



Architetture degli elaboratori - I

Introduzione



Prof. Alberto Borghese

Dipartimento di Informatica

Laboratorio di Sistemi Intelligenti Applicati (AIS-Lab)

alberto.borghese@unimi.it

Università degli Studi di Milano

Patterson & Hennessy: Section 1.12 on the WEB

A.A. 2023-2024

1/70

<http://borghese.di.unimi.it/>



Sommario della lezione



- Informazioni su corso ed esame
- Architettura dell'elaboratore
- Ciclo di esecuzione di un'istruzione
- Storia dell'elaboratore.

A.A. 2023-2024

2/70

<http://borghese.di.unimi.it/>



Obiettivo del corso



- Fornire i fondamenti per capire cosa succede dentro ad un elaboratore.
- Quali sono le problematiche e come viene elaborata l'informazione?
- Qual'è il linguaggio di un elaboratore (ISA)? Come funziona? (programmazione in piccolo).
- **Sviluppo della capacità di analisi e progettazione (sintesi).**

A.A. 2023-2024

3/70

<http://borghese.di.unimi.it/>



Architettura base del corso: MIPS R3000



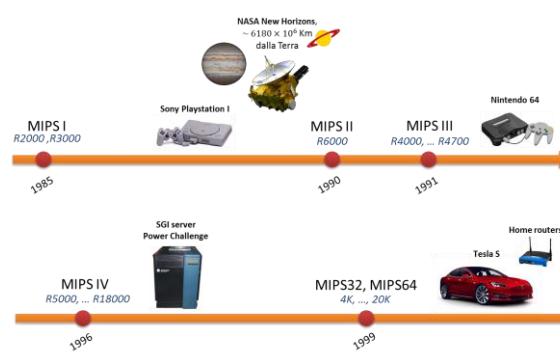
► Features-front



MIPS 7000, ARMv8, sistemi embedded che montano Windows CE, PlayStation 2, router, gateway...



Samsung S21



Exynos ARM Processor



A.A. 2023-2024

4/70

<http://borghese.di.unimi.it/>



MIPS



E' un'architettura semplice ma potente. La semplicità dell'architettura emerge anche a livello di Assembler (Architettura II).

"Hello world" in Assembler x86 (SO Linux)

```
.file "hello_world.c"
.section .rodata
.LC0:
.string "Hello world!"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
.pushq %bp
.cfi_offset %bp, -16
.cfi_offset $16, -16
.moving %rsp, %rbp
.cfi_offset %register 6
movl $.LC0, %edi
call puts
movl $0, %eax
popq %rbp
.cfi_offset %cfa 7, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609"
.section .note.GNU-stack,"",@progbits|
```

"Hello world" in MIPS (32 bit)

```
.data
hello: .asciiz "\nHello, World!\n"

.text
.globl main
main:
    li $v0, 4
    la $a0, hello
    syscall

    li $v0, 10
    syscall
```



Capire l'HW per scrivere SW efficace



Architettura I (dalle porte logiche alla CPU)

- Implementazione HW delle operazioni
- Implementazione HW delle sequenze di operazioni
- Porte logiche ed algebra di Boole
- Circuiti combinatori
- Circuiti sequenziali
- Macchine a stati finiti
- Firmware e micro-programmi
- CPU di base

Architettura II

- CPU avanzate
- Gestione delle gerarchie di memoria e memoria virtuale
- Parallelizzazione dell'esecuzione
- Gestione dell'I/O
- Architetture avanzate (GPU, DSA)



Architettura I - 6 CFU (Cognomi A-G) (Cognomi H-Z Prof. Nicola Basilico)


Sito principale:

http://borghese.di.unimi.it/Teaching/Architettura_I/_Arch_I.html

Programma:

http://borghese.di.unimi.it/Teaching/Architettura_I /Programma_2023-2024.html

Materiale sensibile su ARIEL.

Let's try to keep the course interactive

Orario turno I Prof. Borghese:

Lunedì, lezione, ore 8.30-10.30, aula G11
Giovedì, lezione, ore 8.30-10.30, aula G11

Strumento principale di contatto: email (alberto.borghese@unimi.it)

Ricevimento su appuntamento

A.A. 2023-2024

7/70

<http://borghese.di.unimi.it/>



Il sito del corso



Architettura I 2023-2024 AIS-lab

Turno I: cognomi A-G – Prof. **Alberto Borghese**
(Il turno 2, cognomi H-Z, verrà erogato dal Prof. **Nicola Basilico**)

Laboratorio: Dott.ssa Gabriella Trucco (Turno A), Dott. Massimo Rivolta (Turno B), Dott. Matteo Re (Turno C)

Corso di laurea triennale in Informatica, Università di Milano, A.A. 2023-2024, Primo Semestre.

Avviso: Il corso inizia il Lunedì 2 Ottobre 2023. La prova scritta dell'Appello del 20 settembre 2023 si terrà in Aula Chiusa, Dipartimento di Matematica, via Salmoiraghi 50, con inizio alle ore 8.30.

Orario del corso:
Lunedì, lezione, ore 8.30-10.30, aula G11
Giovedì, lezione, ore 8.30-10.30, aula G11

Orario del laboratorio:
Turno B – Guardare la pagina del laboratorio
Turno C – Guardare la pagina del laboratorio
Per il laboratorio fare riferimento al sito ARIEL del corso.

Programma dettagliato A.A. 2023-2024.

Materiale bibliografico del corso: [Esempio](#)

Program and bibliographic material: [in english](#)

Per il programma, temi d'esame e materiale dei corsi degli anni precedenti click [here](#).

N.B. Il diritto a scaricare il materiale accessibile da questa pagina è riservato solamente agli studenti regolarmente iscritti al corso.
Notice: The right to download the material accessible from this page is granted only to the students regularly enrolled in the Bachelor University course.

Modalità d'esame: scritto + orale (per la parte di teoria). Le votazioni parziali hanno validità di **2 appelli, approssimativamente 6 mesi**. Per dettagli sulla modalità della parte di laboratorio, consultare il sito della Dott.ssa Trucco

Tempi d'esame:	23 Gennaio 2023	22 Febbraio 2023	21 Giugno 2023	26 Luglio 2023	20 Settembre 2023
	22 Gennaio 2022	24 Febbraio 2022	23 Giugno 2022	19 Luglio 2022	27 Settembre 2022
	21 Gennaio 2021	18 Febbraio 2021	22 Giugno 2021	20 Luglio 2021	22 Settembre 2021
ore 9.00	ore 9.00	ore 9.00	ore 9.00	ore 9.00	ore 9.00
	23 Gennaio 2020	20 Febbraio 2020	23 Giugno 2020	23 Luglio 2020	22 Settembre 2020
	23 Gennaio 2019	21 Febbraio 2019	20 Giugno 2019	22 Luglio 2019	19 Settembre 2019
	23 Gennaio 2018	28 Febbraio 2018	20 Giugno 2018	20 Luglio 2018	27 Settembre 2018
	23 Gennaio 2017	24 Febbraio 2017	15 Giugno 2017	20 Luglio 2017	24 Settembre 2017
	29 Gennaio 2016	24 Febbraio 2016	15 Giugno 2016	21 Luglio 2016	24 Settembre 2016
	29 Gennaio 2015	22 Febbraio 2015	18 Giugno 2015	21 Luglio 2015	21 Settembre 2015
	24 Gennaio 2014	27 Febbraio 2014	26 Giugno 2014	24 Luglio 2014	30 Settembre 2014
	06 Febbraio 2013	28 Febbraio 2013	22 Giugno 2013	22 Luglio 2013	26 Settembre 2013
	23 Gennaio 2012	28 Gennaio 2012	21 Giugno 2012	23 luglio 2012	24 settembre 2012

20°C Sollegggiato

Q Cerca

ITÀ

02/10/2023 15:06

A.A. 2023-2024

8/70

<http://borghese.di.unimi.it/>



Programma



Programma corso di Architettura x

Programma del corso di Architettura degli Elaboratori - parte I
Programma A.A. 2023-2024

N.B. Il diritto a scaricare il materiale accessibile da questa pagina è riservato esclusivamente agli studenti regolarmente iscritti al corso.
Notice: The right to download the material accessible from this page is granted only to the students regularly enrolled in the hereabove University course.

Le lezioni di esercitazione sono riportate in colore rosso, le lezioni di laboratorio in blu e le lezioni frontali in nero.

Le slide sono da considerare bozze avanzate fino al giorno della lezione. Le slide in versione definitiva, saranno disponibili sul sito il giorno dopo la lezione.

Data	Contenuto della lezione
02/10/2023	Introduzione. L'architettura di riferimento. Il ciclo di esecuzione di un'istruzione. Storia dell'Elaboratore. <i>[Prof. Borgese, ultima modifica 04/10/22].</i>
05/10/2023	Codifica dell'informazione. Operazioni su numeri binari. Le operazioni fondamentali: somma e sottrazione. Rappresentazione binaria dei numeri decimali. <i>[Esercizi. Prof. Borgese, ultima modifica 06/10/22].</i>
06/10/2023	Esercitazione sulla codifica binaria e sulle operazioni fondamentali. <i>[Codifica IEEE754 e codifica Brain Float di Google dei numeri in virgola mobile. Prof. Borgese, ultima modifica 10/10/22].</i>
Logica combinatoria	
10/10/2023	Laboratorio - Codifica dell'informazione numerica: notazione posizionale, cambio di base, somma e sottrazione, complemento a 2, overflow (2 ore).
12/10/2023	L'algebra combinatoria: variabili ed operatori. Implementazione circolare (parte logica). Dal circuito alla funzione Algebra Booleana. Le porte universali. <i>[Prof. Borgese, ultima modifica 17/10/22].</i>
13/10/2023	Dalla logica combinatoria alla logica sequenziale: la prima funzione Booleana. Circuito adder. <i>[Prof. Borgese, ultima modifica 26/10/22].</i>
17/10/2023	Laboratorio - Codifica dell'informazione numerica: espressione binaria dei numeri nei diversi sistemi valori (sottrazione) (2 ore).
18/10/2023	Implementazione circolare di funzioni logiche mediante PLA e ROM. Circuiti combinatori notevoli. <i>[Prof. Borgese, ultima modifica 24/10/22].</i>
Le unità aritmetico-logiche	
23/10/2023	Addizionatore. Anticipazione del rapporto. <i>[Prof. Borgese, ultima modifica 08/11/22].</i>
03/11/2023	Laboratorio - Introduzione a Logica: presentazione della piattaforma e realizzazione di semplici circuiti combinatori (manipolazioni algebriche) (2 ore).
07/11/2023	Moltiplicatori hardware. Progettazione di una ALU. I due stadi. <i>[Prof. Borgese, ultima modifica 03/11/22].</i>
08/11/2023	Comparazione e Overflow. Temporizzazione dei circuiti basodati suLatch. Circuiti sequenziali: 1 latch SC. <i>[Prof. Borgese, ultima modifica 08/11/22].</i>
10/11/2023	Laboratorio - SOP, POS (seconda forma canonica), commutatività, mappe di Karnaugh (3 ore)
La logica sequenziale	
02/11/2023	Latch sincrono e flip-flop. <i>[Prof. Borgese, ultima modifica 10/11/22].</i>
06/11/2023	Macchine a stati finiti. Dalle specifiche al progetto. State Transition Graph. State Transition Table. Codifica della STT. Sintesi del circuito. <i>[Prof. Borgese, ultima modifica 14/11/22].</i>
07/11/2023	Laboratorio - Circuiti combinatori: decoder, multilevel, sommatore Half Adder e Full Adder (2 ore).
09/11/2023	Esercitazione sulle macchine a stati finiti e di segnale sulla prima parte del corso. <i>[Prof. Borgese, ultima modifica 22/11/22].</i>
13/11/2023	Lezione sospesa per prova in rete di matematica.
14/11/2023	Laboratorio - Circuiti combinatori: addizionatore a 4 bit, circuito somma e differenza, rilevamento overflow (2 ore).
15/11/2023	Laboratorio - Circuiti combinatori: Moltiplicazione, ALU (3 ore).
23/11/2023	Prima prova in rete: fino alla lezione 10 (Macchine a Stati Finiti). Ore XXXXX in aula XXXXX. Per partecipare al collegio occorre licenziarsi sul SIEA. Testo Versione 1. Testo versione 2. Risultati.
Il firmware	
27/11/2023	Introduzione al firmware. Circuiti firmware della moltiplicazione intera. <i>[Prof. Borgese, ultima modifica 28/11/22].</i>

- Le slide sono solo una traccia, occorre capire in profondità
- Gli argomenti sono collegati gli uni agli altri per tutte e due i corsi.

A.A. 2023-2024

9/70

<http://borghese.di.unimi.it/>



Materiale didattico

http://borghese.di.unimi.it/Teaching/Architettura_I/References.rtf

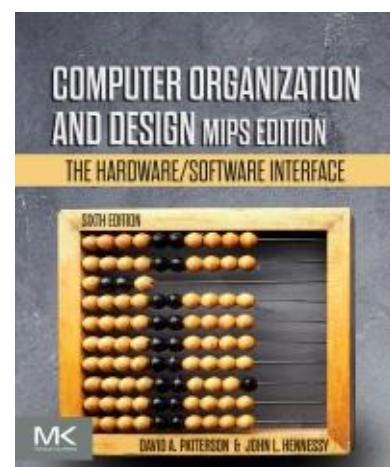


Testo di base (è disponibile sia in inglese che in italiano):

Struttura e progetto dei calcolatori:
l'interfaccia hardware-software,
D.A. Patterson and J.L. Hennessy, Quinta edizione, Zanichelli, gennaio 2022. [Edizione MIPS](#) (Nota: la quinta edizione Zanichelli è la traduzione della sesta edizione inglese).

"Computer Organization & Design: The Hardware/Software Interface",
D.A. Patterson and J.L. Hennessy, Morgan Kaufmann Publishers, Sixth Edition, 2020 ([MIPS edition](#)).

Il testo copre il contenuto dei corsi di Architettura I e II. Parte del materiale (appendici) si trova su WEB sul sito dell'editore (sia per la versione italiana che per la versione inglese).



<http://borghese.di.unimi.it/>



Materiale didattico

http://borghese.di.unimi.it/Teaching/Architettura_I/References.rtf

Per un approfondimento sui circuiti combinatori e sequenziali:

"Progettazione digitale" F. Fummi,
M.G. Sami, C. Silvano, McGrawHill.
Terza edizione, 2023.



A.A. 2023-2024

11/70

<http://borghese.di.unimi.it/>



Non solo teoria



«learn by doing» is equally important → laboratorio

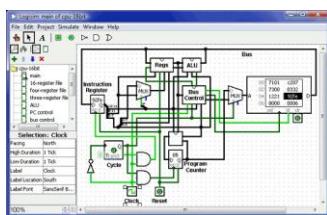
Dott.ssa Gabriella Trucco (A-FAS)
Gabriella.trucco@unimi.it



Dott. Massimo Rivolta (FAT-MOR)
Massimo.rivolta@unimi.it



Dott. Matteo Re (MOS-Z)
Matteo.Re@unimi.it



Simulatore di circuiti digitali Logisim:
<http://ozark.hendrix.edu/~burch/logisim/>

12/70

<http://borghese.di.unimi.it/>



Modalità di esame



Parte teorica (2/3 del voto). Riferimento: Prof. Borghese.

Prova scritta + orale

- 2 Appelli a Gennaio / Febbraio
- 2 Appelli a Giugno / Luglio
- 1 Appello a Settembre

In alternativa:

2 prove in itinere (compitini) durante l'anno. I compitini sostituiscono interamente scritto e orale. L'orale con i compitini è facoltativo.

Laboratorio (1/3 del voto). Progetto di laboratorio in Logisim o prova scritta (verrà deciso nella fase iniziale del corso di laboratorio)



Studiare è



- Acquisire un quadro generale di un argomento
- Riflettere sui concetti
- Riflettere sulla relazione tra i concetti
- Sperimentare l'applicazione dei concetti
- Collegare un argomento con gli altri argomenti del corso e di altri corsi nonché con le conoscenze pregresse.
- lo studio è una **funzione attiva**, che richiede energia e impegno: occorre **pensare**.



Studiare non è



- Imparare a memoria il contenuto delle slide
- Imparare a memoria il contenuto del libro
- Leggere e ripetere
- L'energia e la fatica messa nello studio si traducono alla fine in tempo risparmiato e in una maggiore soddisfazione personale: **si impara.**



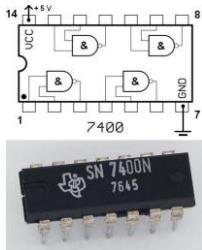
Sommario della lezione



- Informazioni su corso ed esame
- **Architettura dell'elaboratore**
- Ciclo di esecuzione di un'istruzione
- Storia dell'elaboratore.



Contenuto del corso



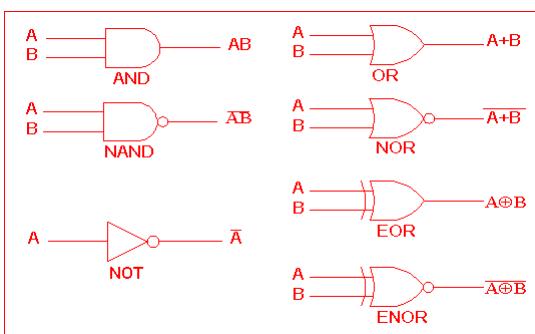
circuito integrato =
integra in se porte
logiche

From logic
gates to
.....
multi-core and
GPUs



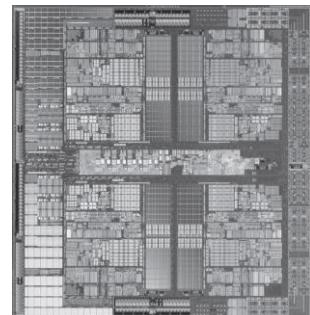
Data stream
Newsletter
Recension
Case Stud
Flussi RSC

CONFRONTO



A.A. 2023-2024

17/70



<http://borghese.di.unimi.it/>

dalle porte logiche ai multicore



Le architetture



tutte le case
hanno finestre,
porte, muri...



La casa



A.A. 2023-2024

18/70

<http://borghese.di.unimi.it/>



Le architetture



dispositivi
diversissimi ma
al loro interno
hanno
architetture
simili

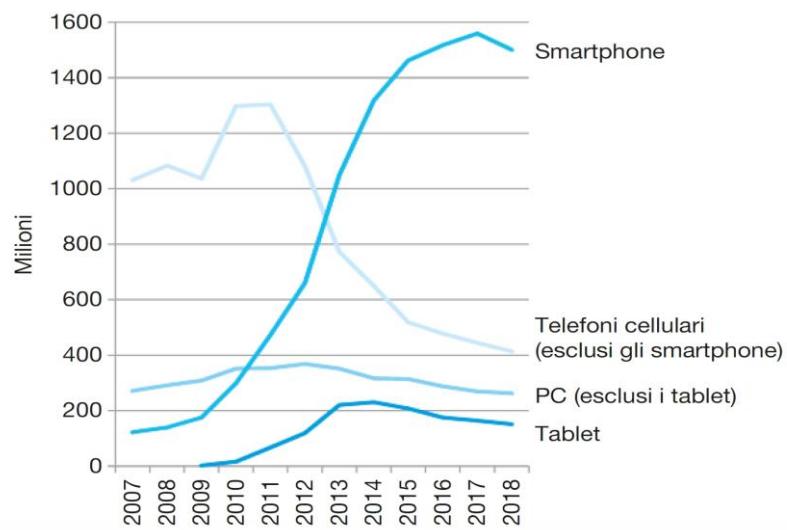


A.A. 2023-2024

19/70



I calcolatori nel mondo: verso l'era PostPC



A.A. 2023-2024

20/70

<http://borgheze.di.unimi.it/>

quanto velocemente evolvono i calcolatori?



La legge di Moore

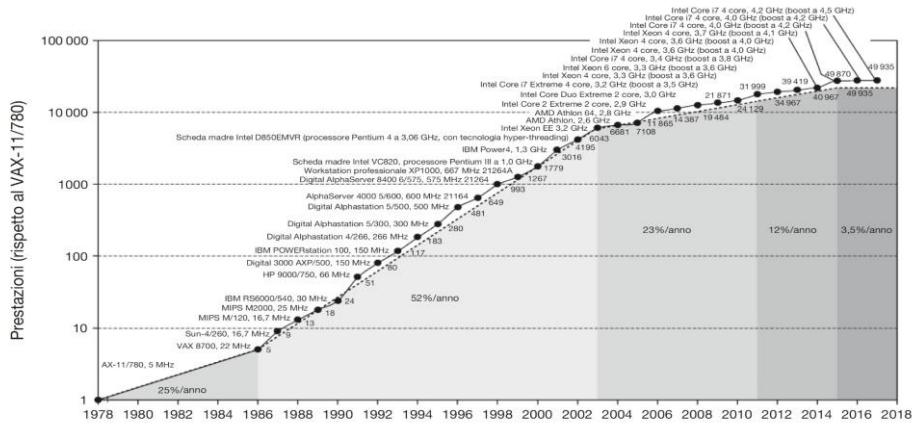


Figura 1.17 Crescita nelle prestazioni dei processori a partire dalla metà degli anni Ottanta. Questo grafico riporta le prestazioni relative al VAX11/780, misurate attraverso i benchmark SPECint (vedi il paragrafo 1.10). Prima della metà degli anni Ottanta, l'aumento delle prestazioni dei processori era dovuto principalmente alla tecnologia, ed era dell'ordine del 25% all'anno. L'aumento delle prestazioni nel periodo seguente è stato di circa il 52% all'anno, grazie a nuove idee nella progettazione delle architetture e nell'organizzazione dei calcolatori. Questo ha portato a un aumento delle prestazioni che nel 2002 è stato di sette volte l'aumento che si sarebbe verificato con un aumento di prestazioni del 25% all'anno. A partire dall'anno 2002 il limite sulla potenza assorbita, il parallelismo implicito delle istruzioni e la latenza della memoria hanno rallentato l'aumento delle prestazioni delle architetture monoprocessoressi a un 3,5% all'anno. (Da J.L. Hennessy, D.A. Patterson. *Computer Architecture: A Quantitative Approach*, Ed. 6. Waltham, MA: Elsevier, 2017).

In circa 18 mesi raddoppiano le prestazioni ed il numero di transistor e raddoppiano le capacità delle memorie (DRAM). **Legge di Moore.** Non vale più!! => Domain Specific Architectures
La velocità di accesso alla memoria cresce molto più lentamente.

raddoppio prestazioni ogni 18 mesi
velocità CPU cresce veloce e la valocità della memoria non riesce a starci dietro



Determinanti della legge di Moore



Il primo circuito integrato nel 1961 conteneva 4 (**quattro!**) transistor. Nel 1965 erano già 64 transistor e nel 1975 erano 32.000. In un Core i7 del 2012 si trovano **1,4 miliardi** di transistor.



Nel 2014 sono stati prodotti 250×10^{18} transistor (250 miliardi di miliardi, 25 volte il numero di stelle della via lattea e 75 volte il numero di galassie dell'Universo conosciuto). Ogni secondo vengono prodotti 8 miliardi di transistor. Più transistor nel 2014 che fino al 2011.

Abbiamo incontrato la barriera dell'energia e siamo nell'era postPC. I programmi devono essere efficienti anche in senso energetico. Occorre che consumino poca energia => **Come possiamo aumentare il numero di transistor, consumare poca energia e aumentare le prestazioni?** Conoscere l'organizzazione dei calcolatori.

La legge di Moore riguardava il numero di transistor che possono essere impacchetti tale per cui il costo per transistor è minimo (c'è un guadagno di scala all'aumentare del numero di transistor fino ad un certo valore, ma oltre questo valore i difetti rendono la produzione meno vantaggiosa)

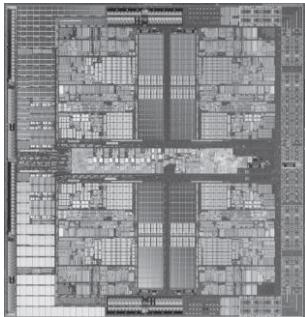
I fattori previsti da Moore erano:

- Aumento della dimensione dei chip (più transistor per chip)
- Diminuzione delle dimensioni (chip più piccoli, aumento del numero di chip, integrazione di chip)
- «Device cleverness» (multi-core)

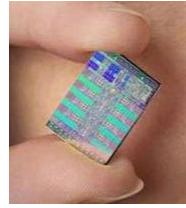
Da IEEE Spectrum, April 2015



Architetture recenti



AMD Barcelona
(quad-core)



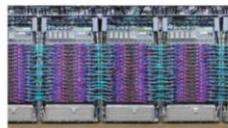
Cell processor
(IBM, Sony, Toshiba)
Playstation 3



Cloud TPU v3
420 teraflops
128 GB HBM



nVidia 9800 GTX,
Streaming processors
128 core



Cloud TPU v3 Pod
100+ petaflops
32 TB HBM
2-D toroidal mesh network



Obiettivo di un'architettura



Elabora in modo adeguato un input per produrre l'output.



- Le unità di *ingresso* (tastiera, mouse, rete, interfacce con dispositivi di acquisizione, ecc.) permettono al calcolatore di acquisire informazioni dall'ambiente esterno.
- L'architettura di elaborazione.
- Le unità di *uscita* (terminale grafico, stampanti, rete, ecc.) consentono al calcolatore di comunicare i risultati ottenuti dall'elaborazione all'ambiente esterno.



Cosa fa un elaboratore?



- Algoritmi (sequenza di istruzioni). **sono fatti da calcoli e operazioni logiche (scelta sul cosa fare dopo)**
Calcoli (calcolatore).
- **Operazioni logiche** (elaboratore).

- Programma (Ada Byron Lovelace, 1830) = *Algoritmi in Software.*



Come lo fa? Hardware.

Input ==> Elaborazione ==> Output

- Terza rivoluzione della nostra civiltà: la rivoluzione agricola, la rivoluzione industriale e la **rivoluzione dell'informatica**.



Operazioni elementari e codifica dell'informazione



Operazioni elementari necessarie ad eseguire algoritmi:

Calcolo (somma, sottrazione, prodotto....)

Controllo del flusso (if, for....)

