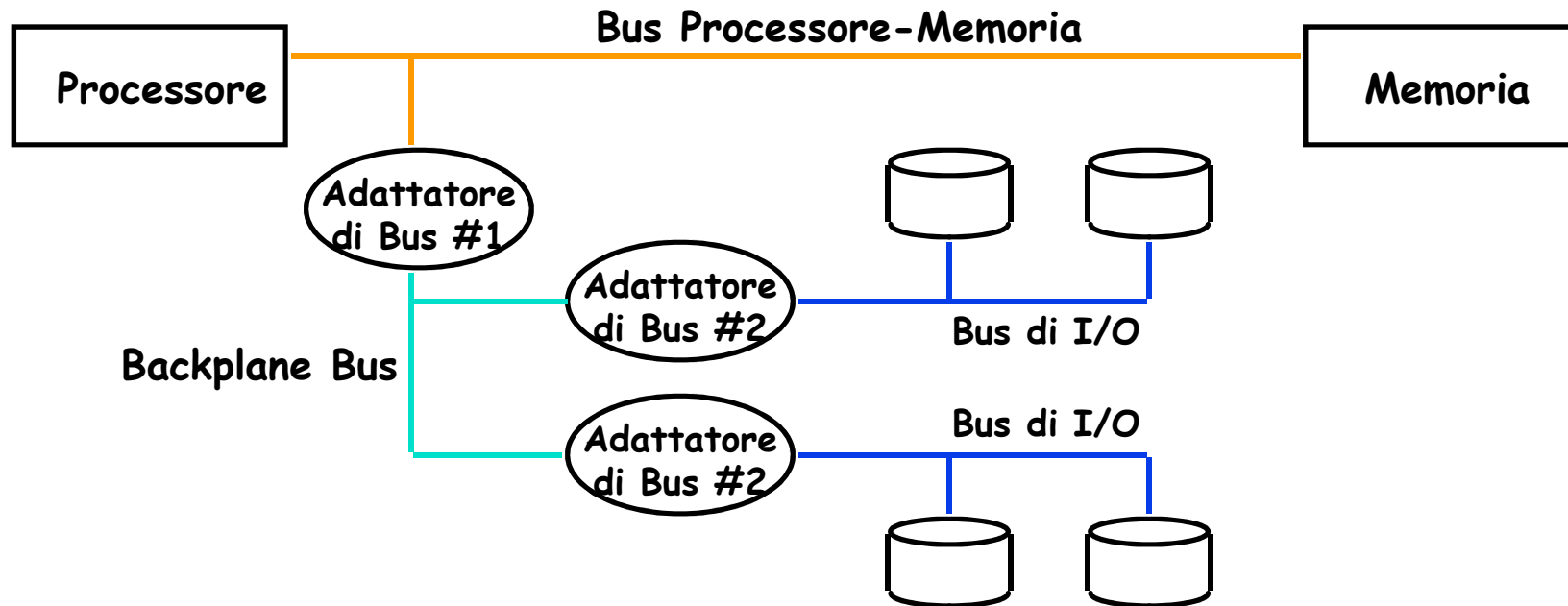
- Si può usare un sistema a singolo bus (backplane bus) per la
 - Comunicazione fra processore e memoria
 - Comunicazione fra dispositivi di I/O e memoria
- Vantaggi: semplice e a basso costo
- Svantaggi: lento e principale collo di bottiglia del sistema
- Esempio: PC-AT della IBM

Sistema con due bus



- I bus di I/O si connettono al bus processore-memoria attraverso degli "adattatori" o "bridge"
 - Bus processore-memoria: usato principalmente per il traffico processore-memoria
 - Bus di I/O: forniscono gli slot di espansione per i dispositivi di I/O
- Esempio: Macintosh-II della Apple
 - NuBus: collega(va) processore, memoria, e alcuni dispositivi di I/O
 - Bus SCSI: collega(va) il resto dei dispositivi di I/O

Sistema con tre bus



- Un basso numero di backplane bus si inserisce sul bus processore-memoria
 - Il bus processore-memoria e' usato per il traffico processore memoria
 - I bus di I/O sono connessi al backplane bus
- Vantaggio: il carico sul bus del processore e' fortemente ridotto

Come si definisce un bus?

Protocollo delle Transazioni

Specifica Temporizzazioni e Segnali

Insieme di fili

Specifiche Elettriche

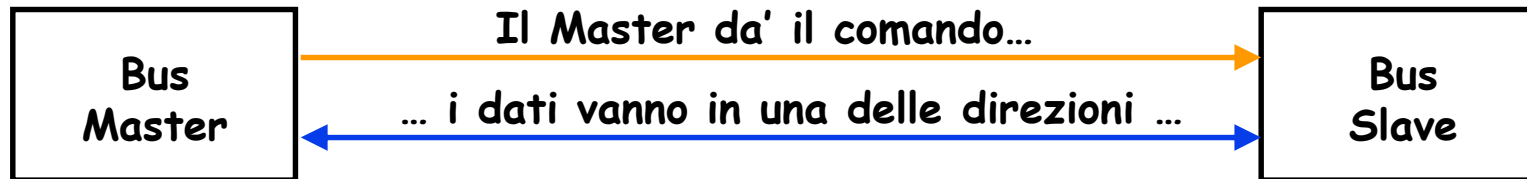
**Caratteristiche fisiche e meccaniche,
e tipi di connettori**

Organizzazione generale di un bus



- **Linee di Controllo**
 - Segnalano la richiesta (request) e la conferma (acknowledgment)
 - Indicano che tipo di informazione sta transitando sulle linee dati
- **Linee Dati** trasportano le informazioni fra chi invia (sorgente dati o source) e destinatario (destination)
 - **Dati (veri e propri) e Indirizzi**
 - Comandi complessi

Master/Slave e Transazioni



- Una transazione sul bus include tre fasi
 - Decidere chi e' master
 - fase di arbitraggio
 - Dare il comando (o l'indirizzo)
 - fase di richiesta
 - Trasferire i dati
 - fase di azione
- Il **Master** e' l'unita' che:
 - Ha il controllo del bus
 - Inizia la transazione dando un comando o specificando un'indirizzo
- Lo **Slave** e' l'unita' che:
 - Viene attivata dalla transazione ovvero risponde alla richiesta
 - Invia i dati al master, se il master richiede dati
 - Preleva i dati inviati dal master, se il master vuole mandare dati

Bus Sincroni e Bus Asincroni

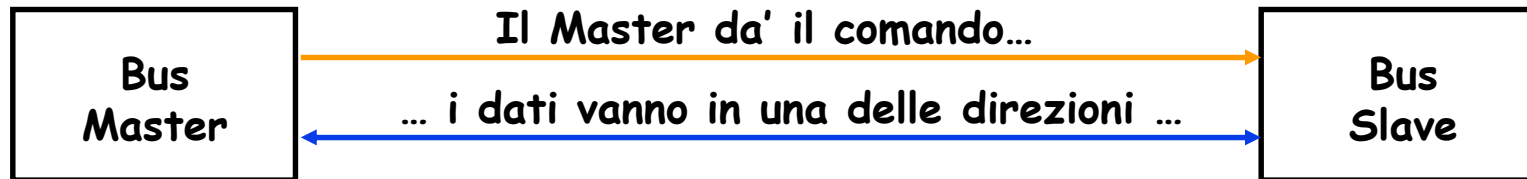
- **Bus Sincrono:**

- Fra le linee di controllo c'e' il segnale di clock
- Il protocollo di comunicazione e' fisso ed e' 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 velocita'

- **Bus Asincrono:**

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

Arbitraggio: ottenere accesso al Bus

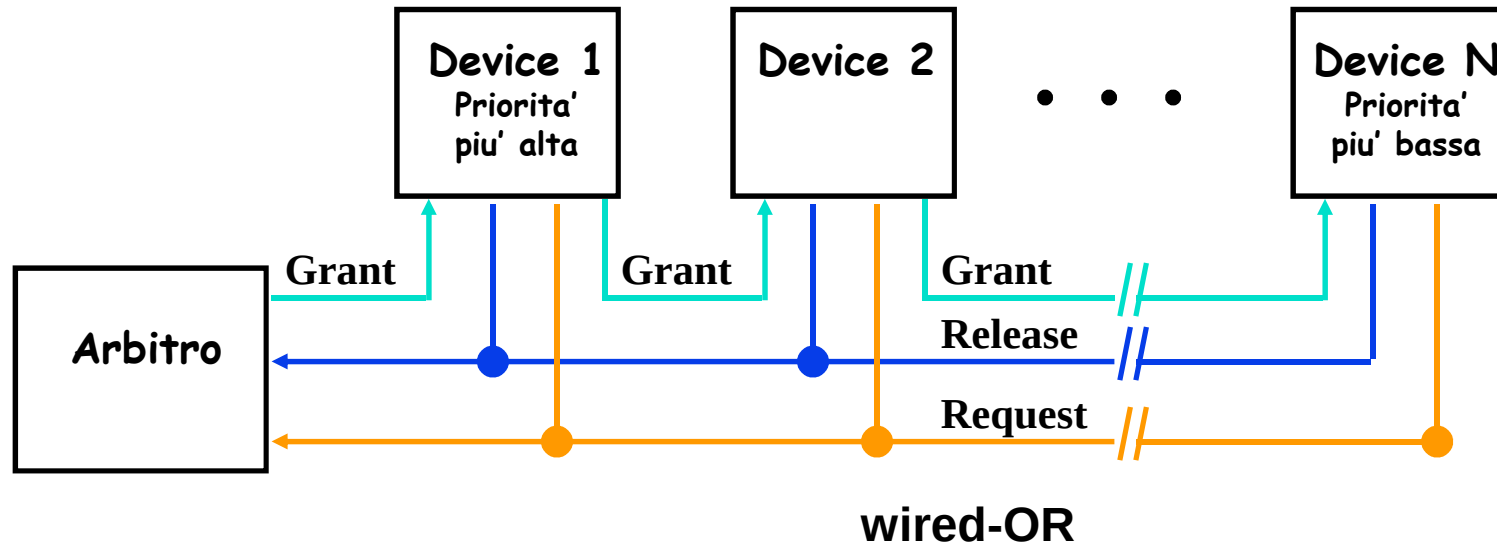


- E' uno dei problemi piu' importanti nella progettazione di un bus
 - Come viene prenotato il bus da un dispositivo che desidera usarlo?
- Innanzitutto per semplificare la situazione si deve arrivare alla definizione di chi e' il master e chi e' lo slave
 - A quel punto solo il bus-master potra' controllare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

Bus-Master multipli: necessita' di Arbitraggio

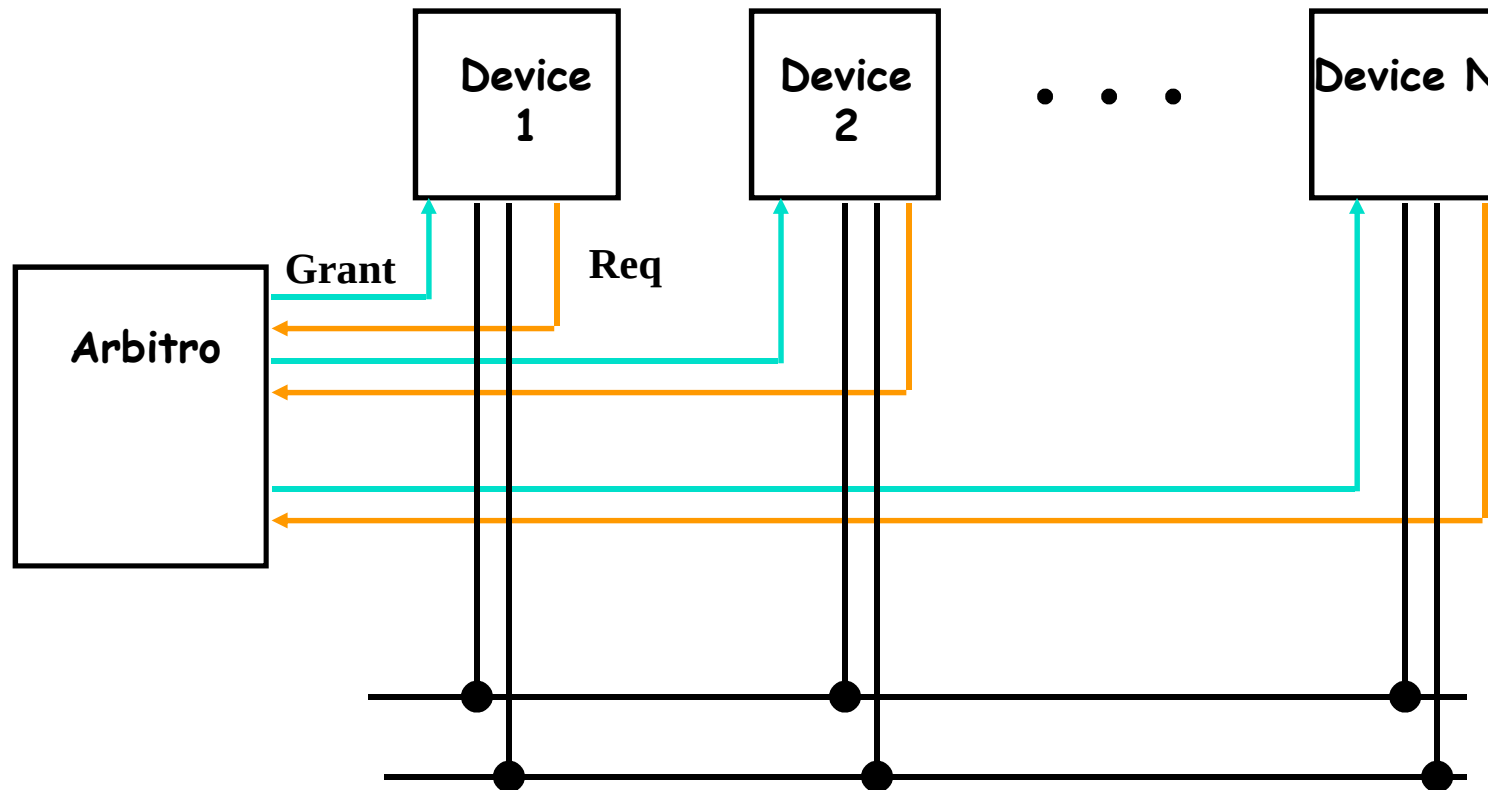
- Schema di arbitraggio del bus
 - Un bus-master che voglia usare il bus segnala la richiesta ad un Arbitro
 - Il bus-master non puo' usare subito il bus resta in attesa del permesso (grant)
 - Il bus-master dovra' infine segnalare all'Arbitro che ha finito di usare il bus
- L'Arbitro cerca di bilanciare due fattori
 - **Priorita' della richiesta**: il dispositivo a priorita' maggiore deve essere servito per primo
 - **Fairness**: anche se un dispositivo ha bassa priorita' 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 e' possibile garantire la fairness:
 - Un dispositivo a valle potrebbe rimanere bloccato indefinitamente
 - Il tempo di propagazione del segnale di grant puo' limitare la velocita' del bus
- Es.: VMEbus

Arbitraggio Centralizzato Parallelo

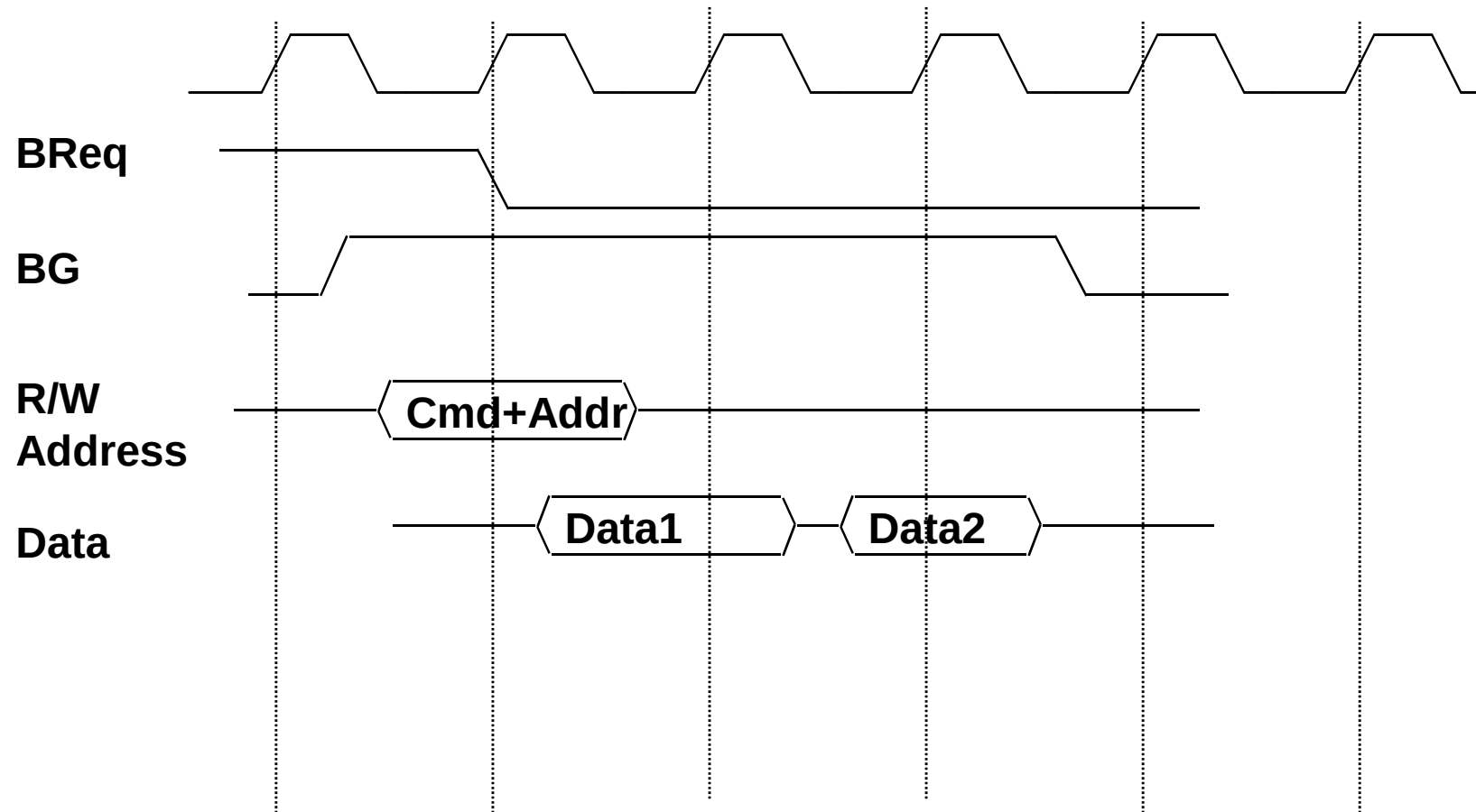


- Usato nella quasi totalita' dei bus processore-memoria e nei bus di I/O ad alta velocita'
- Inoltre e' usato nei bus: Multibus I, EISA, PCI

Arbitraggio Distribuito

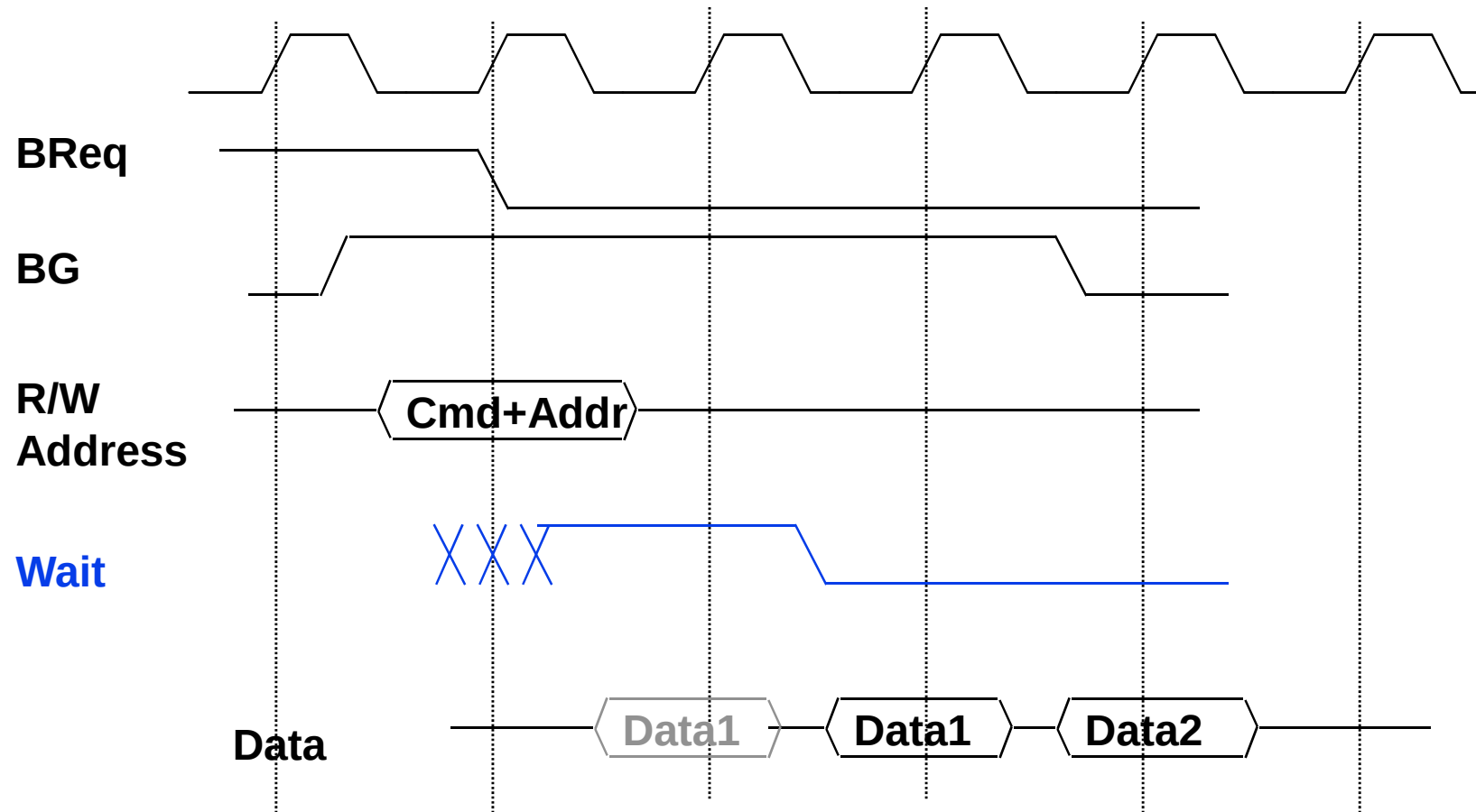
- La scelta del Master non viene fatta da una entita' precisamente identificabili
- Esempi:
 - Multibus I: arbitraggio distribuito in daisy-chain
 - Inizialmente il bus e' controllato da un'unita' a priorita' intermedia M
 - La richiesta da parte di un'unita' A a priorita' piu' alta attiva la logica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

Protocollo Sincrono (Semplice)



- Anche un bus di memoria e' piu' complesso di questo
 - La memoria (slave) puo' prendere piu' cicli per rispondere
 - Dovra' quindi segnalare che non e' in grado di accettare indirizzi...

Protocollo Sincrono (Tipico)



- Con WAIT lo slave indica che e' pronto per il trasferimento dei dati (es. lettura)...
- Poi il trasferimento ha luogo alla velocita' del bus

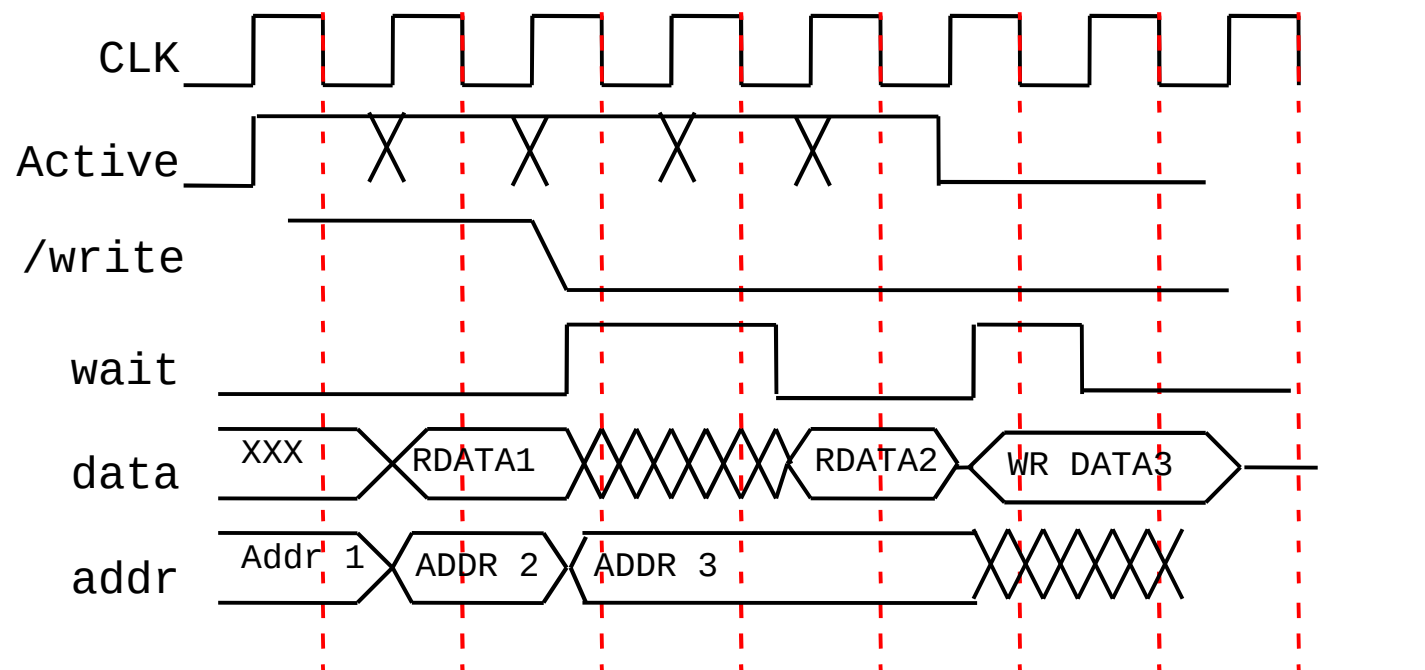
Come aumentare la banda del bus

- **Uso di linee indirizzi/dati separate anzichè 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 master (processore-cache)



Aumento delle Transazioni su Bus Multimaster

- Arbitraggio sovrapposto (overlapped)
 - Si guadagna tempo effettuando la fase di arbitraggio per la transazione successiva a quella in corso
- “Bus parking”
 - Il master mantiene il bus ed effettua varie transazioni senza passare nuovamente alla fase di arbitraggio fino a che qualche altro master 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 verterà 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

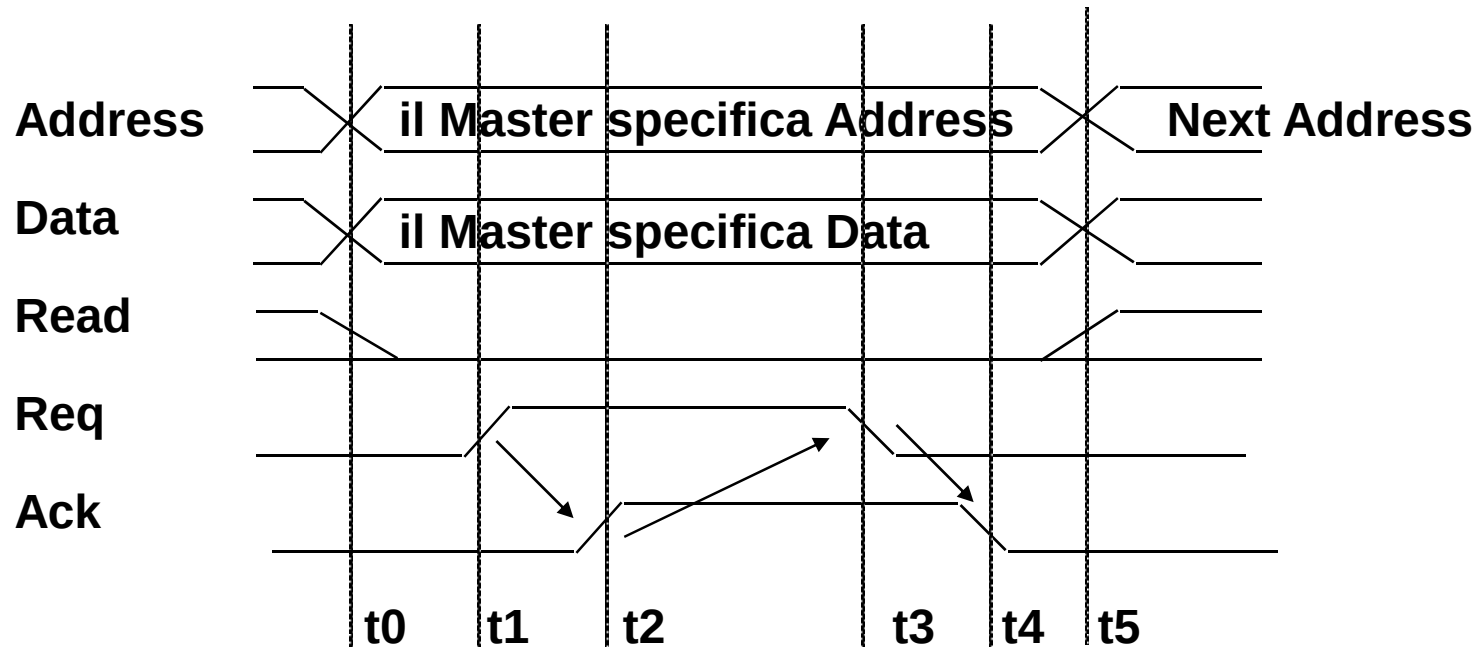
Bus di Memoria per server multiprocessore (1993)

Nome bus	MBus	Summit	Challenge	XDBus
Ditta	Sun	HP	SGI	Sun
Clock (MHz)	40	60	48	66
Linee Indirizzi	36	48	40	muxed
Linee Dati	64	128	256	144 (parity)
Bit dati	256	512	1024	512
Clock/trasferim.		4	5	4?
Picco (MB/s)	320(80)	960	1200	1056
Master	Multi	Multi	Multi	Multi
Arbitraggio	Centrale	Centrale	Centrale	Centrale
Slots		16	9	10
Busses/system	1	1	1	2
Lunghezza		~30 cm	~30 cm	~35 cm

Bus di I/O

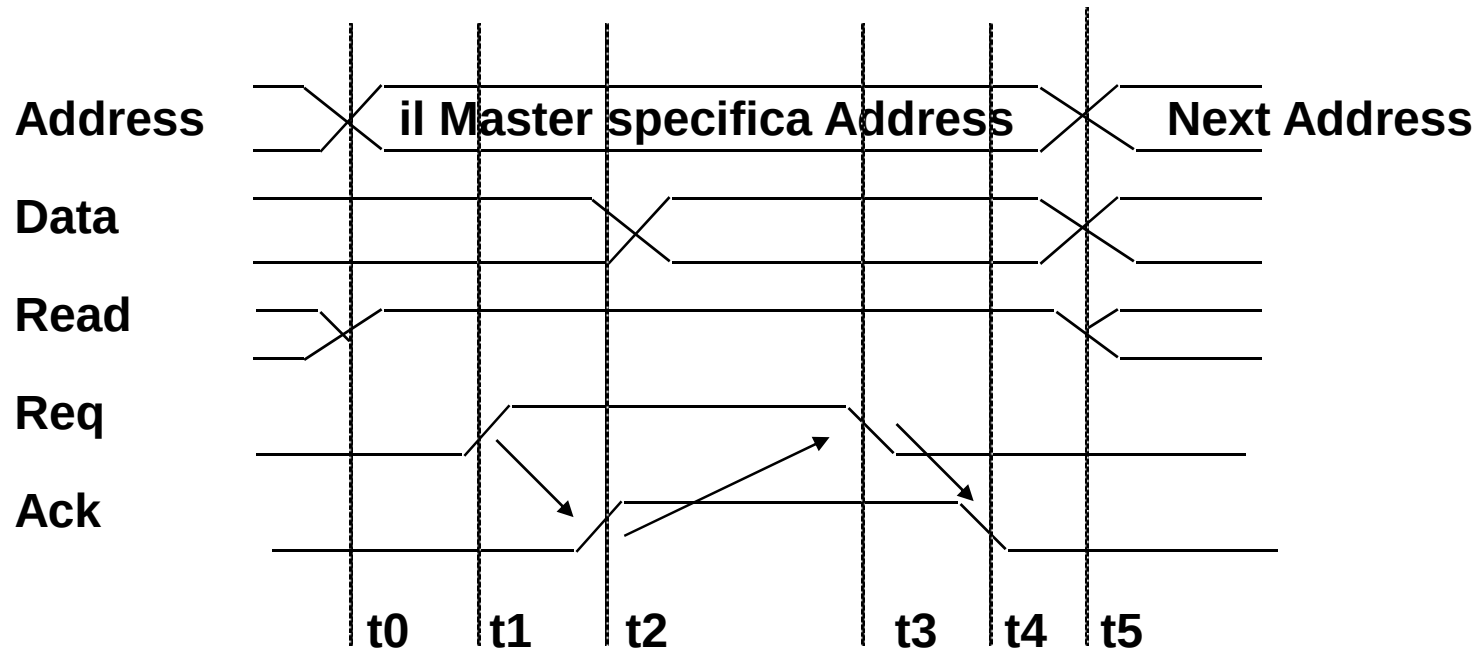
- Progettato per supportare una vasta gamma di dispositivi
 - L'insieme dei dispositivi si potrà evolvere nel tempo
- E' necessario poter gestire velocita' di comunicazione arbitrarie e situazioni con:
 - Processore veloce - I/O lento
 - Processore lento - I/O veloce

Handshake Asincrono: Scrittura



- t0 : Il Master ha ottenuto il controllo e specifica l'indirizzo, la direzione e dati; attende poi un certo intervallo di tempo affinché lo Slave capisca di essere coinvolto in quel trasferimento...
- t1: Il Master attiva la linea "Request"
- t2: Lo Slave attiva la linea "Ack", per indicare di aver ricevuto il dato → ha finito
- t3: Il Master rilascia "Req" → ha capito che lo Slave ha finito
- t4: Lo Slave rilascia "Ack" → ha capito che il Master ha capito

Handshake Asincrono: Lettura



- t0 : Il Master ha ottenuto il controllo e specifica l'indirizzo, la direzione e i dati; attende poi un certo intervallo di tempo affinché lo Slave capisca di essere coinvolto in quel trasferimento...
- t1: Il Master attiva la linea "Request"
- t2: Lo Slave attiva la linea "Ack", per indicare che e' pronto a trasmettere il dato
- t3: Il Master rilascia "Req" avendo ricevuto il dato → ha finito
- t4: Lo Slave rilascia "Ack" → ha capito che il Master ha finito

Backplane/IO Bus (1993)

Nome Bus	SBus	TurboChannel	MicroChannel	PCI
Ditta	Sun	DEC	IBM	Intel
Clock (MHz)	16-25	12.5-25	async	33
Indirizzamento	Virtuale	Fisico	Fisico	Fisico
Bit dati	8,16,32	8,16,24,32	8,16,24, 32,64	8,16,24, 32,64
Master	Multi	Singolo	Multi	Multi
Arbitraggio	Centrale	Centrale	Centrale	Centrale
32bit read (MB/s)	33	25	20	33
Picco (MB/s)	89	84	75	111 (222)
Potenza max (W)	16	26	13	25

Bus di I/O ad alta velocita'

- Esempi
 - grafica
 - Rete ad alta velocita'
- Consentono di collegare un basso numero di dispositivi
- I trasferimenti dati viaggiano sempre a velocita' massima
- Si adottano tecniche particolari di trasferimento come il DMA (Direct Memory Access)
 - Un piccolo processore pompa byte da/verso la memoria
- Ognuna delle due parti puo' aver bisogno di bloccare momentaneamente il trasferimento
 - i buffer che prima o poi si riempiono

Riepilogo degli caratteristiche dei bus

<i>Caratteristica</i>	<i>Obiettivo=prestazioni</i>	<i>Obiettivo=basso costo</i>
Larghezza Bus	linee indirizzi e dati separate	linee indirizzi e dati multiplexate
Dim. Dati	larga=velocita' (e.g., 32 bit)	bassa=basso_costo (e.g., 8 bit)
Dim. Trasfer.	Word multiple → meno gestione bus	trasferim. a singola word sono piu' semplici
Bus-masters	Multipli (rechiiede arbitraggio)	Singolo master (nessun arbitraggio)
Clock	Sincrono	Asincrono
Protocollo	con "pipeline"	seriale

Bus Punto-Punto

- Bus utilizzati per collegamenti ad altissima velocità, ad es. (anche nei PC):
 - HYPERTRANSPORT™ – dal 2001 (www.hypertransport.org)
 - Es. Processori AMD
 - QUICKPATH™ – dal 2008
 - Es. Processori Intel (Nehalem e successivi)
- Velocità di trasferimento: 30/40 GB/s

Lezione 12

Tecniche di pilotaggio dei dispositivi di I/O

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Gestione dell'Input/Output

- Le periferiche si differenziano notevolmente fra loro perche'
 - Trasferiscono differenti quantita' di dati
 - Funzionano a velocita' diverse
 - Utilizzano 'formati di dato' differenti
- Tipicamente funzionano a velocita' piu' basse rispetto a CPU e Memoria
- Per interfacciarsi con il resto del sistema tipicamente una periferica ha bisogno di un "controller" o "interfaccia"

Velocita' 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

• 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)

• ETHERNET

- classica (1980) 10 MB/s = 1.25MB/s
- 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

• 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)

Requisiti di Banda nel Multimedia

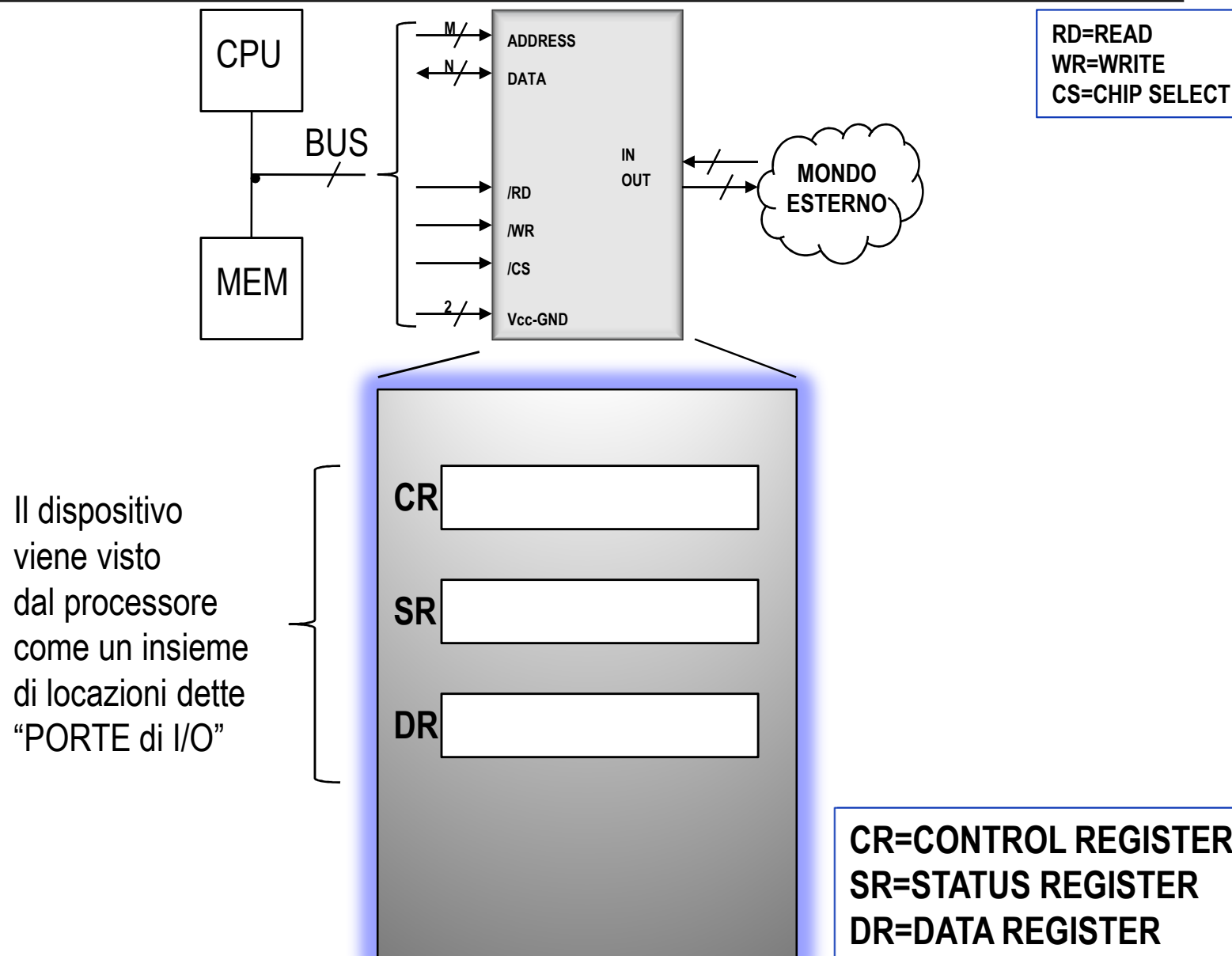
- **Reduced Quality Audio (Mono)**
 - (11,050 audio samples / sec) (8-bit samples) (1 channel) = 0.1 Mbps (11 kB/s)
- **High Quality Audio (Stereo)**
 - (44,100 audio samples / sec) (16-bit samples) (2 channels) = 1.4 Mbps (176 kB/s)
- **Reduced Quality Video**
 - QVGA: 320x240@15fps (16-bit color / pixel) = 18 Mbps (2.2 MB/s)
- **High Quality Video**
 - VGA: 640x480@30fps (24-bit color / pixel) = 221 Mbps (27.6 MB/s)
- **La compressione cambia radicalmente queste quantita'...**
 - H.264 encode
 - HD720p→1280x720@30fps(24bit)=633Mbps→14-56Mbps+100Gop/sec
 - HD1080p→1920x1080@30fps(24bit)=1424Mbps→20-80Mbps
 - Digital TV
 - 2004: 9000op/pixel →450 Gop/s, 2008: 18000 op/pixel →900 Gop/s

Architettura del Sistema di I/O

- L'architettura del sistema di I/O e'...
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 piu' 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

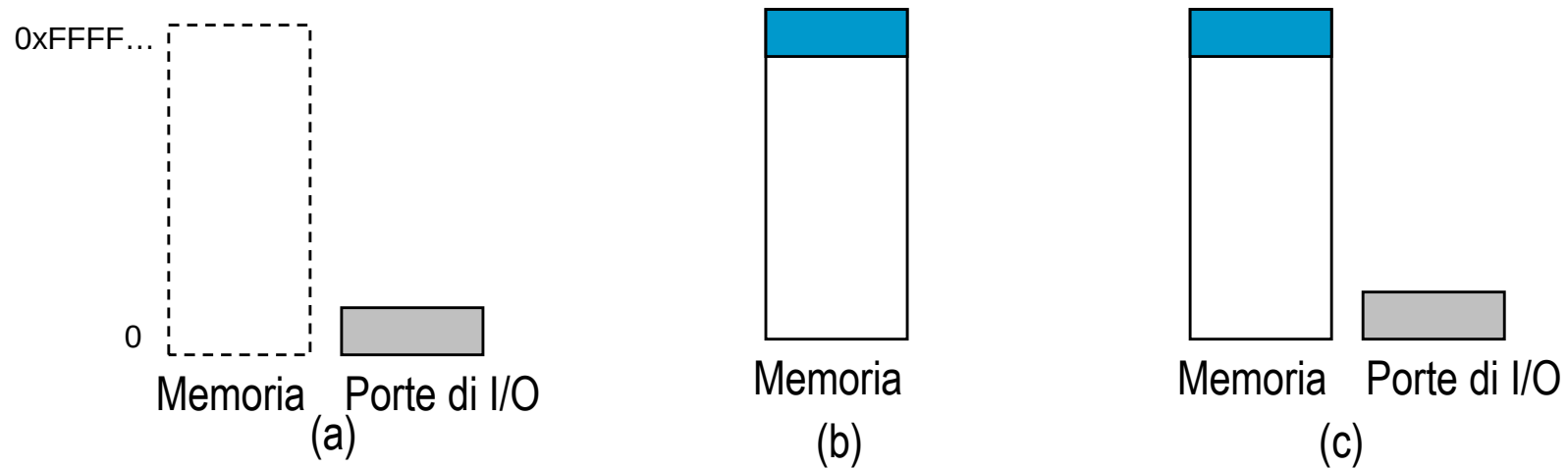
Nota: nel seguito si parlara' di "dispositivo" sottintendendo che ci si riferisce sempre al "controller del dispositivo"

Visione elettrica e logica del dispositivo



Metodi di comunicazione fra processore e dispositivo

- (a) Spazi separati di I/O e memoria
- (b) Memory-mapped I/O
- (c) Soluzione ibrida



Istruzioni per la comunicazione fra processore e disp.

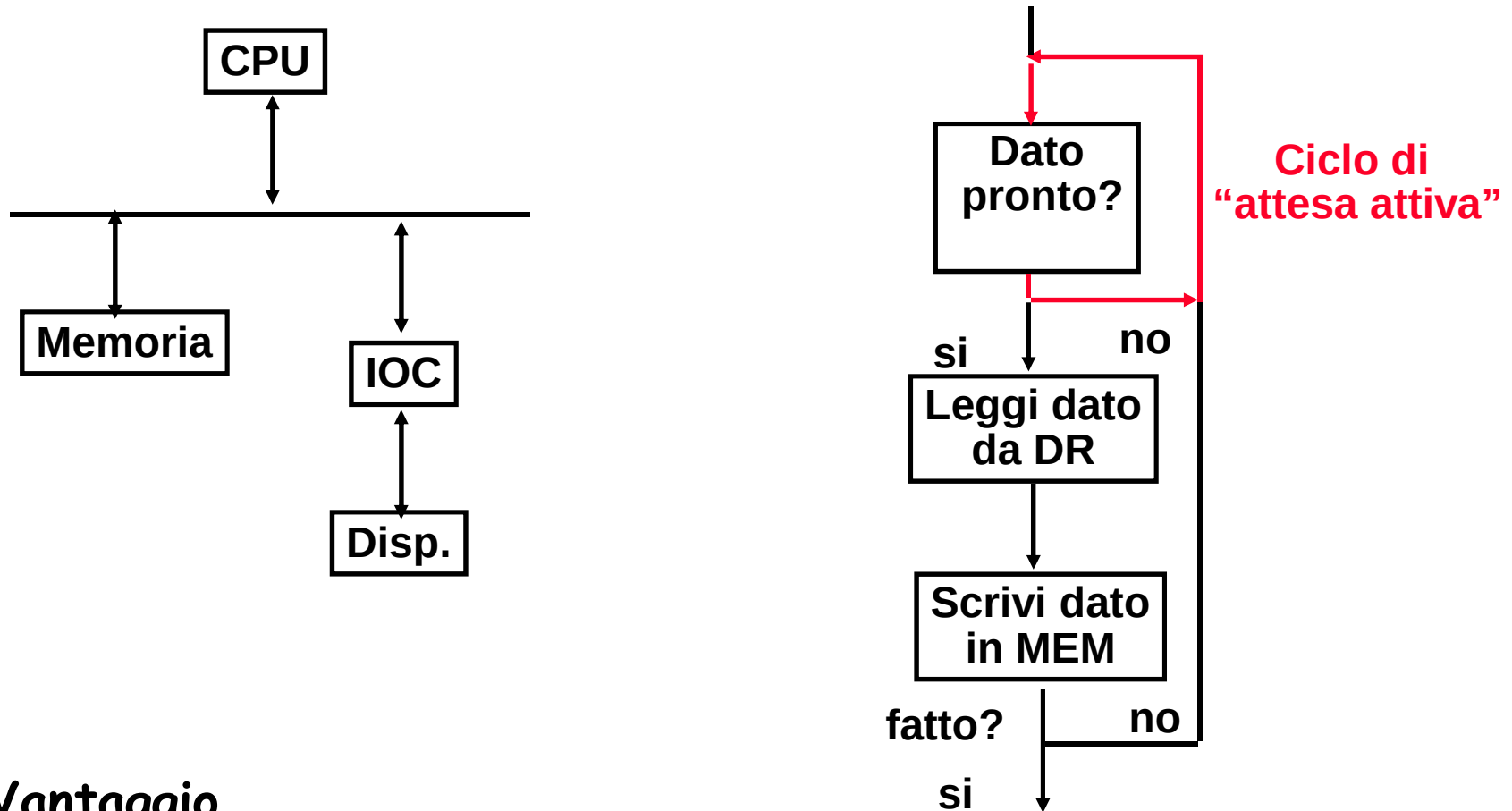
- Se si usano i metodi (a) o (c) per scambiare dati fra processore e memoria si rendono necessarie
 - Istruzioni speciali di I/O (es. Architetture x86)
 - L'istruzione speciale specifica il numero del dispositivo
 - Questo puo' essere trasmesso "come un indirizzo" sul bus di I/O
 - L'istruzione speciale specifica il comando da dare al dispositivo
 - Questo puo' essere trasmesso "come un dato" sul bus di I/O
- Se si usa il metodo (b) non e' necessario introdurre nuove istruzioni per comunicare con l'I/O
 - E' 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 perche' risiederanno in una zona riservata allo spazio di indirizzamento del kernel
 - Questa tecnica e' 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:**
 - Ogniqualevolta 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 e' notificato che l'operazione e' finita con un interrupt dal DMA/IOP verso il processore

Polling

Protocollo di Polling (o di I/O Programmato)

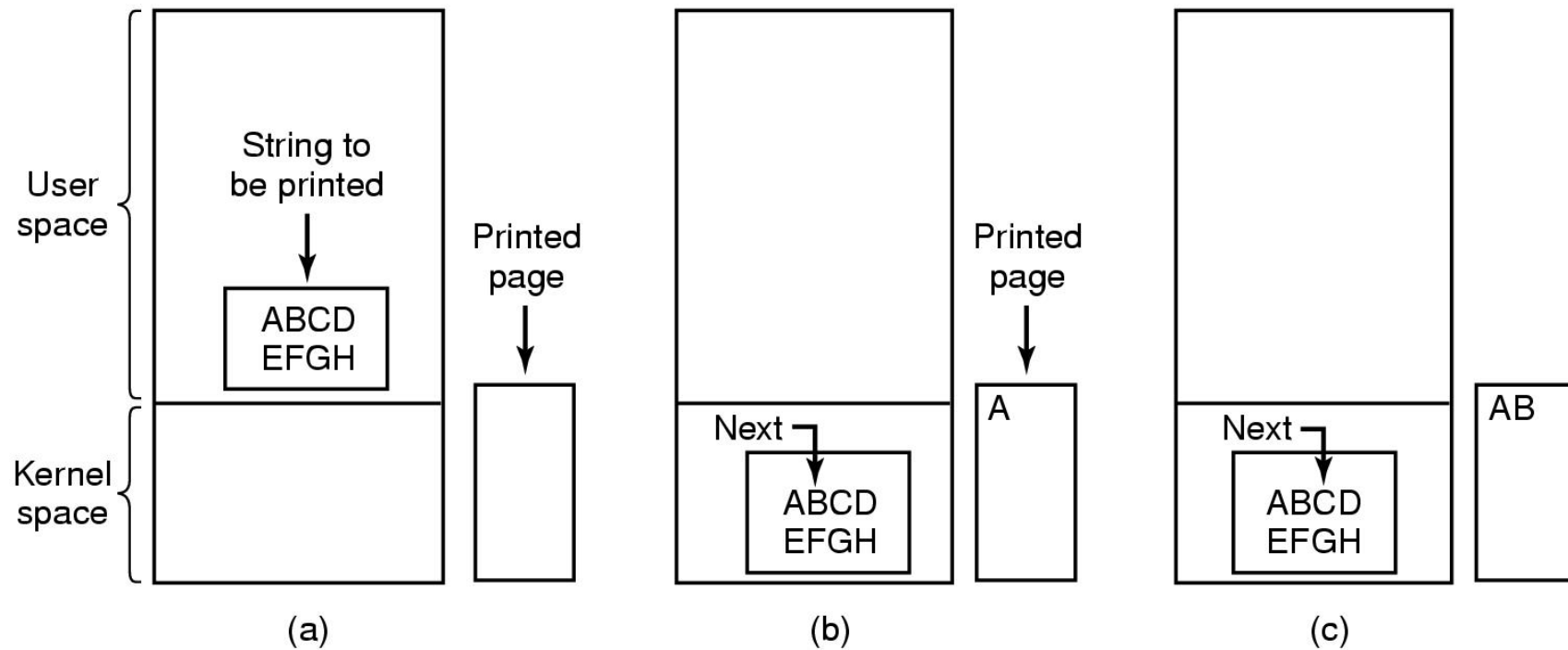


- **Vantaggio**
 - **Semplice:** il processore ha il controllo totale e fa tutto il lavoro
- **Svantaggio**
 - **La gestione del polling puo' consumare molto tempo del processore**
 - **Trucco:** distribuire i controlli di I/O fra codice che effettua altre operazioni

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 puo' attendere oppure ritornare a vedere piu' tardi

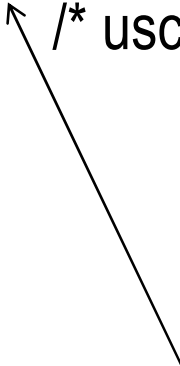
Esempio di Polling: stampa di una stringa



Stampa di una stringa con Polling

```
copy_from_user(buffer, p, count);      /* p e' 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();
```

Notare il punto e virgola !



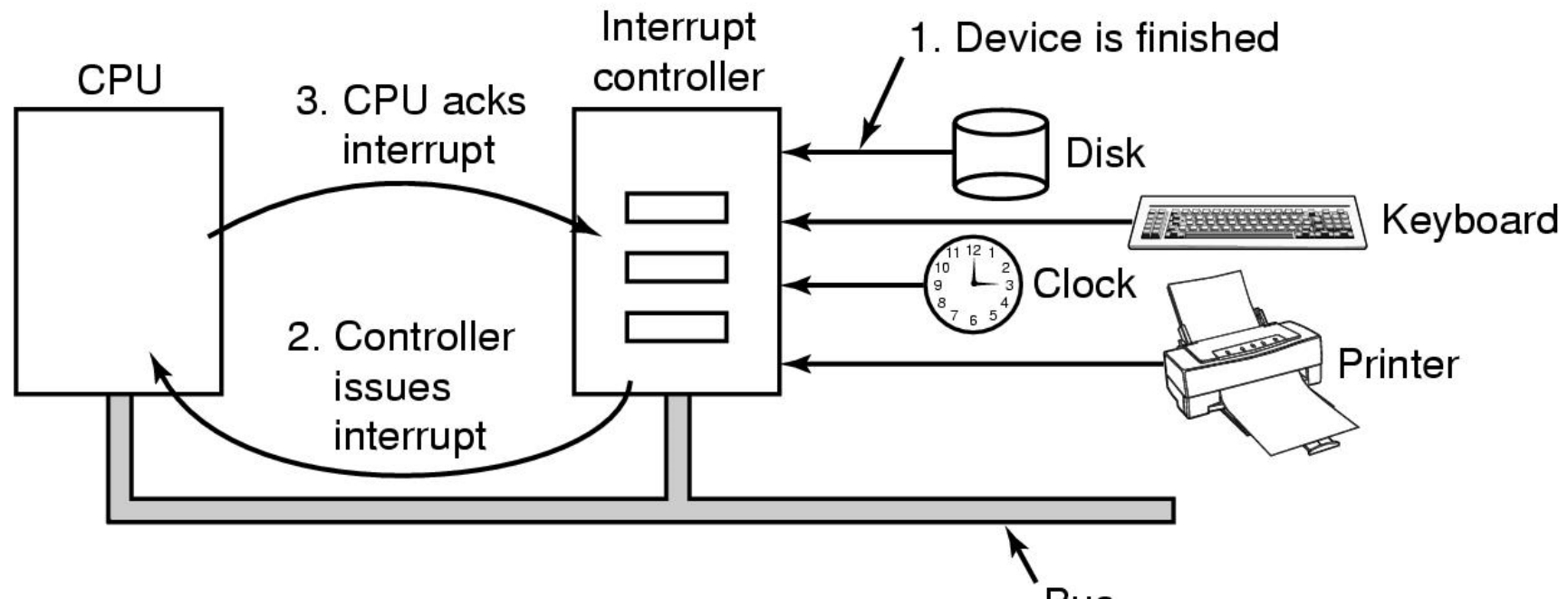
Interrupt

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

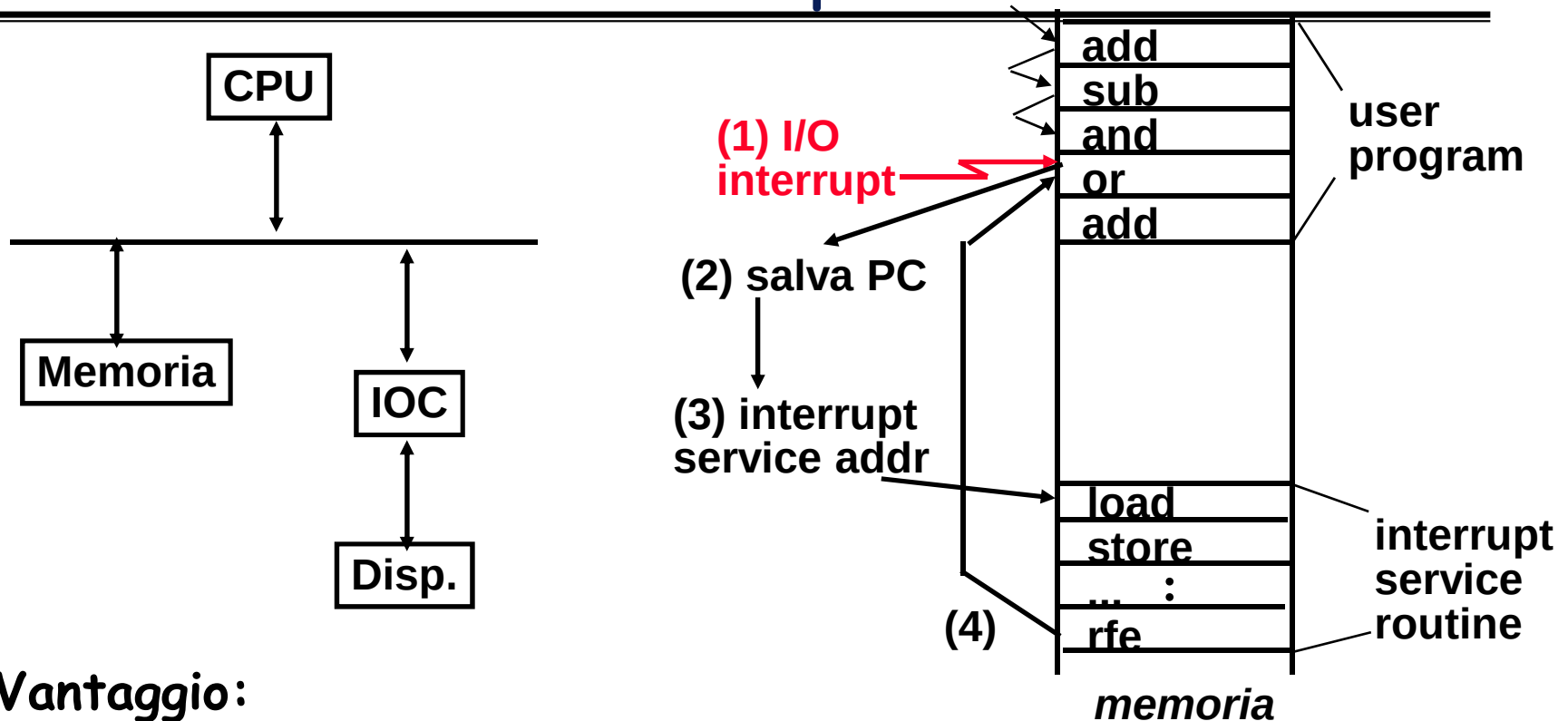
- La CPU dà un comando di lettura
- Il controller di I/O prende un dato dal dispositivo mentre la CPU si occupa di altro
- Il controller di I/O interrompe la CPU
- La CPU richiede i dati
- Il controller di I/O trasferisce i dati al processore



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



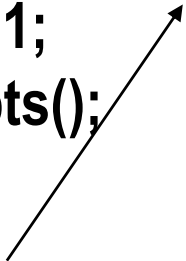
- **Vantaggio:**
 - Il programma utente e' bloccato solo durante il tempo effettivo per trasferire il dato dal dispositivo alla memoria
- **Svantaggio, e' necessario hardware dedicato per**
 - Generare l'interrupt (lato controller di I/O)
 - Accorgersi dell'interrupt (lato CPU)
 - Salvare lo stato della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



Inizialmente avro' k=1

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();
```

Interrupt

- Per la CPU, l'interrupt e' esattamente come un'eccezione salvo che
 - L'interrupt e' un evento asincrono
 - Non e' provocato da istruzioni
 - Non impedisce il completamento di alcuna istruzione
 - Ad esso fa seguito un trasferimento di informazioni
- L'implementazione dell'interrupt e' piu' complicata di quella di un'eccezione, perche'
 - Deve consentire di identificare il dispositivo che ha generato l'interrupt
 - Le richieste di interruzione possono avere priorita' diversa
 - Le richieste di interruzione provenienti da interrupt diversi possono sovrapporsi (interrupt multipli)

Identificare il dispositivo che ha generato l'interrupt

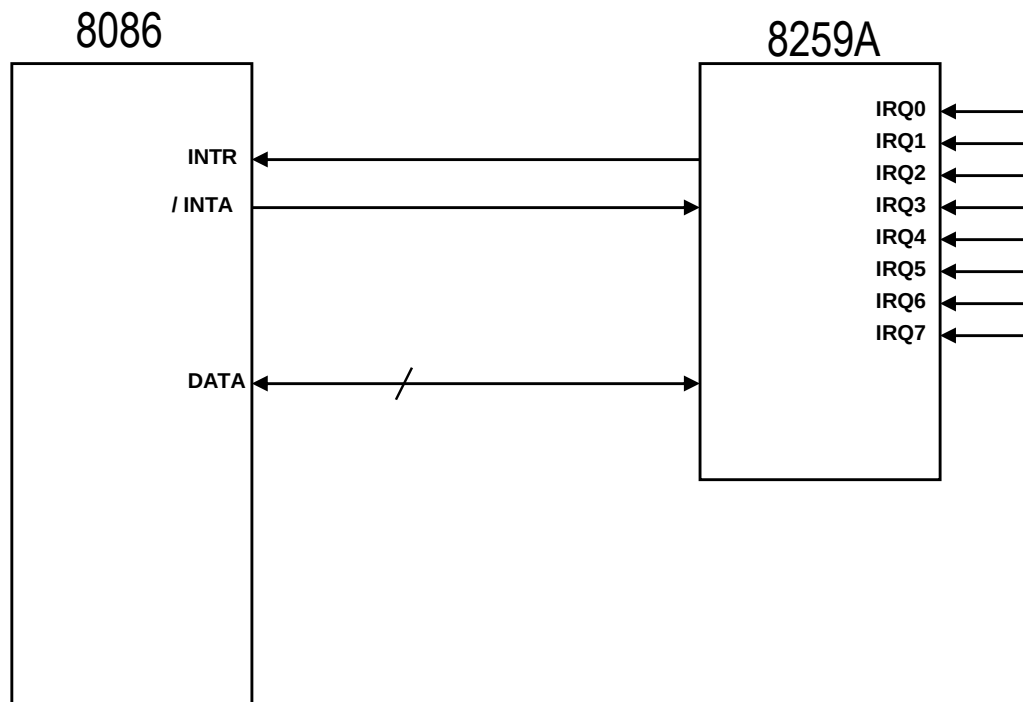
- Differenti linee per ogni controller
 - Usato nei PC
 - Svantaggio: limita il numero dei dispositivi
- Software poll
 - La CPU chiede ad ognuno dei controller se ha generato un interrupt
 - Lento
- Daisy Chain (Hardware poll)
 - Il segnale di Interrupt Acknowledge viene inviato in daisy-chain
 - Il controller e' 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

Interrupt Multipli

- Ogni interrupt ha una priorita' associata
- Dispositivi a piu' alta priorita' possono interrompere anche quando si servono dispositivi a piu' bassa priorita'
- Nel caso di bus mastering l'ulteriore interrupt puo' venire solo dall'attuale master

Esempio - PC

- 80x86 ha una sola linea di interrupt
- I sistemi basati su 8086 usano un "controller di interrupt" chiamato 8259A
- Il chip 8259A ha 8 linee di interrupt

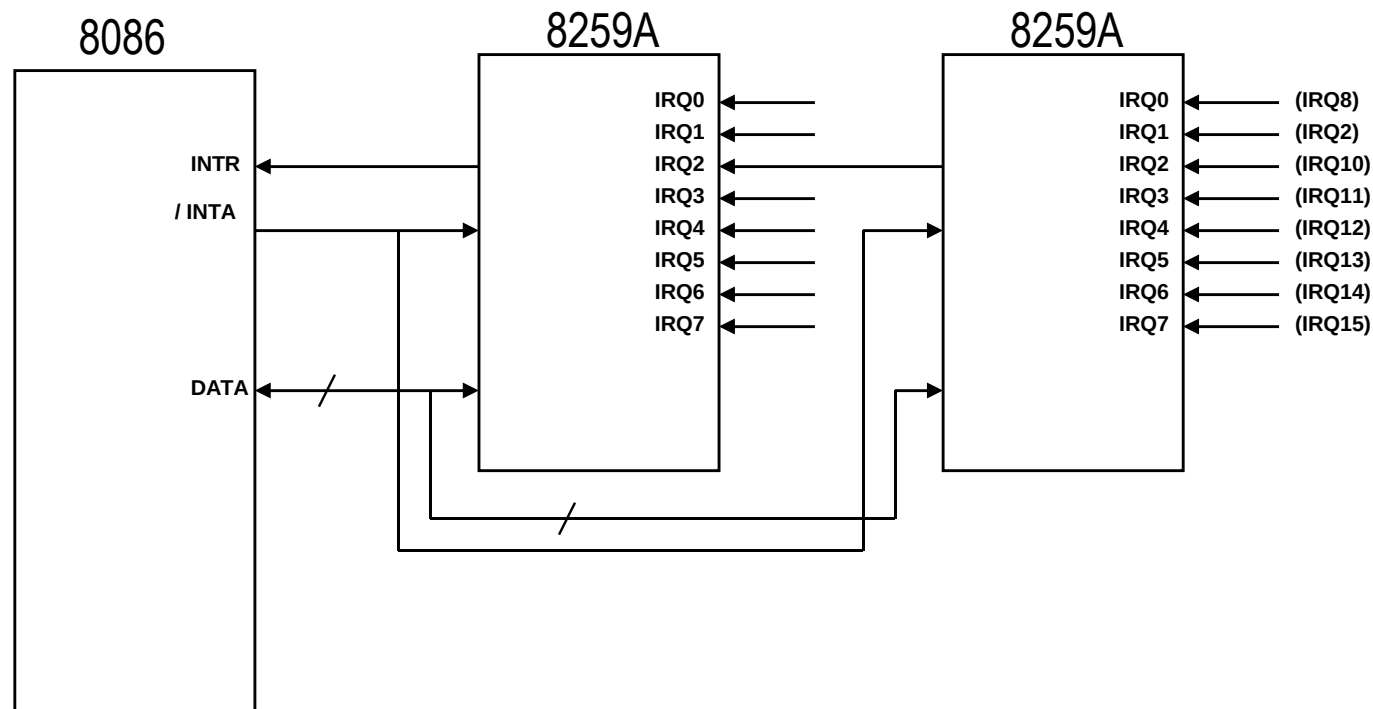


Sequenza di Eventi

- L' 8259A accetta interrupt
- L' 8259A determina la priorit 
- L' 8259A avvisa l' 8086 (attiva la linea INTR)
- La CPU invia una conferma (linea INTA)
- L' 8259A mette sul bus dati della CPU il "vettore" cioe' un numero che identifica la sorgente di interrupt
- La CPU serve l'interrupt mandando in esecuzione la relativa routine di servizio

Cascading

- Nei PC si usarono due 8259A in cascata
- Fornisce 15 linee di interrupt
- Per mantenere la compatibilit  il vecchio interrupt 2 viene rimappato sull'interrupt 9
- Dai 486 in poi sono entrambi incorporati nella CPU



DMA

Direct Memory Access

Direct Memory Access

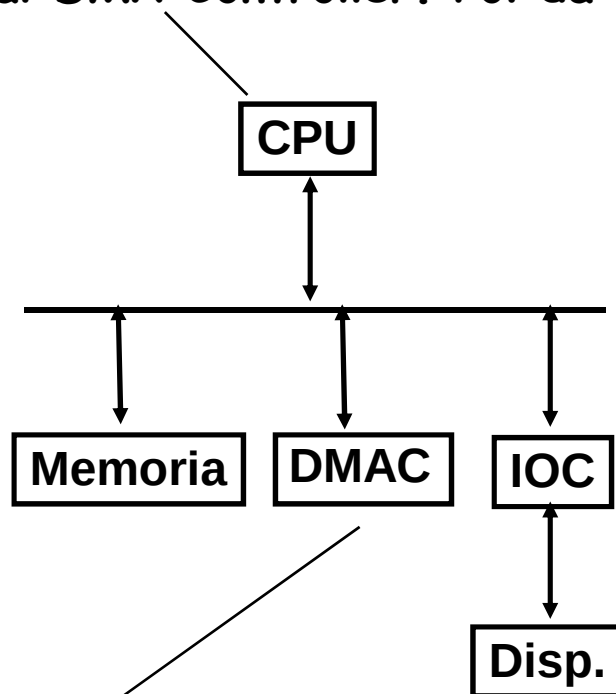
- Interrupt frequenti e polling richiedono l'assistenza del processore
 - La velocita' di trasferimento e' limitata
 - La CPU e' impegnata in troppe attivita'

...il DMA e' la risposta

- Esterno alla CPU
- Agisce come master sul bus
- Trasferisce blocchi di dati da/verso la memoria senza l'intervento della CPU

Funzionamento del DMA

La CPU invia un indirizzo iniziale, direzione e lunghezza del trasferimento al DMA Controller. Poi da' lo "start"...

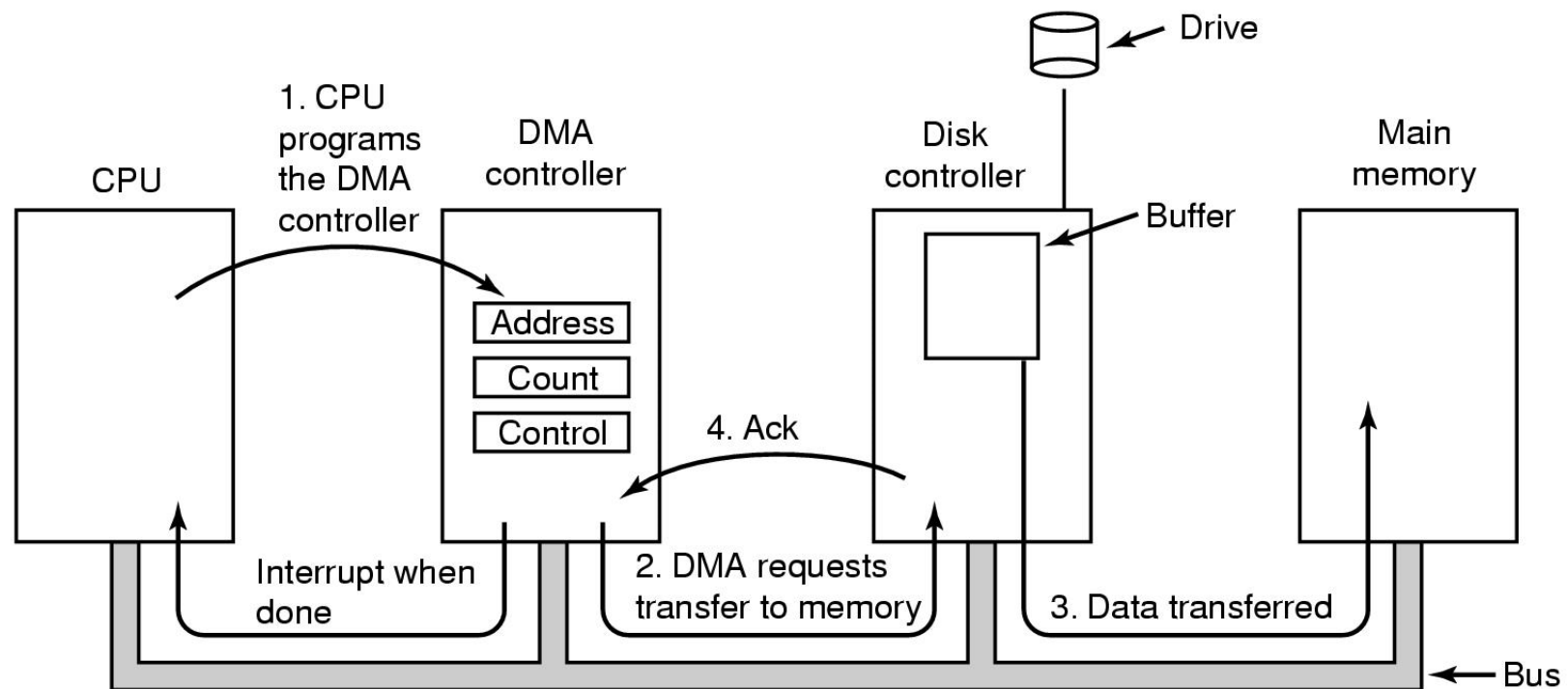


Il DMA Controller genera tutti i necessari segnali per indirizzare la periferica e la memoria e per gestire l'handshake

DMA - dettaglio

- **La CPU dice al DMA Controller:**
 - Indirizzo iniziale della zona di memoria coinvolta
 - Indirizzo del controller del dispositivo
 - Direzione (Read/Write)
 - Quantita' di dati da trasferire
- **LA CPU puo' svolgere altro lavoro**
- **Il DMA Controller si occupa del trasferimento**
- **Il DMA Controller manda un interrupt quando ha finito**

DMA -dettaglio (2)



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();
```

Routine di servizio
dell'interrupt

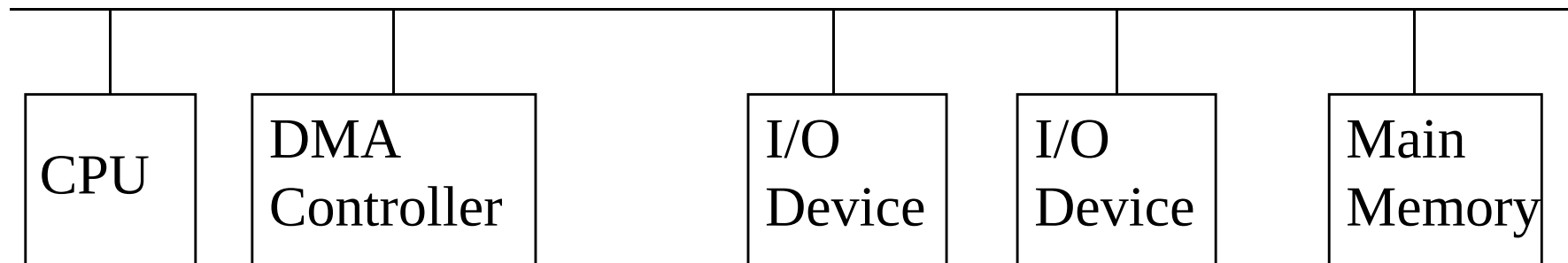
```
acknowledge_interrupt();  
unblock_user();  
return_from_interrupt();
```

DMA: Cycle Stealing

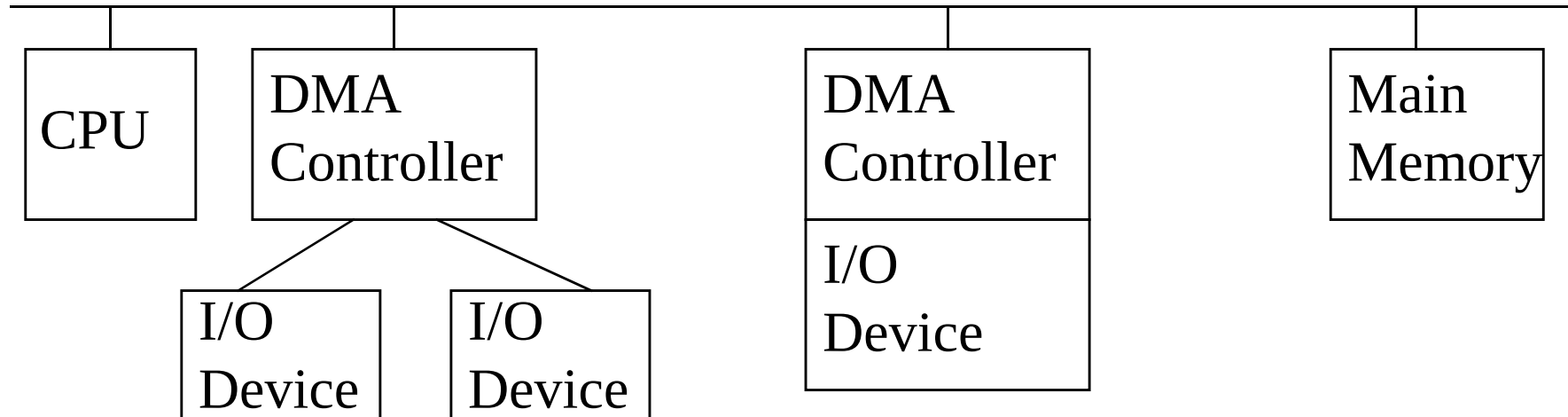
- Il DMA controller occupa il bus per un ciclo (di bus)
- Trasferisce una word di dati
- Non e' un interrupt
 - La CPU non deve fare salvataggio/ripristino stato !
- La CPU eventualmente attende di piu' quando dovra' fare accesso al bus
 - Es. Per fare fetch/read/write
 - La CPU attende un poco ma sicuramente meno di quanto dovrebbe attendere per trasferire il dato da sola

DMA: Configurazioni (1)

- Singolo Bus, DMA controller separato
- Ogni trasferimento usa il bus due volte
 - Da dispositivo a DMA e da DMA a memoria
- LA CPU trova il bus occupato per piu' tempo

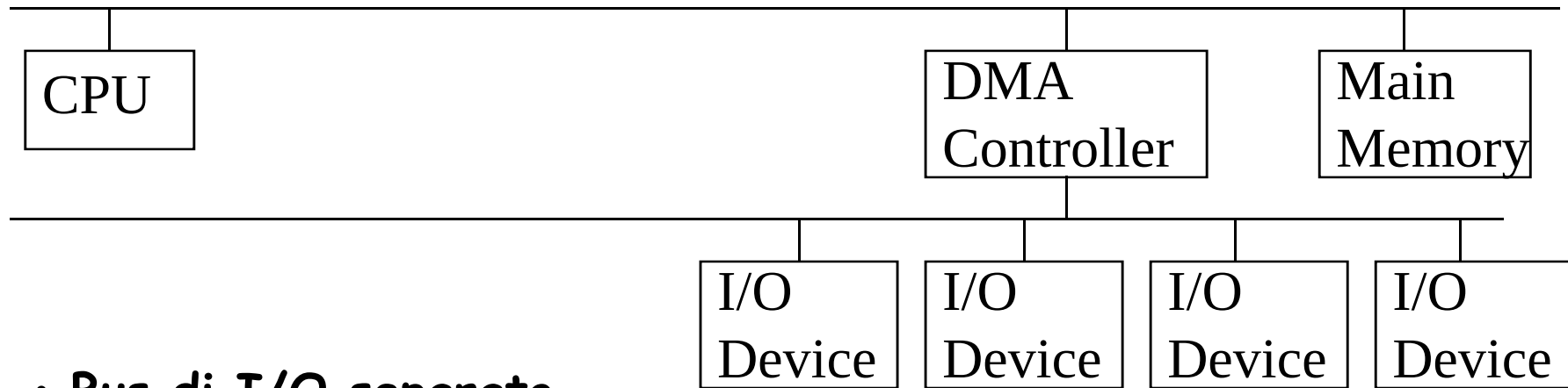


DMA: Configurazioni (2)



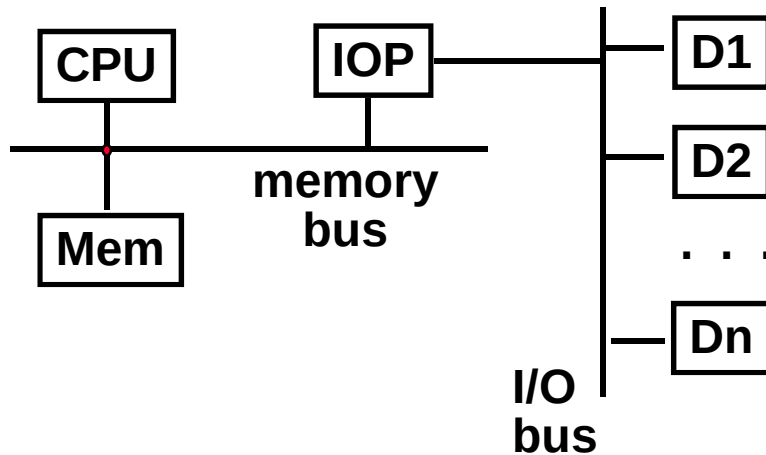
- Singolo Bus, DMA controller integrato
- Il controller puo' gestire piu' di un dispositivo
- Ogni trasferimento usa il bus una sola volta
 - dsa DMA a memory
- La CPU dimezza le volte in cui trova occupato il bus

DMA: Configurazioni (3)



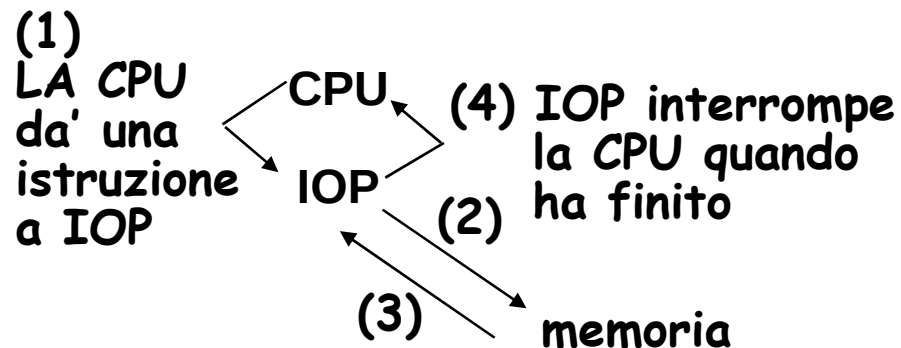
- **Bus di I/O separato**
- **Il bus supporta tutti i dispositivi abilitati al DMA**
- **Ogni trasferimento usa il bus una sola volta**
 - DMA to memory
- **La CPU dimezza le volte in cui trova occupato il bus rispetto al primo caso**

Delegare le operazioni di I/O: IOP (I/O Processor)



I trasferimenti da/verso la memoria sono controllati direttamente dall'IOP

IOP effettua cycle stealing



CONTROLLER

DMA

DMA Controller (DMAC)

- Periferica dedicata che può controllare il bus:
 - Inizializza e controlla gli accessi al bus
 - tra un dispositivo di I/O e la memoria
 - tra due aree di memoria
 - Agisce come una implementazione hardware della routine di gestione dell'interrupt di buffer-pieno o buffer-vuoto
- Tre tipi di controllers DMA:
 - 1D, un singolo registro per gli indirizzi
 - 2D, due registri per l'indirizzamento
 - 3D, tre o più registri per l'indirizzamento

Struttura di un Controller DMA Generico (1)

- **Costituito da:**

- **Address Generator**

- Genera gli indirizzi di memoria o di periferica coinvolti nei trasferimenti. Costituito da un registro base e un contatore auto-incrementante

- **Address Bus**

- Dove finiscono gli indirizzi generati per accedere ad una specifica locazione di memoria o della periferica

- **Data Bus**

- Utilizzato per trasferire dati dal DMA alla destinazione. Molte volte il trasferimento avviene direttamente dalla sorgente alla destinazione, con il controller DMA che solamente seleziona i due dispositivi

Controller DMA Generico (2)

- **Bus Requester**

Utilizzato per richiedere il bus alla CPU (necessario in una corretta configurazione multimaster)

- **Local Peripheral Control**

Permette al controller DMA di selezionare la periferica e gestirla (necessario per indirizzamento singolo o implicito)

- **Interrupt signals**

I controllers DMA possono interrompere la CPU appena il trasferimento è terminato o se si è verificato un errore

Controller DMA Generico: funzionamento (1)

1) Programmazione del controller:

- Definizione indirizzo base e dimensione del trasferimento
- Definizione comunicazione con il processore
 - Es. definizione delle condizioni su cui si generano interrupt
- Definizione comunicazione con le periferiche
 - Es. Mappare le periferiche sulle linee di richiesta di DMA e politica di arbitraggio per richieste simultanee al DMA
- Definizione del tipo di trasferimento dei dati
 - all-at-once , individuale od altro

Source Address	FF FF 01 04
Base Address	00 00 23 00
Count	00 00 00 10
Byte Transferred	00 00 00 00
Status	OK

Es. Registri
di controllo
del DMA

Controller DMA Generico: operazioni (2)

2) Inizio del trasferimento

- Assume che il controller sia stato configurato correttamente
- Il trasferimento ha inizio su una esplicita richiesta o verso il DMAC direttamente o attraverso il processore (attivato su interrupt)

3) Richiesta del bus

- Il DMAC invia una richiesta alla CPU che rilascia il bus (supportato nei processori piu' recenti)
- In sistemi senza gestione delle richieste su bus, il DMAC deve funzionare in "bus-stealing": il processore resta bloccato su un accesso al bus mentre il DMAC opera

4) Generazione dell'indirizzo

- Una volta che il controller ha il bus, deve attivare la locazione di memoria. Sono utilizzate diverse interfacce:
 - Non multiplexata
 - Multiplexata
- inoltre il DMAC fornisce altri segnali di controllo, come R/W, strobe per lavorare sul bus
- I piu' recenti DMAC sono abbastanza generici per poter funzionare con diverse famiglie di processori

Controller DMA Generico: operazioni (3)

5) Trasferimento dei dati

- Il trasferimento può avvenire o attraverso un buffer interno al controller o direttamente dalla periferica

6) Aggiornamento dell'indirizzo

- Una volta terminato il trasferimento viene calcolato il nuovo indirizzo per il trasferimento successivo e viene aggiornato il contatore dei trasferimenti