L'informazione viene rappresentata utilizzando solamente due simboli (base 2: 0,1 -> acceso, spento). Ogni elemento (cifra) può assumere solo due valori: bit (binary digit)

I calcoli ed i controlli sono eseguiti utilizzando **esclusivamente!** le 3 **operazioni** fondamentali della **logica classica**: **AND, OR, NOT.**



I principi delle Architetture



Turing: "Universal Turing machine" (1936). Macchina di esecuzione di algoritmi universale. ha lavorato sul progetto Enigma

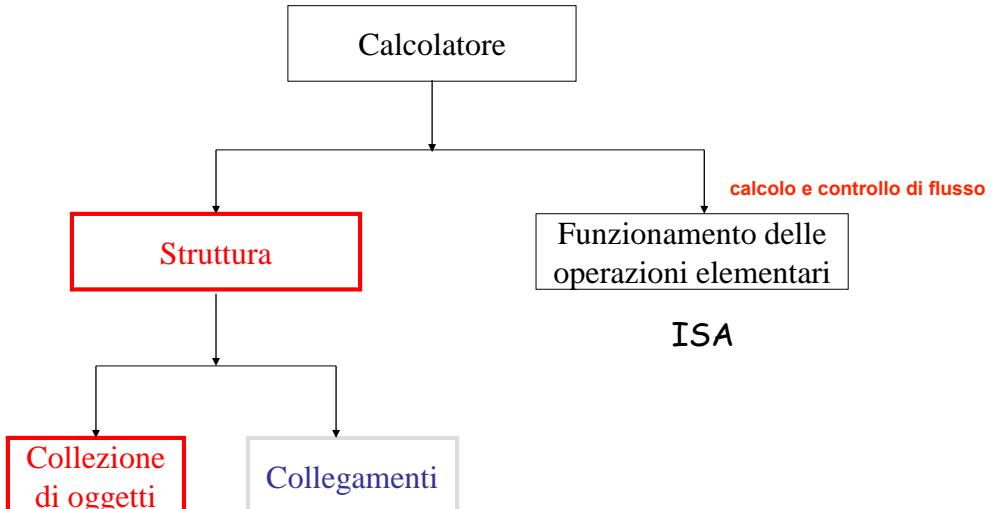
principi detti da Turing

I principi come sono stati codificati da Von Neumann negli anni 40.

- Dato che il dispositivo è essenzialmente una macchina di calcolo, ci sarà un'unità che è devota essenzialmente ai calcoli (ALU). negli anni 30 calcolatori elettromeccanici
- I dati e le istruzioni sono memorizzate separatamente in una memoria read/write
- Ci sarà una parte che gestisce tutto il sistema di elaborazione: trasferimento dei dati, comanda le operazioni, comanda I/O. Livello gerarchico superiore: UC unità di controllo
- Un computer deve essere collegato all'esterno. Occorre quindi un equipaggiamento per l'I/O.
- Il contenuto della memoria può essere recuperato in base alla sua posizione (indirizzo), e non è funzione del tipo di dato.
- L'esecuzione procede sequenzialmente da un'istruzione alla seguente (algoritmo, sequenza di passi....). Nelle architetture più avanzate l'esecuzione procede sequenzialmente per gruppi di istruzioni.



Descrizione di un elaboratore

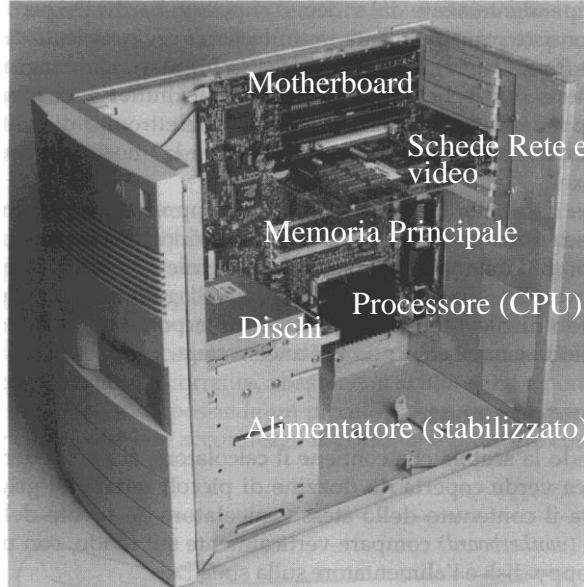




Struttura dell'elaboratore



architettura vecchia



A.A. 2023-2024

29/70

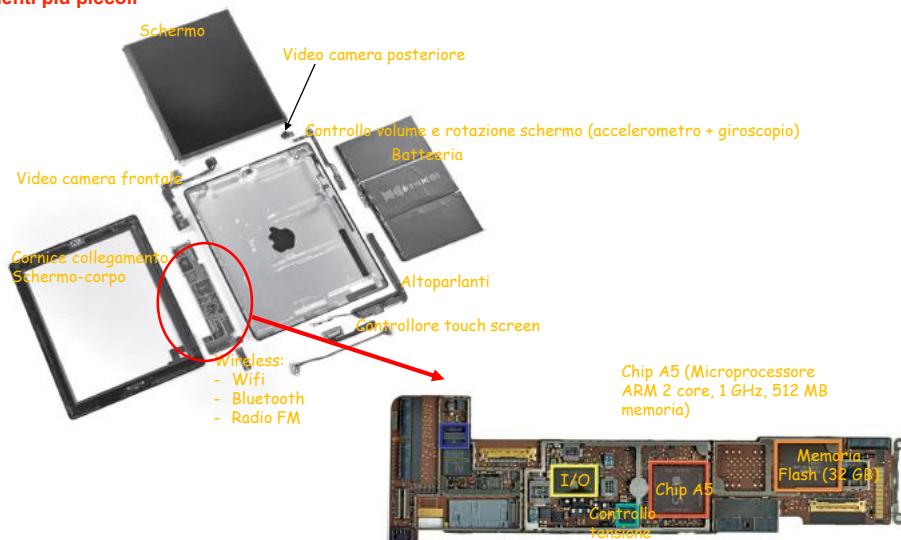
<http://borghese.di.unimi.it/>



Struttura di un PMD - I-Pad2



architettura nuova: componenti più piccoli



A.A. 2023-2024

30/70

<http://borghese.di.unimi.it/>



Struttura dell'elaboratore: descrizione



- Elementi principali di un elaboratore:
 - Unità centrale di elaborazione (*Central Processing Unit - CPU*).
 - Memoria di lavoro o memoria principale (*Main Memory - MM*) e dischi. **mantengono memoria quando spengo**
- Sulla motherboard: collegamenti principali di un calcolatore:
 - Bus di sistema (dati, indirizzi, controllo)
 - Bus di I/O (USB, Firewire): interfacce per i dispositivi di *Input/Output - I/O*: memoria di massa (dischi magnetici o a stato solido, pen drive), rete, altri dispositivi.
- In alternative, collegamenti attraverso i bridge



Unità centrale di elaborazione (*Central Processing Unit - CPU*)



- La *CPU* provvede ad eseguire le istruzioni che costituiscono i diversi programmi elaborati dal calcolatore.
- Eseguire un'istruzione vuol dire **operare delle scelte, eseguire dei calcoli** a seconda dell'istruzione e dei dati a disposizione.fa i calcoli e opera delle scelte (if e ciclo for)

all'interno della CPU



Elementi principali della CPU



- **Banco di registri (Register File)** ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale. Il register file è evoluto in cache + registri.

puntatore all'indirizzo della memoria

- **Registro Program counter (PC)**. Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzioni;

il contenuto della cella di memoria puntata dal program counter

- **Registro Instruction Register (IR)**. Contiene l'istruzione in corso di esecuzione,

- **Unità per l'esecuzione delle operazioni aritmetico-logiche (Arithmetic Logic Unit - ALU)**. I dati forniti all'ALU provengono direttamente da registri interni alla CPU. Possono provenire anche dalla memoria, ma in questo caso devono essere prima trasferiti in registri interni alla CPU. Dipende dalle modalità di indirizzamento previste;

- **Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile (Floating Point Unit – FPU), sommatori ausiliari, ecc.;**

- **Unità di controllo**. Controlla il flusso e determina le operazioni di ciascun blocco.
unità di controllo sopra e unità di elaborazione sotto



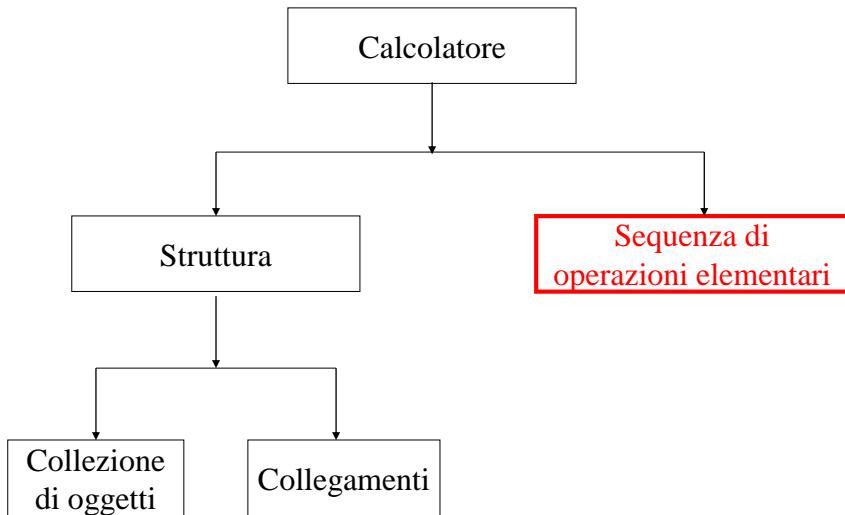
Sommario della lezione



- Informazioni su corso ed esame
- Architettura dell'elaboratore
- Ciclo di esecuzione di un'istruzione
- Storia dell'elaboratore.



Descrizione di un elaboratore



A.A. 2023-2024

35/70

<http://borghese.di.unimi.it/>



Ciclo di esecuzione di un'istruzione MIPS



A.A. 2023-2024

36/70

<http://borghese.di.unimi.it/>



Esempio di istruzione



Somma: 0x80000: addi \$s3, \$s2, 4
00100010010100011000000000000000100

Somma il contenuto del registro \$s2 con la costante 4 e scrivi il risultato nel registro \$s3



Lettura dell'istruzione (fetch)



- Istruzioni e dati risiedono nella memoria principale, dove sono stati caricati attraverso un'unità di ingresso.
 - L'esecuzione di un programma inizia quando il registro PC punta alla (contiene l'indirizzo della) prima istruzione del programma in memoria.
 - Il segnale di controllo per la lettura (READ) viene inviato alla memoria.
 - Trascorso il tempo necessario all'accesso in memoria, la parola indirizzata (in questo caso la prima istruzione del programma) viene letta dalla memoria e trasferita nel registro IR.
 - Il contenuto del PC viene incrementato in modo da puntare all'istruzione successiva.



Decodifica dell'istruzione



- L'istruzione contenuta nel registro IR viene decodificata per essere eseguita. Alla fase di decodifica corrisponde la predisposizione della CPU (apertura delle vie di comunicazione appropriate) all'esecuzione dell'istruzione.
- In questa fase vengono anche recuperati gli operandi. Nelle architetture MIPS gli operandi possono essere solamente nel Register File oppure letti dalla memoria.



Esecuzione



Viene selezionato il circuito / i circuiti appropriati per l'esecuzione delle operazioni previste dall'istruzione e determinate in fase di decodifica.

L'esecuzione può prevedere: calcolo, interazione con la memoria, controllo di flusso.



Scrittura in register file (write-back)



- Il risultato dell'operazione può essere memorizzato nei registri ad uso generale oppure in memoria.
- Non appena è terminato il ciclo di esecuzione dell'istruzione corrente (termina la fase di Write Back), si preleva l'istruzione successiva dalla memoria.



Esempio di istruzione di somma



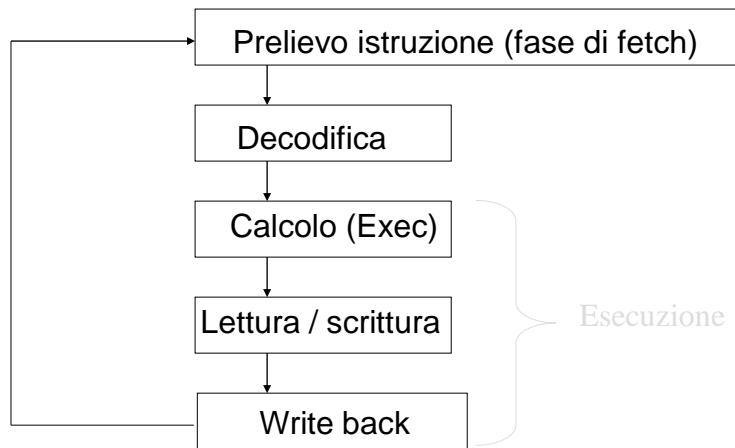
Somma: 0x80000: addi \$s3, \$s2, 4
 0x80000: 00100010010100011000000000000000100 alla macchina arriva stringa di 32 bit

Somma il contenuto del registro \$s2 con la costante 4 e scrivi il risultato nel registro \$s3

Fase di fetch: Caricamento dell'istruzione dall'indirizzo 0x80000.
Decodifica: Preparazione della CPU a svolgere una somma.
 Determinazione dei segnali di controllo.
 Lettura degli operandi (operando 1 nel registro \$s2,
 operando 2 è la costante 4).
Esecuzione: Esecuzione della somma.
Memoria: Nulla
Write-back: Scrittura del registro \$s3.



Ciclo di esecuzione di un'istruzione MIPS



A.A. 2023-2024

43/70

<http://borghese.di.unimi.it/>



Sommario della lezione



- Informazioni su corso ed esame
- Architettura dell'elaboratore
- Ciclo di esecuzione di un'istruzione
- **Storia dell'elaboratore.**

A.A. 2023-2024

44/70

<http://borghese.di.unimi.it/>



Storia dell'elaboratore



Filo conduttore:

Aumento della velocità di elaborazione

Diminuzione della dimensione dei componenti.

Aumento della capacità e velocità dell'I/O. considerare anche il consumo energetico

Adozione di tecnologie diverse (meccanica, elettrica, elettronica).



Storia del calcolatore (i primi passi)

- **Abaco**, Babilonesi, X secolo a.C. antico strumento di calcolo, utilizzato come ausilio per effettuare operazioni matematiche
- 1300 • B. Pascal (**Pascalina**, somma e sottrazione). macchina meccanica per somma sottrazione



- G. von **Leibnitz** (moltiplicazioni e divisioni come addizioni ripetute). stessa idea di pascal ma per moltiplicazioni e divisioni



Le calcolatrici



fine 1800

- Sviluppo di calcolatrici da tavolo meccaniche (diffusione nel commercio).



Millionaire, Steiger, 1892

Moltiplicazioni in un
“colpo di manovella”.



- Texas Instruments (1972) – prima calcolatrice tascabile.
1970 calcolatrici elettroniche con piccole possibilità di memoria

A.A. 2023-2024

47/70

<http://borghese.di.unimi.it/>



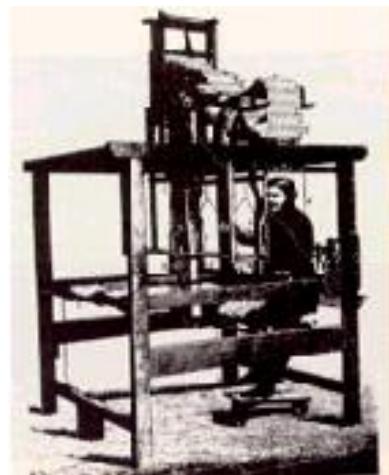
Un'architettura efficace



Una macchina per risolvere un problema industriale.

Telaio Jacquard (1801)

- Programma di lavoro su schede
- Macchina dedicata (antesignana delle macchine CAM).



A.A. 2023-2024

48/70

<http://borghese.di.unimi.it/>



Charles Babbage



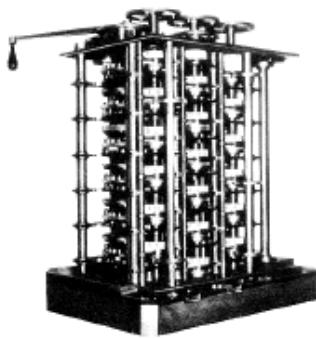
Le prime architetture furono pneumatiche

0 e 1 rappresentati dalle valvole aperte/chiuse

Charles Babbage

- Papà del calcolatore moderno.
- “Analytical Engine” i comandi erano a vapore!
- Utilizza il concetto di programma su (su schede) proposto da Ada Lovelace (1830).

esperimento fallito -> troppe perdite



Nasce l'IBM (1900-1930)



- Non solo architettura.....
- H. Hollerith: **Schede perforate a lettura elettromeccanica** (relais) combinato con il «Millionaire» **utilizzata inizialmente per censimento americano**

Meccanismo più semplice di gestione del controllo.

*Nel 1890, 46,804 macchine censirono 62,979,766 persone in pochi giorni.
Il censimento precedente, del 1870, durò 7 anni!!*

- T.J. Watson rilevò il brevetto e fondò l' IBM fondendo la società di Hollerith con altre piccole società (1932).

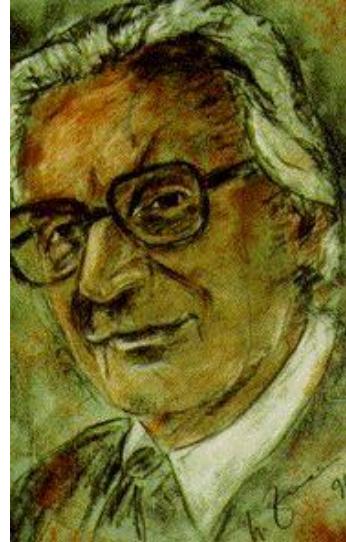


Il papà non riconosciuto



Konrad Zuse, 1936
Ingegnere civile.

Z1 -> 1938
Z3 -> 1941



Auto-ritratto del 1994

A.A. 2023-2024

51/70

<http://borghese.di.unimi.it/>



Storia dell'elaboratore - Mark I - 1944

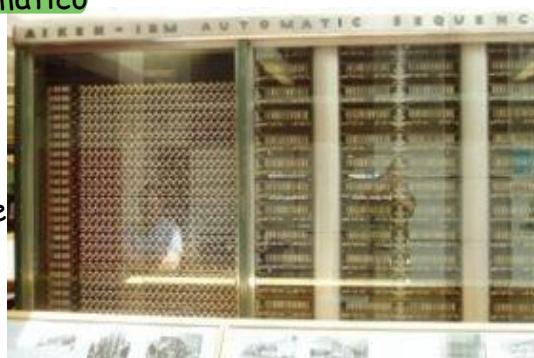


Primo computer automatico

Elettromeccanico

molto lento 15.3s per divisione
6s per moltiplicazione

OpCode + operandi



Automatic Sequence Controlled Calculator - H. Aiken, IBM

A.A. 2023-2024

52/70

<http://borghese.di.unimi.it/>



Storia dell'elaboratore (IIa Guerra mondiale)



- ABC - Atanasoff Berry Computer (University of Iowa).

Ampio utilizzo di elettrovalvole.

Memoria rigenerativa (cancellabile e riscrivibile).

Non funzionò mai completamente

A.A. 2023-2024

53/70

<http://borghese.di.unimi.it/>



La prima generazione (ENIAC: 1946-1955)

Elettronica (valvole: diodo, triodo). Aumento di prestazioni di 1,000 volte.

mai stato brevettato

- **ENIAC** (Electronic Numerical Integrator And Calculator), University of Pennsylvania.

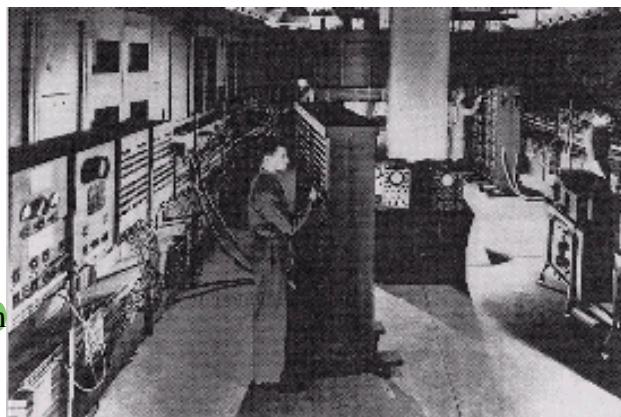
Caratteristiche:

- 20 registri da 10 cifre.
- 18,000 valvole.
- 70,000 resistenze.
- 10,000 condensatori.
- 6,000 interruttori.
- Dimensioni: 30mx2.5m

circa 14.000 lampadine

- Consumo: 140kW

- 100 operazioni/s.
- 30 tonnellate



- Il programma veniva realizzato cambiando manualmente il cabaggio.



Defining characteristics of five early digital computers



Computer	First operation	Place	Decimal /Binary	Electronic	Programmable	Turing complete
<u>Zuse Z3</u>	May 1941	<u>Germany</u>	binary	No	By punched film stock	Yes (1998)
<u>Atanasoff-Berry Computer</u>	Summer 1941	<u>USA</u>	binary	Yes	No	No
<u>Colossus</u>	December 1943 / January 1944	<u>UK</u>	binary	Yes	Partially, by rewiring	No
Harvard Mark I – IBM ASCC	1944	<u>USA</u>	decimal	No	By punched paper tape	Yes (1998)
<u>ENIAC</u>	1944	<u>USA</u>	decimal	Yes	Partially, by rewiring	Yes
	1948	<u>USA</u>	decimal	Yes	By Function Table <u>ROM</u>	Yes



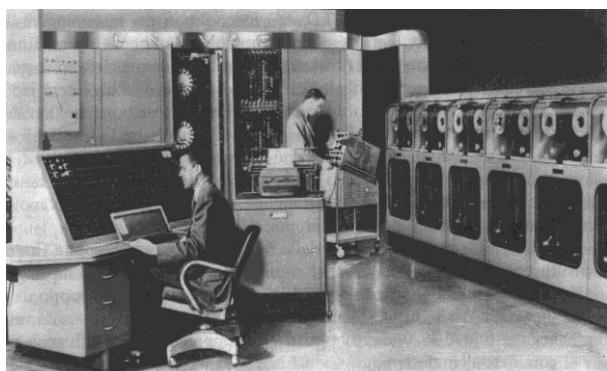
Eckbert & Mauchly



- **EDVAC**, Eckert, Mauchly, Von Neuman. Moore school, Pennsylvania University. **Programma memorizzato**.
- **EDSAC**, Eckert, Cambridge, 1949, (=> Mark I, 1948).

• **UNIVAC I**
 (Universal Automatic Computer) I (1951), Eckert e Mauchly.
 E' il primo calcolatore commercializzato.

48 esemplari a 1M\$





La seconda generazione (1952- 1963)



- Introduzione dell'elettronica allo stato solido.
- Introduzione delle memorie ferromagnetiche.

IBM:

- Modello 701 – 1953 per calcolo scientifico.
- Modello 702 – 1955 per applicazioni gestionali

anno 1960 • IBM704 - Memoria con nuclei di ferrite: 32,000 parole e velocità di commutazione di pochi microsecondi = qualche kHz).

- IBM709 nel 1958 - Introduzione del “canale” di I/O.
- IBM 7094 (1962) Introduzione della formalizzazione del controllo di flusso.
- **Introduzione del Fortran** (Formula Translator). **primo linguaggio alto livello**

CDC:

- CDC 6600 - Primo supercalcolatore. 1962.
- CDC 3600 - Multi-programmazione. 1963.

Digital equipment

- PDP - 1

A.A. 2023-2024

57/70

<http://homes.dsi.unimi.it/~borgheze>



La comunicazione tra i componenti



Switch centralizzato
(multiplexor -> bridge)

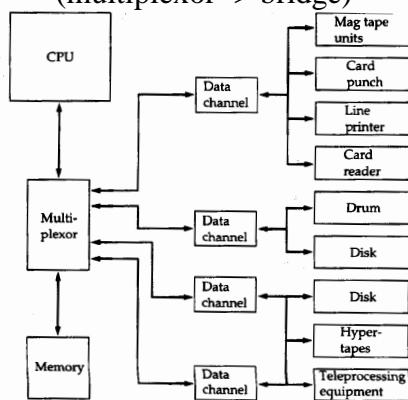
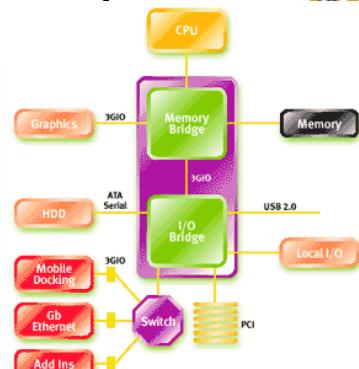
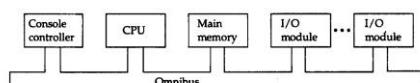


Figure 2.5 An IBM 7094 Configuration

Programma di “canale”



Architettura a nodo comune
(a bus, cf. bus PCI)



A.A. 2023-2024

58/70

Figure 2.9 PDP-8 Bus Structure



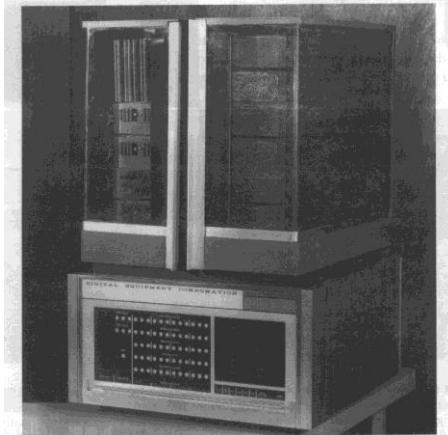
La terza generazione (1964-1971)



- Introduzione dei circuiti integrati (LSI).
- IBM360 (1964) - Prima famiglia di calcolatori (architettura di calcolatori). Costo 360,000\$
Registri a 32 bit.
Clock 1-4Mhz.

perchè al momento
del lancio era
coperto da una
minigonna

- Digital PDP-8 (1965) - Il primo minicalcolatore.
Costo < 20,000\$.
- PDP-11 (1970).



A.A. 2023-2024

59/70

<http://borghese.di.unimi.it/>



La quarta generazione (1971-1977)



- Cray I (1976) - Primo supercalcolatore. Vettoriale (cf. SIMD)
operazioni sui vettori



A.A. 2023-2024

60/70

<http://homes.dsi.unimi.it/~borghese>



La quarta generazione (1971-1977)



- **Introduzione del microprocessore** (VLSI).
- **Memorie a semiconduttori**.
- Intel 4004 (1971, F. Faggin) - 2,300 transistor. Sommatore a 4 bit. 16 registri a 4 bit + RAM + ROM -> Sistema MCS-4.
- Intel 8080 (1974) - 8bit su chip.

Xerox research laboratories & Steve Jobs

Primo Personal Computer:

MacIntosh II di Apple Computer (1977).

Sistema operativo a finestre:

Lisa (1984), MacIntosh II, 1985.

Processore Motorola.

Costo medio 2,000\$.



A.A. 2023-2024

61/70

<http://borghese.di.unimi.it/>



La quinta generazione: i PC (1978-2003)



Il primo PC (1981) IBM

Sistema operativo DOS (Microsoft di Bill Gates).

Processore Intel 8086.

Windows 1.0 nel 1987.

Coprocessore Matematico Intel 8087.

intel (windows) vince sul mercato:
mondo aperto rispetto a apple

PC come Workstation

Potenziamento della grafica. Coprocessore grafico (acceleratori).