7) Aggiornamento del processore

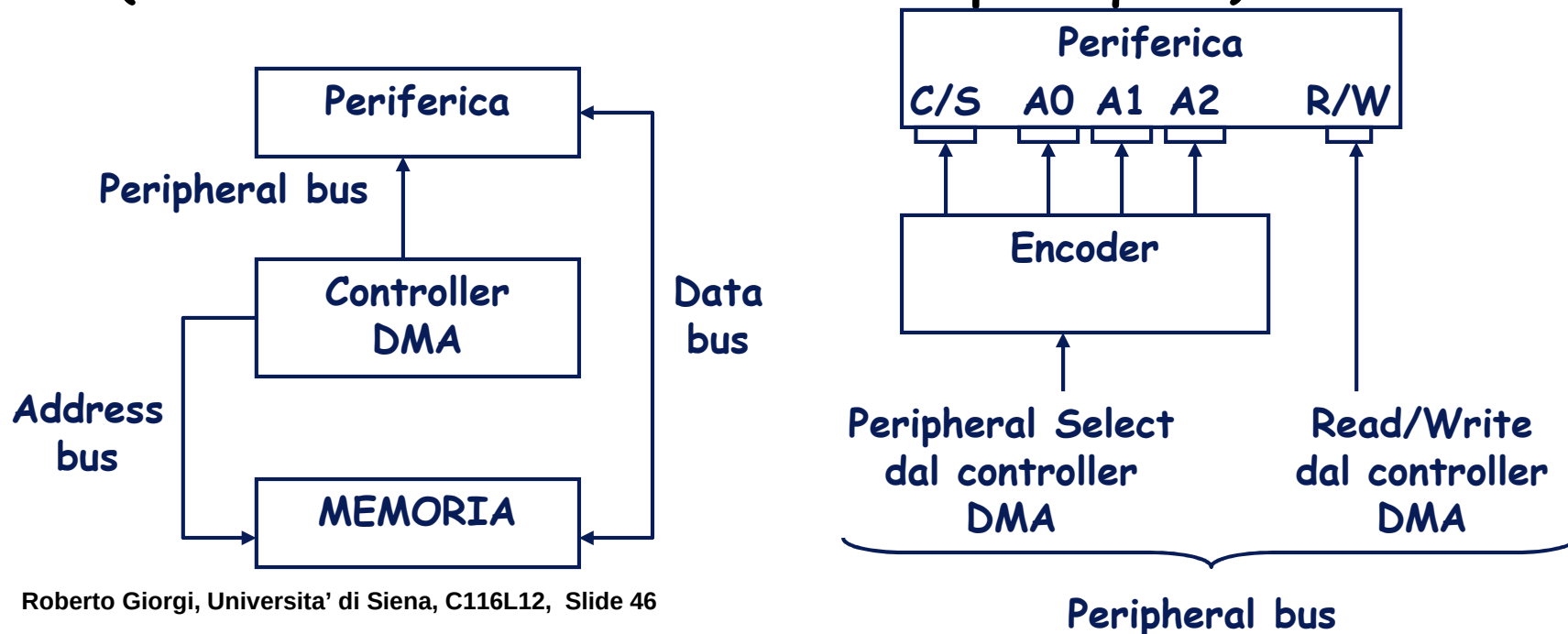
- Il controller notifica al processore l'avvenuto trasferimento di un blocco o un eventuale errore o altri eventi, inviando ad esso un interrupt

Modelli di controllers DMA

- Vari modi di funzionamento in base:
 - Trasferimento dei dati
 - Singolo
 - Doppio
 - Indirizzamento
 - singolo
 - complesso

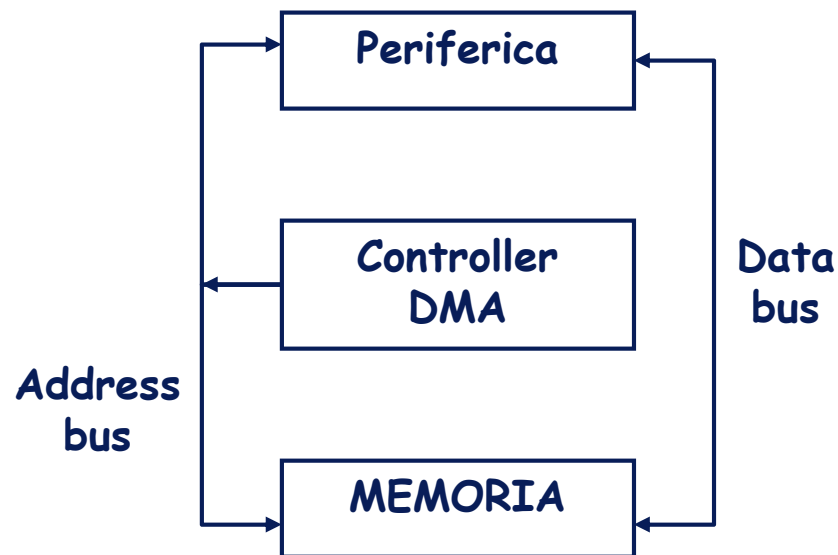
Trasferimento dati singolo (indirizzamento implicito)

- Il DMA utilizza l'address bus per indirizzare la locazione coinvolta nel ciclo di bus verso la memoria
- Utilizza il bus di periferica (chip select) per selezionare la periferica e rendere disponibile il suo bus dati
 - In conseguenza dell'attivazione del Chip Select puo' essere necessario un encoder per selezionare un dato indirizzo (implicito) della periferica
- Se e' la periferica ad iniziare, abbassa la linea di request (comune l'uso della linea di interrupt request)



Trasferimento dati doppio

- Si usano due indirizzi e due accessi per trasferire dati dalla periferica (o memoria) a un'altra locazione di memoria
 - Consuma 2 CICLI di bus
 - Utilizza un buffer temporaneo interno al controller

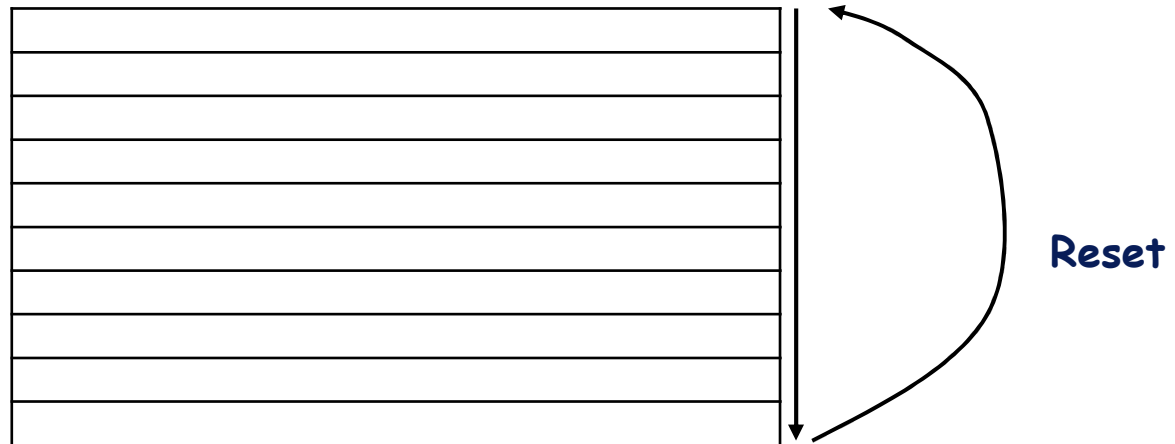


1D Model

- Usa un indirizzo base e un contatore per definire la sequenza di indirizzi da usare nei cicli DMA
- Svantaggi:
 - Una volta che il blocco di dati è stato trasmesso, l'indirizzo ed il contatore sono resettati potendo sovrascrivere accidentalmente dati precedenti
 - Diventa necessario un interrupt dal DMA al processore ogni volta che il contatore termina il conteggio per cambiare l'indirizzo base
- Utilita'
 - Si puo' implementare un buffer circolare con wrap around automatico

Locazione iniziale
di memoria

Locazione finale
di memoria

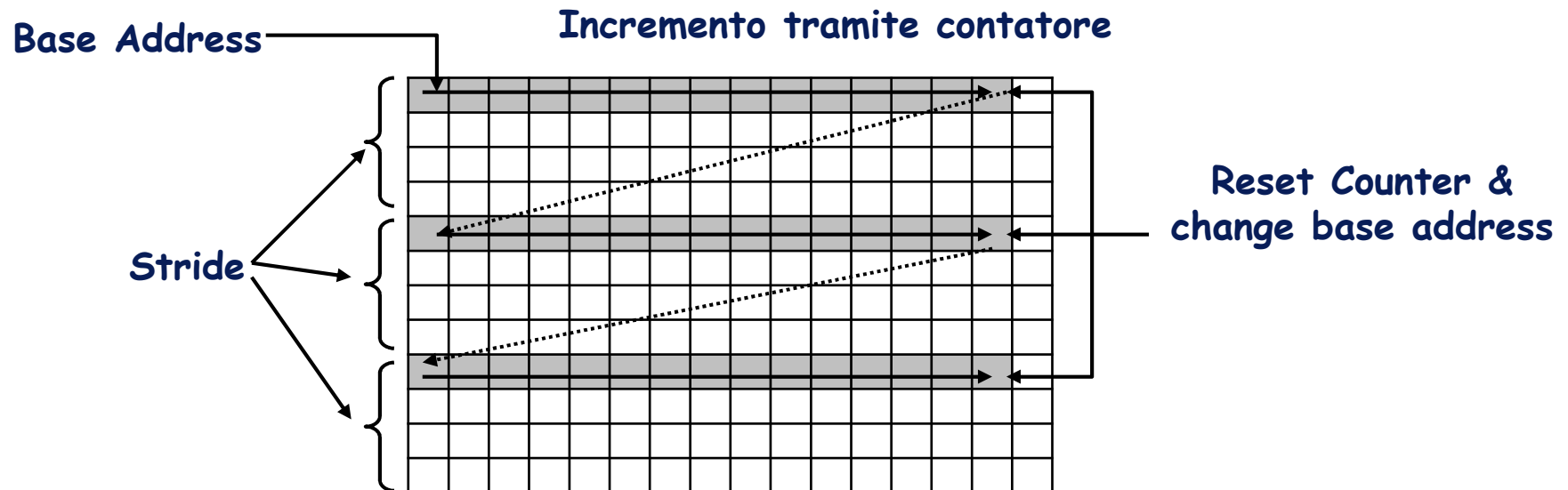


2D Model

- Es. Implementazione di protocolli di comunicazione basati sullo scambio di pacchetti
 - Devo spezzare grosse quantita' di dati in pacchetti e aggiungere gli header
- Invece di riportare l'indirizzo di base al valore originale, viene sommato un valore di "stride" alla fine di un trasferimento di un blocco
 - Permette al DMA di utilizzare blocchi di memoria non contigui
- Il registro contatore e' diviso in due:
 - Un registro per l'offset all'interno del blocco
 - Un registro per contare il numero totale di blocchi o i byte totali trasferiti

2D Model

- Posso facilmente spezzare un blocco in pacchetti pronti per l'inserimento degli header
- Successivamente si puo' usare la modalita' 1D per inviare i dati alla scheda di rete

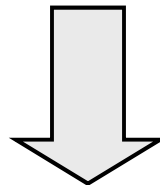


3D model

- Utilizza un meccanismo a "stride" come nel modello 2D
- Inoltre posso cambiare automaticamente lo stride per creare blocchi a indirizzi variabili
- E' possibile simulare tale modalità con un DMA 2D e una procedura software che riprogramma ogni volta il DMAC per cambiare lo stride

Problema non risolto

- Ogni DMA deve essere pre-programmato con un blocco di parametri per poter funzionare correttamente
- L'interfaccia hardware è comune per quasi tutti i differenti set di parametri
- Ogni periferica che necessita di un trasferimento deve chiedere alla CPU di programmare correttamente il DMA (interrupt)
- Spesso una certa periferica programma il DMAC con lo stesso set di parametri

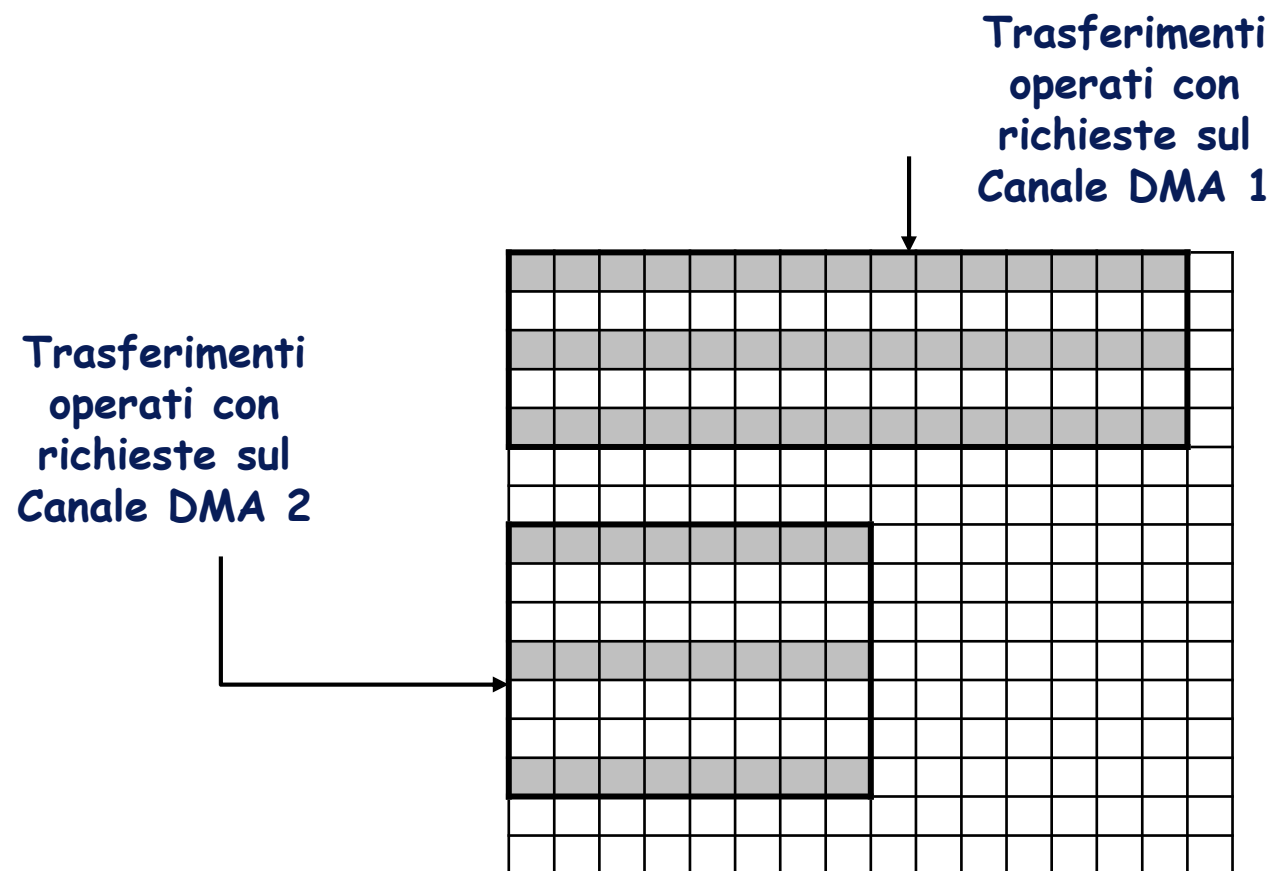


- Ancora un notevole carico di lavoro sulla CPU dato dalla gestione degli interrupt delle periferiche e dalla programmazione del DMA

Canali DMA (Blocchi di controllo)

- Ideati per ridurre l'overhead del processore
- Ad ogni periferica viene assegnata una linea esterna di request (canale DMA)
- Per ogni linea sono programmati una sola volta i parametri per quel tipo di trasferimento (blocchi di controllo)
- Quando una periferica richiede un trasferimento, asserisce il suo canale DMA e inizia il trasferimento in accordo con i parametri assegnati a quel canale
- Possibilità di condividere un singolo controller con più periferiche senza eccessivo overhead per il processore
- Nei PC tipicamente ci sono 4 canali DMA

Uso dei Canali DMA

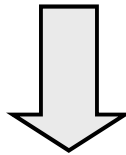


Concatenazione

- **Estensione dell'idea dei canali DMA: chaining (concatenazione)**
 - Canali collegati a catena in modo da generare pattern piu' complessi
 - Una volta terminato il lavoro di un canale, il controllo passa a quello successivo concatenato
 - Possibilità di schemi complessi di indirizzamento

Concorrenza di accesso ai canali DMA

- Cosa accade se arrivano richieste multiple di trasferimento al controller DMA?



- Necessità di un arbitraggio
 - Canali con priorità maggiore
 - Servizio dei canali con politica Round-Robin

Condivisione della banda del bus

- Il controller DMA compete con il processore per l'utilizzo del bus
- CPU senza cache (es. 80286, MC68000) utilizzano dall' 80% al 95% della larghezza di banda del bus
 - Ritardi nell'accesso al bus degradano enormemente le prestazioni e aumentano il tempo di latenza degli interrupt
- Per dispositivi con cache, l'interferenza del DMA sulle prestazioni è molto minore

Condivisione della banda del bus - 2

- Per consentire il giusto compromesso al progettista, molti DMA utilizzano differenti tipi di accesso al bus:
 - Trasferimento singolo
 - il bus ritorna alla CPU alla fine di ogni singolo trasferimento
 - Perdo tempo per le negoziazioni del bus e per iniziare e chiudere le transazioni di DMA
 - Trasferimento a blocchi
 - il bus è restituito alla CPU dopo il trasferimento di un blocco
 - Puo' essere molto lungo, ma poi termina
 - Il DMAC usa efficacemente il bus
 - Trasferimento su domanda
 - il bus viene trattenuto dal DMA per tutto il tempo che la periferica richiede
 - In teoria anche per un tempo lunghissimo!
 - Il DMAC puo' usare il bus in maniera non ottimale (lasciando "buchi"), ma "se e' onesto" funziona meglio del trasferimento a blocchi

INTEL 8237 (1)

- Il più comune controller DMA, utilizzato su tutti i PC IBM compatibili
- Oggi integrato sul chipset della scheda madre
- Supporta 4 modalità di trasferimento:
 - Trasferimento singolo
 - Trasferimento a blocchi
 - Modalità a domanda
 - Modalità a cascata - speciale modalità che gestisce più controllers 8237 a cascata per consentire più canali DMA

INTEL 8237 (2)

- Trasferimento dati
 - Da periferica a memoria
 - Da memoria a memoria (unendo due canali, ma non e' usato nei PC)
- Verify Transfer Mode :
 - Speciale modalità di trasferimento dati
 - Usato nei PC per generare indirizzi dummy per il refresh della memoria DRAM
 - Guidata da un interrupt (ogni 15 μ s) derivato da un canale del timer 8253
- Gestione interna della concorrenza:
 - Schema a priorità fissa
 - Schema a priorità variabile

Motorola MC68300

- Processori MC68000/MC68020 con controller DMA integrato sul chip
- Architettura:
 - Due canali DMA completamente programmabili
 - Velocità di trasferimento dati
 - 12.5 Mbytes/s in dual address mode a 25 MHz
 - 50.0 Mbytes/s in single address mode a 25 MHz
- Data la sua integrazione sulla CPU, può gestire trasferimenti tra memoria (o periferiche) interna ed esterna
 - Per i trasferimenti interni è possibile specificare la quantità di banda da occupare (25, 50, 75 o 100%)
 - Per i cicli esterni due modalità di trasferimento
 - burst
 - Single
- I registri sorgenti e di destinazione possono essere programmati indipendentemente per rimanere costanti o incrementare

Uso di una CPU separata con firmware

- Utilizzata quando non è disponibile un controller DMA
- LA "CPU separata" richiede una propria memoria e un programma che la faccia funzionare senza creare occupazione del bus della memoria (per la programmazione)
- La CPU DMA istruisce su come effettuare trasferimenti DMA
- Vantaggio di questa tecnica:
 - possibilità di essere programmata con software di alto livello per processare e trasferire i dati
- Molti dei processori usati nei sistemi embedded ricadono in questa categoria

Lezione 13

Esempi di dispositivi di I/O

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

- Timer
- Seriale

Timer Intel-8254

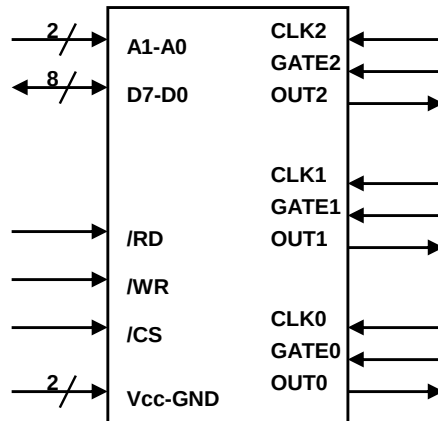
- A cosa puo' servire:
 - Generare ritardi programmabili in maniera accurata
 - Real time clock
 - Conteggio eventi
 - Generatore di frequenza programmabile
 - Generatore di onda quadra programmabile
 - Generatore di "impulso digitale"
 - Moltiplicatore binario di frequenza
 - Generatore di forma d'onda complessa
 - Controllo di motori
- E' un buon esempio per mostrare la complessita' interna di un'interfaccia di I/O

Timer Intel-8254

- **Visione Funzionale del chip:**
 - 3 contatori indipendenti a 16-bit
 - 6 modalita' di conteggio
- **Ogni timer/counter supporta sei modi operativi:**
 - MODO 0: Interrupt al termine del conteggio**
 - MODO 1: Programmable one shot**
 - MODO 2: Rate generator**
 - MODO 3: Square wave rate generator**
 - MODO 4: Software triggered strobe**
 - MODO 5: Hardware triggered strobe**

Timer Intel-8254

- **Visione Elettrica del chip:**



- **A1-A0:** indirizzamento
- **D7-D0:** scambio dati su 8 bit
- **/RD, /WR, /CS:** Read, Write, Chip-Select
- **Vcc-GND:** alimentazione

- **CLKj:** clock del timer j
- **GATEj:** trigger hardware o congelamento conteggio
- **OUTj:** forma d'onda prodotta dal timer j

Timer Intel-8254 (2)

- Visione logica del chip (tutti registri a 8 bit):

A1-A0 /RD /WB

00	1	0	CRO		Counter Register 0 Scrivendo in questo registro fisso la Time Constant 0
01	1	0	CR1		Counter Register 1 Scrivendo in questo registro fisso la Time Constant 1
10	1	0	CR2		Counter Register 2 Scrivendo in questo registro fisso la Time Constant 2
11	1	0	CWR		Control Word Register Scrivendo in questo registro fisso il funzionamento di un dato contatore oppure do' comandi particolari

Nota: la Time Constant e' a 16 bit. CRj funziona come un "bus" a 8 bit verso un registro interno a 16 bit che contiene tale Time Constant

Programmazione del timer

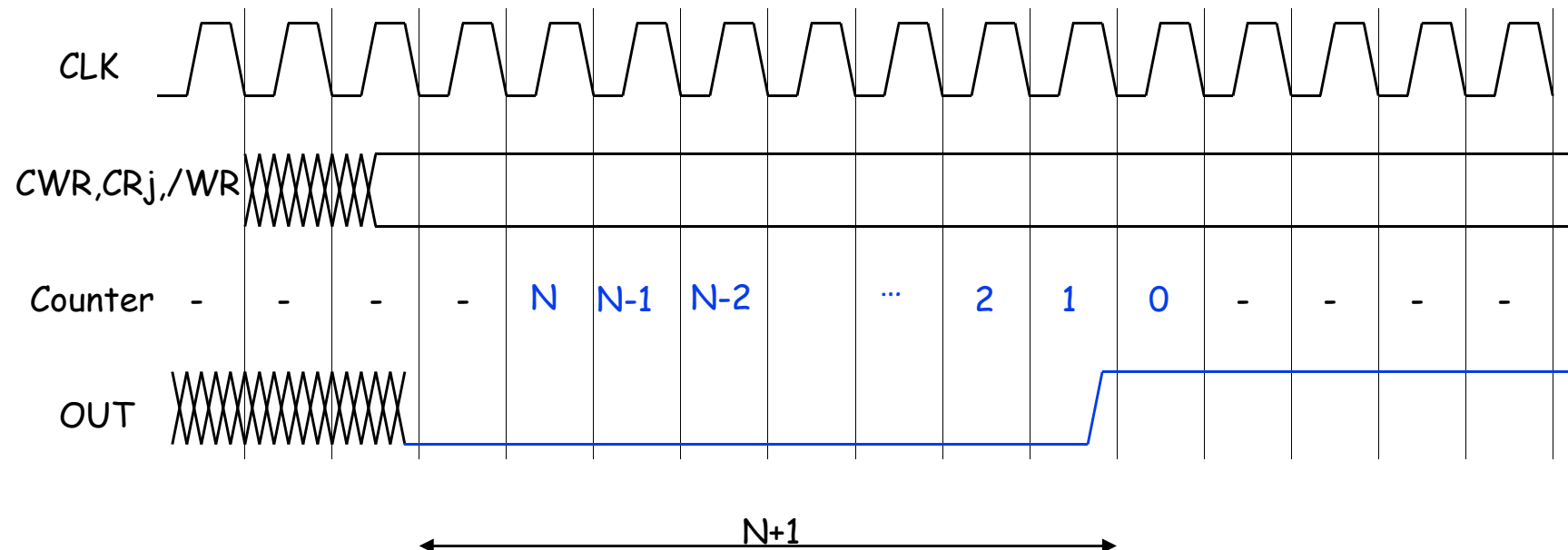
- 1) Scrivere in CWR per **selezionare uno dei contatori** e il **modo**
- 2) Scrivere in CRj la **costante di tempo**
- 3) **Abilitare** il conteggio con un segnale alto sul pin GATE
- 4) L'uscita **OUT** produrrà un segnale di uscita alla fine del (oppure durante il) conteggio a seconda del modo prescelto

Modalita' di lettura/scrittura

- Poiche' nei dispositivi di I/O si tende ad utilizzare un numero limitato di indirizzi (nel nostro caso 2) si usa un registro per dare i comandi (CWR) e gli altri registri per leggere o scrivere dati (CR0,CR1,CR2)
- I contatori sono a 16 bit ma posso scrivere solo 8 bit per volta
 - Devo perciò specificare, nelle operazioni di lettura/scrittura, se sto lavorando:
 - Solo sul byte meno significativo (LSB)
 - Solo sul byte più significativo (MSB)
 - Prima sul byte meno significativo e poi su quello più significativo (LSB+MSB)

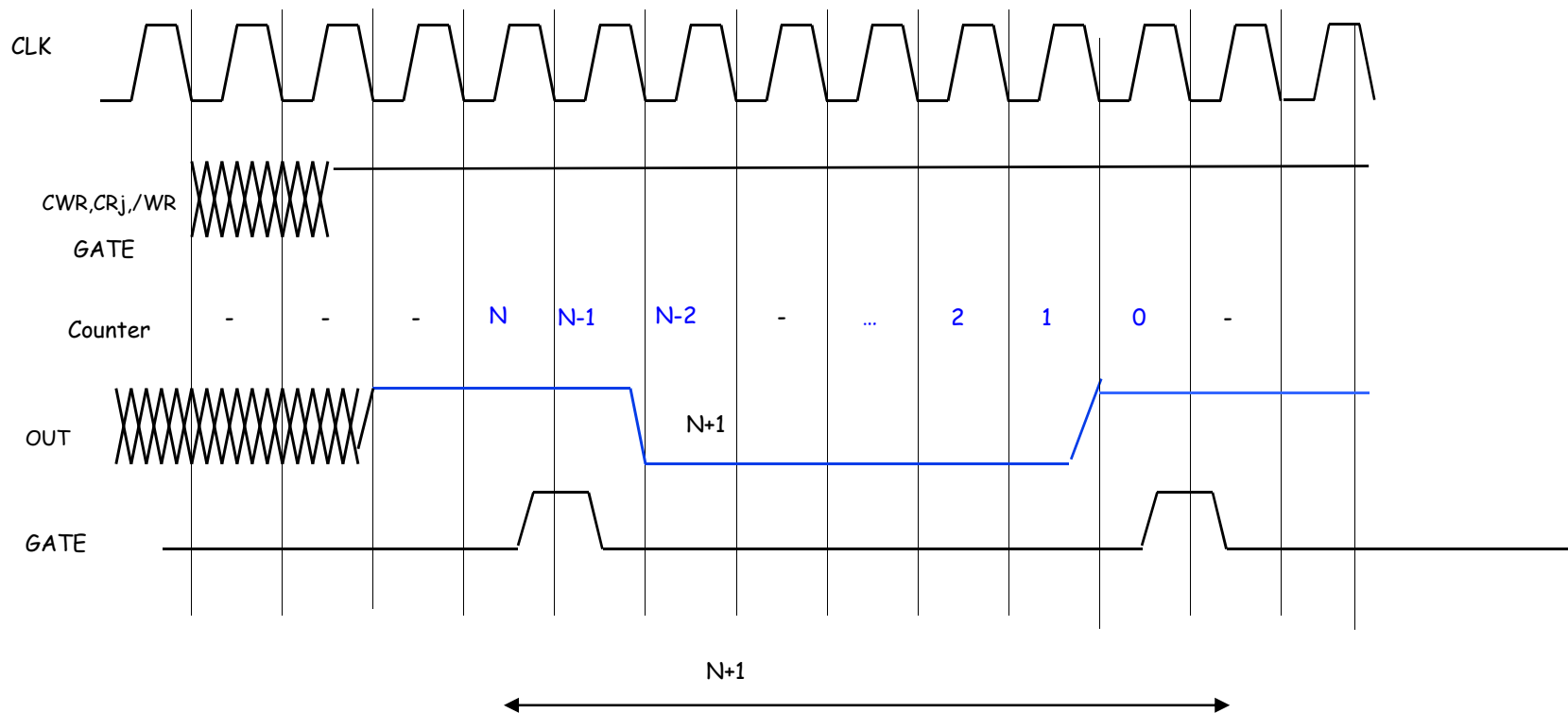
Modi di conteggio

- **Modo 0: interrupt al termine del conteggio**



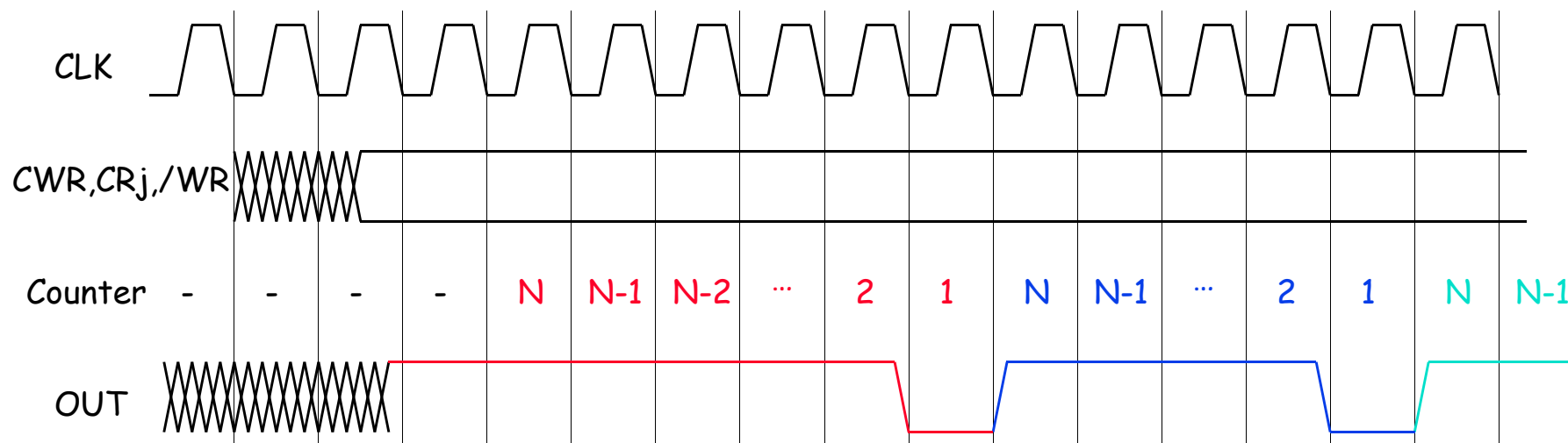
Modi di conteggio

- **Modo 1: one shot.** L'uscita è inizialmente alta e diventa bassa sull'impulso di clock che segue il trigger (pulsazione one shot), e vi rimane fino a quando il contatore non raggiunge lo zero.



Modi di conteggio

- **Modo 2: rate generator (divisore di frequenza)**



• Il duty cycle e' $(N-1)/N$

• Se f e' la frequenza del segnale di clock,
la forma d'onda d'uscita avra' frequenza f/N .

Lettura del conteggio (1)

- Ci sono tre maniere per leggere il valore di un contatore
 - 1) lettura semplice
 - 2) Counter Latch Command
 - 3) Read-Back Command
- La lettura semplice consiste nel leggere il valore di CR_j previo bloccaggio del conteggio tramite il segnale $GATE_j$
 - Se non si blocca il conteggio il valore letto puo' essere inconsistente

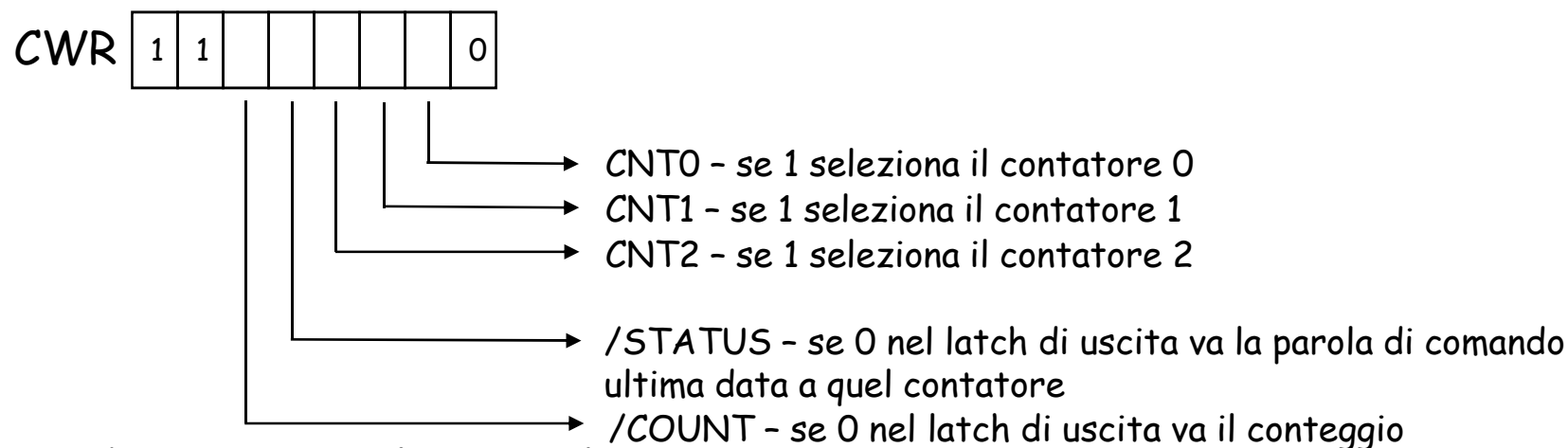
Lettura del conteggio (2)

- Counter Latch Command
 - Serve per leggere il valore corrente del contatore
 - Si realizza in questo modo
 - $CWR \leftarrow cc00 \text{ ' - - - - }$
(cc=0, 1 o 2 indica il contatore, '-' indica non specificato)
 - Lettura di 1 o 2 byte da CRcc
(a seconda dello schema di lettura/scrittura impostato)

Lettura del conteggio (3)

- Read-Back Command

1) Si realizza in questo modo: scrittura in CWR delle seguenti info



2) Successiva lettura di CRx

- Vantaggi:

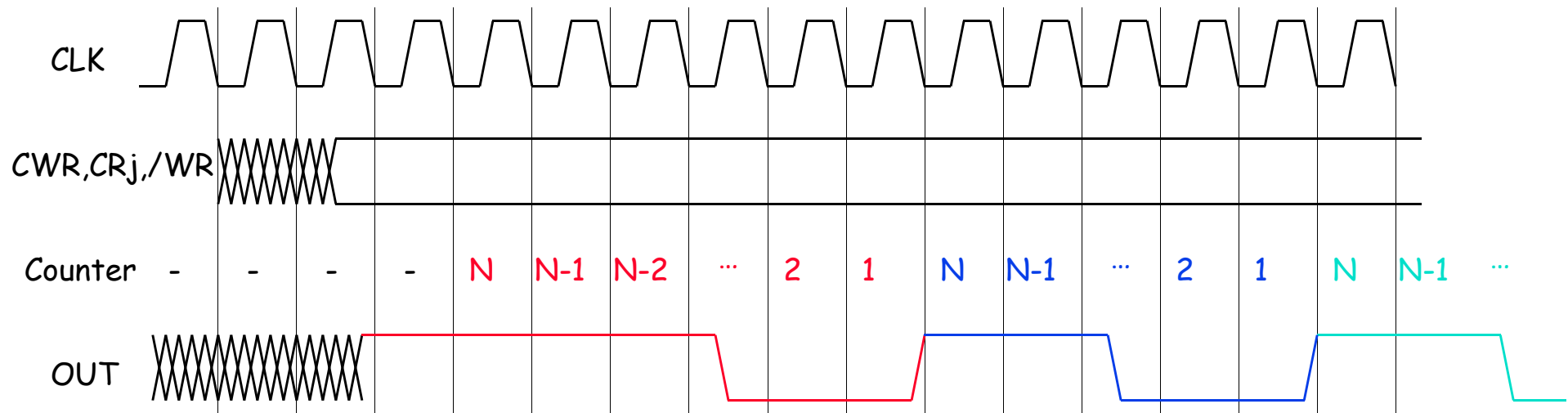
- Consente di leggere i conteggi di piu' contatori simultaneamente (caso di bit /COUNT attivo)
- Consente di leggere come sono stati programmati i 3 contatori (caso di bit /STATUS attivo)
 - In particolare, i 6 bit meno significativi (danno i valori precedentemente scritti in CWR per quel contatore

Ulteriori dettagli

- E' possibile
 - Programmare vari altri modi (es. modi 3,4,5 - v. slide successive)
 - Leggere lo stato relativo ad altre informazioni interne

Modi di conteggio

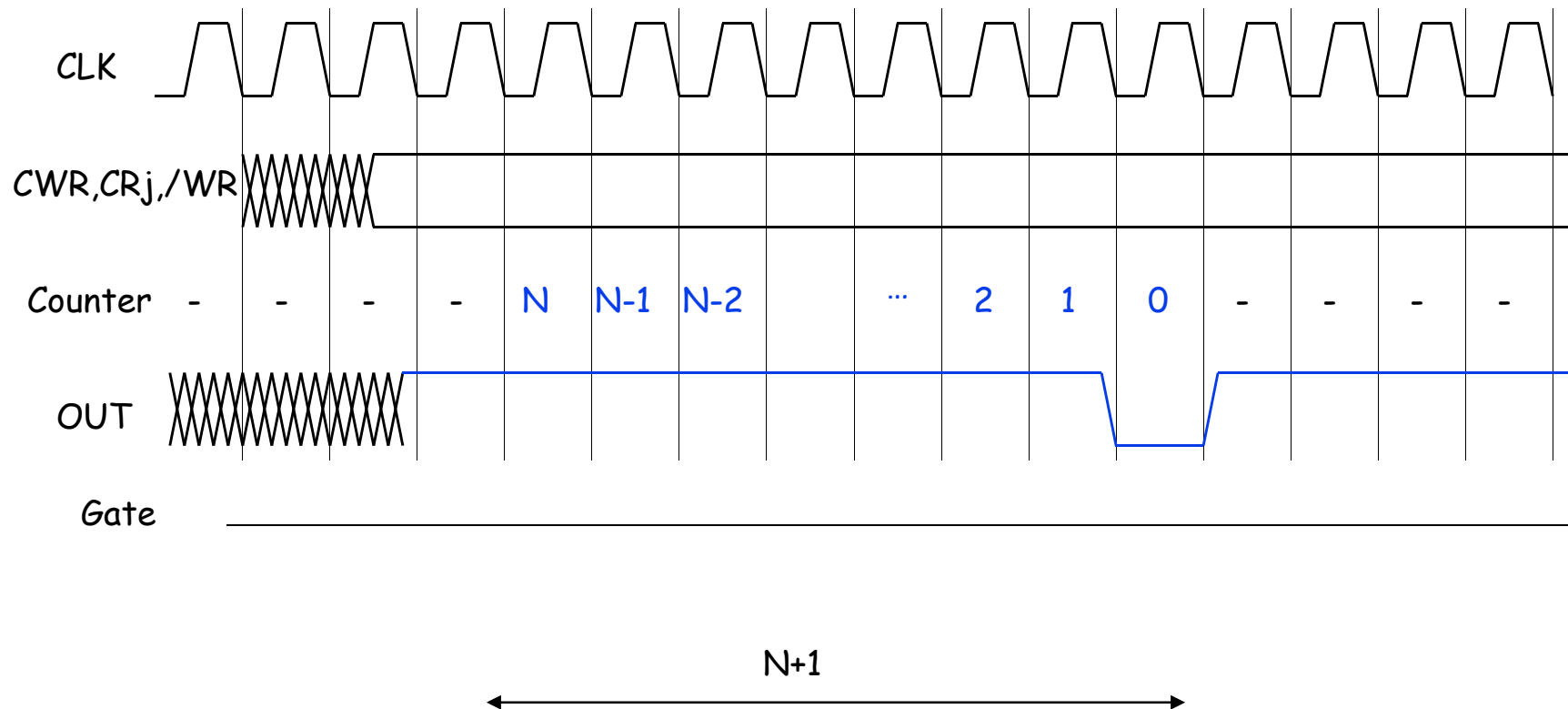
- **Modo 3: square wave mode (generatore d'onda quadra)**



- **OUT** rimane alto nei periodi $N, N-1, \dots, (N+1)/2$
- **OUT** rimane basso nei periodi $N/2, (N-1)/2, \dots, 2, 1$
- Il duty cycle e' $\frac{1}{2}$ se N e' pari, altrimenti e' $((N+1)/2)/N$

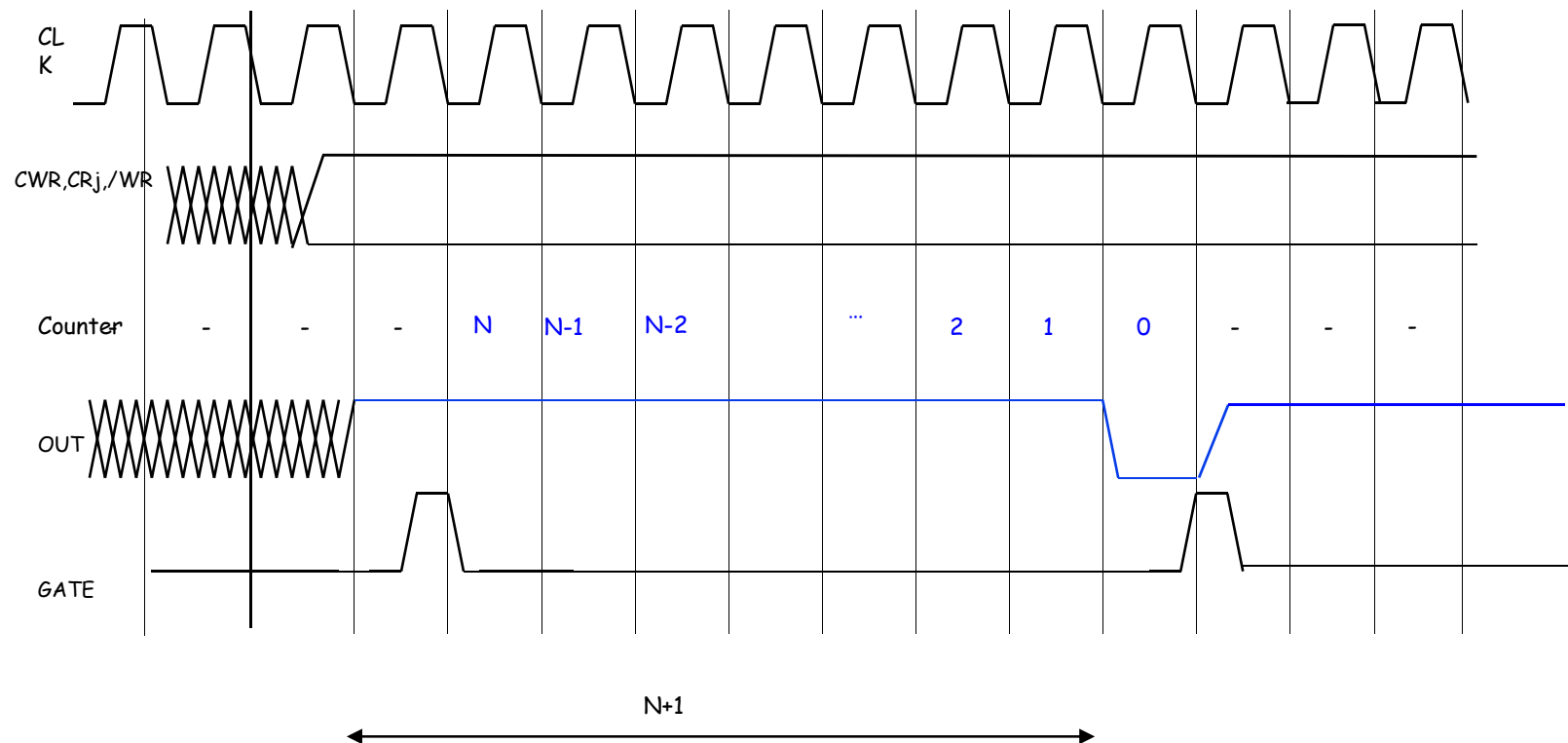
Modi di conteggio

- **Modo 4: software triggered strobe (generatore d'impulso)**



Modi di conteggio

- **Modo 5: Hardware triggered strobe (generatore d'impulso)**



Il Chip 8254 nei Personal Computer

- PORTE DI I/O ASSEGNATE
 - CR0 $\equiv$ 0x0040
 - CR1 $\equiv$ 0x0041
 - CR2 $\equiv$ 0x0042
 - CWR $\equiv$ 0x0043
- CLK0=CLK1=CLK2 = 1.19MHZ
- **TIMER-0: timer di sistema**
 - GATE0 sempre attivo
 - OUT0 genera IRQ0
- **TIMER-1: timer per il rinfresco della memoria**
 - GATE1 sempre attivo
 - OUT1 genera impulsi con un periodo di 15ms
- **TIMER-2: generatore di beep**
 - GATE2 controllabile tramite bit0 di SPR, Speaker Register (porta di I/O 0x0061)
 - OUT2 in AND col bit1 di SPR, l'uscita va poi sul beeper

Riepilogo risorse 8254

Chip name	Register name		I/O port	IRQ
Intel 8254	CR0	Counter Register 0	0x0040	IRQ0
Intel 8254	CR1	Counter Register 0	0x0041	-
Intel 8254	CR2	Counter Register 0	0x0042	-
Intel 8254	CWR	Control Word Register	0x0043	-

Glue logic	SPR	Speaker Register	0x0061	

Esempio di programmazione del Timer 8254

- **Generare una interruzione ogni 50ms**
 - $T_c = 1/1.19\text{MHz} = 840\text{ns}$
 - $N+1 = 50 \cdot 10^{-3} / T_c \rightarrow N = 59499$

```
int init_8254 (void)
```

```
{
```

```
    int cr_l, cr_h;
```

```
    outp(0x0043, 0x30); /* contatore 0, costante di tempo a 16 bit,  
                        modo 0, conteggio binario */
```

[→V. SLIDE SUCCESSIVA](#)

```
    cr_l = 59499 & 0x000000FF;
```

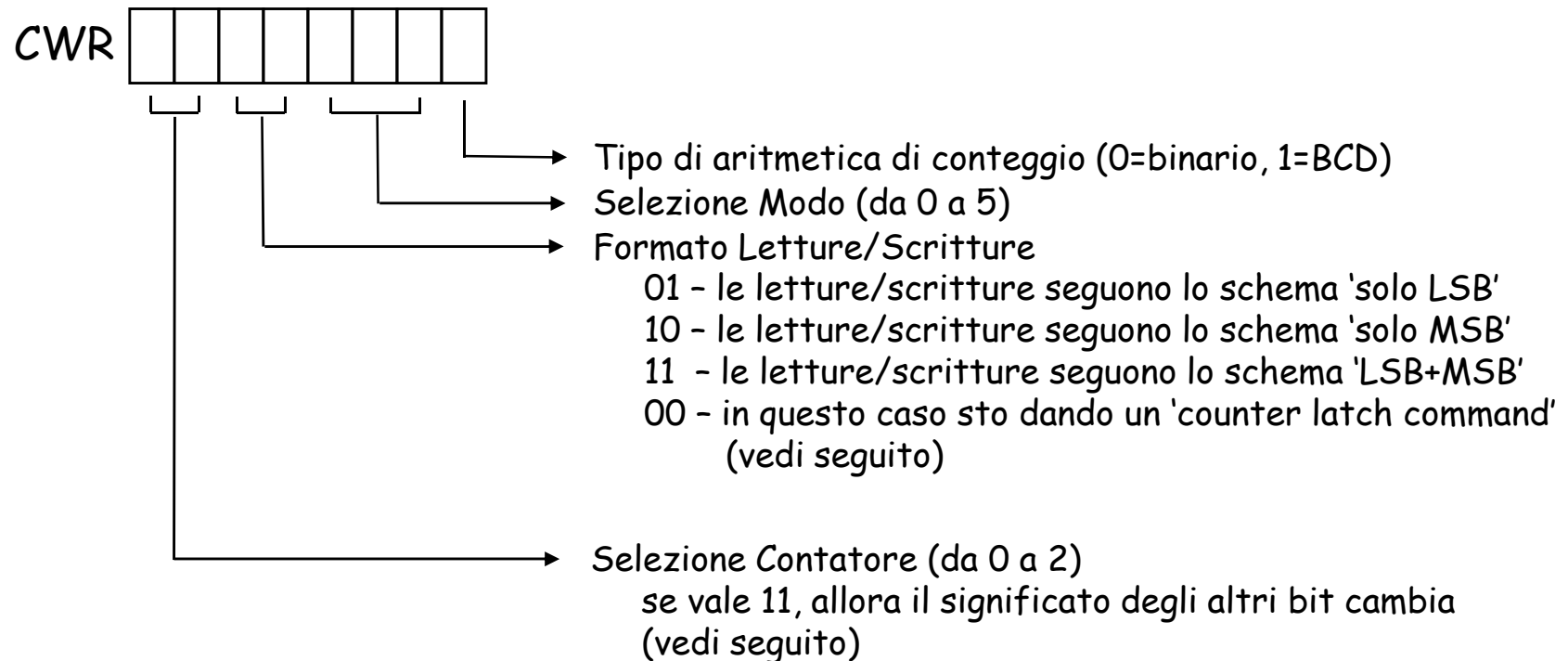
```
    cr_h = (59499 >> 8) & 0x000000FF;
```

```
    outp(0x0040, cr_l);
```

```
    outp(0x0040, cr_h);
```

```
}
```

Programmazione del Timer (2)



Es.

$CWR \leftarrow 1011'0110$ significa che il timer #2 deve effettuare un conteggio binario in modalita' #3, e le scritture/letture seguono lo schema LSB+MSB

Comunicazione Seriale Asincrona

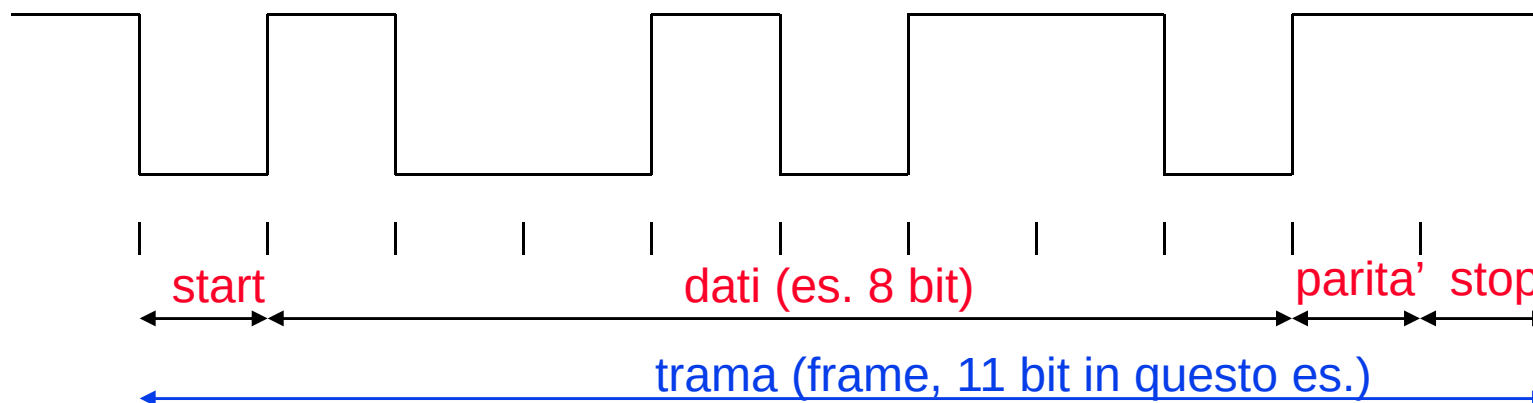
Comunicazione Seriale Asincrona (ASC)

- **Comunicazione**: ad un certo istante, e' possibile individuare
 - Un dispositivo che agisce da trasmettitore (Master o TX)
 - Un dispositivo che agisce da ricevitore (Slave o RX)
- **Seriale**: i bit che compongono il dato vengono trasmessi sul mezzo trasmissivo in sequenza
 - La trasmissione potra' essere anche wireless!
 - La trasmissione potra' usare 1 filo o piu' fili elettrici (modalita' differenziale, cfr. Ethernet baseT)
- **Asincrona**: la trasmissione dei dati avviene seguendo un protocollo di comunicazione asincrono
 - Nessun segnale di clock e' associato alla comunicazione

Caratteristiche principali di una ASC

- La velocità della comunicazione è caratterizzata dal numero di bit trasmessi per secondo o **bit-rate**
 - Se T è il tempo per trasmettere un bit, $1/T$ è il bit-rate
- Il dato è costituito da una **trama (frame)** di bit
 - L'informazione è tipicamente racchiusa tra un **header** o preambolo e un **trailer** o coda
 - L'header serve a marcare l'inizio della trasmissione
 - Il trailer serve a marcare la fine della trasmissione e a trasmettere un codice di verifica (CRC o parità)
- Ricevitore e trasmettitore si sincronizzano usando l'header
 - Il clock in ricezione cerca così di ottenere stessa fase del clock del trasmettitore: la frequenza deve però essere nota!

Trama seriale



Frame comunicazione seriale

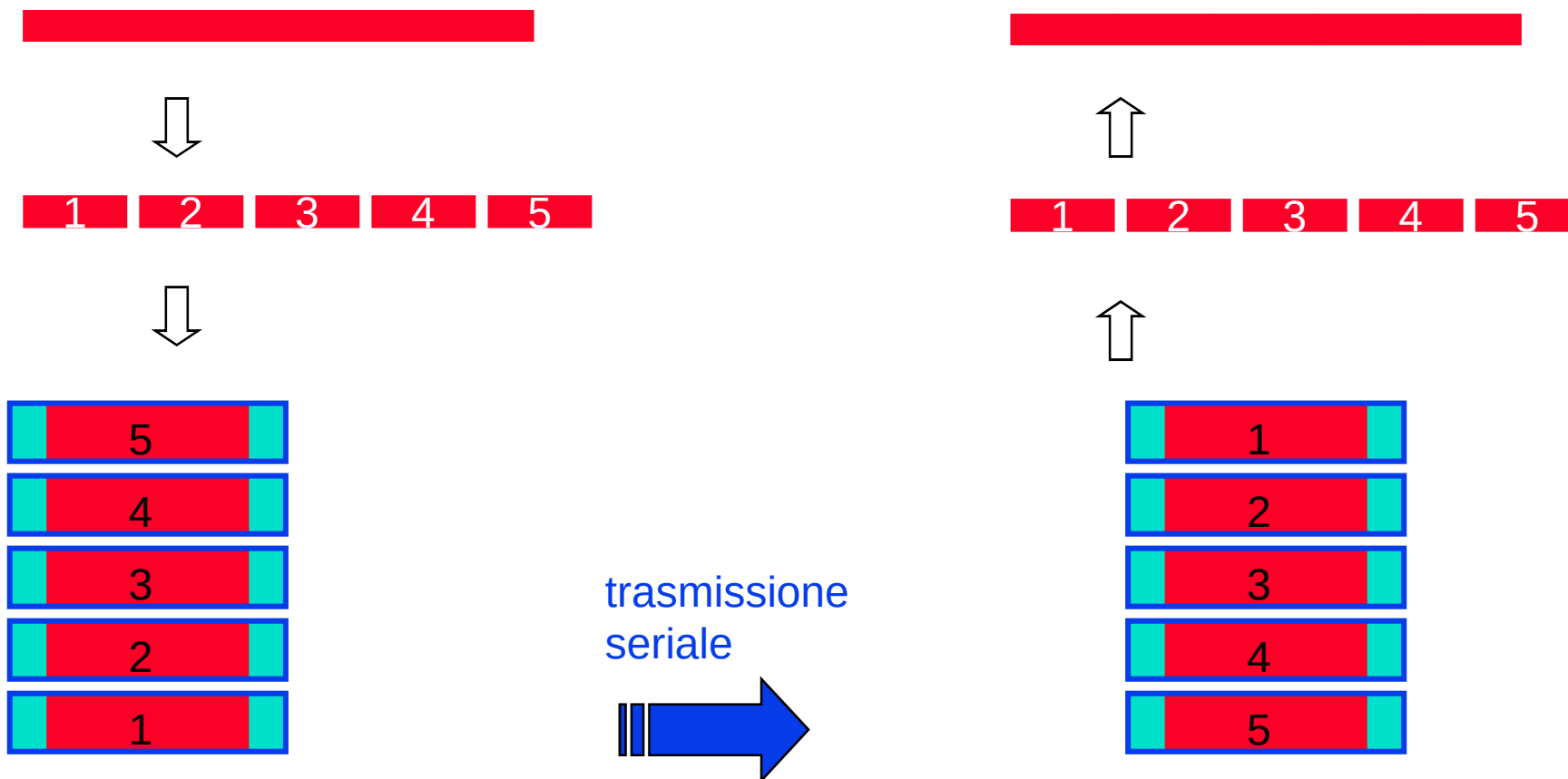


Informazioni che e' necessario sapere a priori

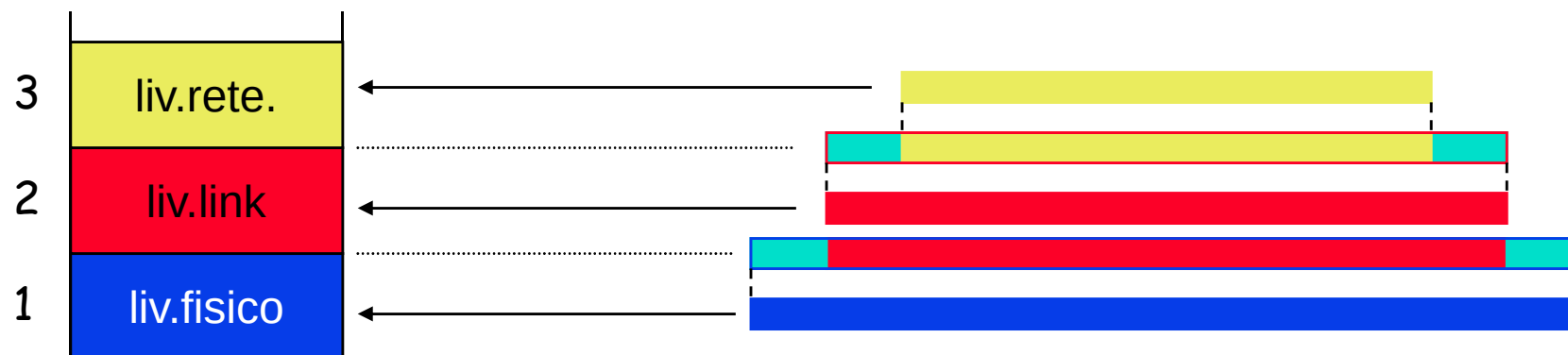
- Velocita' di trasmissione (bit-rate, es. 1200 bit/s)
- Formato della trama (es. 1 bit di start, 8 bit dati, 1 bit di parita', parita' positiva, 1 bit di stop)

Pacchettizzazione

Es. Trasmissione di 5 byte



Stratificazione (Stack di Comunicazione)



Stack di comunicazione:

- verso l'alto, ogni livello fornisce una comunicazione con più proprietà
- verso il basso i dati vengono 'arricchiti' con ulteriori info di controllo

Gestione delle reti per astrazioni successive

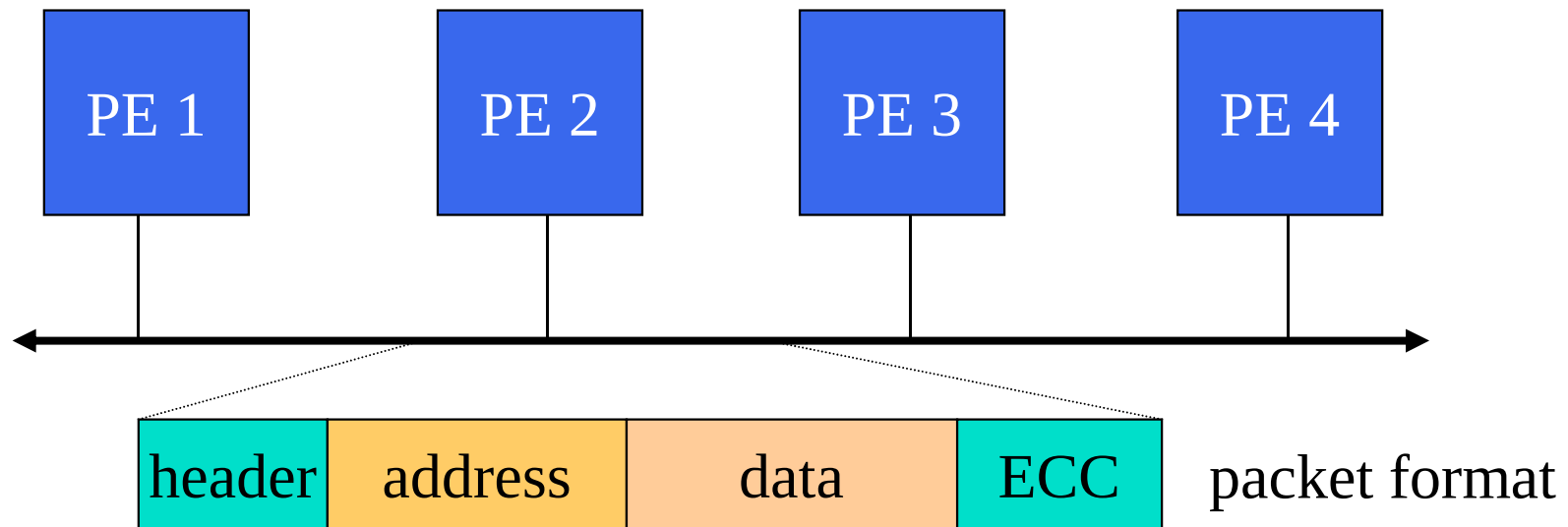
- International Standards Organization (ISO) ha sviluppato un modello per descrivere le reti chiamato **Open Systems Interconnection (OSI)**
 - Modello a 7 livelli
 - Standard noto come **ISO/OSI**
- Fornisce uno standard per classificare i componenti delle reti e le operazioni coinvolte

Modello OSI

7	applicazione	interfaccia utente
6	presentazione	formato dati (tipi dei dati, trasformazione dei servizi)
5	sessione	comunicazione fra le applicazioni ai due estremi (raggruppamento dati, checkpointing)
4	trasporto	connessione (gestione connessione, ottimizzazione gestione di rete)
3	rete (network)	servizio end-to-end (comunicazione dati multi-hop end-to-end)
2	data link	trasporto dati affidabile (rilevazione errori e gestione singolo link (single-hop))
1	fisico	caratteristiche meccaniche/elettriche (connettori, codifica dei bit,...)

Livelli 1 & 2: Reti Seriali a Bus

- Una unica connessione fisica comune



- I2C, USB, FireWire, Ethernet, ...

Standard Ethernet

Standard Ethernet - Livelli 1 & 2

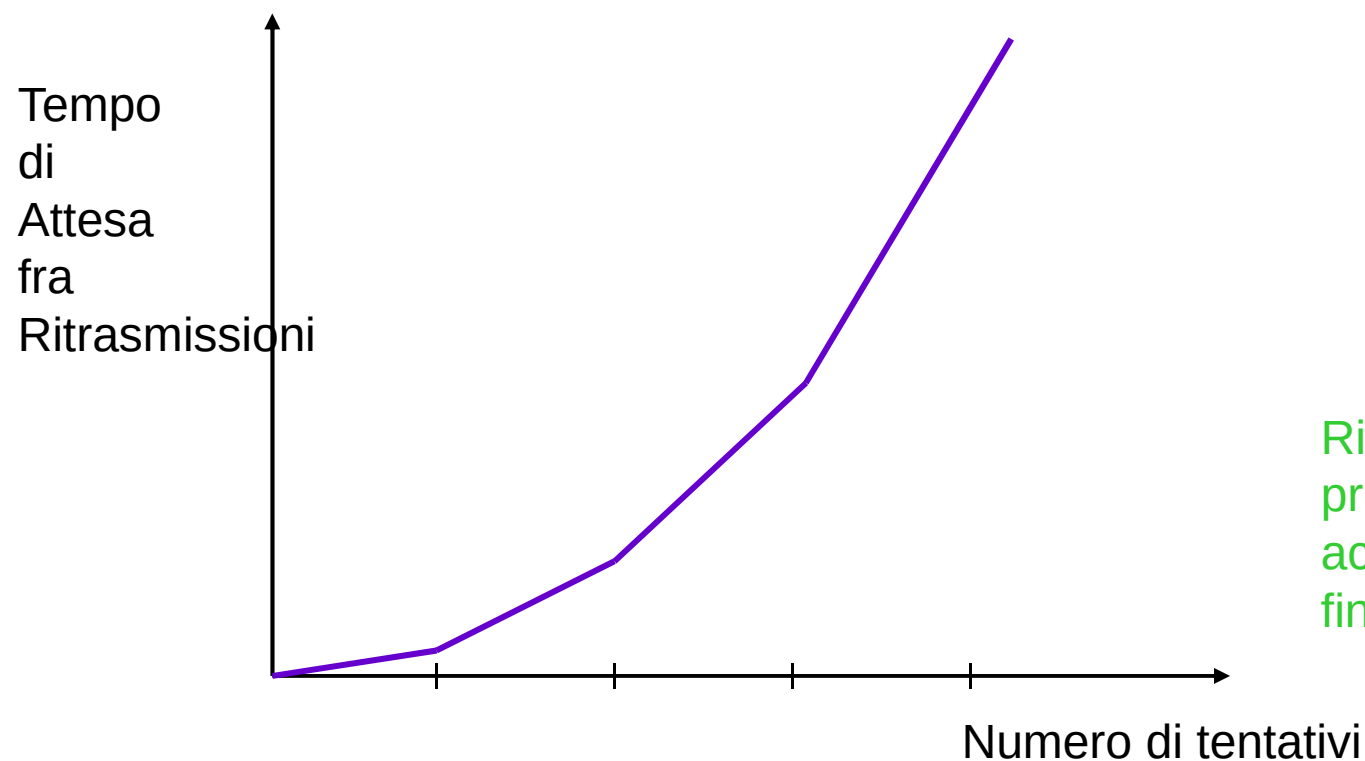
- Risale ai primi anni'80
- Standardizzato come IEEE 802.3
- Dominante nelle reti locali (LAN)
- Versioni:
 - 10 Mb/s (IEEE 802.3, o Ethernet)
 - 100 Mb/s (IEEE 802.3u, o Fast Ethernet)
 - 1 Gb/s (IEEE 802.3ab, o Gigabit Ethernet)
 - 10 Gb/s (IEEE 802.3ae, o 10-Gigabit Ethernet)
 - 100 Gb/s (IEEE 802.3ba, o 100-Gigabit Ethernet)
- Wireless LAN (IEEE 802.11, o Wi-Fi)
- Low-Power WLAN (IEEE 802.15, o Bluetooth)
- Topologia: "a bus"
- Obiettivo: ottenere una comunicazione affidabile usando un mezzo trasmissivo eventualmente non affidabile

CSMA/CD, Carrier Sense Multiple Access with Collision Detection

- Un nodo "ascolta prima di parlare" (carrier sense)
- Se il mezzo e' libero si attende un tempo "Defer Time"
- Se ci sono collisioni (multiple access) l'energia sul mezzo aumenta
- Un nodo cerca di rilevare collisioni durante la trasmissione (collision detect)
- Se un nodo rileva una collisione, continua a trasmettere 4-6 bytes per essere sicuro che la collisione venga rilevata dagli altri nodi e poi "lascia" il mezzo
- Se un nodo ha rilevato collisione, la trasmissione viene ripetuta dopo $R \cdot \Delta t$ per altri 15 tentativi
- Dopo 16 tentativi l'errore viene segnalato ai livelli superiori
- Δt e' fisso mentre R viene calcolato secondo un "algoritmo di back off"

Algoritmo di back off per il calcolo di R

- Sia n l' n -esimo tentativo di trasmissione ($n=1,2,\dots, 15$)
- $K = \min(n, 10)$
- R e' un numero casuale t.c. $0 \leq R \leq 2^K$



Risolve in maniera
probabilmente
accettabile i conflitti
fino a 1024 nodi...

Formato del Pacchetto Ethernet



- In dettaglio

• Preamble	101010101010...	}	8 Bytes
• Start frame	11		
• Source address	6 Bytes	}	Ethernet FRAME
• Destination address	6 Bytes		
• Length	2 Bytes		
• Data Payload	i dati veri e propri!		
• Padding	Eventuali byte in coda al payload		
		46-1500 Bytes	64-1518 Bytes
• CRC	4 Bytes		

- Segue un IPG (Inter Packet Gap) di almeno 96 bit...

Prestazioni della rete Ethernet

- La Qualita' del servizio (Quality-of-Service, o QoS) tende a decrescere in maniera non-lineare ad alti livelli di carico
- Una conseguenza dell'algoritmo di back off e' che non e' possibile stabilire il tempo massimo entro cui verra' trasmessa con successo una trama
 - Non si possono garantire deadline real-time!
 - Si possono ottenere livelli molto buoni di servizi purché si riesca a mantenere il carico ad un livello appropriato

Porta Seriale

Porte Seriali

- Sui Personal Computer e' presente la cosiddetta **porta seriale**
- Metodo efficiente di comunicazione tra dispositivi
 - Tipicamente denominata UART (Universal Asynchronous Receiver and Transmitter)
 - Es. 16550A, 16450, 8250, 8251A, MC68681
- Semplice porta seriale costituita da:
 - 2 registri a scorrimento (shift register)
 - Uno trasmettitore (Tx)
 - L'altro ricevente (Rx)
 - Uscita del Tx collegata all'ingresso del Rx
 - Trasmissione e ricezione guidate dallo stesso clock
 - Quando Tx è vuoto o Rx è pieno (ricevuto 1 byte) viene generato un interrupt

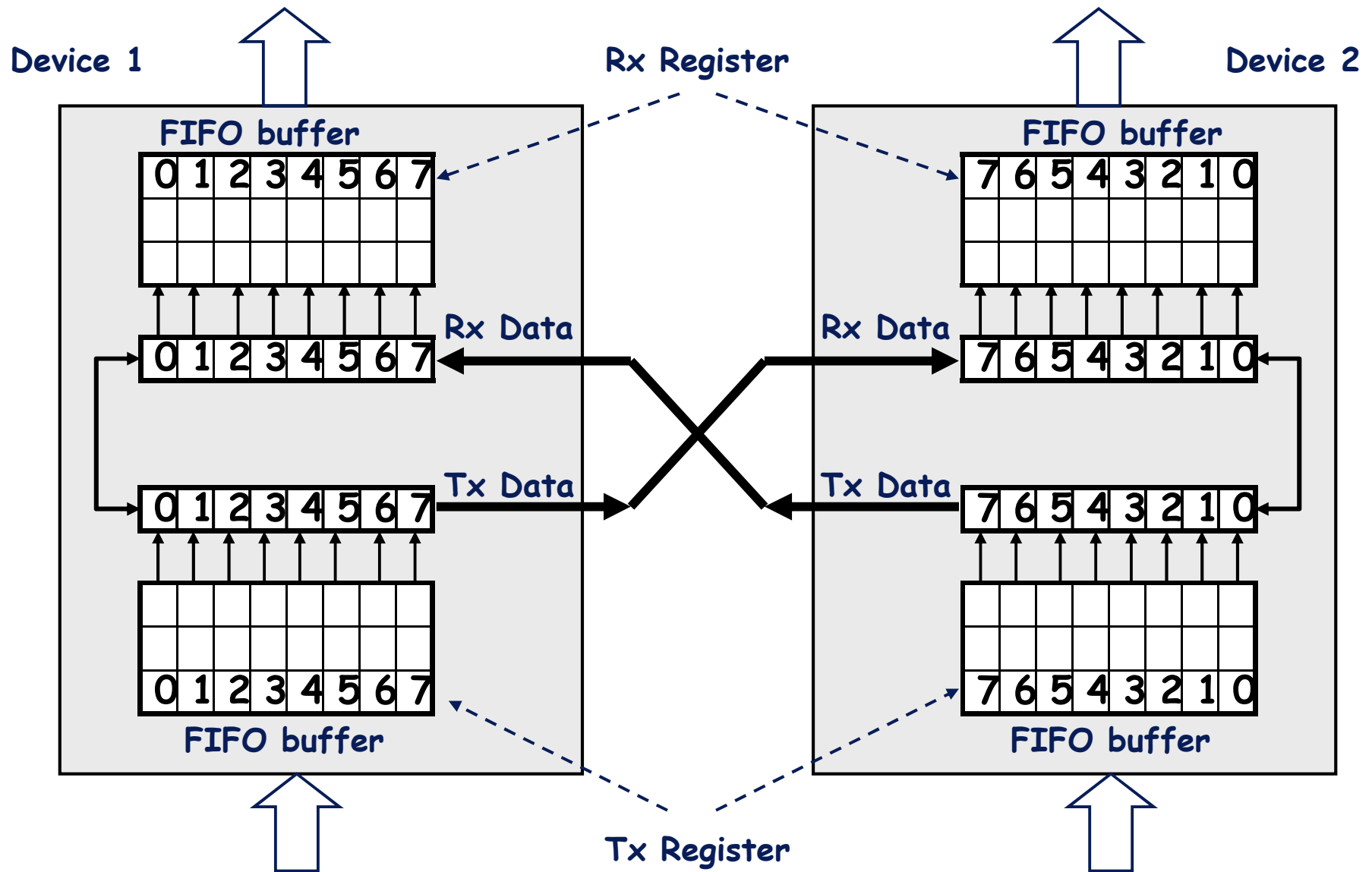
Struttura fisica della porta seriale

- **Trasmissione seriale asincrona full-duplex**
 - trasmissione contemporanea in entrambi i versi: i due dispositivi fungono sia da Tx che da Rx
 - Sono sufficienti 3 fili di collegamento (Tx,Rx,GND)
- **Trasmissione seriale asincrona half-duplex**
 - Ad un certo istante, trasmissione in una sola direzione: uno dei due dispositivi funge da Tx l'altro da Rx
 - Sono sufficienti 2 fili (Tx, GND)
- **Fisicamente costituita da**
 - 2 registri a scorrimento (shift register)
 - Uno per trasmettere (Tx)
 - Uno per ricevere (Rx)
 - Uscita del Tx collegata all'ingresso del Rx
 - Quando Tx è vuoto o Rx è pieno (ricevuto 1 byte) viene generato un interrupt

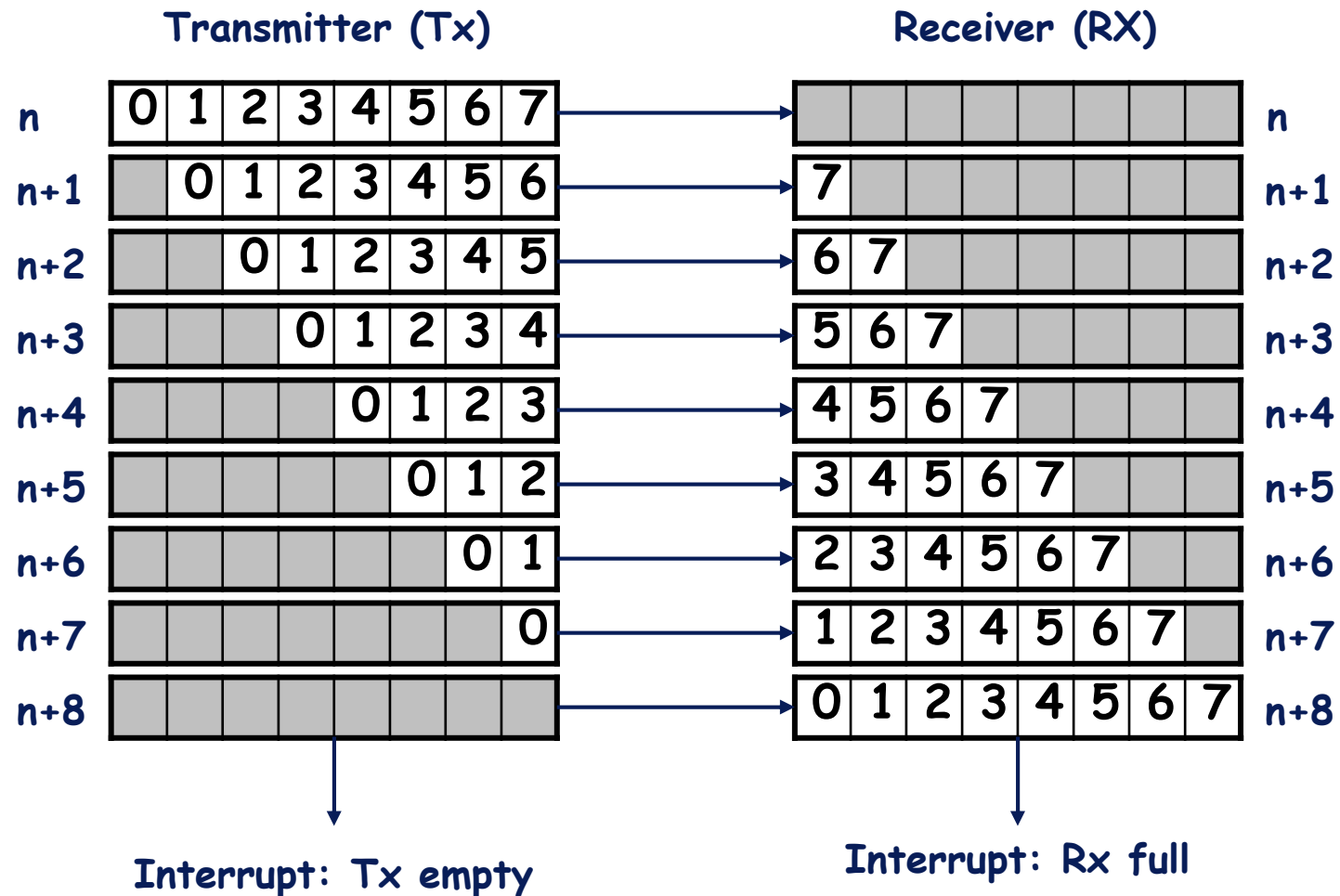
Uso di un buffer FIFO

- Utilizzo di un buffer FIFO per evitare la perdita di dati
 - in Rx:
dato ricevuto prima che sia stato letto quello precedente
 - in Tx:
dato inserito prima che sia stato trasmesso il precedente
- Dimensione del buffer FIFO:
 - Aumenta il throughput della porta
 - Riduce l'overhead della CPU (attesa della Tx o Rx)

Interfaccia Seriale Asincrona con Buffer FIFO



Es: trasmissione di un byte



Esempio di trasmissione su porta seriale

- Lato Tx:
 - Appena il dato è nel Tx-Register, viene trasmesso
 - Viene generato un interrupt per segnalare che il byte è stato trasmesso
- Lato Rx:
 - Il buffer riceve bit dopo bit il dato
 - Appena viene ricevuto un intero byte, viene trasferito nell' Rx-register
 - Viene generato un interrupt ed il byte viene letto
- In entrambi i casi
 - Viene messo a 1 un bit nel registro di stato (a fine trasmissione o ricezione)
 - Per poter proseguire, tale bit deve essere azzerato
 - Es. dalla routine di servizio

Controllo del flusso

- **Necessario per impedire che Tx invii più dati di quelli che Rx può leggere**
- **Due metodi**
 - **Hardware:**
 - il chip UART del Rx individua un potenziale intasamento e attiva una linea apposita per avvertire il Tx
 - Tx interrompe la trasmissione
 - Appena Rx può ricevere dati, la linea viene disattivata
 - **Software:**
 - Il Rx invia caratteri speciali di controllo (XON e XOFF) al Tx
 - XOFF interrompe una trasmissione
 - XON riprende la trasmissione

Standard EIA-RS232C

- **Segnali:**
 - In trasmissione, 1 logico= $[-15V, -5V]$, 0 logico= $[+5V, +15V]$
 - In ricezione, 1 logico= $[<-3V]$, 0 logico= $[>+3V]$
 - Il Tx deve vedere una capacita' $<2.5nF$, una resistenza $<7K\Omega$
 - Il Tx deve poter supportare un corto circuito con corrente $<0.5A$
 - Massima lunghezza cavo:
 - per 20Kb/s $\rightarrow$ 16m,
 - per 9.6Kb/s $\rightarrow$ 75m,
 - per 1.2Kb/s $\rightarrow$ 1000m

Standard EIA-RS232C

- I connettori più comuni sono due:
 - DB-25: connettore D type a 25 pin
 - DB-9 : connettore D type a 9 pin

DB-25	Sigla	Segnale	DB-9
1	GND	Chassis Ground	Not Used
2	TXD	Trasmit Data	3
3	RXD	Receive Data	2
4	RTS	Request To Send	7
5	CTS	Clear To Send	8
6	DSR	Data Set Ready	6
7	GND	Signal Ground	5
8	DCD	Data Carrier Detect	1
20	DTR	Data Terminal Ready	4
22	RI	Ring Indicator	9

Segnali (1)

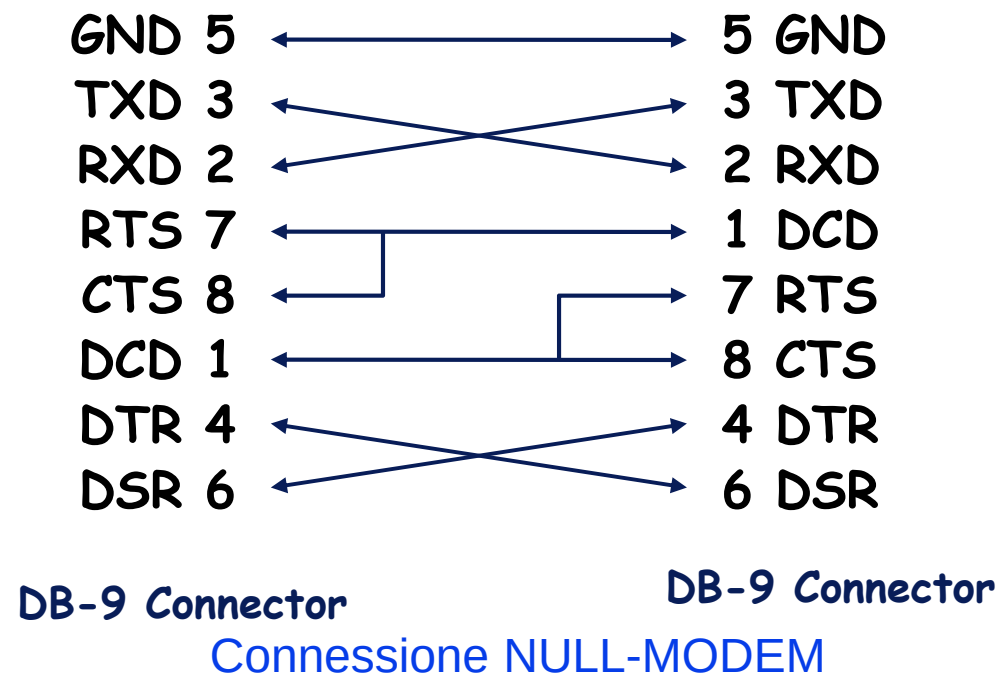
- **TXD (Transmit Data):**
linea di trasmissione dei dati
connessa alla linea RXD del Rx
- **RXD (Receive Data):**
linea di ricezione del dato,
connessa alla linea TXD del Tx
- **RTS (Request to send):**
utilizzata in congiunzione con CTS
segnala che Tx è pronto ad inviare dati
- **CTS (Clear To Send):**
utilizzata in congiunzione con RTS
segnala che Rx è pronto a ricevere i dati

Segnali (2)

- **DSR (Data Set Ready):**
indica che il data-set (modem) è pronto a stabilire la comunicazione
- **DTR (Data Terminal Ready):**
indica che il data-terminal (UART) è pronto a stabilire la comunicazione
- **DCD (Data Carrier Detect):**
viene usato quando è connesso un modem indica che il data-set (modem) ha rivelato la portante sulla linea telefonica
- **RI (Ring Indicator):**
viene usato quando è connesso un modem e indica che ci sono degli squilli in arrivo sulla linea telefonica

Modem Cables

- Molte funzionalità di questi segnali derivano dal fatto che inizialmente furono pensati per connettere modem
- Per connessioni con altre periferiche, come stampanti, i segnali relativi ai modem devono essere "annullati" con l'uso di
 - cavi diretti oppure
 - cavi null-modem, es. per connettere due PC

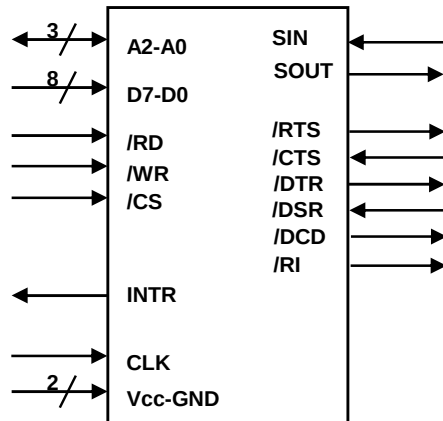


La Porta Seriale nei Personal Computer

- Tipicamente implementata con un chip tipo 16550A
 - Bit-rate: compreso fra 50 e 115000
 - Formato trama:
 - 1 bit di start
 - da 5 a 8 bit di dati
 - con o senza parita'
 - 1, 1.5, 2 bit di stop
 - Riconoscimento dell'errore
 - Di parita' (numero di 0 o 1 non corrispondente)
 - Di framing (bit di stop non ricevuti correttamente)
 - Di overrun (arrivo di un nuovo dato prima di aver prelevato il precedente)
 - Emissione/ricezione di break
 - Persistenza in stato attivo per un tempo piu' lungo della trama
 - Gestione di 4 pin di ingresso e 4 di uscita, utili per rispettare lo standard EIA-RS232C
 - Buffer dati da 16 byte

UART Intel-16550A

- **Visione Elettrica del chip:**



- **A2-A0:** indirizzamento
- **D7-D0:** scambio dati su 8 bit
- **/RD, /WR, /CS:** Read, Write, Chip-Select
- **Vcc-GND:** alimentazione
- **CLK:** clock locale per derivare le temporizzazioni
- **SIN, SOUT:** segnali Tx e Rx
- **/RTS, /CTS, /DTR, /DSR, /DCD, /RI:** segnali per l'interfacciamento secondo EIA-RS232C
- **Altri pin** possono essere presenti per usi generali e in dipendenza delle versioni specifiche del chip

UART Intel-16550A

• Visione logica del chip (tutti registri a 8 bit):

A2-A0	access	DLAB bit	mnemonic		
000	R	0	RBR		Receive Buffer Register Dove si trova il dato ricevuto
000	W	0	THR		Transmitter Holding Register Dove si mette il dato da trasmettere
001	RW	0	IER		Interrupt Enable Register
010	R	0	IIR		Interrupt Identification Register
010	W	0	FCR		FIFO Control Register
011	RW	-	LCR		Line Control Register
100	RW	-	MCR		Modem Control Register I bit 0 e 1 sono dei latch su /RTS e /DTS (in scrittura gli altri bit possono essere messi a 0)
101	RW	-	LSR		Line Status Register
110	RW	-	MSR		Modem Status Register
111	RW	-	SCR		Scratchpad Register Registro di appoggio da usare liberamente
001	RW	1	DLM		Divisor Latch MSB
000	RW	1	DLL		Divisor Latch LSB

Visione logica: DLM e DLL

- Avendo posto a 1 il bit di accesso ai Divisor Latches (DLAB=1) posso scrivere/leggere in DLM, DLL il valore della costante C che determina il bit-rate
 - DLM, DLL vengono interpretati come un intero senza segno a 16 bit
 - Detta FCK la frequenza del clock esterno e BR il bit-rate desiderato, si ha

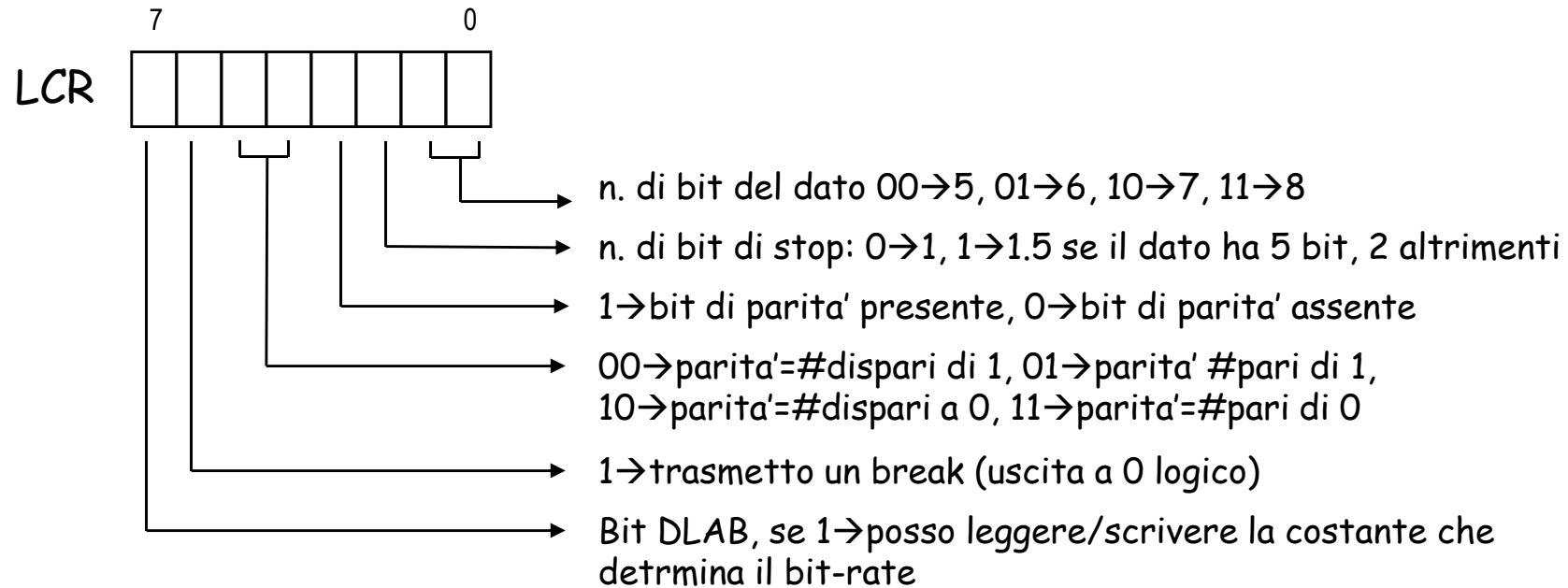
$$C = \frac{F_c}{16 \cdot BR}$$

Nota: nei PC $F_c = 1.8432\text{MHz}$
Es. per ottenere un $BR = 9600$
 $\rightarrow C = 12$

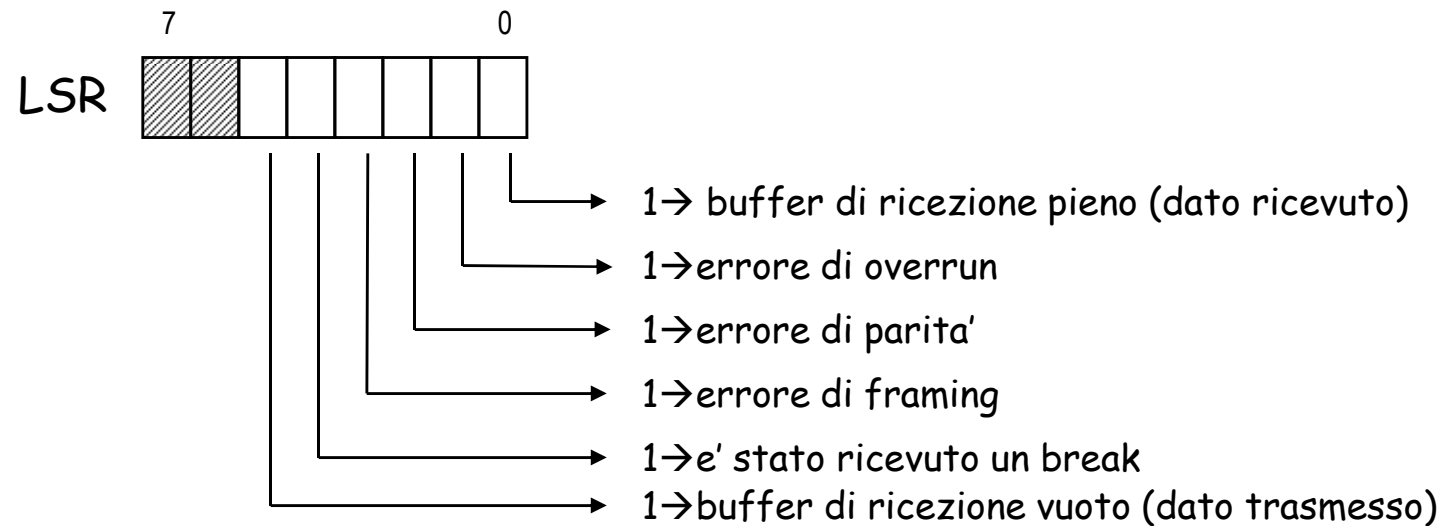
BR	C
300	384
1200	96
4800	24
9600	12
115200	1

- DLAB non e' altro che il bit 7 di LCR
- I registri RBR, THR, IER, IIR, FCR si accedono ponendo DLAB=0
- Per i restanti registri il valore di DLAB e' influente

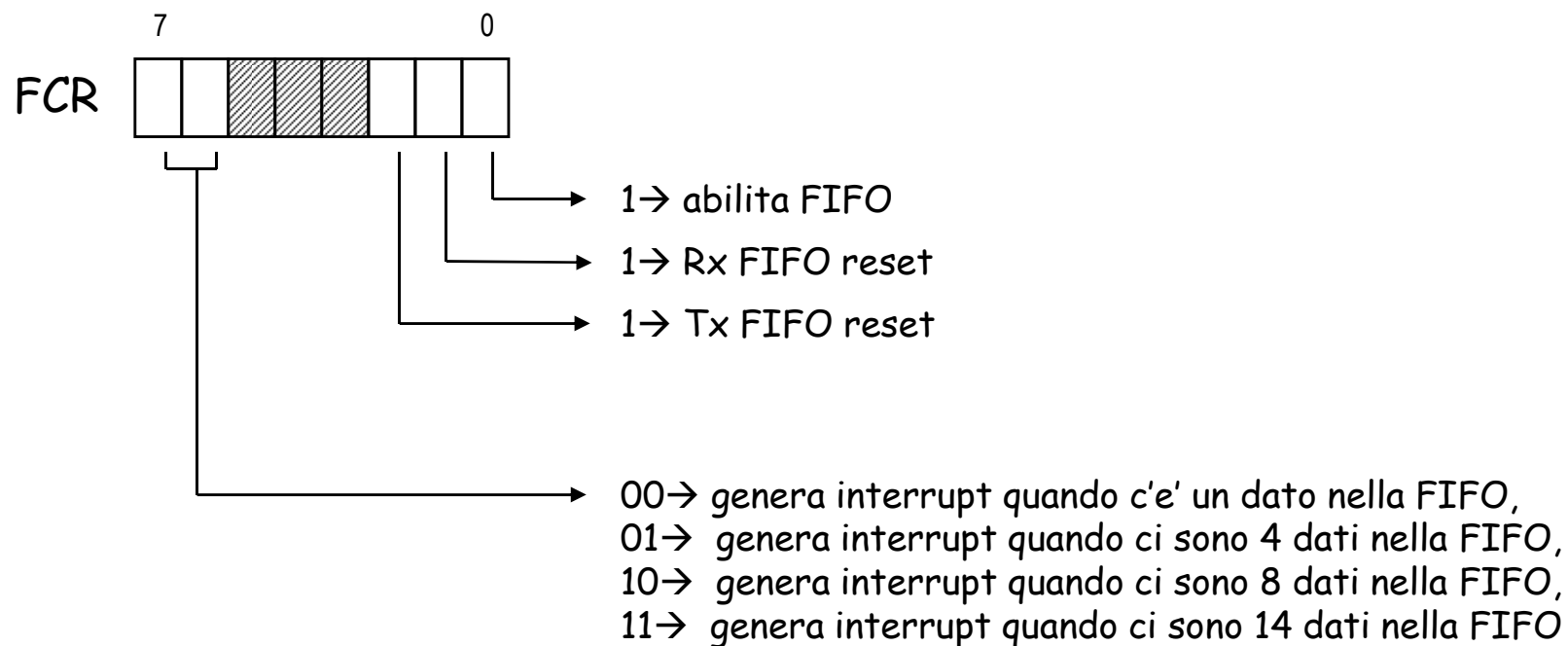
Visione logica: LCR, Line Control Register



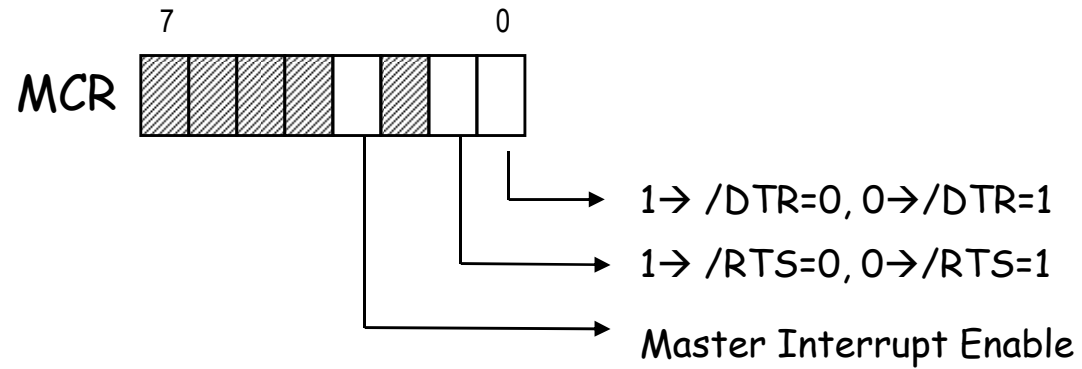
Visione logica: LSR, Line Status Register



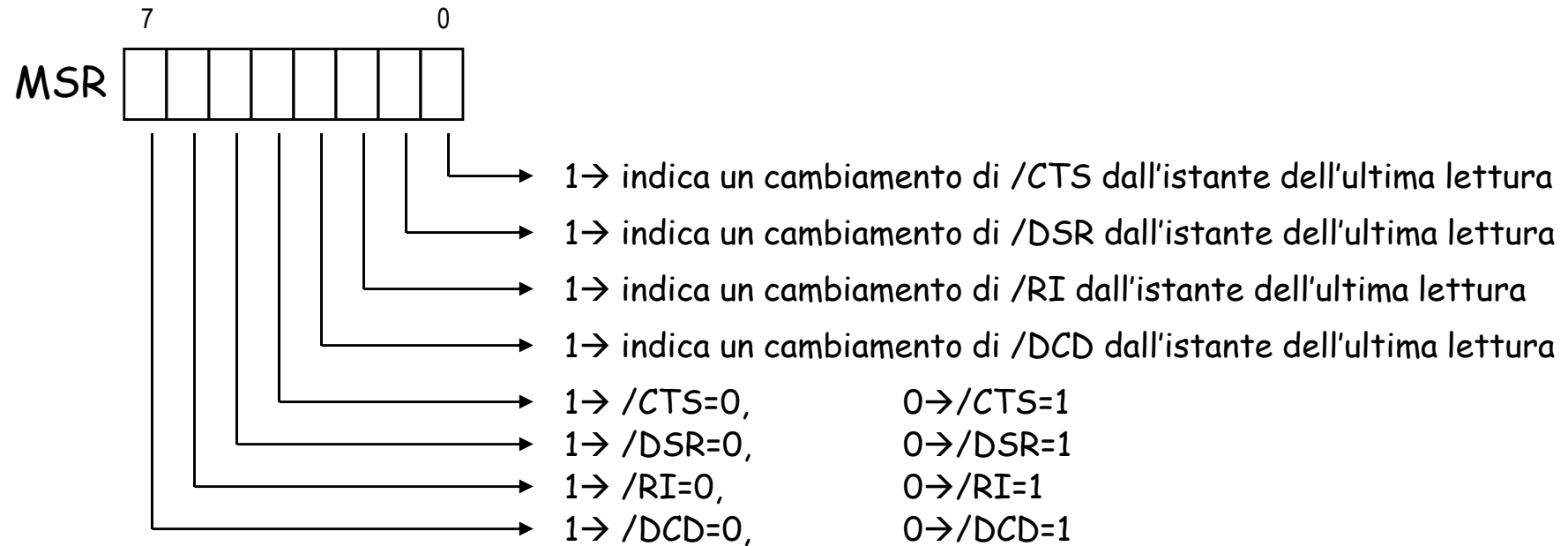
Visione logica: FCR, FIFO Control Register



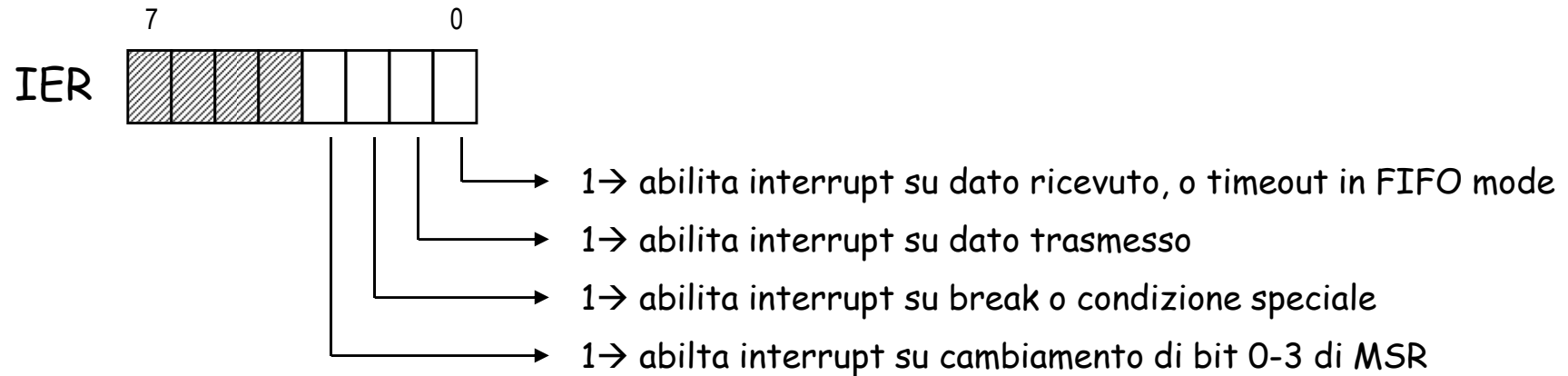
Visione logica: MCR, Modem Control Register



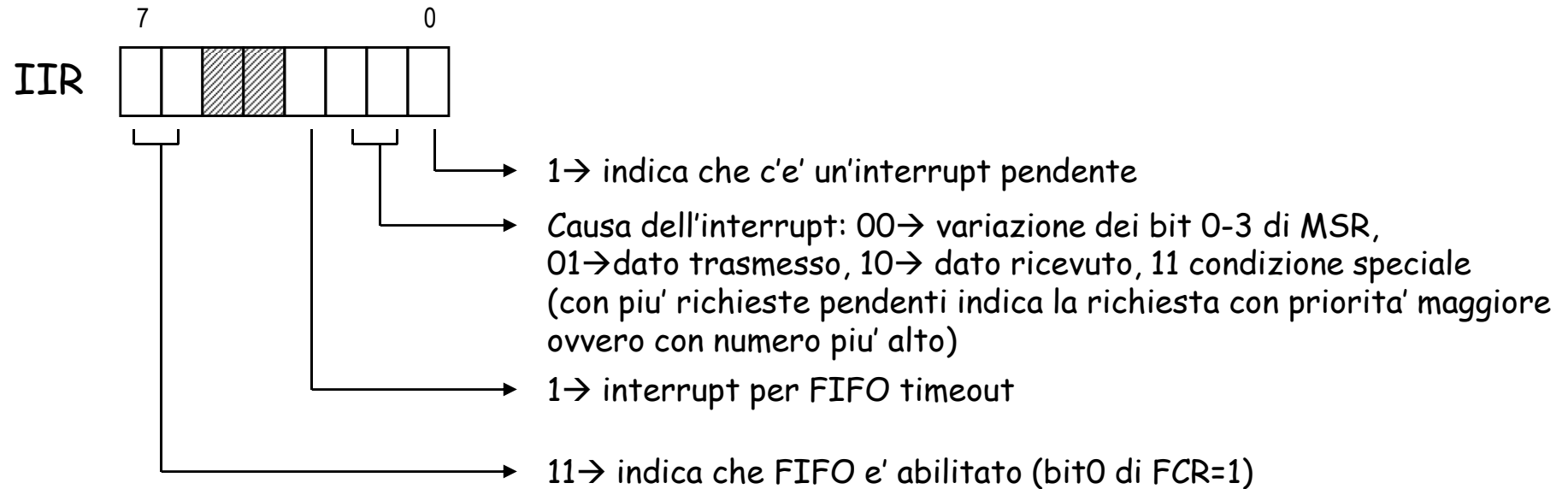
Visione logica: MSR, Modem Status Register



Visione logica: IER, Interrupt Enable Register



Visione logica: IIR, Interrupt Identification Register



- Si rimanda ai data sheet per ulteriori dettagli e varianti

Il chip 16550 nei Personal Computer

- PORTE DI I/O ASSEGNATE e INTERRUPT (Risorse)
 - 0x03F8-0x3FF porta seriale #1 (COM1 o ttyS0) con IRQ4
 - 0x02F8-0x2FF porta seriale #2 (COM2 o ttyS1) con IRQ3
 - 0x03E8-0x3EF porta seriale #3 (COM3 o ttyS2) con IRQ4
 - 0x02E8-0x2EF porta seriale #4 (COM4 o ttyS3) con IRQ3

- Nota: poiche' i pin SIN, SOUT, /RTS, /CTS, /DSR, /DCD, /RI sono a livelli TTL (0-5V), si rendono necessari dei chip 75188, 75189 per adattare i livelli di tensione a quelli richiesti dallo standard RS-232C.

Esempio di programma che utilizza la porta seriale (1)

```
#include <sys/types.h>
#include <sys/stat.h> /* open */
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h> /* stdin, stdout, stderr */
#include <sys/ioctl.h>
#include <termios.h>
#include <stdlib.h>
#include <sys/time.h> /* timeval */
#include <stdlib.h>
#include <string.h> /* memset used by FD_ZERO */

#define DEFAULT_DEV "/dev/ttyS0"
#define DEFAULT_SPEED 115200

int translate_speed(int spd);
void fd_setup(int fd, int speed, int flow);

int main(int argc, char **argv)
{
    int serfd, i, c=0, error=0, len, do_echo = 0;
    char txdata[127], rxdata[256];

    if (argc > 2) {
        printf("wrong syntax ! ./sertest [-e]");
        exit(1);
    }

    if (argc == 2) {
        if ((strcmp("-e", argv[1]))==0) do_echo=1;
        else printf("wrong syntax ! ./sertest [-e]\n");
    }

    printf("Opening %s\n", DEFAULT_DEV);
    serfd = open(DEFAULT_DEV, O_RDWR);

    if (serfd < 0) exit(1);

    fd_setup(serfd, DEFAULT_SPEED, 1);

    for (i=0;i<127;i++) txdata[i]=i+64;
```

```
if (!do_echo)
{
    while (!error){
        len = 0;
        i=0;
        printf("-%d- Sending data...", c++);
        write(serfd, txdata, 127);
        printf("done.\n");

        /* wait for 127 chars */
        while (len < 127)
        {
            len += read(serfd, rxdata+len, 127);
            printf("Got [%d/127]...\r", len);
            fflush(stdout);
        }

        /* now compare the received data */
        printf("Got response, comparing... ");

        if (memcmp(txdata, rxdata, 127) == 0)
            printf("OK.\n");
        else
            error = 1;
    }
}
```

Esempio di programma che utilizza la porta seriale (2)

```
else
{
    printf("Running as echo server.\n");
    while (!error) {
        len = 0;

        /* wait for 127 chars */
        while (len < 127)
        {
            len += read(serfd, rxdata+len, 127);
            printf("Got [%d/127]...\r", len);
            fflush(stdout);
        }

        /* now compare the received data */
        printf("-%d- Checking RX data... ", c++);

        if (memcmp(txdata, rxdata, 127) == 0)
        {
            printf("OK...echoing.\n");
            write(serfd, rxdata, 127);
        }
        else
        {
            printf("ERROR\n");
            error = 1;
        }
    }
}

printf("Not OK !\n");
for (i=0;i<127;i++)
    printf("%c", txdata[i]);

printf("\n");
exit(1);
}
```

```
int translate_speed(int spd)
{
    int speed_c = 0;

    switch(spd) {
        case 9600:    speed_c = B9600; break;
        case 19200:   speed_c = B19200; break;
        case 38400:   speed_c = B38400; break;
        case 57600:   speed_c = B57600; break;
        case 115200:  speed_c = B115200; break;
        case 230400:  speed_c = B230400; break;
        case 460800:  speed_c = B460800; break;
        default:      printf(
            "Bad baudrate %d is not valid.\n", spd); break;
    }
    return speed_c;
}

void fd_setup(int fd, int speed, int flow)
{
    struct termios t;

    if (fd < 0) { perror("fd_setup"); exit(1); }

    if (tcgetattr(fd, &t) < 0)
        { perror("tcgetattr"); exit(1); }

    cfmakeraw(&t);

    t.c_cflag &= ~CBAUD;
    t.c_cflag |= translate_speed(speed) | CS8 | CLOCAL;
    t.c_oflag = 0; /* turn off output processing */
    t.c_lflag = 0; /* no local modes */

    if (flow) t.c_cflag |= CRTSCTS;
    else t.c_cflag &= ~CRTSCTS;

    if (tcsetattr(fd, TCSANOW, &t) < 0)
        { perror("fd_setup : tcsetattr"); exit(1); }
    return;
}
```

Lezione 16

Introduzione al sottosistema di memoria

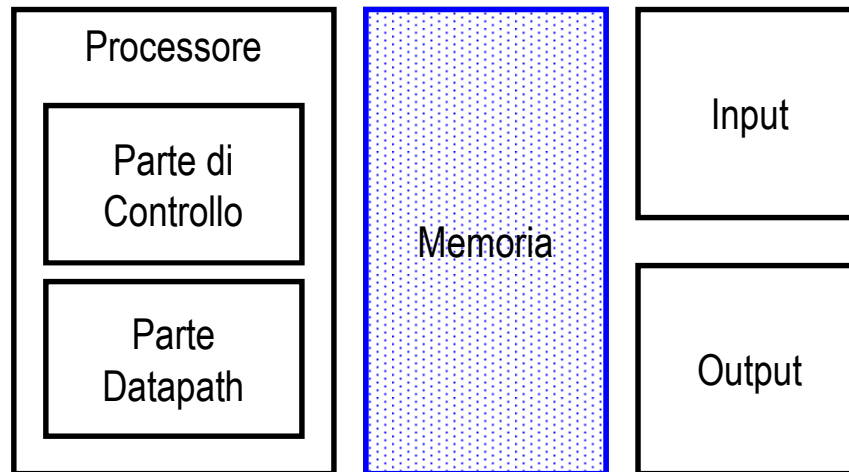
<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)
Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Other material is adapted from CS152 Copyright (C) 2000 UCB

Dove siamo

- **I Cinque Componenti Classici di un Calcolatore**



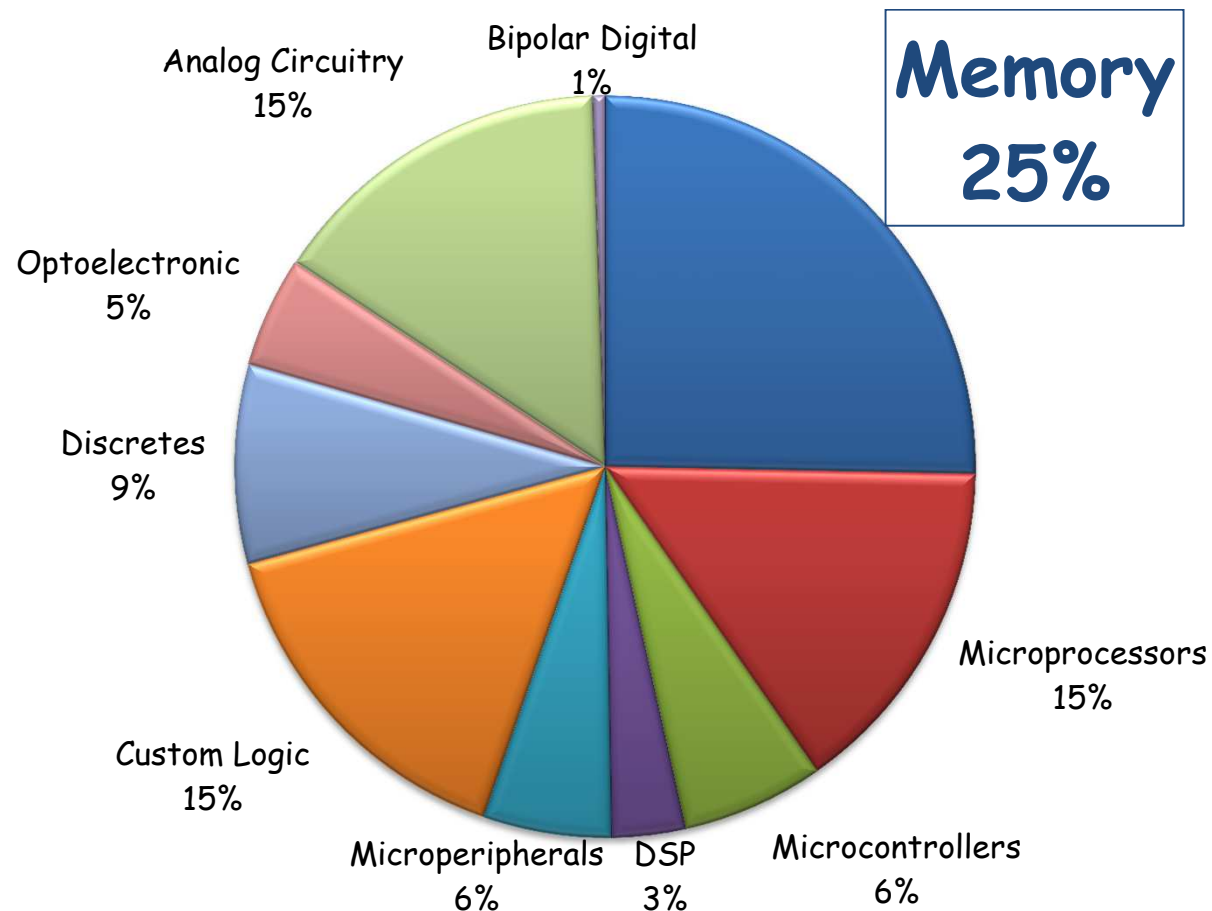
- **Scaletta:**

- Tecnologie usate per realizzare le memorie "volatili": **DRAM** e **SRAM**
- Tecnologie usate per l'accesso alla memoria DRAM: **EDO**, **SDRAM**, **DDR**, ...

Market Share dei Principali Componenti Elettronici

SITUAZIONE RIFERITA ALL'ANNO 2000

SOURCE: Primer on Semiconductors, SG Cowen, 3rd Ed. Jan. 2001



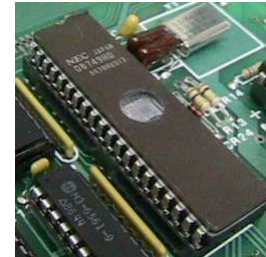
ANNO 2007: MEMORIA CIRCA 37% DEL MERCATO TOTALE PER HIGH PERFORMANCE COMPUTING (SOURCE: BCCRESEARCH.COM)
PREV. 2012: MARKET SHARE COSTANTE, CRESCITA CIRCA 9% ANNUO

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 e' indipendente dalla locazione fisica dell'informazione
 - Memoria principale
 - Due tipi: **Non-Volatili** e **Volatili**

Memorie accesso casuale di tipo Non-Volatile

- 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"
- 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)



Memorie accesso casuale di tipo Volatile

- Conservano i dati solo se alimentate
- Principali tipi di implementazioni:
 - **DRAM** (Dynamic Random Access Memory)
 - ✓ Alta densita', basso consumo, basso costo
 - ☹ Lente, dinamiche → Necessita' di essere "rinfrescata" periodicamente
 - **SRAM** (Static Random Access Memory)
 - ✓ Veloci, statiche → basta alimentare il chip per non perdere i dati
 - ☹ Bassa densita', alto consumo, alto costo



Approfondimento: v. slide "Costi e Prestazioni delle Memorie"

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

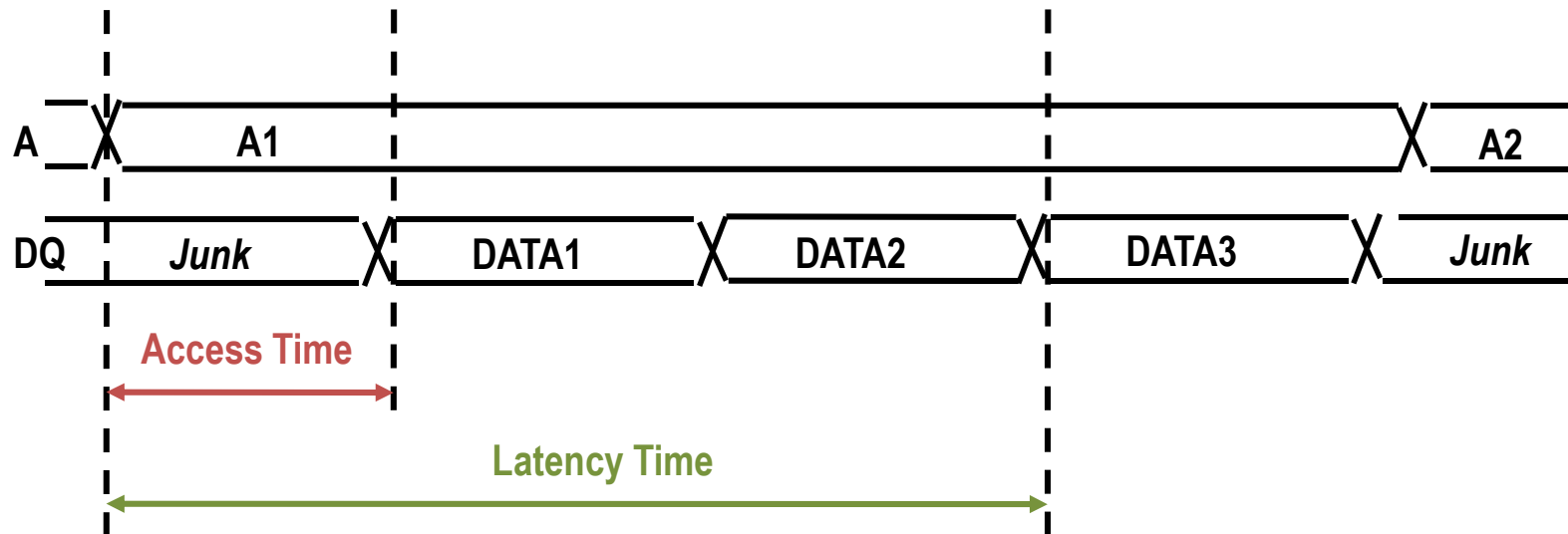
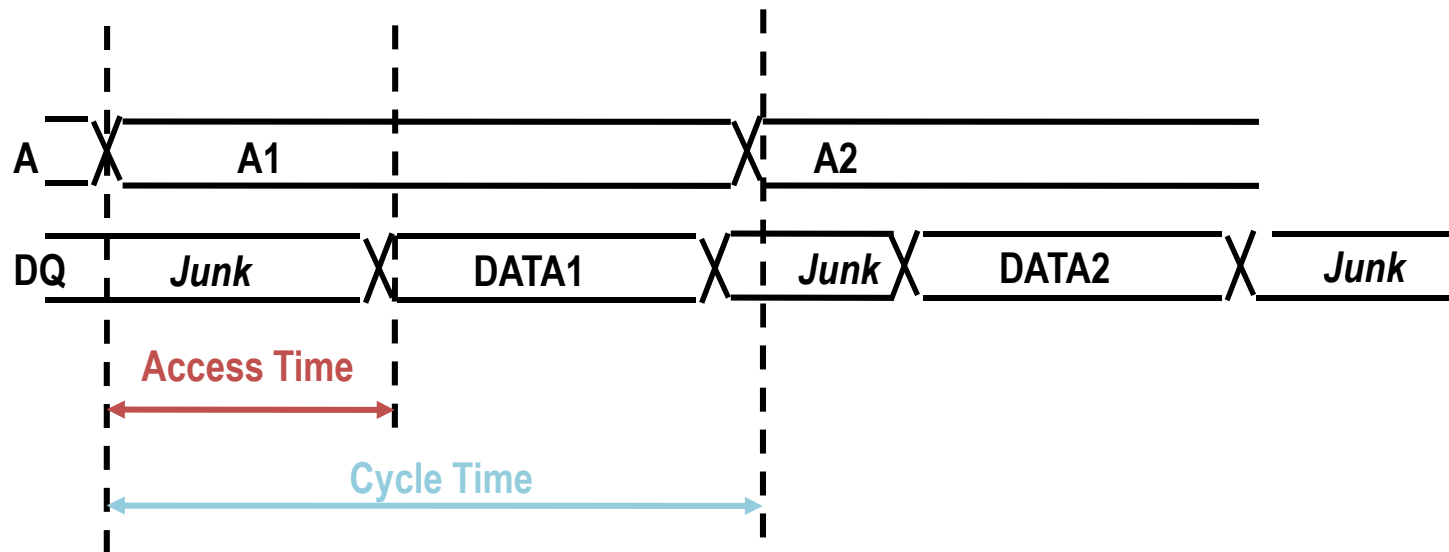
t_l = tempo di latenza (Latency Time)

- 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)

Tempi caratteristici delle memorie

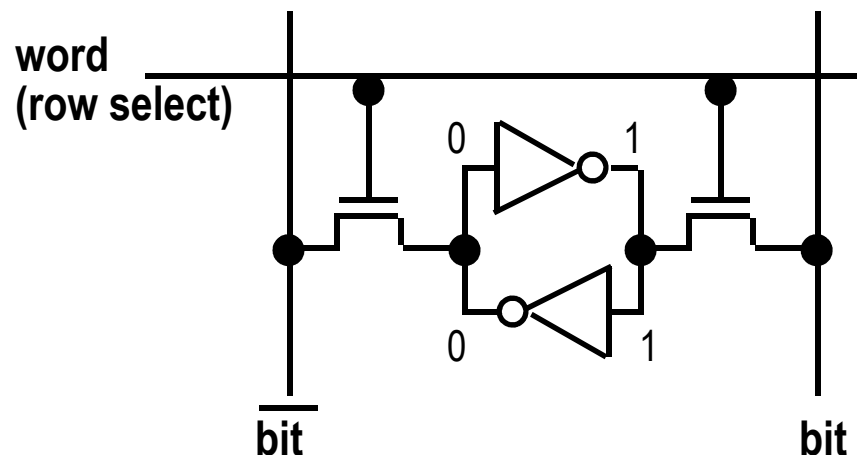


DRAM versus SRAM

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

Cella di RAM Statica (SRAM)

Cella a 6 Transistor

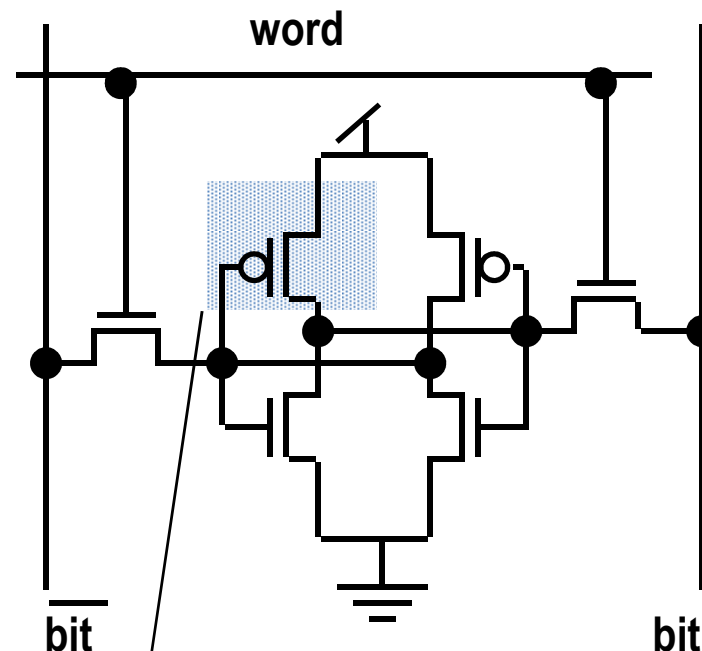


Scrittura

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

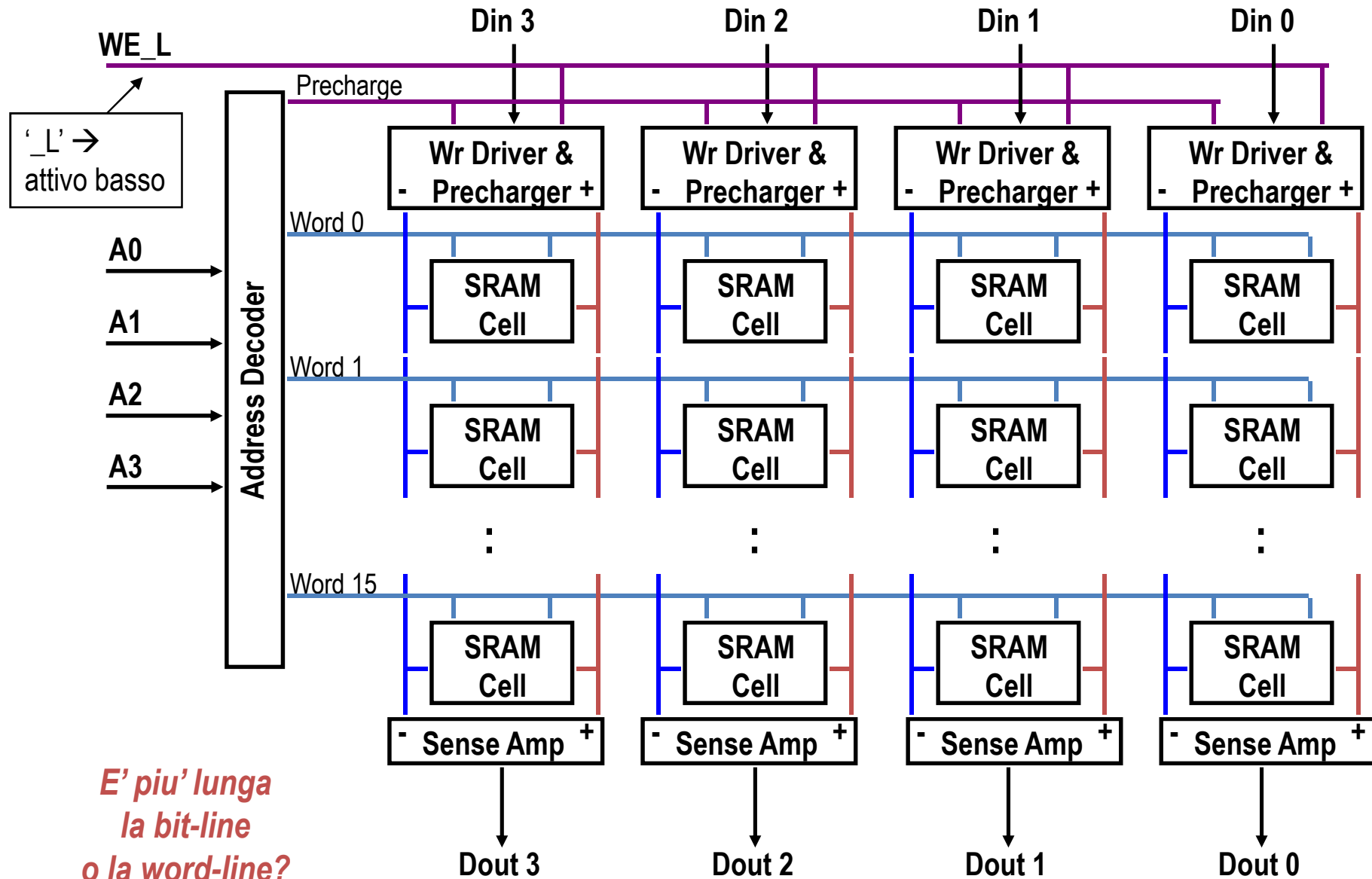
Lettura

1. Precaricare bit e /bit a V_{dd} *
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

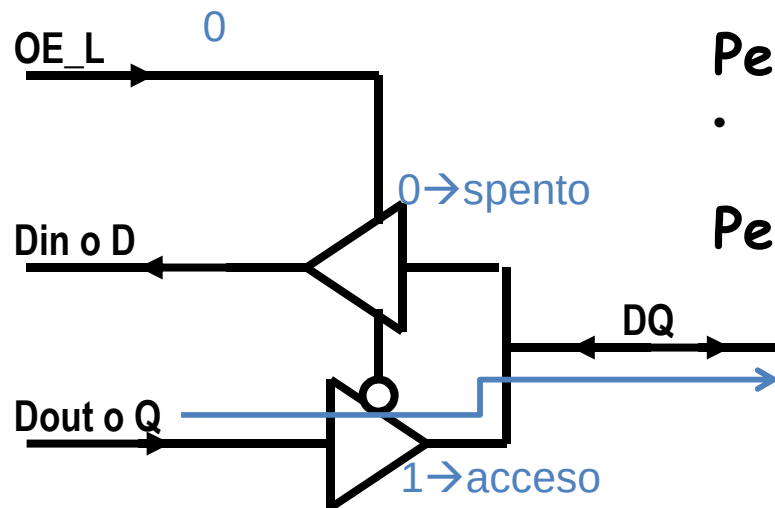


Puo' essere sostituito da una resistenza di pull-up per risparmiare area
→ Cella a 5 transistor

Organizzazione delle celle SRAM: 16-word x 4-bit



SRAM: Bus dati



Per Din e Dout si usa un unico filo

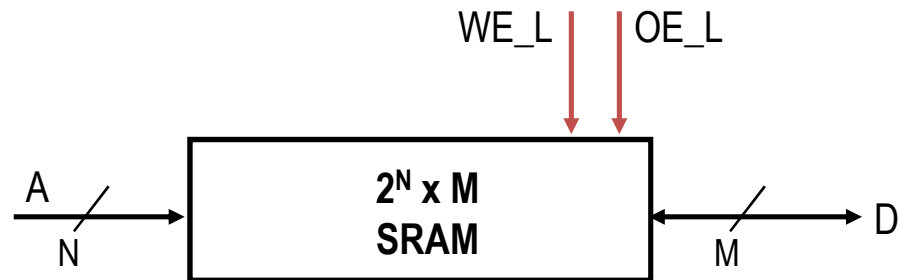
- Il motivo principale e' perche' sul bus dati posso avere scambio in entrambe le direzioni (lettura o scrittura)

Per decidere la direzione si usa /OE:

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

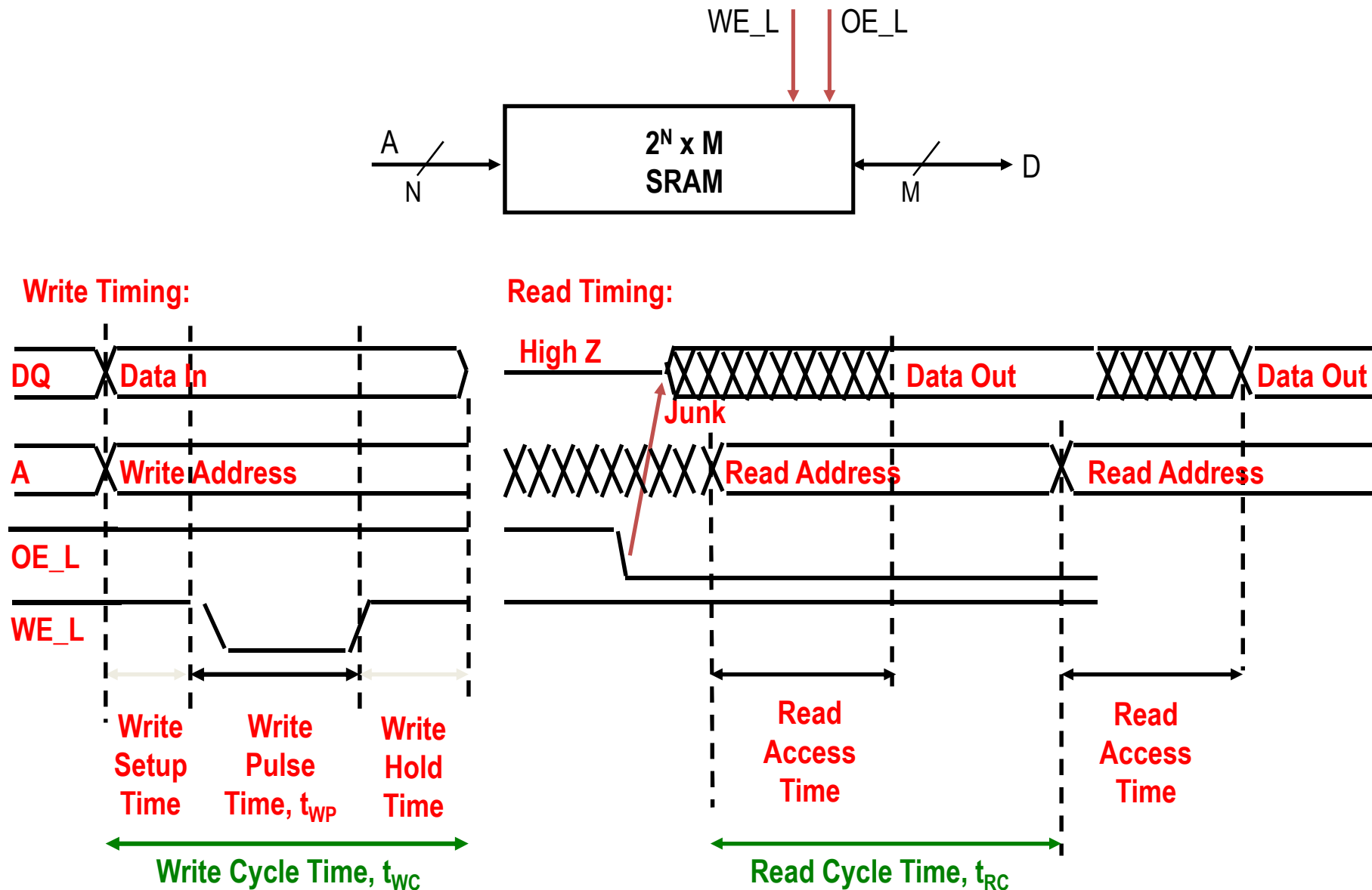
- Le possibili combinazioni sono:
 - **Scrittura:**
 - **WE_L** attivo (basso), **OE_L** disattivo (alto) → **DQ** e' un ingresso
 - **Lettura:**
 - **WE_L** disattivo (alto), **OE_L** attivo (basso) → **DQ** e' un'uscita
 - Evitare di attivare contemporaneamente i segnali **WE_L** e **OE_L**
 - Il risultato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

SRAM: Diagramma Logico



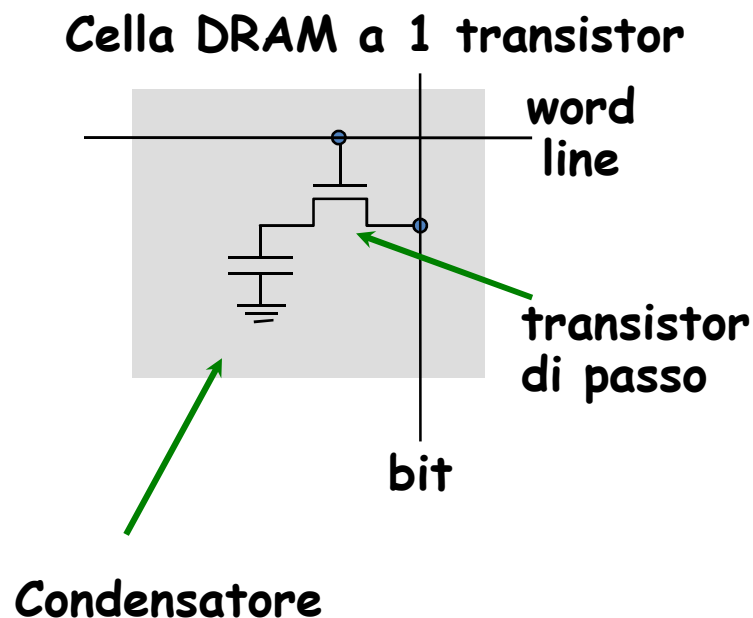
- La memoria ha una capacita' di 2^N word ad M -bit
- Altri segnali sempre presenti
 - CE_L → Chip Enable
 - V_{cc} , GND → Alimentazione

SRAM: Temporizzazioni



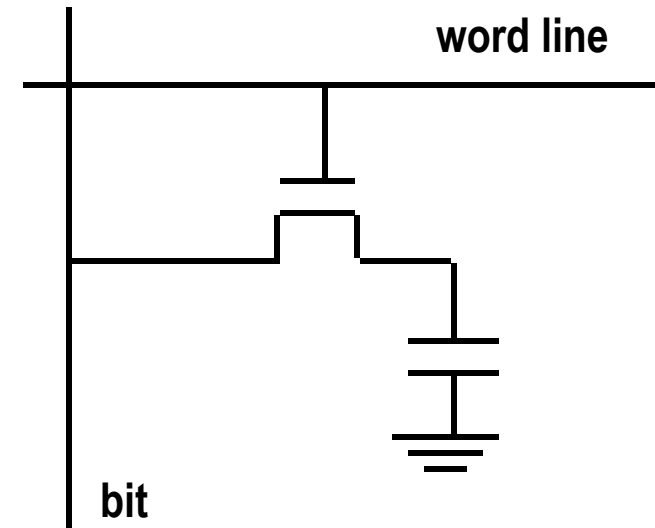
Cella di RAM dinamica (DRAM)

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



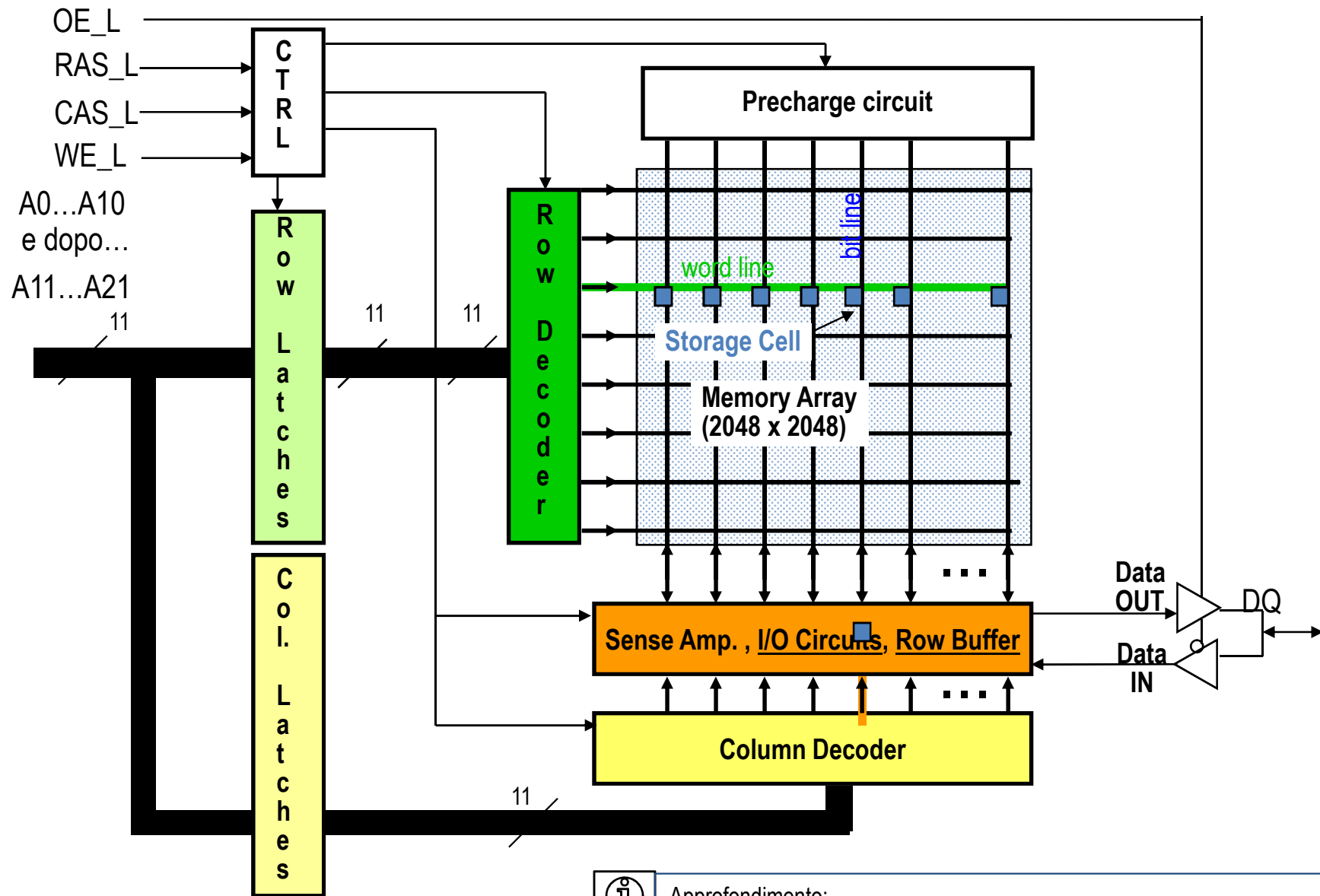
DRAM: Funzionamento della cella a 1 transistor

- **Scrittura:**
 1. Caricare la bit-line col dato
 2. Selezionare la riga (word-line)
- **Lettura:**
 1. Precaricare la bit-line a $V_{dd}/2$
 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 tutte le celle
 - Ripristino il contenuto di ogni cella



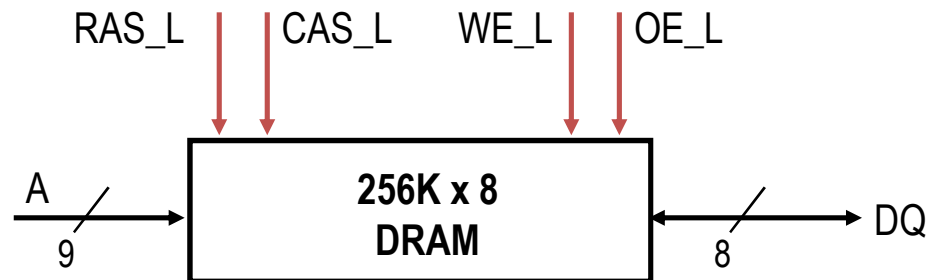
N.B. La cella ha
un solo filo di accesso
(word-line)
→ separazione ROW/COL

DRAM: Organizzazione logica di memoria a 4 Mbit



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

DRAM: Diagramma Logico

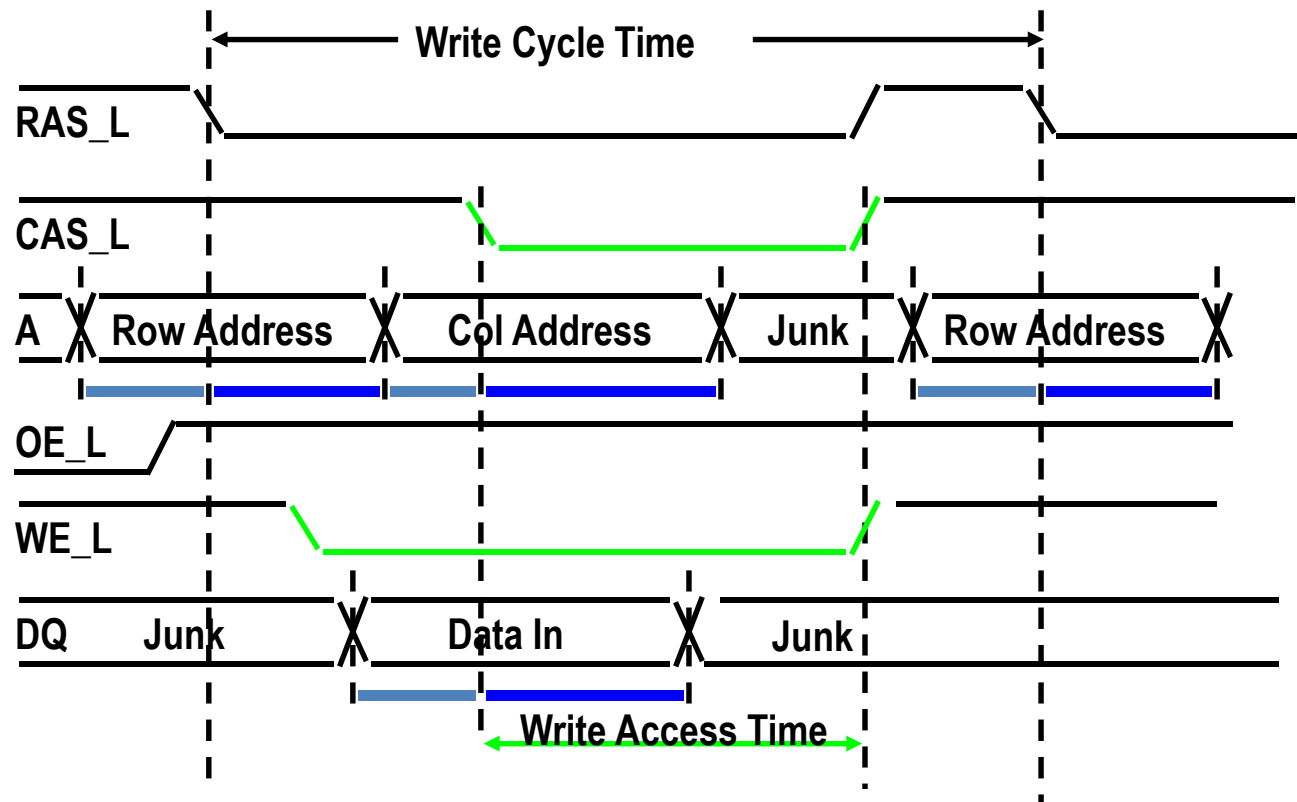


- Din e Dout sono multiplexati su DQ:
 - **Scrittura:**
 - WE_L attivo (basso), OE_L disattivo (alto) → DQ e' un ingresso
 - **Lettura:**
 - WE_L disattivo (alto), OE_L attivo (basso) → DQ e' un'uscita
- Gli indirizzi di Riga e di Colonna condividono i pin A
 - RAS_L va basso → i latch di riga memorizzano i pin A
 - CAS_L 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'e' e invece di DQ ci sono Din e Dout

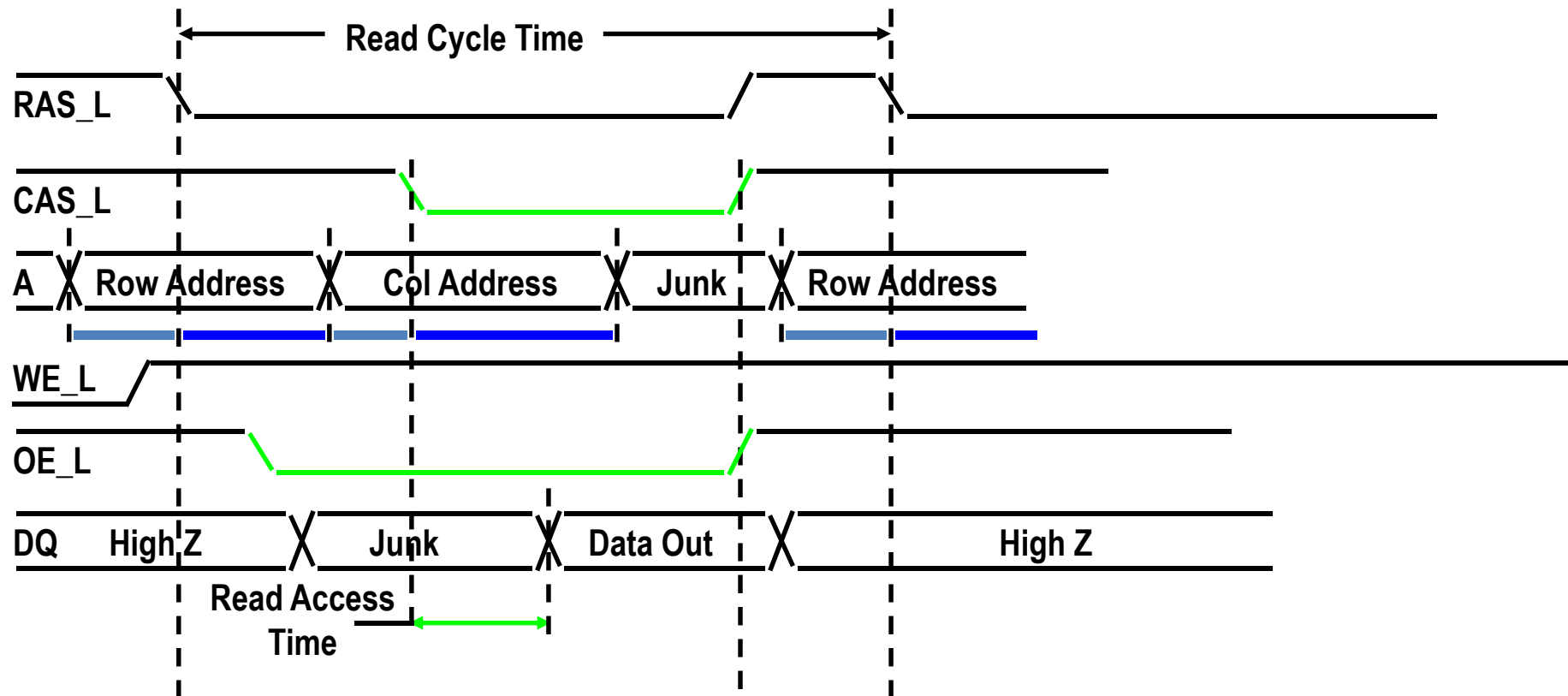
DRAM: Ciclo di scrittura

- L'accesso inizia con l'attivazione di RAS_L



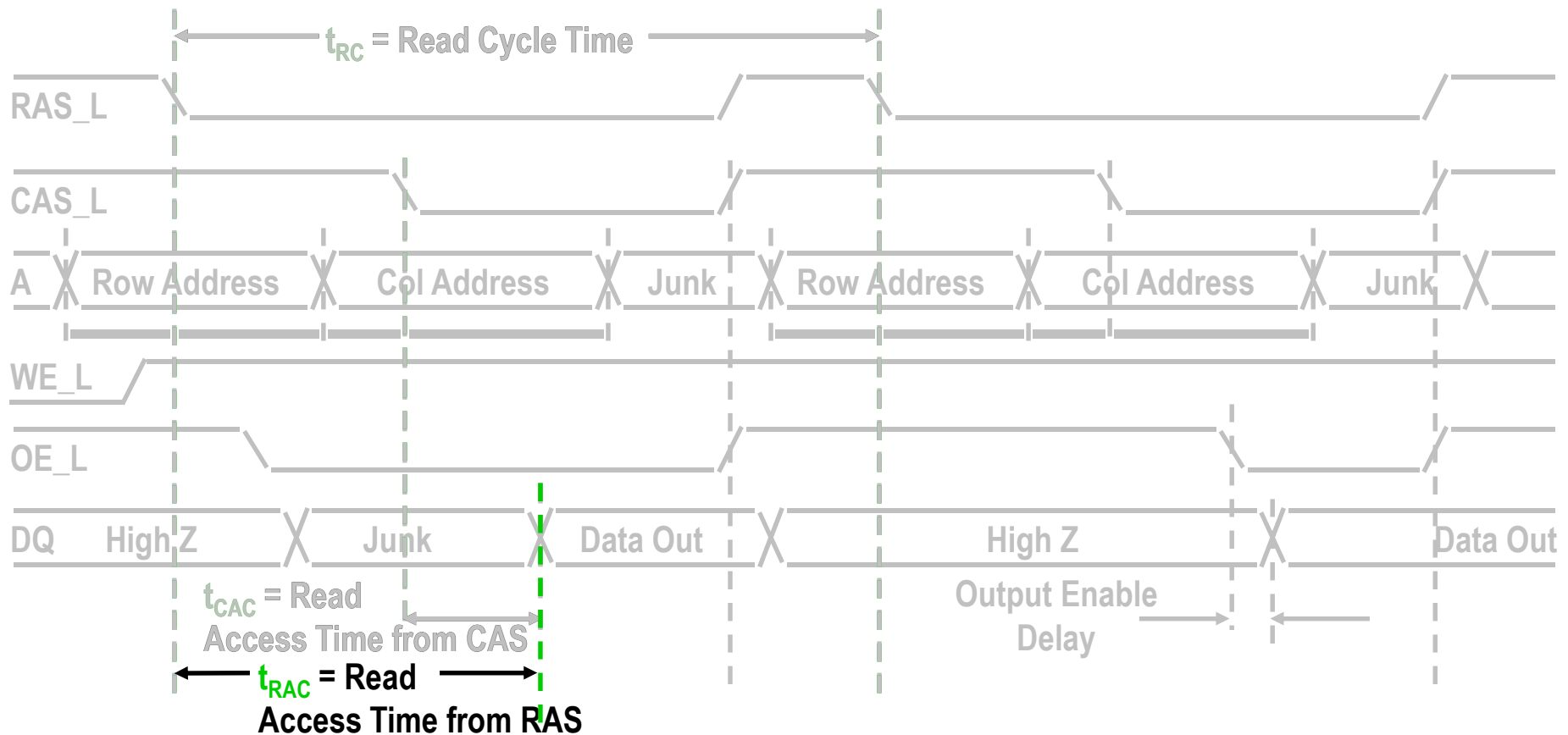
DRAM: Ciclo di lettura

- L'accesso inizia con l'attivazione di RAS_L



t_{RC} , t_{RAC} e t_{CAC}

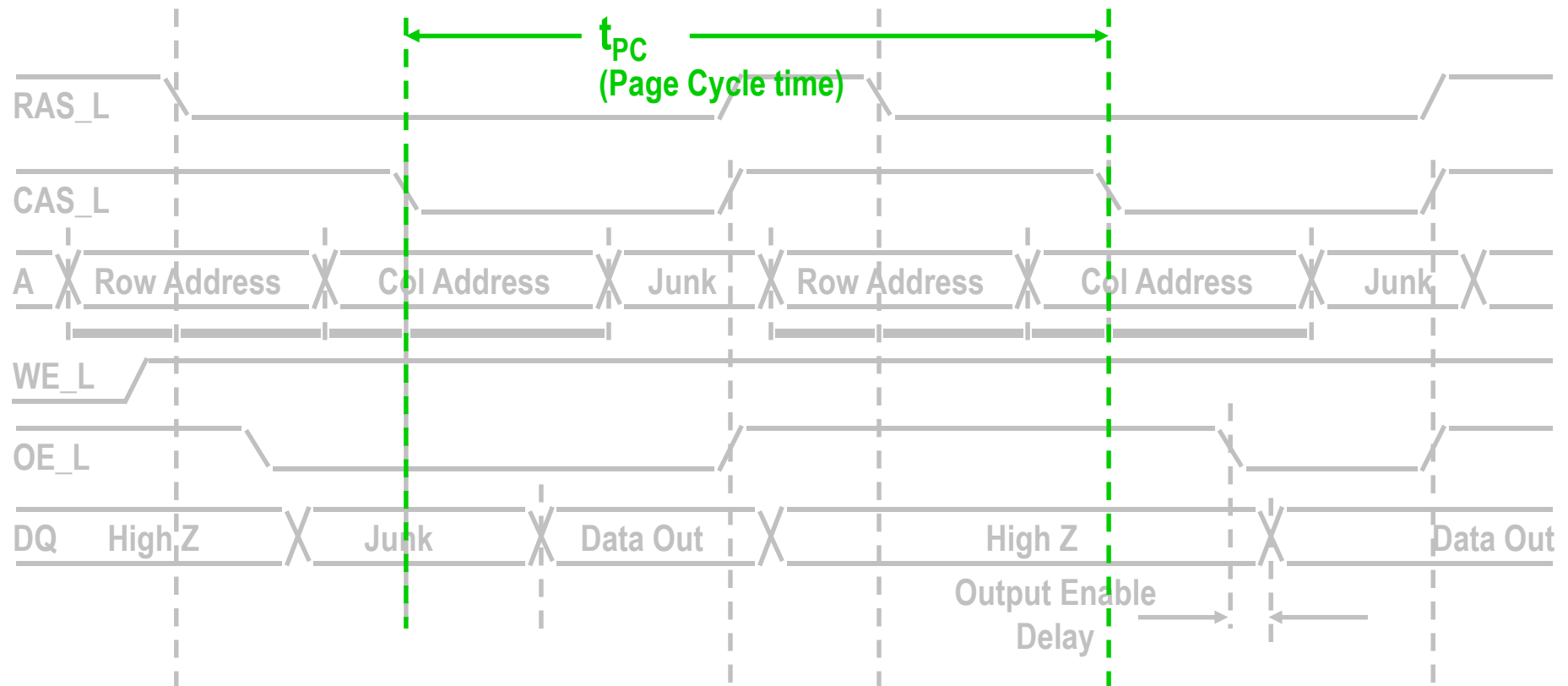
- t_{RC} : tempo minimo fra due accessi a righe
 $t_{RC} = 110 \text{ ns}$ per una DRAM a 4Mbit
- t_{RAC} (e t_{CAC}): tempo minimo fra discesa di RAS (CAS) e dati validi
 t_{RAC} e' data come "velocita' della DRAM" ($t_{RAC}=60\text{ns}$ con $t_{RC}=110\text{ns}$)



t_{PC}

- t_{PC} : tempo minimo fra l'inizio dell'accesso a una colonna e l'inizio dell'accesso alla colonna successiva

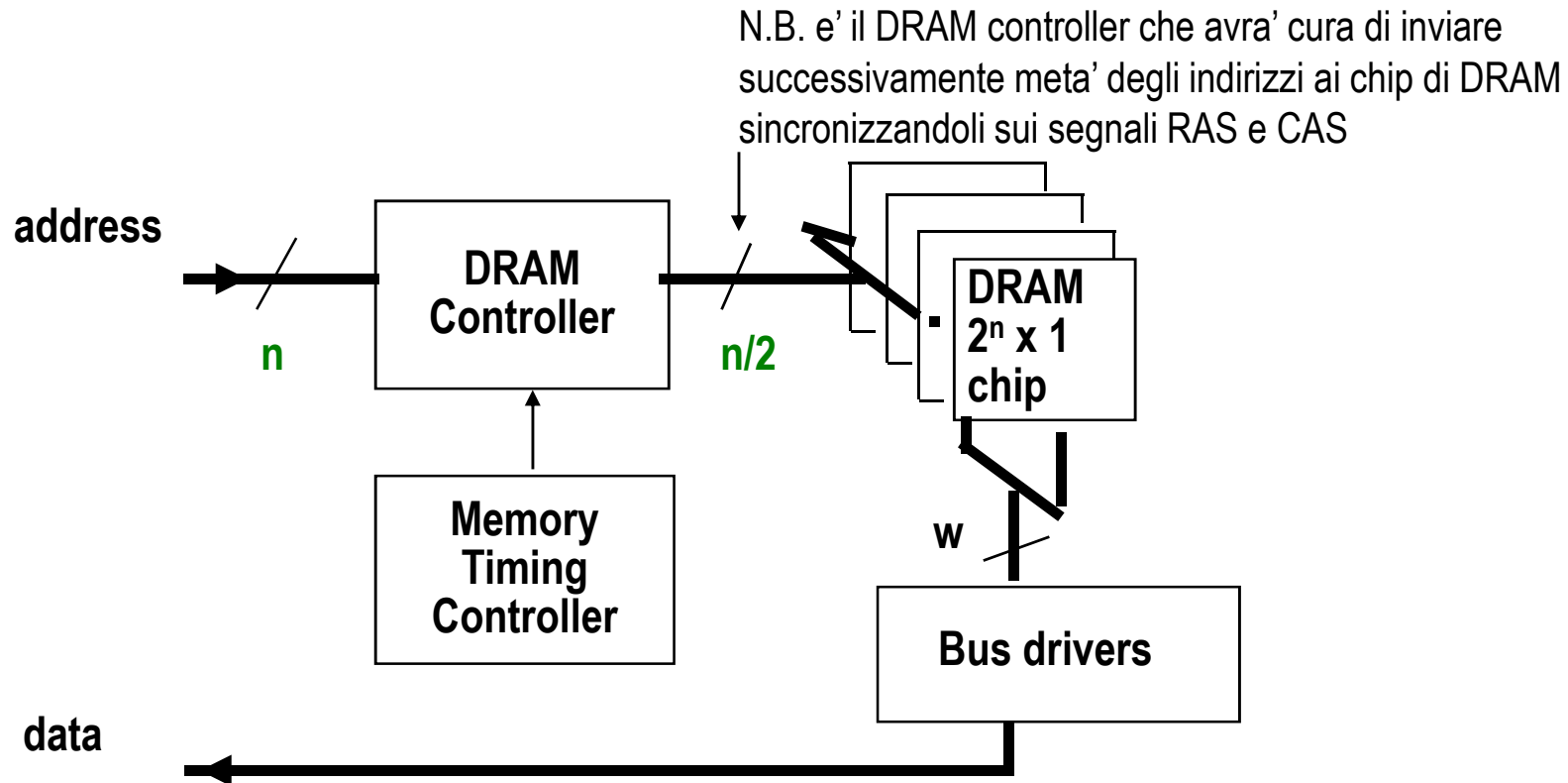
35 ns per una DRAM a 4Mbit e con t_{RAC} di 60 ns



DRAM: Prestazioni

- Una DRAM da 60 ns (t_{RAC}) puo'
 - Effettuare un accesso a righe distinte ogni 110 ns (t_{RC})
 - Effettuare un accesso a colonna in (t_{CAC}) 15 ns, ma il tempo fra due accessi a colonna deve essere almeno 35 ns (t_{PC}).
 - I ritardi di indirizzamento esterni e tempi di "turnaround" sul bus fanno si che questo tempo salga a 40-50 ns
- Questi tempi non includono il tempo per inviare l'indirizzo fuori dal microprocessore (ritardo di indirizzamento) ne' il ritardo introdotto dal controller della memoria
 - Pilotaggio dei chip di DRAM, controller esterno, bus turnaround, modulo SIMM, pin, ...
 - Per una DRAM da "60 ns" (t_{RAC}) si puo' dire che e' gia' buono un tempo di accesso pari a 180-250 ns

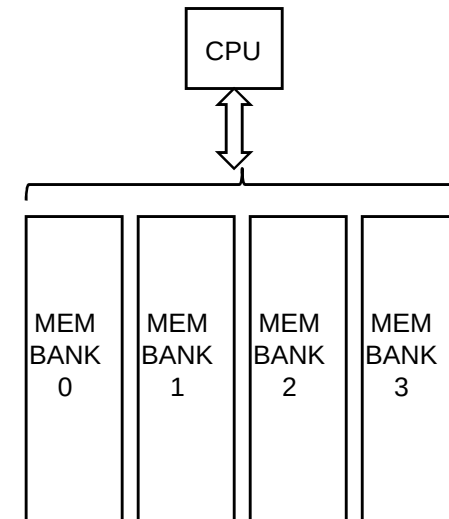
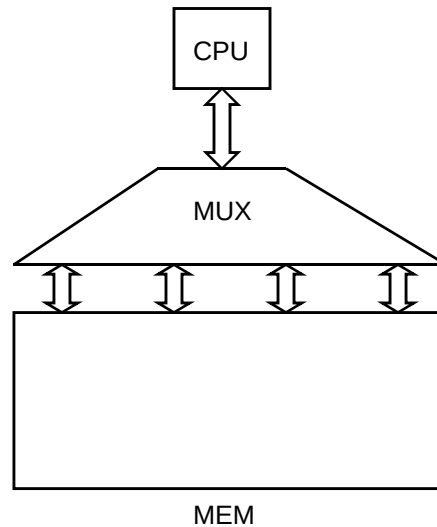
Sistema di Memoria



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

$$T_c = T_{\text{cycle}} + T_{\text{controller}} + T_{\text{driver}}$$

Tecniche per migliorare le prestazioni della memoria



Bus Semplice:

CPU, Bus, Memoria
in tutti i casi un bus
a larghezza fissa
(es. 32 bits)

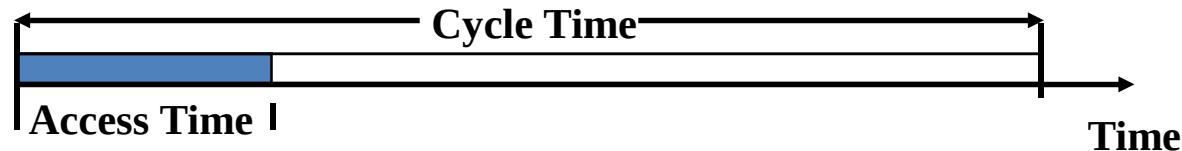
Bus largo:

CPU/Mux: 1 word
Mux/Memoria: N words
(es. 64 & 256 bits)

Interleaving:

CPU, Bus: 1 word
Memoria: N Banchi

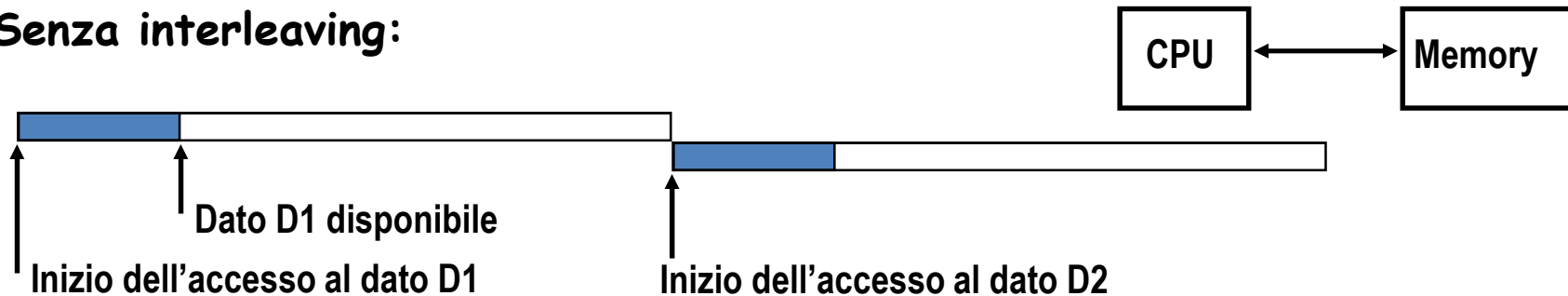
Interleaving degli accessi in memoria



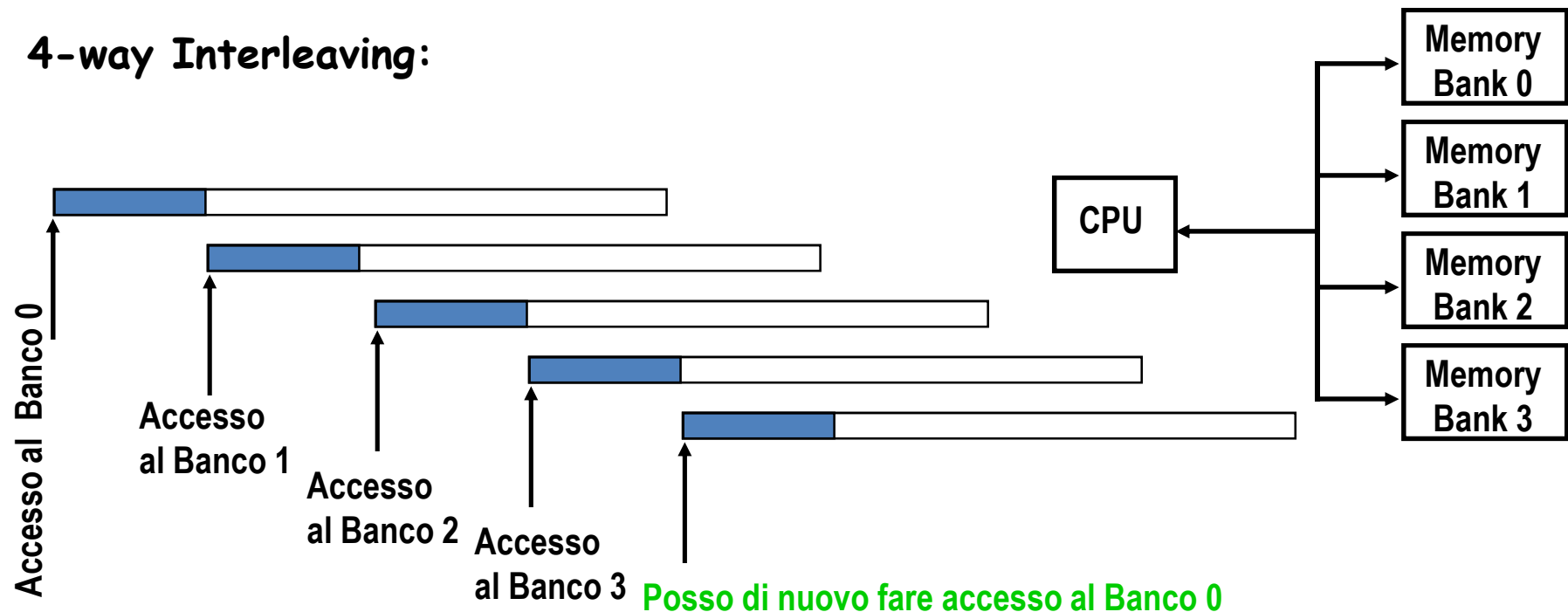
- **Tempo di Ciclo della DRAM (Read/Write)**
 - Indica quanto frequentemente posso fare accessi
- **Tempo di Accesso della DRAM (Read/Write)**
 - Indica quanto tempo occorre per ottenere il dato rispetto all'inizio dell'accesso
- **Il tempo di Ciclo della DRAM (Read/Write) e' molto maggiore del suo Tempo di Accesso DRAM**
 - Almeno di un fattore 2:1
- **Perche' non iniziare un nuovo accesso mentre sto attendendo il dato precedente?**

Increasing Bandwidth - Interleaving

Senza interleaving:



4-way Interleaving:



Confronto fra le tecniche viste

- **Modello dei tempi di accesso**
 - 1 ciclo per inviare l'indirizzo
 - 6 cicli per accedere al dato
 - 1 ciclo per inviare (indietro) il dato
 - Il blocco minimo di trasferimento sia pari a 4 word

- *Bus semplice* $= 4 \times (1+6+1)$ $= 32$ cicli
- *Bus largo* $= 1 + 6 + 1$ $= 8$ cicli
- *Interleaving* $= 1 + 6 + 4 \times 1$ $= 11$ cicli

Scelta del numero dei banchi

- Il numero di banchi dovrebbe essere circa uguale al numero di periodi di clock per accedere ad una word di un banco
 - Questo consentirebbe di avere un dato ad ogni ciclo di clock
- Problema: la disponibilita' di DRAM a maggiore capacita' rende difficile avere piu' banchi

DRAM: diminuzione del numero di chip nel tempo

(da Pete MacWilliams, Intel)

Generazione di DRAM *La capacità di un chip di DRAM cresce del 60% / anno* →

		'86	'89	'92	'96	'99	'02
		1 Mb	4 Mb	16 Mb	64 Mb	256 Mb	1 Gb
Memoria tipica di un PC	4 MB	32 →	8				
	8 MB		16 →	4			
	16 MB			8 →	2		
	32 MB				4 →	<u>1</u>	
	64 MB				8 →	2	
	128 MB					4 →	<u>1</u>
	256 MB					8 →	2

La memoria tipica
"cresce" del 25-30%
all'anno

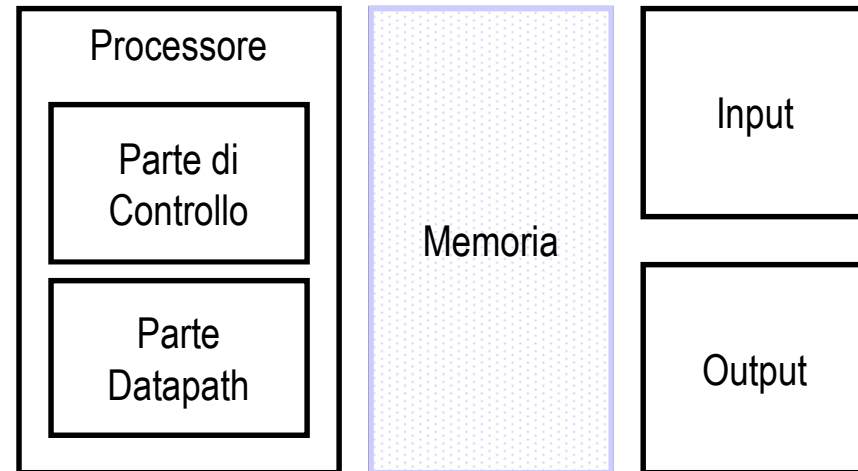


Lezione 17

MECCANISMI DI MIGLIORAMENTO DELLE PRESTAZIONI (1)

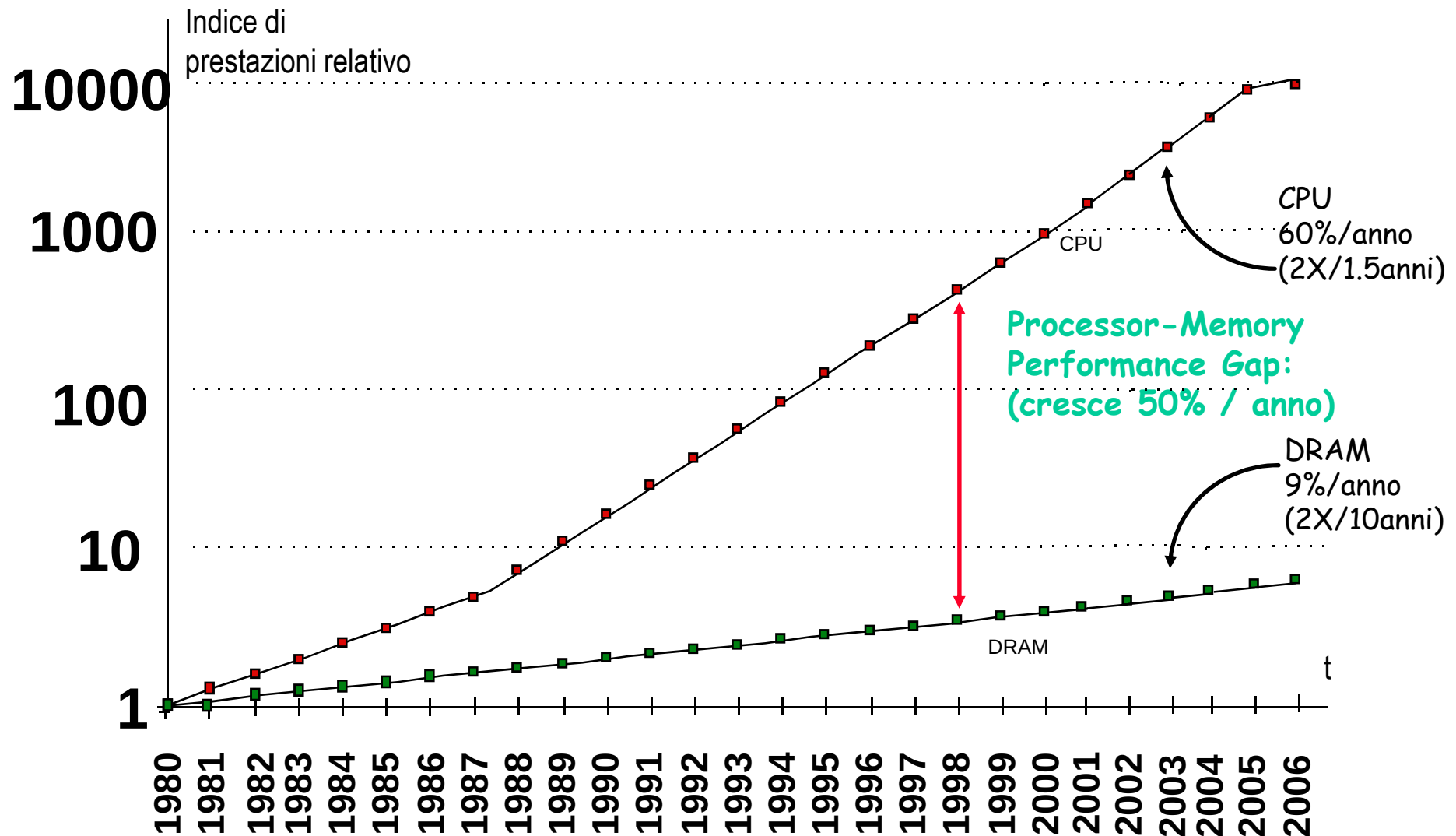
Introduzione alla memoria cache

- **Gerarchia di Memoria**
- **Memoria Cache**



Some figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher. Other material is adapted from CS152 Copyright (C) 2000 UCB

Gap Prestazioni Processore-Memoria



→ Necessita' di nuove architetture per le memorie

Gap Processore-Memoria: un esempio

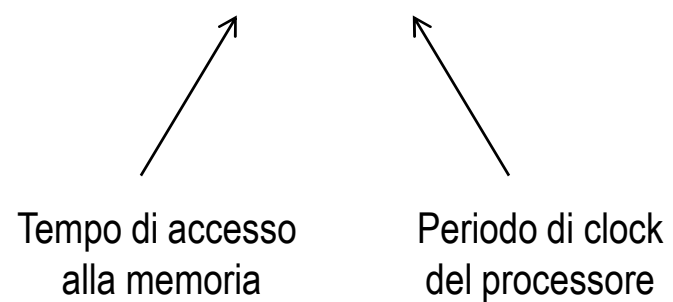
Valutiamo il gap Processore-Memoria in termini di istruzioni per eseguite dal processore durante 1 accesso in memoria

Alpha 7000: 340 ns / 5.0 ns = 68 clk

Alpha 8400: 266 ns / 3.3 ns = 80 clk

Alpha xxxx: 180 ns / 1.7 ns = 108 clk

Tempo di accesso
alla memoria



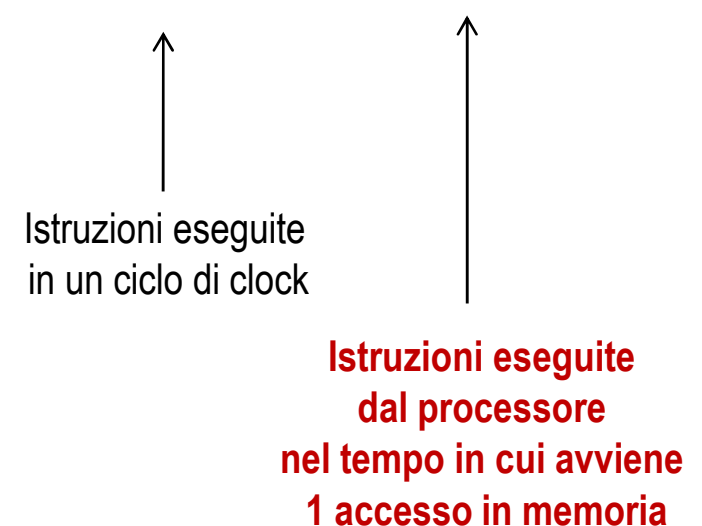
Periodo di clock
del processore

× 2 istr/clk → 136 istr.