Introduzione di elaborazione parallela (multi-threading) con esecuzione parzialmente sovrapposta (pipeline).

Processori RISC (Reduced Instruction Set Code).

MMU (Unità intelligenti per la gestione della memoria).

Definizione di GL -> OpenGL (Workstation Silicon Graphics)



SGI - Indigo2

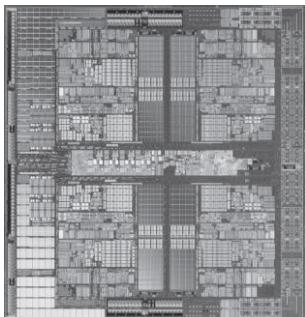
A.A. 2023-2024

62/70

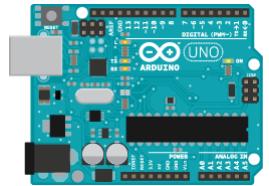
<http://borghese.di.unimi.it/>



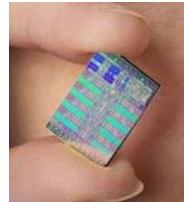
La sesta generazione (postPC)



AMD Barcelona
(quad-core)



Arduino microcontroller
(M. Banzi)



Cell processor
(IBM, Sony, Toshiba)
Playstation 3



nVidia 9800 GTX, Streaming
processors 128 core



A.A. 2023-2024

63/70

<http://borghese.di.unimi.it/>



Caratteristiche della sesta generazione



- Attualmente la frequenza di clock limite è 4Ghz: **barriera dell'energia**.
- **Rivoluzione del parallelismo:** la soluzione è quella di utilizzare diversi microprocessori (core) più piccoli e veloci.
 - Cell (IBM, Sony, Toshiba): 9-core microprocessors, 2006 (playstation 3, Sony).
 - Multi-core (Core2 Intel, AMD Barcelona...)
 - Schede grafiche di ATI e Nvidia (dal 2000) → CUDA programming language
 - Settembre 2006. Prototipo Intel con 80 processori on single chip.
Obiettivo è raggiungere 1,000,000 Mflops.
- **Come?**
 - Parallelizzazione del codice. (e.g. RapidMind Development Platform).
 - Nuovo modo di ragionare durante la programmazione software.
 - Tool di aiuto.
 - Parallelizzazione automatica del codice è ancora molto lontana.
 - Problema principale è la coerenza dei dati.

A.A. 2023-2024

64/70

<http://borghese.di.unimi.it/>



Il futuro



- Integrazione dei media.
- Wearable devices

- PC + telefono
- Wearable PC

Calcolatori ottici.
Calcolatori chimici.

- Co-processori on-board, specializzati per:
Ricerca in data-base.
Genomica.
Machine learning (it is a reality!)

Domain Specific Architecture

- Macchine intelligenti e sensibili.
- Sistemi multimediali.

A.A. 2023-2024

65/70

<http://borghese.di.unimi.it/>



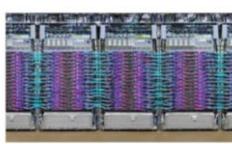
Il futuro



- Pervasive computing
- Dedicated architectures



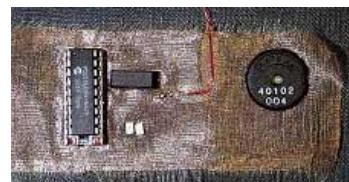
Cloud TPU v3
420 teraflops
128 GB HBM



Cloud TPU v3 Pod
100+ petaflops
32 TB HBM
2-D toroidal mesh network



E-textile



Circuito con CPU stampato
su stoffa



Smart watch 5

A.A. 2023-2024

66/70

<http://borghese.di.unimi.it/>



Classificazione dei calcolatori



- Centri di calcolo (Google, ...)
- Cluster (gruppi di calcolatori che lavorano per risolvere un problema complesso).
- Server (calcolatore in grado di eseguire un gran numero di processi in un'un'unità di tempo).
- Workstation
- Fissi (desktop)
- Portatili (laptop)
- Palmari.
- Smart phone: I-Phone, Blackberry...
- Microcontrollori (micro-architetture: Arduino, Raspberry PI,...)
- FPGA (architetture digitali programmabili)

A.A. 2023-2024

67/70

<http://borghese.di.unimi.it/>



Alcuni problemi



La velocità delle memorie non cresce con la velocità del processore.

Memorie gerarchiche – cache.
Aumento della parola di memoria.
high-speed bus (gerarchie di bus).

Tecniche di velocizzazione dell'elaborazione.

Predizione dei salti.
Scheduling ottimale delle istruzioni (analisi dei segmenti di codice).
Esecuzione speculativa.

Tecniche di I/O.

UDP.
Trasferimento in streaming (DMA).
Architetture dedicate alla grafica (GPU)

A.A. 2023-2024

68/70

<http://borghese.di.unimi.it/>



Caratteristiche comuni



Architettura di riferimento (Von Neuman)

Ciclo di esecuzione delle istruzioni

A.A. 2023-2024

69/70

<http://borghese.di.unimi.it/>



Sommario della lezione



- Informazioni su corso ed esame
- Architettura dell'elaboratore
- Ciclo di esecuzione di un'istruzione
- Storia dell'elaboratore.

A.A. 2023-2024

70/70

<http://borghese.di.unimi.it/>



Codifica dell'informazione

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimenti al testo: Paragrafi 2.4, 2.9, 3.1, 3.2, 3.5 (codifica IEEE754)



Sommario

Rappresentazione binaria dell'Informazione

Sistema di numerazione binario

Conversione in e da un numero binario

Operazioni elementari su numeri binari interi: somma, sottrazioni.

I numeri decimali



Il linguaggio



A.A. 2023-2024

3/51

<http://borghese.di.unimi.it/>

Per farsi capire da un calcolatore, occorre parlare la sua stessa lingua.

Il linguaggio è costituito da:

- **Semantica (significato delle parole)**
- **Sintassi,**



Rappresentazione dell'informazione



Le parole del linguaggio possono essere rappresentate mediante simboli. Questi simboli possono essere associati ai suoni e sono le lettere dell'**alfabeto**.

L'alfabeto italiano è costituito da 21 lettere: A, B, CZ.

Le stringhe di lettere dell'alfabeto rappresentano le parole, e.g. «t a v o l o» e sono associate alla semantica delle parole (a oggetti).

- **Diversi alfabeti possono essere usati per rappresentare gli stessi oggetti.**
- I simboli degli alfabeti possono assumere diverse forme: segni su carta, livelli di tensione, fori su carta, segnali di fumo.
- La scelta della rappresentazione è in genere vincolata al tipo di utilizzo ed al tipo di operazioni che devono essere fatte sulle informazioni stesse.

A.A. 2023-2024

4/51

<http://borghese.di.unimi.it/>



Alfabeto binario

è l'alfabeto usato
all'interno di un
calcolatore



L'alfabeto italiano è composto da 21 simboli: A, B, C Z.

L'alfabeto dei calcolatori è costituito da 2 simboli, che rappresentiamo come: 0, 1 (bit – binary digit).

Gruppi di 8 bit vengono chiamati Byte.

Un bit è l'unità di informazione base e può assumere due valori: – vero o falso – acceso o spento –

Rappresentazione binaria.

Il linguaggio di base mediante il quale ogni informazione deve essere codificata ha associato un alfabeto di 2 soli simboli (0 e 1)

Il testo viene scritto utilizzando solo 2 simboli.



Codifica dei caratteri alfanumerici



Consideriamo per il testo scritto un alfabeto costituito da:

- Caratteri alfabetici minuscoli / maiuscoli
 - Caratteri speciali: è, è, ò, à, ù
 - Caratteri di controllo (a capo, spaziatura di una linea, cancella carattere....)
 - Segni di interpunkzione: |, ?, ^....
- Come rappresentiamo tutti i simboli di questo alfabeto con solo 2 simboli?



0	32	64	96	^	128	Q	160	À	192	L	224	¤	
1	¤	33	†	65	À	97	à	129	Ù	161	Í	193	Ł
2	¤	34	”	66	B	98	b	130	É	162	Ó	194	T
3	♥	35	”	67	C	99	c	131	Ã	163	Ú	195	–
4	♦	36	”	68	D	100	d	132	Ã	164	Ñ	196	–
5	◊	37	”	69	E	101	e	133	Ã	165	ñ	197	+
6	♦	38	”	70	F	102	f	134	Ã	166	“	198	–
7	•	39	”	71	G	103	g	135	Ã	167	”	199	–
8	■	40	”	72	H	104	h	136	È	168	Œ	200	■
9	◊	41)	73	I	105	i	137	Œ	169	Œ	201	”
10	■	42	*	74	J	106	j	138	È	170	”	202	■
11	◊	43	+	75	K	107	k	139	Ý	171	Œ	203	”
12	◊	44	,	76	L	108	l	140	Ý	172	Œ	204	”
13	”	45	–	77	M	109	m	141	À	173	í	205	=
14	”	46	–	78	N	110	n	142	Ã	174	à	206	”
15	*	47	/	79	O	111	o	143	Ã	175	”	207	”
16	▶	48	Ø	80	P	112	p	144	È	176	”	208	”
17	◀	49	1	81	Q	113	q	145	Æ	177	”	209	”
18	‡	50	Z	82	R	114	r	146	fl	178	”	210	”
19	!!	51	3	83	S	115	s	147	Ô	179	”	211	”
20	¶	52	4	84	T	116	t	148	ö	180	”	212	”
21	§	53	5	85	U	117	u	149	Ã	181	”	213	”
22	–	54	6	86	U	118	u	150	Ã	182	”	214	”
23	‡	55	7	87	U	119	u	151	Ã	183	”	215	”
24	↑	56	8	88	X	120	x	152	Ù	184	”	216	”
25	↓	57	9	89	Y	121	y	153	Ã	185	”	217	”
26	→	58	:	90	Z	122	z	154	Ü	186	”	218	”
27	←	59	:	91	I	123	ı	155	ç	187	”	219	”
28	↳	60	<	92	”	124	ı	156	£	188	”	220	”
29	↶	61	=	93	J	125	յ	157	¥	189	”	221	”
30	▲	62	>	94	^	126	–	158	₹	190	”	222	”
31	▼	63	?	95	–	127	¤	159	ƒ	191	”	223	”
												255	



Il codice ASCII (American Standard Code for Information Interchange)

- 8 bit
- 0-31 codici di controllo.
- 32-127 standard ASCII
- 128-255 extended ASCII

- Metto in corrispondenza gruppi di 8 caratteri binary (bit) con ogni simbolo dell'alfabeto.

Prima versione è del 1963

A. Esempio: "Casa" sarà scritta come: 01000011 01100001 01110011 01100001 – 67 97 115 97



L'UNICODE



<http://www.unicode.org>. Codifica su 8, 16, 32 bit alfabeti diversi.

Latin	Malayalam	Tagbanwa	General Punctuation
Greek	Sinhala	Khmer	Spacing Modifier Letters
Cyrillic	Thai	Mongolian	Currency Symbols
Armenian	Lao	Limbu	Combining Diacritical Marks
Hebrew	Tibetan	Tai Le	Combining Marks for Symbols
Arabic	Myanmar	Kangxi Radicals	Superscripts and Subscripts
Syriac	Georgian	Hiragana	Number Forms
Thaana	Hangul Jamo	Katakana	Mathematical Operators
Devanagari	Ethiopic	Bopomofo	Mathematical Alphanumeric Symbols
Bengali	Cherokee	Kanbun	Braille Patterns
Gurmukhi	Unified Canadian Aboriginal Syllabics	Shavian	Optical Character Recognition
Gujarati	Ogham	Osmanya	Byzantine Musical Symbols
Orkney	Runic	Cypriot Syllabary	Musical Symbols
Tamili	Tagalog	Tai Xuan Jing Symbols	Arrows
Telugu	Hanunoo	Yijing Hexagram Symbols	Box Drawing
Kannada	Buhid	Aegean Numbers	Geometric Shapes



Sommario



Rappresentazione binaria dell'Informazione

Sistema di numerazione binario

Conversione in e da un numero binario

Operazioni elementari su numeri binari interi: somma, sottrazioni

I numeri decimali



Proprietà di potenze e logaritmi



$$2^K \times 2^M = 2^{(K+M)} \quad 2^{K^M} = 2^{K \cdot M} = 2^{M^K} \quad 2^{-K} = \frac{1}{2^K}$$
$$2^3 \times 2^2 = 2^{(3+2)} = 32$$

Il logaritmo è l'operazione inversa dell'elevamento a potenza:

- $N = 2^M \Leftrightarrow M = \log_2(N)$
- $32 = 2^5 \Leftrightarrow 5 = \log_2(32)$

Se ho M cifre, dove ciascuna cifra può assumere 2 valori, il numero totale di combinazioni sarà 2^M

$$\log_2(2^M) = M \quad \log_2 K = -\log_2\left(\frac{1}{K}\right)$$

$$\log_2 KM = \log_2 K + \log_2 M$$

$$5 = \log_2 (4 \times 8) = \log_2 4 + \log_2 8 = 2 + 3$$



Codifica binaria



Per poter rappresentare un numero maggiore di informazioni (rispetto a vero/falso) è necessario utilizzare sequenze di bit.

Il processo secondo cui si fa corrispondere a un'informazione una configurazione di bit prende il nome di **codifica dell'informazione**.

Codifica esplicita (e.g. lettere dell'alfabeto)

Codifica implicita.



Esempio di codifica binaria implicita



Dato un insieme di elementi, identifico i diversi elementi con un numero d'ordine.

0	000	A = tigre
1	001	B = cane
2	010	C = topo
3	011	D = elefante
4	100	E = gatto
5	101	F = cervo
6	110	G = gazzella
7	111	H = rinoceronte

Quanti bit servono per rappresentare un'informazione?

Quanti oggetti diversi possiamo rappresentare con parole binarie di k bit?

Con una parola di 1 bit rappresentiamo 2 oggetti (1 bit ha due possibili valori).

- Supponiamo di avere parole di k bit. Quanti oggetti riescono a rappresentare?



Esempio di codifica binaria esplicita



In decimale i caratteri utilizzati per rappresentare i numeri sono: 0, 1, 2, 3, 4,9. In binario si utilizzano sempre i simboli: 0,1.

Una stringa di cifre binarie rappresenta un numero.

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111



Osservazioni sulla codifica binaria



Il linguaggio di un elaboratore elettronico è fatto di due segnali: **on** e **off**, rappresentati dai simboli **1** e **0** (**alfabeto binario**).

- Sia le istruzioni che i dati sono rappresentati da **parole (word)** di numeri binari. Sequenze di bit trattate come un unicum nell'elaboratore.
- Un alfabeto binario non limita le funzionalità di un elaboratore a patto di avere parole di lunghezza sufficiente.

Una stringa binaria non ha significato semantico di per sé.

- 00000011 00101000 11010000 00100000 rappresenta 4 caratteri alfanumerici: (End_of_Text; ‘(; ‘D’; ‘ ‘).
- 00000011 00101000 11010000 00100000 rappresenta un'istruzione di addizione in MIPS su 32 bit (add \$k0, \$t0, \$t9).
- 0000001100101000110100000100000 rappresenta un numero intero: 53,006,368



Codifica negli elaboratori



Un elaboratore elabora stringhe di simboli binari. Il numero di elementi che più frequentemente circola nell'elaboratore si chiama **parola** (parole di 32 / 64 bit).

Queste stringhe di simboli possono essere associate sia a dati che a istruzioni.

Diverse elaborazioni di una stessa stringa producono risultati diversi:

- print su schermo (associa ai byte, delle matrici di pixel)
- La Unità di controllo associa alla parola i comandi e i dati delle istruzioni (configura la CPU).
- La ALU esegue la somma sulle parole che rappresentano i numeri.



Numerazione Simbolica



Sistema di **numerazione mediante simboli** (numerazione romana: I, V, X, L, C, M) **il cui valore non dipende dalla posizione**: e.g. XXXI = 31, XI = 11....

Sistema di numerazione posizionale (decimale): {cifra, peso}.

Il peso è la base elevata alla posizione della cifra.

1 ha un valore diverso in queste due scritture: **simboli che cambiano significato in base alla posizione**

100	1 centinaia
1000	1 migliaia

Dipende dalla **posizione** dell'1.



Numerazione Posizionale



Alfabetto della numerazione:

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} numerazione araba decimal.

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} numerazione esadecimale.

{0, 1} **numerazione binaria**.

Sistemi di numerazione binario, ottale ed esadecimale.

Conversioni decimale -> binario e viceversa.



Numeri



Matematica

Informatica

Numeri naturali

- Risoluzione 1 unità
 - Contano quantità tangibili
 - $0 \leq n < +\infty$
- unsigned

Numeri relativi

- Risoluzione 1 unità
 - Contano quantità tangibili e il loro complemento.
 - $-\infty < n < +\infty$
- int, integer

Numeri decimali

- Risoluzione dipendente dalla codifica
 - Numeri con la virgola
- real (ma non sono reali!)
float



Codifica posizionale di un numero intero



Fondata sul concetto di **base**: $B = [b_0, b_1, b_2, b_3, \dots]$.

Ciasun elemento, N , può essere rappresentato come combinazione lineare degli elementi della base ($k \geq 0$): $N = \sum_k c_k b_k = \sum_k c_k 10^k$

Esempi:

$$\text{base 10} \quad \bullet 764_{10} = 7 \times 10^2 + 6 \times 10^1 + 4 \times 10^0 \quad b_k = B^k = 10^k$$

$$\text{base 10} \quad \bullet 12_{10} = 1 \times 10^1 + 2 \times 10^0 \quad b_k = B^k = 10^k$$

$$\text{base 2} \quad \bullet 100_2 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 4_{10} \quad b_k = B^k = 2^k$$

$$k \geq 0$$



Codifica posizionale di un numero decimale



Fondata sul concetto di **base**: $B = [b_0, b_1, b_2, b_3, \dots]$.

Ciasun elemento, N , può essere rappresentato come combinazione lineare degli elementi della base: $N = \sum_k c_k b_k = \sum_k c_k 10^k$

Esempi:

$$\bullet 764,3_{10} = 7 \times 10^2 + 6 \times 10^1 + 4 \times 10^0 + 3 \times 10^{-1} \quad b_k = B^k = 10^k$$

$$\bullet 12,21_{10} = 1 \times 10^1 + 2 \times 10^0 + 2 \times 10^{-1} + 1 \times 10^{-2} \quad b_k = B^k = 10^k$$

$$\bullet 100,11_2 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 4,75_{10} \quad b_k = B^k = 2^k$$

$$-oo < k < +oo$$



Tassonomia ed unità di misura



Prefissi:

k - chilo (mille: 10^3).	m - milli (un millesimo: 10^{-3})
M - mega (un milione: 10^6).	μ - micro (un milionesimo: 10^{-6})
G - giga (un miliardo: 10^9).	n - nano (un miliardesimo: 10^{-9})
T - tera (1000 miliardi: 10^{12})	p - pico (un millesimo di miliardo: 10^{-12})
P - peta (1,000,000 miliardi: 10^{15})	f - femto (un millonesimo di miliardo: 10^{-15})

Hertz - numero di ciclo al secondo nei moti periodici (clock).

- MIPS - Milioni di istruzioni per secondo.
- MFLOPS - Milioni di istruzioni in virgola mobile (FLOating point) al secondo.



Terminologia



Bit = binary digit.

- 1 byte = 8 bit.
- 1kbyte = 2^{10} byte = 1,024 byte **k sta per 1000**
- 1Mbyte = 2^{20} byte = 1,048,576 byte. **M sta per Milione**
- 1Gbyte = 2^{30} byte = 1,073,741,824 byte.
- 1Tbyte = 2^{40} byte = 1,099,511,627,776 byte.

▫ **Parola** (word) numero di bit trattati come un unicum dall'elaboratore.

▫ Le parole oggi arrivano facilmente a **64 bit (8 Byte)**.



Approssimazione



Multipli del bit

Prefissi SI			Prefissi binari			
Nome	Simbolo	Multipli	Nome	Simbolo	Multipli	
kilobit	kbit	10^3	kibibit	Kibit	2^{10}	1 024 bit
megabit	Mbit	10^6	mebibit	Mibit	2^{20}	1 024 Kib
gigabit	Gbit	10^9	gibibit	Gibit	2^{30}	1 048 576 Kib = 1 gibibit
terabit	Tbit	10^{12}	tebibit	Tibit	2^{40}	1 024 Gbit
petabit	Pbit	10^{15}	pebibit	Pibit	2^{50}	1 024 Tbit
exabit	Ebit	10^{18}	exbibit	Eibit	2^{60}	1 024 Pbit
zettabit	Zbit	10^{21}	zebibit	Zibit	2^{70}	1 024 Ebit
yottabit	Ybit	10^{24}	yobibit	Yibit	2^{80}	1 024 Zbit

Base 10

Base 2



Sommario



Rappresentazione binaria dell'Informazione

Sistema di numerazione binario

Conversione in e da un numero binario

Operazioni elementari su numeri binari interi: somma, sottrazione

I numeri decimali



Conversione da base 2 a base 10



Un numero $N = [c_0, c_1, c_2, c_3, \dots]$ in base 10, $B = [b_0, b_1, b_2, b_3, \dots]$ si trasforma in base $R = [r_0, r_1, r_2, r_3, \dots]$, facendo riferimento alla formula:

$$N = \sum_k c_k b_k = \sum_{k=0}^{N-1} d_k r^k$$

- ciascuna cifra k -esima viene moltiplicata per la base corrispondente.
- i valori così ottenuti sono sommati per ottenere il numero in notazione decimale.

Base binaria:

$$\begin{aligned} 101\ 1101\ 0101_{\text{due}} &= 1 \times 2^{10} + 0 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + \\ &\quad 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = \\ &1024 + 256 + 128 + 64 + 16 + 4 + 1 = 1493_{\text{dieci}} = \\ &1 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 \end{aligned}$$

25/51

<http://borgheze.di.unimi.it/>

A.A. 2023-2024



“Spelling” di un numero



$$\begin{aligned} 1493 &= 3 \times 1 + 9 \times 10 + 4 \times 100 + 1 \times 1000 \\ &= 3 \times 10^0 + 9 \times 10^1 + 4 \times 10^2 + 1 \times 10^3 \end{aligned}$$

Vogliamo rappresentare 1493_{dieci}

Unità $1493 = 10 \times 149 + 3$ \leftarrow Cifra meno significativa

Decine $(10x) \quad 149 = 10 \times 14 + 9$

Centinaia $(100x) \quad 14 = 10 \times 1 + 4$

Migliaia $(1000x) \quad 1 = 10 \times 0 + 1$ \leftarrow Cifra più significativa

$$N = \sum_k c_k b_k = \sum_{k=0}^{N-1} d_k r^k$$

A.A. 2023-2024

26/51

<http://borgheze.di.unimi.it/>



“estrazione” delle cifre decimali



$$1493 = 3 \times 1 + 9 \times 10 + 4 \times 100 + 1 \times 1000 \\ = 3 \times 10^0 + 9 \times 10^1 + 4 \times 10^2 + 1 \times 10^3$$

Vogliamo estrarre le cifre di 1493_{dieci} . Porto le cifre alla destra della virgola:

$$1493 / 10 = 149,3$$

\rightarrow 149 decine \rightarrow 3 unità (resto)

quando divido per 10 il resto che
ottengo alla prima iterazione
corrisponde alle unità

$$N = \sum_k c_k b_k$$

$$1493/10 = 10^{-1} \times (3 \times 10^0 + 9 \times 10^1 + 4 \times 10^2 + 1 \times 10^3) = \\ (3 \times 10^{-1} + 9 \times 10^0 + 4 \times 10^1 + 1 \times 10^2)$$



“estrazione” delle cifre decimali



Vogliamo estrarre le cifre di 1493_{dieci} . Porto le cifre alla destra della virgola: $149 = 9 \times 10^0 + 4 \times 10^1 + 1 \times 10^2$ decine + 3 unità

$$1493 / 10 = 149,3$$

\rightarrow 149 decine \rightarrow 3 unità (resto)

$$14 = 4 \times 10^0 + 1 \times 10^1 \text{ centinaia}$$

$$149 / 10 = 14,9$$

\rightarrow 14 centinaia \rightarrow 9 decine (resto)

$$14 / 10 = 1,4$$

\rightarrow 1 migliaia \rightarrow 4 centinaia (resto)



“estrazione” delle cifre decimali



$$1493 = 3 \times 1 + 9 \times 10 + 4 \times 100 + 1 \times 1000 \\ = 3 \times 10^0 + 9 \times 10^1 + 4 \times 10^2 + 1 \times 10^3$$

Vogliamo estrarre le cifre di 1493_{dieci} . Porto le cifre alla destra della virgola:

$$\begin{aligned} 1493 / 10 &= 149,3 \\ 149 / 10 &= 14,9 \\ 14 / 10 &= 1,4 \\ 1 / 10 &= 0,1 \end{aligned}$$

→ esamino 149 → 3 unità
 → esamino 14 → 9 decine
 → esamino 1 → 4 centinaia
 → termina → 1 migliaia

Le cifre sono il resto della divisione ricorsiva del numero.

Termina quando non è rimasto nulla del numero.

A.A. 2023-2024

<http://borgheze.di.unimi.it/>



Conversione base 10 -> base 2

“estrazione” delle cifre binarie



Vogliamo rappresentare 1493_{dieci} in binario: $\underline{\underline{10111010101_{\text{due}}}}$

Resto

$$\begin{aligned} 1493 / 2 &= 746 \ R = 1 && \leftarrow \text{Bit meno significativo} \\ 746 / 2 &= 373 \ R = 0 \quad 373 * 2 + 0 = 746 && (\text{LSB} - \text{Least Significant Bit}) \\ 373 / 2 &= 186 \ R = 1 \quad 186 * 2 + 1 = 373 \Rightarrow (186 * 2 + 1) * 2 = 373 \\ 186 / 2 &= 93 \ R = 0 \\ 93 / 2 &= 46 \ R = 1 \\ 46 / 2 &= 23 \ R = 0 \\ 23 / 2 &= 11 \ R = 1 \\ 11 / 2 &= 5 \ R = 1 \\ 5 / 2 &= 2 \ R = 1 \\ 2 / 2 &= 1 \ R = 0 \\ 1 / 2 &= 0 \ R = 1 && \leftarrow \text{Bit più significativo} \\ &&& (\text{MSB} - \text{Most Significant Bit}) \end{aligned}$$

$$N = \sum_k c_k b_k$$

30/51

<http://borgheze.di.unimi.it/>

A.A. 2023-2024



Conversione base 10 -> base n: algoritmo



Un numero x in base 10 si trasforma in base n , *intera*, utilizzando il seguente procedimento:

- Dividere il numero x per n
- Il resto della divisione è la cifra di posto 0 in base n
- Il quoziente della divisione è a sua volta diviso per n
- Il resto ottenuto a questo passo è la cifra di posto 1 in base n
- Si prosegue con le divisioni dei quozienti ottenuti al passo precedente fino a che l'ultimo quoziente è 0.
- l'ultimo resto è la cifra più significativa in base n



Esercizi



Dati i numeri decimali 23456, 89765, 67489, 121331, 2453, 111010101

- si trasformino in base 3
- si trasformino in base 7
- si trasformino in base 2
- Dati i numeri 23456_7 , 121331_5 , 2453_8 , 111010101_2 convertire ciascun numero in decimale e in binario



Codifica esadecimale



Il codice esadecimale viene utilizzato come **forma compatta** per rappresentare numeri binari:

- 16 simboli: 0,1,...,9,A,B,...,F

- Diverse notazioni equivalenti:

0x9F

9F₁₆

9Fhex

da esadecimale a decimale

$$0x9F = 9 \times 16^1 + 15 \times 16^0 = 159_{10}$$

F esadecimale

corrisponde a

15 decimale

Cifre esadecimale	Valori decimali	Equivalenti binari
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Valori posizionali esadecimali	Valori decimali
16 ⁻³	1/4096=0.000 244 140 625
16 ⁻²	1/256=0.003 906 25
16 ⁻¹	1/16=0.062 5
16 ⁰	1
16 ¹	16
16 ²	256
16 ³	4096
16 ⁴	65 536
16 ⁵	1 048 576



Conversione esadecimale -> binario



Vogliamo rappresentare 9Fhex in binario. E' semplice.

- Ogni simbolo viene convertito in un numero binario di 4 cifre:

9hex --> 1001_{due}

Fhex --> 1111_{due}

9Fhex --> 1001 1111_{due}

- È sufficiente ricordarsi come si rappresentano in binario i numeri decimali da 0 a 15 (o derivarli)



Conversione binario -> esadecimale



Da binario ad esadecimale si procede in modo analogo:

- Ogni gruppo di 4 cifre viene tradotto nel simbolo corrispondente:

Esempio: convertire **01101011_{due}** in esadecimale:

1011_{due} --> **B_{hex}**

0110_{due} --> **6_{hex}**

0110 1011_{due} --> **6B_{hex}**

00000011001010001101000000100000 - add \$k0, \$t0, \$t9

0x0328d020



Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione

I numeri decimali



Somma



1110

← Riporto

101

01111 +

1473 +

00110 =

719 =

10001

2192

$11 + 6 = 17$ in decimale

Il riporto vale sempre 0/1

Vorrei definire solo l'operazione di somma e non utilizzare la sottrazione



Numeri negativi: complemento a 1



I numeri negativi sono complementari ai numeri positivi: $a + (-a) = 0$

Codifica in complemento a 1: il numero negativo si ottiene cambiando 0 con 1 e viceversa.

Problema:

000 0

Doppia codifica per lo 0.

001 1

risolto con il complemento a 2

010 2

011 3

100 -3

101 -2

110 -1

111 0

Doppia negazione riporta il numero al numero positivo (scambio 2 volte gli 0 con 1 e gli 1 con 0).

$$\begin{aligned} 2 - 2 &= 0 \Rightarrow 2 + (-2) = 0 \\ 010 - 010 &= 0 ? 010 + 101 = 111 = 000 \end{aligned}$$



Numeri negativi: complemento a 2



I numeri negativi sono complementari ai numeri positivi: $a + (-a) = 0$

Codifica in complemento a 2: il numero negativo si ottiene cambiando 0 con 1 e viceversa, e sommando 1.

000	0	
001	1	negativo: $110 + 1 = 111 = -1$
010	2	negativo: $101 + 1 = 110 = -2$
011	3	negativo: $100 + 1 = 101 = -3$
-4 + 0	100	-4
-4 + 1	101	-3
-4 + 2	110	-2
-4 + 3	111	-1

**Io 0 è considerato
positivo -> quindi ho
4 numeri positivi e 4
numeri negativi**

NB La prima cifra è il bit di segno.

**1 numero negativo
0 numero positivo**



Perché complemento a 2?



- La rappresentazione in complemento a due deve il suo nome alla proprietà in base alla quale la somma senza segno di un numero di n bit e del suo complemento è pari a 2^n (peso del bit *nessuno*, fuori dalla capacità di rappresentazione).

Per numeri codificati su **4 bit** + 1 bit di segno:

$$\begin{array}{ll} 7 + (-7) = & 00111 + 11001 = 10000 = 2^4 \\ 5 + (-5) = & 00101 + 11011 = 10000 = 2^4 \end{array}$$

- e quindi il complemento (o negazione) di un numero x in complemento a due è pari a $2^n - x$, ovvero il suo complemento a 2^n .

Per numeri codificati su 4 bit + 1 bit di segno:

$$\begin{aligned} 2^4 = 16 - 7 &= 10000 + 11001 = (1)1001 = 9_{10} \\ 2^4 = 16 - 5 &= 10000 + 11011 = (1)1011 = 11_{10} \end{aligned}$$



Doppia negazione



faccio la somma del negativo per fare la sottrazione

I numeri negativi sono complementari ai numeri positivi: $a + (-a) = 0$

Segue che $-(-a) = +a$ mi riporto al numero originario

Codifica in complemento a 2: il numero negativo si ottiene cambiando 0 con 1 e viceversa, e sommando 1.

000	0	
001	1	
010	2	
110	-2	
111	-1	

$$10 + 1 = 11$$

Esempio:

$$-(-2)_{10} = +2_{10}$$

$-(010)_2 \Rightarrow$ Complemento a 1 $\Rightarrow 010 \Rightarrow$ Sommo 1 (complemento a 2) $\Rightarrow 110_2$
 $-(110)_2 = 2_{10}$ Complemento a 1 $\Rightarrow 001 \Rightarrow$ Sommo 1 (complemento a 2) $\Rightarrow 010_2$ c.v.d.



Sottrazione



Sommo i seguenti 2 numeri $11 + (-13)$:

$$01011_2 = 11_{10}$$

$$10011_2 = -13_{10}$$

E' equivalente ad effettuare la differenza: $11 - 13$

$$\begin{array}{r}
 00110 \\
 01011 + \\
 10011 = \\
 \hline
 11110 \rightarrow -2_{10}
 \end{array}$$

$$+13_{10} = 01101_2 \Rightarrow \text{complemento a 1} \Rightarrow 10010 + 1 = 10011_2 = -13_{10}$$



Codifica dei numeri relativi (interi) su N bit



Occorre coprire tutti gli N bit a disposizione. Codifica su 16 bit:

da 0 a infinito
per informatici UNSIGNED

da -infinito a +infinito
per informatici INT

11 è positivo e mi bastano 4 bit ma devo riempire gli altri 12 bit con 0

Numeri naturali: $11_{10} = 1011_2 = 0000\ 0000\ 0000\ 1011$

Inserisco 0 fino a coprire tutti i bit; gli zeri sono parte integrante del numero

Numeri relativi: $+5_{10} = 0101_2 = 0000\ 0000\ 0000\ 0101 = +5_{10}$

Numeri relativi: $-5_{10} = 1011_2 = 1111\ 1111\ 1111\ 1011 = -5_{10}$

-8 +3
Replico il primo bit, quello del segno

Nota bene. Queste rappresentazioni del numero -5_{10} sono sbagliate:

1000 0000 0000 1011 sarebbe sbagliato = $-16,395_{10}$

0000 0000 0000 1011 sarebbe sbagliato = $+11_{10}$



Capacità di rappresentazione Numeri Interi (relativi)

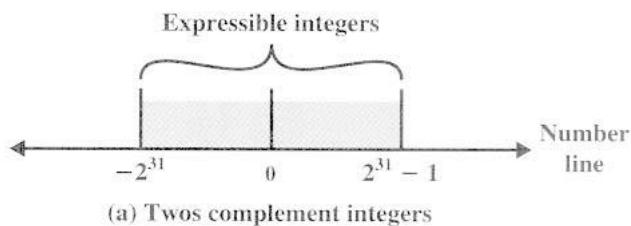


non considero i numeri decimali

Interi con segno su N bit. Range: $-2^{N-1} \leq n \leq 2^{N-1} - 1$.

Esempio: Visual C++. Intero è su 4byte (word di 32 bit):

$-2^{31} = -2.147.483.650 \leq n \leq 2.147.483.649 = 2^{31} - 1$
10.000.000.000 supera la capacità di rappresentazione



Risoluzione della codifica: 1 unità



Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione.

I numeri decimali



Conversione base 10 -> base n: algoritmo



Un numero $x.y$ in base 10 si trasforma in base n usando il seguente procedimento.

Per la parte intera, x , si applica l'algoritmo visto in precedenza:

- Dividere il numero x per n
 - Il resto della divisione è la cifra di posto 0 in base n
 - Il quoziente della divisione è a sua volta diviso per n
 - Il resto ottenuto a questo passo è la cifra di posto 1 in base n
-
- Si prosegue con le divisioni dei quozienti ottenuti al passo precedente fino a che l'ultimo quoziente è 0.
 - l'ultimo resto è la cifra più significativa in base n



«Estrazione» delle cifre decimali contenute dopo la virgola



Vogliamo estrarre le cifre di $0,3672_{\text{dieci}}$. Porto le cifre alla **sinistra** della virgola:

$$\begin{array}{ll} 0,3672 * 10 = \mathbf{3,672} & \rightarrow \text{esamino } 0,672 \rightarrow 3 \text{ decimi} \\ 0,672 * 10 = \mathbf{6,72} & \rightarrow \text{esamino } 0,72 \rightarrow 6 \text{ centesimi} \\ 0,72 * 10 = \mathbf{7,2} & \rightarrow \text{esamino } 0,2 \rightarrow 7 \text{ millesimi} \\ 0,2 * 10 = \mathbf{2,0} & \rightarrow \text{termina} \quad \rightarrow 2 \text{ decimillesimi} \end{array}$$

$$N = \sum_k c_k b_k$$

K è negativo per le cifre dopo la virgola (la parte frazionaria del numero)



Conversione base 10 -> base 2 “estrazione” delle cifre binarie dopo la virgola



Vogliamo rappresentare $0,625_{\text{dieci}}$ in binario: $\mathbf{0,101}_{\text{due}}$

$$\begin{array}{lll} 0,625 * 2 = \mathbf{1,250} = 1 + 0,250 & => 1 & \text{moltiplico x 2 la parte} \\ 0,250 * 2 = \mathbf{0,500} = 0 + 0,5 & => 0 & \text{dopo la virgola, finche non} \\ 0,500 * 2 = \mathbf{1,000} = 1 + 0,0 & => 1 & \text{ottengo cifra senza} \\ & 0,0000 & \text{decimali, e leggo i prodotti} \\ & & \text{interi ottenuti dall'alto} \\ & & \text{verso il basso} \end{array}$$

$$1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} = 1/2 + 1/8 = 0,5 + 0,125 = 0,625$$



Conversione base 10 -> base n: algoritmo per la parte frazionaria



Un numero x,y in base 10 si trasforma in base n usando il seguente procedimento.

Per la parte frazionaria, y :

- Moltiplicare il numero y per n
- La prima cifra del risultato coincide con la cifra di posto 1 dopo la virgola.
- Si elimina la parte intera ottenuta e si considera la nuova parte frazionaria.
- La parte frazionaria ottenuta viene moltiplicata per la base n .
- La prima cifra del risultato coincide con la cifra di posto 2 dopo la virgola.
- Si prosegue con le moltiplicazioni della parte frazionaria fino a quando non diventa 0 o non si esaurisce la capacità di rappresentazione.



Conversione base 10 -> base 2 “estrazione” delle cifre binarie dopo la virgola



Vogliamo rappresentare $0,626_{\text{dieci}}$ in binario: $0,1010000001\dots_{\text{due}}$

$$\begin{array}{ll}
 0,626 * 2 = 1,252 = 1 + 0,252 & \Rightarrow 1 \\
 0,252 * 2 = 0,504 = 0 + 0,504 & \Rightarrow 0 \\
 0,504 * 2 = 1,008 = 1 + 0,008 & \Rightarrow 1 \\
 0,008 * 2 = 0,016 = 0 + 0,016 & \Rightarrow 0 \\
 0,016 * 2 = 0,032 = 0 + 0,032 & \Rightarrow 0 \\
 0,032 * 2 = 0,064 = 0 + 0,064 & \Rightarrow 0 \\
 0,064 * 2 = 0,128 = 0 + 0,128 & \Rightarrow 0 \\
 0,128 * 2 = 0,256 = 0 + 0,256 & \Rightarrow 0 \\
 0,256 * 2 = 0,512 = 0 + 0,512 & \Rightarrow 0 \\
 0,512 * 2 = 1,024 = 1 + 0,024 & \Rightarrow 1 \\
 0,024 * 2 \dots\dots &
 \end{array}$$

$$N = \sum_k c_k b_k$$

$$1*2^{-1} + 1*2^{-3} + 1*2^{-10} = 1/2 + 1/8 + 1/1024 = 0,5 + 0,125 + 0,0009765625 = 0,6259765625$$

$$\lim_{k \rightarrow \infty} \left(\sum_k c_k 2^{-k} \right) = 0,626$$

Errore di approssimazione $< 2^{-10}$



Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione.

I numeri decimali



I circuiti logici: definizione delle funzioni logiche

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti al testo: Appendice B, sezioni B.1 e B.2



Sommario



Variabili ed operatori semplici.

Implementazione circuitale (porte logiche).

Dal circuito alla funzione.

Algebra Booleana.



Le appendici del Patterson sono on-line...



A P P E N D I X

I always loved that word, Boolean.

Claude Shannon
IEEE Spectrum, April 1992
(Shannon's master's thesis showed that the algebra invented by George Boole is the basic mathematical language of the universe.)

A.A. 2023-2024

[http://online.universita.zanichelli.it/
patterson-4e/xstudx-appendici/](http://online.universita.zanichelli.it/patterson-4e/xstudx-appendici/)

The Basics of Logic Design

- B.1 Introduction B-3
- B.2 Gates, Truth Tables, and Logic Equations B-4
- B.3 Combinational Logic B-9
- B.4 Using a Hardware Description Language B-20

» [Constructing a Basic Arithmetic Logic Unit](#)

3/50

<http://borgheze.di.unimi.it/>



Le operazioni logiche fondamentali



All'interno di un elaboratore vengono effettuate **solo** operazioni logiche.

NOT

AND

OR

QUALUNQUE funzione booleana (logica) può essere espressa combinando opportunamente tre funzioni booleane elementari.

Si dice anche che AND, OR, NOT formano un set completo.

A.A. 2023-2024

4/50

<http://borgheze.di.unimi.it/>



Circuiti Booleani



“An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities” G. Boole, 1854: approccio alla logica come algebra.

Variabili (binarie, **0 = FALSE**; **1 = TRUE**).

Operazioni sulle variabili (NOT, AND, OR).

Equivalenza tra operazioni logiche su proposizioni vere/false e operazioni algebriche su variabili binarie.

Utilizzo dell'algebra Booleana per:

- Progettazione (sintesi) dei circuiti digitali. Data una certa funzione logica, sviluppare il circuito digitale che la implementa (implementeremo anche le operazioni aritmetiche come funzioni logiche!).
- Analisi dei circuiti. Descrizione della funzione logica implementata dai circuiti.
- Semplificazione di espressioni logiche per ottenere implementazioni efficienti.

Ma anche:

- Verifica dei protocolli.
- Ottimizzazione con vincoli.



Rappresentazione delle funzioni logiche



	A	B	C	Y	(A,B,C) → Y
0	F=0	F=0	F=0	F=0	Input (dominio)
1	F=0	F=0	V=1	V=1	Output (codominio)
2	F=0	V=1	F=0	F=0	
3	F=0	V=1	V=1	F=0	
4	V=1	F=0	F=0	F=0	
5	V=1	F=0	V=1	V=1	
6	V=1	V=1	F=0	V=1	
7	V=1	V=1	V=1	V=1	

**Tabella della verità
(truth table – TT)**

**per ogni gruppo di variabili in ingresso
viene stabilita l'uscita se è vera o falsa**

Vengono considerate tutte le combinazioni possibili in input, terna A,B,C, dove ciascuna variabile può assumere i valori (V,F). Le combinazioni in ingresso si possono indicizzare con i numeri interi: 0 (000), 1 (001), 2 (010), 3 (011), 4 (100), 5 (101), 6 (110), 7 (111).

Per ogni combinazione in input viene prescritto l'output, Y, desiderato (V,F).

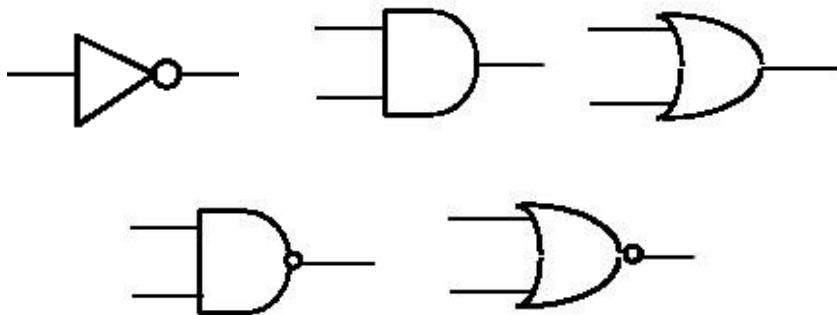
Rappresentazione esaustiva di una funzione.



Porte logiche



Gli **operatori logici** vengono implementati da dei circuiti elettronici che vengono chiamati **porte logiche**



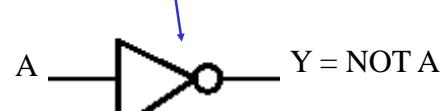
Operatore NOT



A	Y
F = 0	1 = V
V = 1	0 = F

Tabella della verità

Simbolo



"Inverter logico"

Se **A** è vero ($A = \text{TRUE} = 1$), **NOT A** è falso ($\text{FALSE} = 0$) e viceversa.

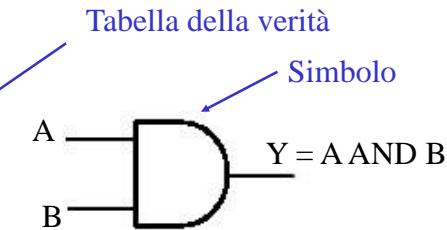
Scrittura algebrica: $\text{NOT } A = \overline{A} = !A$



Operatore AND



A	B	Y
FF	0	0 F
FV	0	0 F
VF	1	0 F
VV	1	1 V



Esempio: "Se c'è il sole **e** c'è Michela (Michele) vado a sciare"

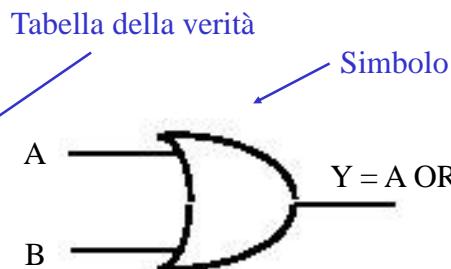
Scrittura algebrica: $Y = A \text{ AND } B = A \cdot B = AB$



Operatore OR



A	B	Y
FF	0	0 F
FV	0	1 V
VF	1	1 V
VV	1	1 V



Esempio: "Se c'è il sole **o** c'è Michela (Michele) vado a sciare"

Scrittura algebrica: $Y = A \text{ OR } B = A + B$



Somma e prodotto logico



Somma logica

=> OR

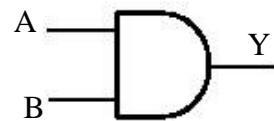
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Prodotto logico

=> AND

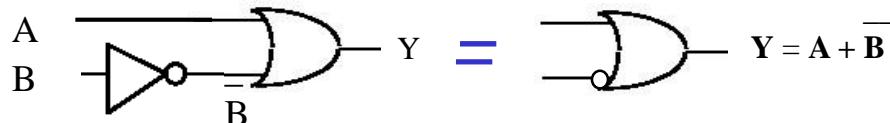
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



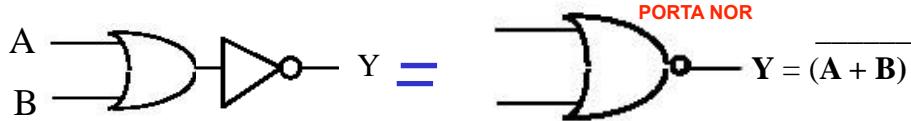
Concatenazione del NOT



è integrato con porte AND e OR



Inserire un cerchietto all'ingresso corrisponde a negare (complementare) l'ingresso.



Inserire un cerchietto all'uscita corrisponde a negare (complementare) l'uscita.

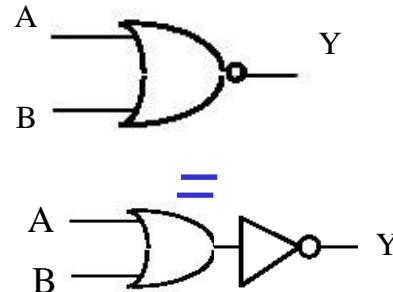


Operatore NOR



A	B	OR(A,B)	Y
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Operatore OR negato



“Not(Or(A,B))”

$$Y = \overline{A + B} = !(A + B)$$

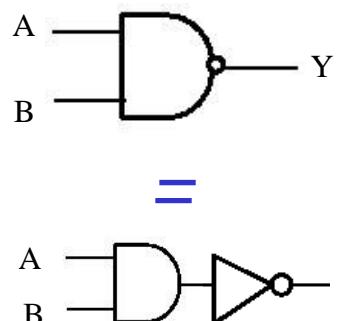


Operatore NAND



A	B	AND(A,B)	Y
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Operatore AND negato



“Not(And(A,B))”

$$Y = \overline{AB}$$

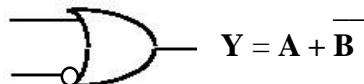


Tabella della verità



B	A	B	Y
0	0	0	1
0	0	1	0
1	1	0	1
0	1	1	1

$$Y = OR(A, \bar{B})$$



Questa funzione è diversa dalla funzione OR(A,B)
La tabella della verità è diversa.



Precedenza degli operatori



- 1) NOT
- 2) AND
- 3) OR

$$Y = A + B \bar{C} \text{ va intesa come: } Y = A + (B (\bar{C}))$$

Ordine di esecuzione:

- 1) \bar{C} - negazione
- 2) $B \bar{C}$ - AND
- 3) $A + BC$

$$\text{Sarebbe sbagliato leggere l'espressione logica come: } Y = (A+B) \bar{C} \neq A + B \bar{C}$$



Concatenazione degli operatori - I



In assenza di parentesi, AND ha la priorità sull'OR ed il NOT su entrambi:

NOT -> AND -> OR

$$A + B \cdot C = A + (B \cdot C)$$

per eseguire prima OR occorre scrivere esplicitamente: $(A+B) \cdot C$

In assenza di parentesi, la negazione ha la priorità sugli altri operatori.

$$\text{NOT } A \cdot C = (\text{NOT}(A)) \cdot C = \bar{A}C$$

A B C Eseguo prima la negazione. Quindi eseguo prima l'AND di A e B, nego il risultato dell'AND ed infine eseguo l'AND con C:

- 1) AB
- 2) !(AB)
- 3) (!AB) C

A.A.



Concatenazione degli operatori - II



$$\bar{A} \bar{B} = (\bar{A}) (\bar{B})$$

- 1) Prima eseguo la negazione di A e la negazione di B in parallelo.
- 2) Poi eseguo l'AND.

$$\bar{A} \bar{B} C = [\bar{A} (\bar{B})] C$$

- 1) Prima eseguo la negazione di A e la negazione di B in parallelo.
- 2) Eseguo il prodotto logico di !A, !B e C.

Anche sulle negazioni esiste una gerarchia:

$$\overline{\overline{A} \overline{B}} = \overline{\overline{A}} \overline{\overline{B}} = ! ((!A) (!B))$$

- 1) Prima eseguo la negazione di A e la negazione di B in parallelo.
- 2) Eseguo poi l'AND di !A e !B = AND(!A, !B)
- 3) Infine nego il risultato dell'AND

A.A.



Porte logiche a più ingressi

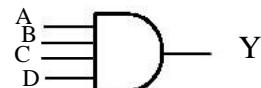


- Rappresentano circuiti che forniscono in uscita il risultato di operazioni logiche elementari applicate a più variabili in ingresso.
- Le variabili in ingresso possono essere n.

Ad esempio:

$$Y = A \text{ AND } B \text{ AND } C \text{ AND } D$$

uscita = 1 se tutti gli ingressi = 1



$$Y = A \text{ OR } B \text{ OR } C \text{ OR } D$$

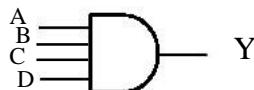


Porte logiche: tabella della verità



A	B	C	D	AND	OR
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	0	1
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	1	1

$$Y = A \text{ AND } B \text{ AND } C \text{ AND } D$$



$$Y = A \text{ OR } B \text{ OR } C \text{ OR } D$$





Sommario



Variabili ed operatori semplici.

Implementazione circuitale (porte logiche).

Dal circuito alla funzione.

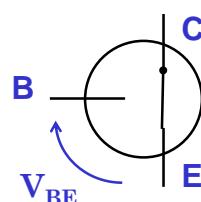
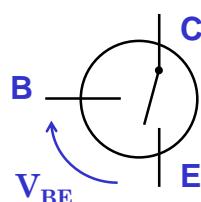
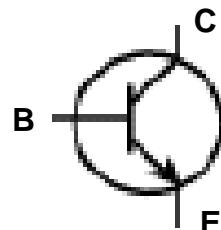
Algebra Booleana.