× 4 istr/clk → 320 istr.

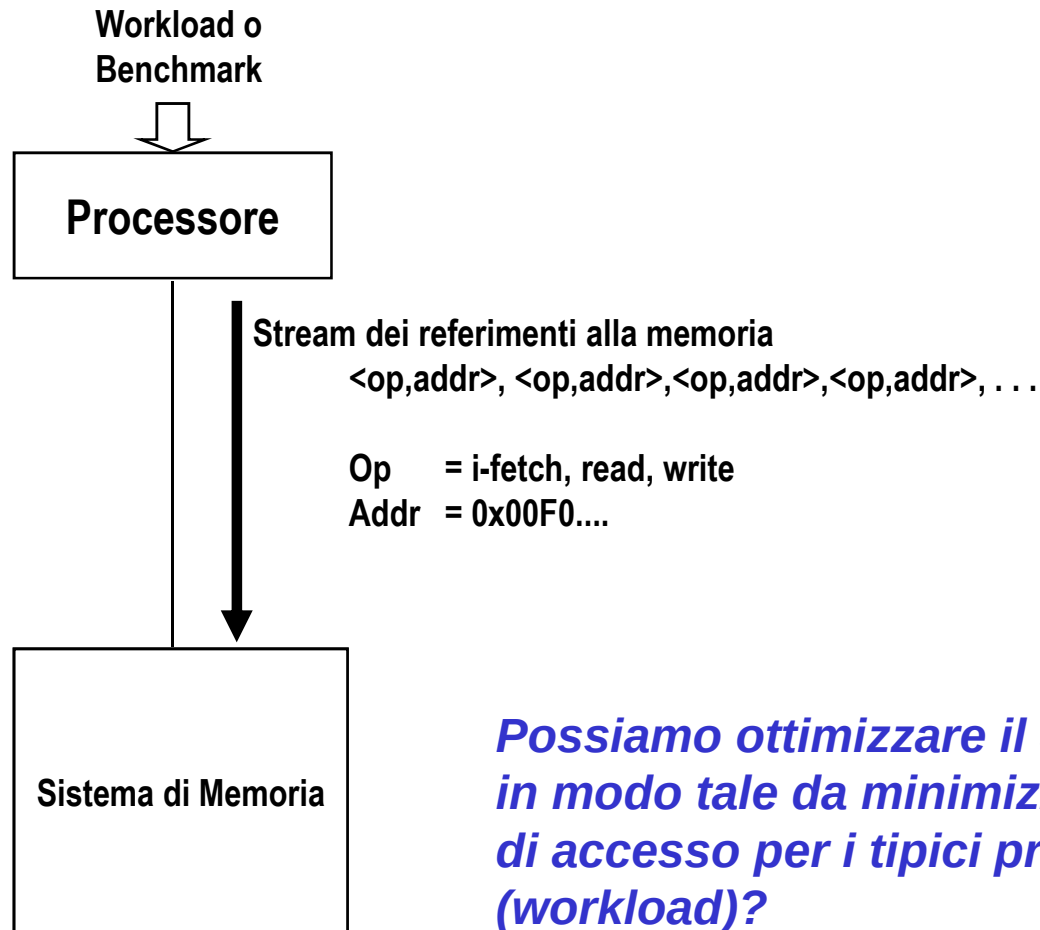
× 6 istr/clk → 648 istr.

Istruzioni eseguite
in un ciclo di clock



Istruzioni eseguite
dal processore
nel tempo in cui avviene
1 accesso in memoria

"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 Localita' 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 Localita'

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

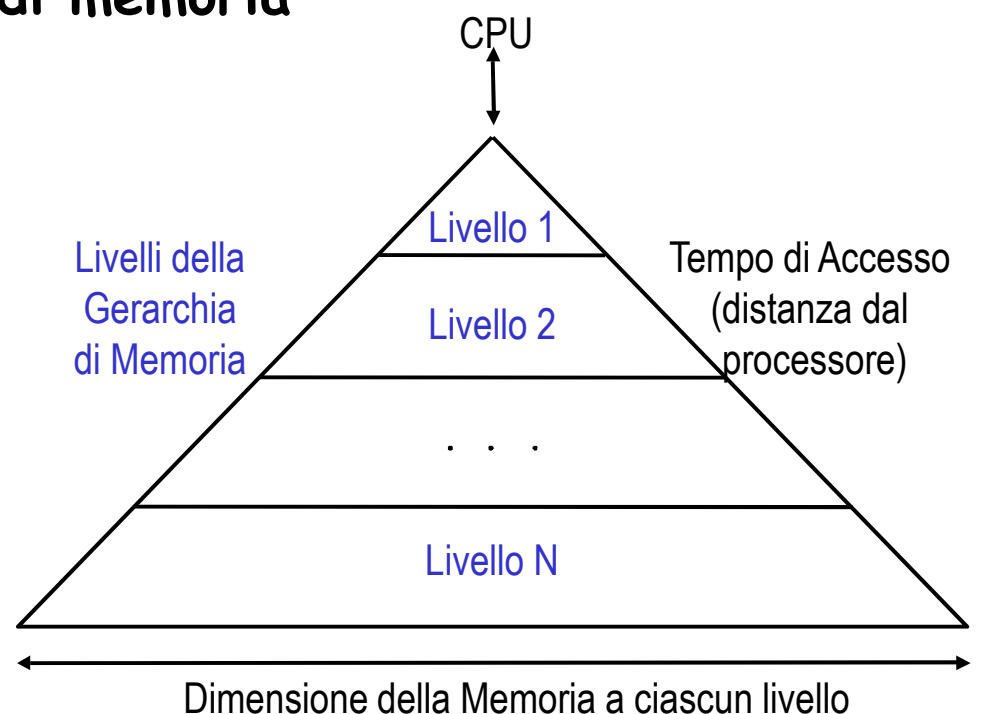
→ **MEMORIA CACHE**

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

→ creo un secondo livello di memoria un po' piu' 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 quantita' di memoria pari a tutta quella disponibile all'ultimo livello (realizzata con la tecnologia piu' economica)
 - Fornisce un tempo di accesso che si avvicina a quello del primo livello (in cui posso usare la tecnologia piu' veloce possibile). Es. Con 2 soli livelli:
 - Primo Livello: Cache, tecnologia SRAM
 - Secondo Livello: Memoria Principale, tecnologia DRAM
 - Matematicamente:
 - h = probabilita' di trovare un dato nel primo livello
 - Maggiore LOCALITA' comporta $h \rightarrow 1$
 - m = probabilita' di trovare un dato nel livello successivo
 - t_h = tempo di accesso al dato dal primo livello
 - t_m = tempo di accesso al dato nel livello successivo
- $\rightarrow t_a = h * t_h + m * t_m$ se $h \rightarrow 1$ allora $t_a \rightarrow t_h$

Terminologia

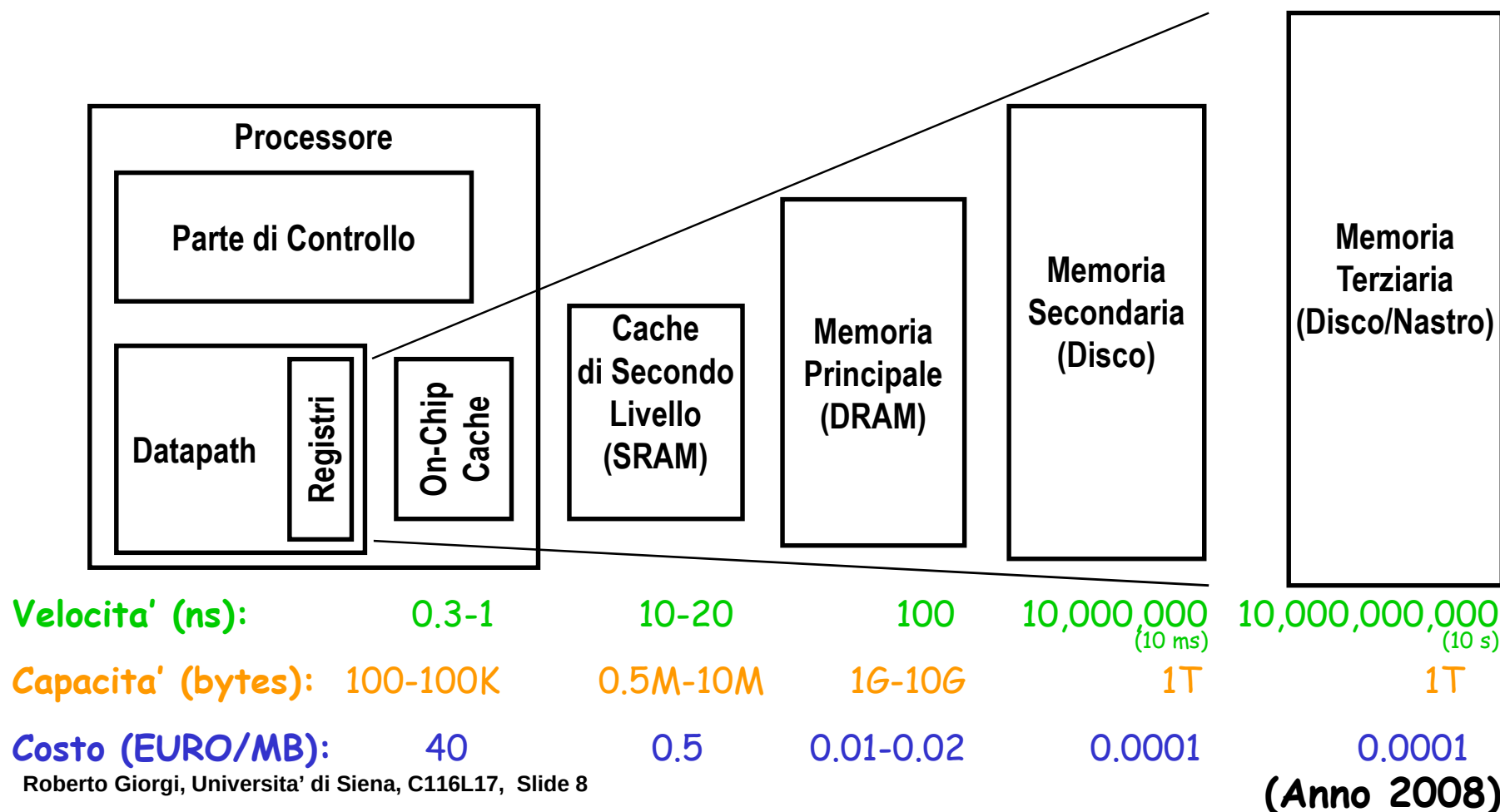
blocco: insieme di byte a locazioni contigue ed indirizzi multipli della dimensione del blocco

hit: il riferimento puo' essere soddisfatto nel livello di memoria immediatamente successivo

miss: il riferimento richiede un accesso a livelli di memoria oltre il successivo (piu' lenti)

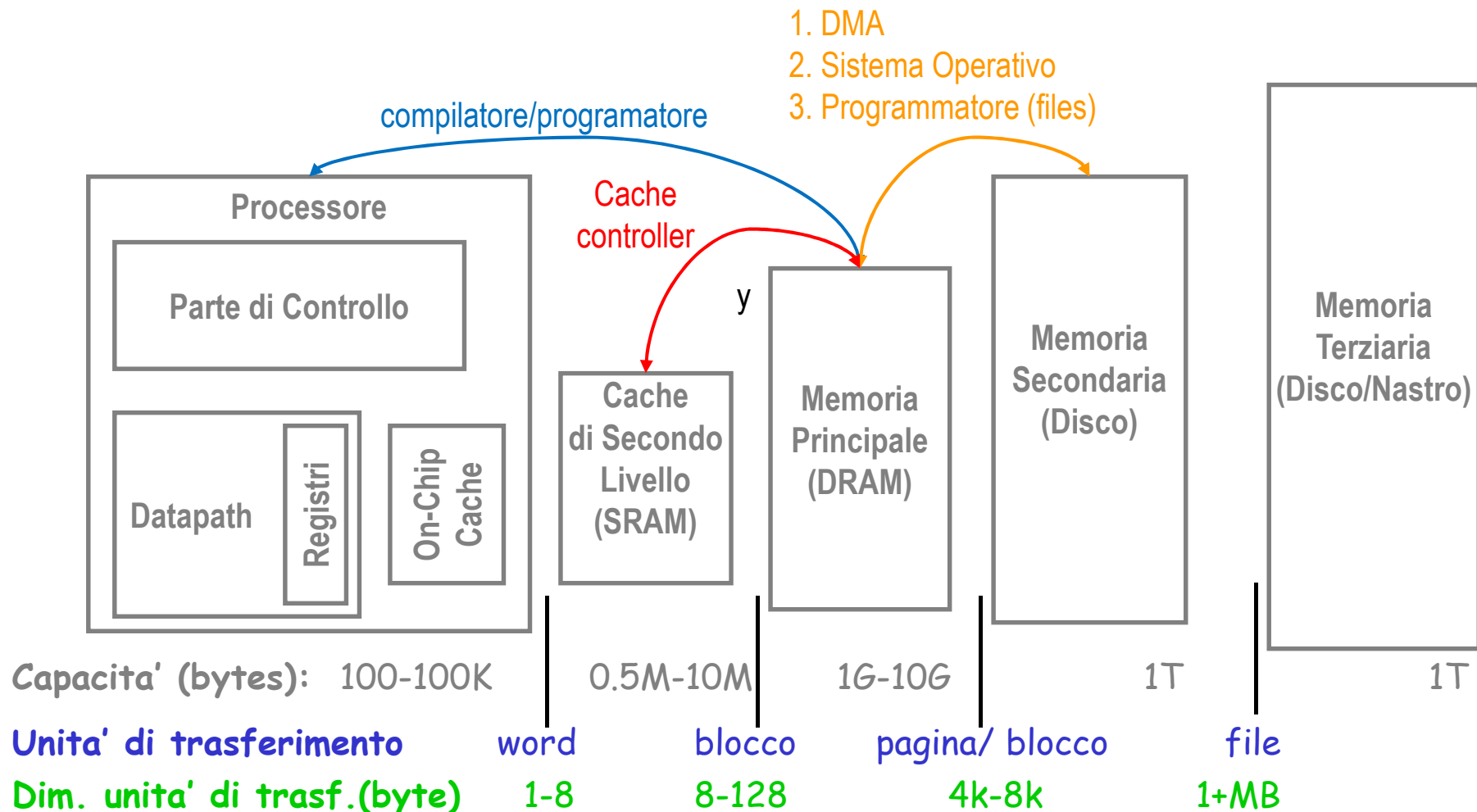
Organizzazione gerarchica della memoria

- Sfruttando il principio di localita' e' possibile:
 - Presentare all'utente tutta la memoria presente al livello della tecnologia meno costosa (es. Disco o nastro)
 - Fornire la velocita' di accesso offerta dalla tecnologia piu' veloce



Unita' di trasferimento ai vari livelli della gerarchia

- Ai livelli piu' alti la dimensione delle unita' di trasferimento e' piu' piccola (trasferimenti piu' veloci)



Memoria Cache

- Scopo della **cache** e' 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 localita' la cache tende a memorizzare i blocchi piu' usati dal processore (working set)
- Normalmente si ha:
 - **B** = dimensione del Blocco di cache in byte = 2^b
 - **C** = dimensione della Cache di cache in byte = 2^c
 - **M** = dimensione della Memoria in byte = 2^m

$$M \gg C$$

ovvero

$$N_M \gg N_C$$

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:

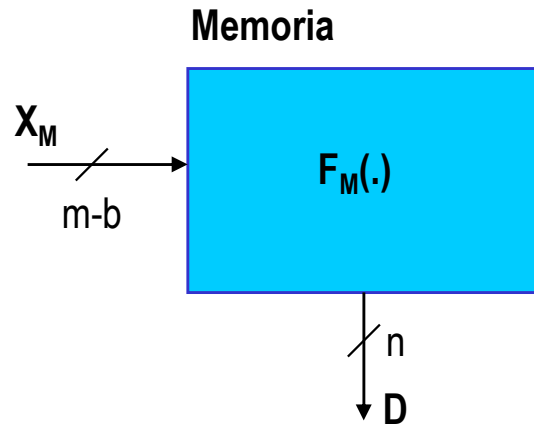
$$W_C \subset W_M$$

W_C = insieme dei blocchi del mio programma presenti in cache

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

Modello "Black-Box" della Cache (1)

X_M = indirizzo del blocco di memoria ($0 \leq X_M < N_M \equiv X / B$ (X indirizzo al byte))

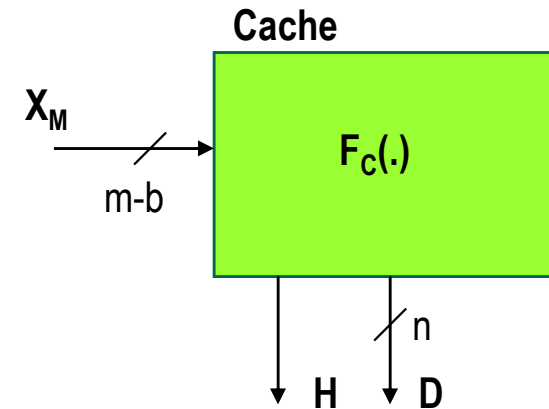


F_M = Funzione di prelievo
del blocco dalla memoria

$$D = F_M(X_M)$$

$F_M : MEM [.]$

$F_C : \dots$ dipende



F_C = Funzione di prelievo
del blocco dalla cache

$$\begin{cases} D = F_C(X_M) = F_M(X_M) & \text{se } H=1 \\ D = \text{niente} & \text{se } H=0 \end{cases}$$

H = Segnale di Hit

1 se il blocco e' in cache

→ diro' anche che ho avuto un **hit**

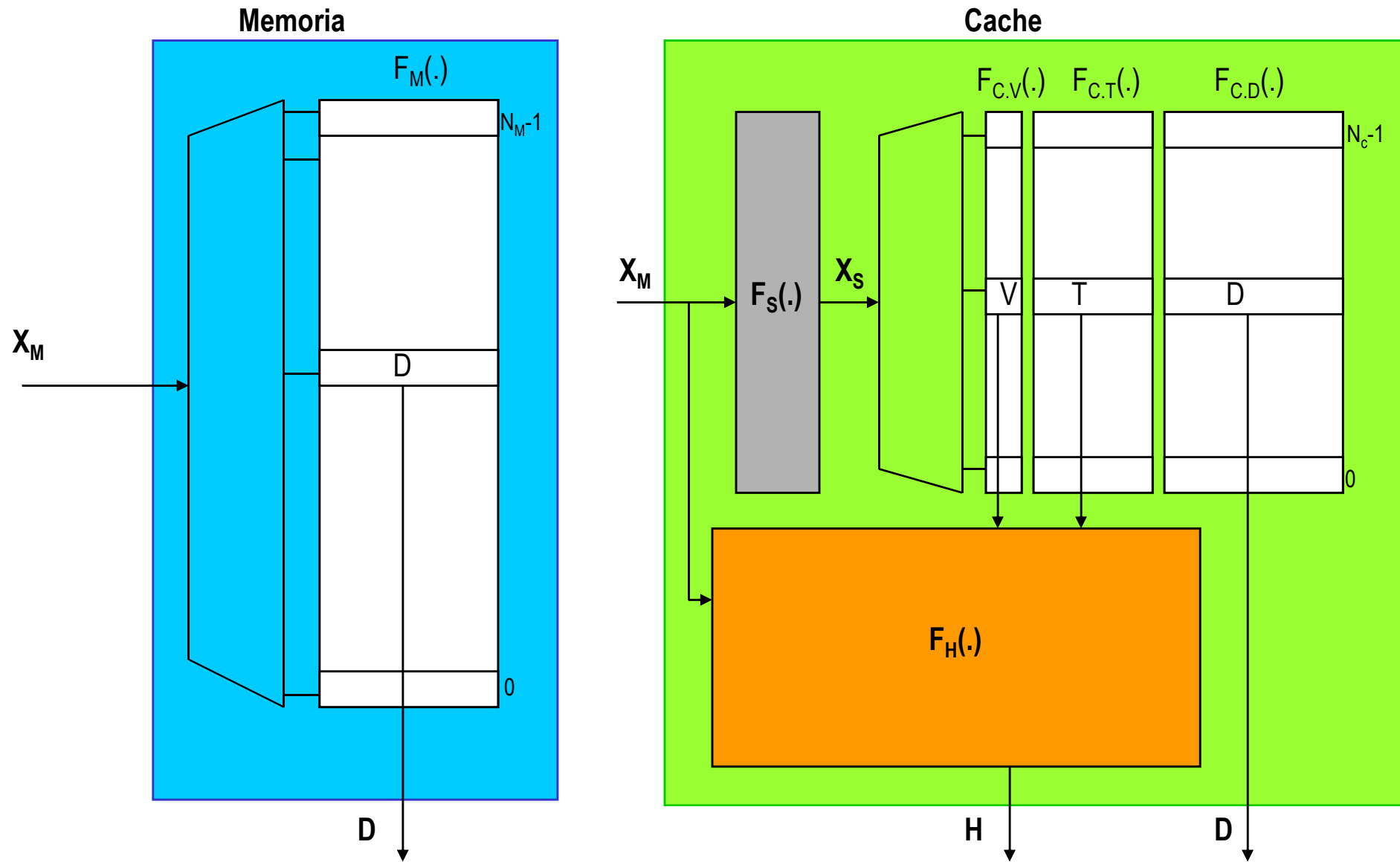
0 se il blocco non e' in cache

→ diro' che ho avuto un **miss**

Funzionamento della Cache (1)

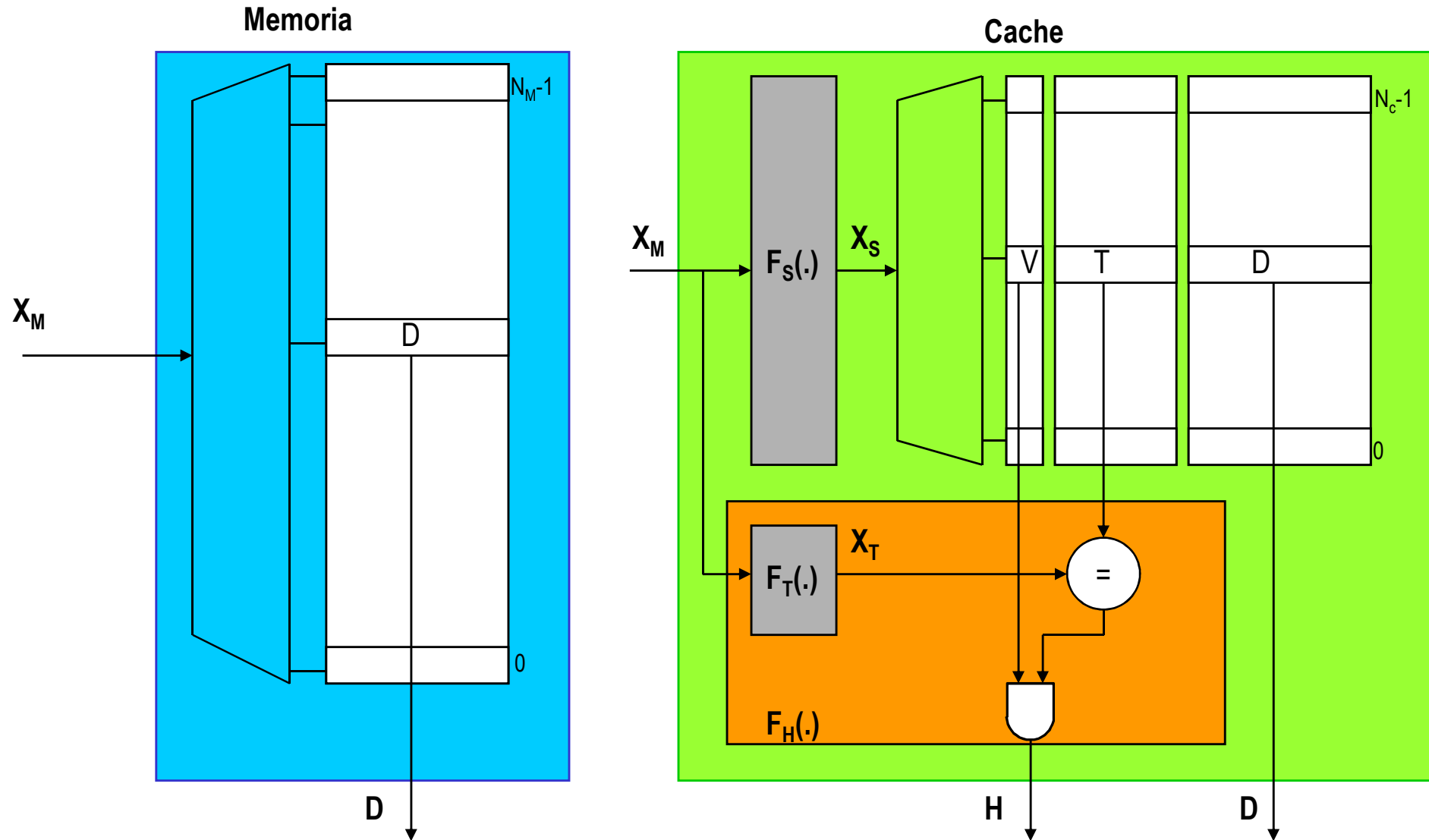
- Come faccio a sapere se il blocco e' in cache?
 - Definisco una "Funzione di Hit" F_H che vale
$$H = F_H(X_M)$$
- Se il blocco e' in cache, come faccio a trovarlo?
 - Per prelevare un blocco dalla memoria facevo $D = F_M(X_M) = \text{MEM}[X_M]$
 - Analogamente, per la cache vogliamo recuperare il dato ad un indirizzo X_S di (un array di memoria) cache (con $0 \leq X_S < N_C$)
$$D = F_{C,D}(X_S) = \text{CACHE}[X_S].D$$
 - Che relazione c'e' fra X_M e X_S (ricordare che $X_S < N_C < N_M$) ?
Per essere sicuri che il dato D (che in memoria e' ad indirizzo X_M) sia recuperabile all'indirizzo di cache X_S ,
occorre memorizzare accanto al dato un etichetta (o "tag"):
$$T = F_{C,T}(X_S) = \text{CACHE}[X_S].T$$
 - Inoltre per essere sicuri che a una data coppia $\langle T, D \rangle$ corrisponda un dato effettivamente inserito in cache,
si aggiunge un bit di validita'
$$V = F_{C,V}(X_S) = \text{CACHE}[X_S].V$$

Modello "Black-Box" della Cache (2)



$F_S: ?$ $F_H: ?$

Modello "Black-Box" della Cache (3)



Funzionamento della Cache (2)

- Affinche' questo meccanismo funzioni e' necessario che si possano ricavare in maniera univoca da X_M
 - X_S per indirizzare il dato D dentro la memoria cache
 - X_T per confrontarlo con il tag T memorizzato nella cache
- L'individuazione del blocco in cache viene quindi fatta combinando due informazioni:

1) Etichetta (o "tag") associata al blocco:

$$X_T = F_T(X_M)$$

2) Indirizzo di localizzazione in cache:

$$X_S = F_S(X_M)$$

- Allora

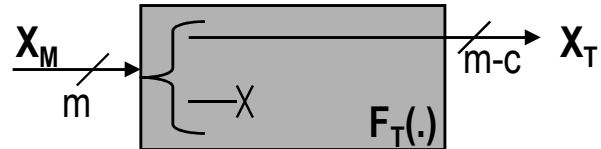
$$F_H : (X_T == T \ \&\& \ V == 1 ? 1 : 0)$$

ovvero

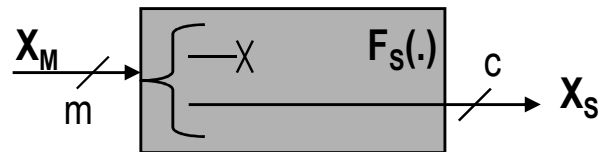
$$F_H : (F_T(X_M) == T \ \&\& \ V == 1 ? 1 : 0)$$

F_T e F_S nel caso di cache ad ACCESSO DIRETTO

$F_T : (\text{div } N_C)$

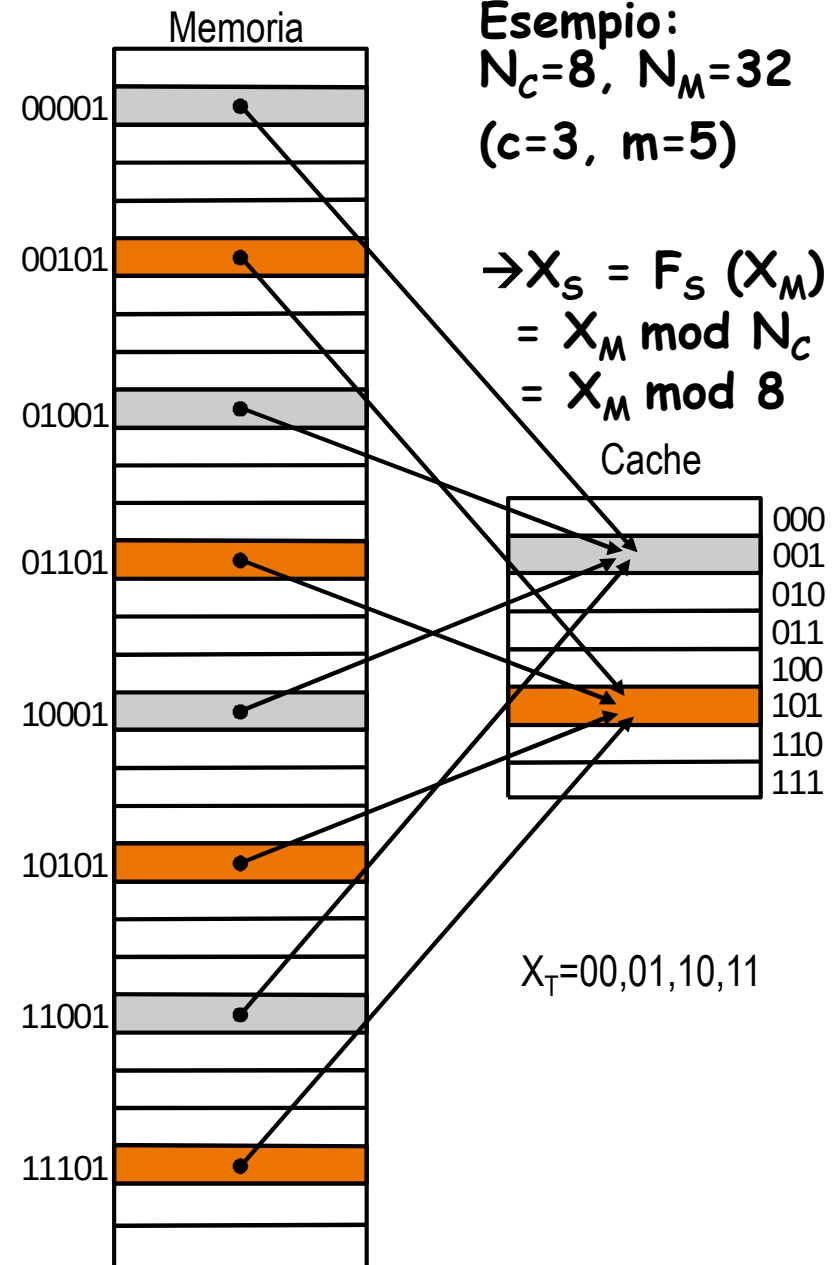


$F_S : (\text{mod } N_C)$



Esempio:
 $N_C=8$, $N_M=32$
 $(c=3, m=5)$

$$\begin{aligned} \rightarrow X_S &= F_S(X_M) \\ &= X_M \bmod N_C \\ &= X_M \bmod 8 \end{aligned}$$



Schema logico di una cache ad accesso diretto

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

X indirizzo al byte

$$X_M = X / B$$

$$= X / 2^2$$

$$X_T = X_M / N_C$$

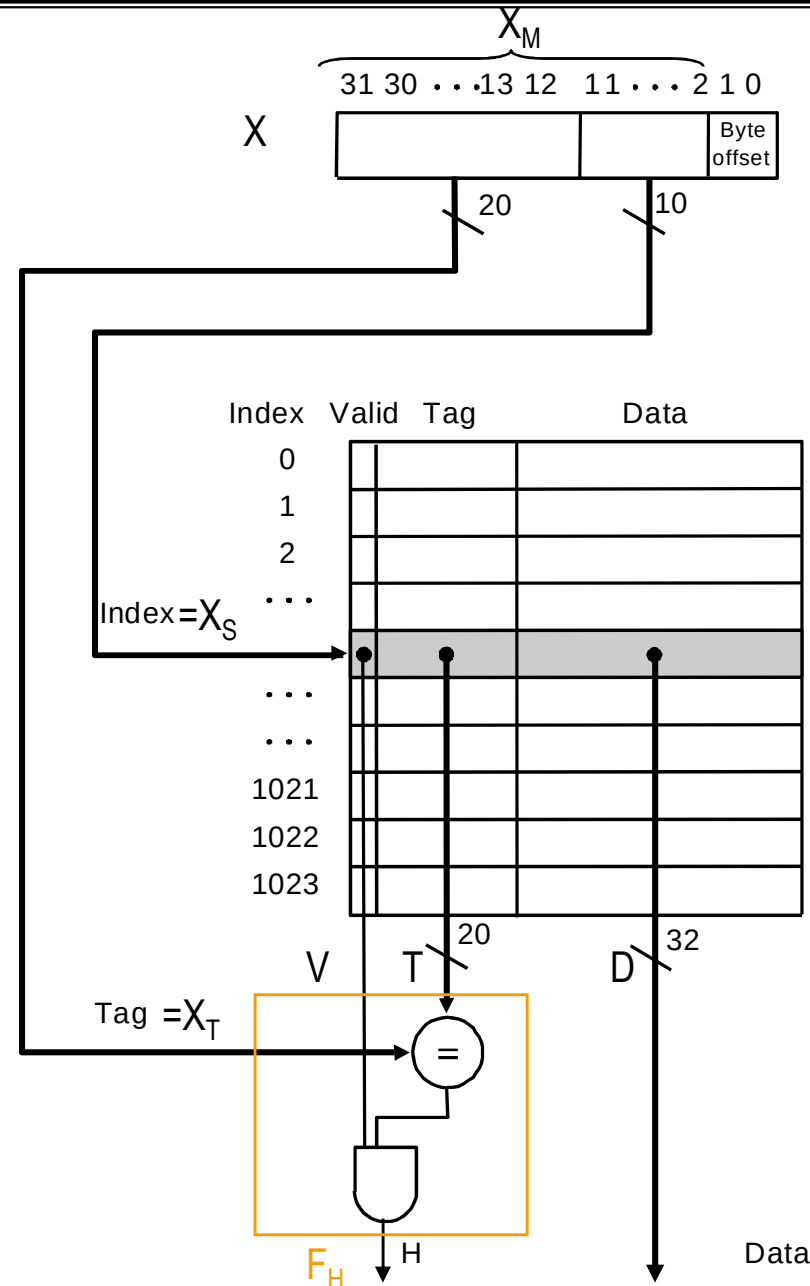
$$= X_M / (C/B)$$

$$= X_M / 2^{10}$$

$$X_S = X_M \bmod N_C$$

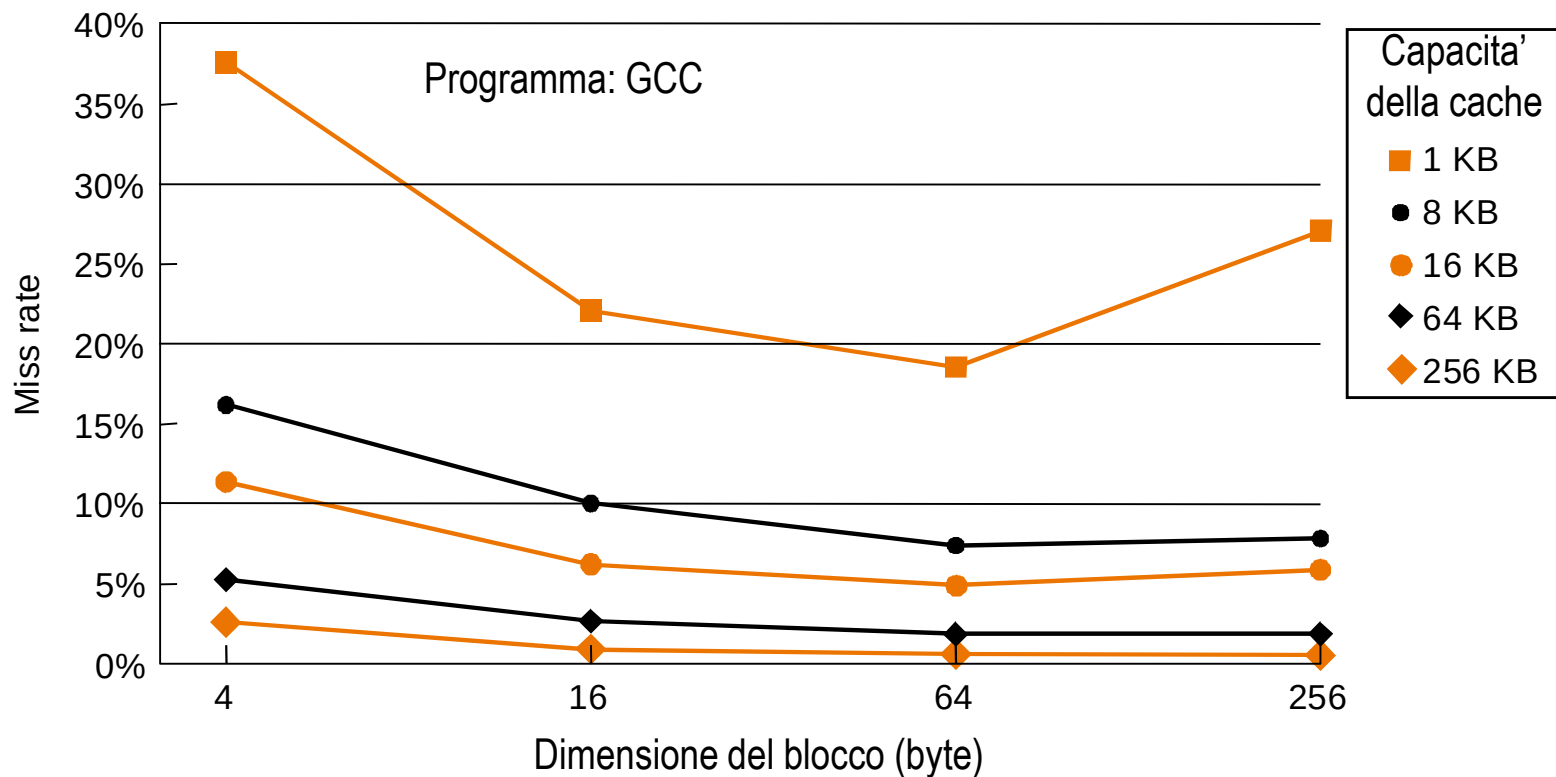
$$= X_M \bmod C/B$$

$$= X_M \bmod 2^{10}$$



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 piu' grandi: sfrutto la localita' spaziale

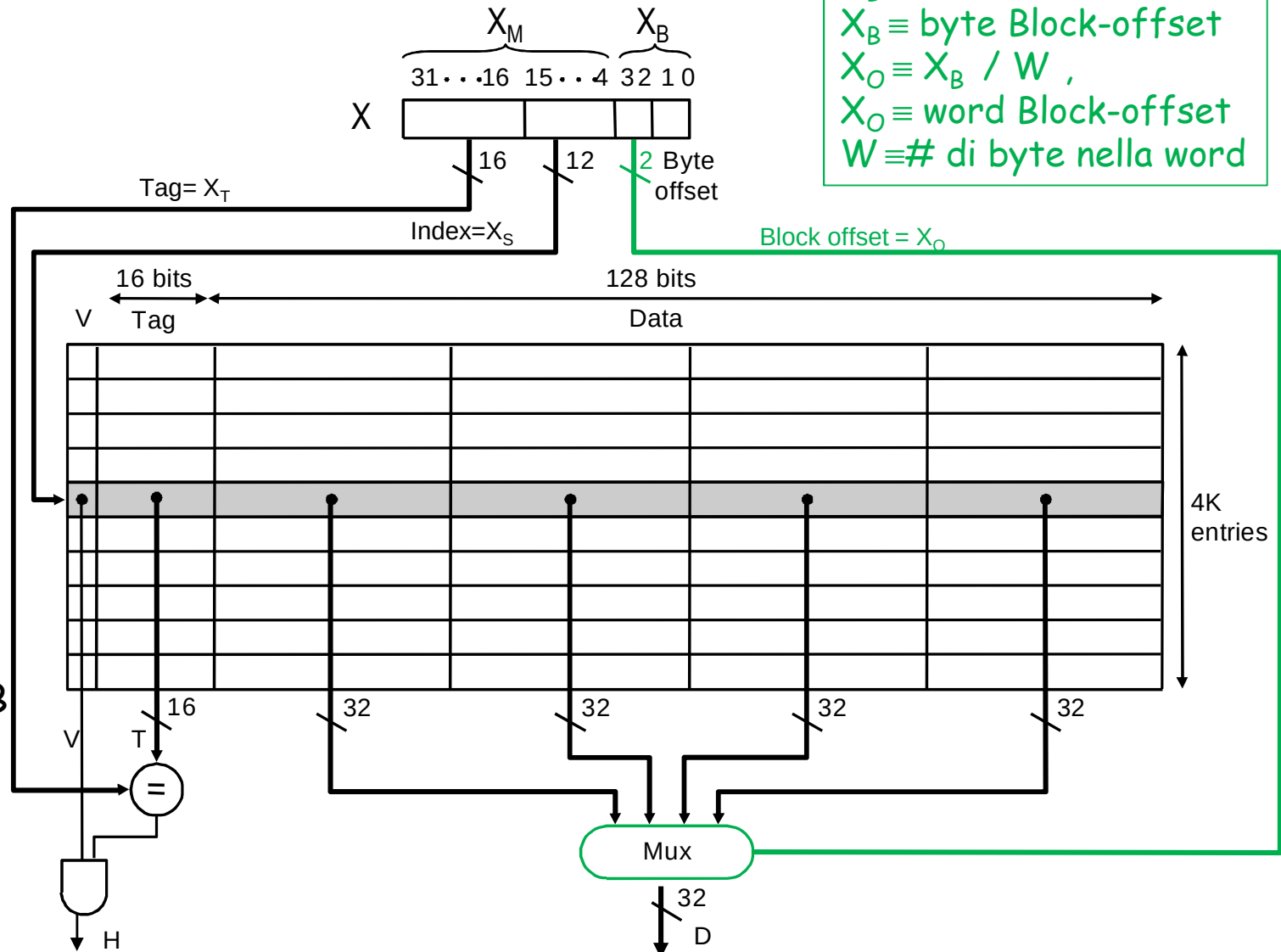
- $B = 16$ byte
- $C = 64$ Kbyte
- $M = 4$ Gbyte

X indirizzo al byte

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

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

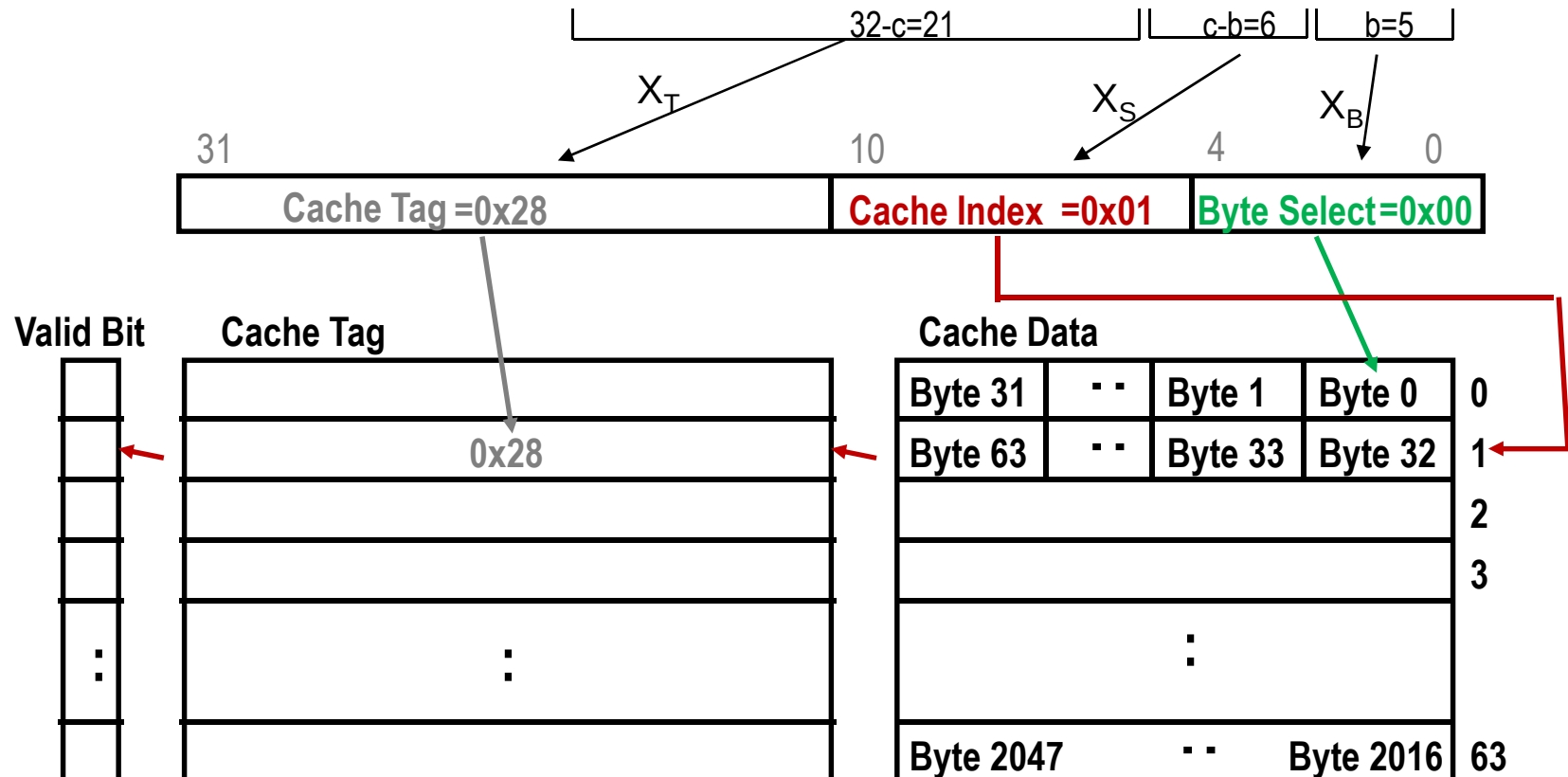
$$X_S = X_M \bmod N_C = X_M \bmod C/B = X_M \bmod 2^{12}$$



Esempio: cache ad accesso diretto $C=2K$, $B=32$

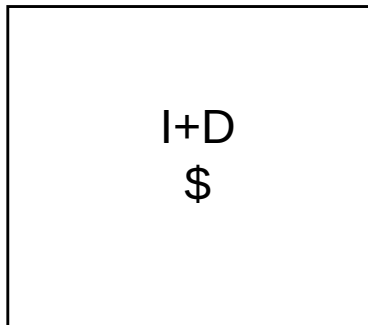
- Per una cache ad accesso diretto di dimensione $C=2^c$ byte e blocco $B=2^b$ byte
 - i $(32 - c)$ bit piu' significativi sono sempre il Tag (X_T)
 - i b bit meno significativi sono sempre il byte-offset nel blocco ($X_B \equiv X \bmod B$)

Indirizzo: 0x00014020 -> 0000'0000'0000'0001'0100'0000'0010'0000

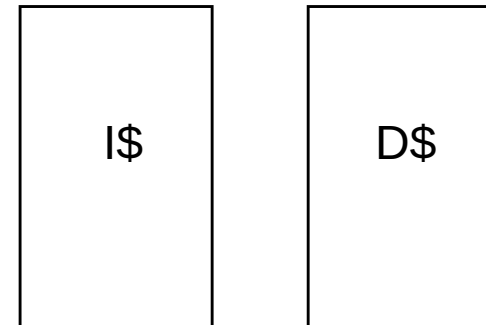


Split Cache

CACHE UNIFICATA



SPLIT CACHE



- La split cache consente di sfruttare maggiormente la localita' 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%

Tempo Medio di Accesso alla Memoria (con CACHE)

• Definiamo

h = Probabilita' di Hit

m = Probabilita' di Miss = $(1-h)$

t_{hit} = Tempo di Accesso in caso di Hit

t_{miss} = Tempo di Accesso in caso di Miss

$t_{penalty}$ = Tempo aggiuntivo rispetto al caso di Hit, per servire un Miss ($= t_{miss} - t_{hit}$)

→ Average Memory Access Time (AMAT):

$$\begin{aligned} \text{AMAT} &= h \times t_{hit} + m \times t_{miss} \\ &= h \times t_{hit} + m \times (t_{hit} + t_{penalty}) \\ &= t_{hit} + m \times t_{penalty} \end{aligned}$$

(MissTime = HitTime + PenaltyTime)

(MissRate + HitRate = 1)

Ovvero Average Memory Access Cycles (AMAC):

$$\text{AMAC} = C_{hit} + m \times C_{penalty}$$

($C_{hit} = t_{hit}/T_c$, $C_{penalty} = t_{penalty}/T_c$)

In altri termini, SE NON HO HIT IN CACHE, i cicli "ideali" (C_{hit}) per l'accesso alla memoria sono aumentati dei cicli di penalty ($C_{penalty}$)

Numero medio di riferimenti per istruzione

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

- Indichiamo quindi con
 $\overline{R}_{INST} = \text{Average References per Instruction}$ $\overline{R}_{INST} = \frac{N_{REF}}{N_{CPU}}$

- $N_{REF,i}$ = numero di riferimenti generati dalla istruzione i-esima

- $N_{REF,i1} = 2$ (es. istruz. LOAD)
- $N_{REF,i2} = 1$ (es. istruz. ADD)

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

- Nell'esempio precedente:
 - $N_{REF} = \text{numero totale di riferimenti} = 3$
 - $N_{CPU} = 2$
 - $\Rightarrow \overline{R}_{INST} = 1.5$

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

CPI modificato

- QUINDI:

- Ogni istruzione genera un numero medio di riferimenti $\overline{R_{INST}}$ e incorrerà in un ulteriore ritardo $C_{penalty}$ per ogni miss che si verifica con una probabilità m

- In altri termini, detti:

$\overline{R_{INST}}$ = Average References per Instructions

m = Miss Rate = (# of Misses) / (# of References)

$C_{penalty}$ = Penalty Cycles = $t_{penalty} * T_c$

il CPI medio si può calcolare come

$$\overline{CPI} = \overline{CPI_{ideal}} + \overline{R_{INST}} * m * C_{penalty}$$

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

$$\overline{CPI_{ideal}} = \sum_{K=1}^T \overline{CPI}_K = \frac{1}{N_{CPU}} \cdot \sum_{K=1}^T N_{CPU,K} \cdot CPI_K$$

Eq. delle prestazioni MODIFICATA

- Equazione delle prestazioni:

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

- /CPI stavolta e' un "CPI-efficace" = "CPI ideale" +
+ numero cicli medi di stallo (penalty) per istruzione

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

→ Ho due modi per aumentare le prestazioni:

- Diminuire il miss rate
- Diminuire i cicli di penalty

Contributi separati delle istruzioni generiche e L/S

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

$$m_I = N_{\text{MISS\_DATI}} / N_{\text{REF},I}$$

$$m_D = N_{\text{MISS\_ISTRUZIONI}} / N_{\text{REF},D}$$

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

$$N_{\text{REF},I} = N_{\text{CPU}}$$

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

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

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

m= miss combinato dati+istruzioni

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

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

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

Rapporto fra T_{CPU} effettivo ed ideale

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

$$\frac{T_{CPU}}{T_{CPU,ideal}} = 1 + \frac{N_{CPU} \cdot m \cdot \bar{R}_{INST} \cdot C_{pen} \cdot T_C}{N_{CPU} \cdot \overline{CPI}_{ideal} \cdot T_C} = 1 + \frac{m \cdot \bar{R}_{INST} \cdot C_{pen}}{\overline{CPI}_{ideal}}$$

$$\boxed{\frac{T_{CPU}}{T_{CPU,ideal}} = 1 + \frac{m \cdot \bar{R}_{INST} \cdot C_{pen}}{\overline{CPI}_{ideal}}}$$

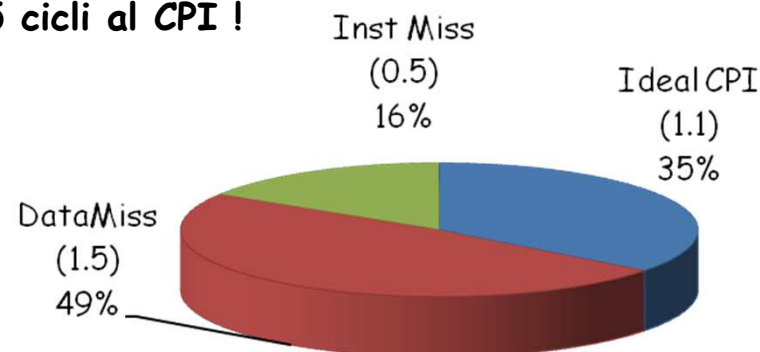
Rallentamento della macchina
reale con cache avente miss m e penalty C_{pen}
($\bar{R}_{INST}$ e $\overline{CPI}_{ideal}$ non dipendono dalla cache)

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 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

Lezione 18

Introduzione alla memoria cache (2)

<http://www.dii.unisi.it/~giorgi/didattica/arc11>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)
Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Other material is adapted from CS152 Copyright (C) 2000 UCB

Cache Associative

- Posso fare in modo che sullo stesso indirizzo di cache insistano più blocchi. 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:

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

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

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

- Sussistono le seguenti relazioni

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

$$N_M = M / B = 2^{m-b} \rightarrow \text{Numero di blocchi della Memoria}$$

(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)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

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

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

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

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

- Data una cache ad 1 via ad 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 e' detto A="numero di vie" o associativita' della cache

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

→ Numero di Set della cache

- L'indirizzo del set sara'

$$X_S = F_{S-SA}(X_M) = X_M \bmod N_S$$

- ...e il tag

$$X_T = F_{T-SA}(X_M) = X_M \div N_S$$

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

Cache Full Associative

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

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

$N_S=1 \rightarrow X_T = X_M$ e $X_S = 0$ (cioe' 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 $Y_{T,k}$
 - La funzione di hit sara' piu' complessa
 - $F_H : (\forall k=1\dots A, X_T == Y_{T,k} \ \&\& \ Y_{V,k} == 1 ? 1 : 0)$

Schema logico di una cache 4-way set associativa

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

X indirizzo
al byte

$$X_M = X / B$$

$$= X / 4$$

$$X_S = X_M \bmod N_S$$

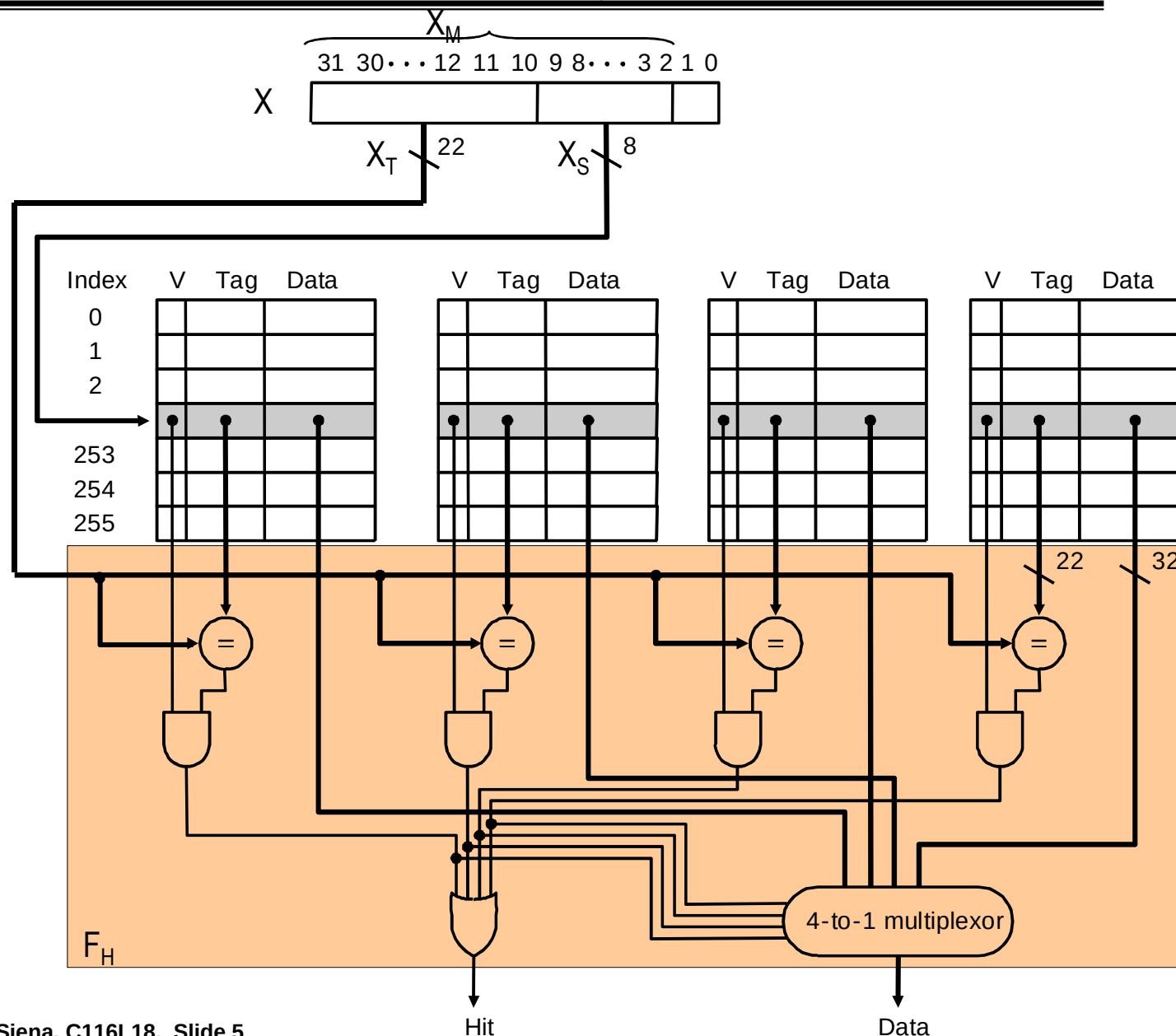
$$= X_M \bmod C/B/A$$

$$= X_M \bmod 2^8$$

$$X_T = X_M / N_S$$

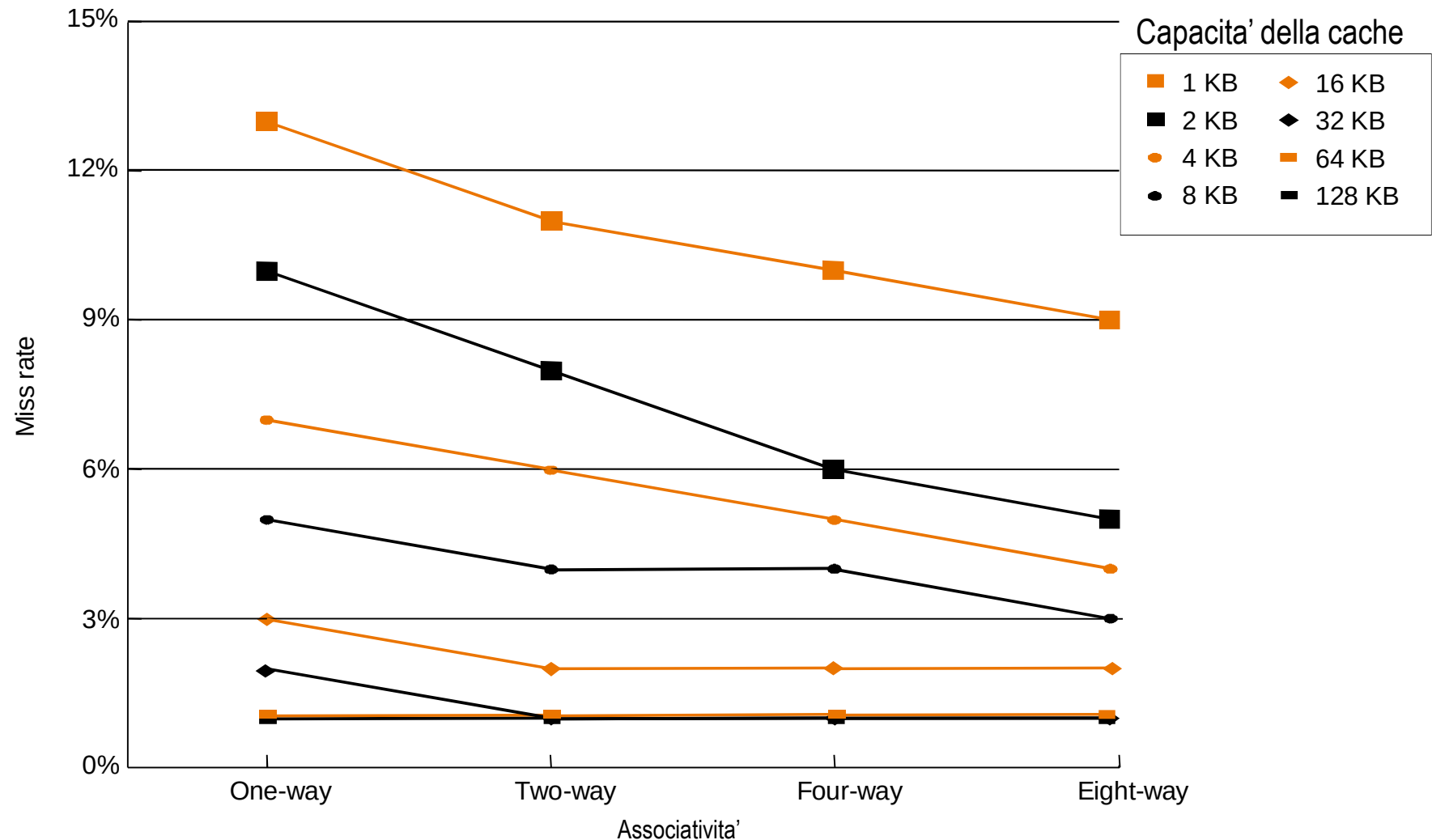
$$= X_M / (C/B/A)$$

$$= X_M / 2^8$$



Prestazioni: Direct-access vs. Set-associative cache

- L'associativita' tipicamente riduce il miss rate



Politica di Rimpiazzamento

- Come si procede quando la cache e' piena?
 - 1) Scelgo il blocco da rimpiazzare
 - 2) Carico il nuovo blocco nella locazione liberata
 - 3) Leggo o scrivo in quel blocco
- La scelta del blocco puo' avvenire secondo una delle seguenti politiche
 - **random**: scelgo un blocco a caso
 - Veloce, economica, prestazioni scarse...
 - **LRU** (Least Recently Used): scelgo il blocco usato meno recentemente
 - Ottime prestazioni, costosa da implementare
 - 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

Politica di Rimpiazzamento LRU

- **LRU (Least Recently Used)**
 - **Idea:** butto il blocco usato (per una lettura o una scrittura) meno recentemente
 - **Vantaggi:** sfrutta la localita' temporale → aver usato il blocco poche istruzioni fa, comporta che probabilmente lo riusero'...
 - **Svantaggi:** con una cache a 2 vie e' facile tenere traccia del blocco LRU (basta un bit); con una cache a 4 vie l'hardware necessario e' abbastanza piu' complesso

	2-vie		4-vie		8-vie	
Capacita'	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Politica di Rimpiazzamento LRU: esempio (1)

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

$$\left. \begin{array}{l} A=2 \\ B=4 \\ C=16 \end{array} \right\} \Rightarrow N_s=2$$

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

R 0
W 2
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?

Politica di Rimpiazzamento LRU: esempio (2)

- Indirizzi 0, 2, 0, 1, 4, 0, ...

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

2: ($X_S=0, X_T=1$) → miss, carico nel set 0 (loc 1)

0: ($X_S=0, X_T=0$) → hit

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

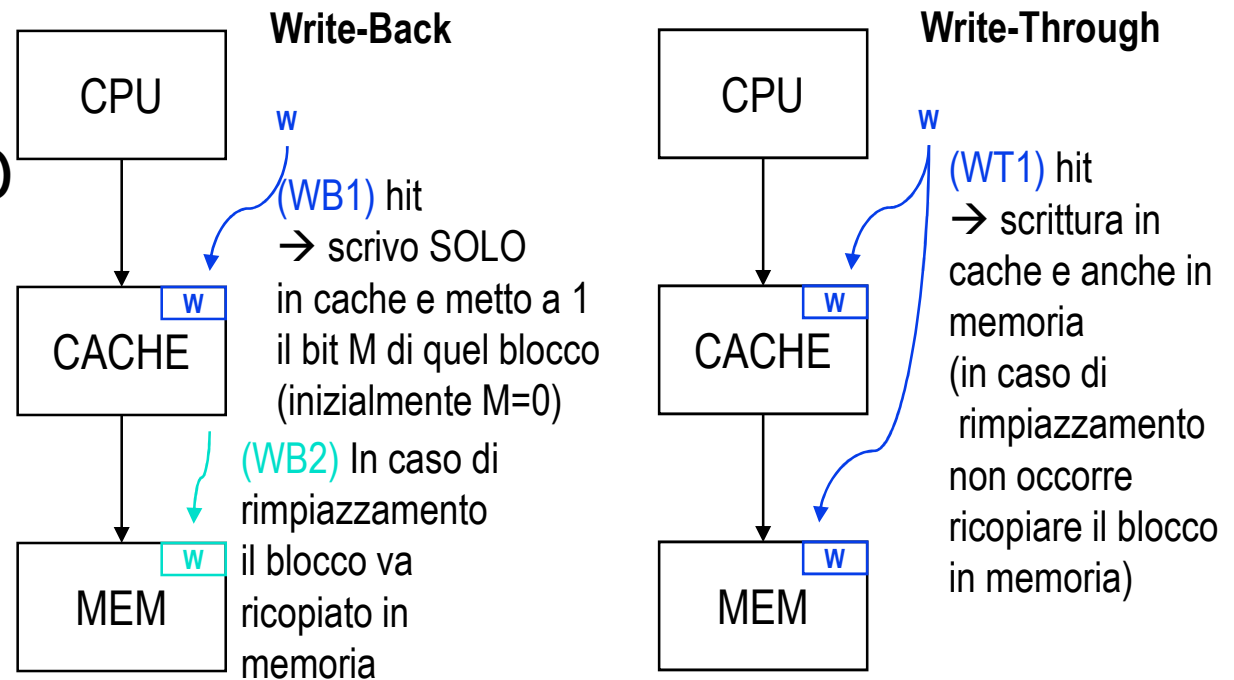
4: ($X_S=0, X_T=2$) → miss, carico nel set 0 (loc 1, rimpiazzo l'1)

0: ($X_S=0, X_T=0$) → hit

	loc 0	loc 1
set 0	0	lru
set 1		
set 0	lru 0	1
set 1		
set 0	0	lru 1
set 1		
set 0	0	lru 1
set 1	0	lru
set 0	lru 0	2
set 1	0	lru
set 0	0	lru 2
set 1	0	lru

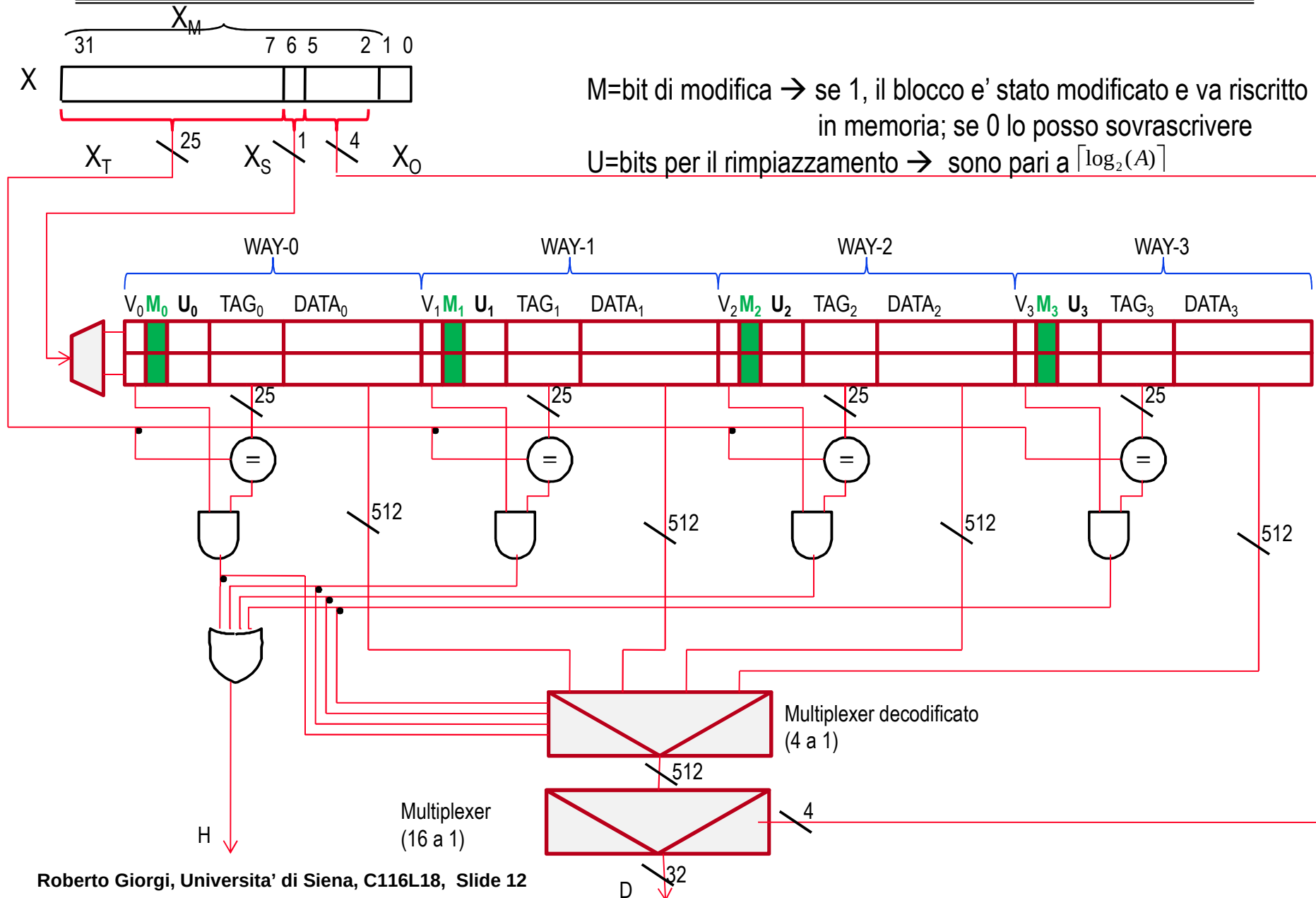
Politica di gestione delle scritture su hit

- Ho due tecniche:
 - Write-Back (WB)
 - Write-Through (WT)



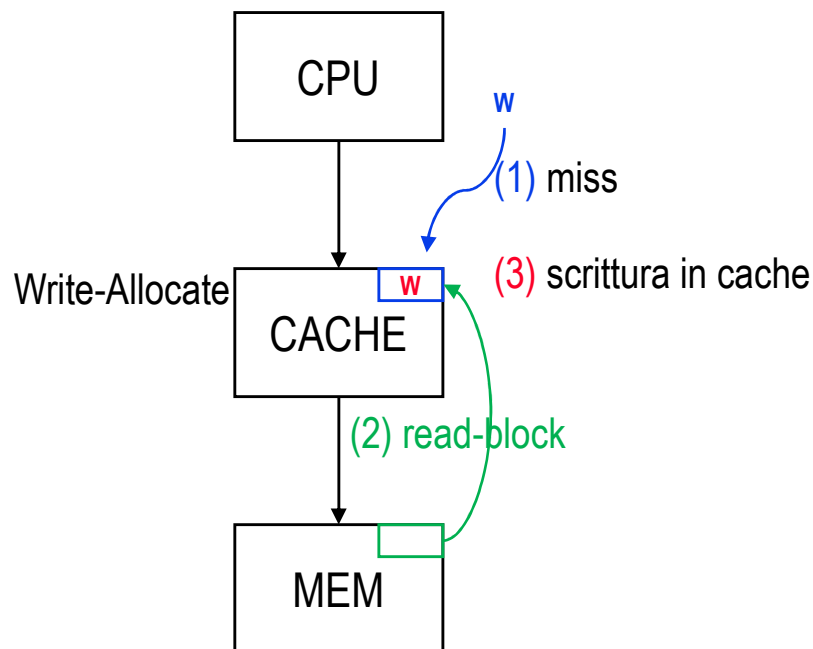
- Confronto Write-Back (WB) / Write-Through (WT):
 - WB: occorre un bit M per ogni blocco
 - + WB: riduco il traffico sul bus di memoria (write), rispetto a WT
 - WB: si introduce un po' di traffico di aggiornamento della memoria

Cache set associativa a 4 vie di tipo Write-Back

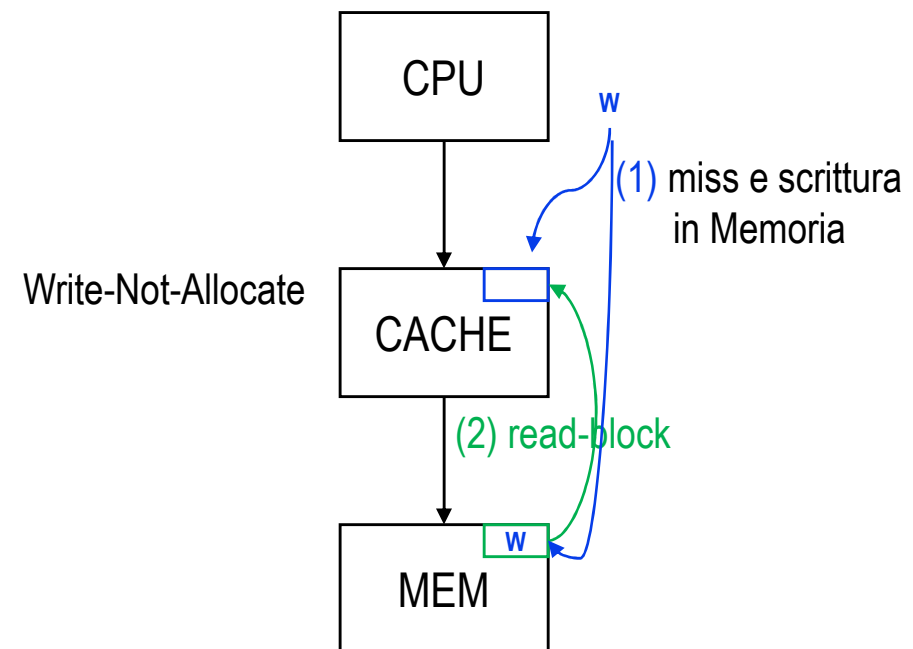


Politica di gestione delle scritture su miss: WA/WNA

- Esistono due tecniche
 - Write Allocate (WA)
 - Write Not-Allocate (WNA)

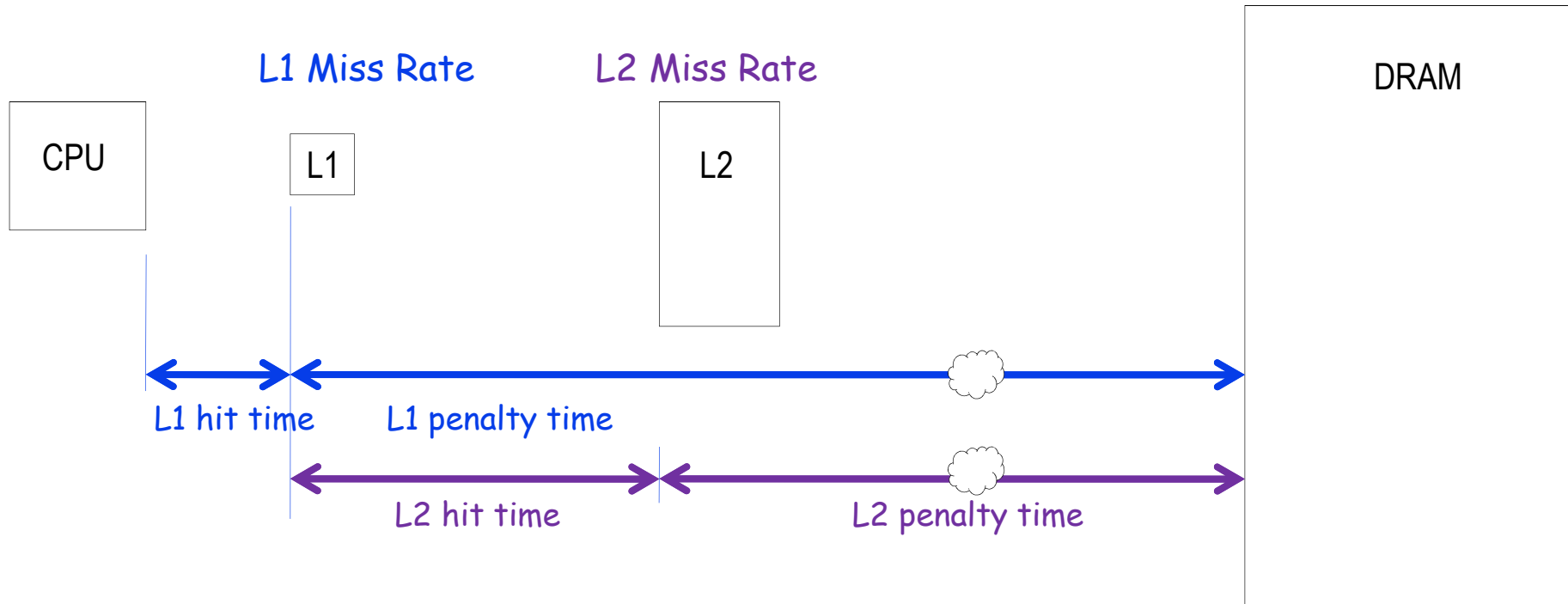


WA, NOTA: Le operazioni 2 e 3 vengono compiute come una singola azione



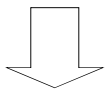
WNA, NOTA: Normalmente l'operazione 2, NON viene compiuta (il beneficio effettivo di prelevare il blocco o meno puo' dipendere fortemente dal tipo di applicazione)

AMAT per gerarchia di memoria con 2 livelli di cache



$$AMAT = L1 \text{ Hit Time} + L1 \text{ Miss Rate} * L1 \text{ Penalty Time}$$

$$L1 \text{ Penalty Time} = L2 \text{ Hit Time} + L2 \text{ Miss Rate} * L2 \text{ Penalty Time}$$



$$AMAT = L1 \text{ Hit Time} + L1 \text{ Miss Rate} * (L2 \text{ Hit Time} + L2 \text{ Miss Rate} * L2 \text{ Penalty Time})$$

Valori tipici

- **L1**
 - **Capacita':** decine di KB
 - **Hit-time:** uno (o due) cicli di clock
 - **Miss-rate tipici:** 1-5%
- **L2:**
 - **Capacita':** 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**
 - **Perche' allora il miss_rate di L2 e' cosi' alto?**

Cache a piu' livelli: esempio

- Supponiamo di avere L1 e L2 con i seguenti dati:
 - L1 Hit-Time = 1 ciclo
 - L1 Miss-rate = 5%
 - L2 Hit-Time = 5 cicli
 - L2 Miss-rate = 15% (% dei miss di L1 che fa miss in L2)
 - L2 Penalty-Time = 100 cicli
 - L1 miss-penalty = $5 + 0.15 \times 100 = 20$
 - AMAT = $1 + 0.05 \times 20 = 2$ cicli
-

- Supponiamo di avere solo L1 con:
 - L1 Hit-Time = 1 ciclo
 - L1 Miss-rate = 5%
 - L1 Penalty-Time = 100 cicli
- AMAT = $1 + 0.05 \times 100 = 6$ cicli