Il Transistor



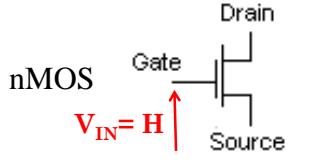
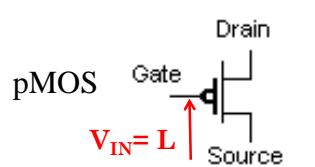
- Componente con funzionamento lineare (amplificazione) o **binario** (interruttore).
- Modello: interruttore tra **Emettitore** e **Collettore**, **comandato** dalla tensione sulla **Base**.
- Le porte logiche sono costituite da transistor.
- 2 casi “estremi”:
 - Tensione **V_{BE} bassa** → C,E isolati (**spento**)
 - Transistor in stato di **INTERDIZIONE**
 - Tensione **V_{BE} alta** → C,E collegati (**acceso**)
 - Transistor in stato di **SATURAZIONE** ($V_C = V_E$)





La tecnologia CMOS: dal 1980 a oggi

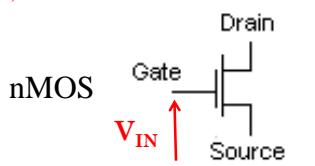
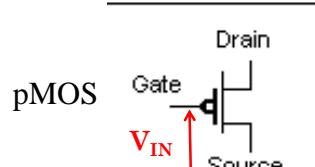


- **CMOS:** Complementary-MOS (Frank Wanlass, 1967 Fairchild)
 - MOS: Metal – Oxide Semiconductor
 - MOS complementari (**N-MOS + P-MOS**) che lavorano “in coppia”: substrati comuni.
- nMOS è acceso se: $V_{GS} >$ soglia
 $V_G \approx V_{dd}$
- pMOS è acceso se: $V_{DG} >$ soglia ($V_{GS} <$ soglia)
 $V_G \approx V_{ss}$
- Un MOS:
- Quando è acceso funziona da corto circuito tra D e S
 - Quando è spento funziona da circuito aperto tra D e S
- Accesso se: $V_{in} > V_{dd}/2$
- 
- 
- Accesso se: $V_{in} < V_{dd}/2$



La tecnologia CMOS: vantaggi



- Vantaggi:
 - Tensione di alimentazione, V_{CC} , “flessibile”:
 $V_{CC} = 1 \div 15$ Volt (vicina a 1 V quasi sempre)
 - Consumo bassissimo:
 - Consuma solo nella transizione
 - In condizioni statiche, consumo praticamente nullo!
- Si può vedere come un ponte levatoio che si alza e si abbassa tra Drain e Source ed è pilotato dal gate. Si consuma energia solo quando viene alzato o abbassato.
- Accesso se: $V_{in} > V_{dd}/2$
- 
- 
- Accesso se: $V_{in} < V_{dd}/2$



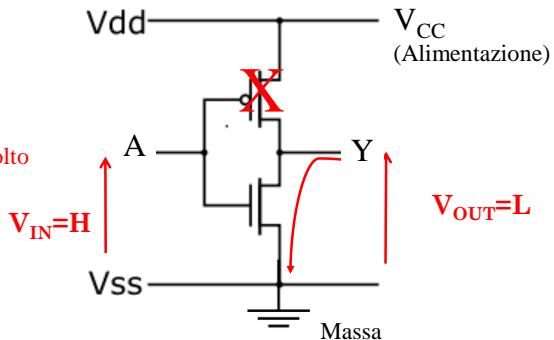
La porta NOT in CMOS



2 MOS complementari

$V_{in} = L = 0V$ il transistor inferiore è spento, $V_{out} = V_{dd} = H$

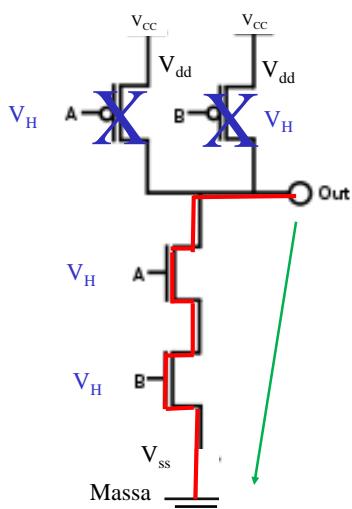
$V_{in} = H = V_{dd}$ passa corrente nel transistor inferiore, la resistenza è molto bassa e $V_{out} \approx 0 = L$. Il transistor superiore è spento.



La porta NOT è chiamata anche Inverter logico.



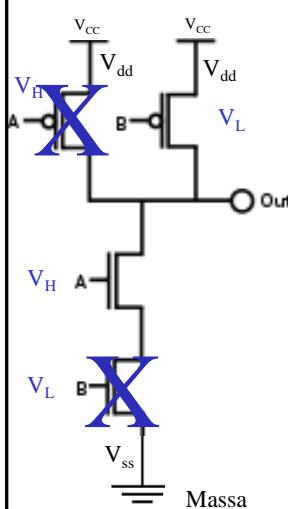
Porta NAND in C-MOS



V_A	V_B	V_{OUT}
$V_L=0$	$V_L=0$	$V_H=1$
$V_L=0$	$V_H=1$	$V_H=1$
$V_H=1$	$V_L=0$	$V_H=1$
$V_H=1$	$V_H=1$	$V_L=0$

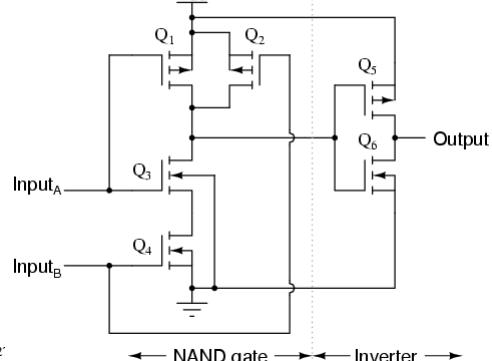


Porta AND in C-MOS



V_A	V_B	V_{OUT}
$V_L=0$	$V_L=0$	$V_H=1$
$V_L=0$	$V_H=1$	$V_H=1$
$V_H=1$	$V_L=0$	$V_H=1$
$V_H=1$	$V_H=1$	$V_L=0$

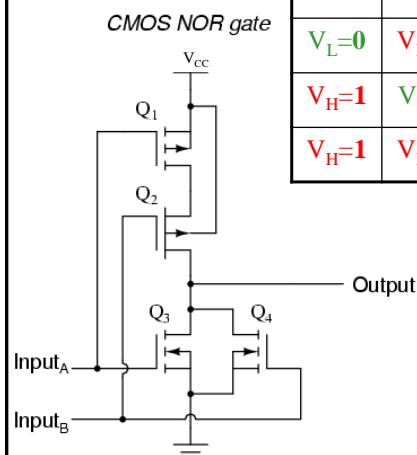
CMOS AND gate



2'

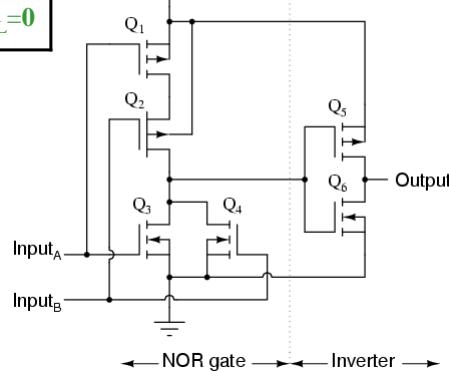


PORTE NOR e OR IN CMOS



V_1	V_2	V_{OUT}
$V_L=0$	$V_L=0$	$V_H=1$
$V_L=0$	$V_H=1$	$V_L=0$
$V_H=1$	$V_L=0$	$V_L=0$
$V_H=1$	$V_H=1$	$V_L=0$

CMOS OR gate



28/50

<http://borgheze.di.unimi.it/>

A.A. 2023-2024

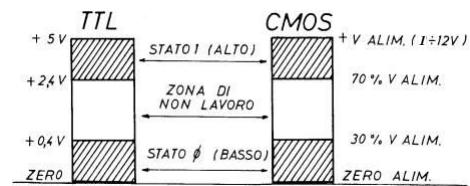


Perchè l'elettronica digitale funziona?



Perchè è progettata per essere resistente al rumore.

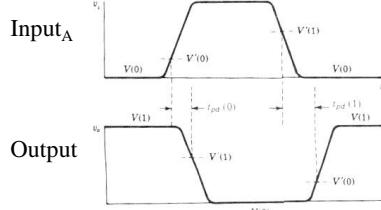
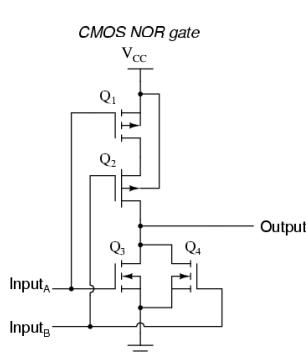
Vengono definiti 2 range di tensioni associati ai valori alto e basso, separati da un gap.



Tempo di commutazione



La commutazione non è istantanea



Più porte in cascata generano un ritardo nella commutazione dell'uscita.

Numero limitato di porte logiche che si possono collegare in ingresso e in uscita (fan-in / fan-out)



Sommario



Variabili ed operatori semplici.

Implementazione circuitale (porte logiche).

Dal circuito alla funzione.

Algebra Booleana.



Funzione



- Una relazione che associa ad ogni elemento dell'insieme $X:\{x\}$, detto dominio, associa un elemento dell'insieme $Y: \{y\}$, detto codominio, indicandola con $y = f(x)$ (Wikipedia). Sinonimi: mappa, trasformazione.
- Le nostre funzioni saranno tipicamente relazioni tra ingressi e uscite multi-dimensional (più variabili in ingresso e più variabili in uscita).
- Nulla è detto sulla forma di questa relazione.
 - Funzione analitica (e.g. $Y = \sin(x) \log(\cos(x/2))\dots$)
 - Funzione a più valori di input e più valori di output (x ed y vettori)
 - Tabella di corrispondenza (tabella di verità, caratteristica: descrizione esplicita ed esaustiva della corrispondenza).
 -



Funzioni logiche



- Funzione a n ingressi e m uscite. Per ciascuna delle 2^n combinazioni degli ingressi, viene definita ciascuna delle m uscite.
- E' una funzione discreta con domini finiti. Si può calcolare per tutti i punti nei quali la funzione è definita.
- La funzione logica sarà implementata da un'opportuna combinazione di porte logiche (NOT, AND, OR) che prende gli n ingressi e li trasforma nell'uscita specificata.
- La funzione logica sarà quindi calcolata da un circuito con n ingressi e m uscite, costituito da porte logiche.
- La funzione può essere rappresentata in 3 modi:
 - Circuito
 - Tabella della verità (Truth Table, TT)
 - Espressione simbolica



Dall'espressione logica alla TT



Data l'espressione: $F = A \text{ AND } B \text{ OR } B \text{ AND } \text{NOT } C = AB + BC$

Questa espressione si legge: $F = (A \text{ AND } B) \text{ OR } (B \text{ AND } (\text{NOT } C))$

Ricaviamo la tabella delle verità:

A	B	C	A and B	B and ($\text{NOT } C$)	F
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1



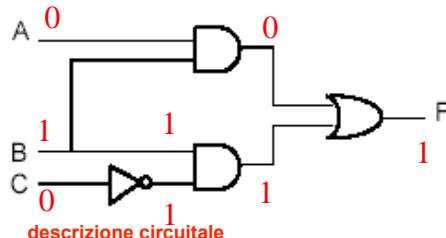
Dal circuito alla TT - 1



$$F = AB + \bar{BC}$$

descrizione algebrica

$$F = (A \text{ AND } B) \text{ OR } (\bar{B} \text{ AND } \text{NOT}(C))$$



descrizione circuitale

Direzione del calcolo

tabella di verità

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

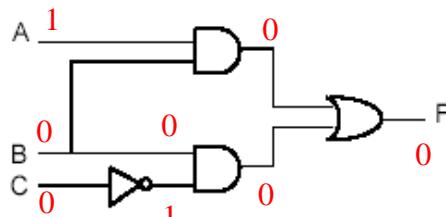


Dal circuito alla TT - 2



$$F = AB + \bar{BC}$$

$$F = (A \text{ AND } B) \text{ OR } (\bar{B} \text{ AND } \text{NOT}(C))$$



Direzione del calcolo

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

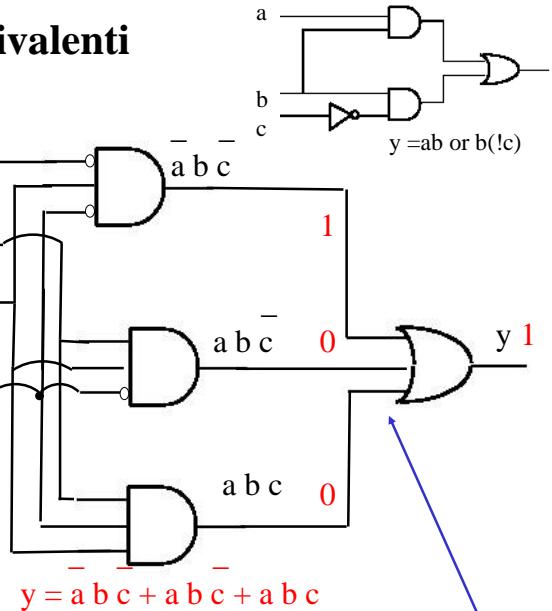


Circuiti equivalenti

a b c	y
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

Espressioni algebriche e tabella danno la stessa uscita y

Espressioni algebriche diverse (e circuiti diversi) possono implementare la stessa funzione logica (stessa TT)



Implementazione circuitale diversa. L'implementazione circuitale di una funzione logica non è unica!

37/50

<http://borgheze.di.unimi.it/>

TT = tabella verità

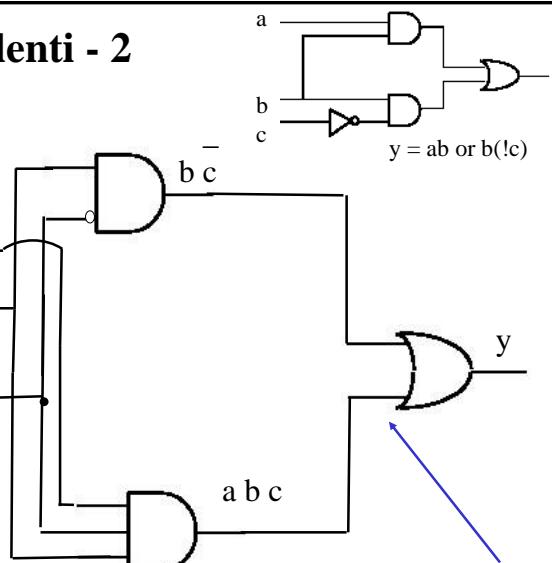


Circuiti equivalenti - 2

a b c	y
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

Funzione e tabella danno la stessa uscita y.

Espressioni algebriche diverse e circuiti diversi possono implementare la stessa funzione logica (stessa TT).



Implementazione circuitale diversa. L'implementazione circuitale di una funzione logica (descritta da una TT) non è unica!

38/50

$y = a b c + b !c$

<http://borgheze.di.unimi.it/>



Sommario



Variabili ed operatori semplici.

Implementazione circuitale (porte logiche).

Dal circuito alla funzione.

Algebra Booleana.



Manipolazione algebrica - I



Identità	AND $1 \cdot x = x$	OR $0 + x = x$
<i>Algebra classica</i>	$I \cdot N = N$	$0 + N = N$

Elemento nullo	$0 \cdot x = 0$	$1 + x = 1$
<i>Algebra classica</i>	$0 \cdot N = 0$	-----

*Uno degli ingressi della porta AND funge da interruttore:
= 0, uscita è sempre =0, interruttore aperto
= 1, uscita è uguale all'ingresso, interruttore chiuso.*

Inverso	$\bar{x} \cdot x = 0$	$\bar{x} + x = 1$
<i>Algebra classica</i>	-----	-----

Idempotenza	$x \cdot x = x$	$x + x = x$
<i>Algebra classica</i>	-----	-----



Manipolazione algebrica - II



Doppia Inversione $\overline{\overline{x}} = x$

Algebra classica -----

Associativa $(x y) z = x (y z)$ $(x + y) + z = x + (y + z)$

Algebra classica $(NM)K = N(MK)$

$(N+M)+K = N+(M+K)$

Commutativa $x y = y x$

Algebra classica $NM = MN$

$x + y = y + x$
 $N + M = M + N$

AND rispetto ad OR **OR rispetto ad AND**

Distributiva $x (y + z) = x y + x z$ $x + y z = (x + y) (x + z)$

Algebra classica $N(M+K) = NM+NK$ -----

Assorbimento $x (x + y) = x$

Algebra classica -----

$x + x y = x$



Principio di dualità



Nell'algebra di Boole vale il principio di dualità.

Il duale di una funzione booleana si ottiene sostituendo AND ad OR, OR ad AND, gli 0 agli 1 e gli 1 agli 0.

Esempi:

- Identità $1 \cdot x = x$ $0 + x = x$
- Elemento nullo $0 \cdot x = 0$ $1 + x = 1$

Proprietà Commutativa:

$$x y = y x \quad x + y = y + x$$

Proprietà distributiva:

$$x (y + z) = xy + xz \quad x + (y z) = x + yz$$

Assorbimento: $x (x + y) = x$

$$x + x y = x$$



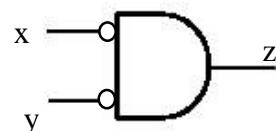
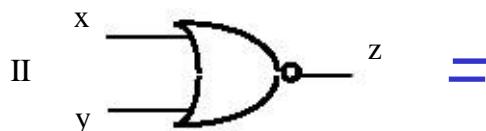
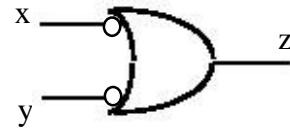
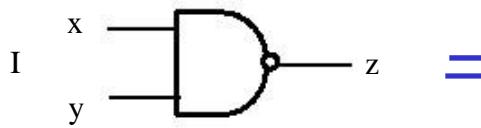
Teoremi di De Morgan



Enunciati:

$$I) \overline{xy} = \overline{x} + \overline{y}$$

$$II) \overline{\overline{x} + \overline{y}} = \overline{x} \overline{y}$$



Regole algebriche - riassunto



Doppia Inversione

$$\overline{\overline{x}} = x$$

AND

OR

Identità

$$1 \cdot x = x$$

$$0 + x = x$$

Elemento nullo

$$0 \cdot x = 0$$

$$1 + x = 1$$

Idempotenza

$$x \cdot x = x$$

$$x + x = x$$

Inverso

$$x \cdot \overline{x} = 0$$

$$x + \overline{x} = 1$$

Commutativa

$$x \cdot y = y \cdot x$$

$$x + y = y + x$$

Associativa

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

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

AND rispetto ad OR

OR rispetto ad AND

Distributiva

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

$$x + y \cdot z = (x + y) \cdot (x + z)$$

Assorbimento

$$x \cdot (x + y) = x$$

$$x + x \cdot y = x$$

De Morgan

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

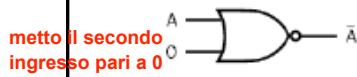
$$\overline{\overline{x} + \overline{y}} = \overline{x} \overline{y}$$



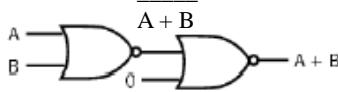
Porta Universale NOR



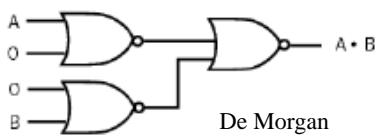
- NOT A = 0 NOR A
- A OR B = (A NOR B) NOR 0
- A AND B = (A NOR 0) NOR (B NOR 0)



NOT



OR



AND

$$\overline{AB} = \overline{\overline{A} + \overline{B}}$$

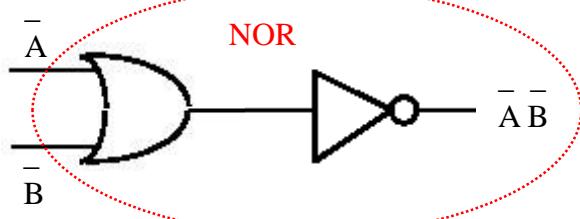
$$AB = \overline{\overline{AB}} = \overline{\overline{\overline{A} + \overline{B}}}$$



De Morgan NOR -> AND

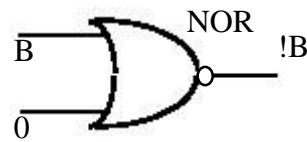
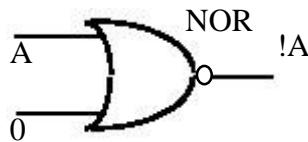


$$\text{De Morgan: } \overline{A+B} = \overline{A}\overline{B}$$



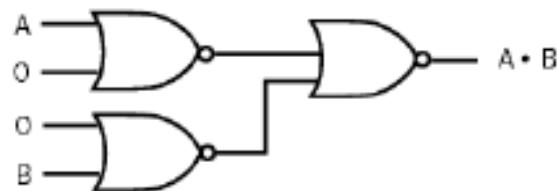
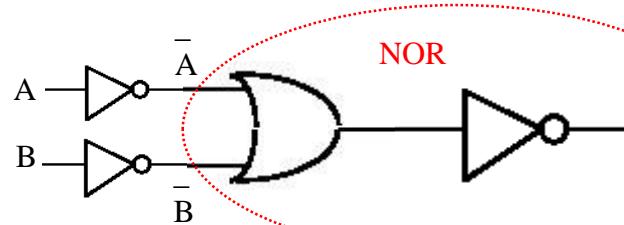
Questo circuito con NOR è equivalente a AND(A,B). Come faccio con le negazioni?

Applico la negazione agli ingressi:





De Morgan NOR -> AND



A.A.

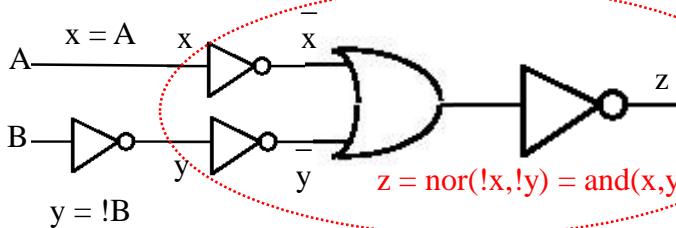
<http://borghese.di.unimi.it/>



De Morgan in generale



Quale funzione è implementata da questo circuito?



Sostituisco il not e ottengo: $z = \text{and}(A, \bar{B}) = A \bar{B}$



Esercizio



Data la funzione booleana:

$$F = (A \text{ AND } B) \text{ OR } (B \text{ AND NOT}(C))$$

Esprimere la funzione F con il solo connettivo logico NOR e disegnare il circuito.

Esprimere la funzione F con il solo connettivo logico NAND e disegnare il circuito.

Costruire le porte logiche: AND, OR, NOT utilizzando solo la porta NAND.



Sommario



Variabili ed operatori semplici.

Implementazione circuitale (porte logiche).

Dal circuito alla funzione.

Algebra Booleana.



I circuiti digitali: dalle funzioni logiche ai circuiti (le SOP)



Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al testo: Patterson Hennessy, Sezione B.3 on-line;
Approfondimento sulle forme canoniche: Fummi et al., Progettazione Digitale, McGrawHill, capitolo 3.



Sommario



I circuiti combinatori.

Semplificazione algebrica.

Dalla tabella della verità al circuito: la prima forma canonica: SOP.

Criteri di ottimalità.



Circuiti combinatori



- Circuiti logici digitali in cui le operazioni (logiche) dipendono solo da una combinazione degli input. Come nelle funzioni a valori reali, il risultato dell'elaborazione del circuito viene aggiornato immediatamente dopo il cambiamento dell'input (si suppone il tempo di commutazione trascurabile; tempo di attesa prima di guardare l'output sufficientemente ampio per permettere a tutti i circuiti la commutazione).
- Circuiti senza memoria. Ogni volta che si inseriscono in ingresso gli stessi valori, si ottengono le stesse uscite. Il risultato non dipende dallo stato del circuito.
- I circuiti combinatori descrivono delle funzioni Booleane. Queste funzioni si ottengono combinando tra loro (in parallelo o in cascata) gli operatori logici: **NOT, AND, OR**.
- Il loro funzionamento può essere descritto come **tabella della verità**.
- Dato un circuito è univoca la funzione algebrica che ne rappresenta il funzionamento e viceversa.



Un po' di tassonomia



- **Espressione logica.** Combinazione di operatori logici. Ad ogni espressione logica è associato un ben preciso circuito.

$$\overline{AB} + BC$$

- **Funzione logica.** Corrispondenza tra un insieme di ingresso (valori possibili di A, B, C) e un insieme di uscita (valori possibili di Y).

$$Y = \overline{A}B + \overline{B}C$$

Una funzione logica viene rappresentata come espressione logica.



Regole manipolazione algebrica



Doppia Inversione

$$\overline{\overline{x}} = x$$

AND

OR

Identità

$$1 \cdot x = x$$

$$0 + x = x$$

Elemento nullo

$$0 \cdot x = 0$$

$$1 + x = 1$$

Idempotenza

$$x \cdot x = x$$

$$x + x = x$$

Inverso

$$x \cdot \overline{x} = 0$$

$$x + \overline{x} = 1$$

Commutativa

$$x \cdot y = y \cdot x$$

$$x + y = y + x$$

Associativa

$$(x \cdot y) \cdot z = x \cdot (y \cdot z)$$

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

AND rispetto ad OR

OR rispetto ad AND

Distributiva

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

$$x + y \cdot z = (x + y) \cdot (x + z)$$

Assorbimento

$$x \cdot (x + y) = x$$

$$x + x \cdot y = x$$

De Morgan

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

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

Si possono dimostrare sostituendo 0/1 alle variabili.



Regole algebriche su più variabili



Commutativa $x \cdot y \cdot z = y \cdot x \cdot z = z \cdot x \cdot y$ $x + y + z = y + x + z = z + x + y$

AND rispetto ad OR

OR rispetto ad AND

Distributiva

$$x \cdot (yh + z) = xyh + xz$$

$$x \cdot h + y \cdot z = (x \cdot h + y) \cdot (x \cdot z)$$

De Morgan

$$\overline{xyz} = \overline{x} + \overline{y} + \overline{z}$$

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

Si possono dimostrare sostituendo 0/1 alle variabili.



Funzione come espressione logica o come tabella delle verità



$$Y = A \cdot B + B \cdot \bar{C}$$

A	B	C	A and B	B and not(C)	Y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1



Una seconda rappresentazione - I



$$Y = (A \cdot B) + (\bar{B} \cdot C)$$

Applico De Morgan ai prodotti logici:

$$\overline{\overline{AB}} = \overline{A} \cdot \overline{B} \quad \overline{\overline{BC}} = \overline{B} + \overline{C}$$

$$Y = (\overline{A} \cdot \overline{B}) + (\overline{B} \cdot C) = (\overline{A} + C) \cdot \overline{B}$$

NB !B e !B non si sommano mediante somma logica perché prima vanno calcolate le negazioni!!

Potrei effettuare la somma se fosse:

$$Y = (\overline{A} + \overline{B}) + (\overline{B} + C) =$$

$$\overline{A} + \overline{B} + \overline{B} + C = \overline{A} + C$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Una seconda rappresentazione - II



$$Y = \overline{(A + B)} + \overline{(B + C)}$$

Voglio sostituire la somma con un prodotto logico.
Applico De Morgan:

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

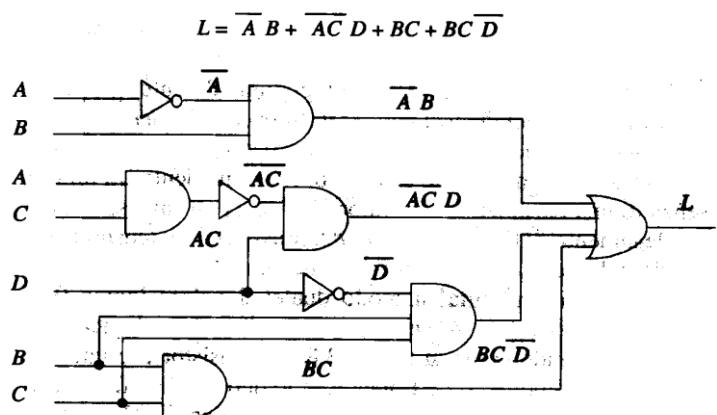
$$Y = \overline{(A + B)} \cdot \overline{(B + C)}$$

*E' la funzione AND
costruita con
un'espressione
logica diversa!*

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Esempio – rappresentazione 1





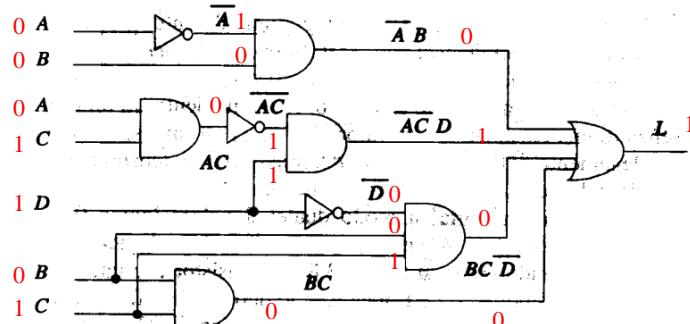
Esempio – tabella verità



A	B	C	D	L
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
.....	0	0	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

per avere uscita = 1 \rightarrow basta che uno degli ingressi sia = 1 perchè sono legati da OR

$$L = \overline{A}B + \overline{AC}D + BC + \overline{BC}D$$



Manipolazione algebrica



Applichiamo De Morgan.

espressione di prima

$$L = \overline{A}B + \overline{AC}D + \overline{BC}D + BC =$$

da prodotti faccio diventare somme

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

$$= \overline{A} + B + \overline{AC} + D + \overline{BC} + D + \overline{B} + C =$$

$$= (\overline{A} + B)(\overline{AC} + D)(\overline{B} + C)(\overline{BC} + D)$$

x y z h

da somme logiche a prodotti

$$\overline{x} + \overline{y} + \overline{z} + \overline{h} = \overline{xyzh}$$

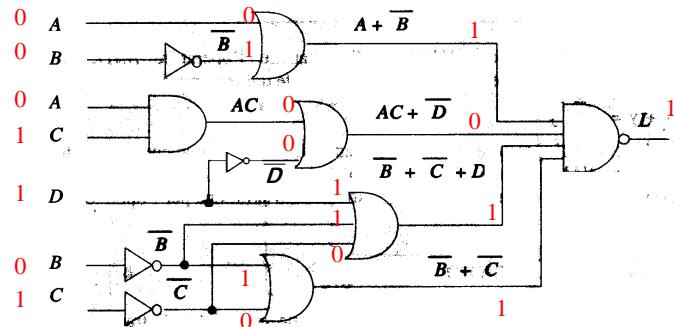


Esempio – rappresentazione 2



A	B	C	D	L
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$L = \overline{(A + \overline{B})} (\overline{AC} + \overline{D}) (\overline{B} + \overline{C} + D) (\overline{B} + \overline{C})$$



Sommario



I circuiti combinatori.

Semplificazione algebrica.

Dalla tabella della verità al circuito: la prima forma canonica: SOP.

Criteri di ottimalità.



Semplificazioni notevoli



Dimostrare che: $A + AB = A + B$

Proprietà distributiva di OR rispetto ad AND:

$$\overline{A} + \overline{AB} = (\overline{A} + \overline{A}) (\overline{A} + \overline{B})$$

Sviluppando il prodotto:

$$(\overline{A} + \overline{B})(\overline{A} + \overline{A}) = \overline{AA} + \overline{A}\overline{A} + \overline{BA} + \overline{BA} = \overline{A} + \overline{AB} + \overline{AB}$$

Raccogliendo \overline{B} :

$$\overline{A} + \overline{AB} + \overline{AB} = \overline{A} + (\overline{A} + \overline{A})B = \overline{A} + B$$

NB: posso anche identificare i 3 «1» della funzione OR:

$$\overline{A} + \overline{AB} = \overline{A}(B + \overline{B}) + \overline{AB} = \overline{AB} + \overline{AB} + \overline{AB} = \overline{A} + B$$



Semplificazioni notevoli



Dimostrare che: $(A + \overline{B})(B + C) = \overline{AB} + AC + BC$

Dimostrare che: $\overline{A} + \overline{AB} = \overline{A} + B$



Esempio di semplificazione algebrica (esercizio)



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

Raccogliendo $\bar{B}\bar{C}$:

$$(\bar{A} + A)\bar{B}\bar{C} + ABC =$$

Proprietà dell'inverso: " $\bar{A} + A = I$ "

$$= 1\bar{B}\bar{C} + ABC =$$

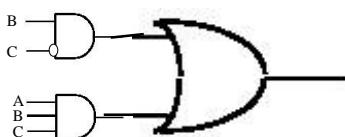
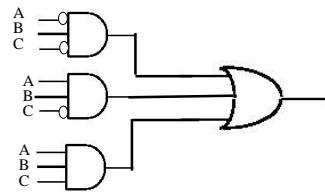
Proprietà dell'identità: " $IB = B$ "

$$= \bar{B}\bar{C} + ABC =$$

Dalla slide precedente:

$$= B(\bar{C} + AC) = B(\bar{C} + A)$$

sono 3 circuiti equivalenti:



Esempi di manipolazione algebrica



$$Y = !xyv + yz + !y!zv + !xy!v + x!yv =$$

$$Y = A !B !C + A B C + A B !C + A !B C = A$$

Somma di prodotti di 3 variabili: A, B, C (inverso dell'esercizio precedente):



Esercizi



- Calcolare le TT per le seguenti funzioni

$$Y = DA + AC + !B$$

$$Y = A + B + C + D$$

$$Y = !D!ABC + !DABC + !D!AB!C + !DAB!C$$

- Trasformare in funzioni equivalenti le seguenti funzioni, semplificandole:

$$Y = !(ABCD)$$

$$Y = !(DA) + !(B + !C)$$



Sommario



I circuiti combinatori.

Dall'espressione logica al circuito. Semplificazione algebrica.

Dalla tabella della verità al circuito: la prima forma canonica: SOP.

Criteri di ottimalità.

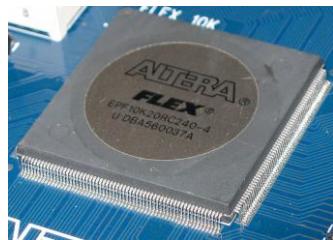


FPGA



Field Programmable Gate Array (Matrice di porte logiche programmabili sul campo)

2020 mercato di 8.5 miliardi di dollari.



La struttura di un FPGA è in generale costituita da una matrice di blocchi logici configurabili, detti CLB (*Configurable Logic Blocks*), connessi fra loro attraverso interconnessioni programmabili. Ai margini di tale matrice vi sono i blocchi di ingresso/uscita, detti IOB (*Input Output Block*).

Da circuiti logici relativamente semplici fino a microprocessori interi:

<https://www.arm.com/resources/designstart/designstart-fpga>

<https://venturebeat.com/2018/10/01/xilinx-will-use-arm-cores-in-fpga-chips/>



Dall'espressione logica / tabella della verità al circuito



Definizione della funzione logica

Semplificazione e fitting sulla FPGA

Programmazione in linguaggi specifici (Verilog e VHDL)

Programmazione mediante POD che si collegano ai piedini di programmazione.



Insiemi di porte logiche le cui connessioni vengono definite (bruciate) attraverso un pod di programmazione collegato con USB a un PC. Questo consente di personalizzare il circuito logico e di implementare il circuito specificato via SW dal PC.

Da circuiti logici relativamente semplici fino a microprocessori interi:

<https://www.arm.com/resources/designstart/designstart-fpga>

<https://venturebeat.com/2018/10/01/xilinx-will-use-arm-cores-in-fpga-chips/>



Funzione come espressione logica o come tabella delle verità



$$Y = \overline{A} \overline{B} \overline{C} + AB$$

$$Y = 1$$

A B C	Y
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

iff

$$A = 0 \ B = 1 \ C = 0$$

OR

$$A = 1 \ B = 1 \ C = 0$$

OR

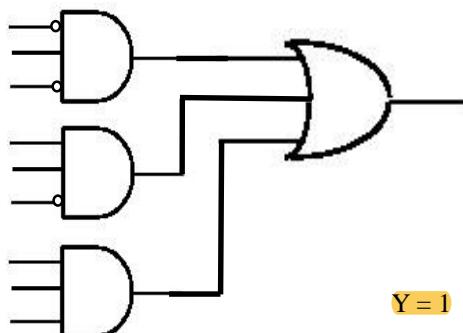
$$A = 1 \ B = 1 \ C = 1$$



Circuito associato



3 AND CHE VANNO IN OR



$$Y = 1$$

A B C	Y
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

iff

$$A = 0 \ B = 1 \ C = 0$$

OR

$$A = 1 \ B = 1 \ C = 0$$

OR

$$A = 1 \ B = 1 \ C = 1$$



SOP

La prima forma canonica



$$Y = \bar{A} \bar{B} \bar{C} + AB$$

Implicante: prodotto delle variabili (in forma asserita o negata) per le quali la funzione vale 1 **non necessariamente tutte le variabili**

A	B	C	Y
0	0	0	0
1	0	0	0
2	0	1	1
3	0	1	1
4	1	0	0
5	1	0	0
6	1	1	1
7	1	1	1

Mintermine, m_j : un implicante che contiene tutte le N variabili della funzione (e.g. ABC) associato agli 1 della funzione.

j indica il numero progressivo in base 10.

$$\text{Prima forma canonica: } F = \sum_{i=1}^Q m_i$$

$$0 \leq Q \leq 2^N$$

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



Mintermini e Maxtermini



$$Y = \bar{A} \bar{B} \bar{C} + AB$$

Mintermine, m_j : un implicante che contiene tutte le N variabili della funzione (e.g. ABC) associato agli 1 della funzione.

j indica il numero progressivo in base 10.

$$F = \bar{A} \bar{B} \bar{C} + A \bar{B} \bar{C} + A \bar{B} C$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Maxtermine, M_k : contiene tutte le N variabili della funzione ed è tale che il loro prodotto logico è uno 0 della funzione.



Funzione come espressione logica nella seconda forma canonica



$$Y = \overline{A} \overline{B} \overline{C} + AB$$

$$Y = 1$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

iff

$NOT(A = 0 B = 0 C = 0)$

AND

$NOT(A = 0 B = 0 C = 1)$

AND

$NOT(A = 0 B = 1 C = 1)$

AND

$NOT(A = 1 B = 0 C = 0)$

AND

$NOT(A = 1 B = 0 C = 1)$



Dall'espressione algebrica alla SOP (Sum Of Product)



- Passare attraverso la tabella della verità:

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- Manipolazione algebrica:

$$\begin{aligned}
 Y &= \overline{\overline{A}} \overline{\overline{B}} \overline{\overline{C}} + AB = \\
 &= \overline{\overline{A}} \overline{\overline{B}} \overline{\overline{C}} + AB(C + \overline{C}) = \\
 &= \overline{\overline{A}} \overline{\overline{B}} \overline{\overline{C}} + ABC + ABC = m_2 + m_6 + m_7
 \end{aligned}$$



La SOP è la prima forma canonica



- La forma canonica di una funzione è la somma dei suoi mintermini.
- Qualunque funzione è esprimibile in forma canonica.

Esempio: $Y = f(A,B,C,D) = \text{AC} + \text{BC} + \text{ABC}$

$$= \overline{\text{A}}(\overline{\text{B}} + \text{B})\overline{\text{C}}(\overline{\text{D}} + \text{D}) + (\text{A} + \overline{\text{A}})\text{BC}(\overline{\text{D}} + \text{D}) + \text{ABC}(\overline{\text{D}} + \text{D}) \\ = \overline{\text{ABCD}} + \overline{\text{ABCD}}$$

La stessa espressione si ricaverebbe dalla tabella della verità:

$$\begin{aligned} Y = & \overline{\text{A}}\overline{\text{B}}\overline{\text{C}}\overline{\text{D}} + \overline{\text{A}}\text{B}\overline{\text{C}}\overline{\text{D}} + \overline{\text{A}}\text{B}\overline{\text{C}}\text{D} + \overline{\text{A}}\text{B}\text{C}\overline{\text{D}} + \overline{\text{A}}\text{B}\text{C}\text{D} + \text{A}\overline{\text{B}}\overline{\text{C}}\overline{\text{D}} + \text{A}\overline{\text{B}}\overline{\text{C}}\text{D} + \text{A}\overline{\text{B}}\text{C}\overline{\text{D}} + \text{A}\overline{\text{B}}\text{C}\text{D} = \\ & \overline{\text{ABC}} + \overline{\text{ABC}} = \\ & \overline{\text{ABC}} + \text{C}(\text{A} + \overline{\text{A}}\text{B}) = \text{ABC} + \text{AC} + \text{BC} \end{aligned}$$



Perchè SOP è una forma canonica



- **Forma universale mediante la quale è possibile rappresentare qualunque funzione booleana.**

• In generale una forma canonica non è una forma ottima, ma un punto di partenza per l'ottimizzazione.

• Si basa su componenti caratterizzanti la struttura della funzione (mintermine), che traducono le condizioni logiche espresse dalla funzione.

Minternine, m_i :

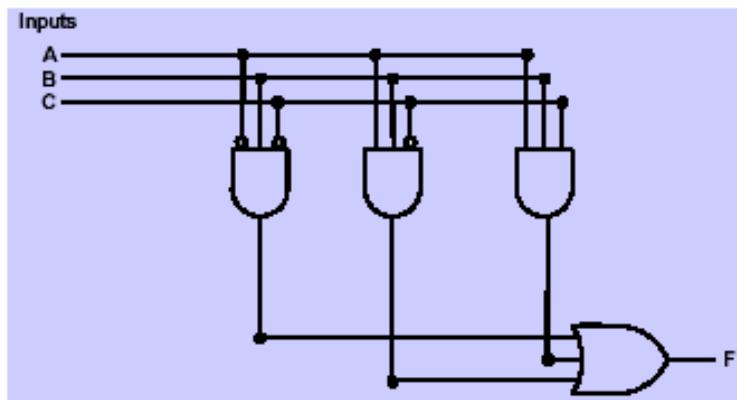
- E' una funzione booleana a n ingressi che vale 1 in corrispondenza della sola i-esima configurazione di ingresso.
- Al massimo, 2^n mintermini per ogni n variabili.
- ogni mintermine è rappresentabile con un AND con n ingressi.



Il circuito della prima forma canonica: SOP



$$F = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$$



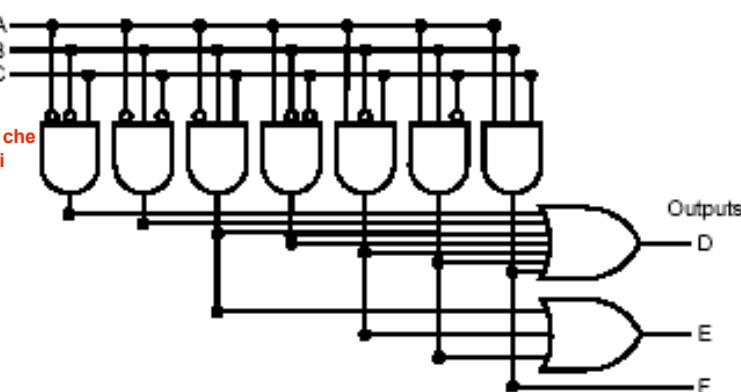
SOP a più uscite



Inputs

A
B
C

primo livello che
rappresenta i
mintermini



Riutilizzo alcune “parti”, in questo caso alcuni mintermini:
quelli che sono contenuti in D, E, F.

Ricavare la funzione in forma di tabella della verità’



Dalla SOP al circuito: osservazioni



- Dalla forma canonica (somma di mintermini) è facile passare al circuito:
 - Ogni mintermine è un AND
 - Tutti gli AND entrano in un OR
- Implementazione regolare
- Solo due livelli di porte
- Blocchi generali personalizzabili purché ci sia un numero sufficiente di componenti elementari.



Sommario



I circuiti combinatori.

Dall'espressione logica al circuito. Semplificazione algebrica.

Dalla tabella della verità al circuito: la prima forma canonica: SOP.

Criteri di ottimalità.



Dall'espressione logica al circuito



Ad ogni espressione logica corrisponde un circuito, ad ogni circuito corrisponde una tabella delle verità, ad ogni tabella della verità, in generale, **non corrisponde** un unico circuito possibile.

- Esistono più espressioni tra loro equivalenti: 2 espressioni sono equivalenti se hanno la stessa tabella di verità.
- Quale è la “migliore”?
- È possibile trovare un metodo di semplificazione sfruttando le proprietà dell’algebra booleana.
- Esistono tecniche automatiche o semi-automatiche di semplificazione.



Valutazione della semplicità di un circuito

circuito semplice = circuito con poche porte logiche

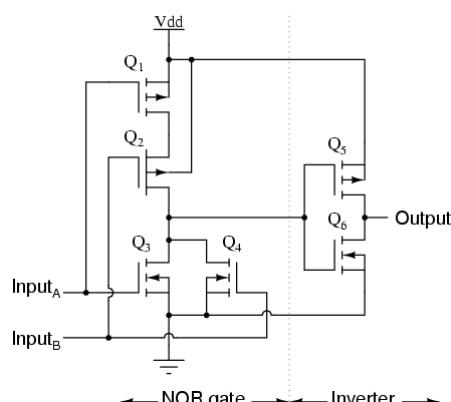


Area (numero di porte) = “ampiezza”

Tempo di commutazione (numero di transistor attraversati = “profondità”)

Soddisfazione di vincoli, potenza dissipata, facilità di debug...

CMOS OR gate

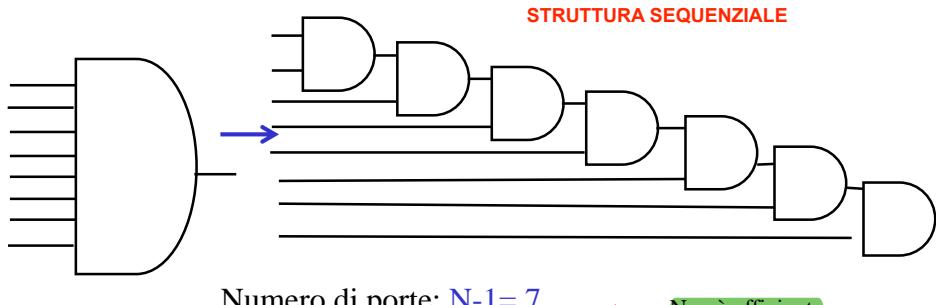




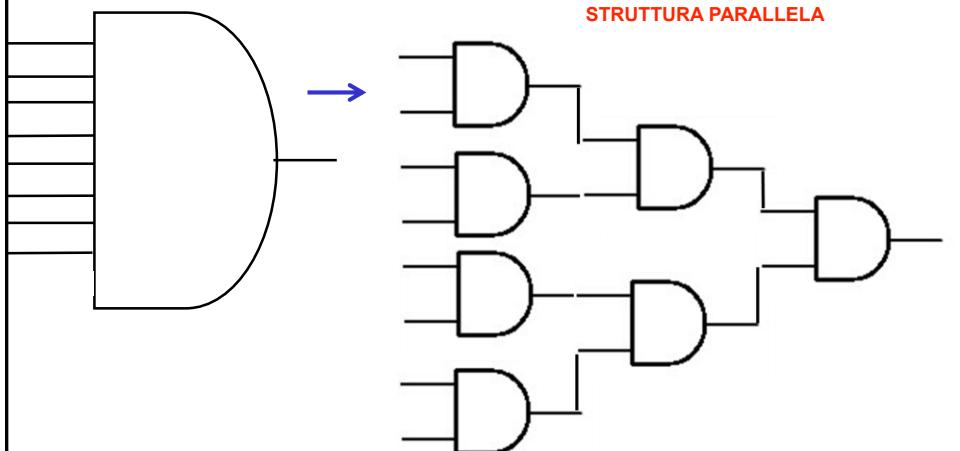
Esempio di trasformazione in un'implementazione con porte a 2 ingressi di un AND a 5 ingressi



- Gli elementi costruttivi di base tipici sono porte a 2 ingressi
 - Porta a N ingressi \rightarrow N-1 porte a 2 ingressi



Parallelizzazione





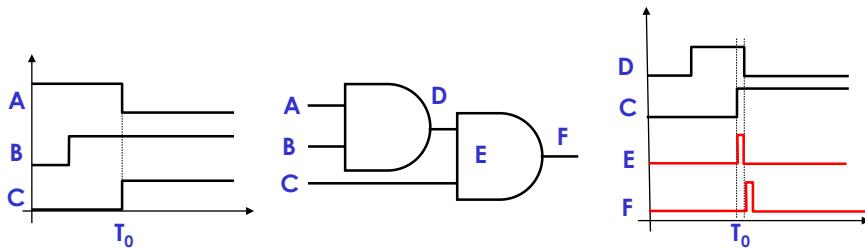
Transistori



- Ogni circuito logico è caratterizzato da un **tempo di commutazione**
 - Più porte devo attraversare, più è lungo il tempo della **transizione del circuito nel suo complesso**.

CAMMINO CRITICO

- max numero di porte da attraversare da ingresso a uscita
- Non si contano gli inverters (inclusi nelle porte)



A e C commutano contemporaneamente in T_0 , E raggiunge il valore corretto dopo un tempo $2 \Delta T$ (la commutazione di D segue la commutazione di B con un ritardo ΔT).



SOP dell'OR



Sintetizziamo la funzione OR come SOP

$$Y = \overline{A}\overline{B} + \overline{A}B + AB \quad \text{Complessità} = 5 - \text{Cammino critico} = 3$$

Semplifico:

$$Y = \overline{A}\overline{B} + \overline{A}B + AB = B(\overline{A} + A) + \overline{A}B = B + \overline{A}B = B + A = A + B = \text{OR}(A,B)$$

A	B	Y
0	0	0
→	0	1
→	1	1
→	1	1



Sommario



I circuiti combinatori.

Dall'espressione logica al circuito. Semplificazione algebrica.

Dalla tabella della verità al circuito: la prima forma canonica: SOP.

Criteri di ottimalità.



Circuiti combinatori notevoli

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimenti sul Patterson Hennessy: Sezione B3.



Sommario



Implementazione circuitale mediante PLA o ROM

Circuiti combinatori notevoli



Circuiti combinatori



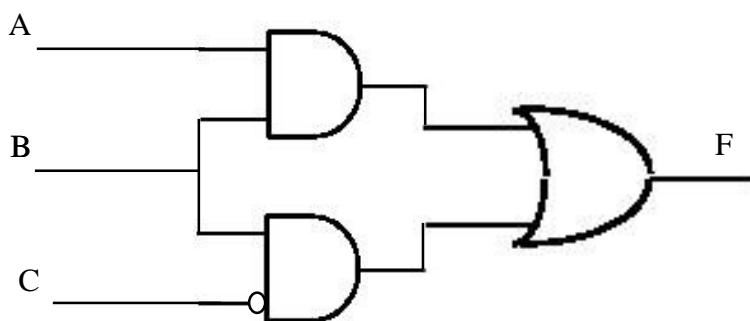
- Circuiti logici digitali in cui le operazioni (logiche) dipendono solo da una combinazione degli input.
- Circuiti senza memoria. Ogni volta che si inseriscono in ingresso gli stessi valori, si ottengono le stesse uscite. Il risultato non dipende dallo stato del circuito.
- I circuiti combinatori descrivono delle funzioni Booleane. Queste funzioni si ottengono combinando tra loro (in parallelo o in cascata) gli operatori logici: **NOT, AND, OR**.
- Il loro funzionamento può essere descritto come **tavola della verità**.
- Come nelle funzioni algebriche, il risultato è aggiornato immediatamente dopo il cambiamento dell'input (si suppone il tempo di commutazione trascurabile, tempo di attesa prima di guardare l'output sufficientemente ampio per permettere a tutti i circuiti la commutazione).



Funzione come circuito



$$F = A B + B \bar{C}$$



Le operazioni algebriche hanno un ordine di precedenza. **NOT AND OR**



Razionale della prima forma canonica



$$F = A B + B \bar{C}$$

A B C	F
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

$$F = \bar{A} \bar{B} \bar{C} + A \bar{B} \bar{C} + A B C$$

$$F = 1$$

iff

$$A = 0 \ B = 1 \ C = 0$$

OR

$$A = 1 \ B = 1 \ C = 0$$

OR

$$A = 1 \ B = 1 \ C = 1$$



La prima forma canonica



- Esiste un metodo per ricavare automaticamente un circuito che implementi una tabella di verità?
- Una **forma canonica** garantisce di poter realizzare una qualunque tabella di verità con solo due livelli di porte OR, AND e NOT;
- Somme di Prodotti (SOP) è la prima forma canonica (OR di AND):

A B C	F
0 0 0	0
0 0 1	0
0 1 0	1 m_2
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1 m_6
1 1 1	1 m_7

$$F = m_2 + m_6 + m_7$$

$$F = \bar{A} B \bar{C} + A \bar{B} \bar{C} + A B C$$

mintermini



Livelli di astrazione intermedi: anche per le linee



descrizione
a livello di
astrazione
più alto



descrizione
a livello di
astrazione
più basso



Tipi di circuiti che implementano le SOP



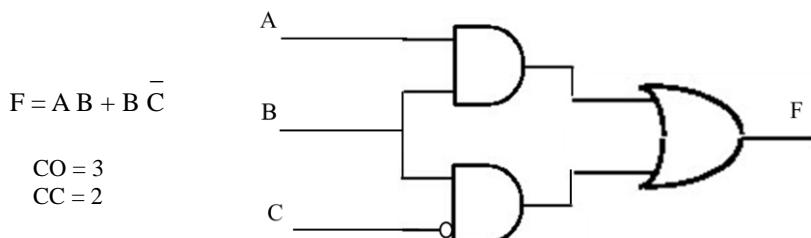
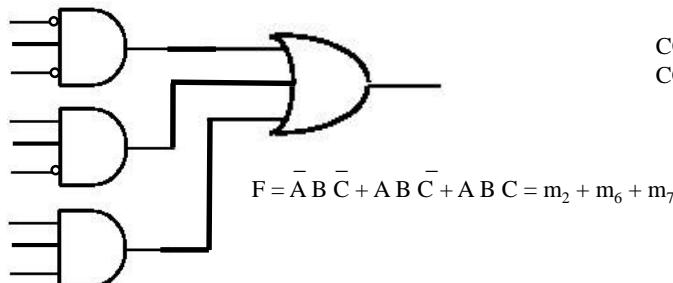
In generale abbiamo funzioni logiche booleane multi-input / multi-output.

I circuiti complessi si ottengono collegando «piccoli» blocchi funzionali, dove ciascun blocco implementa una funzione logica.