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

Lezione 19

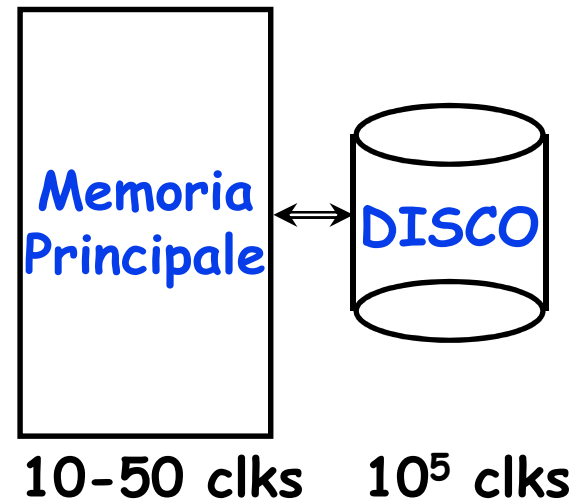
Memoria Virtuale

<http://www.dii.unisi.it/~giorgi/didattica/arc1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.)
Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Disk Caching (!= Memoria Virtuale)

- Idea: utilizzare la Memoria Principale come se fosse una cache del disco
 - Scopo: ridurre il tempo medio di accesso ai file che stanno su disco



- Funzionamento:
 - e' una cache implementata completamente a software
 - Memorizza i blocchi di disco piu' recentemente utilizzati
 - Su hit, ottengo un tempo di accesso di $10^5/10=10^4$ volte maggiore
 - La tecnica e' usata da Windows, Linux e altri sistemi operativi

Memoria Virtuale

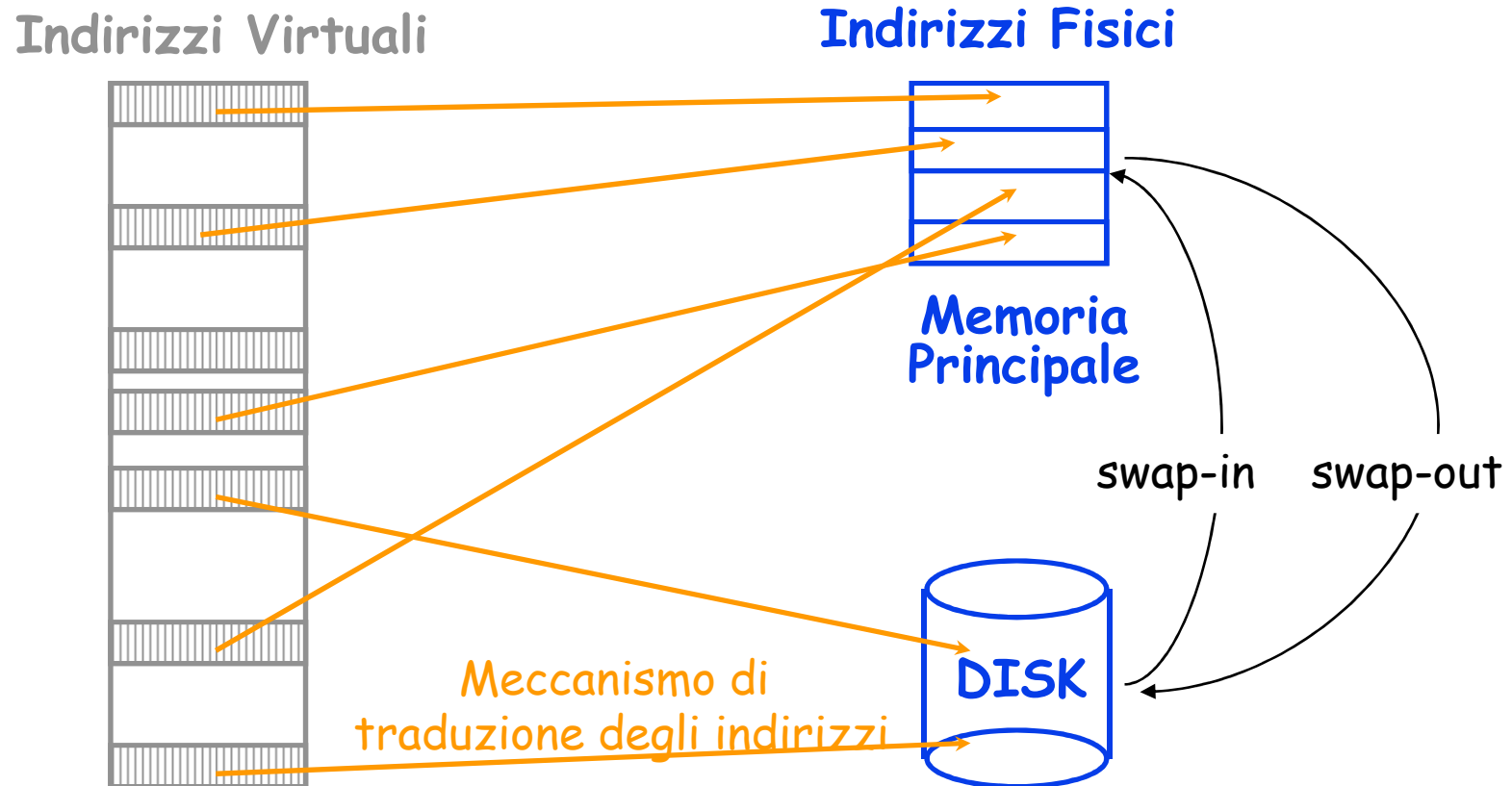
- Obiettivi delle tecniche di caching Memoria $\leftrightarrow$ Disco
 - Disk Caching $\rightarrow$ migliorare il tempo di accesso file
 - Memoria Virtuale $\rightarrow$ creare uno spazio di indirizzamento piu' largo (con tempi di accesso simili alla memoria principale)
- Uso il disco come "memoria esterna" rispetto alla memoria principale: il livello successivo nella gerarchia di memoria
 - La memoria contiene una copia delle istruzioni/dati piu' recentemente usati
 - Tutto il resto sta su disco

Memoria Virtuale: vantaggi

- **Crea l'illusione di un enorme quantita' di memoria (molto piu' ampia della memoria fisica realmente disponibile nel calcolatore)**
 - Rimuove i vincoli di programmazione legati all'avere una memoria di dimensioni limitate
- **Rilocazione dei programmi non necessaria**
 - Il programma viene caricato in uno spazio di memoria che e' completamente vuoto per ogni programma, ovvero ogni programma ha a disposizione tutti gli indirizzi di uno proprio spazio virtuale privato
- **Protezione dei programmi**
 - Porzioni dello spazio di indirizzamento possono essere rese condivise in maniera controllata ed efficiente
 - Un programma non puo' accedere allo spazio di indirizzamento privato di un altro programma
- **Il tutto funziona grazie ancora al principio di localita'**
 - In un certo intervallo di tempo lo working set del programma e' abbastanza limitato

Memoria Virtuale: Funzionamento

- L'unita' di trasferimento fra memoria e disco e' la **pagina**
- La pagina ha una dimensione tipica di 4-16KB



- La Pagina Virtuale indirizzata puo' trovarsi sul disco anziche' nella memoria fisica → swap-in

Tempi di accesso

- Rapporti tipici fra i tempi di accesso di vari tipi di memoria

L1→L2

$$\frac{t_{\text{penalty},L1}}{t_{\text{hit},L1}} = 5$$

L2→MEM

$$\frac{t_{\text{penalty},L2}}{t_{\text{hit},L2}} = 5$$

MEM→DISCO

$$\frac{t_{\text{penalty},MEM}}{t_{\text{hit},MEM}} = 10000$$

- Nel caso di un page-miss, o meglio di un **page-fault**, il tempo di penalty risulta molto grande

Esempio

- Il processore **P**, viaggia a 30 MIPS e 1/3 delle istruzioni generate da un certo programma sono di tipo LD
⇒ **P** fa $30/3 = 10$ milioni di load/secondo
- Supponiamo che L1 e L2 insieme riescano a servire il 95% delle richieste → la **Memoria Principale** deve servire $0.05 \times 10^7 = 5 \times 10^5$ load/secondo
- Supponiamo che il **Disco** possa servire 50 richieste/secondo (considerando un rapporto di 1:10000 per servire un miss).
Determinare il tempo di esecuzione del programma ipotizzando che l'hit-rate della memoria principale assuma i seguenti valori:

<u>Hit Rate Memoria Principale</u>	<u>Richieste al disco per secondo</u>	<u>Tempo di Esecuzione del Programma</u>
100%	0	1 (normalizzato)
99.999%	5	_____
99.99%	50	_____
99.9%	50	_____
90.0%	50	_____

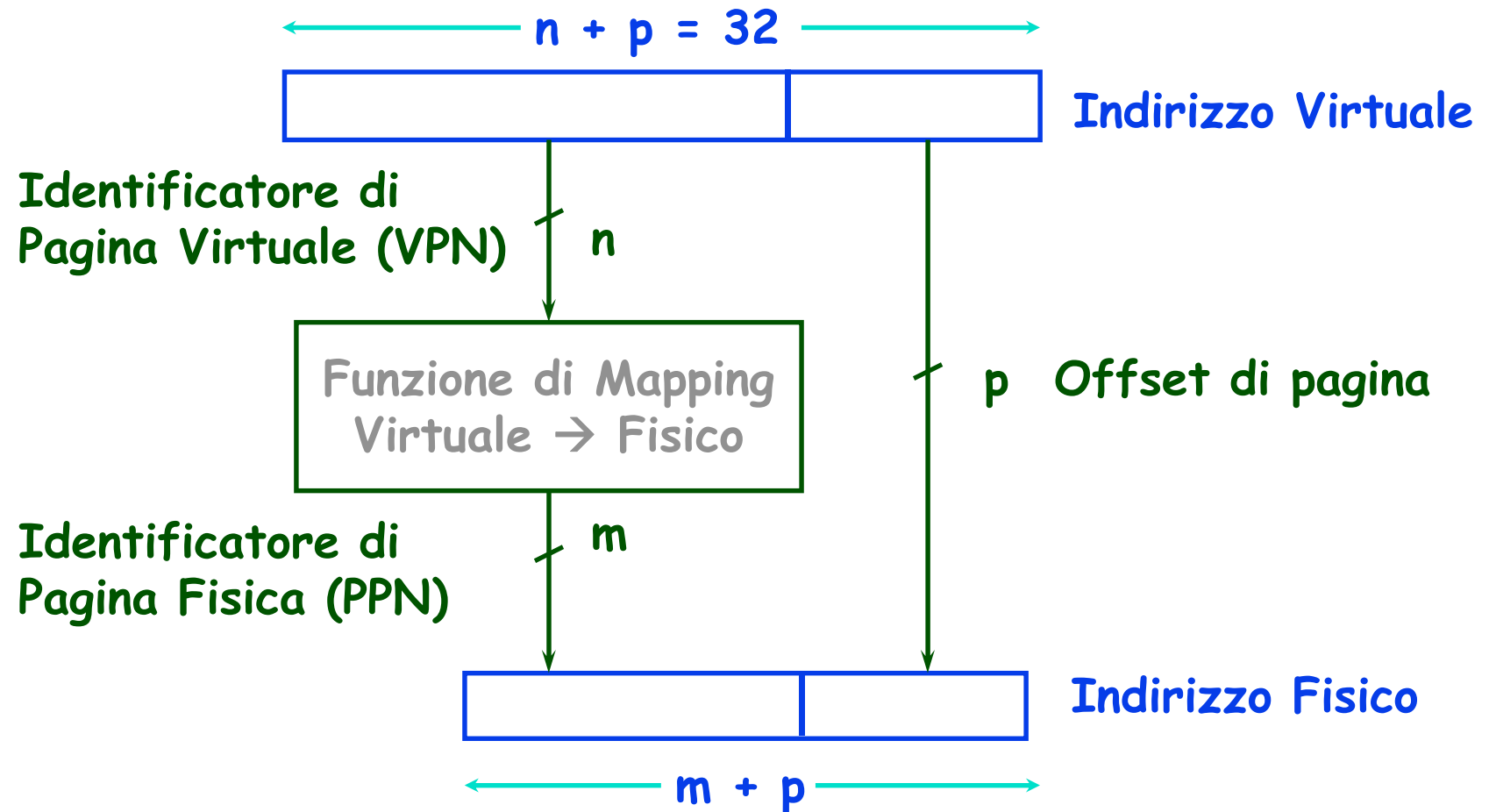
Memoria Virtuale: Caratteristiche

- La memoria principale si comporta come una cache
- Le caratteristiche di tale cache sono
 - $A = A_{MAX} \rightarrow$ cache fully associative: una pagina virtuale puo' essere mappata su qualsiasi pagina fisica
 - $B = 4-16KB$
 - $C =$ Dimensione della Memoria
 - Politica di Rimpiazzamento = LRU
 - Politica di Scrittura = WB (Write-Back)
 - Politica di Scrittura su Miss = WA (Write Allocate)

Memoria Virtuale: Terminologia

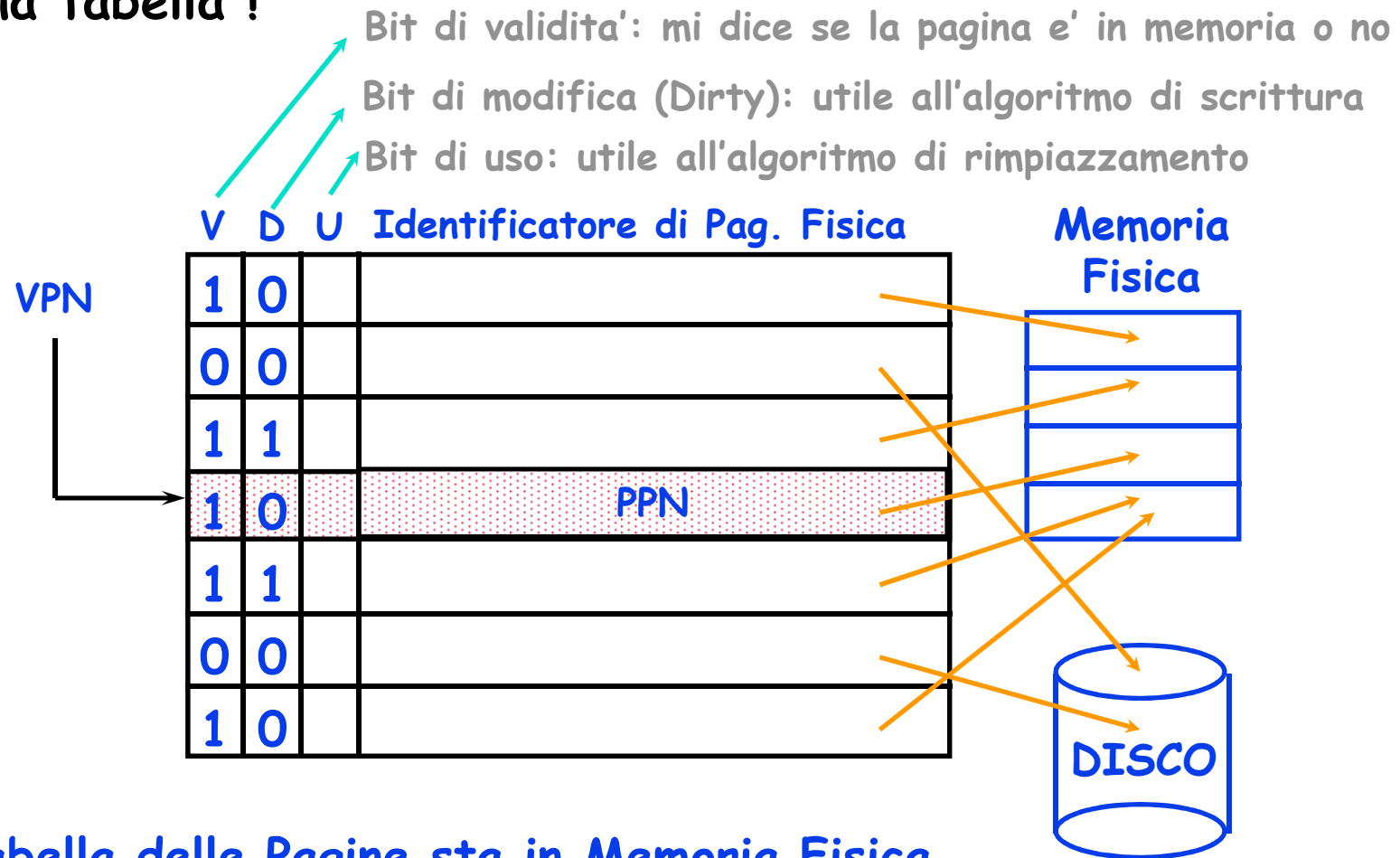
- Working set : Insieme di pagine che e' necessario mantenere in Memoria per ottenere un hit rate accettabile
- Thrashing : Perdita di prestazioni conseguente al fatto che lo working set e' diventato piu' grande della Memoria Principale
- Page Swap : Trasferimento di una pagina nuova in memoria principale e conseguente riscrittura su disco di una pagina "vittima"

Memoria Virtuale: schema logico



Funzione di Mapping Virtuale-Fisico

- E' una tabella !



La Tabella delle Pagine sta in Memoria Fisica
Quanto e' grande questa tabella?

Gestione del Page-Fault (Page-Fault Handler)

- Passi per leggere una locazione della Memoria Principale
 - 1) Presento un indirizzo virtuale al sistema di Memoria
 - 2) Accedo alla Tabella delle Pagine per vedere se la pagina e' in RAM
 - 3) Se la pagina e' in RAM accedo nuovamente alla RAM per prendere il dato
 - 4) Se la pagina non e' in RAM → lancio un interrupt al processore e chiamo una routine di servizio del Sistema Operativo denominata **Page-Fault Handler**
 - 5) Il Page-Fault Handler si occupa di fare il page-swap (tramite il DMA) e porta in RAM la pagina richiesta prelevandola da disco
 - 6) Viene ripetuta la lettura: stavolta la pagina si trovera' in RAM
- Anche se avessi una percentuale del 100% di Hit-Rate in Memoria Principale, la penalty dovuta all'uso della memoria virtuale comporterebbe un raddoppio dei tempi di accesso alla memoria

Tecniche per ridurre la Tabella delle Pagine

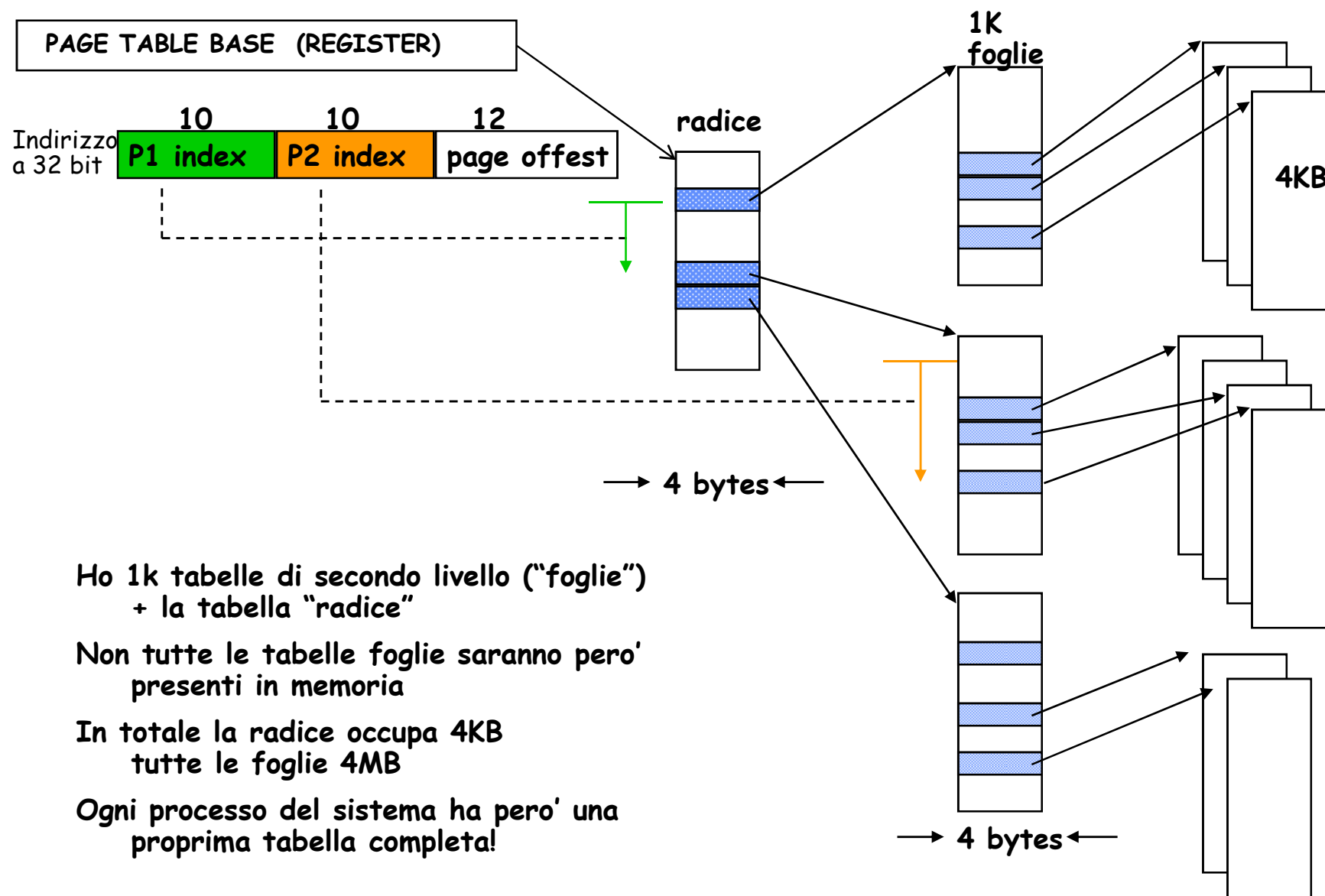
- A) Paginazione della Tabella delle Pagine

- Parte della tabella finisce in memoria virtuale anziche' fisica (→ si potra' generare un doppio page-fault !)
- Staranno in RAM solo le porzioni della tabelle piu' usate...

- B) Tabella delle Pagine Inversa

- Si applica una funzione di hashing all'indirizzo virtuale cosi' che il numero di entry nella tabella delle pagine e' al piu' pari al numero totale di pagine fisiche

A) Tabella delle Pagine a Due Livelli



Ho 1k tabelle di secondo livello ("foglie")
+ la tabella "radice"

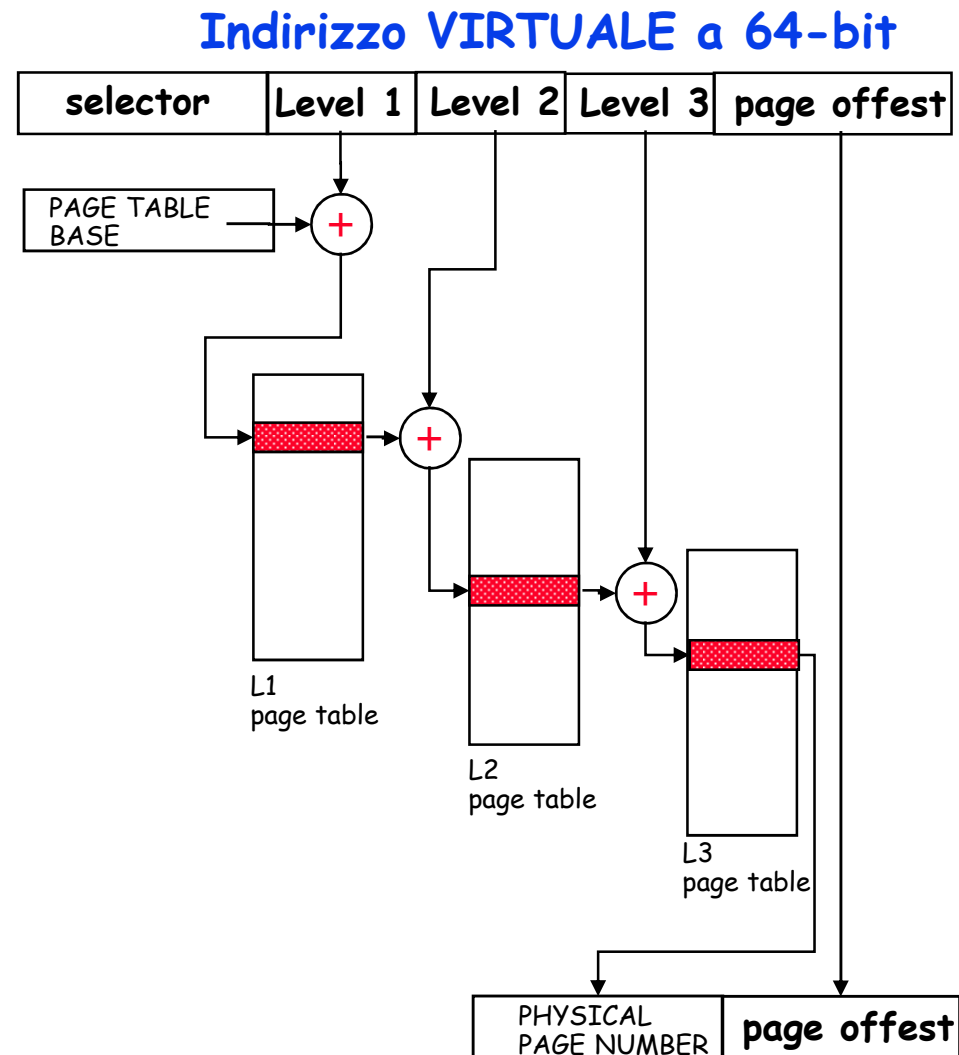
Non tutte le tabelle foglie saranno pero'
presenti in memoria

In totale la radice occupa 4KB
tutte le foglie 4MB

Ogni processo del sistema ha pero' una
propria tabella completa!

A) Tabella delle Pagine a Tre Livelli (Alpha)

- Avendo indirizzi a 64 bit conviene suddividere in tre livelli
 - Nell'Alpha uso in realta' solo 43 bit per lo spazio virtuale



64-bit PHYSICAL address

A) Metodi di accesso Top-Down e Bottom-Up

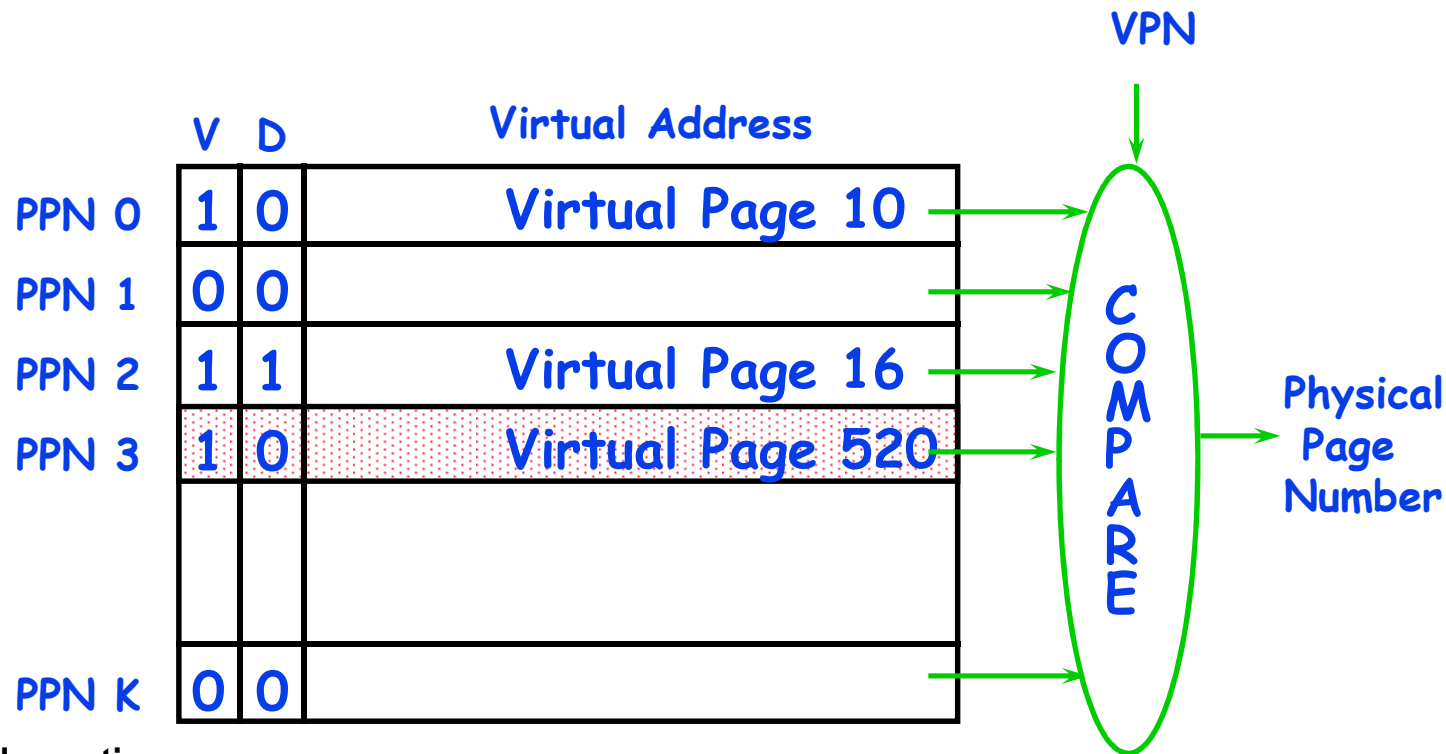
- **Top-Down**
 - Accesso alla root page table (sempre in memoria)
 - Accesso alla leaf page table
 - Accesso alla pagina fisica
 - Totale 3 accessi alla memoria + eventuale fault o doppio fault
 - In generale ho un numero di accessi alla memoria pari a $1 + \text{Num\_Livelli}$
- **Bottom-Up**
 - Accesso diretto all'insieme delle leaf page table* (supposte contigue in memoria e allineate su boundary di 4MB per semplificare l'accesso)
 - Se l'operazione ha successo: bene (ho solo 2 accessi in memoria)
 - Altrimenti: si procede con un accesso bottom-up completo
 - Se va tutto bene, faccio 2 accessi fissi indipendentemente dal numero di livelli della tabella delle pagine

* affinché questo funzioni devo avere un metodo per sapere se ci sono leaf al loro posto oppure no

B) Tabella delle Pagine Inversa (1)

- Una entry nella tabella delle pagine per ogni pagina fisica
- Vantaggi
 - La dimensione scala con la dimensione della memoria fisica anziche' con quella della memoria virtuale
 - La dimensione e' fissa a prescindere dal numero di processi !
- Chiamata "inversa" perche' le entry sono indicizzate per PPN anziche' per VPN
 - In ogni caso la ricerca avviene fornendo un VPN e ottenendo un PPN...
 - Il PPN e' dedotto dalla posizione della VPN trovata
- Tecnica
 - Si applica una funzione di hash al VPN
 - L'hash costituisce un indice nella tabella inversa corrispondente al PPN
 - Poiche' diversi VPN possono produrre lo stesso hash e' necessario un meccanismo per risolvere le collisioni (es. lista concatenata memorizzata all'interno della tabella stessa): la nuova entry e' aggiunta in fondo alla lista
 - E' possibile che si perda un po' tempo per percorrere tale lista concatenata

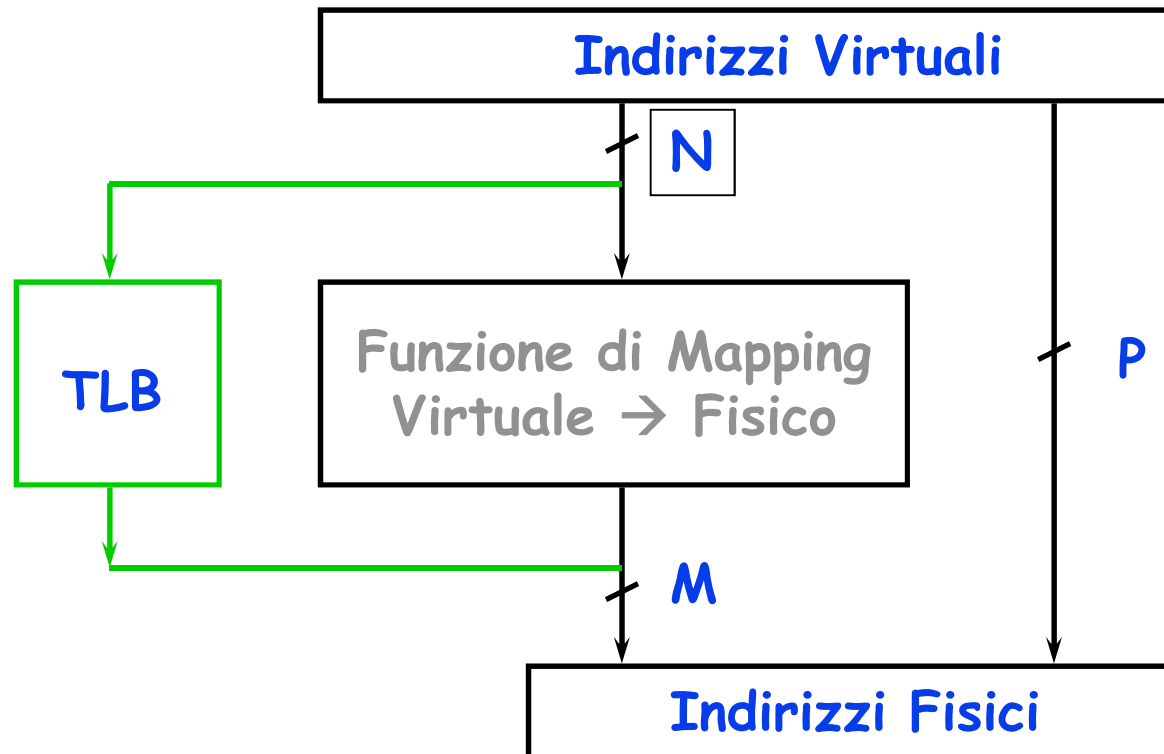
B) Tabella delle Pagine Inversa (2)



Numero di elementi
della tabella = **K**

L' IBM "System 38" (AS400) implementa indirizzi a 64-bit
48 bit costituiscono il numero di pagina virtuale
e tali elementi hanno un tag di soli 12-bit

Velocizzare il meccanismo di traduzione



Uso una piccola cache (TLB=Translation Lookaside Buffer) per avere "a portata di mano" un po' di elementi della Tabella delle Pagine

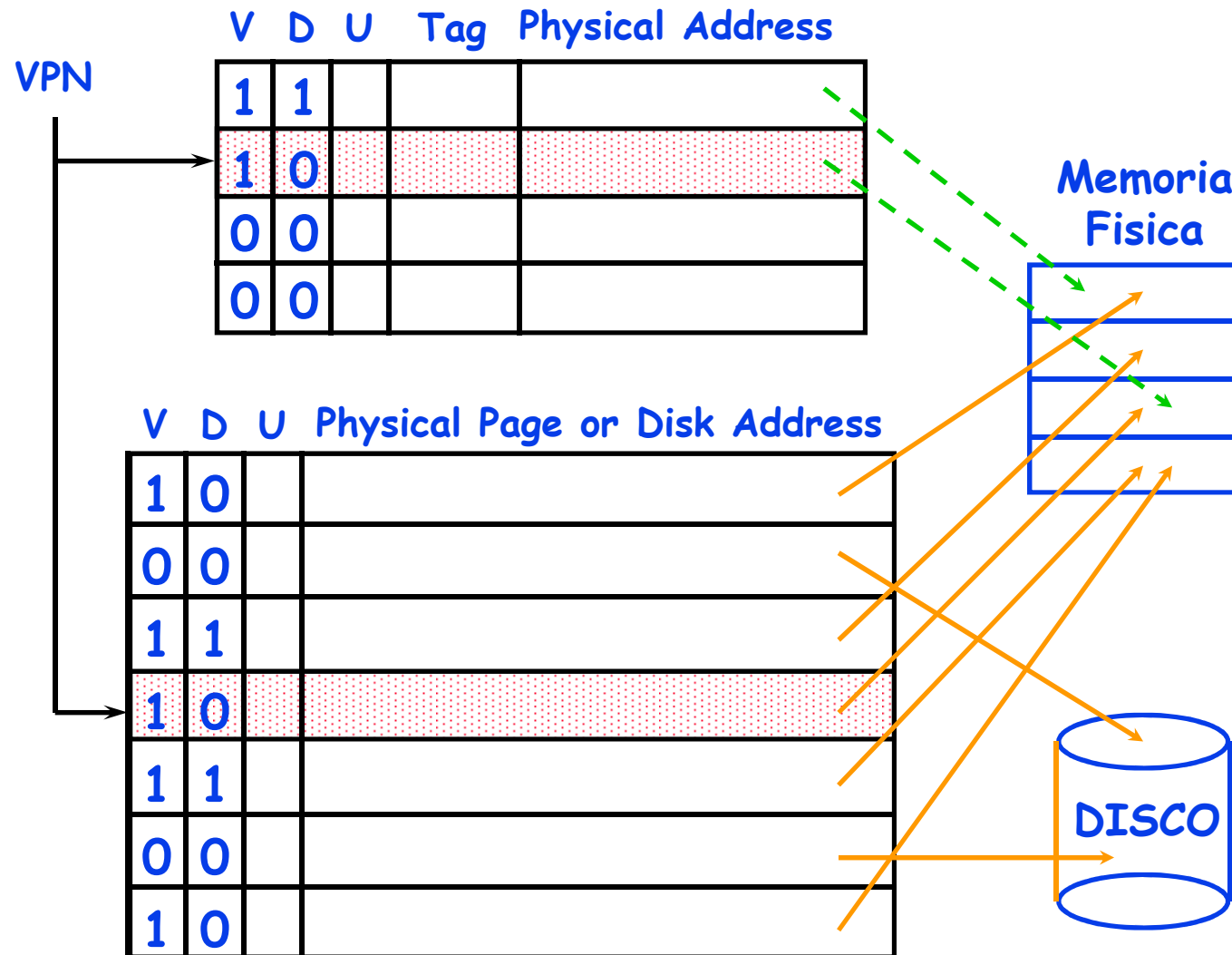
TLB: Caratteristiche

V	D	U	Permessi	VPN	PPN

↖
“tag”

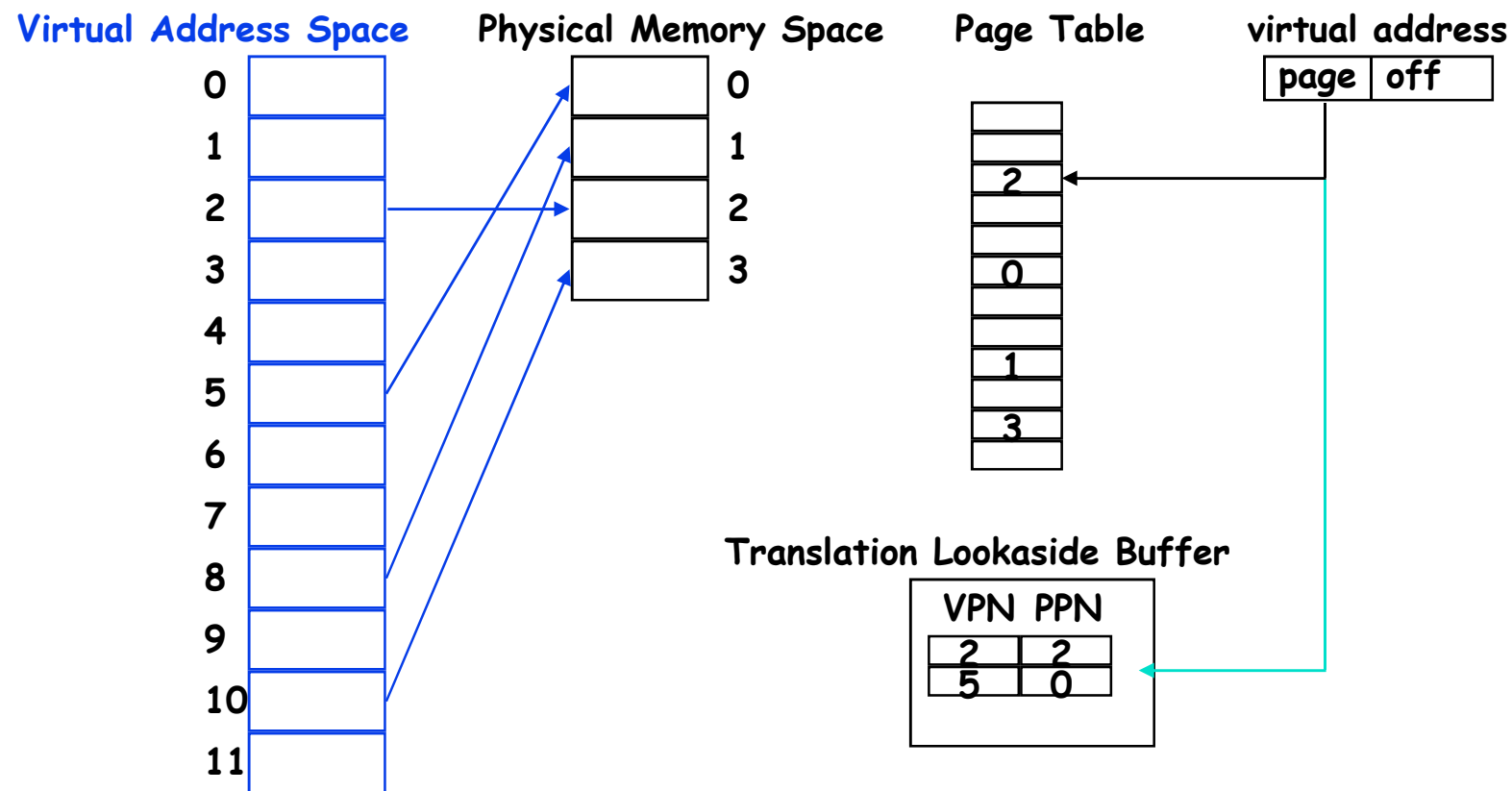
- La TLB e' una cache che puo' essere fully associative, set-associative o ad accesso diretto
- La TLB ha tipicamente non piu' di 128-256 elementi
- Il tempo di accesso alla TLB e' confrontabile col tempo di accesso alla cache (molto piu' basso del tempo di accesso alla memoria principale)

TLB: Organizzazione Logica

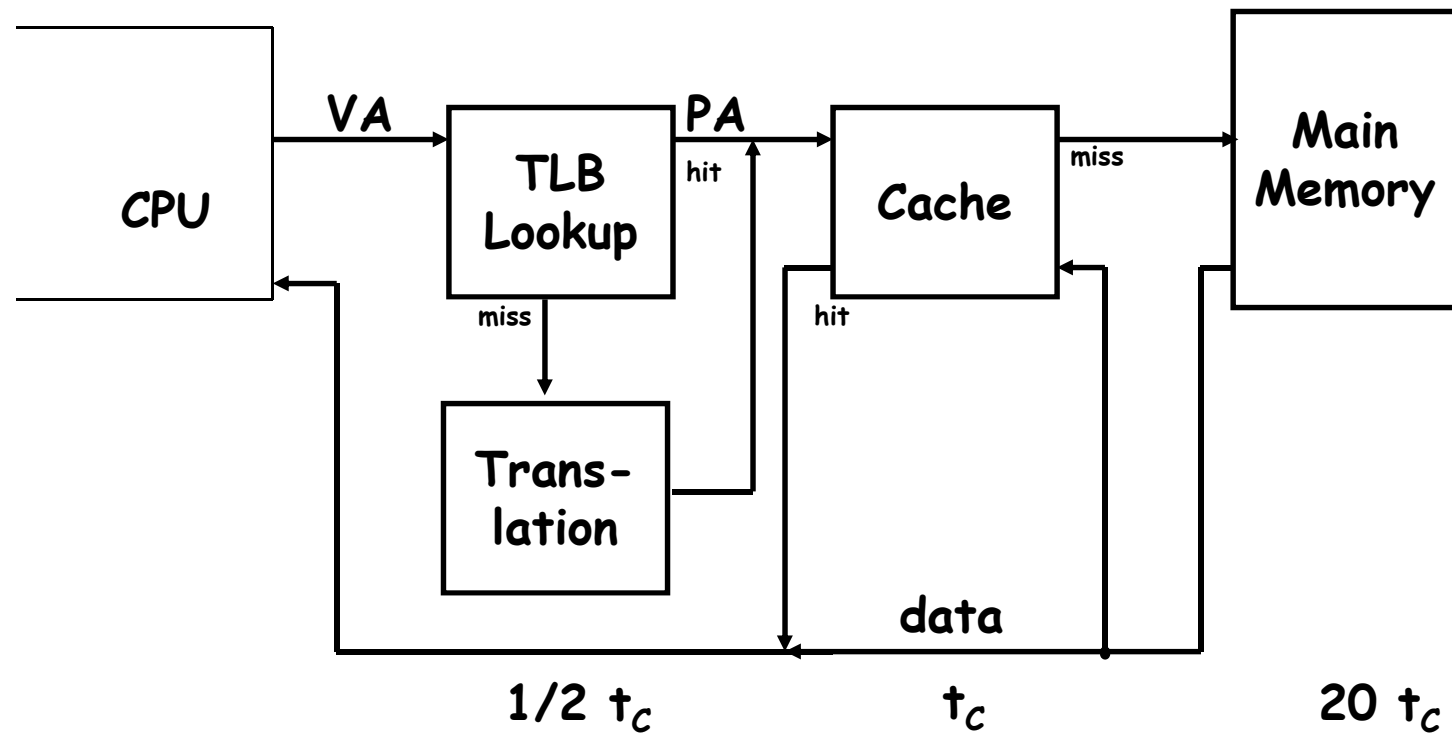


TLB: esempio

- Memoria Virtuale → la memoria agisce da cache per il disco
- La tabella delle pagine mappa indirizzi virtuali in fisici
- La TLB e' una cache delle piu' recenti traduzioni

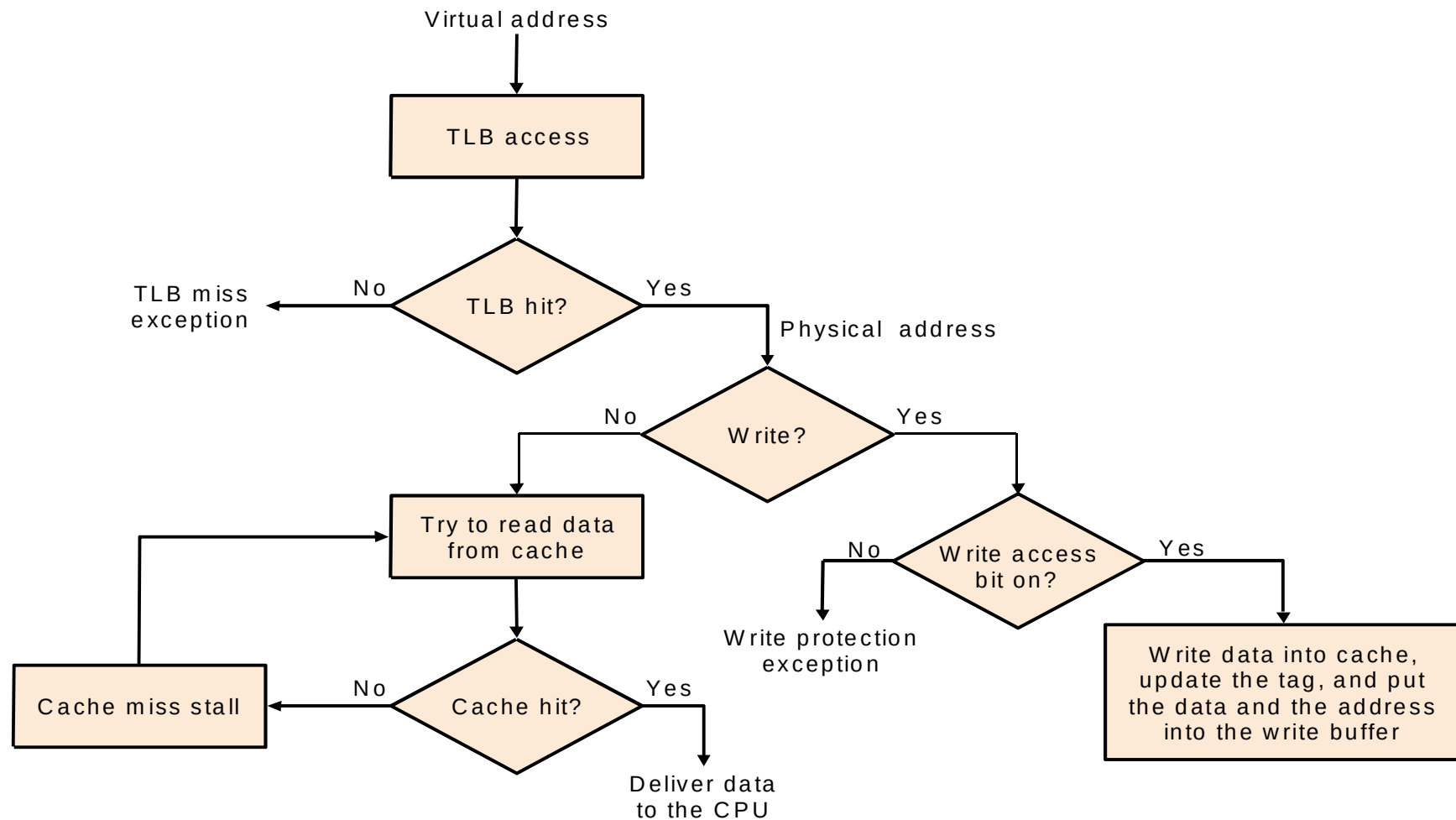


TLB: Schema e interazione con la Cache



VA = Virtual Address
PA = Physical Address

TLB+cache: diagramma di funzionamento



TLB: Valori tipici

Dimensione del blocco: 8 byte

Hit-time: 0.5 - 1 cicli

Miss-penalty: 10-50 cicli

Hit-rate: da 99% a 99.99%

Dimensione della TLB: 32-1024 elementi

Lezione 20

CPU con pipeline

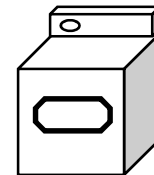
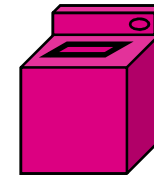
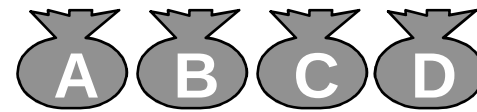
<http://www.dii.unisi.it/~giorgi/didattica/arcal1>

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material. (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED.) Figures may be reproduced only for classroom or personal educational use in conjunction with the book and only when the above copyright line is included. They may not be otherwise reproduced, distributed, or incorporated into other works without the prior written consent of the publisher.

Other material is adapted from CS152 Copyright (C) 2000 UCB

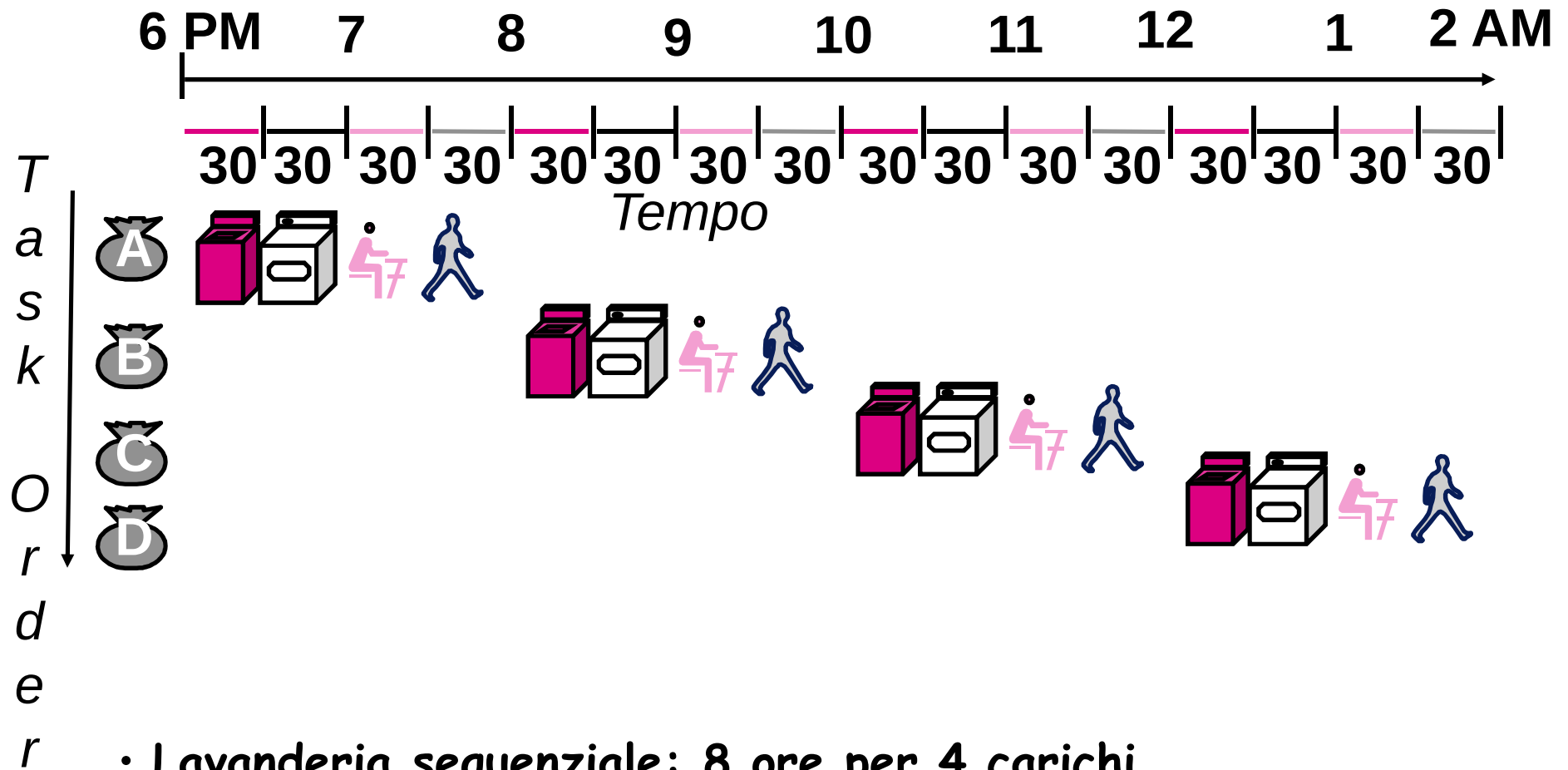
Il Pipelining e' Naturale!

- Esempio della Lavanderia
- Ann, Brian, Cathy, Dave devono:
- Lavare
- Asciugare
- Stirare
- Mettere a posto



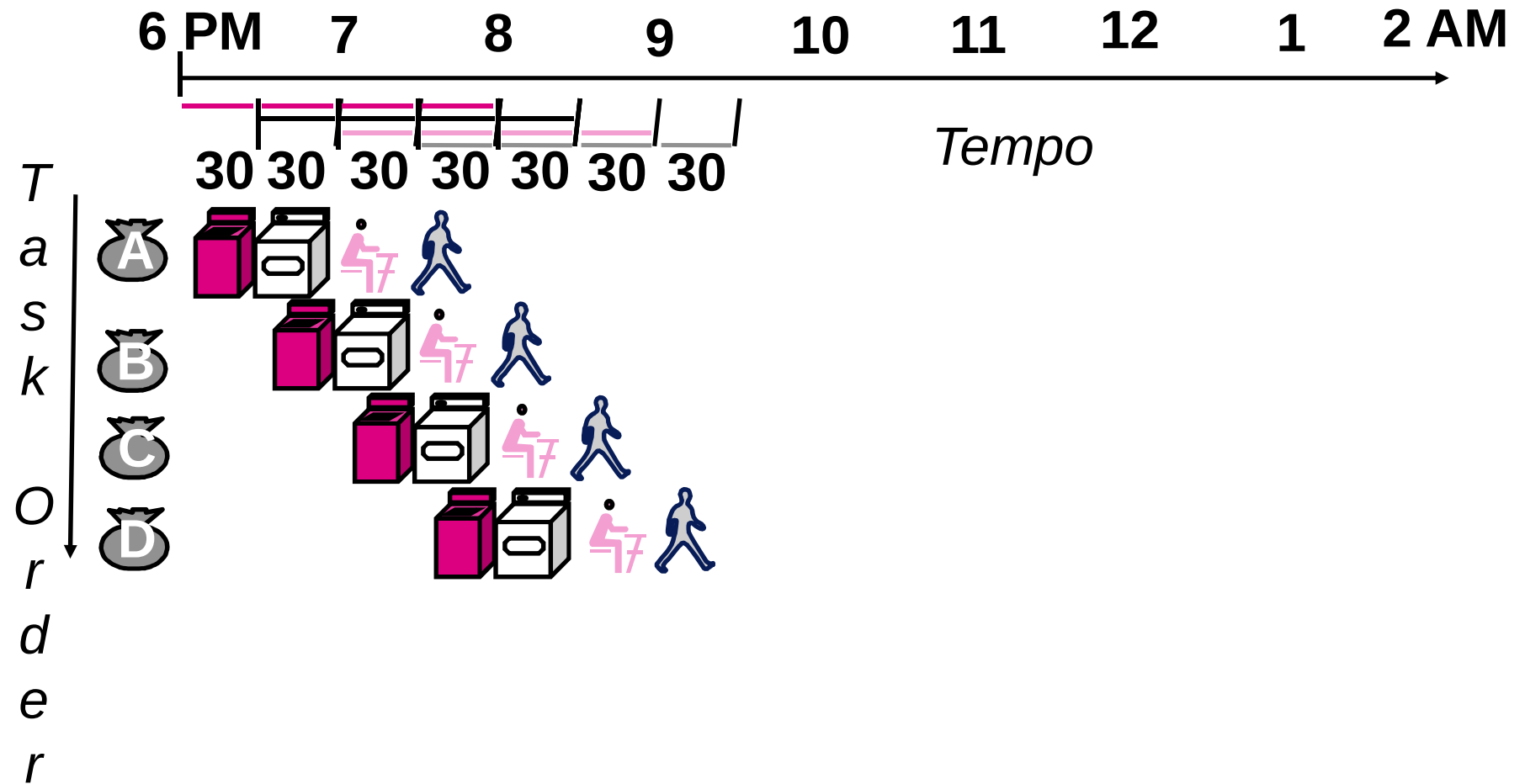
Supponiamo che ognuna di queste operazioni richieda 30 minuti
e che ci sia una sola lavatrice, una asciugatrice, un ferro da stiro, un banco

Lavanderia Sequenziale



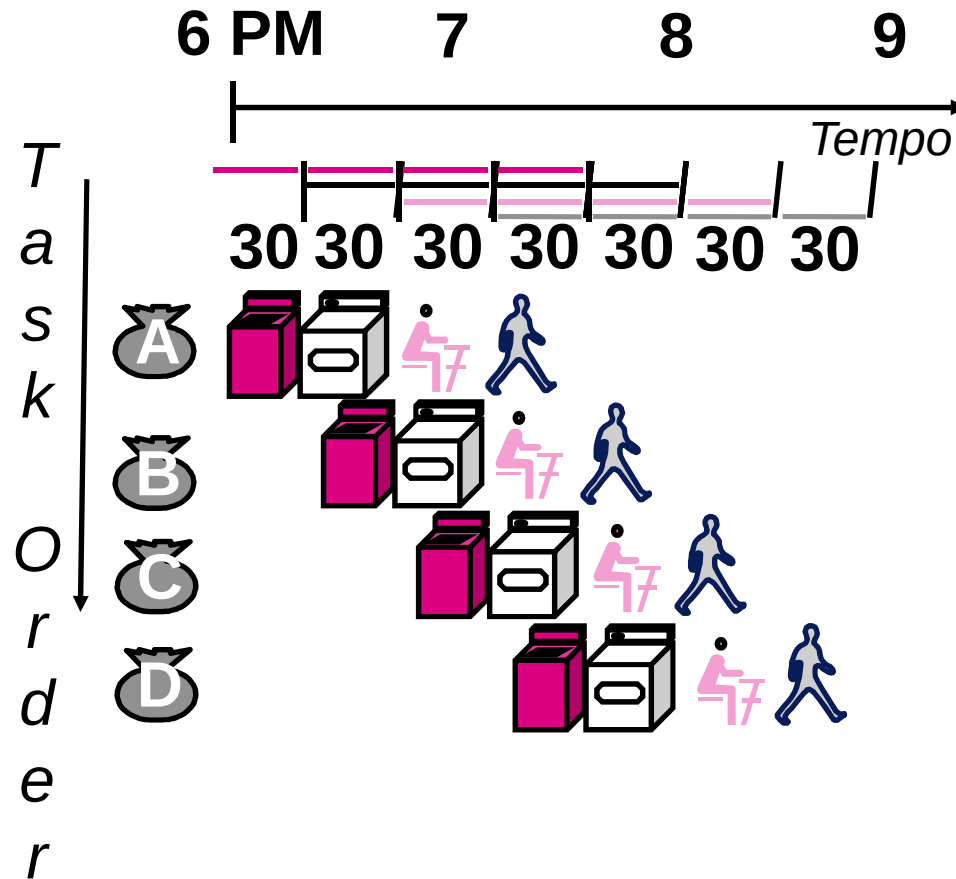
- Lavanderia sequenziale: 8 ore per 4 carichi
- Se utilizzassero il pipelining, quanto impiegherebbero?

Lavanderia a Pipeline: "Start work ASAP"



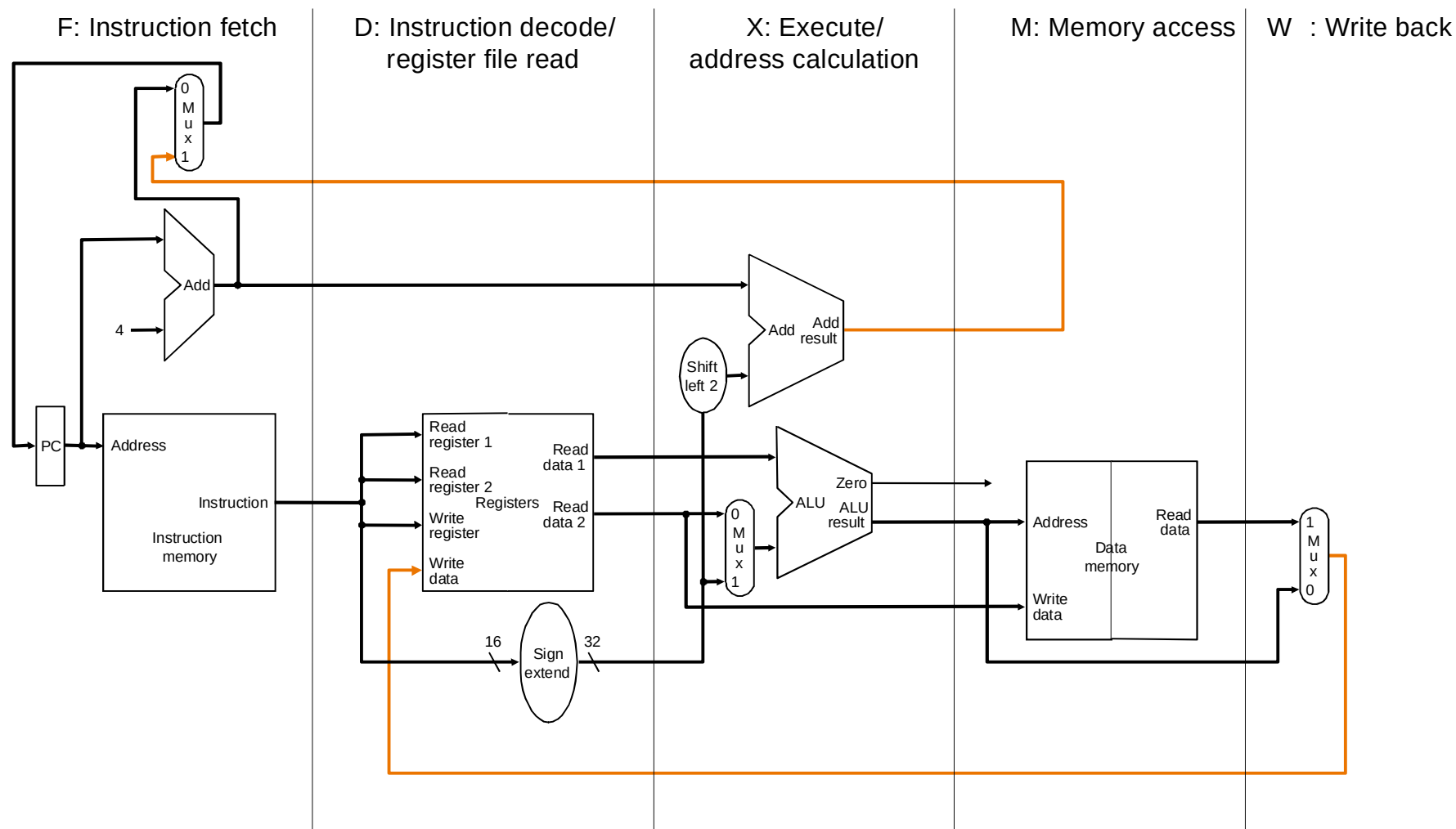
- Lavanderia a pipeline: 3.5 ore per 4 carichi!

Le lezioni di pipelining



- Il pipelining non riduce il tempo di esecuzione di ogni singola istruzione, bensì aumenta il loro **throughput**
- Tutti gli stadi lavorano in contemporanea utilizzando risorse separate
- Incremento potenziale di velocità = **numero di stadi della pipeline**
- La velocità della pipeline **e' limitata dallo stadio più lento**
- Gli stadi della pipeline **non bilanciati** riducono l'incremento di velocità
- Il tempo per **riempire** la pipeline ed il tempo per **svuotarla** riducono l'incremento di velocità
- **Stallo per dipendenze**

Idea di base: suddivisione in stadi della CPU



Cosa dobbiamo aggiungere per suddividere il datapath in stadi?



I cinque passi per eseguire un'istruzione

- **F** = Prelievo dell'istruzione (Instruction Fetch)
- **D** = Decodifica dell'istruzione (Instruction Decode) e lettura dei registri (Register Fetch)
- **X** = Esecuzione, calcolo dell'indirizzo di memoria, completamento del salto (Branch Completion)
- **M** = Accesso alla memoria o completamento di istruzione di tipo R
- **W** = Scrittura del risultato in un registro (Write-Back)

LE ISTRUZIONI RICHIEDONO 3 - 5 CICLI!

Passo 1: Instruction Fetch

- Viene letta dalla memoria l'istruzione puntata dal Program Counter (PC)
- L'istruzione viene posta nell' Instruction Register (IR)
- PC viene incrementato di 4 e riscritto nuovamente in PC
- Tali operazioni si possono descrivere utilizzando un linguaggio RTL (Register-Transfer Language)

$$\begin{aligned} \text{IR} &= \text{Memory}[\text{PC}]; \\ \text{PC} &= \text{PC} + 4; \end{aligned}$$

- Da IR si determinano i valori dei segnali di controllo

Qual e' il vantaggio di aggiornare il PC in questo passo?

Passo 2: Instruction Decode e Register Fetch

- I contenuti dei registri indicati nei campi rs e rt vengono letti e trasferiti nei registri di appoggio A e B
 - Non e' detto che siano successivamente utilizzati...
- Calcolo dell' "indirizzo target" del salto, nel caso in cui l'istruzione sia un salto
- RTL:

```
A = Reg[IR[25-21]];
B = Reg[IR[20-16]];
ALUOut = PC + (sign-extend(IR[15-0]) << 2);
```

- Le linee di controllo non assumono alcun valore in dipendenza dal tipo di istruzione (per il momento, dato che la decodifica e' ancora in corso)

Sign-extend effettua l'estensione da 16 a 32 bit **con segno**

Passo 3 (dipendente dall'istruzione)

- La ALU esegue una delle seguenti tre funzioni, in base al tipo di istruzione:

- Accesso alla memoria:

$$\text{ALUOut} = A + \text{sign-extend}(\text{IR}[15-0]);$$

- Istruzione di tipo R relativa ad una operazione op:

$$\text{ALUOut} = A \text{ op } B;$$

- Salto:

$$\text{if } (A == B) \text{ PC} = \text{ALUOut};$$

Passo 4 (R-type o accesso alla memoria)

- Le istruzioni load e store accedono alla memoria

$\text{MDR} = \text{Memory}[\text{ALUOut}]; \quad (\text{load})$

$\text{Memory}[\text{ALUOut}] = B; \quad (\text{store})$

- Le istruzioni di tipo R possono scrivere il risultato

$\text{Reg}[\text{IR}[15-11]] = \text{ALUOut};$

La scrittura avviene alla fine del ciclo, sul fronte in discesa

Passo 5: Write-Back

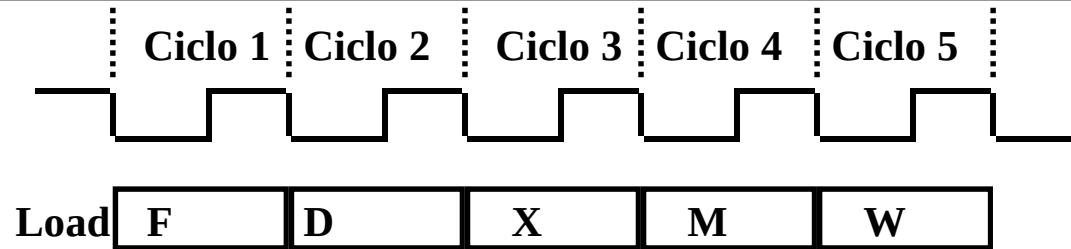
Reg[IR[20-16]] = MDR;

**Cosa succede alle
altre istruzioni?**

Riassumendo

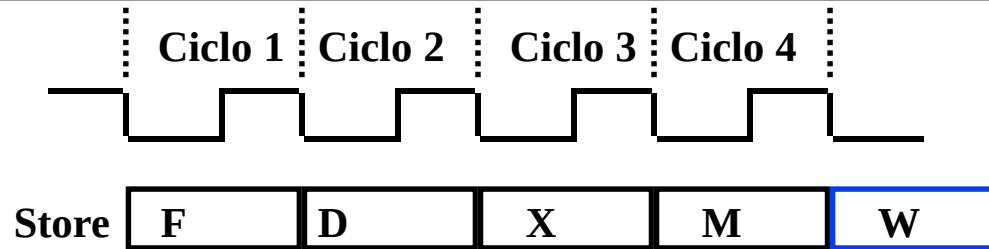
Nome del passo	Operazioni per le istruzioni di tipo R	Operazioni per le istruzioni che accedono alla memoria	Operazioni per branch	Operazioni per jump
Prelievo dell'istruzione	$IR = \text{Memory}[PC]$ $PC = PC + 4$			
Decodifica dell'istruzione /lettura dei registri	$A = \text{Reg}[IR[25-21]]$ $B = \text{Reg}[IR[20-16]]$ $ALUOut = PC + (\text{sign-extend}(IR[15-0]) \ll 2)$			
Esecuzione, calcolo del indirizzo, completamento di branch/jump	$ALUOut = A \text{ op } B$	$ALUOut = A + \text{sign-extend}(IR[15-0])$	if (A ==B) then $PC = ALUOut$	$PC = PC[31-28] \parallel (IR[25-0] \ll 2)$
Accesso alla memoria o completamento dell'istruzione di tipo R	$\text{Reg}[IR[15-11]] = ALUOut$	Load: $MDR = \text{Memory}[ALUOut]$ oppure Store: $\text{Memory}[ALUOut] = B$		
Completamento della lettura della memoria		Load: $\text{Reg}[IR[20-16]] = MDR$		

Ciclo Multiplo: i cinque stadi della load



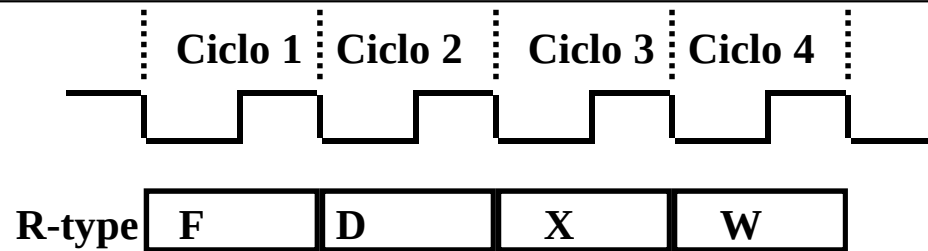
- **F**: Instruction Fetch
 - Prelievo dell'istruzione dalla memoria istruzioni
- **D**: Registers Fetch, Instruction Decode
 - Lettura dei registri e decodifica delle istruzioni
- **X**: Calcolo dell'indirizzo di memoria
- **M**: Lettura dei dati dalla memoria dati
- **W**: Scrittura dei dati nel register file

Ciclo Multiplo: i quattro stadi della store



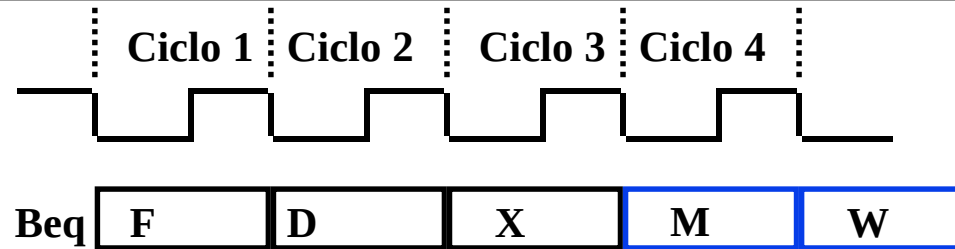
- **F**: Instruction Fetch
 - Prelievo dell'istruzione dalla memoria istruzioni
- **D**: Registers Fetch, Instruction Decode
- **X**: Calcolo dell'indirizzo di memoria
- **M**: Scrittura dei dati in memoria dati

Ciclo Multiplo: i quattro stadi dell'istruzione di tipo R



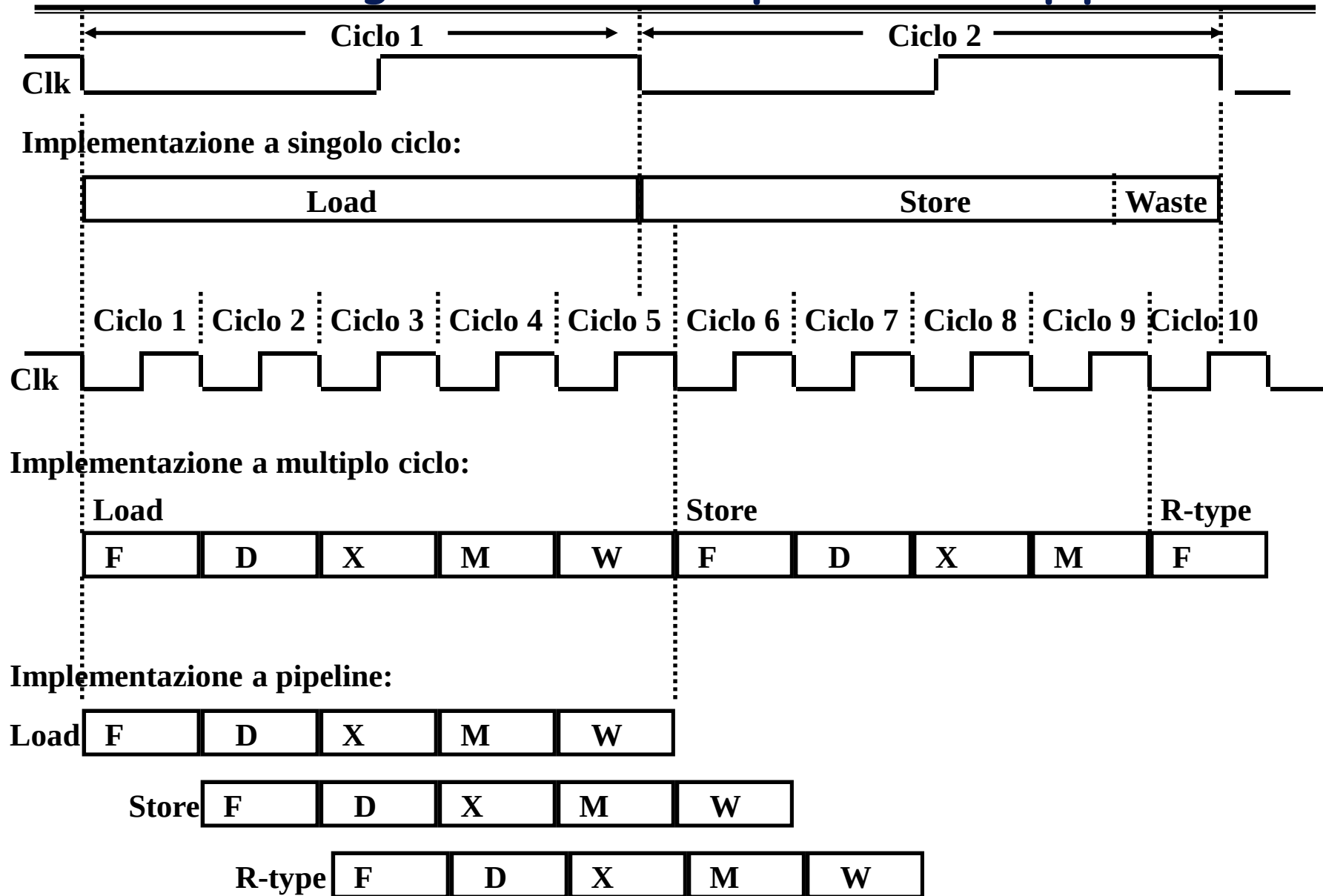
- **F**: Instruction Fetch
 - Prelievo dell'istruzione dalla memoria istruzioni
- **D**: Registers Fetch, Instruction Decode
- **X**:
 - ALU opera su due operandi in registri
 - Aggiornamento del PC
- **W**: Scrittura del risultato dall'ALU nel register file

Ciclo Multiplo: i tre stadi della Beq



- **F**: Instruction Fetch
 - Prelievo dell'istruzione dalla memoria istruzioni
- **D**:
 - Registers Fetch, Instruction Decode
- **X**:
 - confronto dei due operandi in registri,
 - selezione dell'indirizzo di salto corretto
 - memorizzazione dell'indirizzo in PC

Modelli a singolo ciclo e multiplo ciclo vs. pipeline

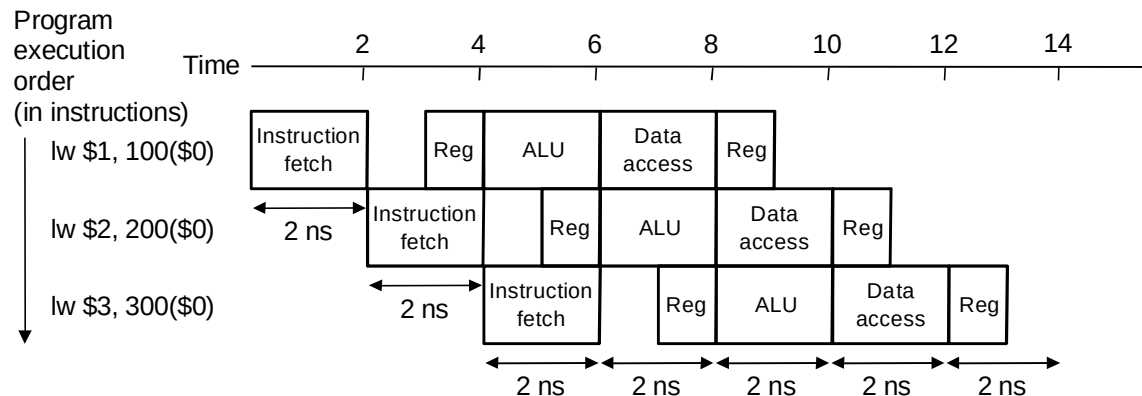
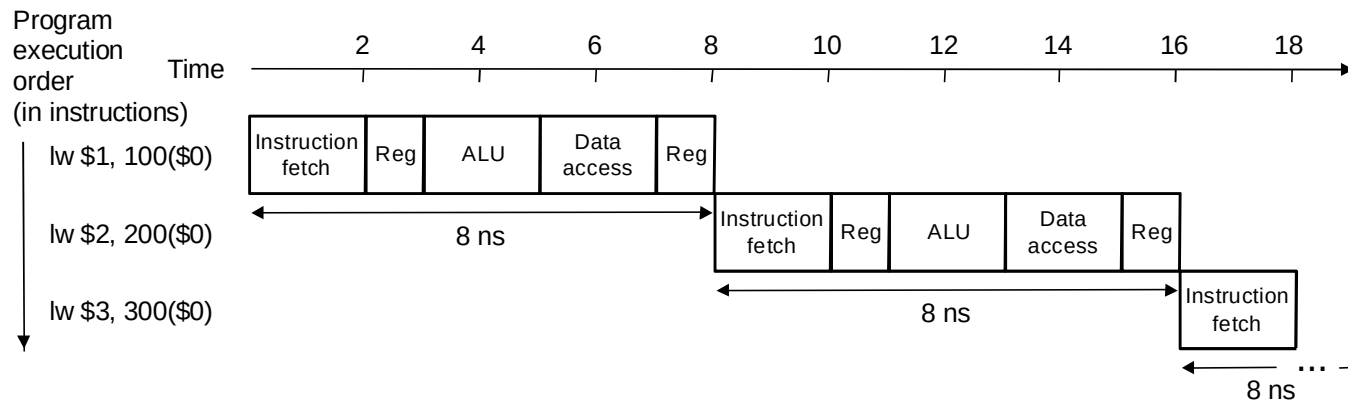


Esercizio: verificare la bontà' del pipelining

- Supponiamo che si eseguano 100 istruzioni
- Macchina a singolo ciclo
 - $45 \text{ ns/ciclo} \times 1 \text{ CPI} \times 100 \text{ inst} = 4500 \text{ ns}$
- Macchina a multiplo ciclo
 - $10 \text{ ns/ciclo} \times 4.6 \text{ CPI (dovuto al mix di istruzioni)} \times 100 \text{ inst} = 4600 \text{ ns}$
- Macchina con pipeline ideale
 - $10 \text{ ns/ciclo} \times (1 \text{ CPI} \times 100 \text{ inst} + 4 \text{ cicli per riempimento} + 4 \text{ cicli per svuotamento}) = 1080 \text{ ns}$

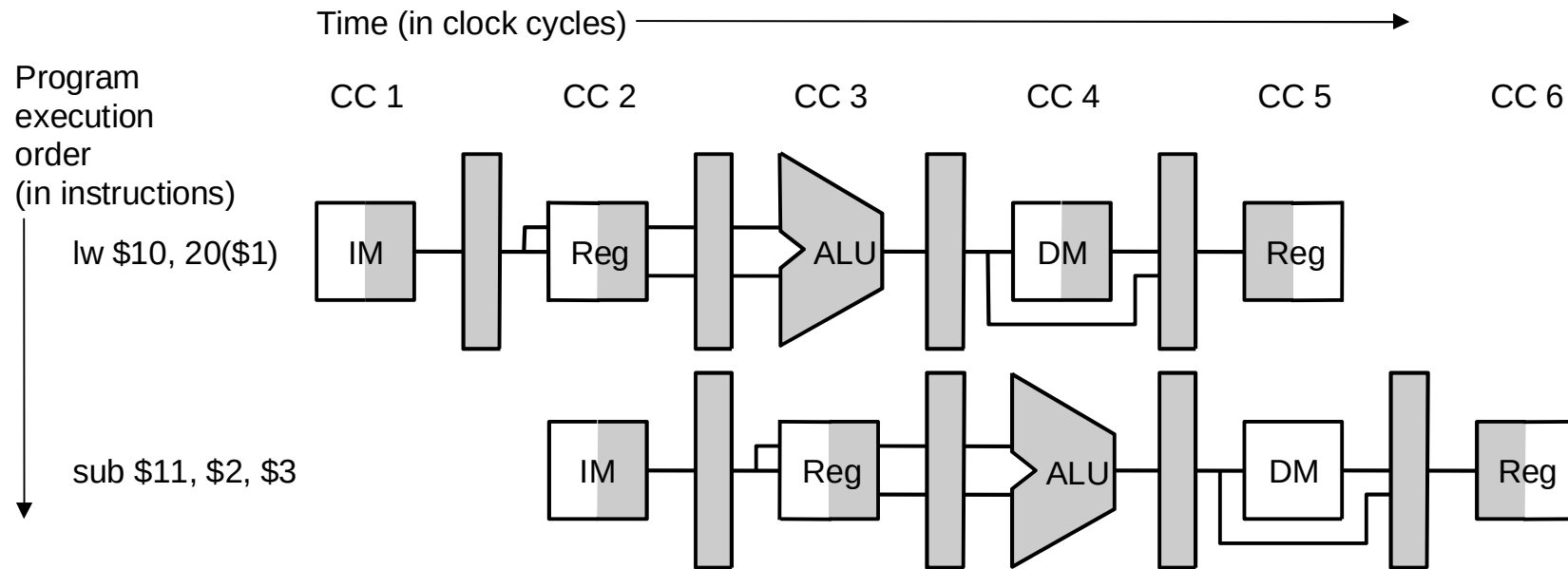
Il Pipelining

- Migliora le prestazioni aumentando il throughput delle istruzioni



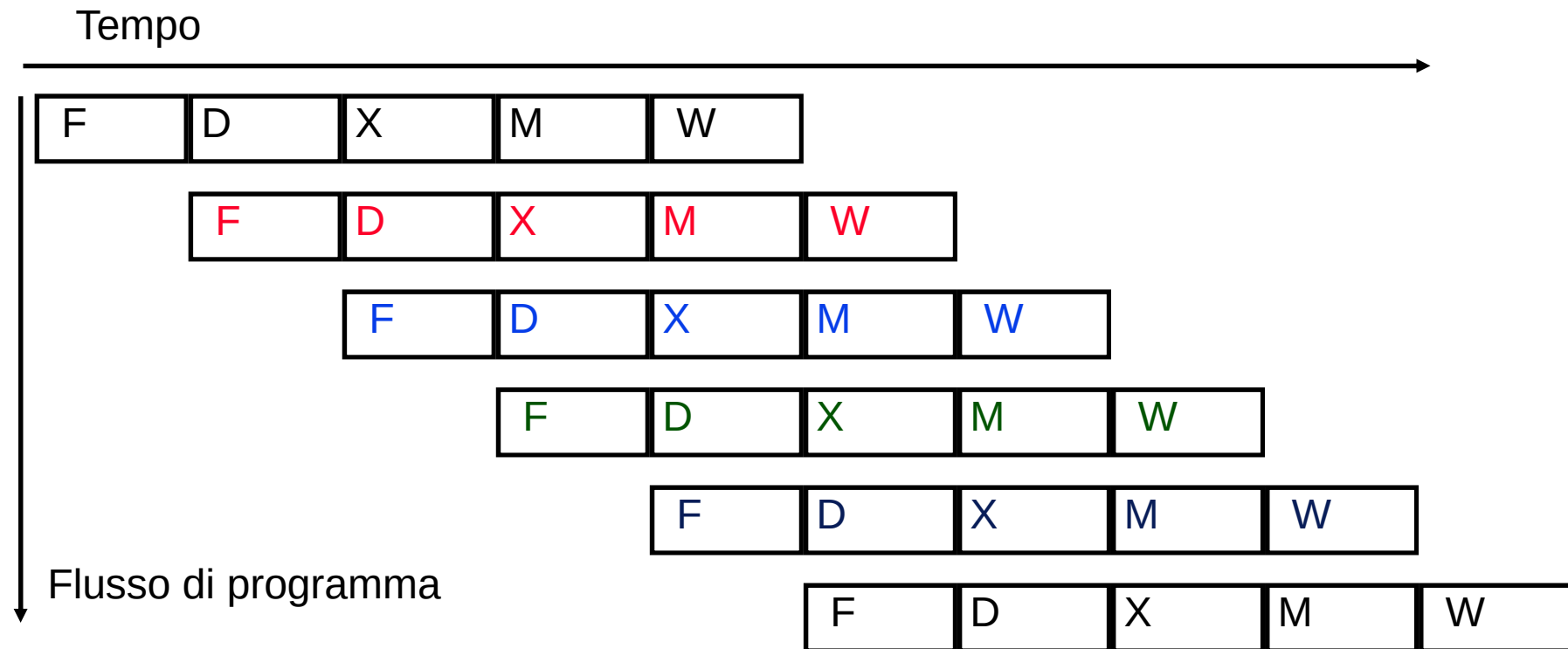
***L' aumento di velocita' ideale e' pari al numero di stadi della pipeline.
Riusciamo ad ottenerlo?***

Rappresentazione grafica delle pipeline

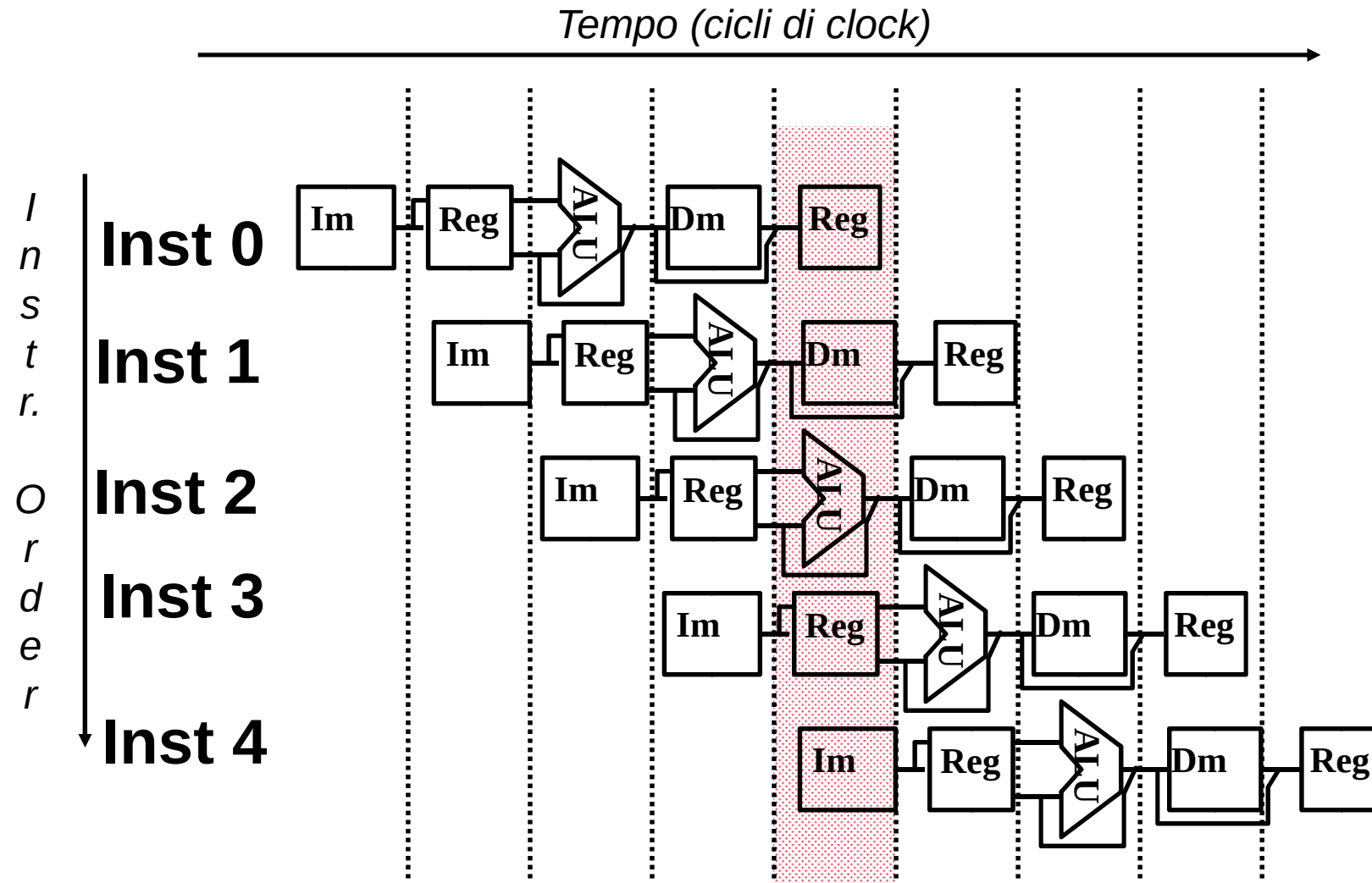


- L'uso della rappresentazione grafica aiuta a capire il datapath:
 - quanti cicli utilizza l'esecuzione di questo codice?
 - cosa fa l'ALU durante il ciclo 4?