- **Logica distribuita.**
- **PLA:** Programmable Logic Array: matrici regolari AND e OR in successione, personalizzabili dall'utente.
- **ROM:** Read Only Memory circuiti ad hoc che implementano una particolare funzione in modo irreversibile.



Logica distribuita



A.A. 2023-2024

9/47

<http://borghese.di.unimi.it/>

PLA (Programmable Logical Array)



- Architettura a 2 livelli:
 - Un primo livello di AND
 - Un secondo livello di OR



- La matrice degli AND ha n linee di ingresso: ciascuna porta ha in ingresso le **n linee** e il loro complemento.
- L'utente specifica per ogni porta AND a disposizione se la linea in ingresso entra direttamente o dopo una negazione.
 - Crea la matrice dei mintermini, bruciando in ingresso alle porte AND le linee che non servono.
- Le uscite della matrice AND entrano nella matrice OR programmata come la precedente per ottenere l'OR dei mintermini della funzione.
 - Si utilizza una porta OR per ogni funzione calcolata (m OR per **m linee** di uscita)

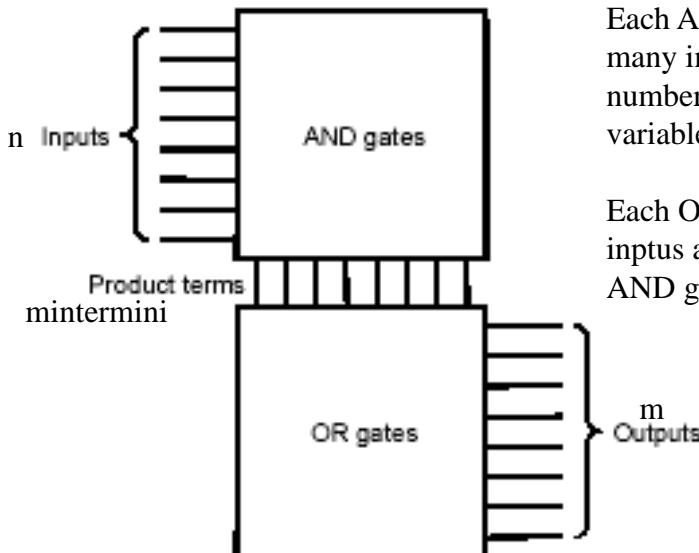


A.A. 2023-2024

<http://borghese.di.unimi.it/>



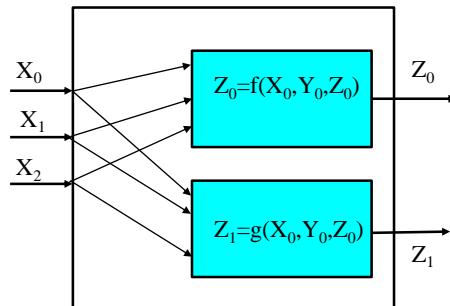
Struttura di una PLA



Sintesi separata delle uscite



X_0	X_1	X_2	Z_0	Z_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



$$Z_0 = \overline{X_0} \overline{X_1} X_2 + \overline{X_0} X_1 \overline{X_2} + X_0 \overline{X_1} \overline{X_2} + X_0 X_1 X_2 = m_1 + m_2 + m_4 + m_7$$

$$Z_1 = \overline{X_0} X_1 X_2 + X_0 \overline{X_1} X_2 + X_0 X_1 \overline{X_2} + X_0 X_1 X_2 = m_3 + m_5 + m_6 + m_7$$

Non è efficiente perché non si sfruttano le operazioni in comune alle due funzioni.



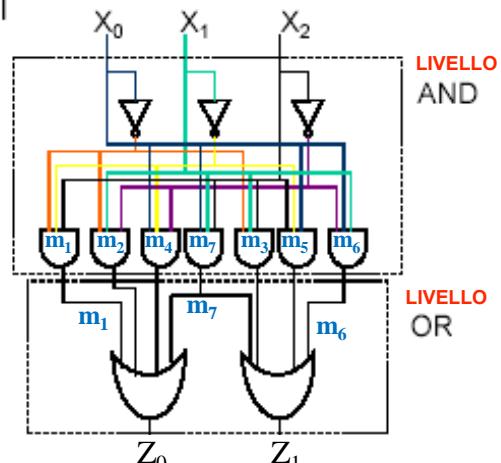
Sintesi mediante PLA



- Realizzare con un PLA la funzione descritta dalla seguente TT

	X_0	X_1	X_2	Z_0	Z_1
	0	0	0	0	0
$m_1 \rightarrow$	0	0	1	1	0
m_2	0	1	0	1	0
m_3	0	1	1	0	1
m_4	1	0	0	1	0
m_5	1	0	1	0	1
$m_6 \rightarrow$	1	1	0	0	1
$m_7 \rightarrow$	1	1	1	1	1

m₇ mintermine comune



$$Z_0 = \overline{X_0} \overline{X_1} X_2 + \overline{X_0} X_1 \overline{X_2} + X_0 \overline{X_1} \overline{X_2} + X_0 X_1 X_2 = m_1 + m_2 + m_4 + m_7$$

$$\text{A.. } Z_1 = \overline{X_0} X_1 X_2 + X_0 \overline{X_1} X_2 + X_0 X_1 \overline{X_2} + X_0 X_1 X_2 = m_3 + m_5 + m_6 + m_7$$

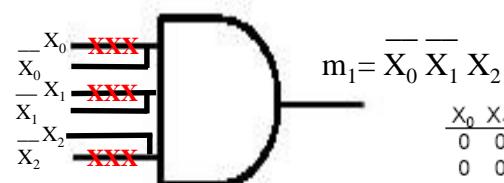
ii.it\



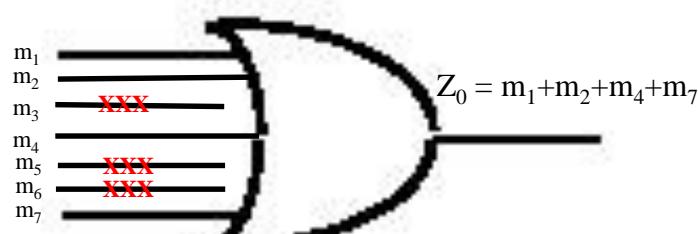
Configurazione di una PLA



Costruzione di una porta AND (m_1)



	X_0	X_1	X_2	Z_0	Z_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



A.A. 202

http://borgheze.di.unimi.it\



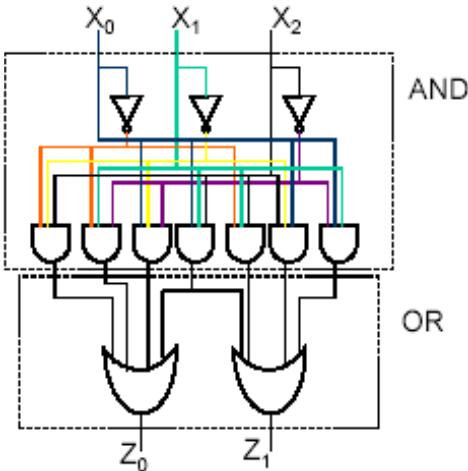
Complessità e cammino critico



descrizione ad alto livello

3 / → un circuito (3 ingressi 2 uscite) → 2 /

X_0	X_1	X_2	Z_0	Z_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



7 AND 2 OR

$$\text{Complessità} = 7 \cdot 2 + 2 \cdot 3 = 20$$

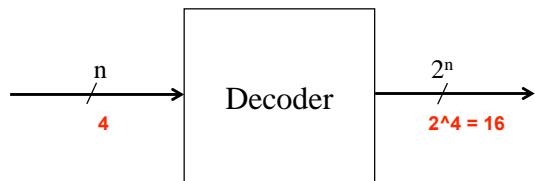
$$\text{Cammino Critico} = \frac{7}{3-1} + \frac{2}{4-1} = 5$$



Decodificatore (decoder)



- E' caratterizzato da n linee di input e 2^n linee di output



- il numero binario espresso dalla configurazione delle linee di input è usato per assegnare la **sola** linea di output di ugual indice (tutte le altre portano 0) si alza 1 SOLA linea di uscita corrispondente a quello che mi arriva in input

- es.: con 4 linee di input e 16 di output (da 0 a 15), se in ingresso arriva il valore 0110 (6 decimale), in uscita si alza la linea di indice 5 (la sesta!), tutte le altre hanno valore 0

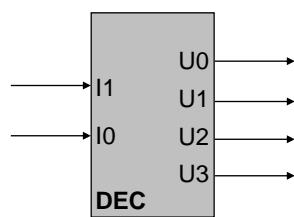
- utilizzato per indirizzare la memoria ad esempio (cf. ROM)



Decoder a 2 ingressi



Tabella della verità



I1	I0	U0	U1	U2	U3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

in ingresso 0 decimale voglio attivo U0
 in ingresso 1 decimale voglio attivo U1
 in ingresso 2 decimale voglio attivo U2
 in ingresso 3 decimale voglio attivo U3

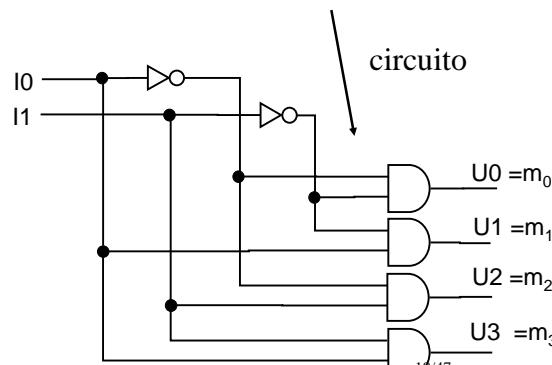
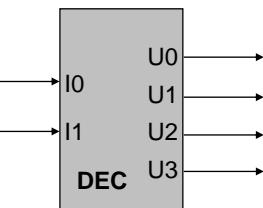


Decoder a 2 ingressi: realizzazione



I1	I0	U0	U1	U2	U3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$$U_i = f_i(I_0, I_1)$$



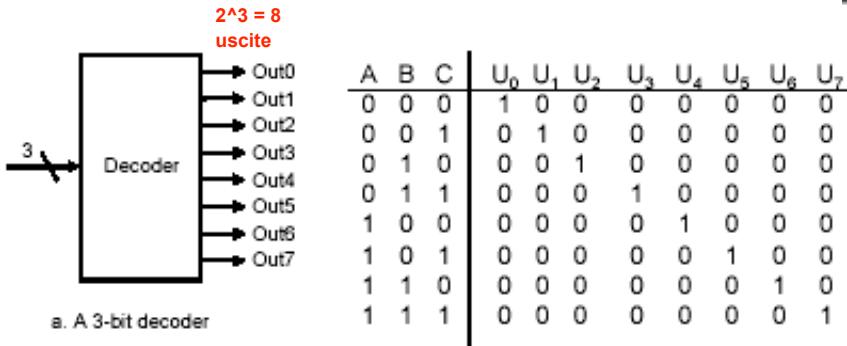
circuito

Tabella della verità

I1	I0	U0	U1	U2	U3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decoder a 3 ingressi



Le funzioni di uscita sono 2^n per n input:

$$U_0 = \sim A \sim B \sim C$$

...

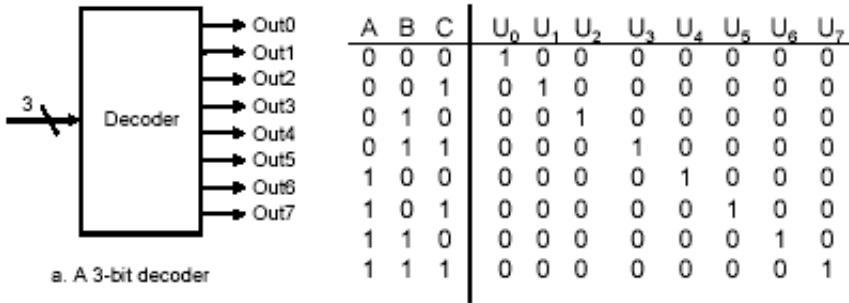
$$U_7 = A BC$$

$$U_j = m_j$$

I mintermini sono AND a 3 ingressi



Valutazione del decoder a 3 ingressi



Le funzioni di uscita sono 2^n per n input:

$$U_0 = \sim A \sim B \sim C$$

...

$$U_7 = A BC$$

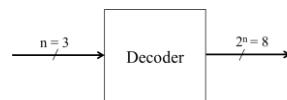
$$U_j = m_j$$

8 AND a 3 ingressi I mintermini sono AND a 3 ingressi
ogni AND ha complessità 2

Complessità: $8 \times 2 = 16$

ogni AND ha 3 ingressi

Cammino critico: 2





Rappresentazione circuitale mediante ROM



- Read Only Memory, memoria di sola lettura.
Funge anche da modulo combinatorio a uscita multipla.
- rappresenta indirizzo/indice rappresenta dato
• n linee di ingresso, m linee di uscita (ampiezza)
a ciascuna delle 2^n (altezza) configurazioni di ingresso
(parole di memoria) è associata permanentemente una
combinazione delle m linee di uscita.
- l'input seleziona la parola da leggere di m bit, che appare in uscita
- L'input funziona da indice all'interno della ROM.
Viene realizzato con un decoder n-a-2ⁿ seguito da
una matrice di m porte OR.

X_0	X_1	X_2	Z_0	Z_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

A.A. 2023-2024

21/47

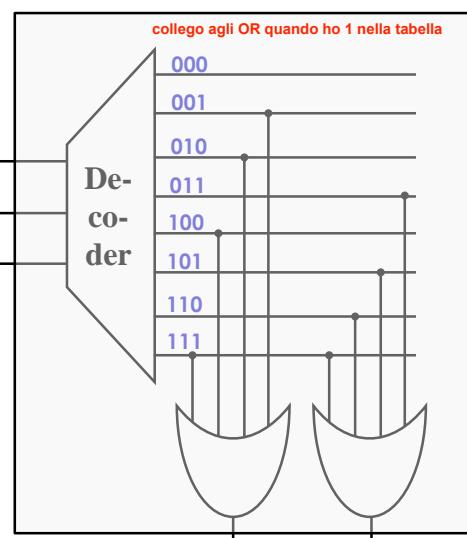


ROM - esempio



	X_0	X_1	X_2	Z_0	Z_1
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

indirizzo
 X_0
 X_1
 X_2

**SOP**

$$Z_0 = \overline{X_0} \overline{X_1} X_2 + \overline{X_0} X_1 \overline{X_2} + X_0 \overline{X_1} \overline{X_2} + X_0 X_1 X_2$$

$$Z_1 = \overline{X_0} X_1 X_2 + X_0 \overline{X_1} X_2 + X_0 X_1 \overline{X_2} + X_0 X_1 X_2$$

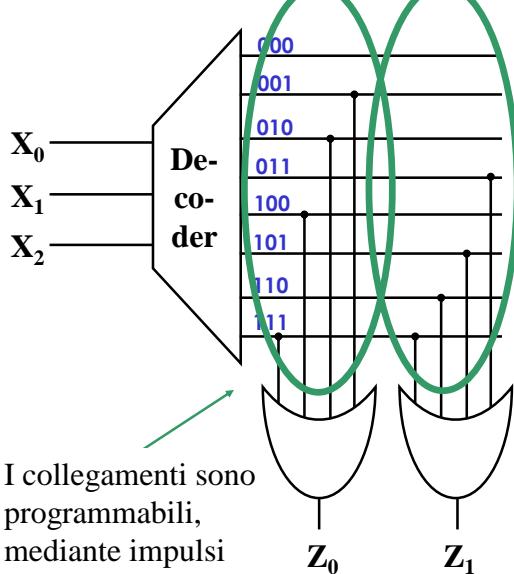


EEPROM



X₀	X₁	X₂	Z₀	Z₁
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

A.A. 2023-2024



<http://borgheze.di.unimi.it/>

brucio connessioni non utili



Valutazione ROM



X₀	X₁	X₂	Z₀	Z₁
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

DECODER HA
COMPLESSITÀ 16
2 OR A 4
INGRESSI

Complessità: $8 \cdot 2 + 2 \cdot 3 = 22$

Cammino critico: $2 + 2 = 4$

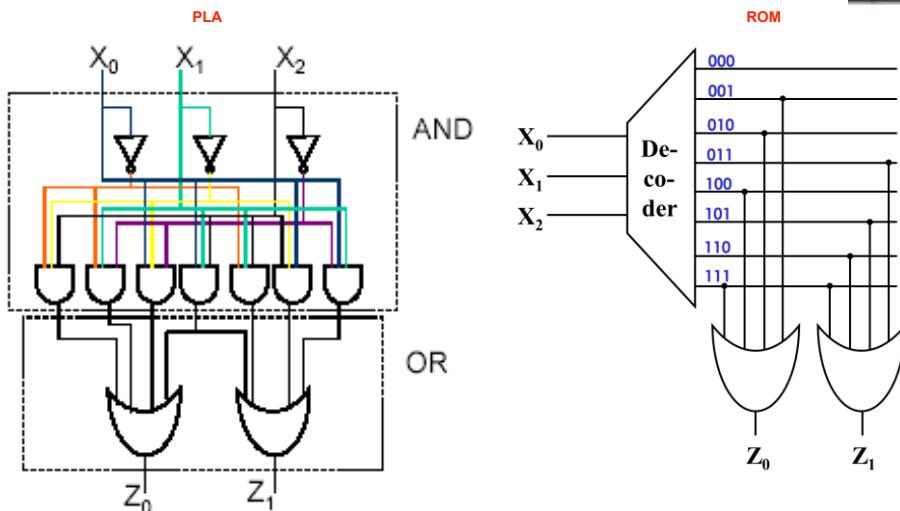
24/47

<http://borgheze.di.unimi.it/>

A.A. 2023-2024



Confronto PLA / ROM



Confronto PLA - ROM



ROM – fornisce un’uscita per ognuna delle combinazioni degli ingressi (mintermini e maxtermini). Decoder con 2^n uscite, dove n è il numero di variabili in ingresso alla ROM. Crescita esponenziale delle uscite. Approccio più generale. Può implementare una qualsiasi funzione, dato un certo numero di input e output.

PLA – contiene solamente i mintermini in uscita al primo livello.

Il loro numero cresce meno che esponenzialmente. **PLA COMPLESSITÀ MINORE RISPETTO A ROM**

E’ più veloce una PLA o una ROM? Valutare in termini di cammino critico.

FPGA – Maggiore libertà. E’ costituita da celle: moduli di PLA. E’ una rete di strutture a 2 livelli.



Esercizi sulla PLA



Realizzare mediante PLA con 3 ingressi con il numero adeguato di linee interne:

- la funzione maggioranza.
- la funzione che vale 1 se e solo se 1 solo bit di ingresso vale 1
- un decoder
- la funzione che vale 0 se l'input è pari, 1 se dispari
- la funzione che calcola i multipli di 3 (con 4 ingressi)

Realizzare le stesse funzioni con una ROM o con logica distribuita e operare un confronto. Fare una valutazione comparata in termini di complessità e cammino critico.



Sommario



Implementazione circuitale mediante PLA o ROM.

Circuiti combinatori notevoli.



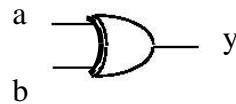
Porta XOR



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

$$SOP: \quad y = \overline{a}\overline{b} + a\overline{b}$$

OR esclusivo RISULTATO = 1 SOLO QUANDO UNO DEI DUE INPUT = 1
Disuguaglianza di 2 ingressi



Complessità e cammino critico di 1 porta logica

$$y = a \oplus b$$



Uscite indifferenti di un tabella delle verità



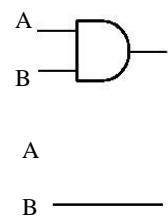
X = “don’t care” = {0,1} a seconda di quello che conviene.
USCITA INDIFFERENTE

A	B	F
0	0	0
0	1	X
1	0	0
1	1	1

Ho 2 possibilità:

$$1) X = 0 \quad F = AB$$

$$2) X = 1 \quad \begin{matrix} SOP \\ F = \overline{A}\overline{B} + AB = B \end{matrix}$$



Diminuisce il numero di porte e si accorta il cammino critico.

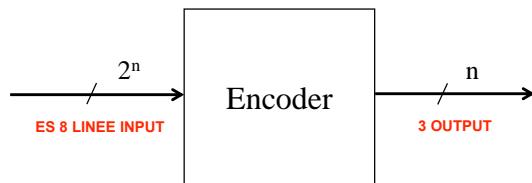


Codificatore (encoder)

CONTRARIO DEL DECODER



- E' caratterizzato da n linee di input e $\text{ceil}(\log_2 n)$ linee di output



- Una sola linea di ingresso può essere attiva.
- il numero binario espresso dalla configurazione delle linee di output rappresenta la linea di ingresso attiva.
- es.: con 16 linee di input e 4 di output, se in ingresso arriva il valore 0000 0100 0000 0000, in uscita leggiamo il numero 10.



La funzione encoder



ABCD	Y ₁ Y ₂	Codifica quattro linee in ingresso: 0,1,2,3
0000	XX	Suppongo X = 0
0001	00	CIRCUITO SEMPLIFICATO
0010	01	$Y_1 = \overline{ABCD} + \overline{ABCD} = CD (A \oplus B)$
0011	XX	$m_4 + m_8$
0100	10	
0101	XX	
0110	XX	
0111	XX	$Y_2 = \overline{ABCD} + \overline{ABCD} = B D (A \oplus C)$
1000	11	$m_2 + m_8$
1001	XX	
1010	XX	Complessità: $2 \cdot 3 + 1 = 7$
1011	XX	Complessità circuito semplificato: 3
1100	XX	
1101	XX	Cammino critico: $2 + 1 = 3$
1110	XX	Cammino critico circuito semplificato: 2
1111	XX	Si può fare di meglio? Come è conveniente impostare le X? http://borgheze.di.unimi.it/



La funzione encoder: ottimizzazione - I



ABCD	Y ₁ Y ₂	
0000	0 X	Consideriamo la prima funzione, Y ₁
0001	0 0	
0010	0 1	Specifichiamo in modo opportuno i valori di uscita indifferenti, X
0011	0 X	
0100	1 0	VOGLIO CREARE DEI MINTERMINI CHE SUCCESSIVAMENTE SI SEMPLIFICHERANNO
0101	1 X	
0110	1 X	
0111	1 X	
1000	1 1	$Y_1 = \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D} + \bar{A} B \bar{C} \bar{D} +$
1001	1 X	$\bar{A} B C D + A \bar{B} \bar{C} \bar{D} + A \bar{B} C \bar{D} + A B \bar{C} \bar{D} +$
1010	1 X	$\bar{A} B C D = (A \oplus B)$
1011	1 X	
1100	0 X	Complessità: 1
1101	0 X	Cammino critico: 1
1110	0 X	
1111	0 X	



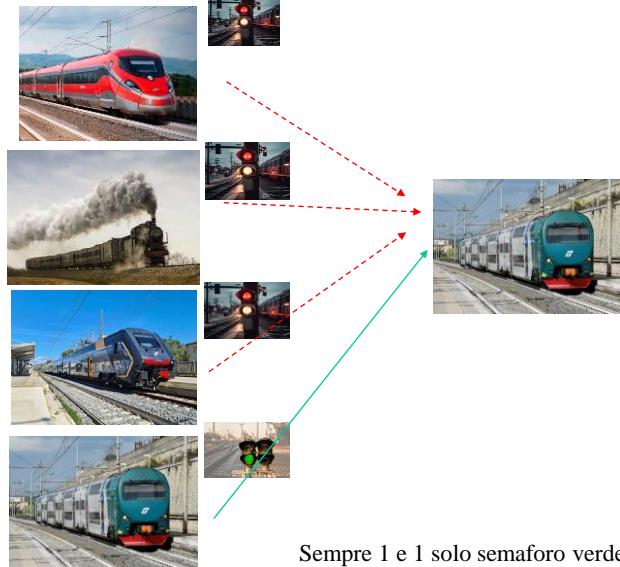
La funzione encoder: ottimizzazione - II



ABCD	Y ₁ Y ₂	
0000	0 0	Consideriamo la prima funzione, Y ₁
0001	0 0	
0010	0 1	Specifichiamo in modo opportuno i valori di uscita indifferenti, X
0011	0 1	
0100	1 0	
0101	1 0	
0110	1 1	
0111	1 1	
1000	1 1	$Y_2 = \bar{A} \bar{B} \bar{C} \bar{D} + \bar{A} \bar{B} C \bar{D} + \bar{A} B \bar{C} \bar{D} + \bar{A} B C \bar{D} +$
1001	1 1	$A \bar{B} \bar{C} \bar{D} + \bar{A} B \bar{C} \bar{D} + A \bar{B} C \bar{D} + A B \bar{C} \bar{D} = (A \oplus C)$
1010	1 0	
1011	1 0	
1100	0 1	Complessità: 1
1101	0 1	Cammino critico: 1
1110	0 0	
1111	0 0	



Multiplexer



Multiplexer (selettore)



- Ha la funzione di un sistema di semafori.
- E' caratterizzato da:
 - n linee di input (data) – $\{x_i\}$ **N BINARI DELLA STAZIONE**
 - k linee di controllo (selezione) .{S}. **SPECIFICA IL SEGNALE DEL SEMAFORO CHE VA ACCESO**
 - 1 linea di output. **PASSA UN SOLO TRENO ALLA VOLTA**
- In base alla linea di controllo viene connessa all'uscita la linea di ingresso selezionata.

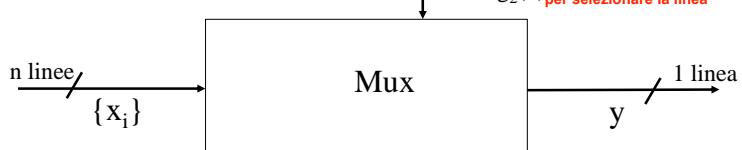
ESEMPIO STAZIONE

Quante linee di controllo, k, servono?

$$k = \text{ceil}(\log_2 n)$$

Esempio: con 4 linee di input (da 0 a 3), se sulle linee di controllo c'è 11, in uscita si avrà il valore presente sulla linea 3

\downarrow
 $S = \log_2(n)$
Selettore linee
se ho 8 linee vuol dire che avrò 3
input, quindi mi serviranno 3 bit
per selezionare la linea



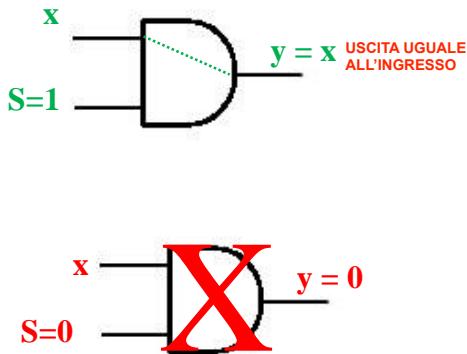


AND come semaforo (interruttore)



Il segnale di selezione S, “apre” la porta opportuna, cioè chiude il cammino opportuno.