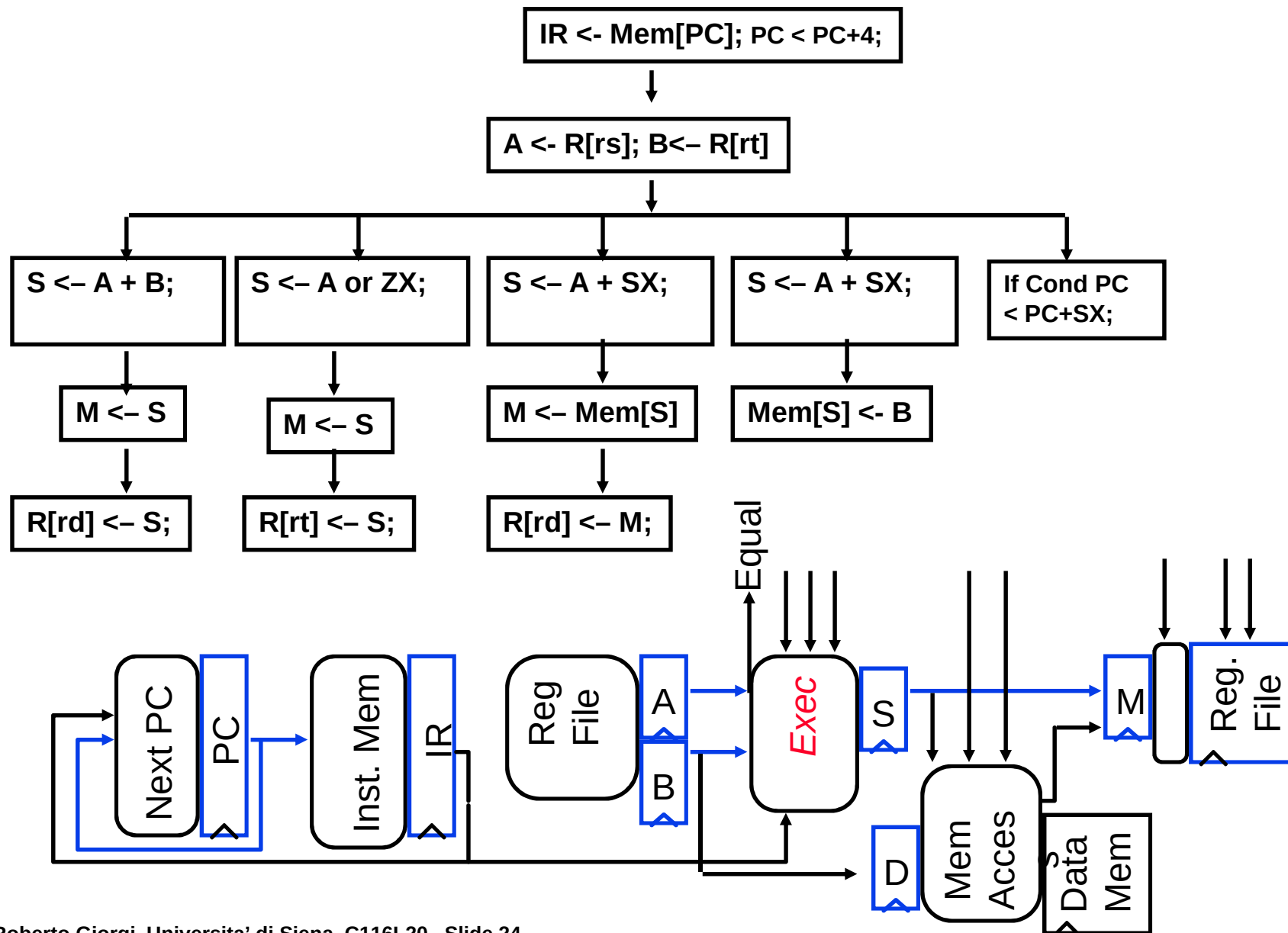
Rappresentazione convenzionale dell'esecuzione a pipeline



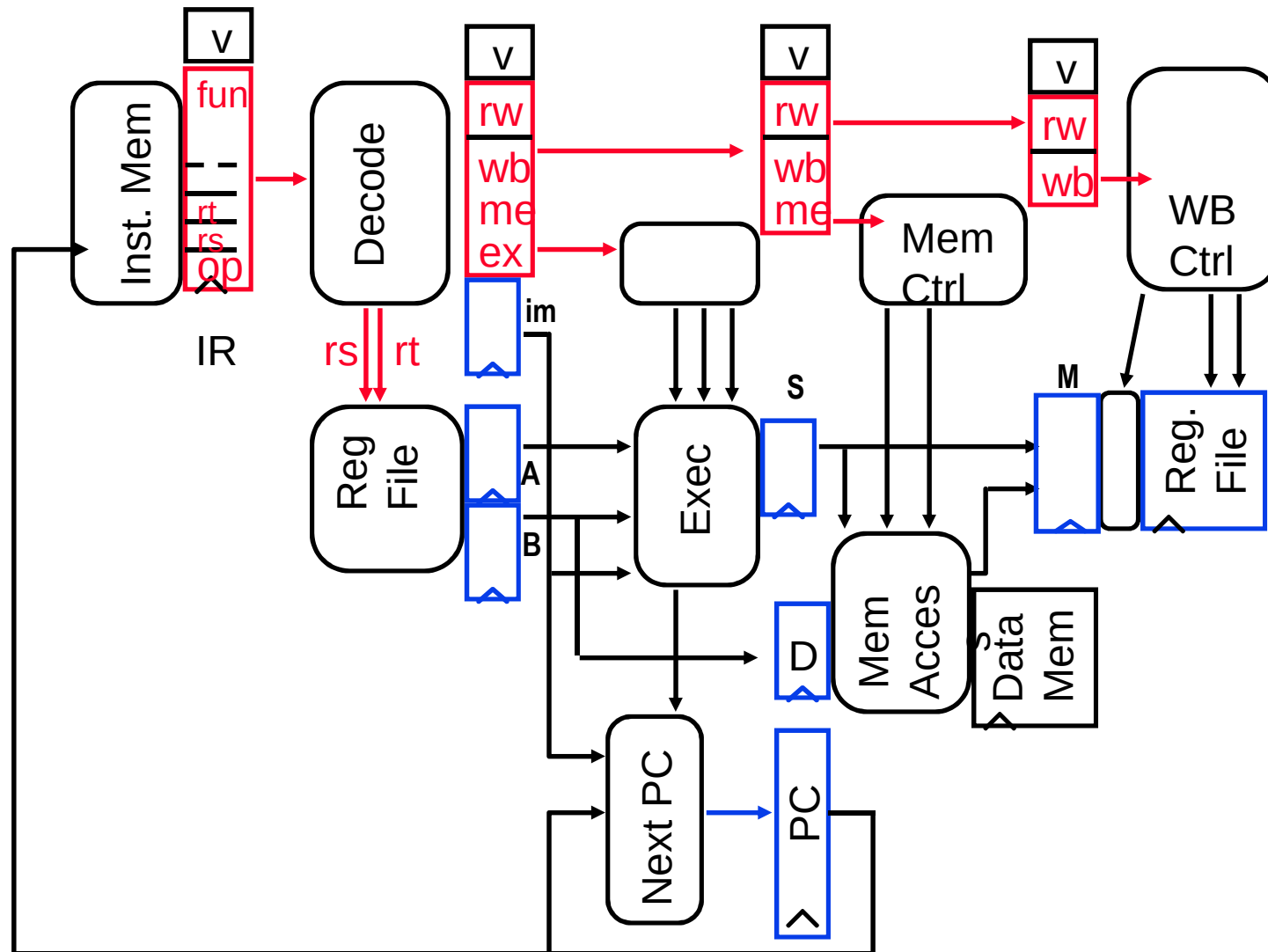
Perche' la pipeline? Utilizzo ottimale delle risorse!



Datapath semplificato



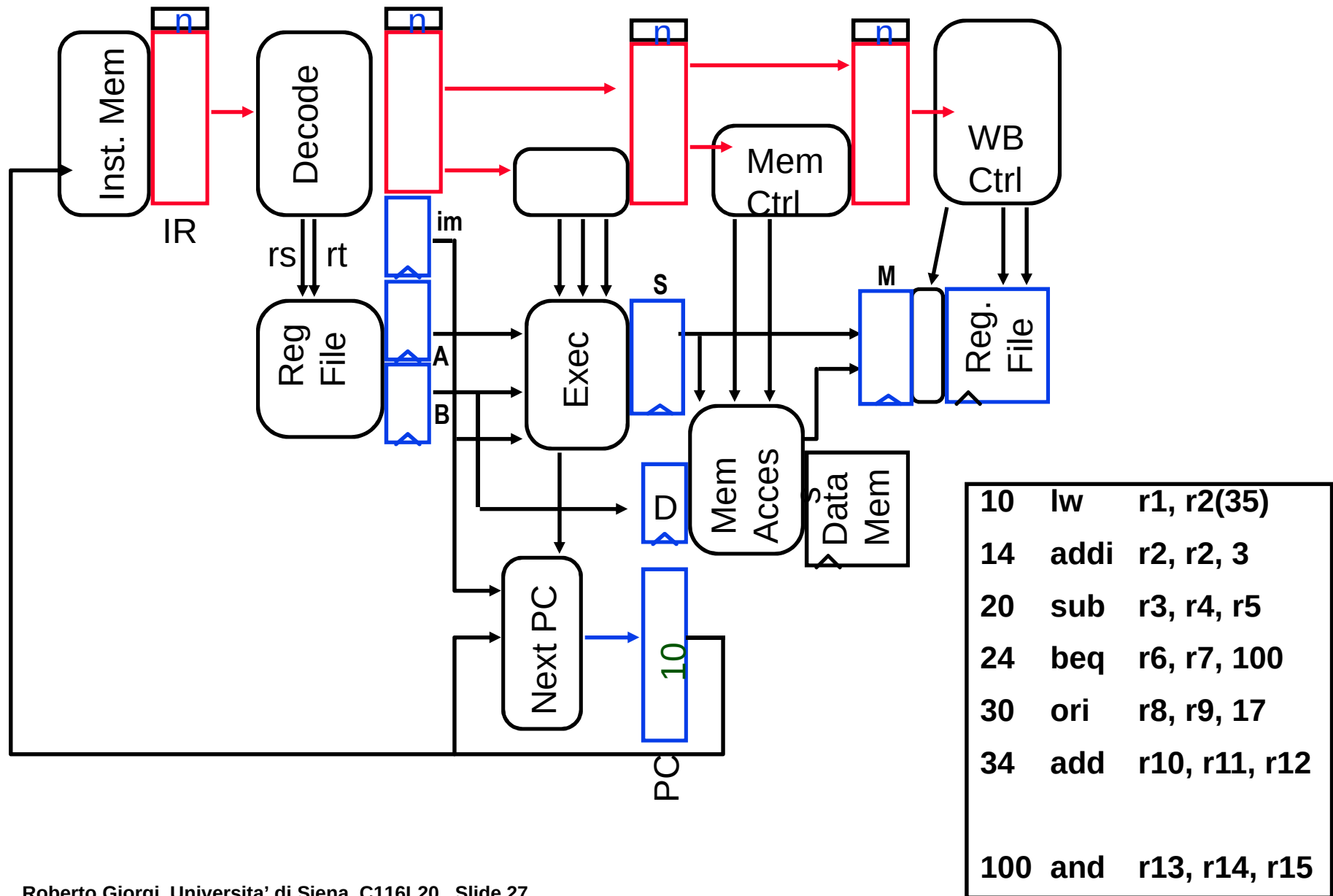
Datapath + Data Stationary Control



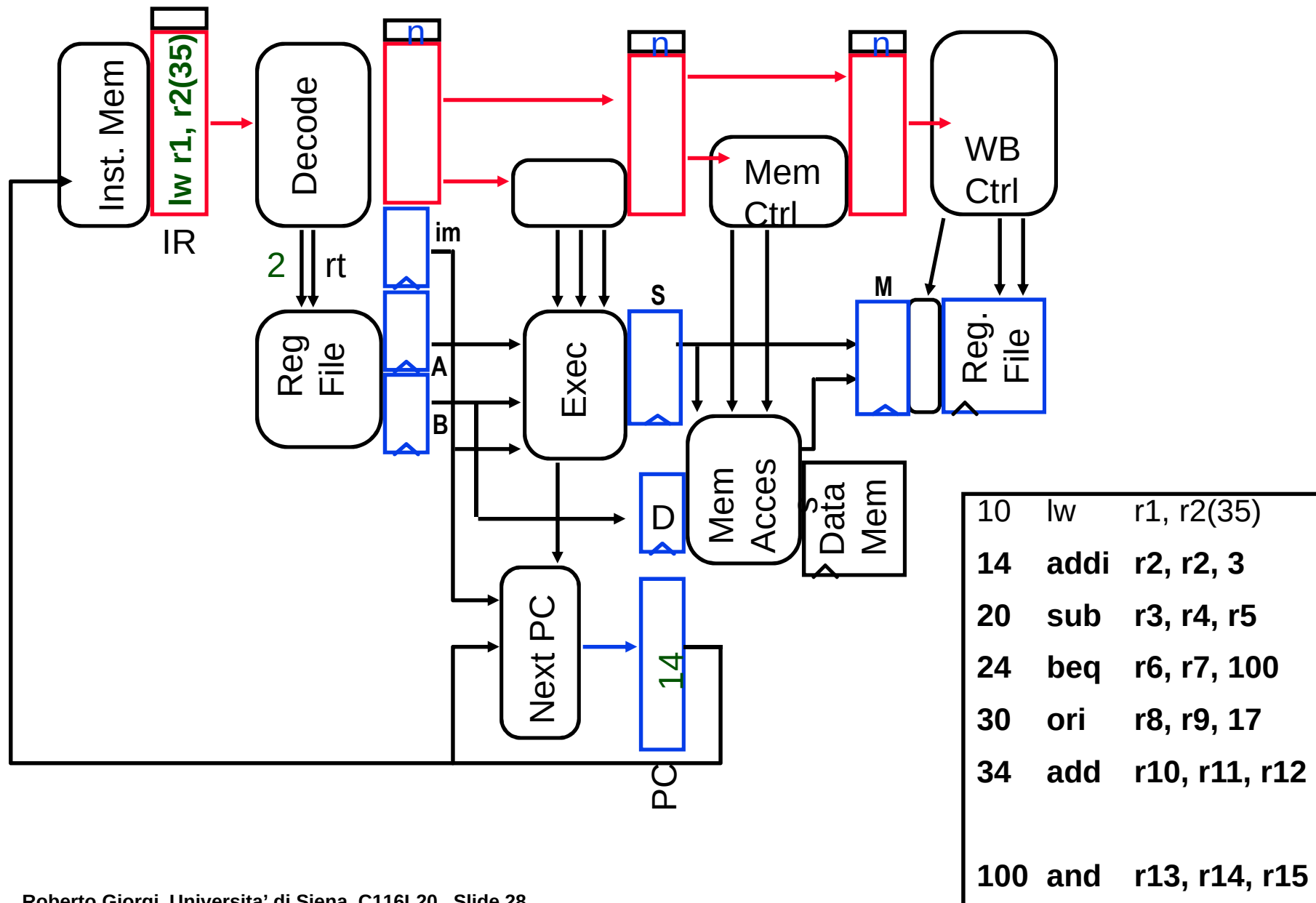
Proviamolo!

10	lw	r1, r2(35)
14	addi	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, r15

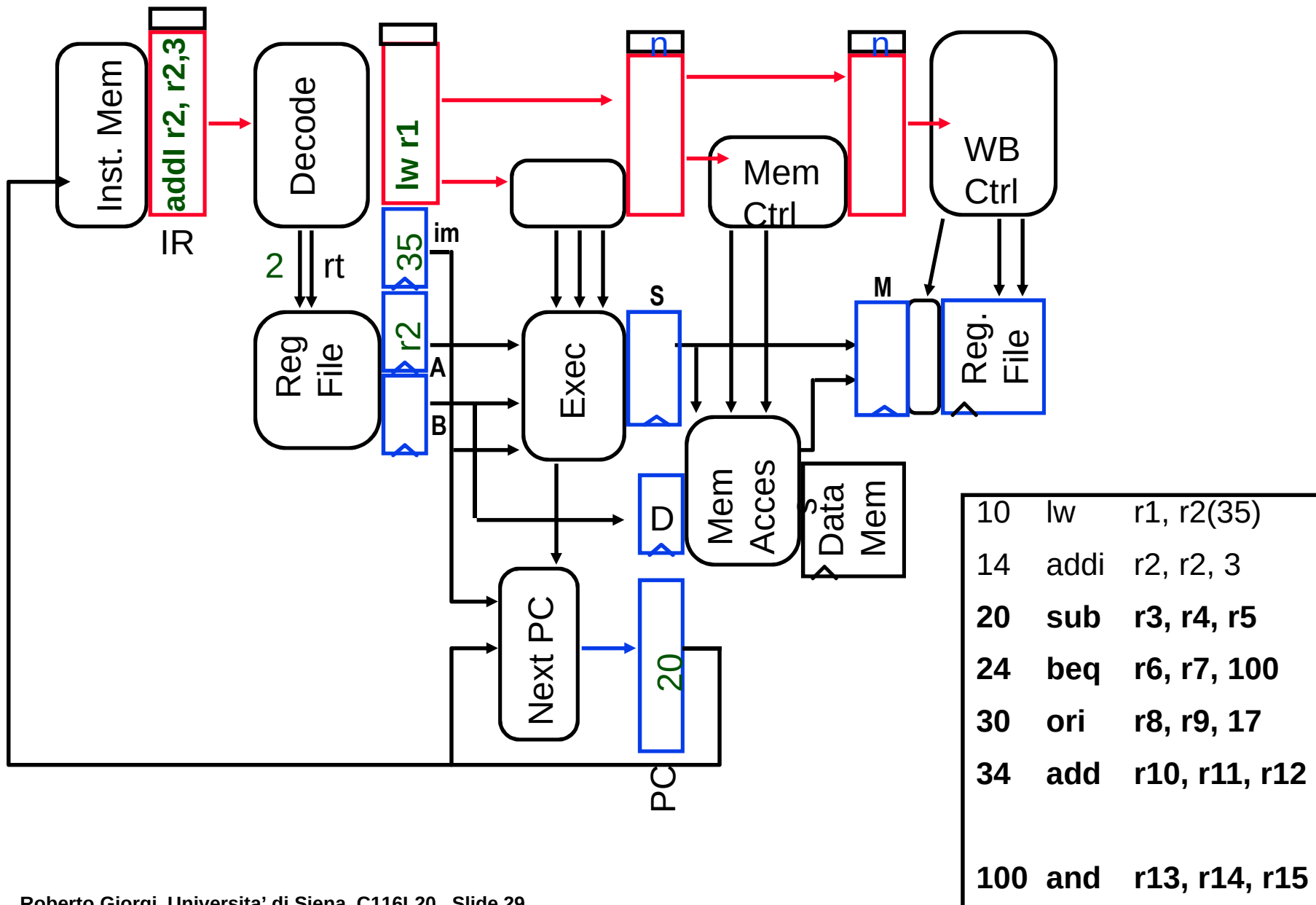
Start: Fetch 10



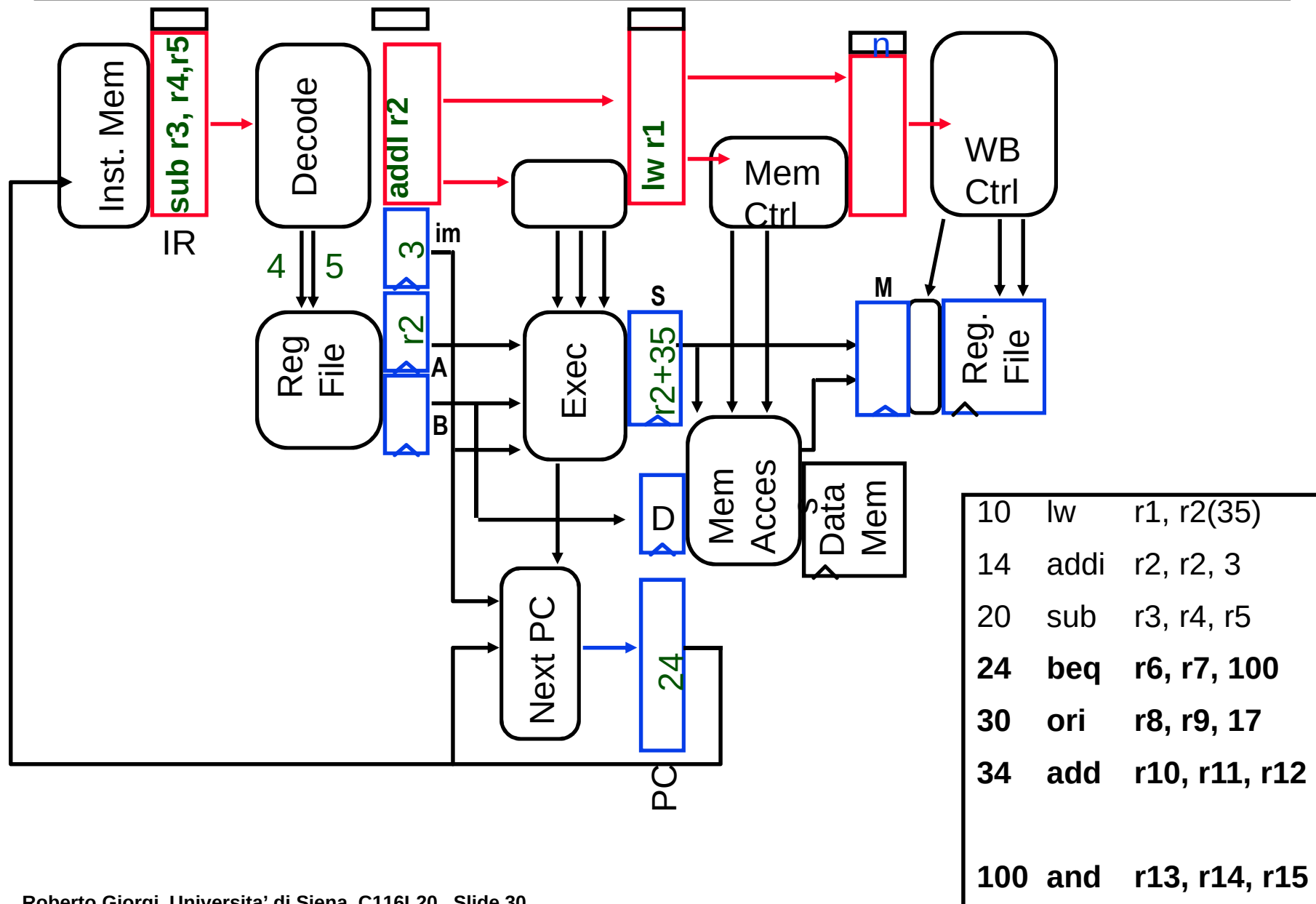
Fetch 14, Decode 10



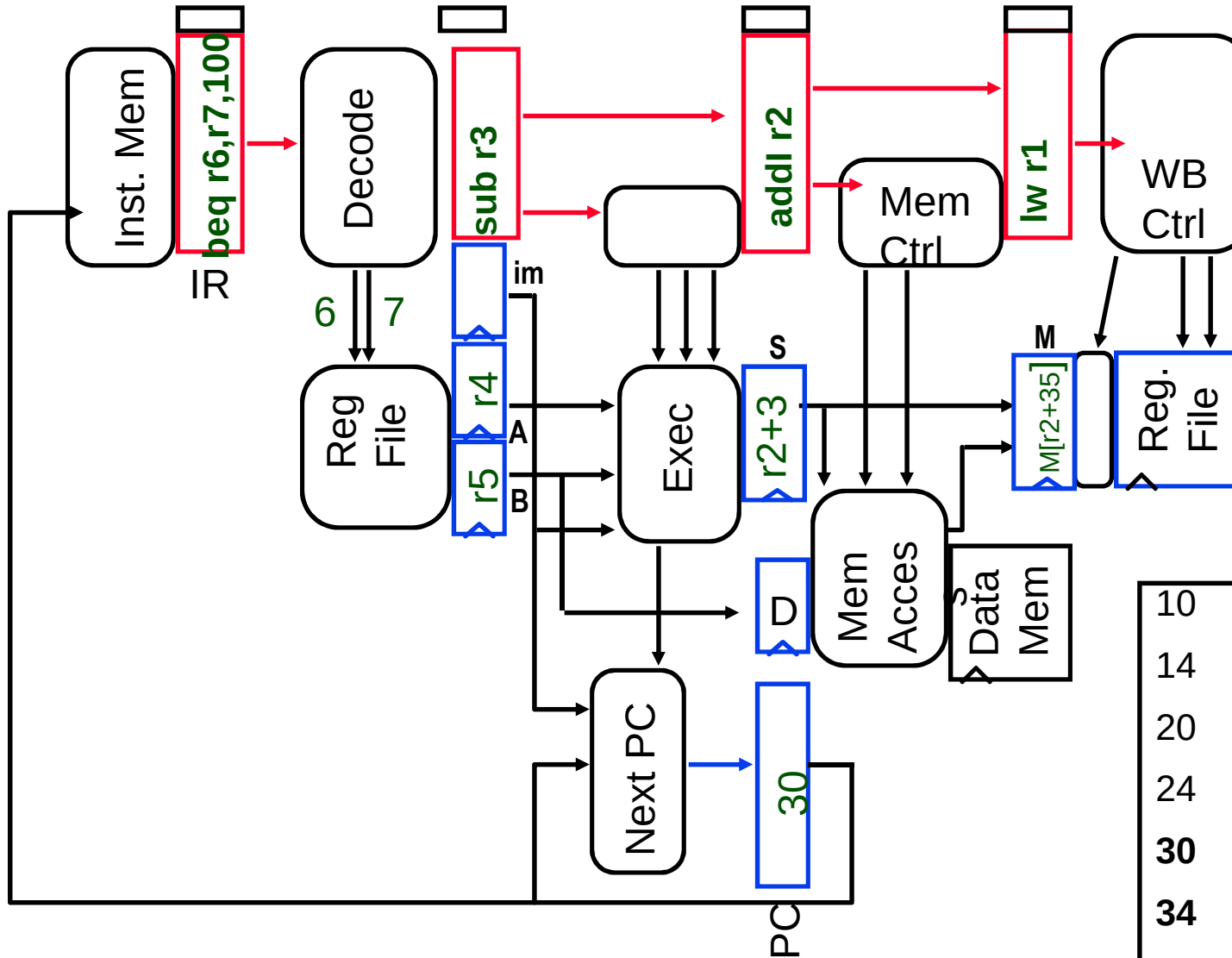
Fetch 20, Decode 14, Exec 10



Fetch 24, Decode 20, Exec 14, Mem 10

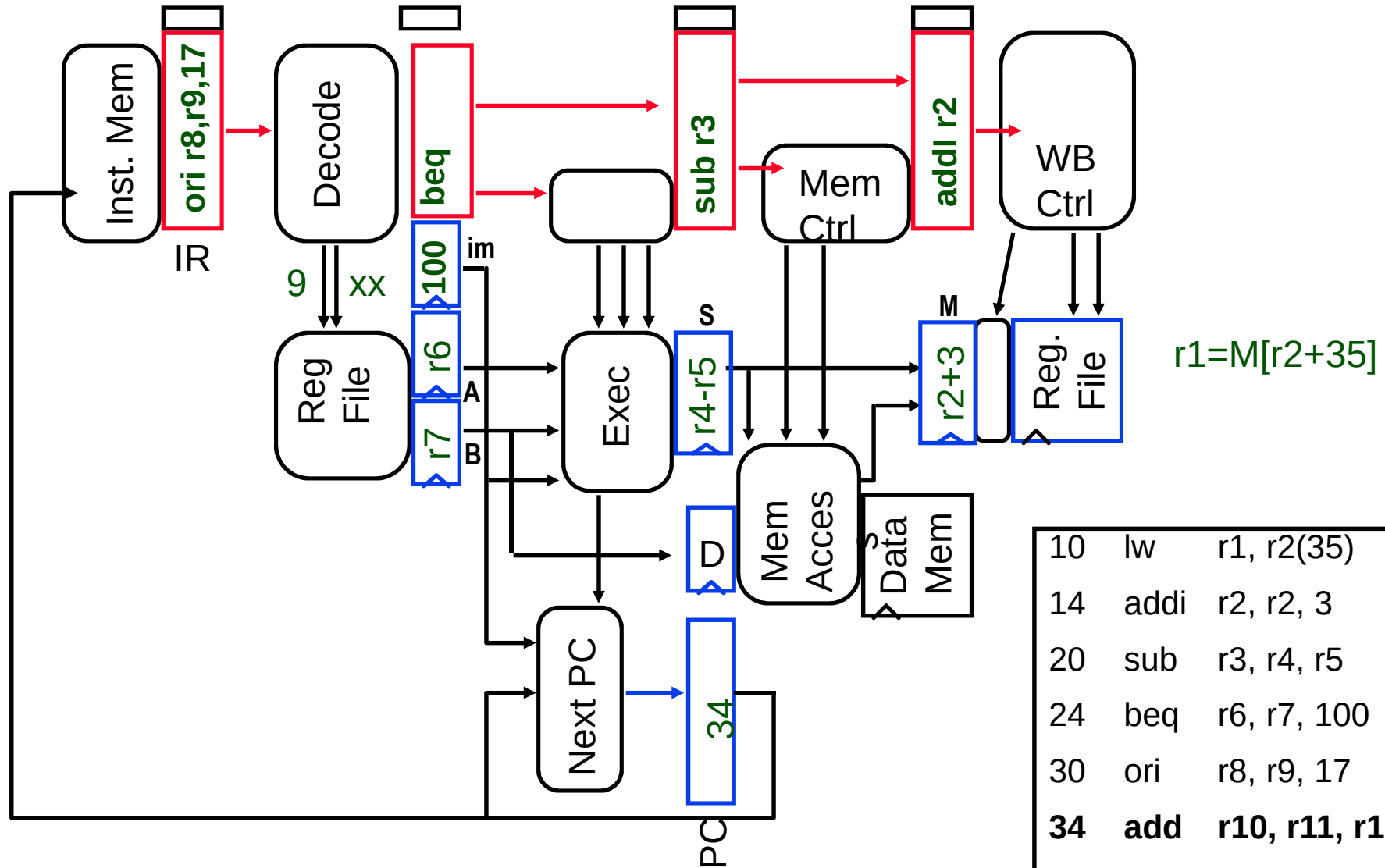


Fetch 30, Dcd 24, Ex 20, Mem 14, WB 10



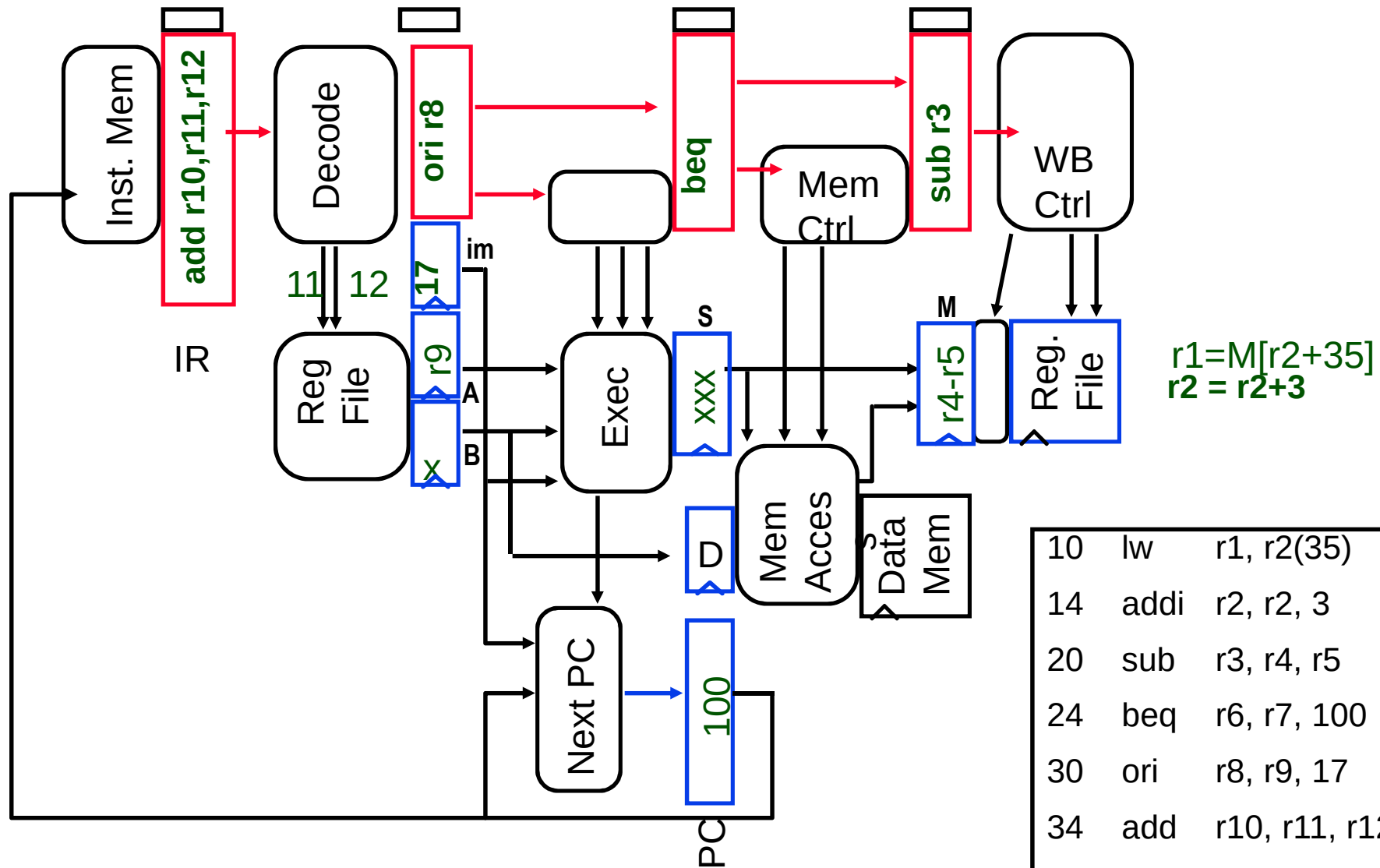
10	lw	r1, r2(35)
14	addi	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, r15

Fetch 34, Dcd 30, Ex 24, Mem 20, WB 14



10	lw	r1, r2(35)
14	addi	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, r15

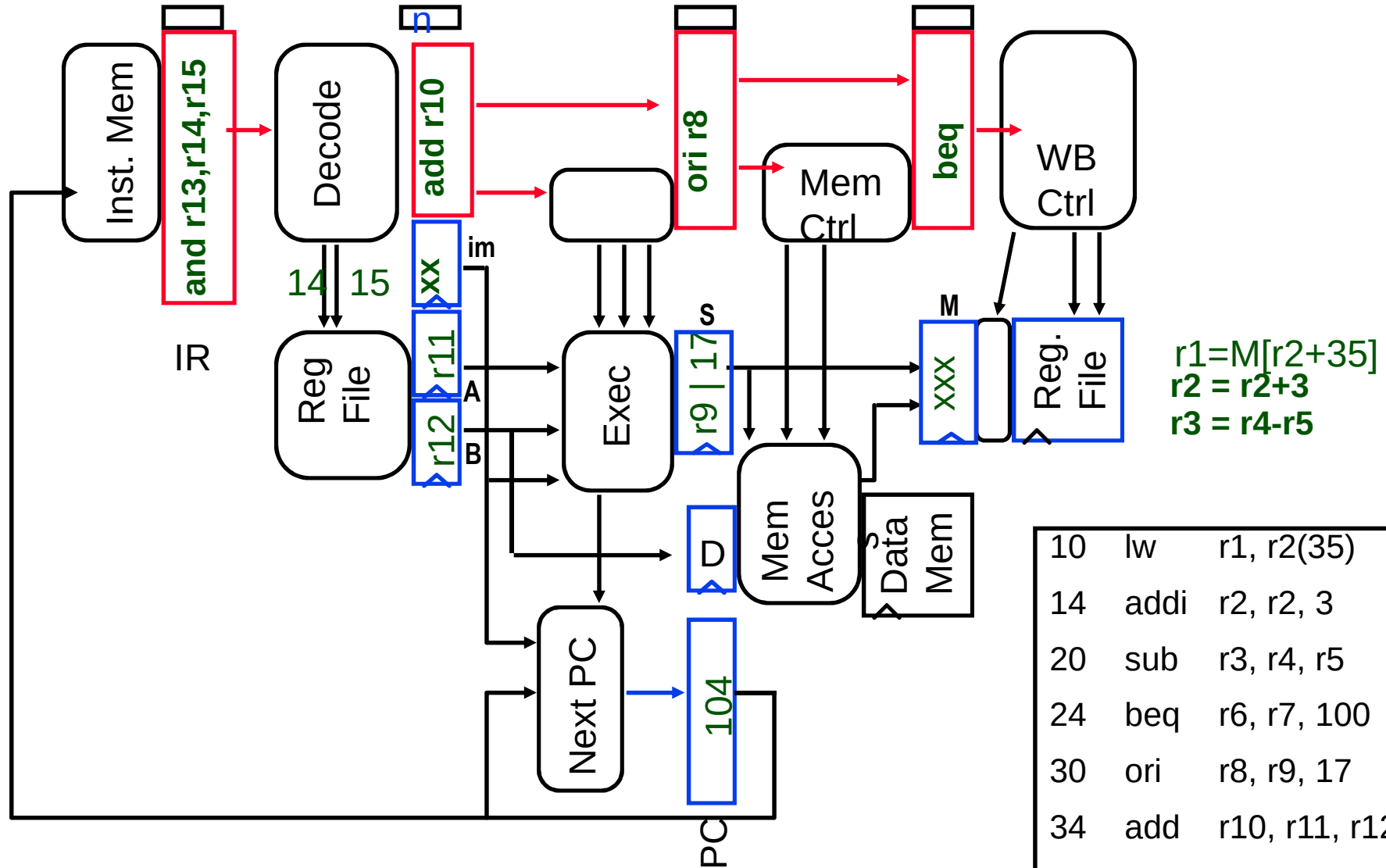
Fetch 100, Dcd 34, Ex 30, Mem 24, WB 20



$r1 = M[r2 + 35]$
 $r2 = r2 + 3$

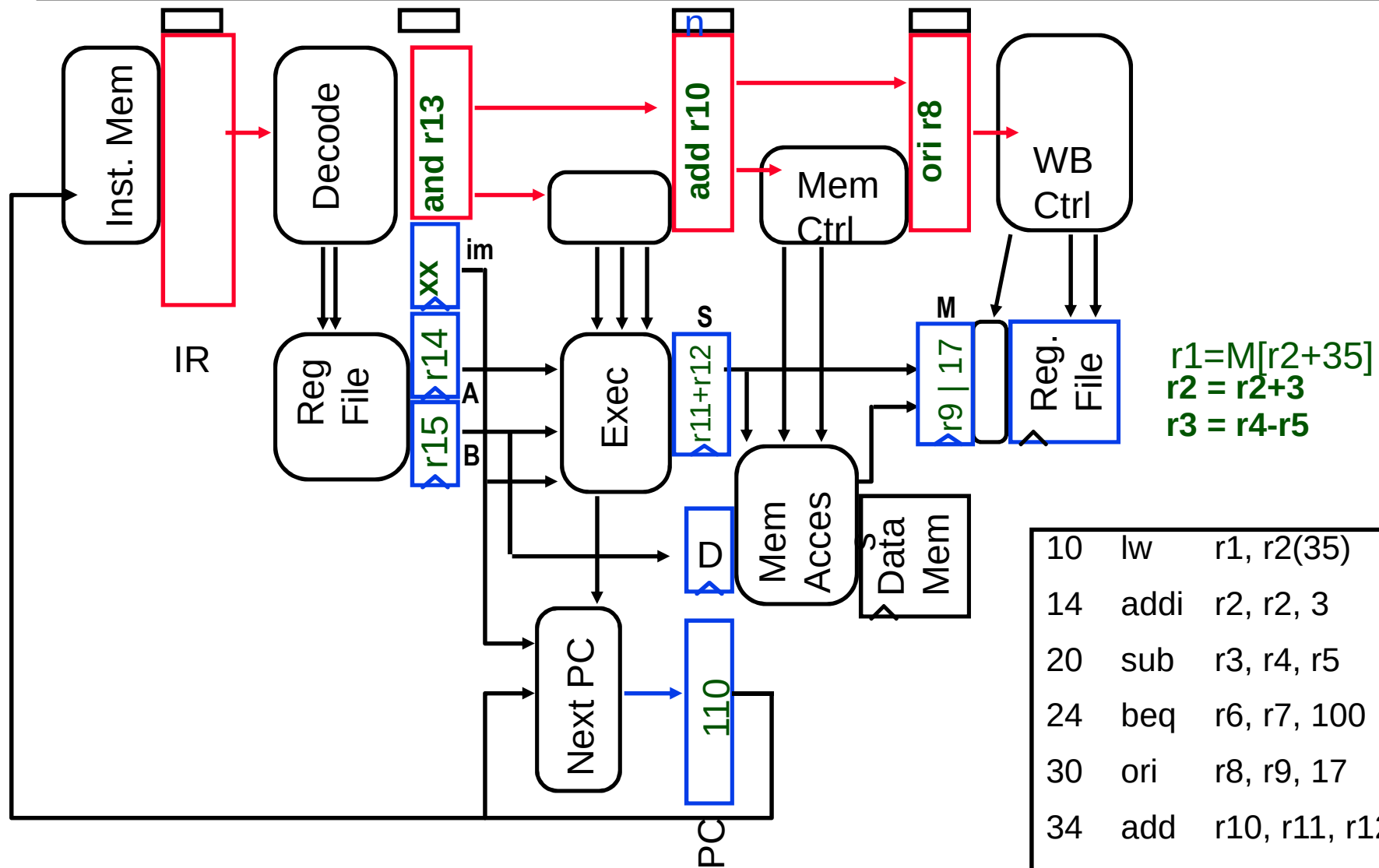
10	lw	r1, r2(35)
14	addi	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, r15

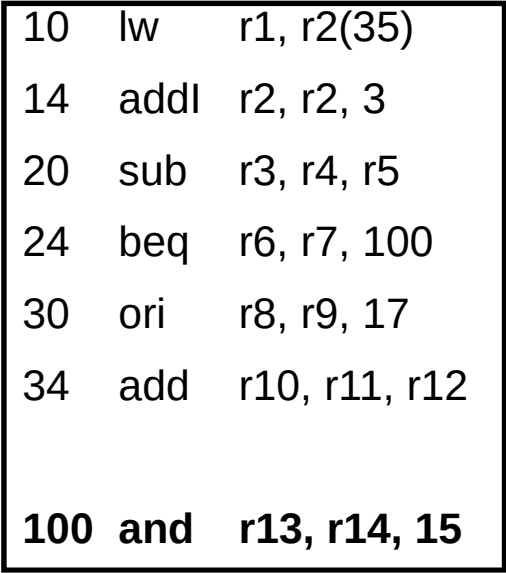
Fetch 104, Dcd 100, **Ex 34**, Mem 30, WB 24



10	lw	r1, r2(35)
14	addi	r2, r2, 3
20	sub	r3, r4, r5
24	beq	r6, r7, 100
30	ori	r8, r9, 17
34	add	r10, r11, r12
100	and	r13, r14, r15

Fetch 108, Dcd 104, Ex 100, **Mem 34**, WB 30



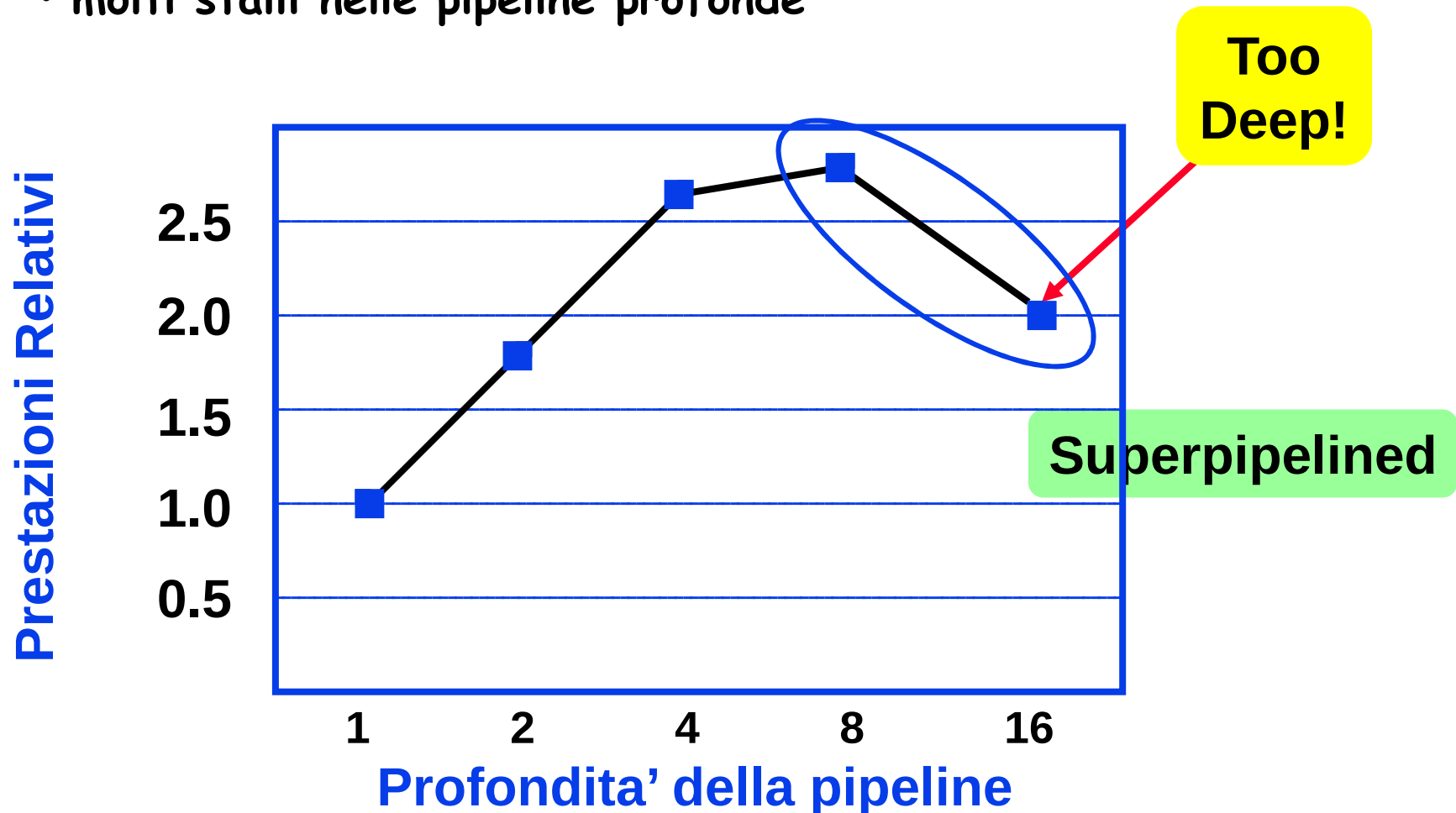


Il pipelining puo' provocare problemi?

- Si': **Criticita' nelle pipeline**
 - **criticita' strutturali**: tentativo di utilizzare le stesse risorse due volte allo stesso tempo
 - Esempio: utilizzo di un'unica macchina lavatrice/asciugatrice anziche' due macchine separate, oppure risorsa impegnata in qualche altro compito
 - **criticita' sui dati**: tentativo di utilizzare un dato prima che sia prodotto
 - Esempio: cerco di effettuare la stiratura prima di avere effettuato l'asciugatura
 - **criticita' sul controllo**: tentativo di saltare pur avendo caricato istruzioni del ramo non-preso
 - Esempio: cerco di fare l'asciugatura a 100 gradi invece che a 30 gradi perche' non ho verificato che tipo di capo sto asciugando
- Le criticita' si possono sempre risolvere **aspettando**, il controllo della pipeline deve:
 - 1) rilevare la criticita'
 - 2) gestirla (o rimandare l'operazione) per risolverla

Profondita' delle pipeline

- La pipeline non puo' essere troppo profonda
- Criticita' sono frequenti
 - molti stalli nelle pipeline profonde



Pipeline nei processori CISC (es. x86)

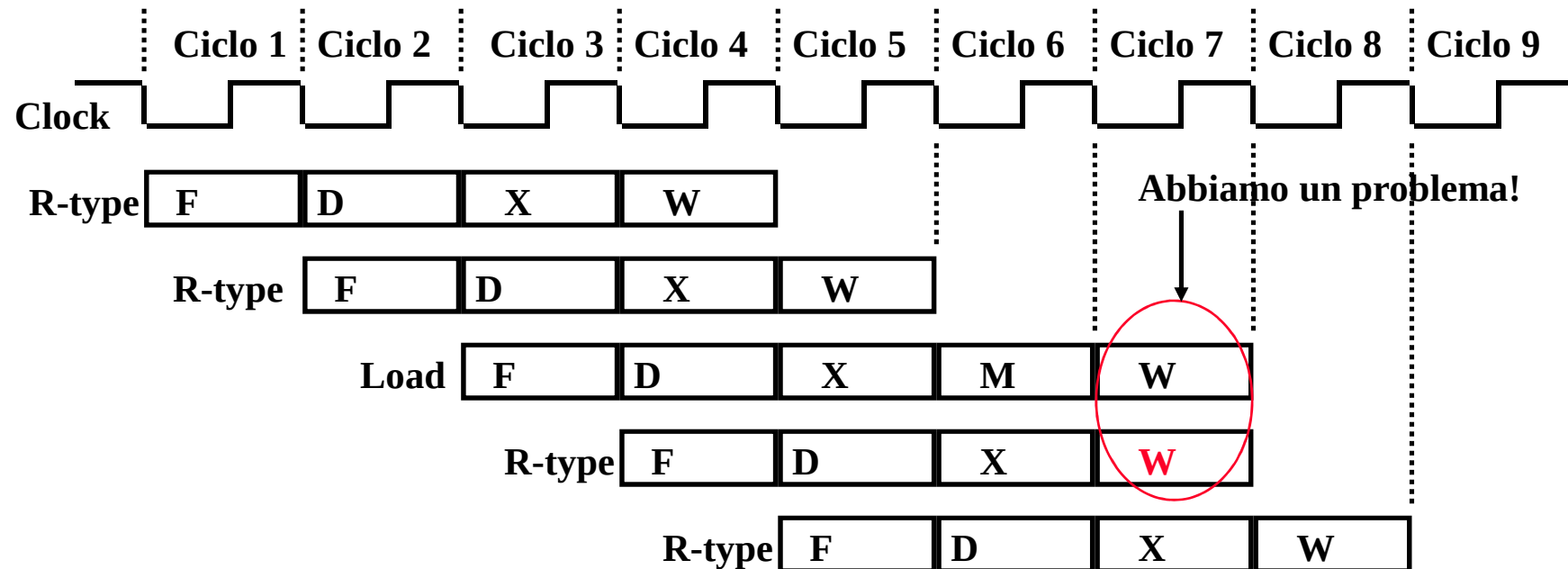
- Praticamente **tutti** i processori oggi usano la pipeline (anche i processori x86)
 - Sovrapposizione degli stadi F, X
- Ostacoli ad un buon funzionamento della pipeline
 - lunghezza variabile delle istruzioni
 - tempo di esecuzione variabile (numero dei cicli di microcodice)
 - sbilanciamento della pipeline
 - "backup" degli stadi di pipe
 - difficoltà del rilevamento delle criticità
 - modi di indirizzamento complessi
 - auto-increment aggiorna address register
 - numerosi accessi alla memoria richiesti
- *Un flusso senza stalli nella pipeline e' molto difficile!*

Ricapitolando

- Il **pipelining** e' un concetto fondamentale
 - Si cerca di non lasciare inutilizzate risorse distinte, schedulando opportunamente le operazioni ad ogni passo
- Per un corretto funzionamento occorre
 - limitare la lunghezza dello stadio piu' lungo
 - rilevare e risolvere criticita'

Criticita' di tipo strutturale

- Supponiamo di avere istruzioni di tipo R a 4 cicli
- Cio' puo' comportare la seguente situazione:

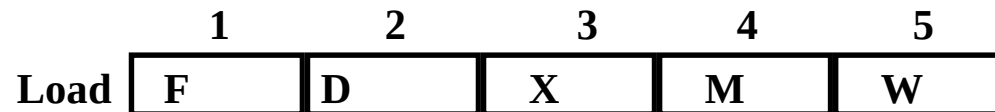


- Due istruzioni stanno tentando di scrivere nei registri contemporaneamente!
 - La porta di scrittura e' unica

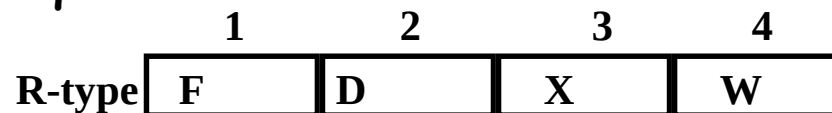
→ Abbiamo una **criticita' strutturale**

Osservazione importante

- Ogni unita' di elaborazione puo' essere utilizzata una volta sola per istruzione
- Ogni unita' di elaborazione deve essere utilizzata nello **stesso** stadio per tutte le istruzioni:
 - La Load utilizza la porta di scrittura del Register File nel suo quinto stadio

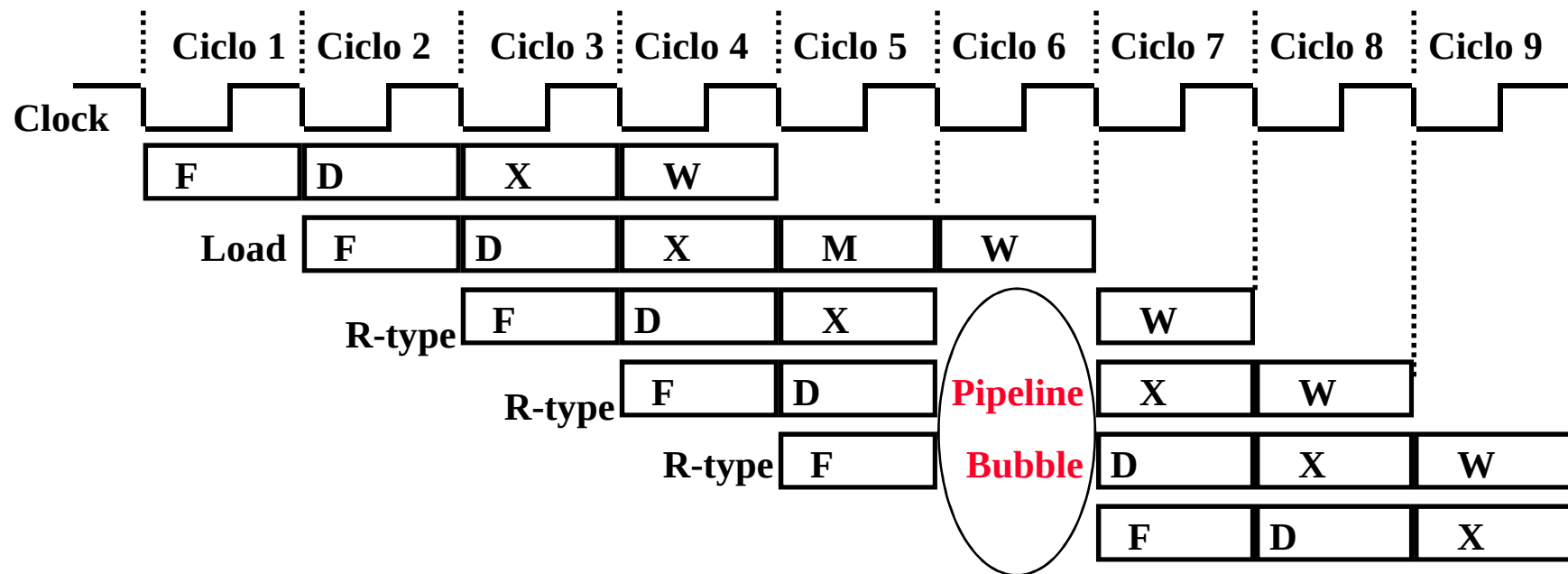


- L'istruzione R-type utilizza la porta di scrittura del Register File nel suo quarto stadio



- *Esistono due modi per risolvere questa criticita' della pipeline.*

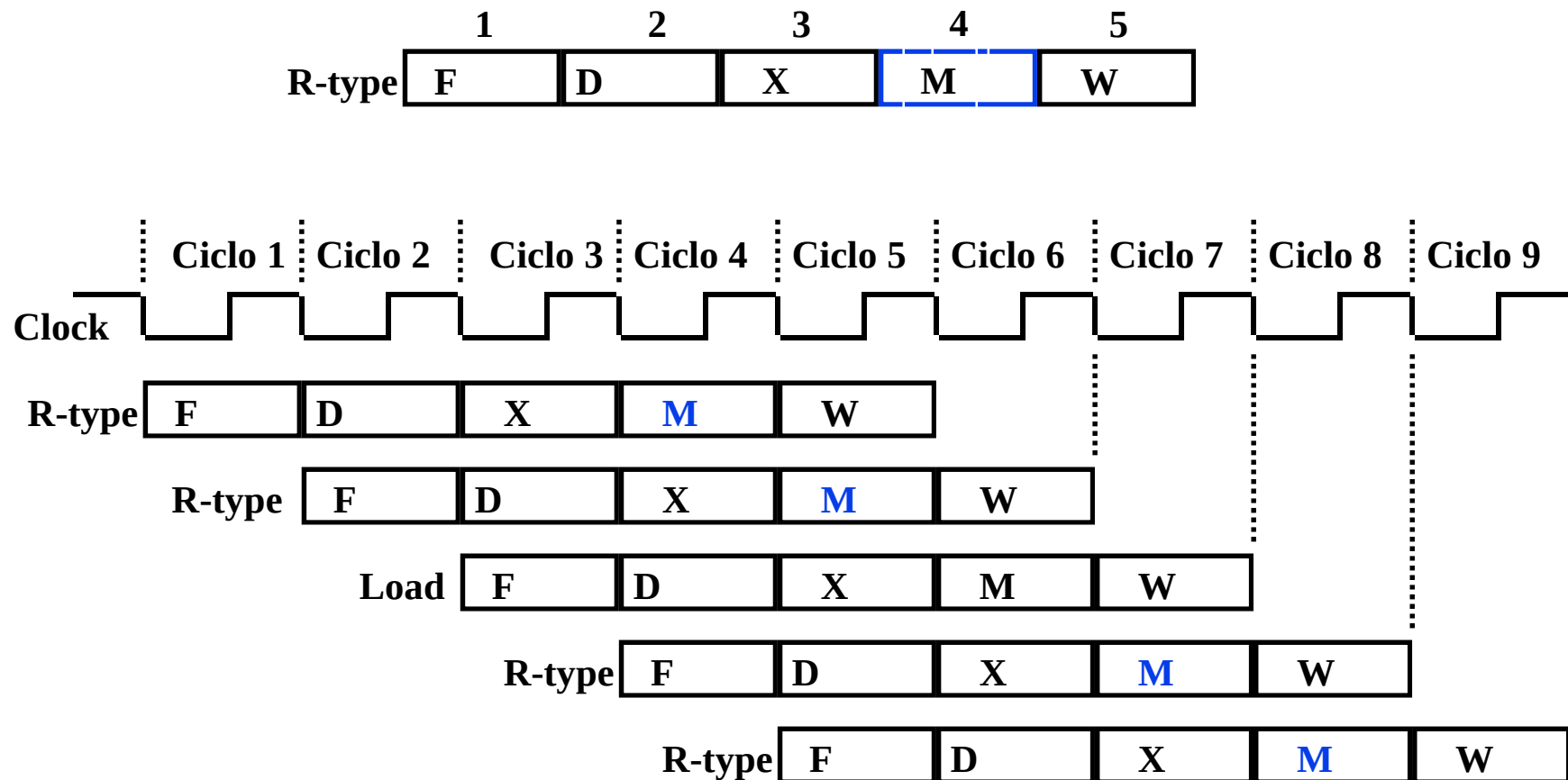
Soluzione 1: Inserire una "bolla" nella pipeline



- Inserire una "bolla" nella pipeline per impedire due scritture in stesso ciclo
 - La logica di controllo puo' essere complessa.
- Nessuna istruzione comincia nel ciclo 6!

Soluzione 2: Rimandare la scrittura del registro per un ciclo (istruzioni di tipo R)

- Ora le istruzioni di tipo R possono utilizzare la porta di scrittura dei registri nello stadio 5
- Lo stadio Mem non fa niente



Criticita' sui dati

- **criticita' sui dati**: tentativo di utilizzare un'unita' prima che sia pronta
- un'istruzione dipende dal risultato di un'istruzione precedente, anch'essa in pipeline

add r1, r2, r3

Data Hazard on r1

sub r4, r1, r3

and r6, r1, r7

or r8, r1, r9

xor r10, r1, r11

Classificazione di criticita' sui dati

- **RAW**

- Un'istruzione successiva cerca di **LEGGERE** un registro prima che l'istruzione precedente lo abbia scritto

- **WAR**

- Un'istruzione successiva cerca di **SCRIVERE** un registro prima che l'istruzione precedente lo abbia letto

- **WAW**

- Un'istruzione cerca di **SCRIVERE** un registro prima che l'istruzione precedente lo abbia scritto (quando questo avviene, il valore non sara' piu' associato all'istruzione piu' recente)

Criticita' sui dati - classificazione

- Read after Write (RAW)
 - L'istruzione 1 deve scrivere
prima che l'istruzione 2 legga
- Write after Read (WAR)
 - L'istruzione 1 legge
L'istruzione 2 scrive (riscrive)
 - L'istruzione 2 *non deve scrivere prima che la 1 legga*
- Write after Write (WAW)
 - Entrambe le istruzioni 1 e 2 scrivono
L'istruzione 2 deve scrivere dopo la 1

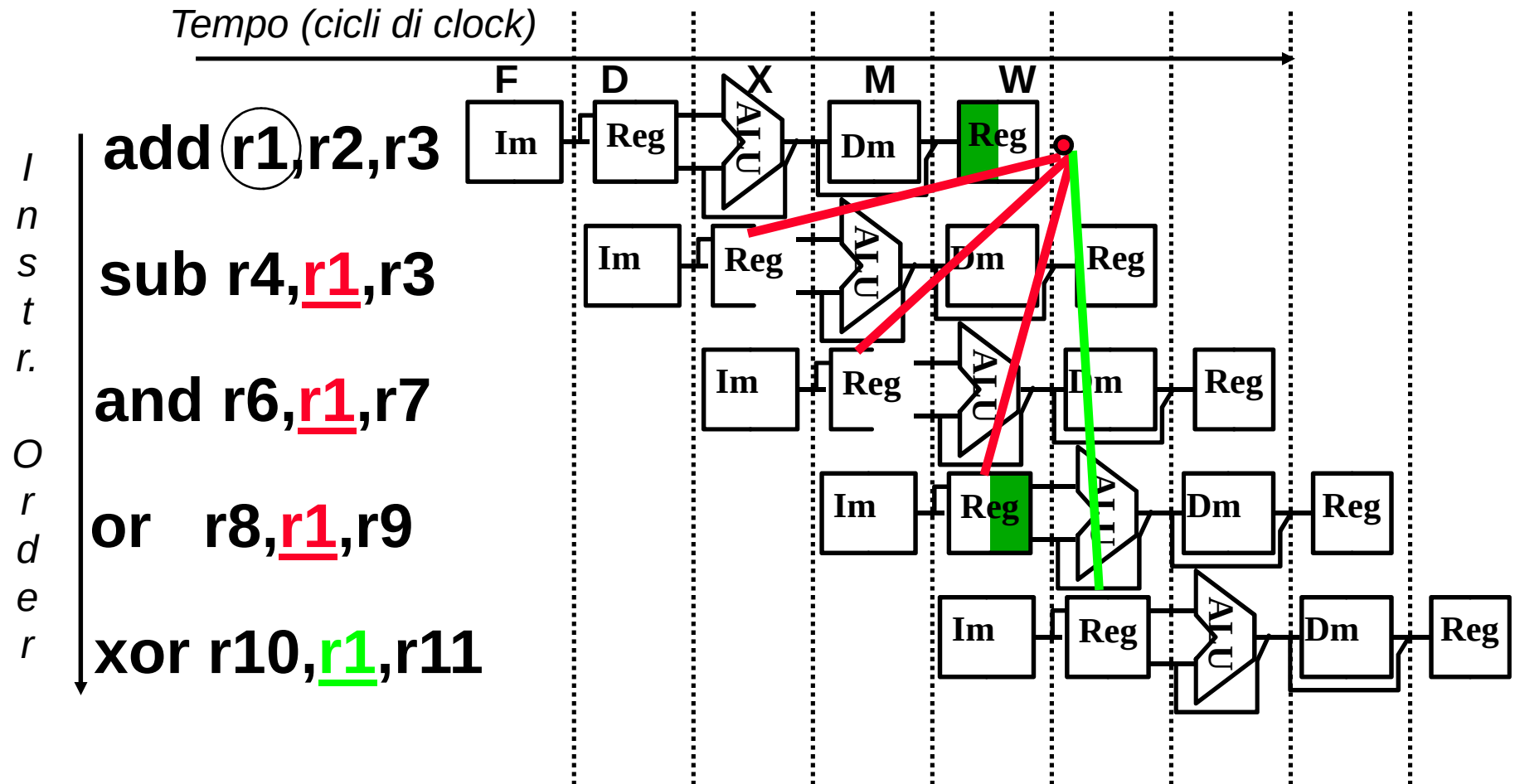
La pipeline deve considerare tutti e tre i casi!

Riassumendo

- Criticita' sui dati
 - RAW - i piu' comuni
 - WAW
 - WAR
- Il compilatore cerca le dipendenze funzionali
 - poi riordina il codice
- Hardware
 - Rete di controllo delle criticita'
 - Sorveglia le dipendenze
 - Rete di propagazione del valore
 - propaga i risultati dallo stadio X o Ma stadi precedenti

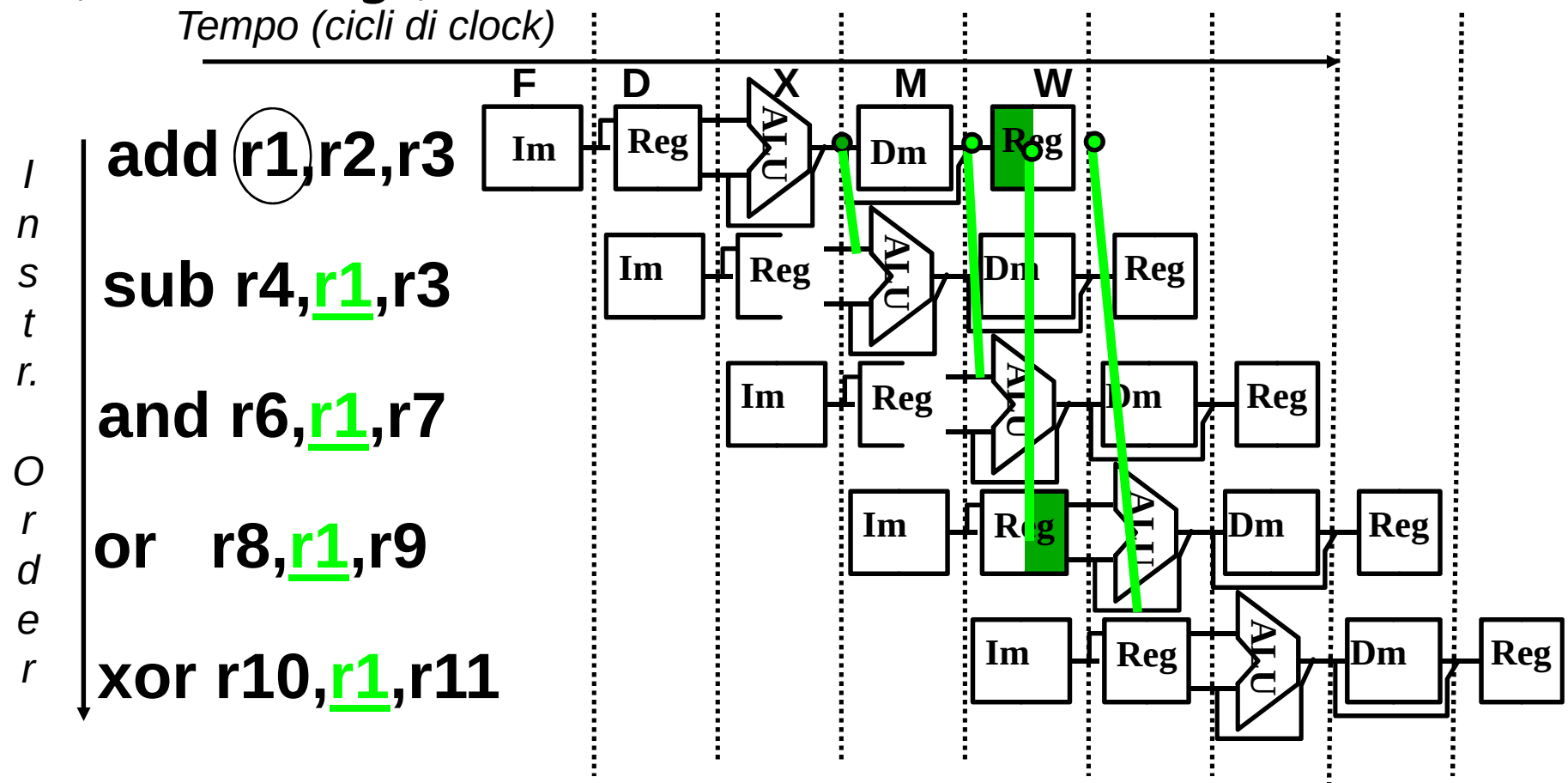
Data Hazard su r1:

- Le dipendenze "indietro nel tempo" sono criticita' di tipo RAW



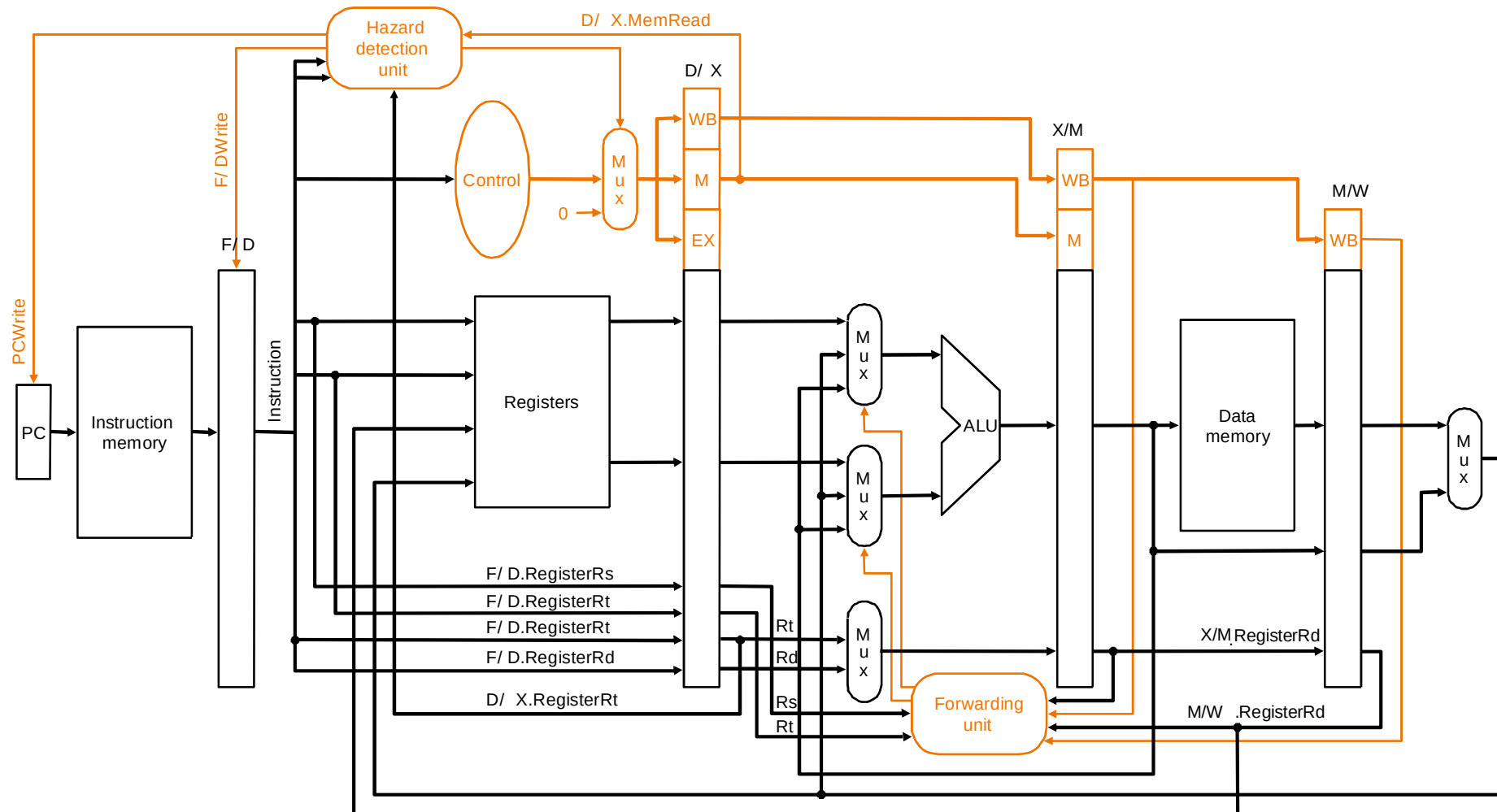
Soluzione per la criticita' sui dati

- “**Propagare**” il risultato da uno stadio all'altro (“forwarding”)



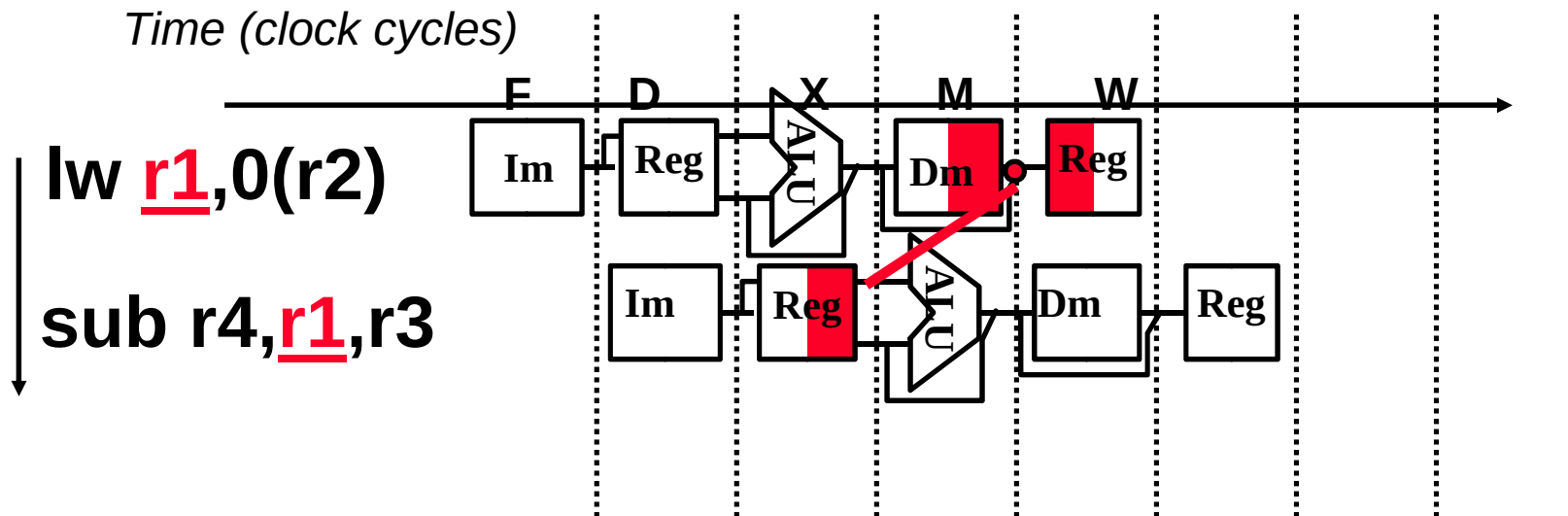
“or” OK se definisco read/write in un modo appropriato

Rete per la propagazione e rete di rilevamento criticita'



Cosa si fa con criticita' su load (detta "load/use")

- Le dipendenze "indietro nel tempo" sono criticita'



- Non si puo' risolvere con propagazione
- Le istruzioni che dipendono da una load devono essere ritardate/mandate in stallo ad hardware
- Questo si chiama **interlock**

Criticita' sul controllo: Branching (1/6)

- Supponiamo che la logica di decisione dei salti sia nello stadio di ALU
 - Allora due istruzioni in piu' dopo il salto saranno sempre prelevate
- Funzionalita' desiderata del salto
 - se il salto non viene effettuato, non perdere tempo e continuare con la normale esecuzione
 - se il salto viene effettuato, non eseguire istruzioni dopo il salto, ma saltare sulla etichetta desiderata

Criticita' sul controllo: Branching (2/6)

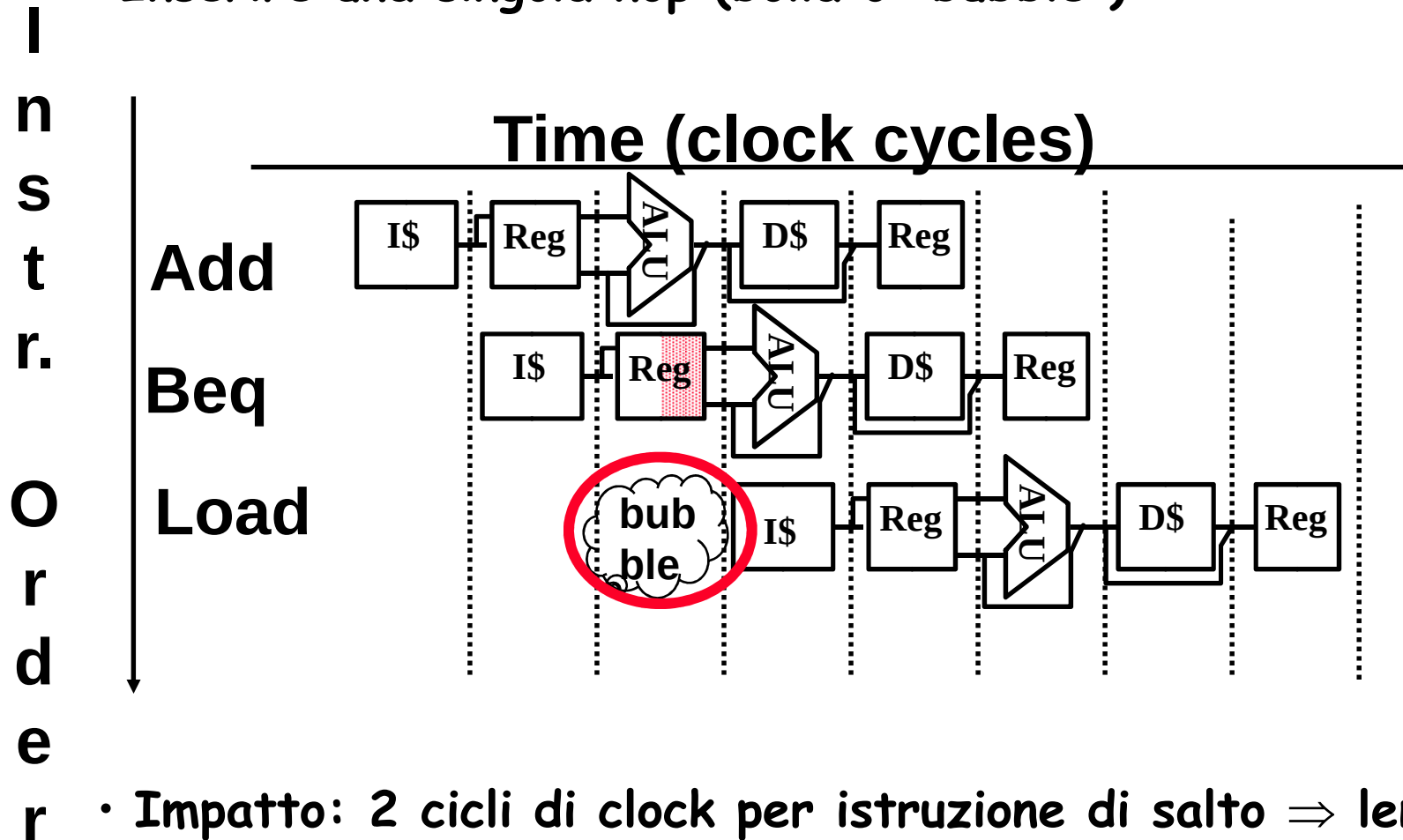
- Soluzione iniziale: sospendere l'esecuzione fino a quando non si prende la decisione
 - Inserire istruzioni "nop": non fanno nulla, solo consumano tempo
 - Svantaggio: ogni salto consuma 3 cicli (anziche' 1) (a patto che il comparatore sia stato messo nello stadio di ALU)

Criticita' sul controllo: Branching (3/6)

- Ottimizzazione #1:
 - Spostare il comparatore (e calcolo branch target) nello stadio 2
 - Non appena l'istruzione viene decodificata (opcode identifica un salto), prendere la decisione e impostare il valore di PC (se necessario)
 - Beneficio: poiche' il salto e' completo nello stadio 2, solo una
 - istruzione del ramo sbagliato viene prelevata, e solo una nop e' richiesta
 - Nota: in questo caso, i salti non fanno nulla negli stadi 3, 4 e 5

Criticita' sul controllo: Branching (4/6)

- Inserire una singola nop (bolla o "bubble")

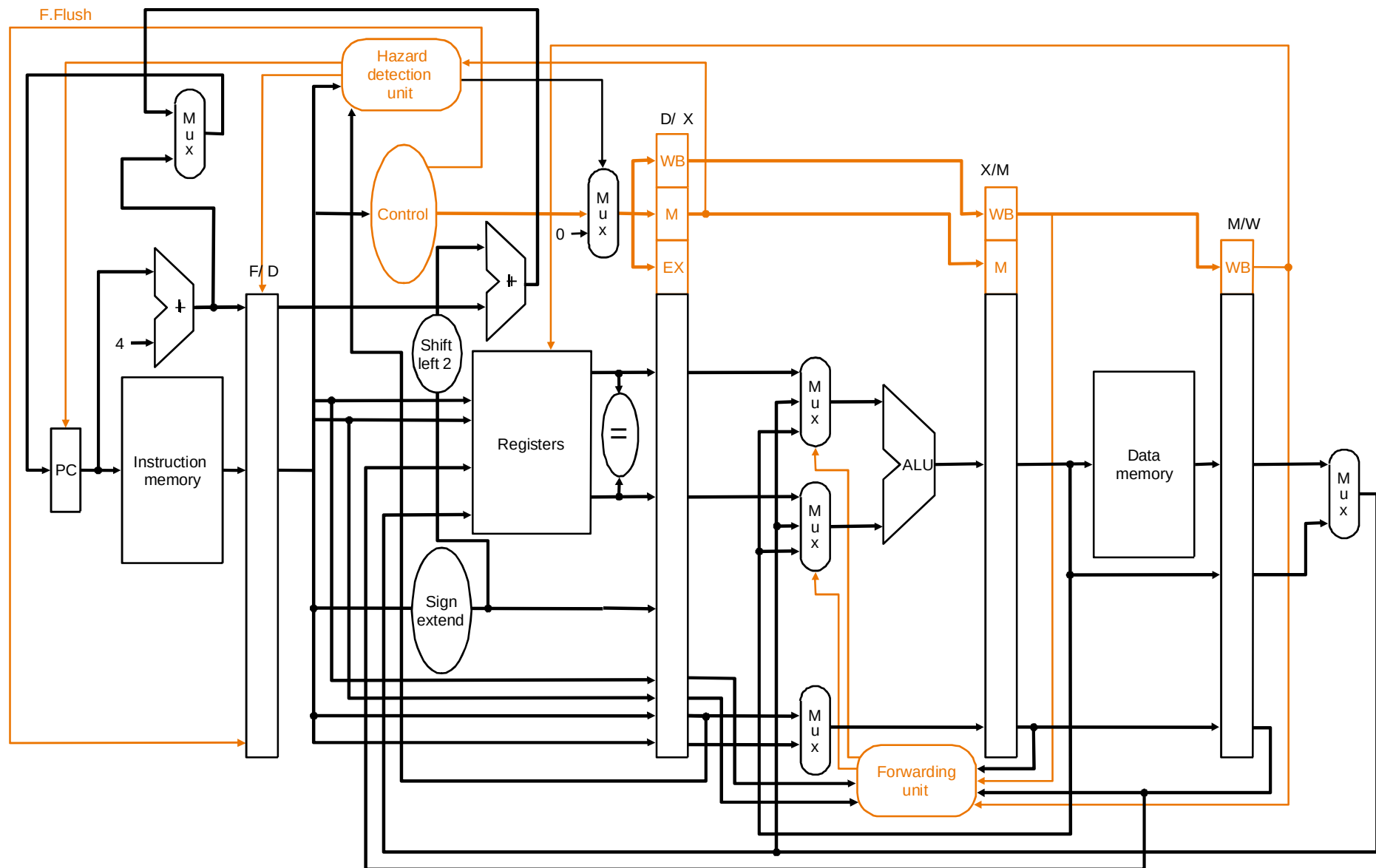


Propagazione e spostamento di decisione del salto

- La propagazione/bypassing attualmente influisce sullo stadio di esecuzione:
 - Invece di utilizzare il valore letto nei registri nello stadio di Decode, utilizzare il valore risultante dalla ALU o dalla memoria
- Lo spostamento della decisione di salto dallo stadio di esecuzione allo stadio di decodifica significa che la propagazione/bypassing deve arrivare fino allo stadio di decodifica
- Cioe', il seguente codice deve comunque funzionare:

```
addiu $s1, $s1, -4  
beq   $s1, $s2, Exit
```

Rete di propagazione con branch anticipato nello stadio D



Criticita' sul controllo: Branching (5/6)

- Ottimizzazione #2: Ridefinire i salti
 - Definizione vecchia: se il salto viene effettuato, nessuna istruzione dopo il salto viene eseguita
 - Definizione nuova: sia che si faccia il salto, sia che non si faccia, viene eseguita l'istruzione che segue immediatamente il salto (detto **branch-delay slot**)

Criticita' sul controllo: Branching (6/6)

- Osservazioni sul Branch-Delay Slot
 - Caso peggiore: si puo' sempre inserire una nop nel branch-delay slot
 - Caso migliore: si puo' trovare un'istruzione prima del salto che puo' essere messa nel branch-delay slot senza influenzare il flusso di programma
 - il riordinamento delle istruzioni e' un metodo comunemente utilizzato per velocizzare i programmi
 - il compilatore deve essere molto furbo per trovare le istruzioni appropriate
 - tipicamente i compilatori sono in grado di riempire circa il 50% di tali posizioni

Esempio: Nondelayed vs. Delayed Branch

Nondelayed Branch

or \$8, \$9, \$10

add \$1, \$2, \$3

sub \$4, \$5, \$6

beq \$1, \$4, Exit

xor \$10, \$1, \$11

Exit:

Delayed Branch

add \$1, \$2, \$3

sub \$4, \$5, \$6

beq \$1, \$4, Exit

or \$8, \$9, \$10

xor \$10, \$1, \$11

Exit:

Quanto dura l'esecuzione?

- Assumiamo: delayed branch, pipeline di 5 stadi, propagazione/bypassing, interlock sulle criticita' di load non risolte

```
Loop:      lw      $t0, 0($s1)
           addu    $t0, $t0, $s2
           sw      $t0, 0($s1)
           addiu   $s1, $s1, -4
           bne     $s1, $zero, Loop
           nop
```

- Quanti cicli di clock sono necessari per eseguire questo codice per un' iterazione di loop?

a) ≤ 5 b) 6 c) 7 d) 8 e) ≥ 9

(dopo 1000 iterazioni, quando la pipeline e' piena)

Quanto dura l'esecuzione?

- Assumere delayed branch, la pipeline di 5 stadi, propagazione/bypassing, interlock sulle criticita' di load non risolte
- Diamo un'occhiata a questo codice:

Loop:

1.	lw	\$t0, 0(\$s1)
2. (data hazard so stall)	addiu	\$t0, \$t0, \$s2
3.	sw	\$t0, 0(\$s1)
4.	addiu	\$s1, \$s1, -4
5.	bne	\$s1, \$zero, Loop
6.	nop	
7.		

(delayed branch so execute nop)

- Quanti cicli di clock sono necessari per eseguire questo codice per un' iterazione di loop?

a) ≤ 5 b) 6 c) 7 d) 8 e) ≥ 9

Riscrivere il ciclo per migliorare le prestazioni

- Riscrivere questo codice per ridurre al minimo i cicli di clock del loop :

Loop: lw \$t0, 0(\$s1)
 addu \$t0, \$t0, \$s2
 sw \$t0, 0(\$s1)
 addiu \$s1, \$s1, -4
 bne \$s1, \$zero, Loop
 nop

- Quanti cicli di clock sono necessari per eseguire il codice riscritto per un' iterazione di loop?

a) 4 b) 5 c) 6 d) 7

Riscrivere il ciclo per migliorare le prestazioni

- Riscrivere questo codice per ridurre il numero dei cicli di clock al minimo per un' iterazione del loop :

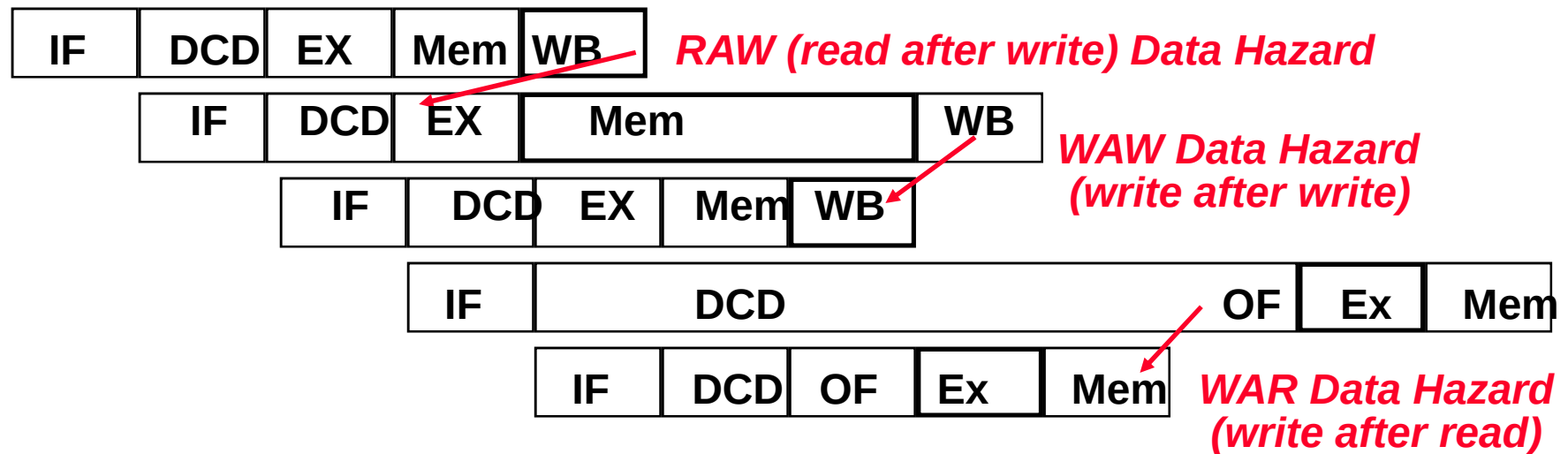
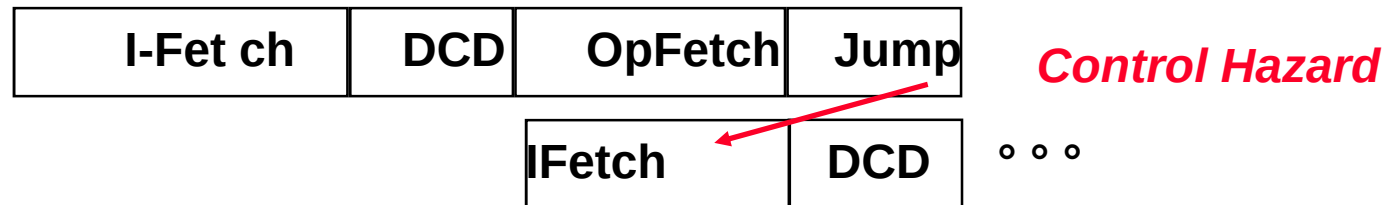
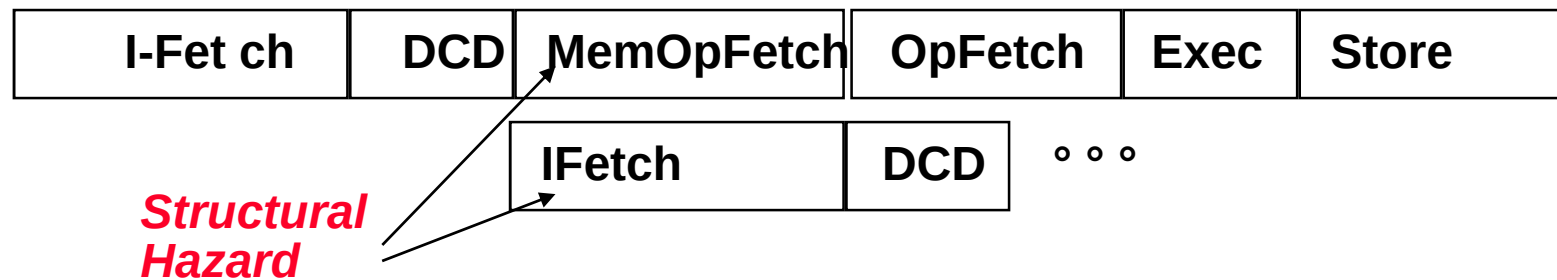
Loop:

1.	lw	<u>\$t0</u>	, 0(\$s1)	(no hazard since extra cycle)
2.	addiu	\$s1,	\$s1, -4	
3.	addu	\$t0,	<u>\$t0</u> , \$s2	
4.	bne	\$s1,	\$zero, Loop	
5.	sw	\$t0,	<u>+4</u> (\$s1)	(modified sw to put past addiu)

- Quanti cicli di clock sono necessari per eseguire il codice riscritto per un' iterazione di loop?

a) 4 **b) 5** c) 6 d) 7

Ricapitolando: Le criticita' nelle pipeline

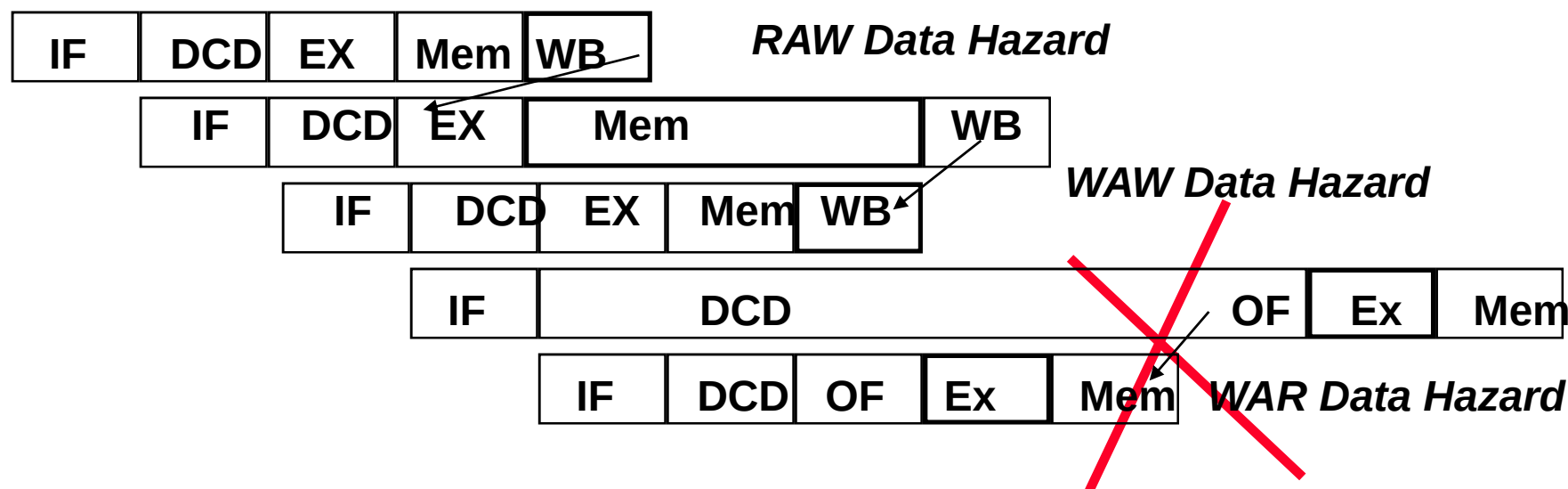


Punti principali del pipelining

- Le pipeline fanno passare informazioni di controllo al loro interno nello stesso modo in cui fanno passare i dati
- Propagazione e stalli sono gestiti dal controllo locale
- Le criticita' limitano le prestazioni
 - Strutturali: richiedono piu' risorse HW
 - Dati: richiedono propagazione, schedulazione dal compilatore
 - Controllo: valutazione preliminare, delayed branch, predizione
- Aumentando la lunghezza della pipe aumenta l'impatto delle criticita'; pipelining migliora il throughput delle istruzioni, e non la loro latenza
- Interruzioni, Instruction Set, FP rendono il pipelining piu' difficile
- I compilatori riducono il costo delle criticita' sui dati e sul controllo
 - Load delay slots
 - Branch delay slots
 - Branch prediction

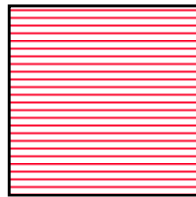
Ricapitolando: Criticita' sui dati

- Evitare alcune durante la progettazione
 - eliminare WAR prelevando presto gli operandi (DCD) in pipe
 - eliminare WAW eseguendo tutti i WB in ordine (nell'ultimo stadio)
- Rivelare e risolvere quelli rimasti
 - Mandare in stallo o propagare (se possibile)



Rivelamento delle criticita'

- Supponiamo che l'istruzione j stia per essere elaborata e l'istruzione precedente i si trovi ancora nella pipeline.
- Una criticita' RAW esiste su registro p se $p \in Wregs(i) \cap Rregs(j)$
 - Tenere traccia delle scritture in attesa (per le istruzioni nella pipe) e confrontare con i registri di operandi della istruzione corrente.
 - Quando l'istruzione viene elaborata, prenotare il suo registro del risultato.
 - Quando l'istruzione e' completa, rimuovere la sua prenotazione di scrittura.



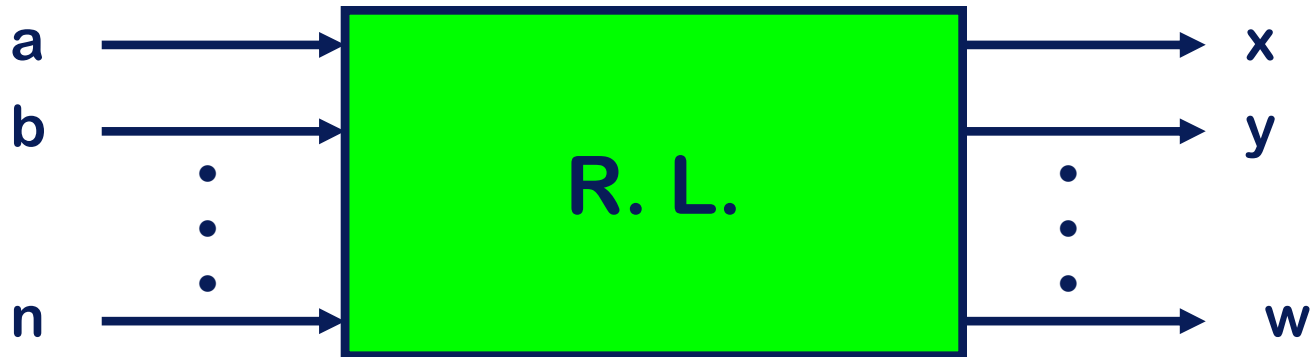
- Una criticita' WAW esiste su registro p se $p \in Wregs(i) \cap Wregs(j)$
- Una criticita' WAR esiste su registro p se $p \in Rregs(i) \cap Wregs(j)$

LEZIONE N° 93

Reti Logiche Sequenziali
(parte prima)

Reti Logiche

- Sistema elettronico che ha in ingresso segnali digitali e fornisce in uscita segnali digitali secondo leggi descrivibili con l'algebra Booleana



- R.L. è unidirezionale

Tipi di reti

- Reti COMBINATORIE

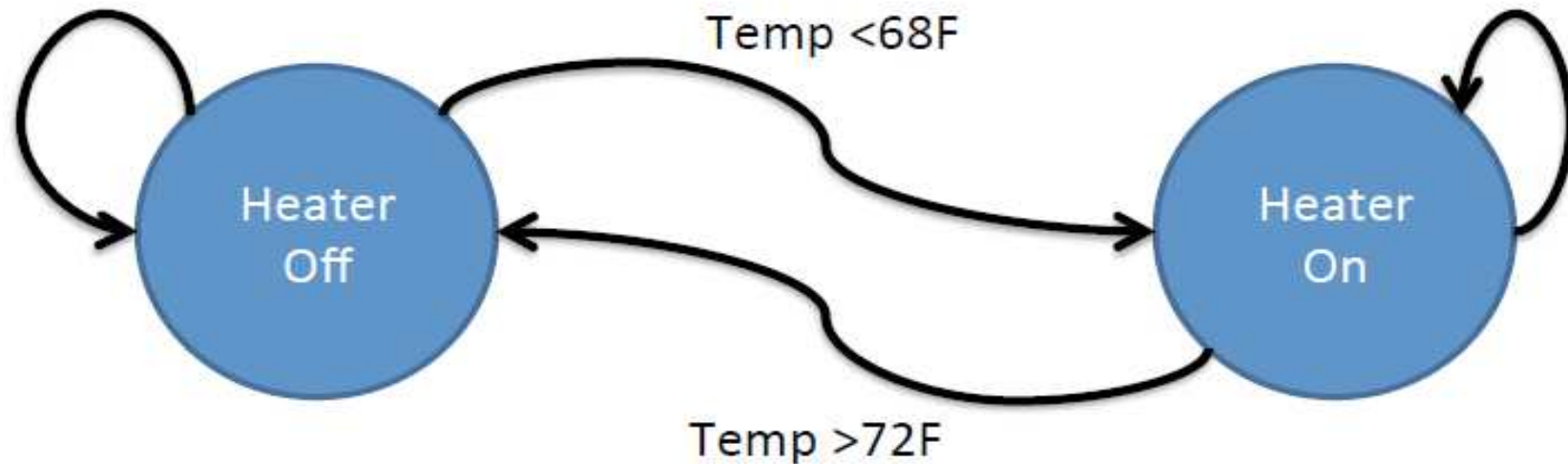
- In qualunque istante le uscite sono funzione del valore che gli ingressi hanno in quell'istante
- Il comportamento (uscite in funzione degli ingressi) è descritto da una tabella
- Non contengono anelli di richiusura

- Reti SEQUENZIALI

- In un determinato istante le uscite sono funzione del valore che gli ingressi hanno in quell'istante e i valori che hanno assunto precedentemente
- La descrizione è più complessa
- Stati Interni →
- Reti dotate di *MEMORIA*

Macchine a Stati Finiti

- Un gruppo di stati e di transizioni



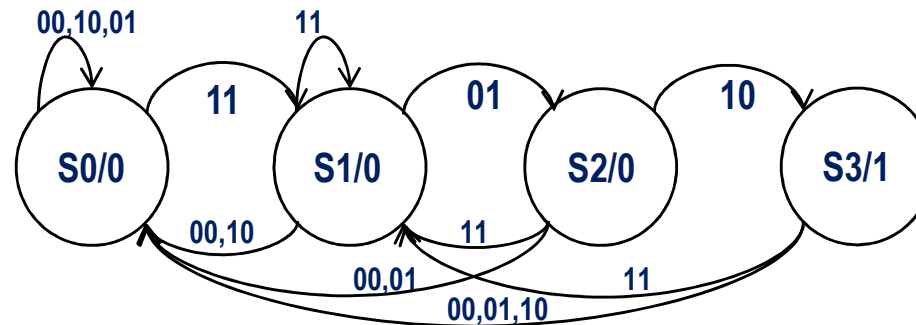
- Ci si sposta da uno stato all'altro seguendo una linea di transazione SE la condizione dalla transizione e' soddisfatta

Macchine a Stati Finiti (2)

- Nel campo dell'architettura dei calcolatori per le FSM:
 - Le transizioni tipicamente avvengono su fronti del clock
 - Sono complete
 - Tutti gli stati definiscono transizioni per tutti gli input
 - Sono deterministiche
- Possono essere trasformate in reti logiche dette Reti Sequenziali
 - Lo vedremo prossimamente.

Teoria delle reti sequenziali

- Negli anni 40-50 si e' consolidata una teoria degli Automi (Chomsky) e in particolare delle cosiddette Macchine a Stati Finiti (FSM)
- Nel 1955 Mealy [2] e nel 1956 Moore [3] propongono dei modelli per implementare una FSM
 - L'idea era che una FSM potesse essere realizzata ricorrendo solo ad elementi di memoria e logica combinatoria opportunamente collegata
 - Pertanto e' nata una teoria di come realizzare delle reti sequenziali che implementino FSM il cui comportamento e' descritto o informalmente (tramite il linguaggio naturale) o formalmente (tramite un diagramma a stati)
 - Esempio: riconoscere una sequenza binaria 11, 01, 10

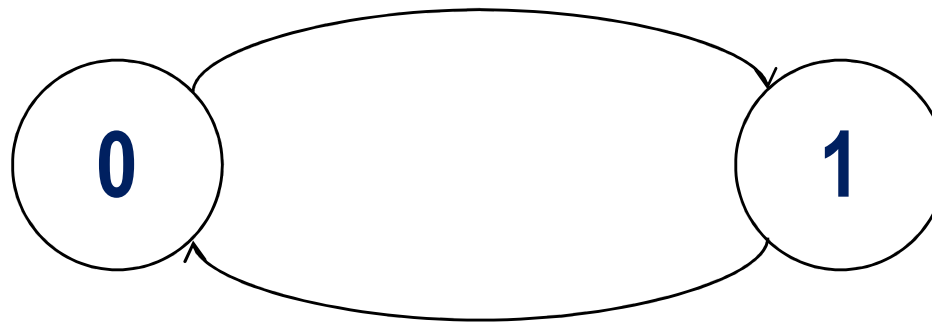


[2] Mealy, George H. (September 1955). "A Method for Synthesizing Sequential Circuits". *Bell System Technical Journal* **34**: 1045–1079

[3] Moore, Edward F (1956). "Gedanken-experiments on Sequential Machines". *Automata Studies, Annals of Mathematical Studies* (Princeton Univ. Press) (34): 129-153.

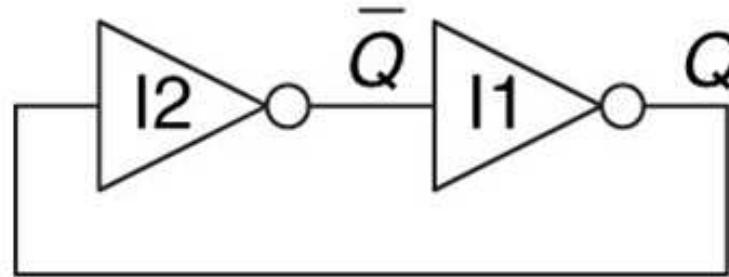
L'elemento di memoria

- La FSM piu' semplice che si puo' immaginare (e che varia il proprio stato) e' una rete a due soli stati
 - Tale rete non rappresenta altro che un bit di informazione e quindi permette di implementare **un elemento di memoria**



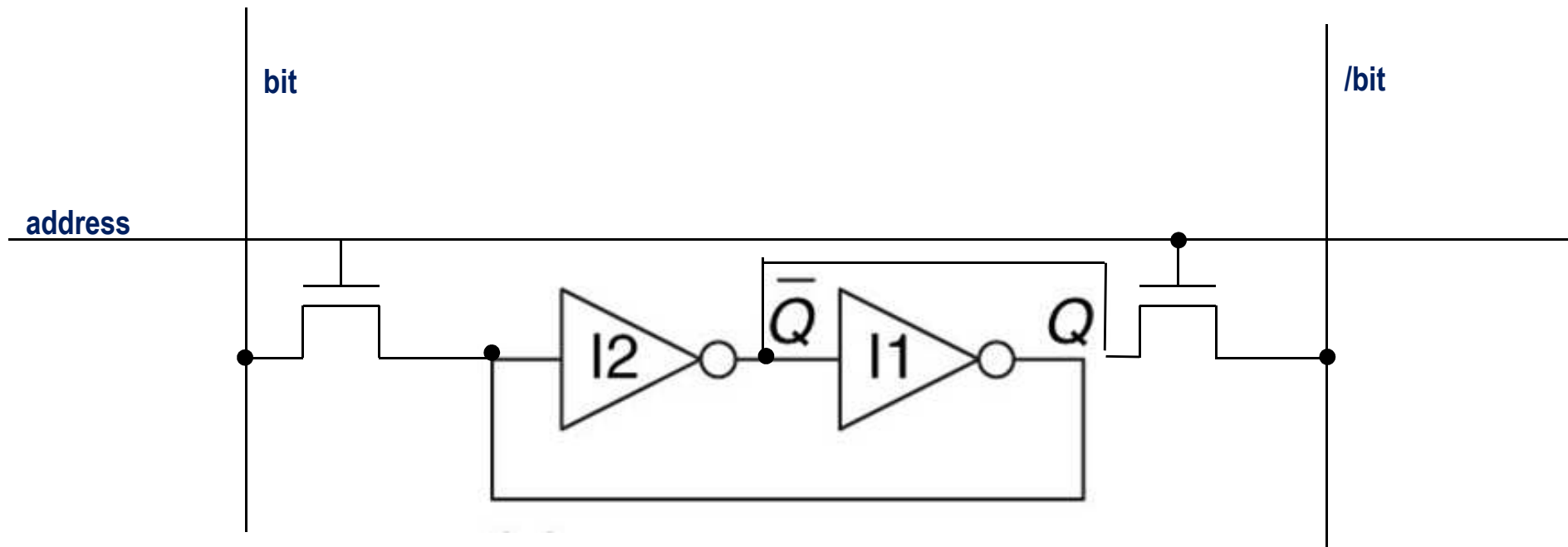
Realizzazione di elementi di memoria con reti combinatorie

- Abbiamo visto che l'elemento piu' semplice che riusciamo a realizzare facilmente su silicio e' **un** inverter...
- Il modo piu' semplice per realizzare un elemento di memoria usando inverter e' di utilizzare **due** inverter... richiusi ad anello



- Questo circuito digitale presenta DUE STATI stabili (si dice peranto che e' **bistabile**):
 - i) $Q=0$ (e di conseguenza $\bar{Q}=1$)
 - ii) $Q=1$ (e di conseguenza $\bar{Q}=0$)
- Pero' presenta solo due uscite e **nessun ingresso**
- Se ben dimensionato fisicamente funziona, altrimenti potrebbe anche oscillare (comportamento **metastabile**)

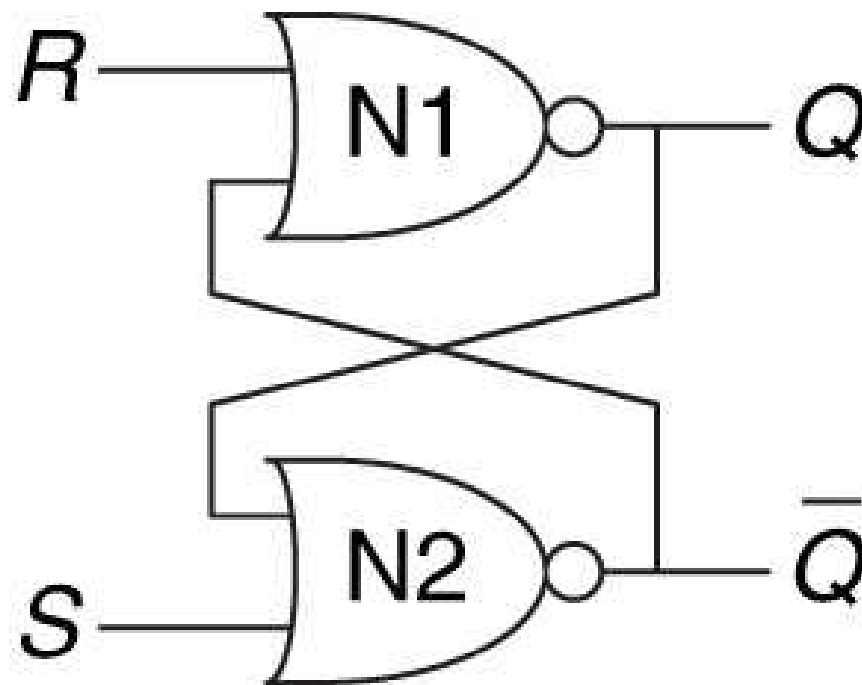
Perfezionando il bistabile usando la tecnologia CMOS



- In questo modo si può attivare l'elemento tramite il filo address e
 - imporre un 1 logico (bit=1, /bit=0) oppure uno 0 logico (bit=0, /bit=1)
 - Oppure leggere il valore Q senza distruggere il contenuto
- Questo tipo di cella di memoria viene usata per realizzare le SRAM (Static RAM) → l'informazione rimane memorizzata fintanto che la cella è alimentata
 - In tecnologia CMOS si realizza con 6 transistor (2 per ogni inverter e due per i due transistor di passo)

Latch SR

- L'elemento di memoria realizzato con due inverter e' in grado di memorizzare l'informazione ma non ha una forma "autocontenuta" per memorizzare il dato.
- Pertanto sono stati sviluppati ulteriori elementi bistabili denominati
 - Latch: elementi che variano lo stato su un livello di un segnale di clock
 - Flip-flop: elementi che variano lo stato su un fronte di un segnale di clock

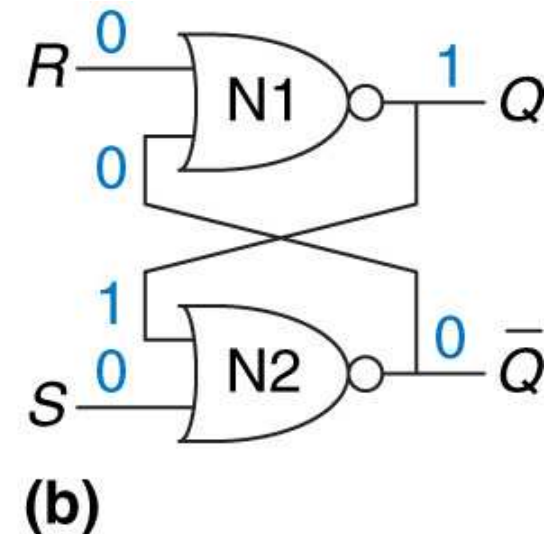
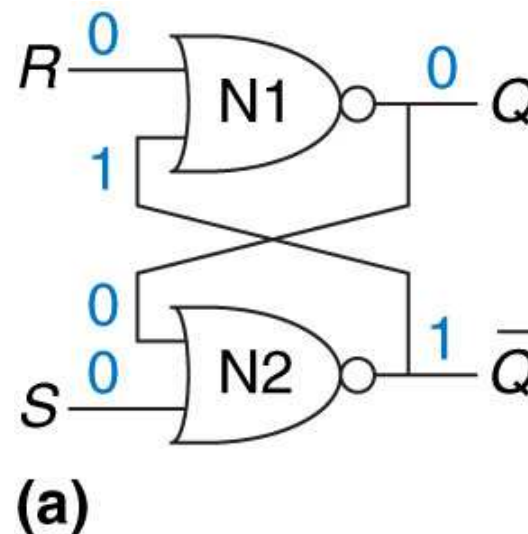


Elemento di memorizzazione in logica sequenziale asincrona

- Il latch SR
 - l'uscita dipende dai segnali presenti sugli ingressi e dai segnali precedentemente presentati in ingresso

S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	---	---

Tabella di Verità

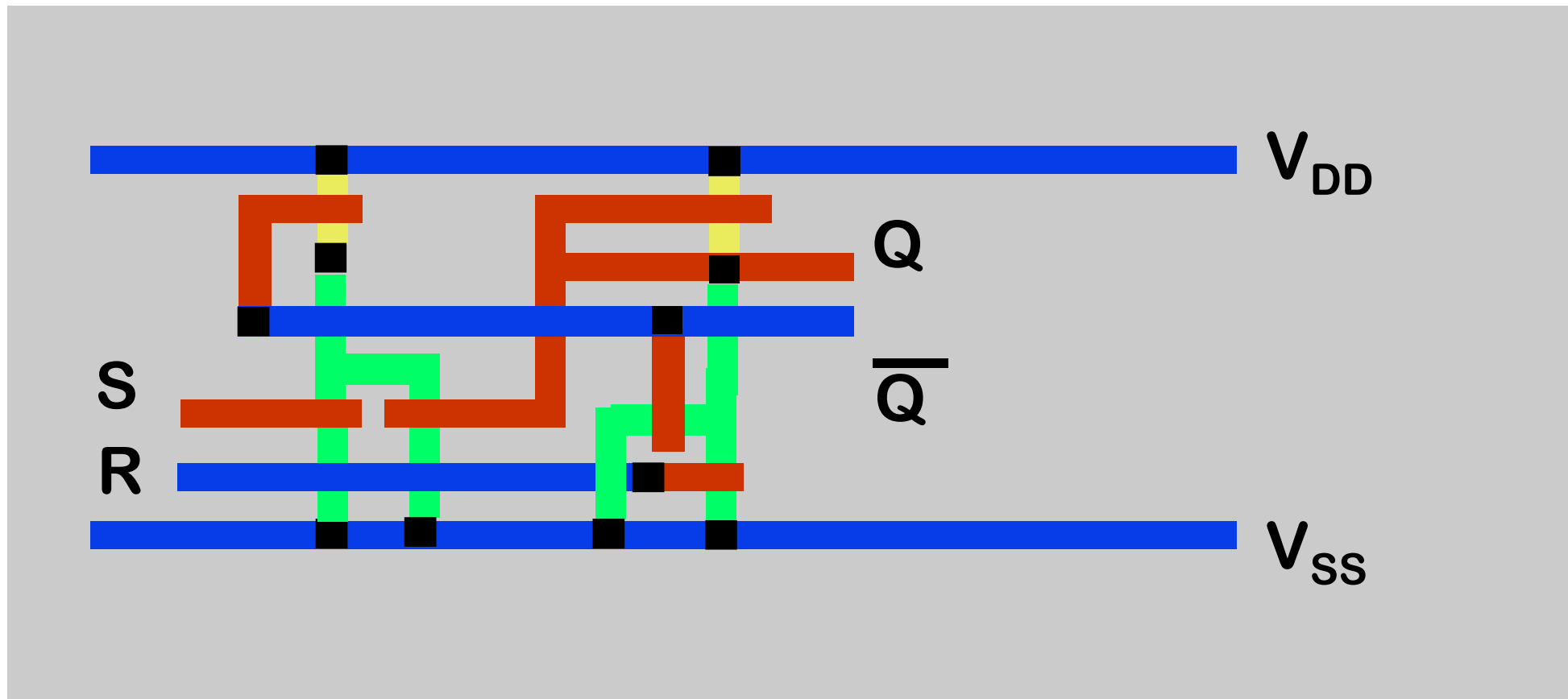
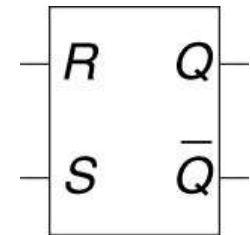


Schema logico

Lay-Out simbolico in tecnologia CMOS (latch SR)

- Transistori necessari = 6

Simbolo →

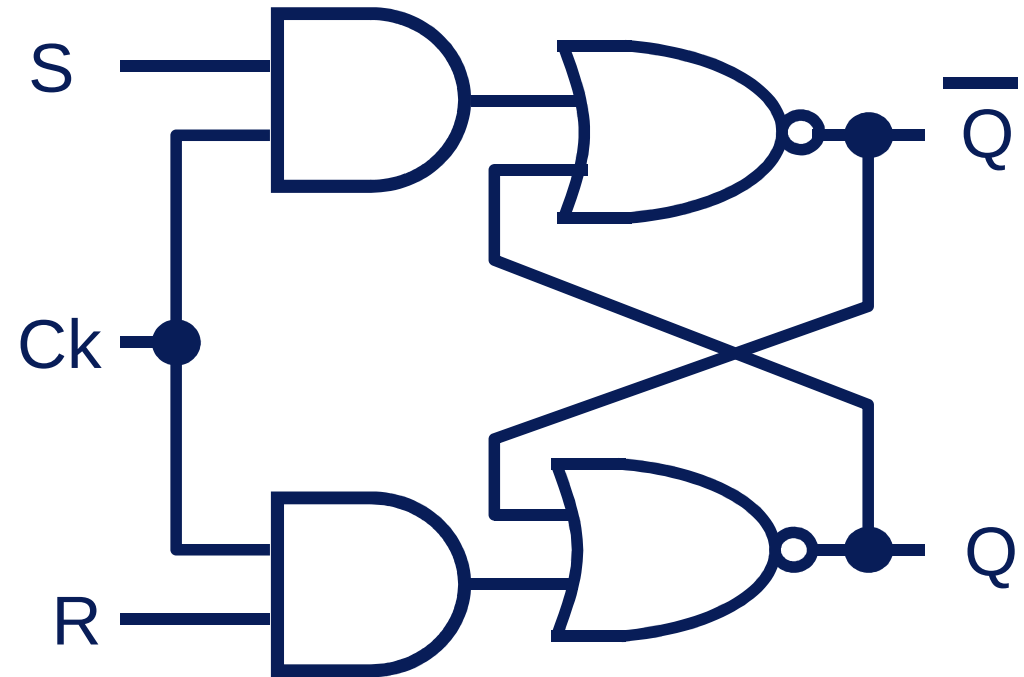


S-R clockato

Sente S e R solo quando il clock vale 1

Ck	S	R	Q	$\overline{Q}$
1	0	0	Q	$\overline{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	---	---
0	X	X	Q	$\overline{Q}$

Tabella di Verità

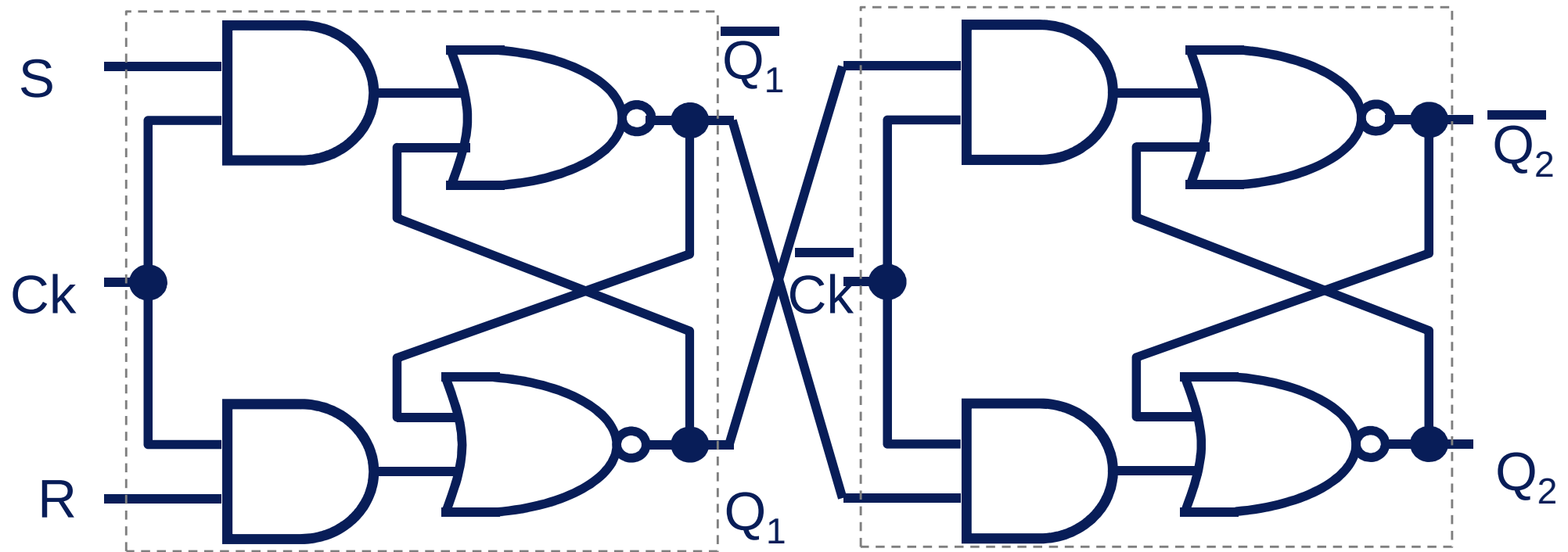


Schema logico

Layout simbolico: (12 transistor)

S-R edge triggered

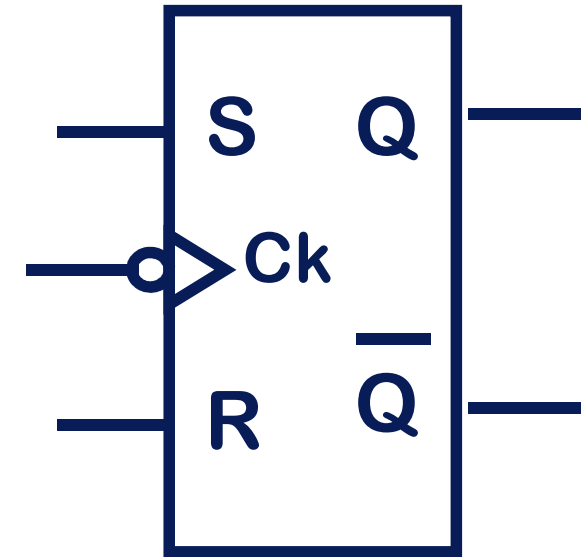
In questo caso vogliamo sensibilita' solo sul fronte in discesa



Layout simbolico: (24 transistor)

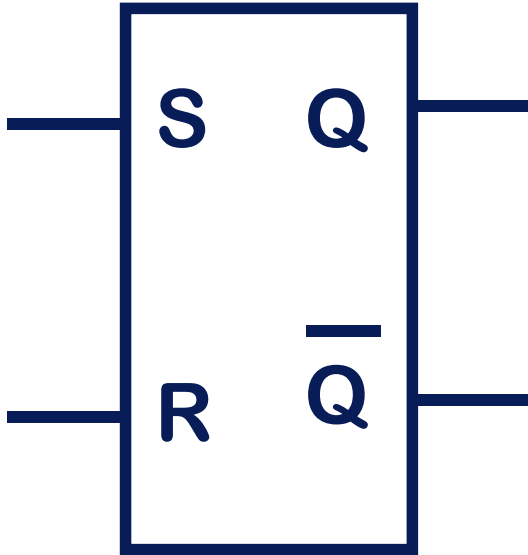
Tabella verita' S-R edge-triggered

Ck	S	R	Q	$\overline{Q}$
	0	0	Q	$\overline{Q}$
	0	1	0	1
	1	0	1	0
	1	1	---	---
0	X	X	Q	$\overline{Q}$
1	X	X	Q	$\overline{Q}$
	X	X	Q	$\overline{Q}$

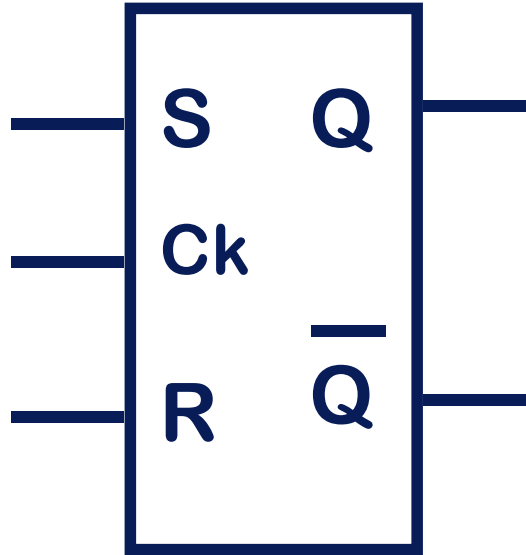


Simboli

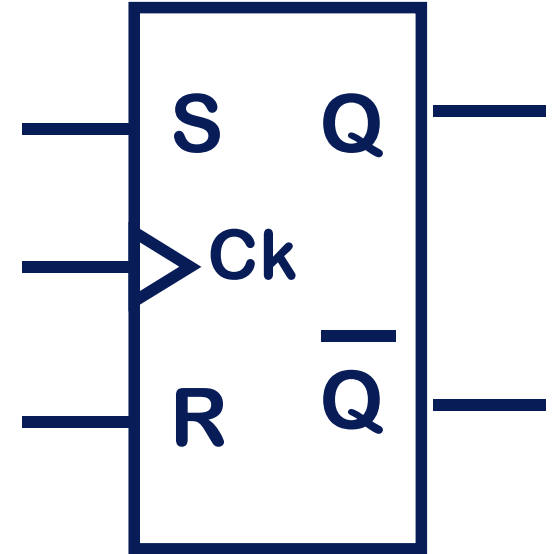
S-R



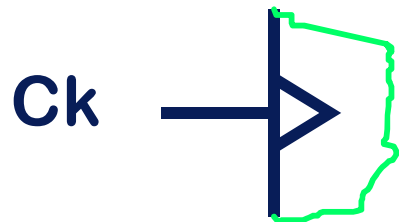
S-R clockato



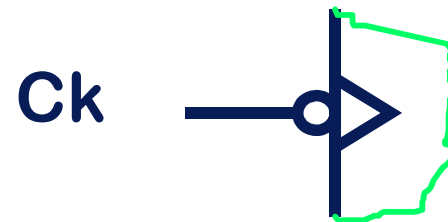
S-R edge triggered



Sensibili sul:



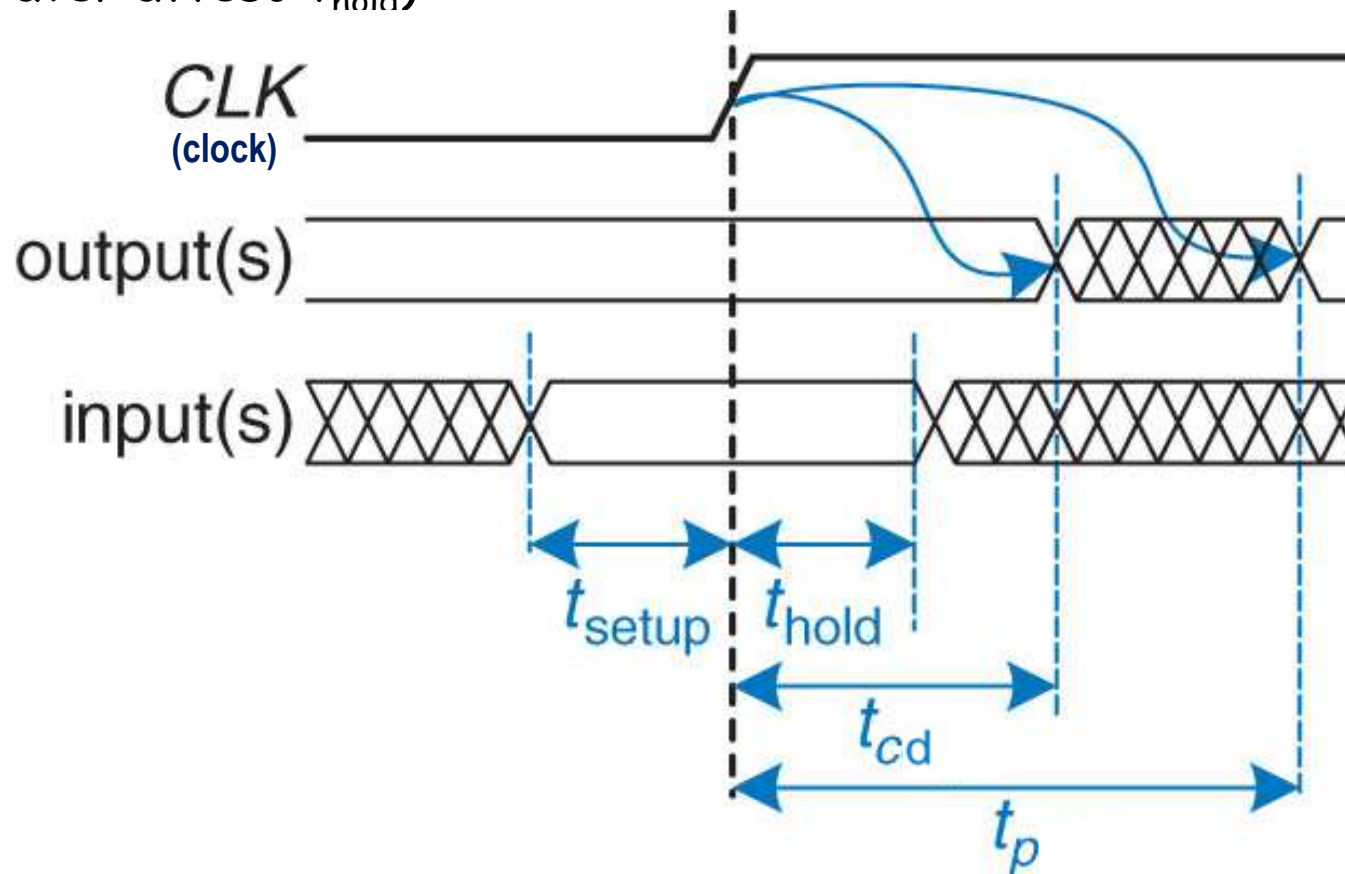
Fronte in salita



Fronte in discesa

tsetup e thold

- I segnali di ingresso SR devono essere stabili durante un fronte del clock (vanno preparati un po' prima $\rightarrow t_{\text{setup}}$)
- Dopo il fronte si possono tranquillamente variare i segnali (dopo aver atteso t_{hold})



- Il **tempo di propagazione** t_p e' il tempo necessario affinché l'uscita sia stabile dal momento in cui ho variato l'ingresso
- Si definisce invece **tempo di contaminazione** t_{cd} il tempo che intercorre da quando ho variato l'ingresso a quando inizia a variare l'uscita

D-Latch

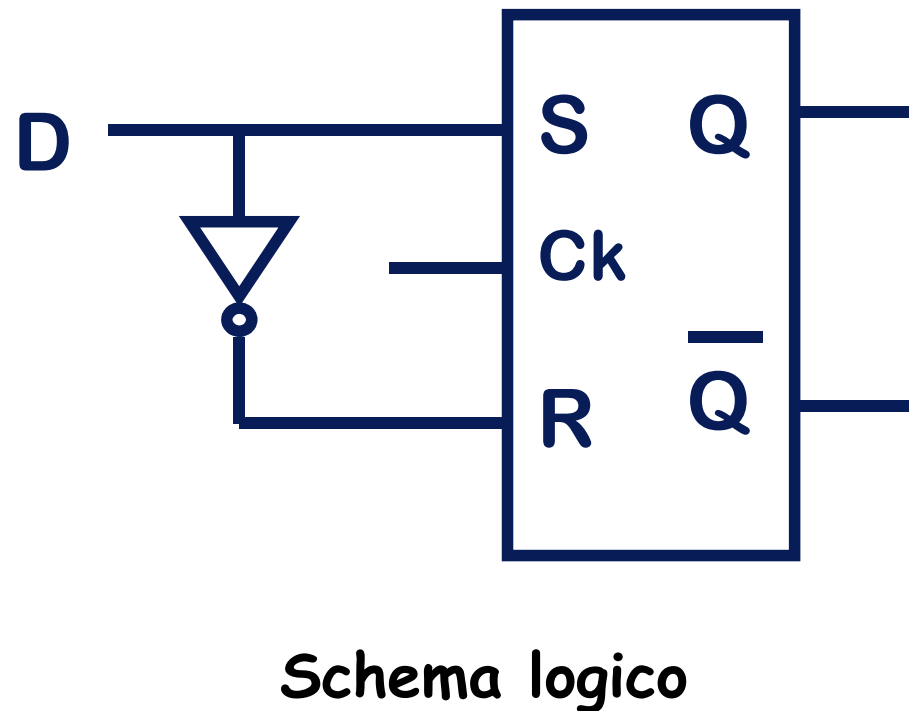
- IL latch SR e' problematico perche' si comporta in modo imprevedibile quando $SR=11$
- SR definiscono anche cosa e quando le uscite vengono cambiate
 - Il progetto dei circuiti digitali e' piu' semplice se si definiscono separatamente cosa cambiare e quando cambiarlo
 - Questo e' esattamente cosa fa il D-latch
- D-latch
 - Il segnale D controlla il dato ovvero cosa deve essere l'uscita
 - Il segnale CLK controlla quando lo stato deve essere cambiato

D-Latch o D-Transparent

- Due ingressi:
 - il dato da memorizzare (D)
 - il segnale di clock (Ck) che indica quando leggere o memorizzare (D)

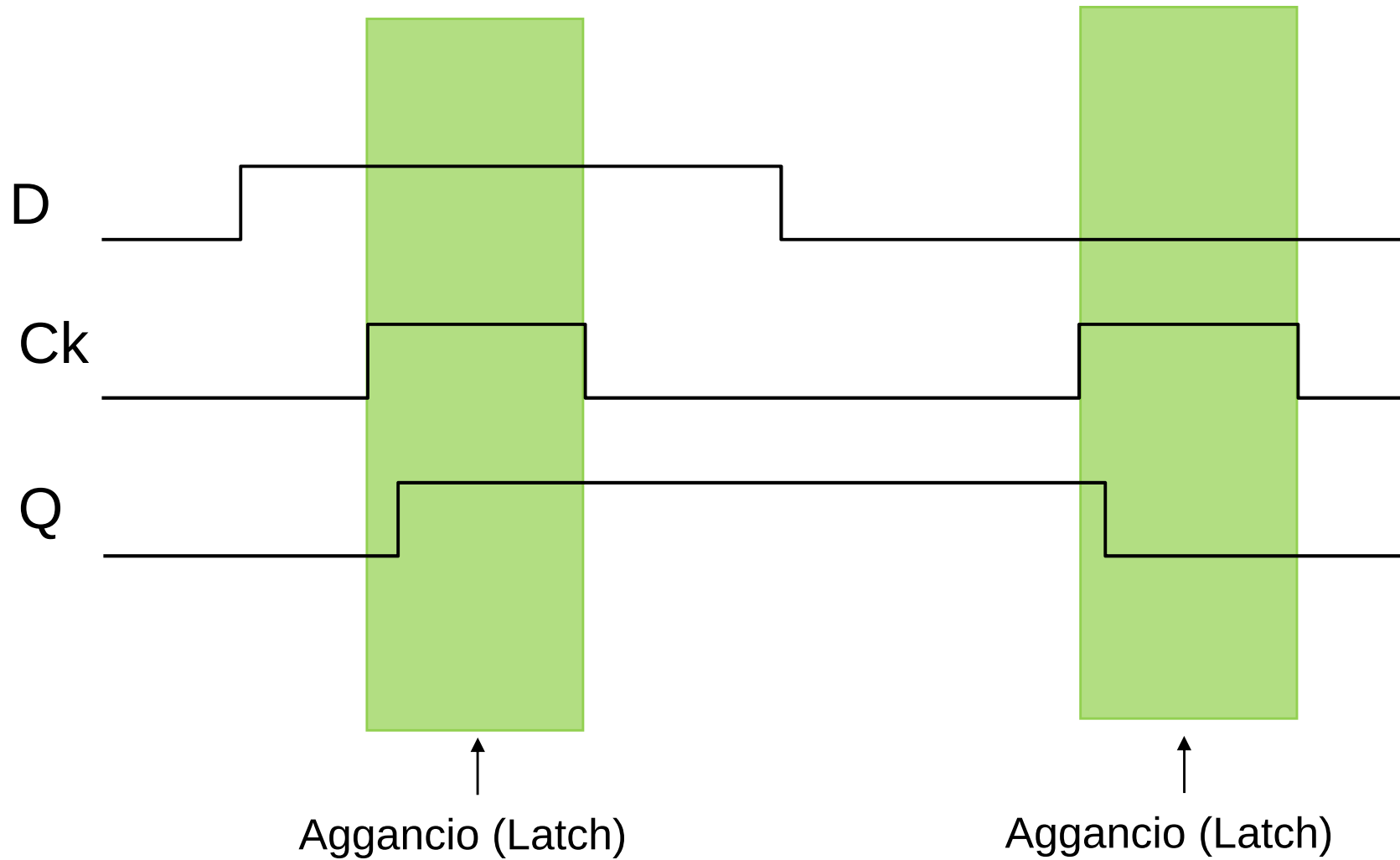
D	Ck	Q
X	0	Q
0	1	0
1	1	1

Tabella di Verità



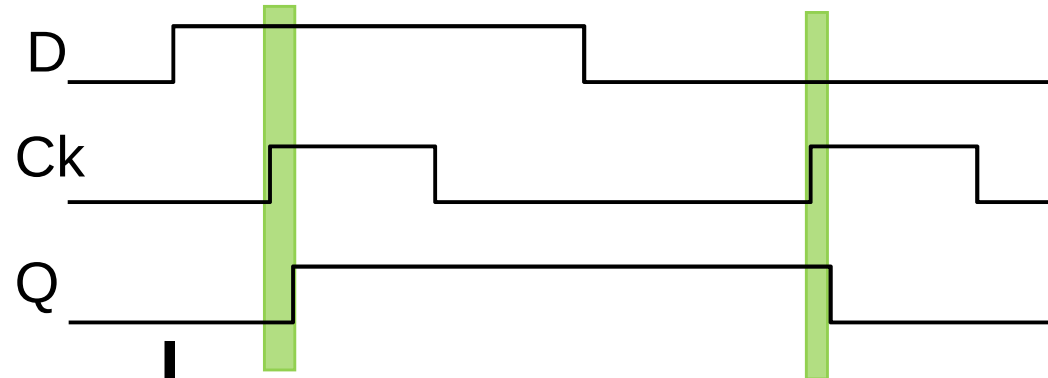
Layout simbolico: (14 transistor)

D-Latch o D-Transparent (esempio)



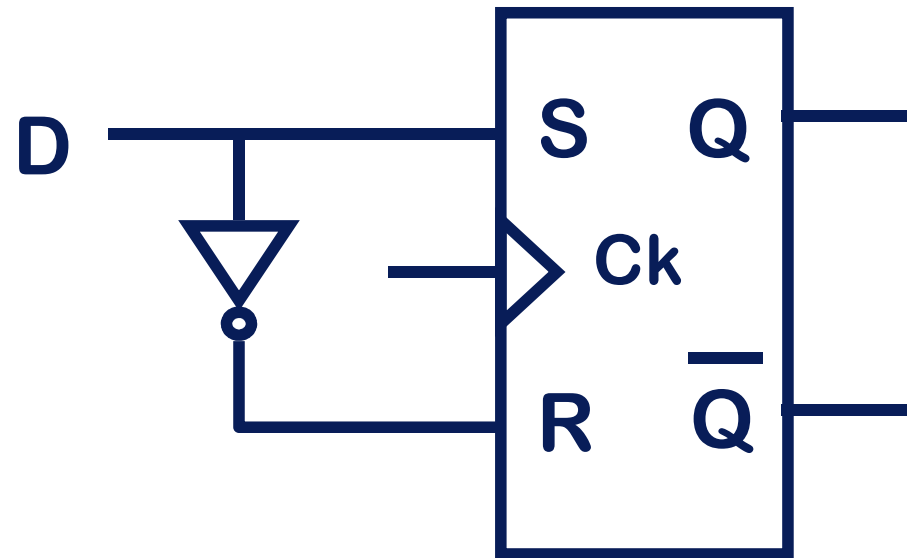
D-edge triggered

- L'uscita cambia solo sul fronte in salita del clock



D	Ck	Q
X	1	Q
X	0	Q
X		Q
0		0
1		1

Tabella di Verità



Schema logico

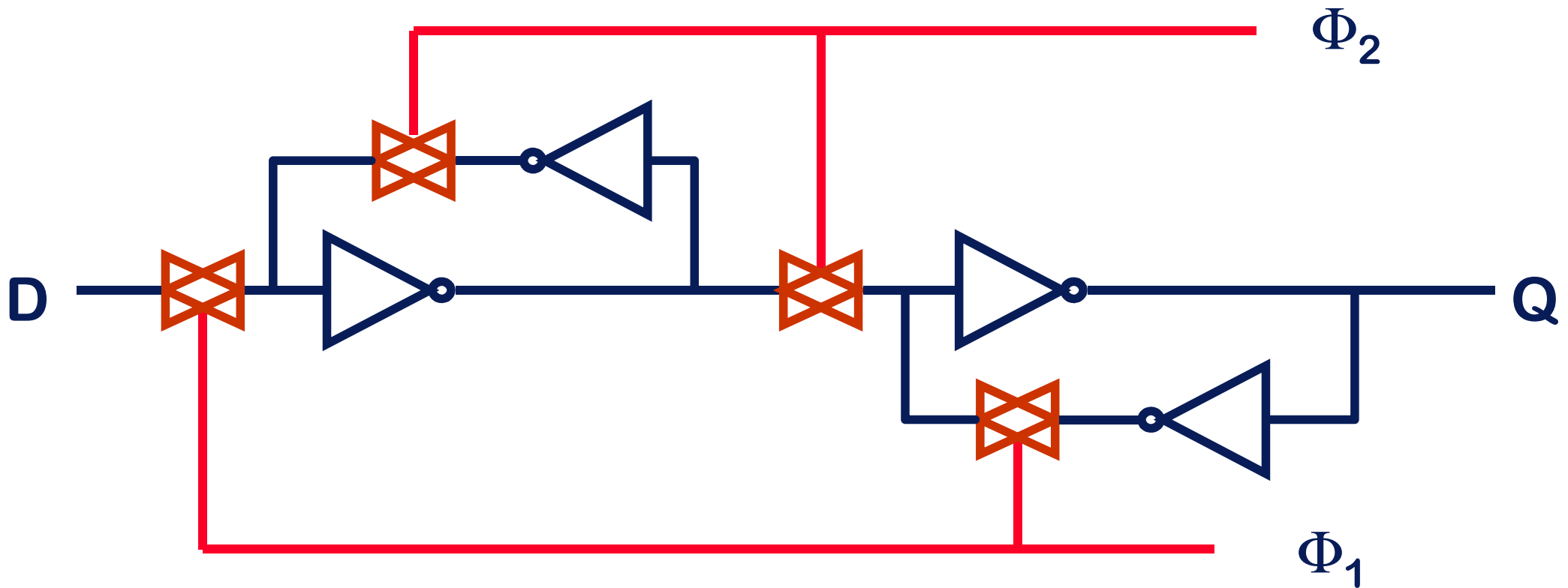
Layout simbolico: (26 transistor)

Latch vs. Flip-flops

- Latch:
l'uscita cambia non appena cambia l'ingresso
E
il clock ha valore alto
- Flip-flop:
l'uscita cambia solo su un fronte del clock
(metodologia edge-triggered)
- Elementi in comune:
 - l'uscita e' pari al valore memorizzato nell'elemento
(non e' necessario fare azioni particolari per leggere il valore)
 - il cambiamento dell'uscita avviene rispetto a un segnale di clock
- Nota:
nella logica sincrona si definisce l'istante
in cui un valore e' pronto per essere letto sugli ingressi del flip-flop
 - non si vuole leggere un dato D mentre la rete che genera D sta ancora "calcolandone" il valore

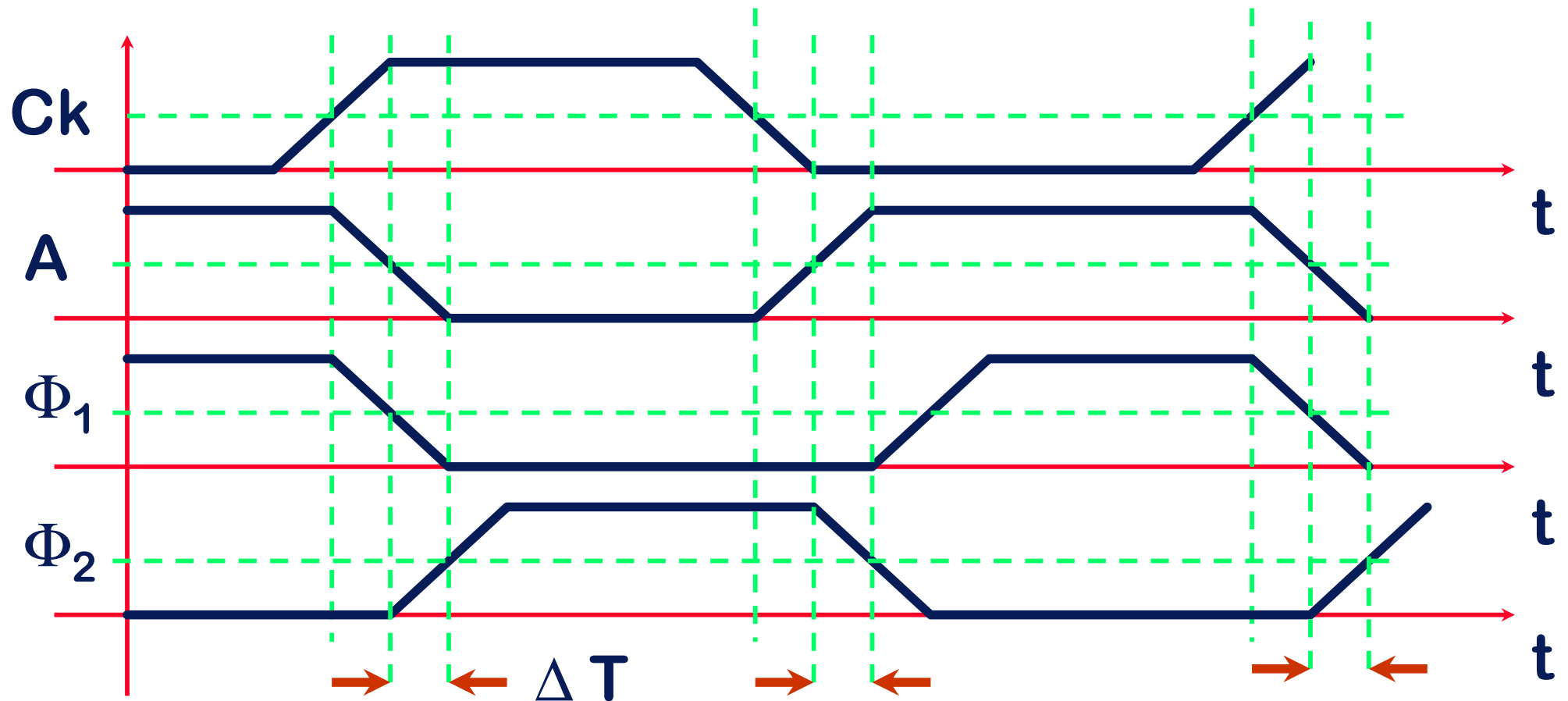
D Edge Triggered

Disponendo di un “clock a due fasi” si puo’ utilizzare una forma piu’ compatta del D-edge-triggerd



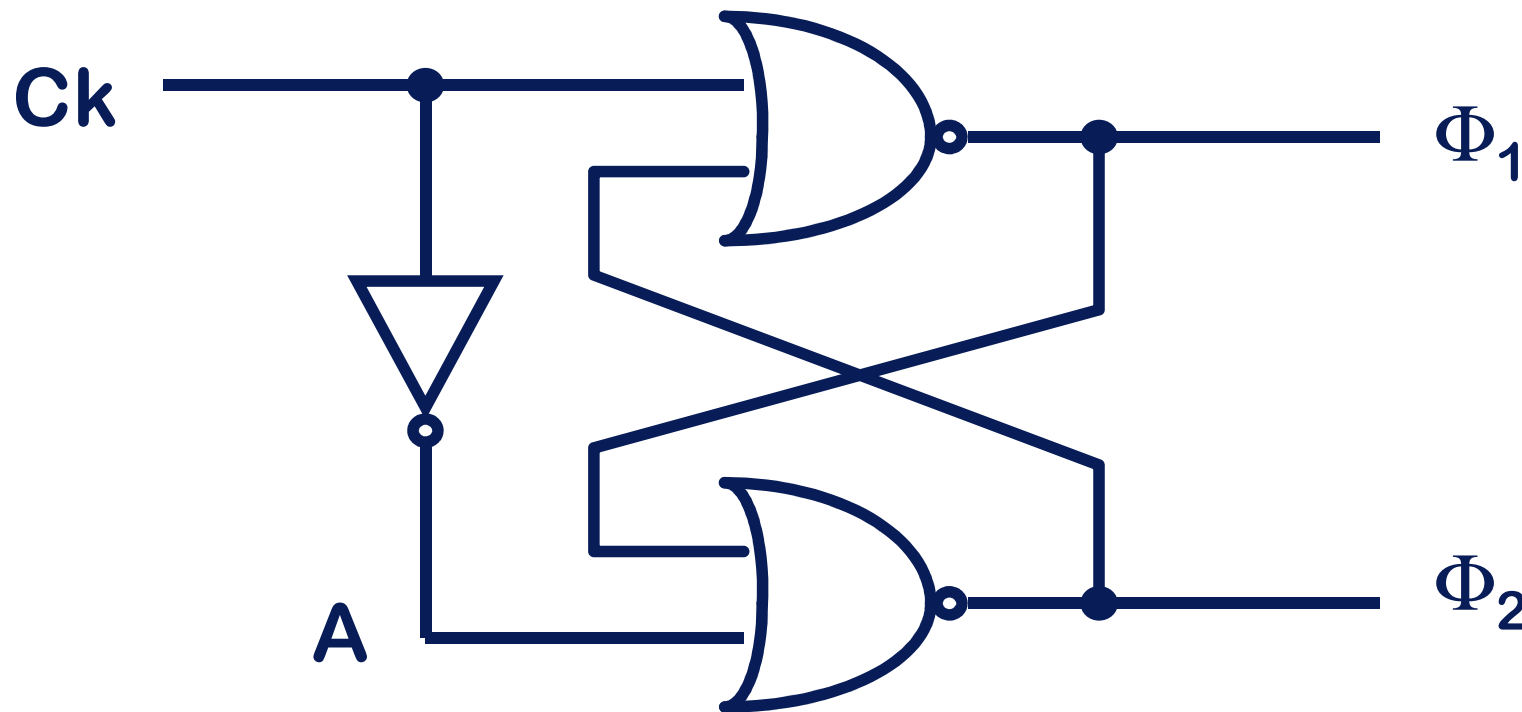
Layout simbolico: (16 transistor)

Forme d'Onda del Clock a due fasi

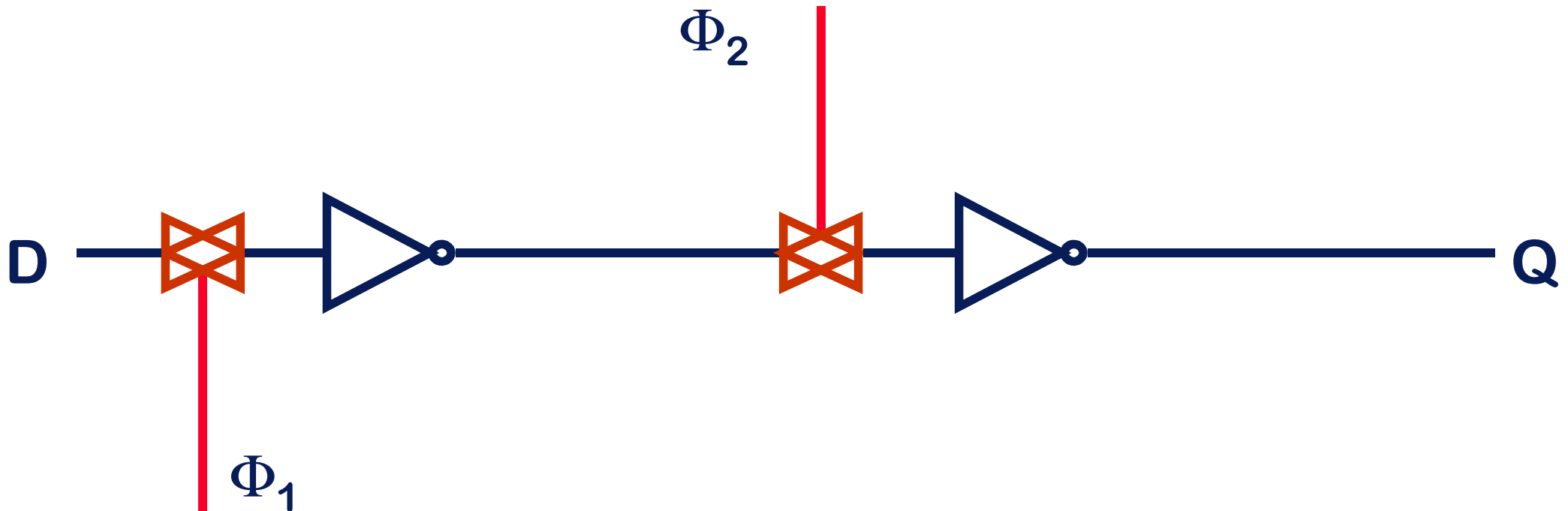


Generatore di Clock a due fasi

- Φ e $\overline{\Phi}$ non sono l'uno la negazione dell'altro !
 - Un opportuna spaziatura garantisce che non siano mai entrambi alti (neanche durante un breve intervallo di tempo)
- Ad es. Φ_1 e Φ_2 si possono generare col seguente circuito

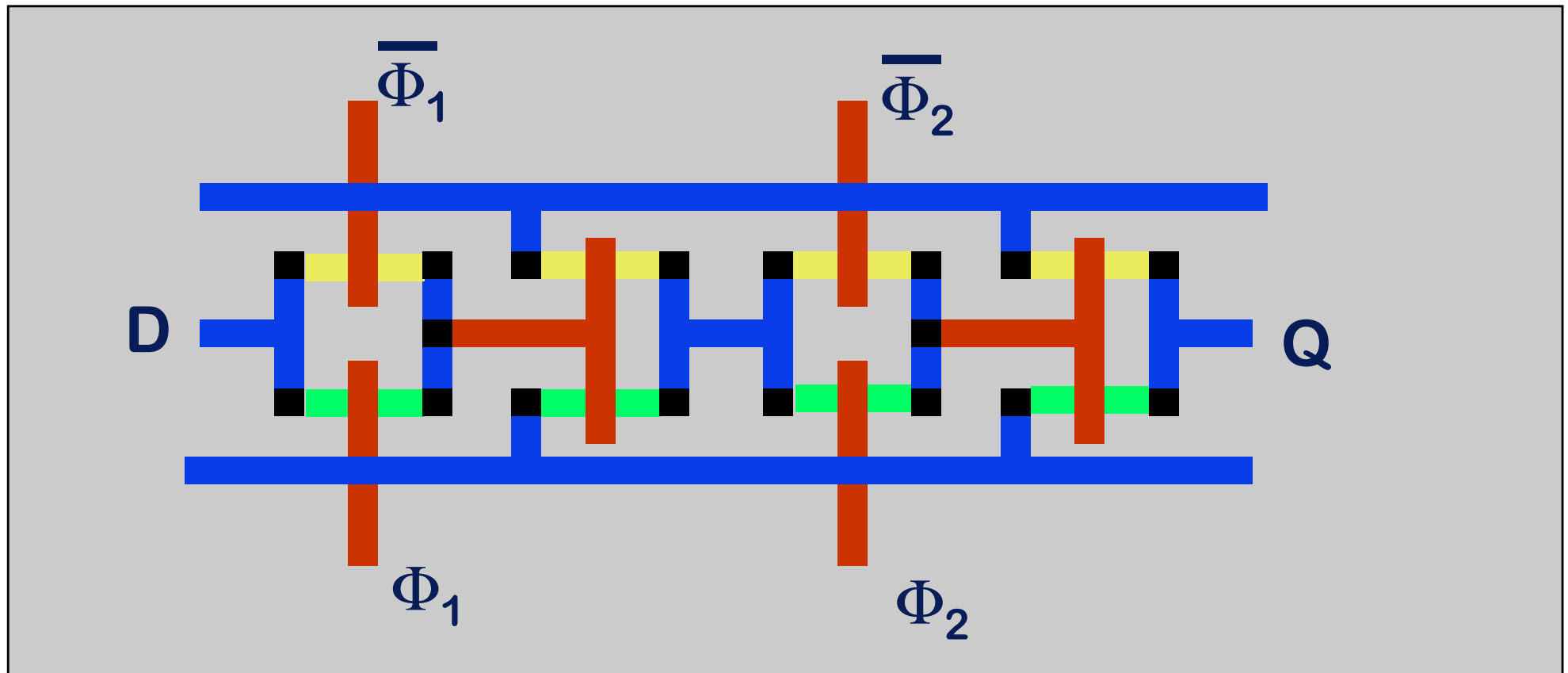


D-Edge Triggered DINAMICO



Layout simbolico: (8 transistor !)

Lay-Out Simbolico D-Edge Triggered DINAMICO

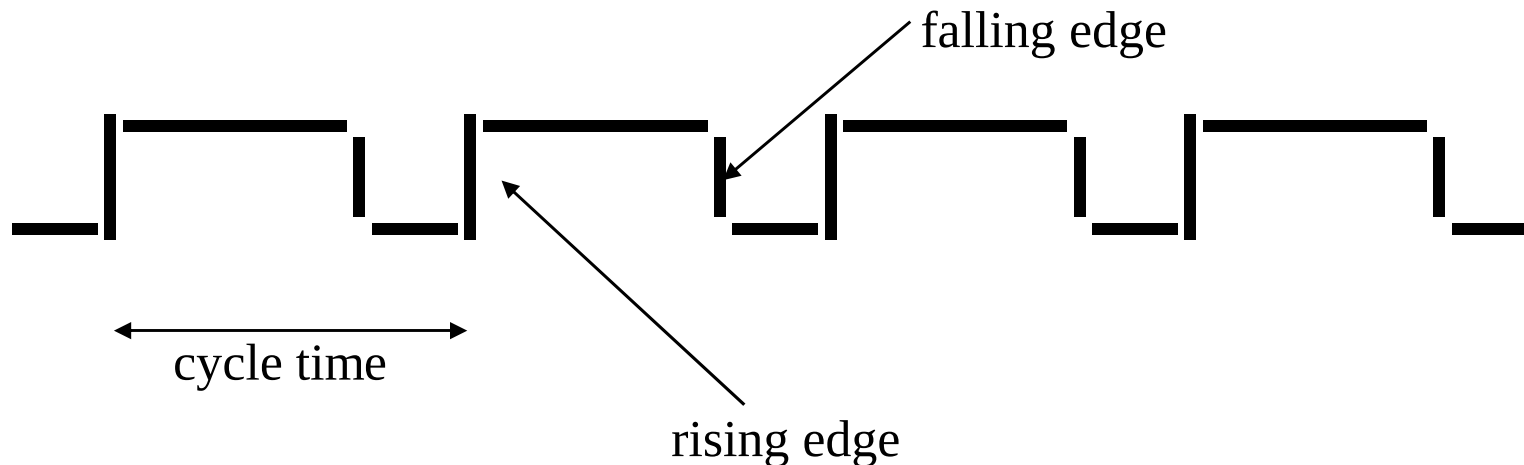


Realizzazione di elementi di memoria

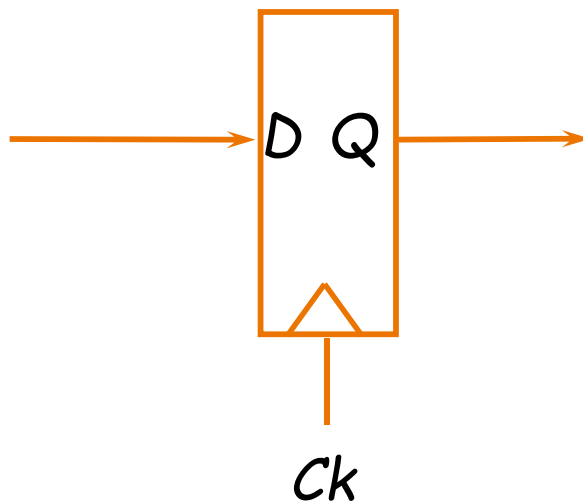
- Un elemento di memoria si puo' anche realizzare **SENZA** ricorrere a reti combinatorie
 - DRAM (RAM Dinamica) → usa un condensatore
 - La carica deve essere rinnovata periodicamente o viene persa
 - STTRAM (Spin Transfer Torque RAM) → usa l'allineamento degli spin degli elettroni in modo da definire il suo stato attraverso una giunzione a tunnel magnetico (MTJ)
 - Molto veloce e con durata illimitata ma non mantiene a lungo l'informazione e ha una bassa densita'
 - PCM (Phase Change Memory) → usa una lega calcogenura (Germanio, Antimonio, Tellurio = $\text{Ge}_2\text{Sb}_2\text{Te}_5$) interposta fra due elettrodi: il passaggio di corrente riscalda il materiale e induce un cambiamento di fase da amorfo a cristallino o viceversa
 - Moderatamente veloce e moderatamente densa, ma richiede abbastanza energia per memorizzare il bit, il materiale e' soggetto a invecchiamento e non e' semplice da fabbricare
 - RRAM (Resistive RAM o Memristor) → opera attraverso il movimento di ioni di ossigeno all'interno di una giunzione metallo-ossido (diossido di Titanio [1])
 - Ha stabilita' e durata molto lunghe ed e' sia molto veloce che densa essendo possibile stratificarle; migliora il comportamento piu' e' piccola

Elementi che memorizzano uno stato

- Possono essere di tipo
 - asincrono (non richiede un segnale di clock)
 - sincrono (richiede un segnale di clock)
- Nella logica sequenziale di tipo sincrono
 - Il cambiamento dell'uscita avviene rispetto a un segnale di clock (dopo un certo tempo rispetto a un riferimento sul segnale di clock)
 - Gli ingressi sono sensibili in corrispondenza di un qualche riferimento sul segnale di clock:

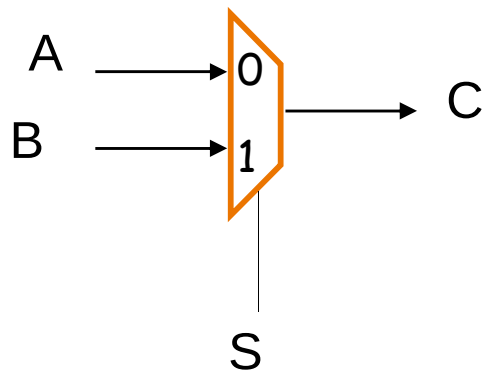


Simbolo del flip-flop D-Edge Triggered (flip-flop D)



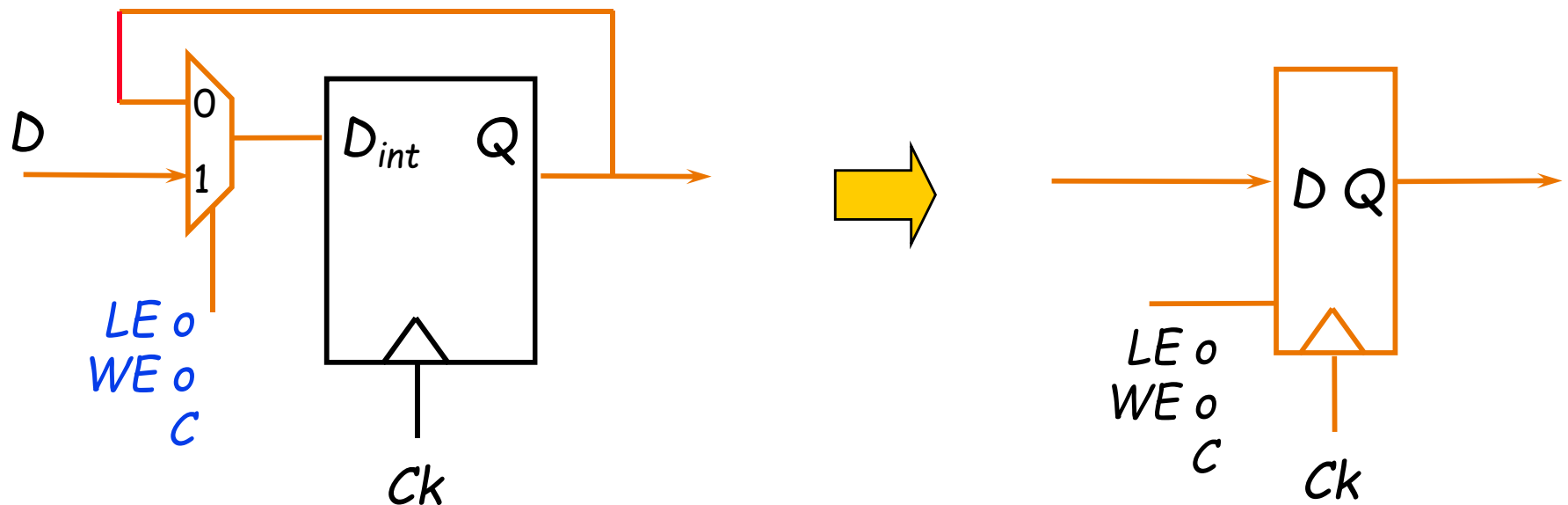
Il Multiplexer

- Seleziona uno degli ingressi, in base a un segnale di controllo, e lo invia in uscita



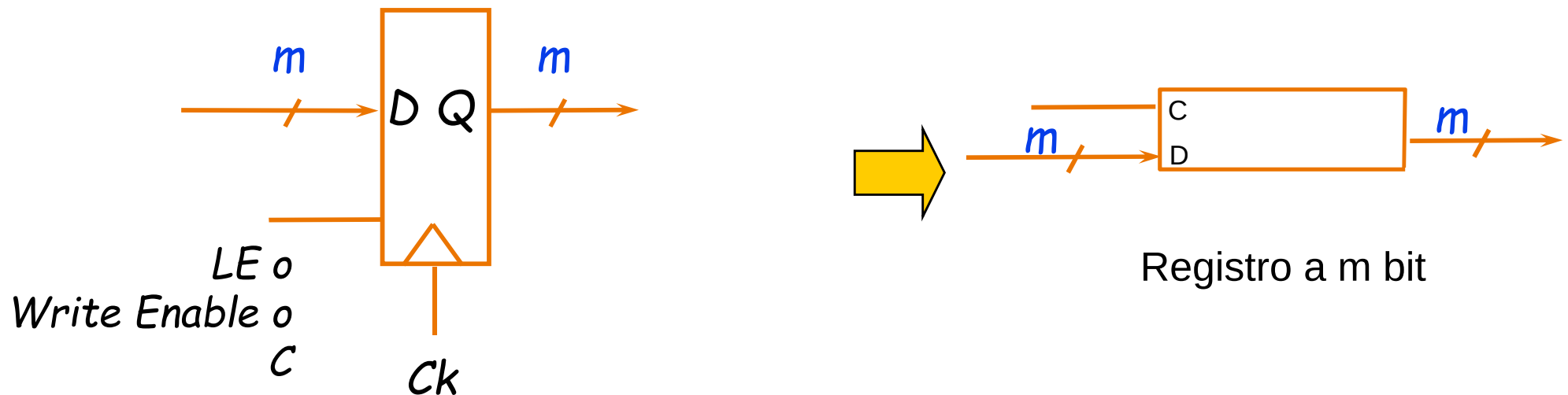
*nota: chiamiamo questo
“multiplexer a 2-ingressi”
anche se in realta' ne ha 3*

Cella di registro



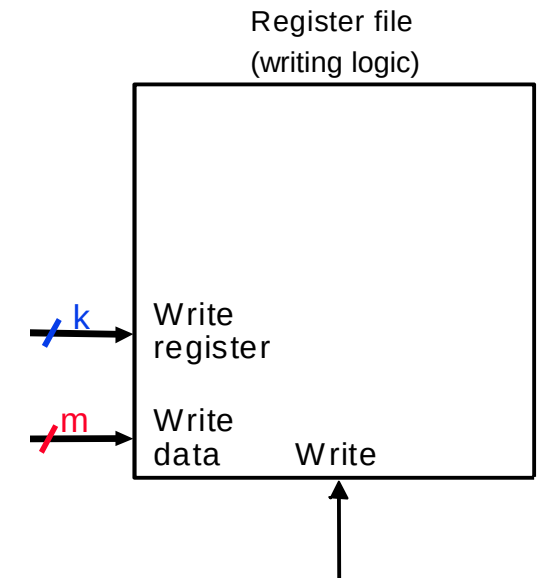
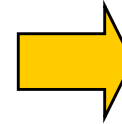
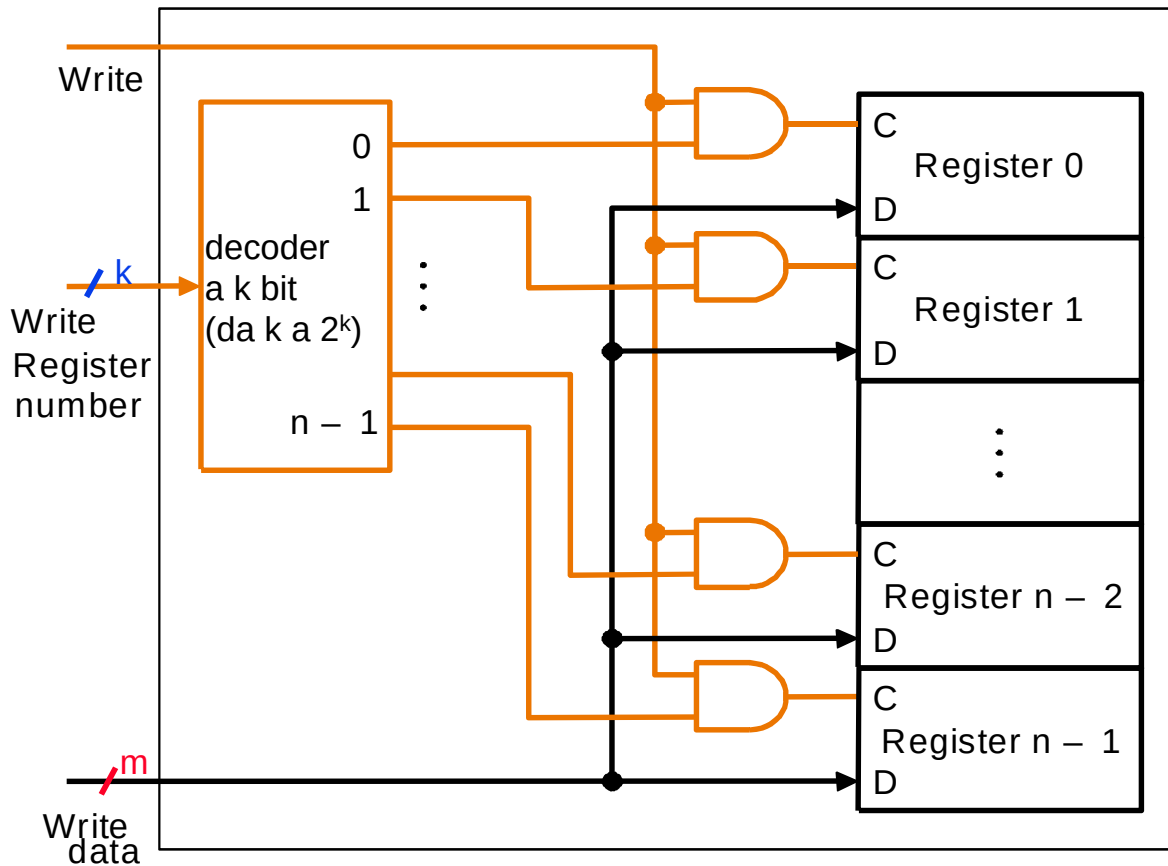
- Aggiungo un multiplexer il cui controllo si chiama
 - LE = Latch Enable o anche
 - WE = Write Enable o anche
 - C = Control

Registro a m bit



- Nell'ultima rappresentazione si intende implicitamente che sia presente un segnale Ck per il clock
- Register file:
 - Composto da n registri di questo tipo
 - Occorre della logica di scrittura
 - Occorre della logica di lettura

Register File: logica di scrittura



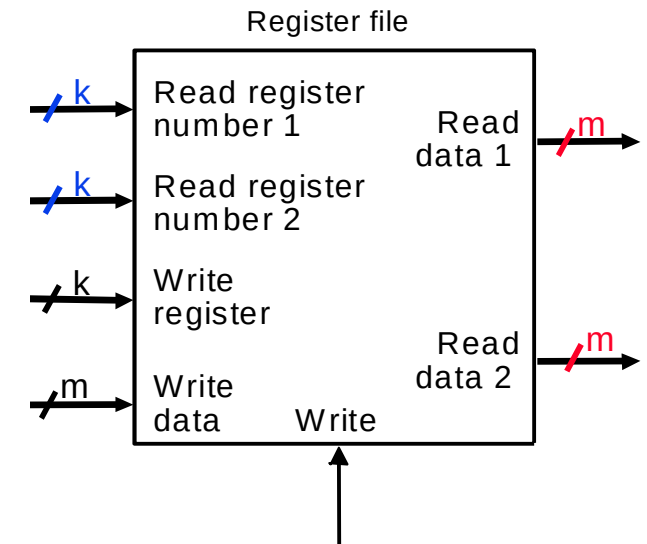
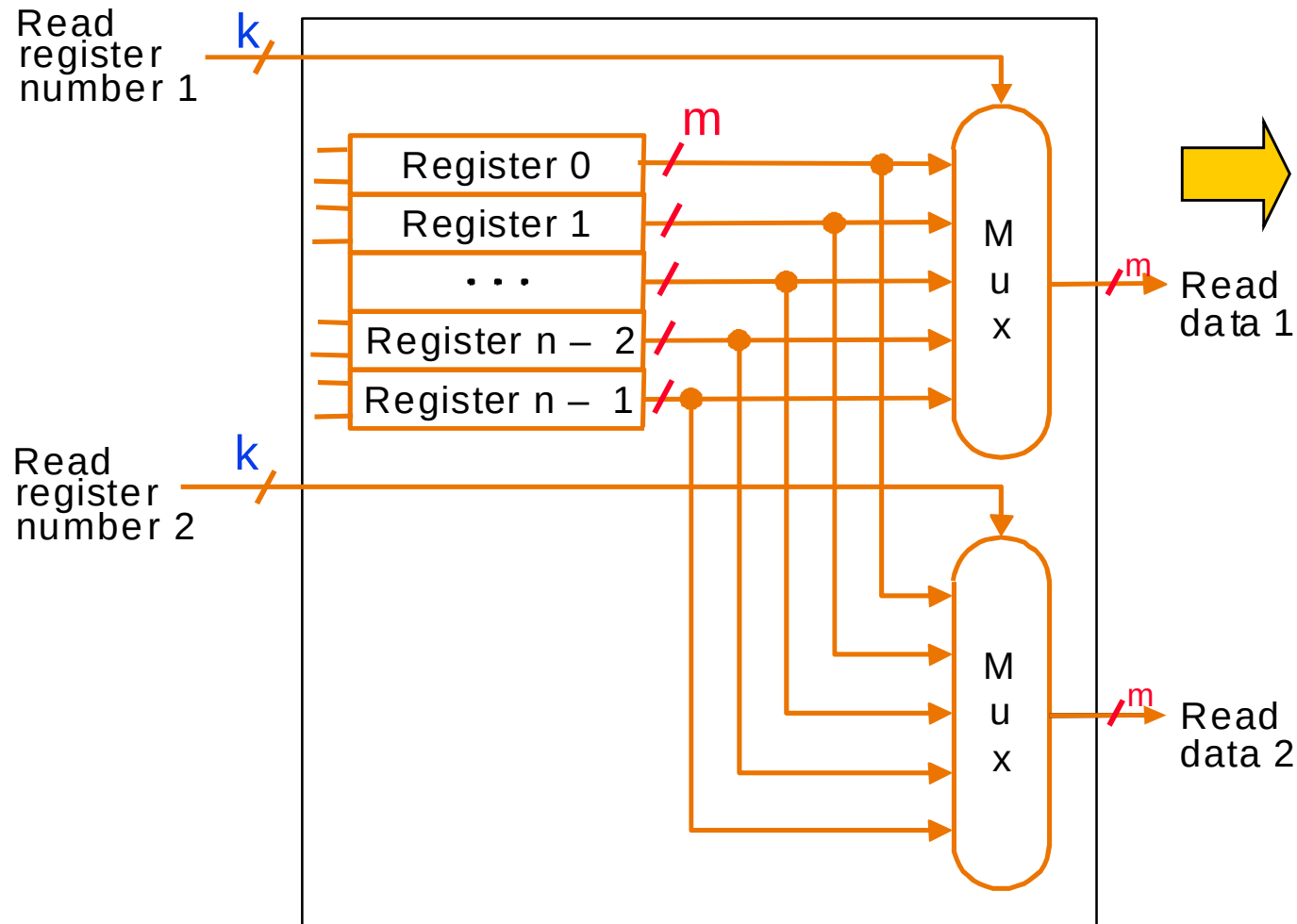
n = numero di registri disponibili (e.g. 32)

$k = \log_2(n)$ (e.g. 5)

m = larghezza dei registri (e.g. 32)

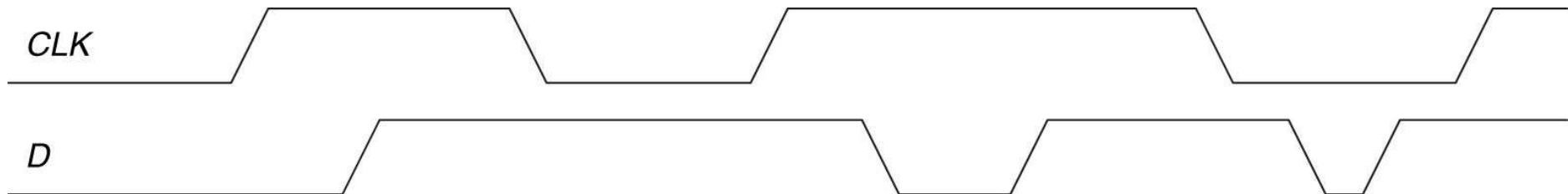
Register File: logica di lettura

- Si usano i flip-flop di tipo D



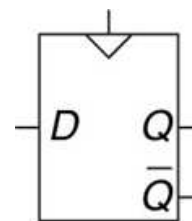
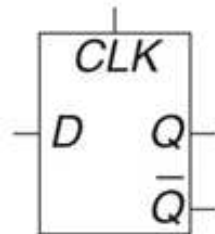
Riepilogo

- Latch e Flip-flop sono i blocchi architetturali fondamentali delle reti logiche sequenziali
- **Esercizio**
 - Vengono applicati i seguenti segnali D e CLK al D-latch e al D-flip-flop: disegnare le forme d'onda di uscita (Q) in entrambi i casi

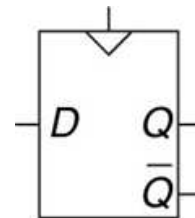
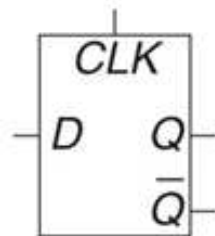
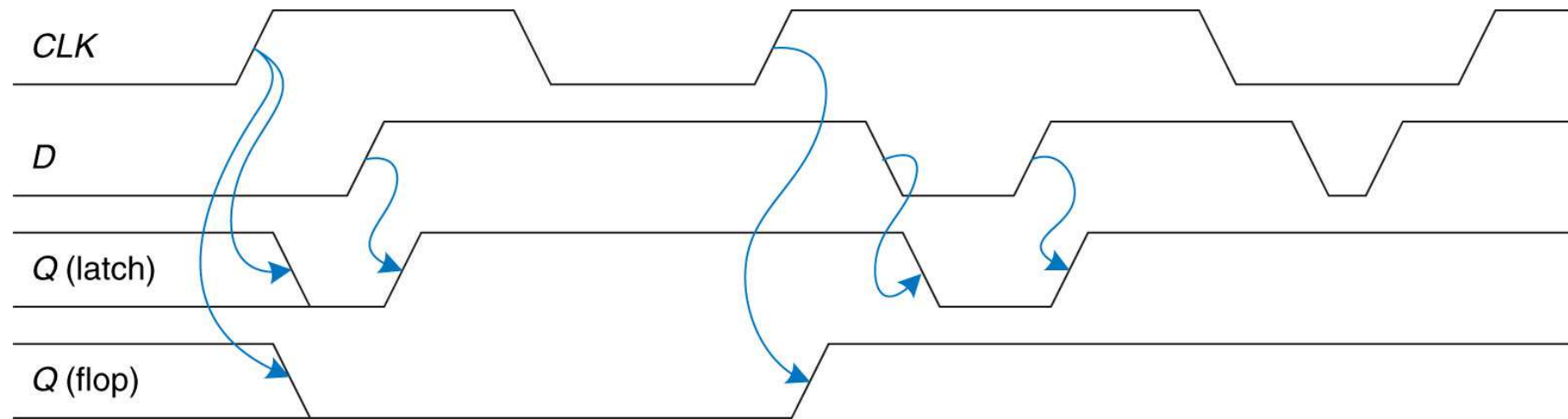


Q (latch)

Q (flop)



Esercizio - D-latch e D-FF



Esercizio2

- Descrivere un D-FF in Verilog

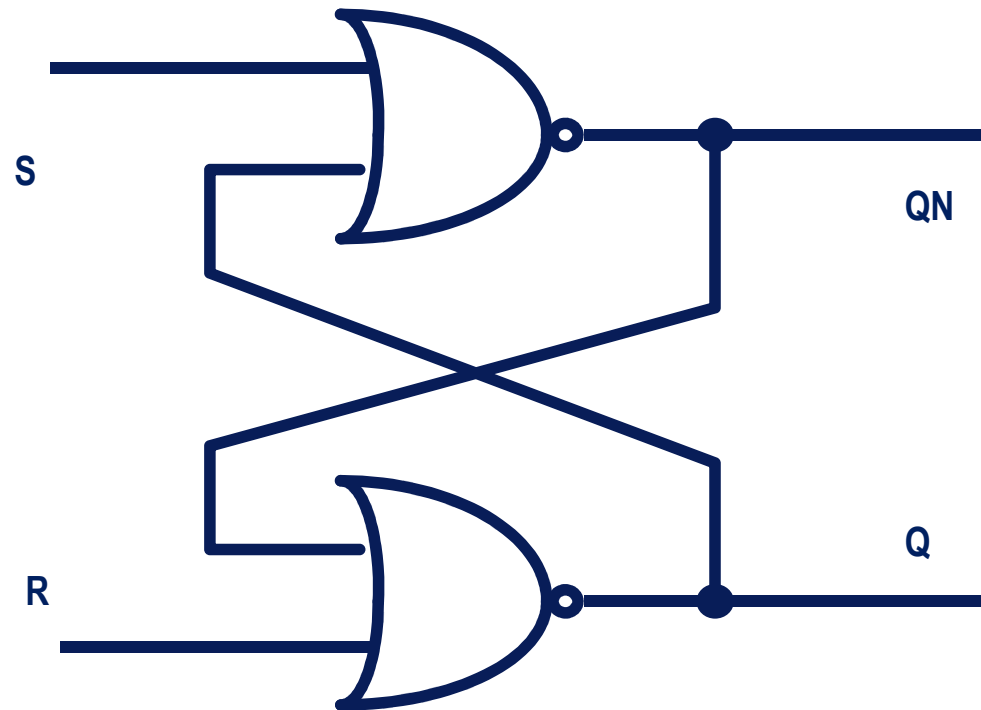
```
module DFF(clock,D,Q,Qbar);  
    input clock, D;  
    output reg Q; // Q is a reg since it is assigned in an always block  
    output Qbar;  
  
    assign Qbar = ~ Q; // Qbar is always just the inverse of Q  
  
    always @(posedge clock) // perform actions whenever the clock rises  
        Q <= D;  
endmodule
```

LEZIONE N° 94

Reti Logiche Sequenziali
(parte seconda)

Latch SR in Verilog

```
module Latch_SR(Q,QN,S,R)
  input S,R;
  output Q,QN;
  assign #1 Q=~(R|QN);
  assign #1 QN=~(S|Q);
endmodule
```



RAM 256x4 in Verilog

Cortesia da Prof. P. Corsini

```
module Ram(data, address, s_, mr_, mw_);  
    input [7:0]  address;  
    inout [3:0]  data;  
    input        s_;  
    input        mr_;  
    input        mw_;  
    reg  [3:0] LOCATION[0:255]; // 256 memory cells  
    wire alfa=~(s_|mr_);  
    wire beta=~(s_|mw_);  
  
    assign data=(alfa==1)? LOCATION[address]:'BZZ ;    //READ  
    always @(mw_ or data)if(beta==1) LOCATION[address]= data; //WRITE  
  
endmodule
```

ROM 4x8

Cortesia da Prof. P. Corsini

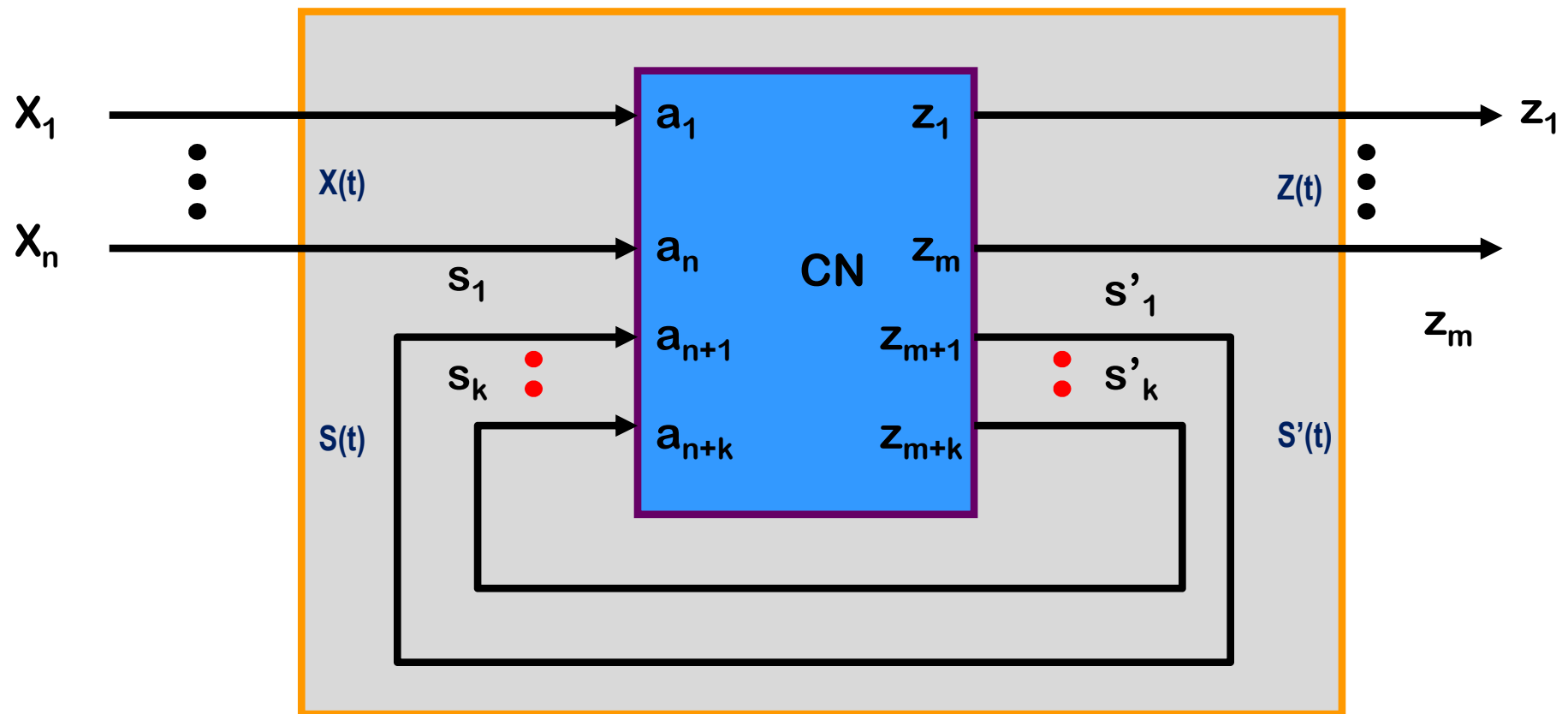
```
module Eprom(d7_d0, address, s_, oe_);  
    input  [1:0]  address;  
    output [7:0]  d7_d0;  
    input                s_;  
    input                oe_;  
  
    wire alfa=~(s_|oe_);  
    assign d7_d0=(alfa==0)? 'BZZ : LOCATION(address);  
  
    function [7:0] LOCATION;  
        input [1:0] address;  
        casex(address)  
            0:      LOCATION = 0;  
            1:      LOCATION = 1;  
            2:      LOCATION = 2;  
            3:      LOCATION = 3;  
        endcase  
    endfunction  
endmodule
```

Macchina di Mealy asincrona

- Le variabili d'uscita, in un determinato istante, sono funzione del valore degli ingressi e delle variabili di stato

$$Z(t) = F(X(t), S(t))$$

$$S'(t) = G(X(t), S(t))$$



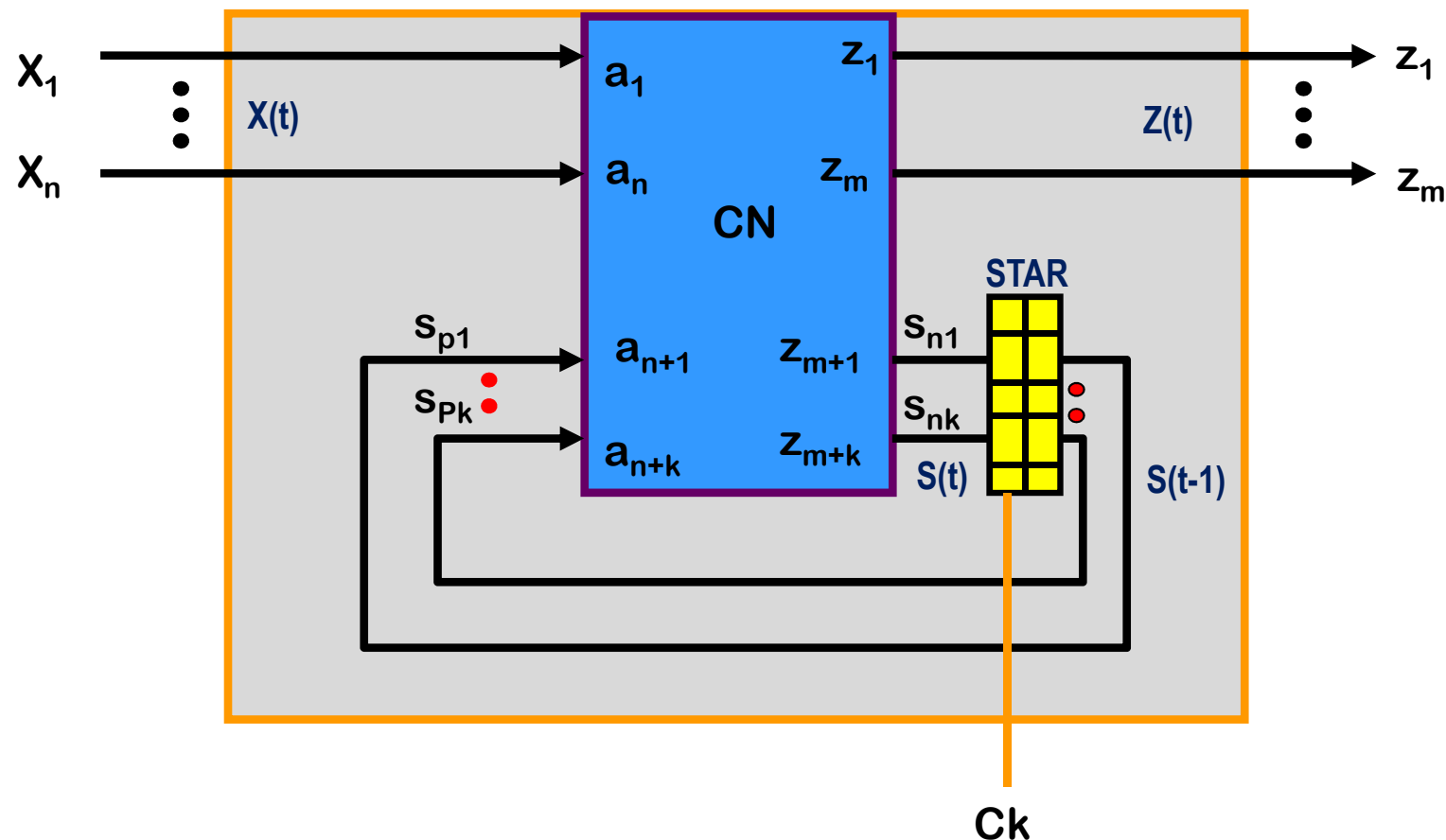
La rete CN è una rete combinatoria

Macchina di MEALY (sincronizzata)

- Le uscite sono funzioni delle variabili di stato e degli ingressi

$$Z(t) = f(X(t), S(t-1))$$

$$S(t) = g(X(t), S(t-1))$$



STAR e' costituito da
Flip-Flop D

Macchina di MOORE (sincronizzata)

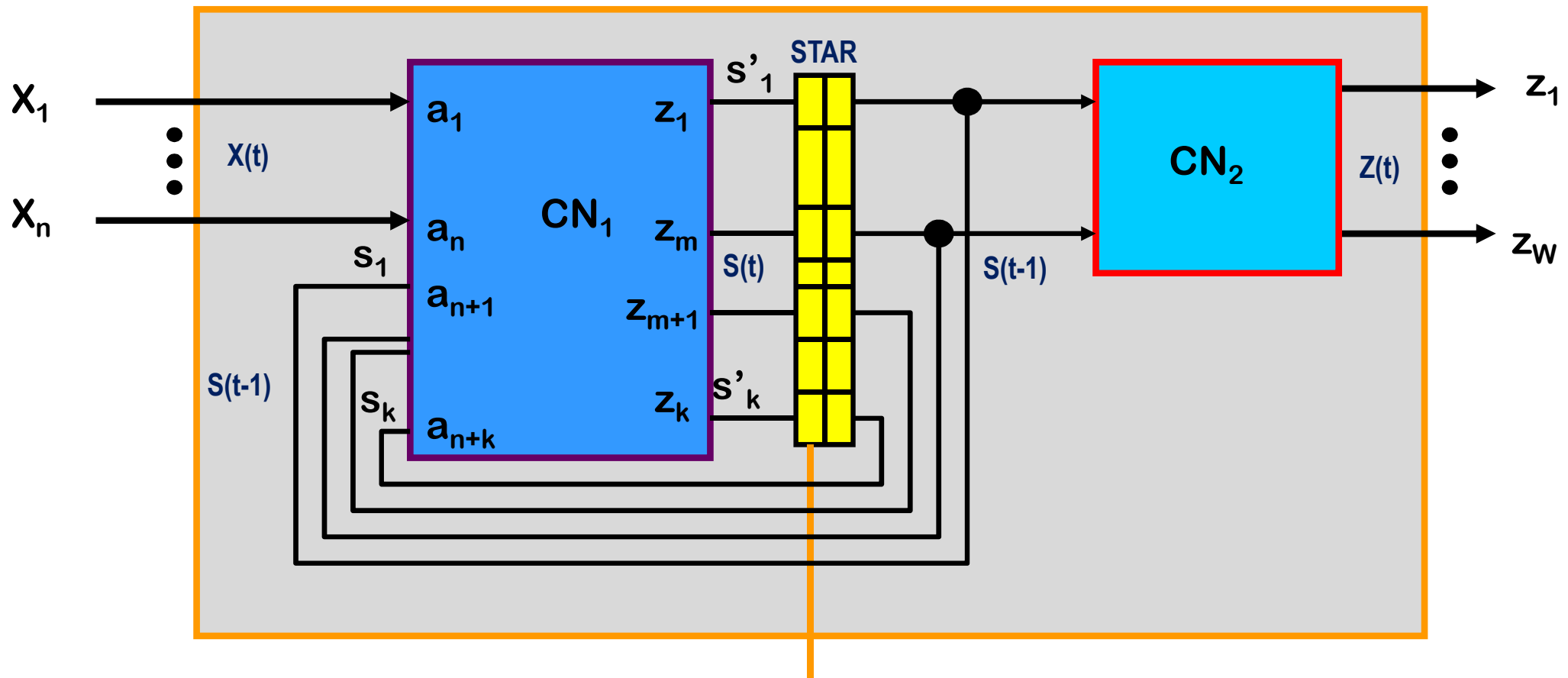
- Le variabili d'uscita, in un determinato istante, sono funzione del sole variabili di stato

$$Z(t) = f(S(t-1))$$

$$S(t) = g(X(t), S(t-1))$$

In MOORE: l'uscita dipende solo dallo stato

$S(t-1)$ e' lo stato presente
 $S(t)$ e' lo stato successivo



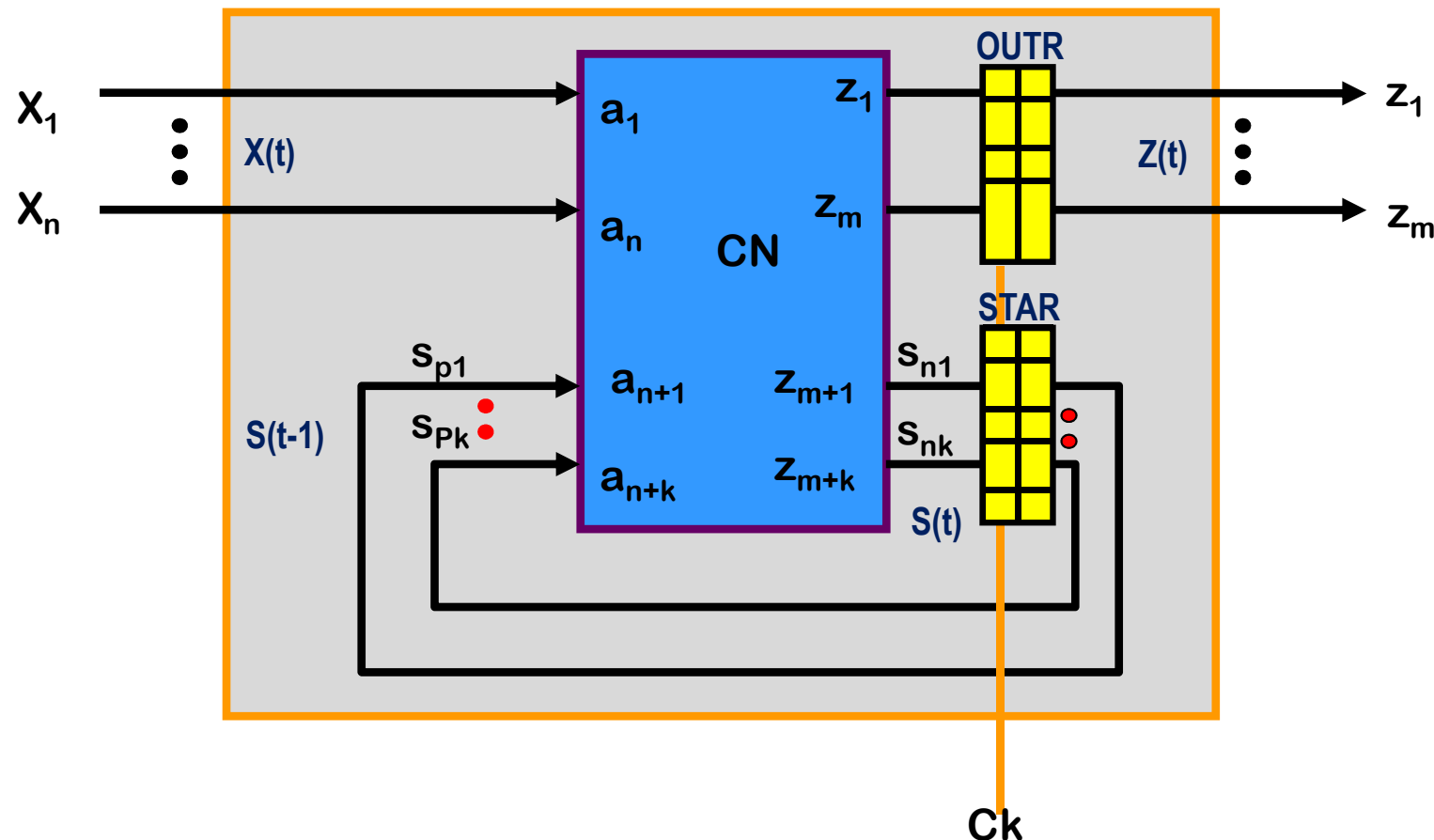
Macchina di Mealy Ritardata (sincronizzata)

- Le uscite sono funzioni delle variabili di stato e degli ingressi, ma risultano sincronizzate

$$Z(t) = f(X(t), S(t-1))$$

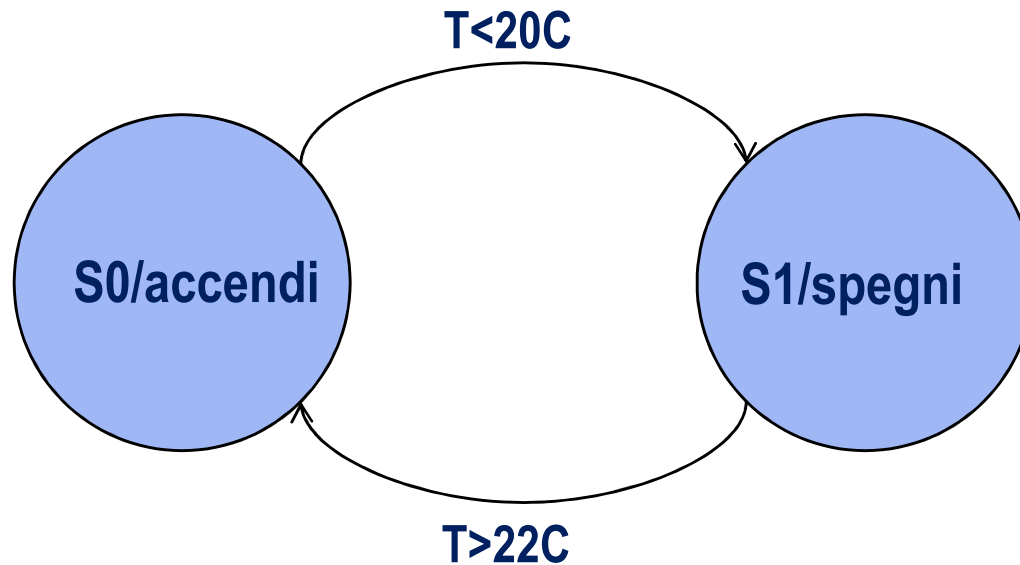
$$S(t) = g(X(t), S(t-1))$$

In MEALY: l'uscita dipende sia dallo stato che dagli ingressi



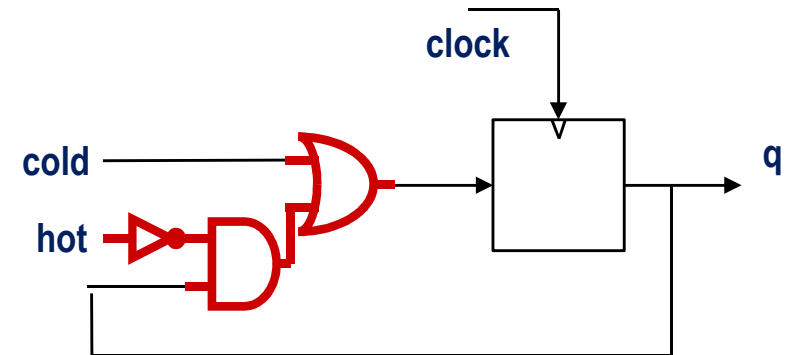
Specifica di una FSM per implementazione con Moore

- S0 → Riscaldamento Acceso
- S1 → Riscaldamento Spento



cold/hot	CN1			
	00	01	11	10
Off (0)	0	0	-	1
On (1)	1	0	-	1

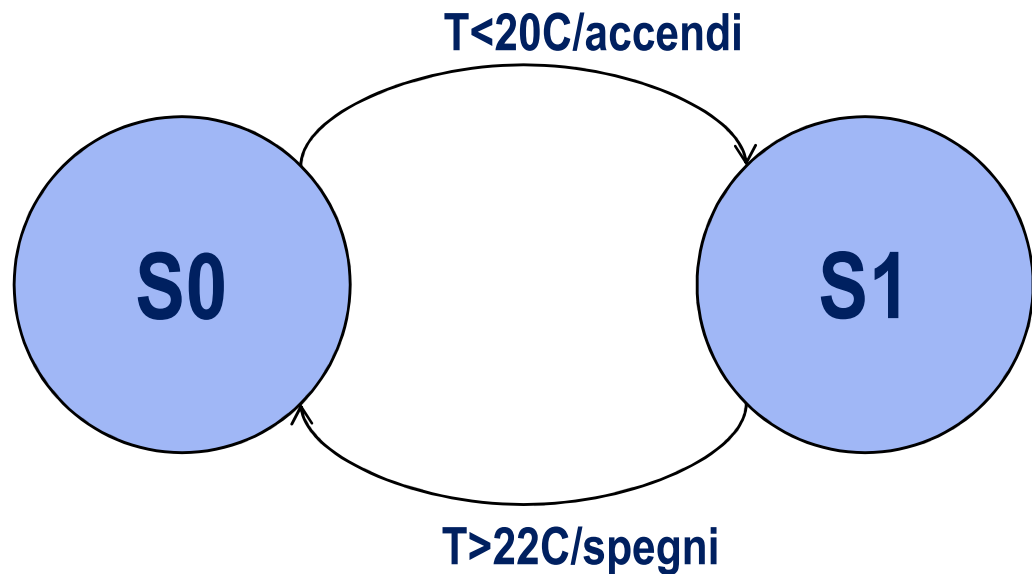
$$q_{next} = cold + \overline{hot} * q$$



In genere MOORE produce piu' stati, ma si puo' implementare usando solo LUT

Specifica di una FSM per implementazione con Mealy

- S0 → Riscaldamento Spento
- S1 → Riscaldamento Acceso



FARE PER ESERCIZIO

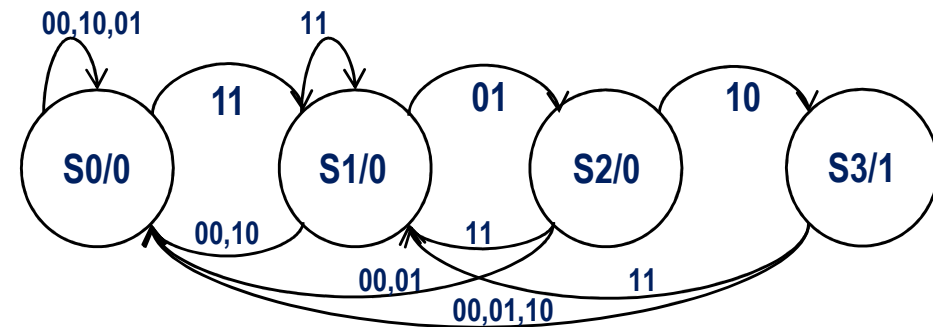
In genere MEALY produce meno stati, ma puo' richiedere sia LUT che logica aggiuntiva

Template generale per Moore in Verilog

```
module Moore(z,x,clock,reset_);
    input          clock, reset_;
    input  [N-1:0]  x;
    output [M-1:0]  z;
    reg      [W-1:0] STAR;
    parameter S0=codifica0, ..., SKmeno1=codificaKmeno1;
    assign z=(STAR==S0)?codifica0:
            (STAR==S1)?codifica1:
            ...
            /*(STAR==SKmeno1)*/ codificaKmeno1;
    always @(reset_==0) #1 STAR<=stato_iniziale;
    always @(posedge clock) if (reset_==1) #3
        casex(STAR)
            S0: STAR<=g0(x);
            S1: STAR<=g1(x);
            ...
            SKmeno1: STAR<=gKmeno1(x);
        endcase
endmodule
```

Esempio - riconoscitore di sequenza 11,01,10 con Moore

```
module ricseq110110moore(z,x,clk,reset_);
    input          clk, reset_;
    input [1:0]     x;
    output          z;
    reg [1:0]       STAR;
    parameter S0='B00, S1='B01, S2='B10, S3='B11;
    assign z=(STAR==S3)?1:0;
    always @(reset_==0) #1 STAR<=S0;
    always @(posedge clk) if (reset_==1) #3
        casex(STAR)
            S0: STAR<=(x=='B11)?S1:S0;
            S1: STAR<=(x=='B01)?S2:(x=='B11)?S1:S0;
            S2: STAR<=(x=='B10)?S3:(x=='B11)?S1:S0;
            S3: STAR<=(x=='B11)?S1:S0;
        endcase
endmodule
```

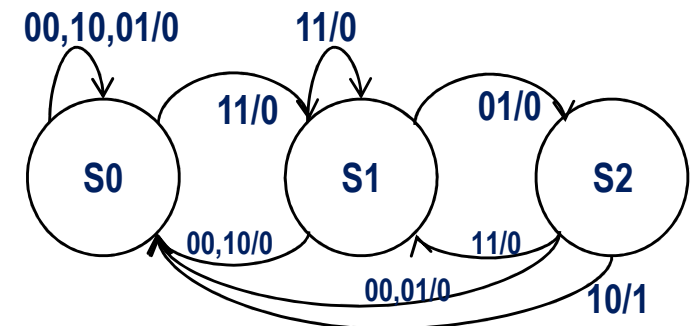


Template generale per Mealy in Verilog

```
module Mealy(z,x,clock,reset_);
    input          clock, reset_;
    input  [N-1:0]  x;
    output [M-1:0]  z;
    reg  [W-1:0]    STAR;
    parameter  S0=codifica0, ..., SKmeno1=codificaKmeno1;
    assign z=(STAR==S0)?f0(x):
            (STAR==S1)?f1(x):
            ...
            /*(STAR==SKmeno1)*/ fKmeno1(x);
    always @(reset_==0) #1 STAR<=stato_iniziale;
    always @(posedge clock) if (reset_==1) #3
        casex(STAR)
            S0: STAR<=g0(x);
            S1: STAR<=g1(x);
            ...
            SKmeno1: STAR<=gKmeno1(x);
        endcase
endmodule
```

Esempio - riconoscitore di sequenza 11,01,10 con Mealy

```
module ricseq110110mealy(z,x,clk,reset_);
  input          clk, reset_;
  input  [1:0]    x;
  output          z;
  reg  [1:0]      STAR;
  parameter  S0=`B00, S1=`B01, S2=`B10;
  assign  z=((STAR==S2)&(x==`B10))?1:0;
  always @(reset_==0) #1 STAR<=S0;
  always @(posedge clk) if (reset_==1) #3
    casex(STAR)
      S0: STAR<=(x==`B11)?S1:S0;
      S1: STAR<=(x==`B01)?S2:(x==`B11)?S1:S0;
      S2: STAR<=(x==`B11)?S1:S0;
    endcase
endmodule
```

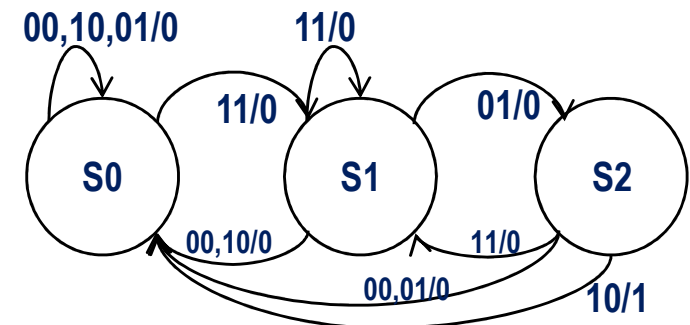


Template generale per Mealy Ritardato in Verilog

```
module MealyRitardato(z,x,clock,reset_);
    input          clock, reset_;
    input  [N-1:0]  x;
    output [M-1:0]  z;
    reg     [W-1:0]  STAR;
    reg     [M-1:0]  OUTR;
    parameter  S0=codifica0, ..., SKmeno1=codificaKmeno1;
    assign      z=OUTR;
    always @(reset_==0) #1 begin STAR<=...; OUT<=...; end;
    always @(posedge clock) if (reset_==1) #3
        casex(STAR)
            S0: begin OUTR<=f0(x); STAR<=g0(x); end
            S1: begin OUTR<=f1(x); STAR<=g1(x); end
            ...
            SKmeno1: begin OUTR<=fKmeno1(x); STAR<=gKmeno1(x); end
        endcase
endmodule
```


Esempio - riconoscitore di sequenza 11,01,10 con Mealy Rit

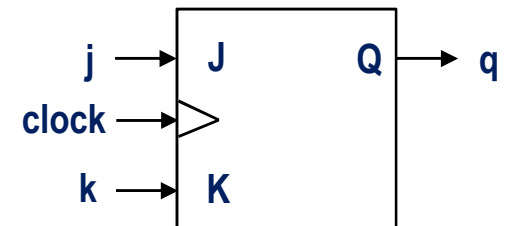
```
module ricseq110110mealyritardato(z,x,clock,reset_);
  input          clock, reset_;
  input [1:0]    x;
  output         z;
  reg [1:0]      STAR;
  reg            OUTR;
  parameter S0=`B00, S1=`B01, S2=`B10;
  assign         z=OUTR;
  always @(reset_==0) #1 begin; OTR<=0; STAR<=S0; end
  always @(posedge clock) if (reset_==1) #3
    casex(STAR)
      S0: OTR<=0; STAR<=(x=='B11)?S1:S0; end
      S1: OTR<=0; STAR<=(x=='B01)?S2:(x=='B11)?S1:S0; end
      S2: OTR<=(x=='B10)?1:0; STAR<=(x=='B11)?S1:S0; end
    endcase
endmodule
```



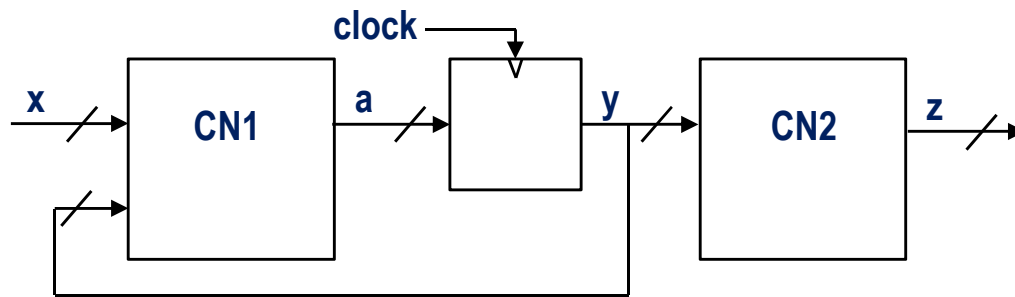
Flip-Flop JK

- Il FF-JK e' tale che in corrispondenza del fronte in salita del clock:
- per JK=10 l'uscita va a 1,
per JK=01 l'uscita va a 0,
per JK=00 il FF conserva (l'uscita resta inalterata)
per JK=11 il FF commuta (l'uscita diventa il suo negato)

```
module FFJK(q,j,k,clock,reset_);  
    input          clock, reset_;  
    input          j,k;  
    output         q;  
    reg            STAR;  
    parameter      S0=0, S1=1;  
    assign q=(STAR==S0)?0:1;  
    always @(reset_==0) #1 STAR<=0;  
    always @(posedge clock) if (reset_==1) #3  
        casex(STAR)  
            S0: STAR<=(j==1)?S1:S0;  
            S1: STAR<=(k==1)?S0:S1;  
        endcase  
endmodule
```



Sintesi del FF-JK con Moore



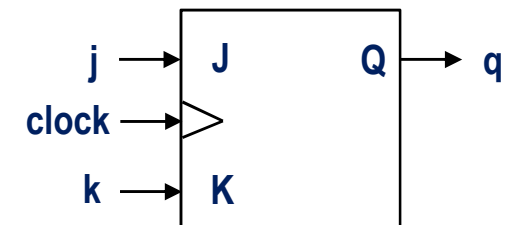
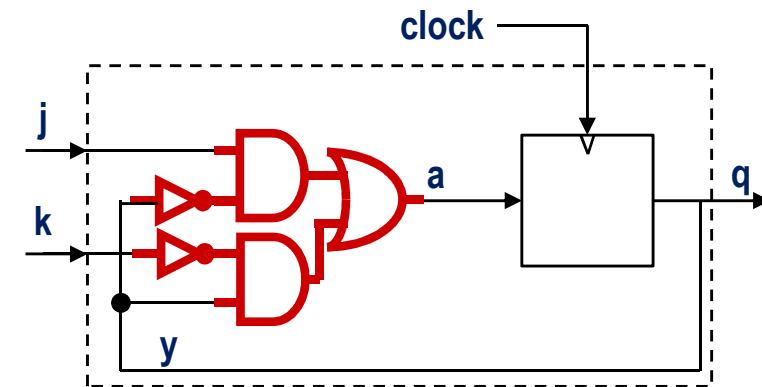
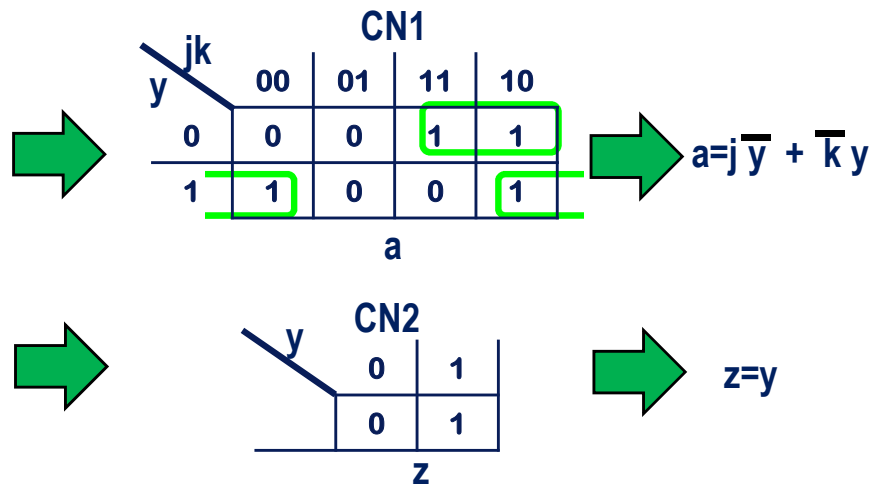
- 2 stati → STAR ha 1 bit
- → a ha 1 bit, y ha 1 bit
z ha 1 bit, (x ha 2 bit)

per JK=10 l'uscita va a 1,
per JK=01 l'uscita va a 0,
per JK=00 il FF conserva
per JK=11 il FF commuta

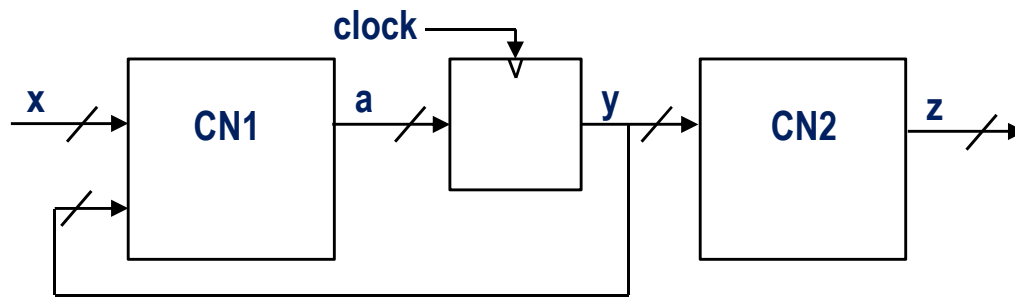
Tabella delle transizioni
e delle uscite

jk \	00	01	11	10	q
s0	s0	s0	s1	s1	0
s1	s1	s0	s0	s1	1

Stato	y0
s0	0
s1	1



Sintesi del riconoscitore di sequenza 11,01,10 con Moore



- 4 stati → STAR ha 2 bit
- → a ha 2 bit, y ha 2 bit
z ha 1 bit, (x ha 2 bit)

Stato	y1 y0
S0	00
S1	01
S2	11
S3	10

Handwritten Karnaugh map for CN1:

		$y_1 y_0$			
		00	01	11	10
$x_1 x_0$	00	00	00	01	00
	01	00	11	01	00
	11	00	00	01	10
	10	00	00	01	00

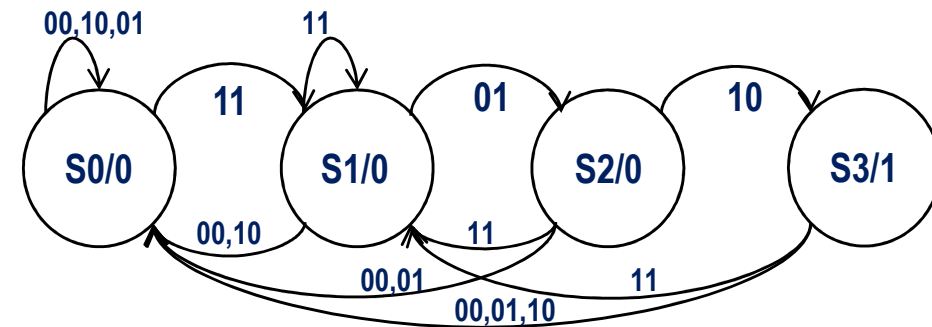
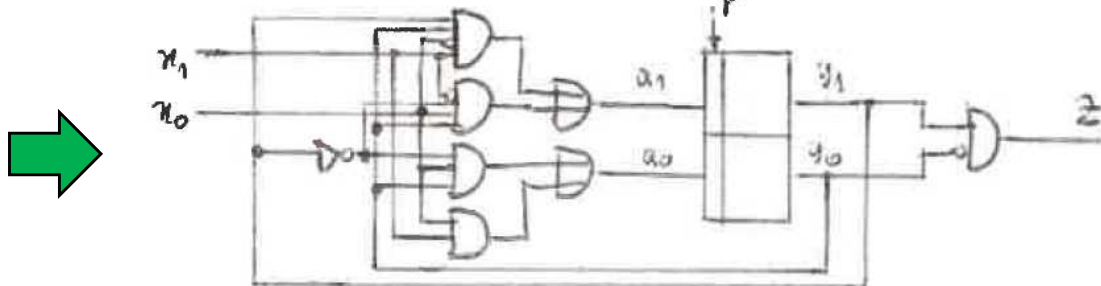
$$a_1 = \bar{x}_1 \bar{x}_0 \bar{y}_1 y_0 + x_1 \bar{x}_0 y_1 y_0$$

$$a_0 = x_1 x_0 + x_0 \bar{y}_1 y_0$$

Handwritten Karnaugh map for CN2:

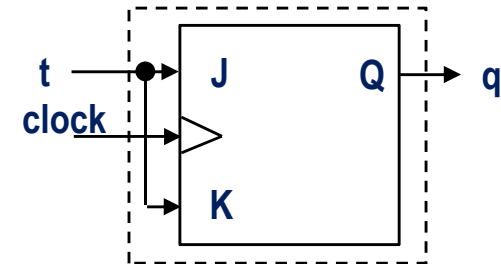
$y_1 y_0$	z
00	0
01	0
11	0
10	1

$$z = y_1 \bar{y}_0$$

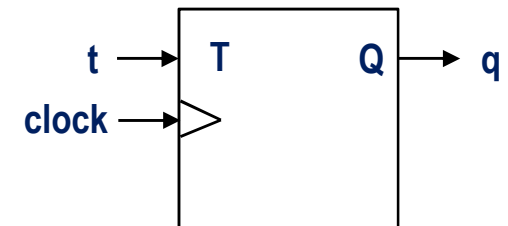


Flip-Flop T (detto FF contatore)

- Il FF-T e' tale che in corrispondenza del fronte in salita del clock:
- per $T=0$ il FF conserva
(l'uscita rimane inalterata)
- per $T=1$ il FF commuta
(l'uscita diventa il suo negato)



```
module FFT(q,t,clock,reset_);  
    input          clock, reset_;  
    input          t;  
    output         q;  
    reg            STAR;  
    parameter S0=0, S1=1;  
    assign q=(STAR==S0)?0:1;  
    always @(reset_==0) #1 STAR<=stato_iniziale;  
    always @(posedge clock) if (reset_==1) #3  
        casex(STAR)  
            S0: STAR<=(t==0)?S0:S1;  
            S1: STAR<=(t==0)?S1:S0;  
        endcase  
endmodule
```



Sintesi del riconoscitore di sequenza 11,01,10 con Mealy Rit

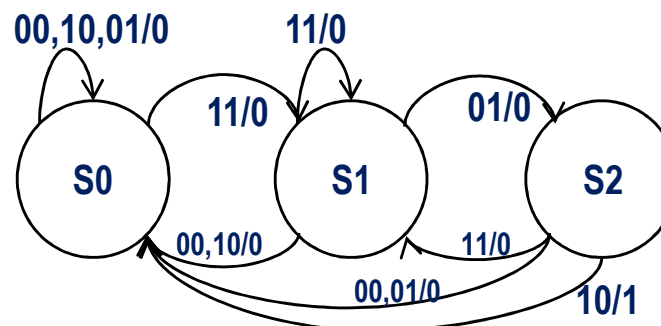
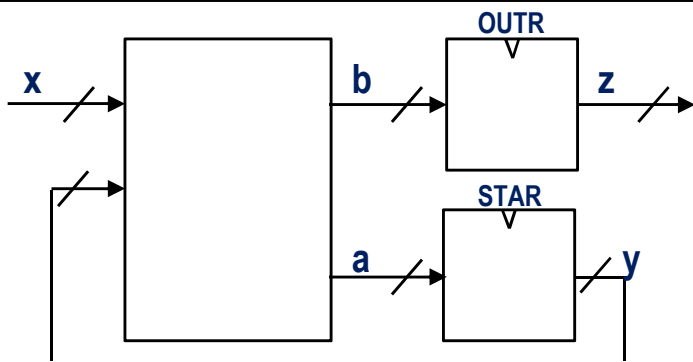


Tabella delle transizioni

x_1x_0	00	01	11	10
S0	S0	S0	S1	S0
S1	S0	S2	S1	S0
S2	S0	S0	S1	S0

Stato	y_1y_0
S0	00
S1	11
S2	01

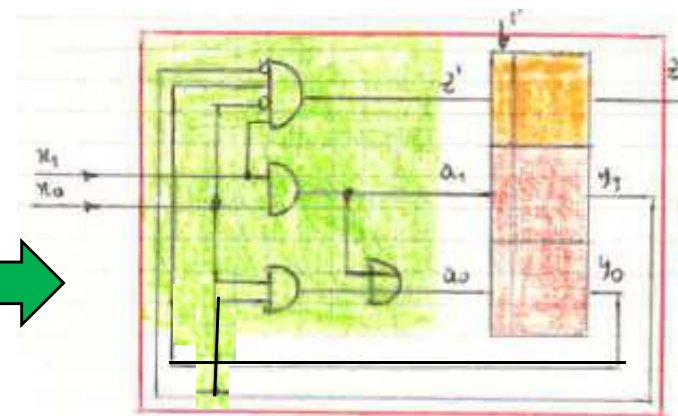
x_1x_0	00	01	11	10
y_1y_0 00	00	00	11	00
01	00	00	11	00
11	00	01	11	00
10	--	--	--	--

x_1x_0	00	01	11	10
y_1y_0 00	0	0	0	0
01	0	0	0	1
11	0	0	0	0
10	-	-	-	-

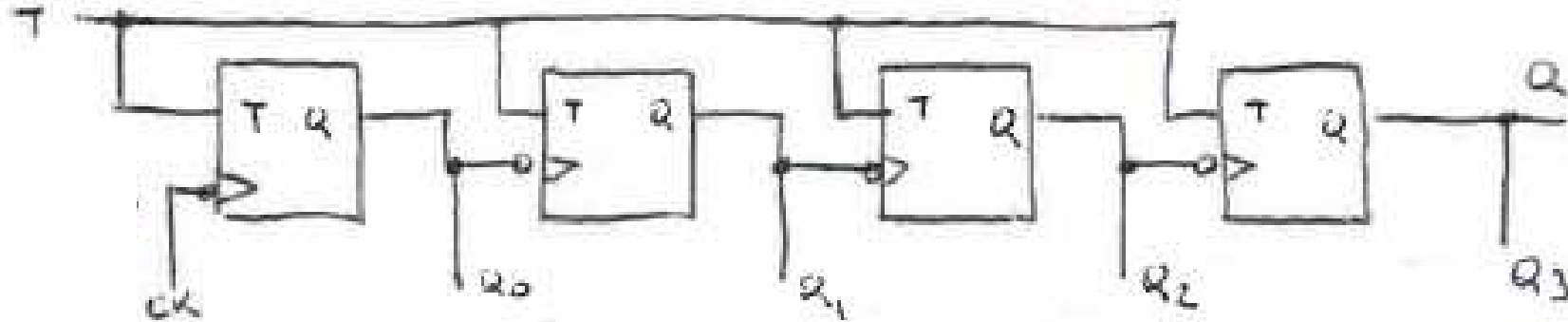
$$a_1 = x_1 x_0$$

$$a_0 = x_1 x_0 + x_0 y_1$$

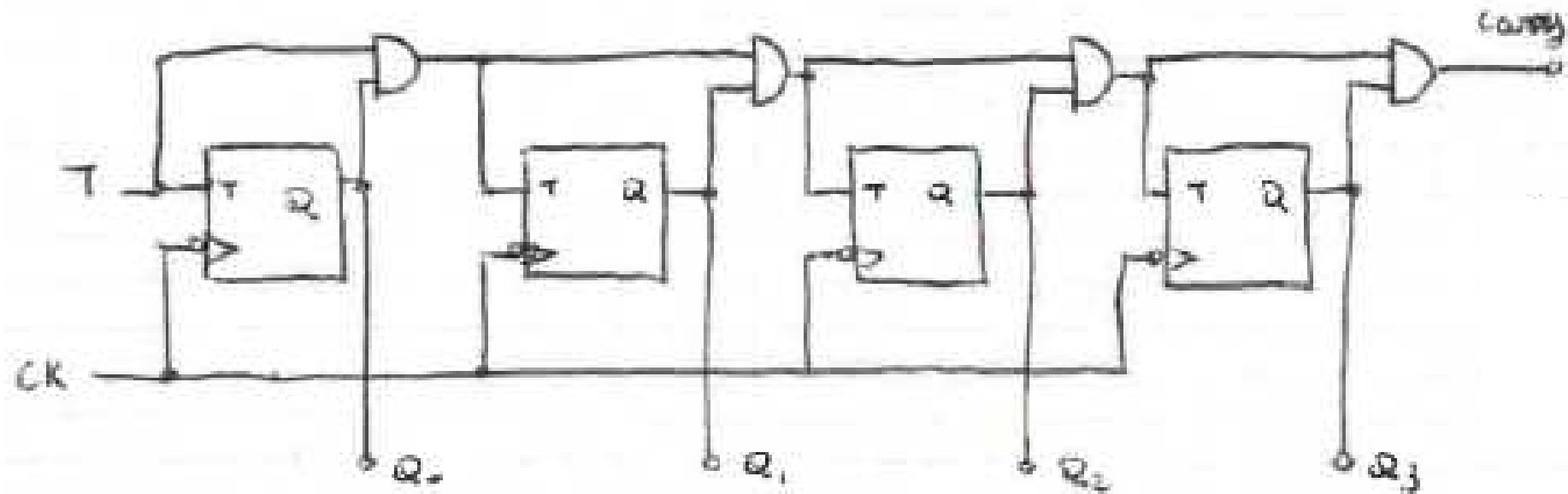
$$b_0 = x_1 \bar{x}_0 \bar{y}_1 y_0$$



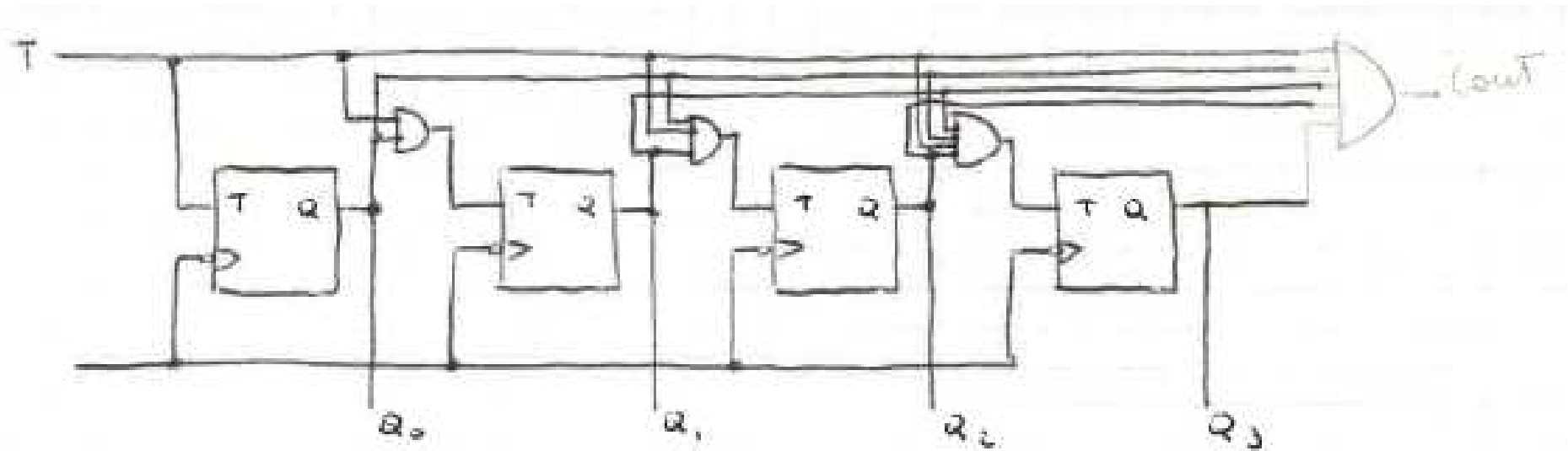
Ripple Counter



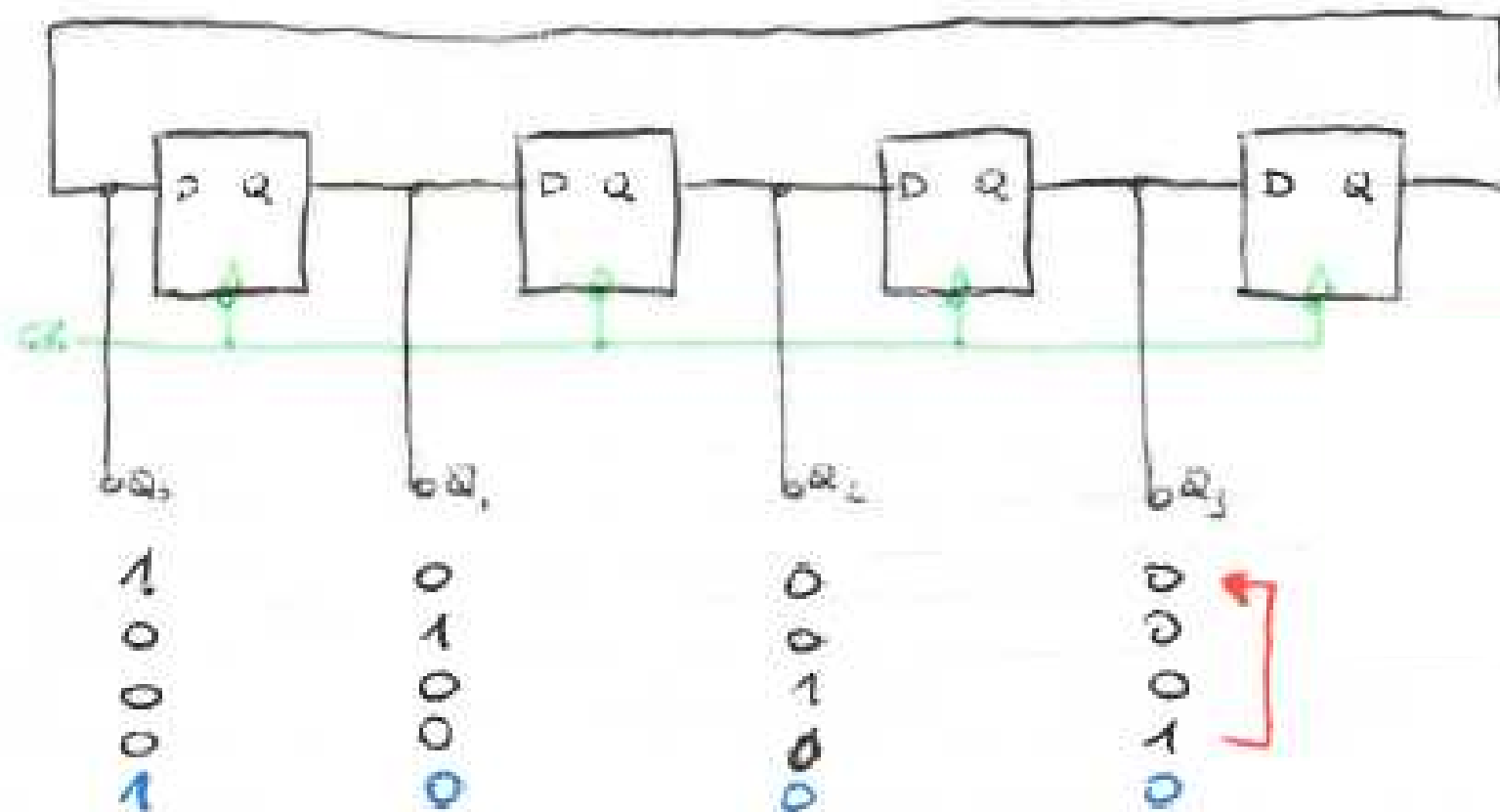
Serial Carry Counter



Parallel Carry Counter



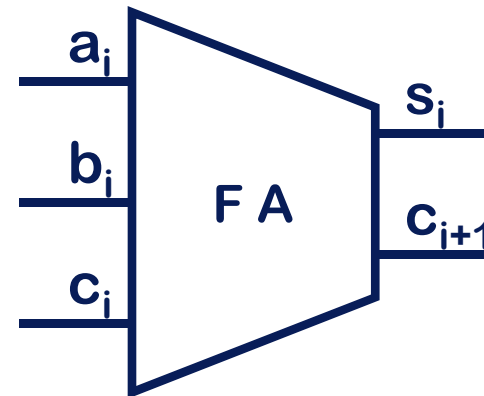
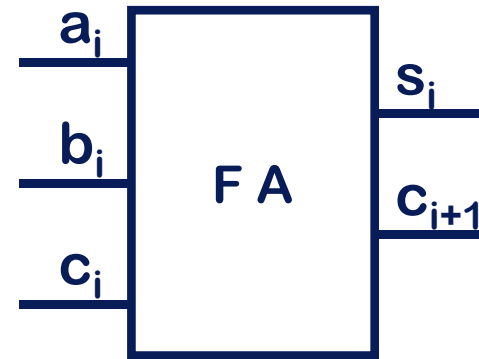
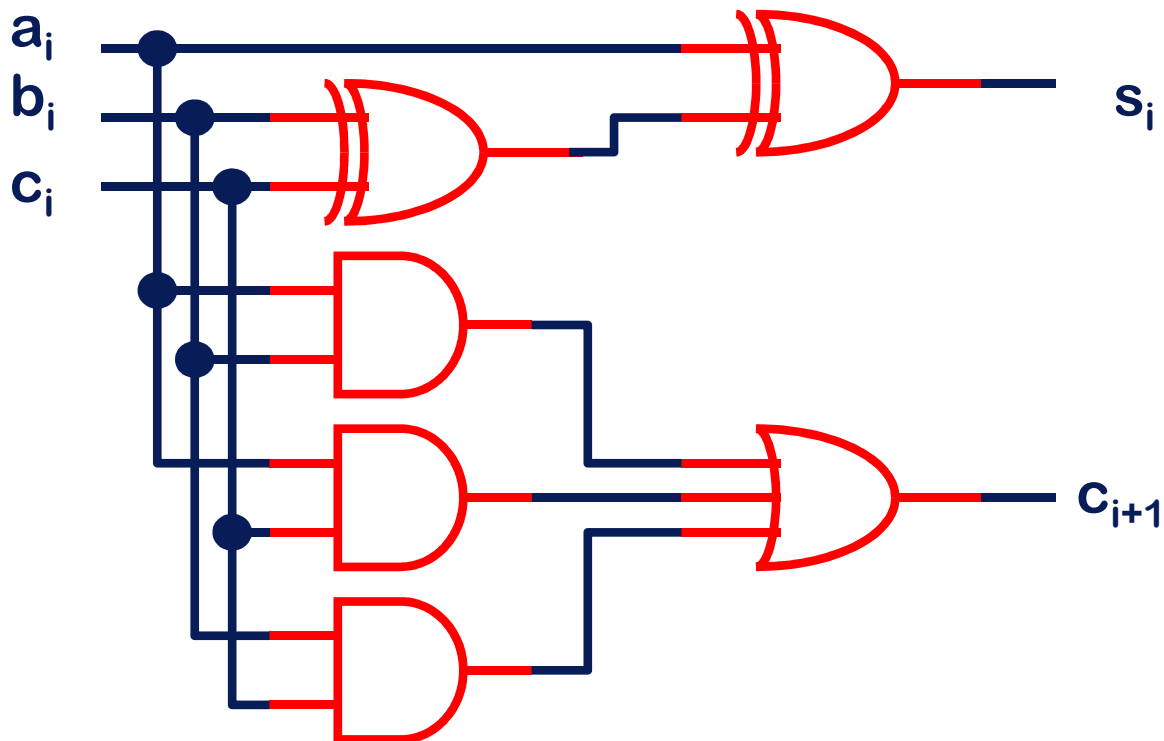
Ring Counter



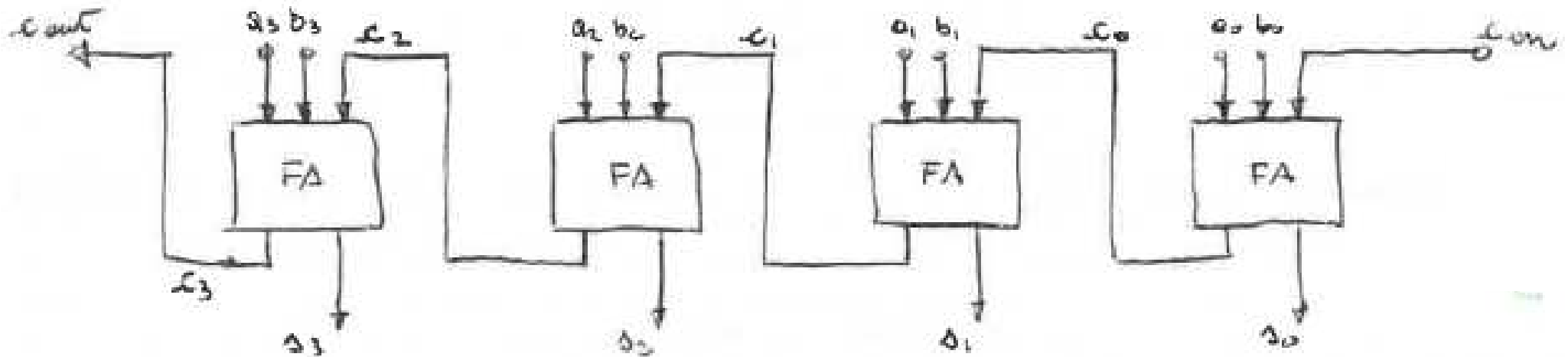
Full Adder (richiamo)

$$s_i = a_i \oplus b_i \oplus c_i = a_i \oplus (b_i \oplus c_i)$$

$$c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$$



Sommatore Parallelo con riporto seriale



$$\tau_{cout} = N \cdot \tau_C$$

$$\tau_{tot} = (N - 1) \cdot \tau_C + \tau_{FA}$$

N = numero di bit del sommatore

τ_C = ritardo per generare il carry

τ_{FA} = ritardo del Full-Adder singolo

τ_{tot} = ritardo totale del sommatore

Carry Look-Ahead Adder

$$\underline{S_i = a_i \oplus b_i \oplus C_{i-1}}$$

$$\underline{C_i = a_i b_i + a_i C_{i-1} + b_i C_{i-1} = a_i b_i + C_{i-1} (a_i + b_i) = a_i b_i + C_{i-1} (a_i \oplus b_i)}$$

$a_i + b_i$ si può sostituire col suo xor perché nell'espressione quando $a_i = 1$ e $b_i = 1$ anche $a_i \cdot b_i = 1$

$$\left[\begin{array}{l} G_i = a_i b_i \\ P_i = a_i \oplus b_i \end{array} \right] \Rightarrow \begin{array}{l} \underline{S_i = P_i \oplus C_{i-1}} \\ \underline{C_i = G_i + P_i \cdot C_{i-1}} \end{array}$$

Gi=generate
Pi=propagate

$$\underline{C_0 = G_0 + P_0 \cdot C_{in}}$$

$$\underline{C_1 = G_1 + P_1 C_0 = G_1 + G_0 P_1 + P_1 P_0 \cdot C_{in}}$$

$$\underline{C_2 = G_2 + P_2 C_1 = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_2 P_1 P_0 \cdot C_{in}}$$

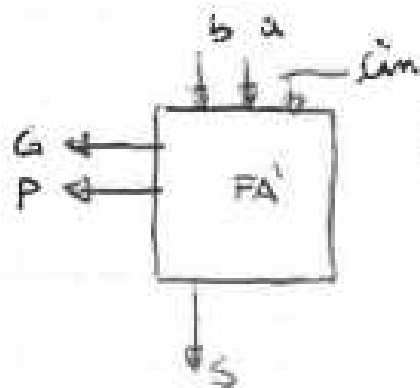
$$\underline{C_3 = G_3 + P_3 C_2 = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + P_3 P_2 P_1 P_0 C_{in}}$$

Carry Look-Ahead Adder (2)

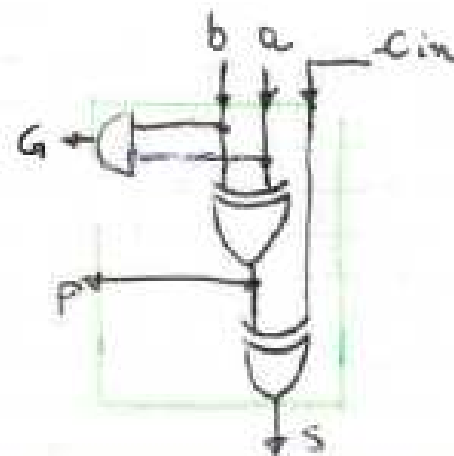
C_{i-1}	a_i	b_i	Δ_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

in questi due casi il carry in uscita è presente perché "si genera" a causa del fatto che nella somma $a_i + b_i$ sono tutti e due "a uno"

in questi due casi il carry in uscita è presente perché "si propaga" dal F.A. precedente !!

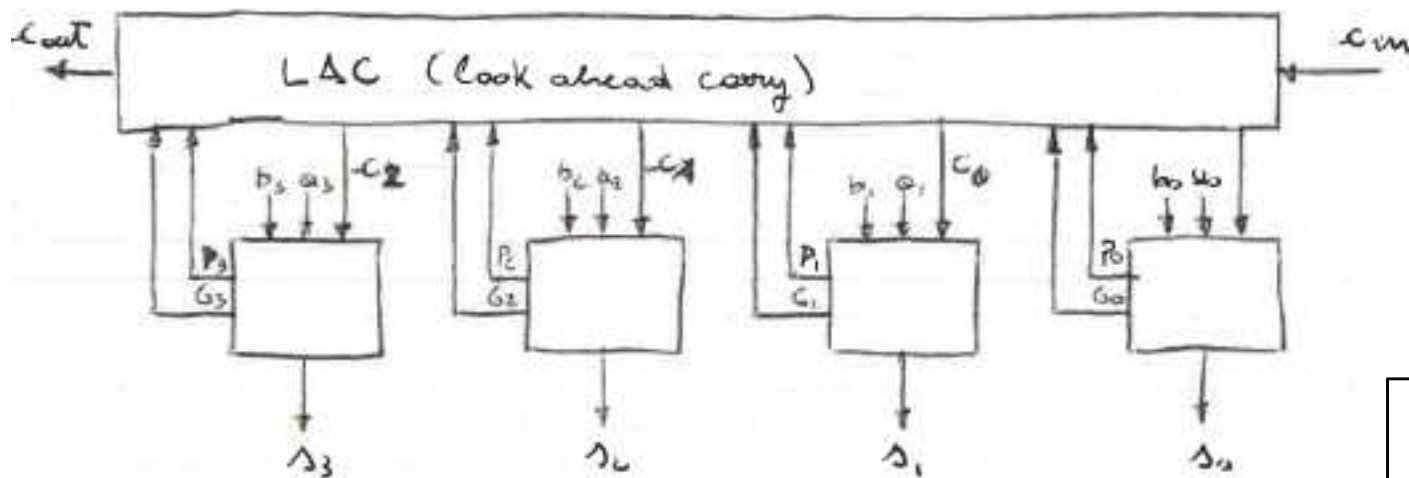
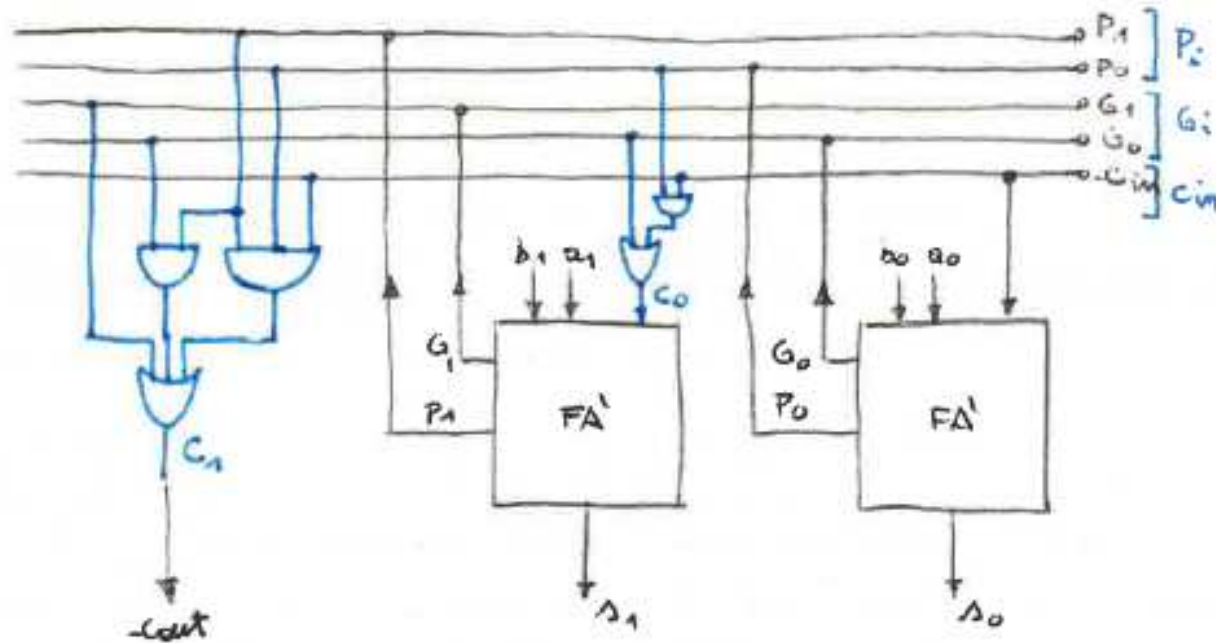


in cui la rete FA è



Carry Look-Ahead Adder (3)

Es. Nel caso pari a 2



$$\tau_{cout} = \tau_{LAC}$$

$$\tau_{tot} = \tau_{LAC} + \tau_{FA'}$$