L'AND funziona da porta di uscita (da semaforo)



S	x	y
0	0	0
0	1	0
1	0	0
1	1	1

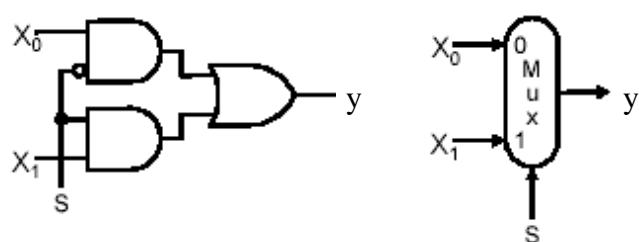


Multiplexer



S	x ₀	x ₁	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Il circuito deve portare in uscita il contenuto della linea x₀ o x₁ a seconda del valore di S





Sintesi della funzione Mux



S	x ₀	x ₁	y
y=x ₀	0	0	0
	0	0	0
	0	1	1
	0	1	1
y=x ₁	1	0	0
	1	0	1
	1	1	0
	1	1	1

$$\text{SOP: } y = \bar{S} \bar{x}_0 \bar{x}_1 + \bar{S} \bar{x}_0 x_1 + S \bar{x}_0 x_1 + S x_0 x_1$$

$$= \bar{S} x_0 + S x_1 \quad \text{cvd}$$

Il mux si comporta come interruttore:

If ($S == 1$) then

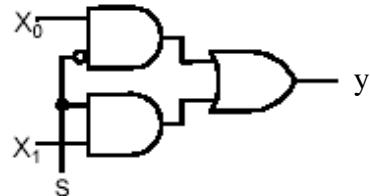
$$y = x_1$$

If ($S == 0$) then

$$y = x_0$$

Complessità: 3

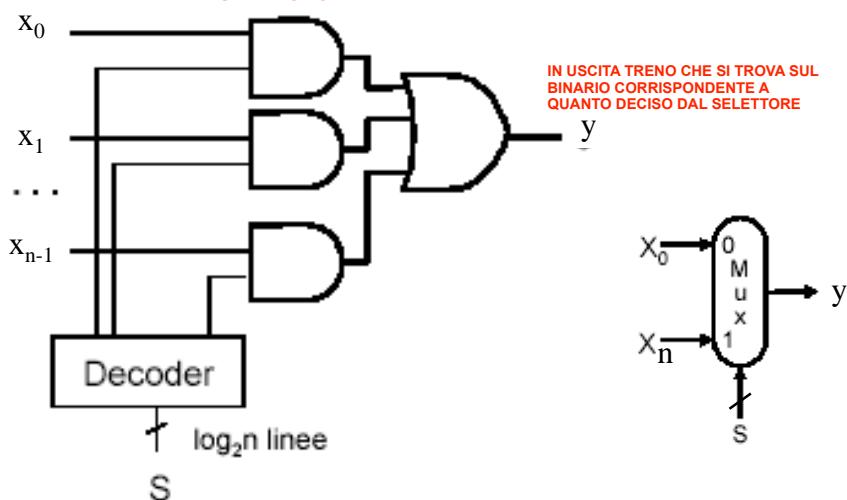
Cammino critico: 2



Mux a n vie.



OGNI BINARIO HA
ASSOCIAUTO UN
SEMAFORO



Una sola linea alla volta viene aperta dal segnale S. Le linee sono mutuamente esclusive.



Complessità di un Mux a 8 vie



8 PORTE AND

Complessità = 8

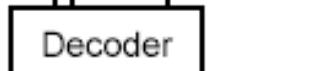


Complessità = 7



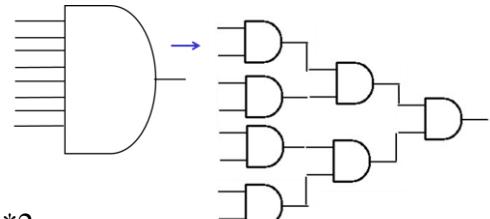
DECODER

Complessità = $16 + 8 + 7 = 31$



$\nabla \log_2 n$ linee = 3

S Complessità = $8 * 2$



Cammino critico di un Mux a 8 vie



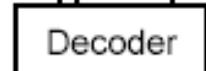
Cammino critico = 1



Cammino critico = 3



CC = $2 + 1 + 3 = 6$



Cammino critico = 2

$\nabla \log_2 n$ linee = 3

S



Il Multiplexer a un ingresso di selezione è l' "if" dell'hardware

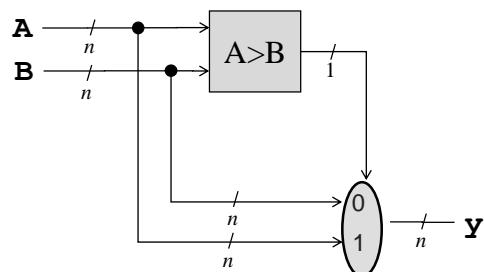


- Software

```
if A>B  
then y=A  
else y=B
```

Esecuzione
condizionata

- Hardware



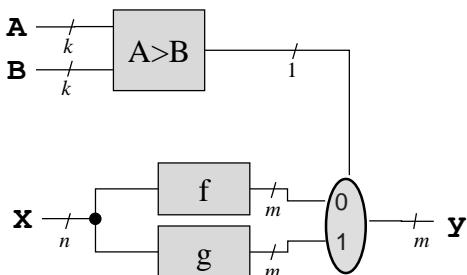
Il Multiplexer a un ingresso di selezione è l' "if" dell'hardware



- Software

```
if A>B  
then y=g(X)  
else y=f(X)
```

- Hardware



Entrambe le funzioni
sono calcolate, ma solo
un risultato alla volta
viene usato!



Comparatore a 1 bit



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

XOR

A _n	B _n	C ₀
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

$$C_0 = \overline{A_0 \oplus B_0}$$



Comparatore a n bit



- E' caratterizzato da:
 - 1 numero su n bit in ingresso, A
 - 1 numero su n bit in ingresso, B
 - 1 uscita

$$C_k = \overline{A_k \oplus B_k}$$

A ₀	B ₀	C ₀	A ₁	B ₁	C ₁
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

...

$$C = C_0 C_1 C_2 \dots \dots \quad C_{n-1} = (A_0 \oplus B_0) (A_1 \oplus B_1) \dots \dots (A_{n-1} \oplus B_{n-1})$$

Attraverso De Morgan:

$$C = \overline{C_0} + \overline{C_1} + \overline{C_2} \dots + \overline{C_{n-1}} = (A_0 \ominus B_0) (A_1 \ominus B_1) \dots \dots (A_{n-1} \ominus B_{n-1})$$



Sommario



Implementazione circuitale mediante PLA o ROM.

Circuiti combinatori notevoli.



I sommatori

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti: Appendice B5 prima parte.



Sommario



Addizionatori

Addizionatori ad anticipazione di riporto



Implementazione di funzioni algebriche



And, Or, Not per ottenere:

Operazioni algebriche (somme, prodotti, sottrazioni e divisioni) su numeri binari.

Operazioni logiche su numeri binari.



Operazione di somma



$$\begin{array}{r} \textcolor{blue}{1110} \\ \textcolor{red}{11} + \textcolor{blue}{1011} \\ \textcolor{red}{6} \quad \boxed{\textcolor{blue}{10}} = \\ \hline \textcolor{red}{17} \quad 10001 \end{array} \quad \begin{array}{l} \leftarrow \text{Riporto} \\ \leftarrow \text{Addendo 1} \\ \leftarrow \text{Addendo 2} \end{array} \quad \begin{array}{r} 111 \\ \textcolor{blue}{01011} + \\ \textcolor{blue}{00110} = \\ \hline 10001 \end{array}$$

3 Attori: addendo 1, addendo 2, riporto.

Gli addendi sono presenti all'inizio
Il riporto viene generato via via che la somma viene svolta

L'operazione viene eseguita **sequenzialmente da dx a sx**.



(Half) Adder a 1 bit

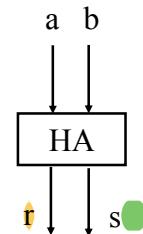


non considero riporto in ingresso

Tabella della verità della somma:

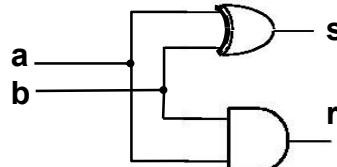
a b	somma	riporto
0 0	0	0
0 1	1	0
1 0	1	0
1 1	0	1

$$a + \\ b =$$



a b	xor
0 0	0
0 1	1
1 0	1
1 1	0

$$s = a \oplus b \text{ XOR} \\ r = ab \text{ AND}$$



La somma è diventata un'operazione logica!

Cammini critici:
Somma = 1;
Riporto = 1;

Complessità
Somma = 1 porta;
Riporto = 1 porta;



Operazione di somma



$$\begin{array}{r} 1110 \\ 1011 + \\ \hline 110 = \end{array}$$

$$10001$$

← Riporto
← Addendo 1
← Addendo 2

$$\begin{array}{r} 111 \\ 01011 + \\ \hline 00110 = \end{array}$$

$$10001$$

3 Attori: addendo 1, addendo 2, riporto.

Gli addendi sono presenti all'inizio

- Non c'è riporto in ingresso per il primo bit (HA)
- Il riporto viene generato via via che la somma viene svolta per i bit successivi al primo

Viene eseguita sequenzialmente da dx a sx.



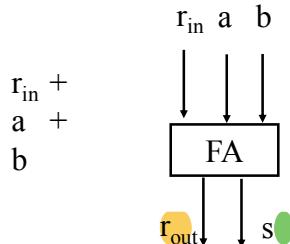
Full Adder a 1 bit

HO ANCHE BIT DI RIPORTO



Tabella della verità della somma completa:

a	b	r _{in}	somma	riporto
0	0	0	0	m ₀
0	1	0	1	m ₁
1	0	0	1	m ₂
1	1	0	0	m ₃
0	0	1	1	m ₄
0	1	1	0	m ₅
1	0	1	0	m ₆
1	1	1	1	m ₇



$$s = m_1 + m_2 + m_4 + m_7$$

$$r_{out} = m_3 + m_5 + m_6 + m_7$$



Full Adder a 1 bit – espressione logica di s



Tabella della verità della somma completa:

$$s = m_1 + m_2 + m_4 + m_7$$

a	b	r _{in}	somma	riporto
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$s = \bar{a} \bar{b} \bar{r}_{in} + a \bar{b} \bar{r}_{in} + \bar{a} b \bar{r}_{in} + a b r_{in} =$$

$$= (\bar{a}b + ab)\bar{r}_{in} + (\bar{a}b + ab)r_{in} = \text{XOR / XNOR}$$

$$= (a \oplus b)\bar{r}_{in} + (\bar{a}b + ab)r_{in} = \begin{array}{c|cc|c} & a & b & \\ \hline 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}$$

$$= (a \oplus b)\bar{r}_{in} + (\bar{a} \oplus b)r_{in} = \begin{array}{c|cc|c} & a & b & \\ \hline 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{array}$$

$$\text{XOR}(a,b) = (\bar{a}b + ab)$$

$$! \text{XOR}(a,b) = \text{XNOR}(a,b) = (\bar{a} \bar{b} + ab)$$



Full Adder a 1 bit – espressione logica di r_{out}



Tabella della verità della somma completa:

$$r = m_3 + m_5 + m_6 + m_7$$

a	b	r_{in}	somma	riporto
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$r_{out} = a b \bar{r}_{in} + \bar{a} b r_{in} + a \bar{b} r_{in} + a b r_{in} =$$

$$1) ab + (a \oplus b) r_{in}$$

$$2) a r_{in} + (a \oplus r_{in}) b$$

Quale è meglio?

dal punto di vista circuitale sono uguali -> ho AND e XOR

CONVIENE UTILIZZARE COMBINATA CON SOMMA LA NUMERO 1 PERCHÈ POSSO RIUTILIZZARE PORTA XOR

STESSA



PORTA
LOGICA
UTILIZZATA
PER SOMMA
E RIPORTO

CIRCUITO FULL ADDER PRIMA IMPLEMENTAZIONE Implementazione circuitale



$$s = (a \oplus b) \bar{r}_{in} + (a \oplus b) r_{in}$$

USO LA NUMERO 1

$$r_{out} = ab + (a \oplus b) r_{in}$$

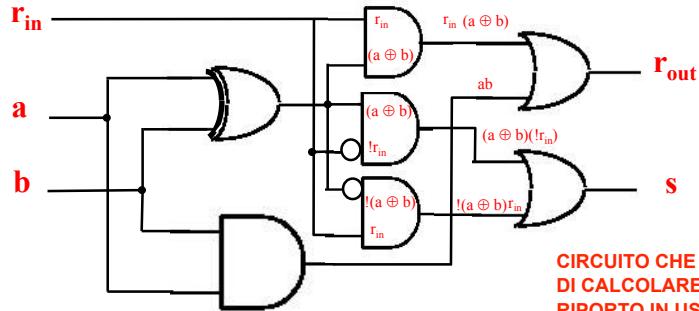
Complessità: 7 porte logiche
(Riutilizzo l'XOR 2 volte)

$$s = (a \oplus b) \bar{r}_{in} + (a \oplus b) r_{in}$$

USO LA NUMERO 2

$$r_{out} = a r_{in} + (a \oplus r_{in}) b$$

Complessità: 8 porte logiche
(Riutilizzo l'XOR 1 volta)



CIRCUITO CHE CONSENTE
DI CALCOLARE SOMMA E
RIPORTO IN USCITA

Complessità: 7 porte logiche.

Cammini critici: $s \rightarrow 3$; $r_{out} \rightarrow 3$



Semplificazione circuitale



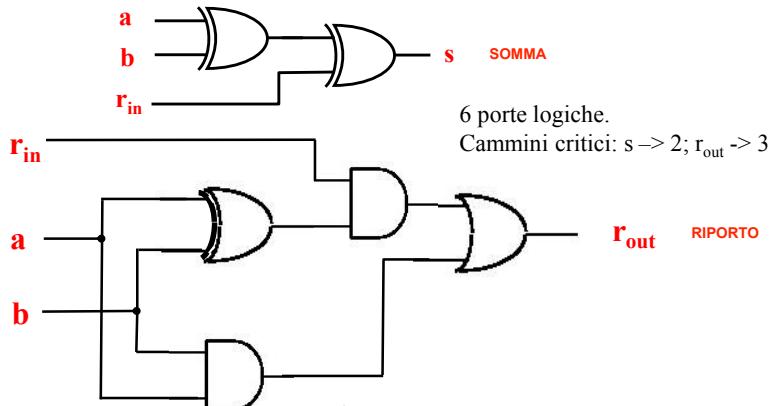
$$s = (a \oplus b) \bar{r}_{in} + (\overline{a \oplus b}) r_{in} = a \oplus b \oplus r_{in}$$

SEMPLIFICO ESPRESSIONE DI SOMMA

IPOTIZZANDO A XOR B = Z

$$z \triangleq (a \oplus b) \rightarrow z \bar{r}_{in} + \bar{z} r_{in} = (z \oplus r_{in}) = ((a \oplus b) \oplus r_{in}) = a \oplus b \oplus r_{in}$$

$$r_{out} = ab + (a \oplus b) r_{in}$$



A.A. 2023-2024

11/41

<http://borghese.di.unimi.it/>



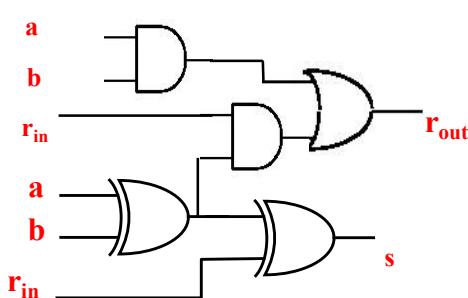
Semplificazione ulteriore



UTILIZZO UN UNICO CIRCUITO RISPETTO SLIDE PRIMA

$$s = a \oplus b \oplus r_{in}$$

$$r_{out} = ab + (a \oplus b) r_{in}$$



5 porte logiche.
Cammini critici: s → 2; r_{out} → 3

**CIRCUITO FINALE
FULL HADDER**

s - rilevatore di (dis)parità **VALE 1 QUANDO O 1 O 3 BIT IN INGRESSO SONO UGUALI AD 1**
r_{out} – riporto se generato (**a = b = 1**) o se propagato (**a ⊕ b = 1**) **r_{out} = r_{in}**

A.A. 2023-2024

12/41

<http://borghese.di.unimi.it/>

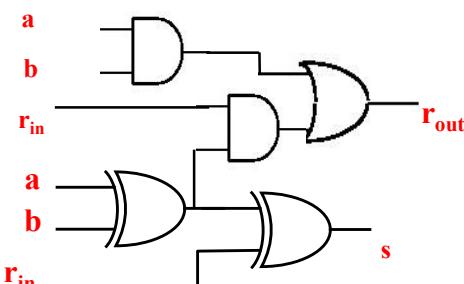
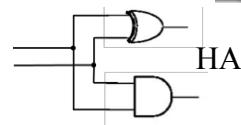


Circuito costruito con HA

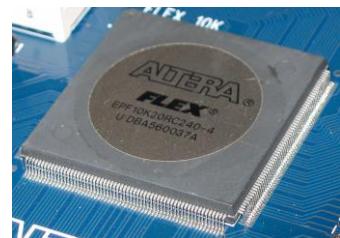


$$s = a \oplus b \oplus r_{in}$$

$$r_{out} = ab + (a \oplus b) r_{in}$$



Esempio di fitting

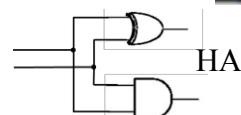


Circuito costruito con HA

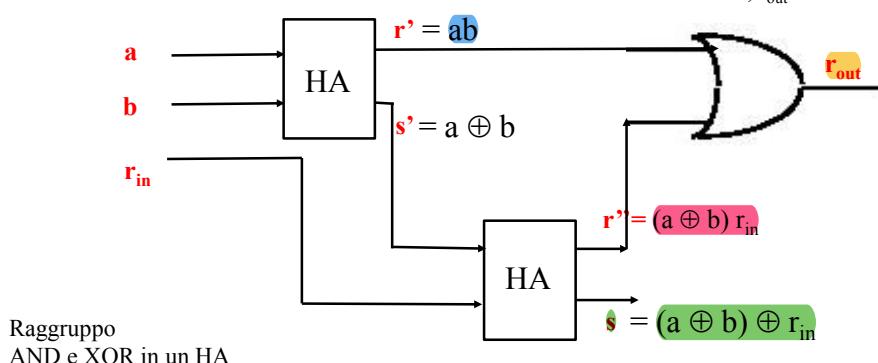


$$s = a \oplus b \oplus r_{in}$$

$$r_{out} = ab + (a \oplus b) r_{in}$$



5 porte logiche.
Cammini critici: $s \rightarrow 2$; $r_{out} \rightarrow 3$

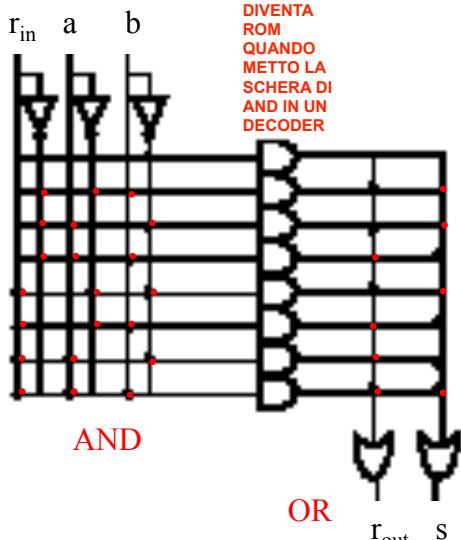




FULL ADDER CON PLA Implementazione mediante PLA



a	b	r _{in}	somma	r _{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



SOP: costruisco i mintermini e li sommo



Esercizi con ROM e PLA



Implementare il circuito del Full Adder mediante ROM

Scrivere il circuito che esegue la somma di: $3 + 4$ in base 2.

Riportare tutte le uscite delle porte logiche.

Scrivere il circuito che esegue la seguente sottrazione: $5 - 2$ in base 2. Riportare tutte le uscite delle porte logiche.



Sommario



Addizionatori

Addizionatori ad anticipazione di riporto



OR su più bit



1	0	0	1
---	---	---	---

OR

1	1	0	0
---	---	---	---

=

1	1	0	1
---	---	---	---

Ogni bit viene elaborato separatamente



AND su più bit



1	0	0	1
---	---	---	---

AND

1	1	0	0
---	---	---	---

=

1	0	0	0
---	---	---	---

Ogni bit viene elaborato separatamente



Operazione di somma



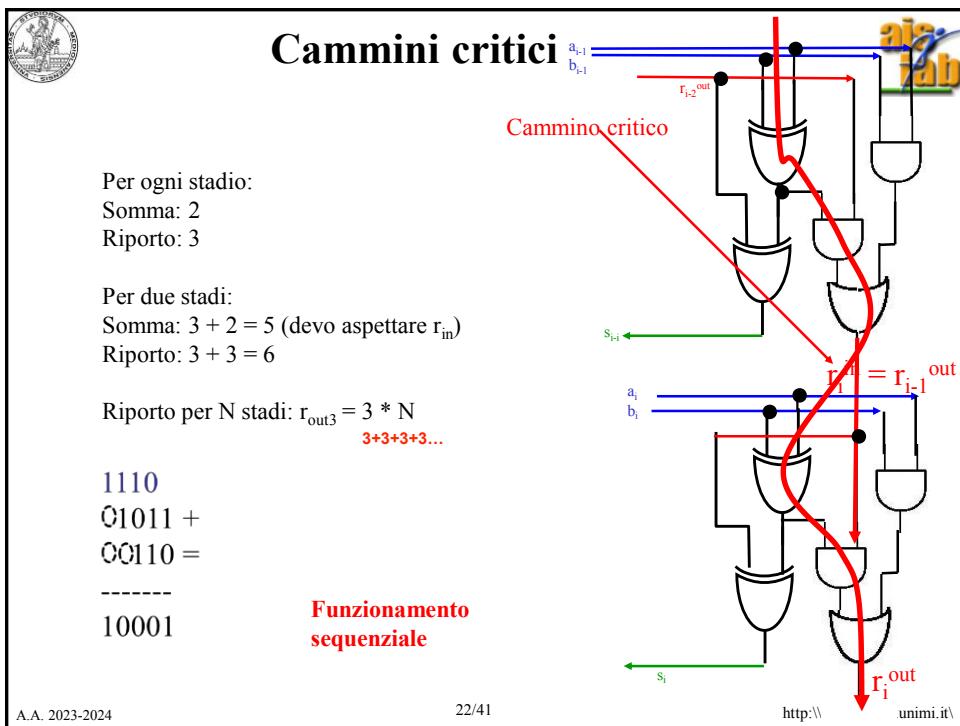
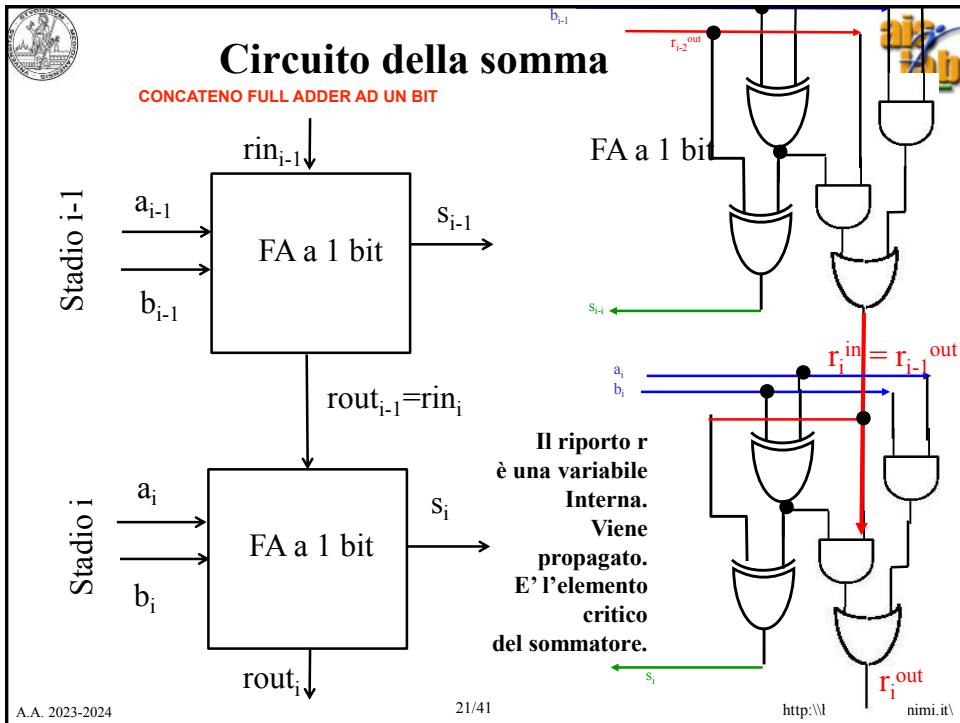
$$\begin{array}{r} 1110 \\ 01011 + \\ 00110 = \\ \hline 10001 \end{array}$$

← Riporto
← Addendo 1
← Addendo 2

Per ogni bit ho 3 Attori: addendo 1, addendo 2, riporto.

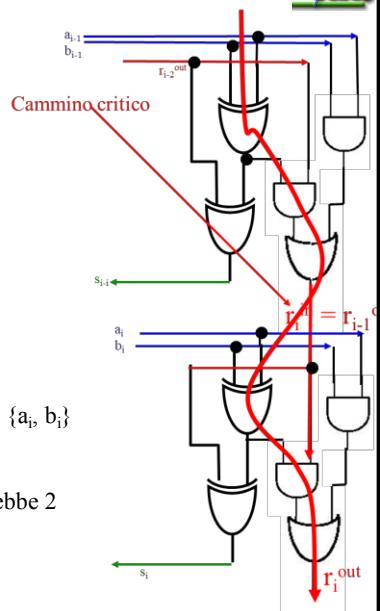
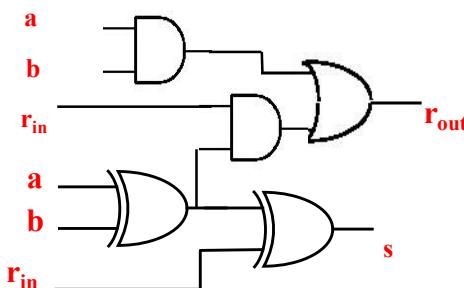
Gli addendi sono presenti all'inizio
Il riporto viene generato via via che la somma viene svolta

Viene eseguita sequenzialmente da dx a sx.





Osservazioni sul cammino critico



I termini g_i si possono calcolare a partire dagli ingressi $\{a_i, b_i\}$ senza aspettare il riporto in ingresso.

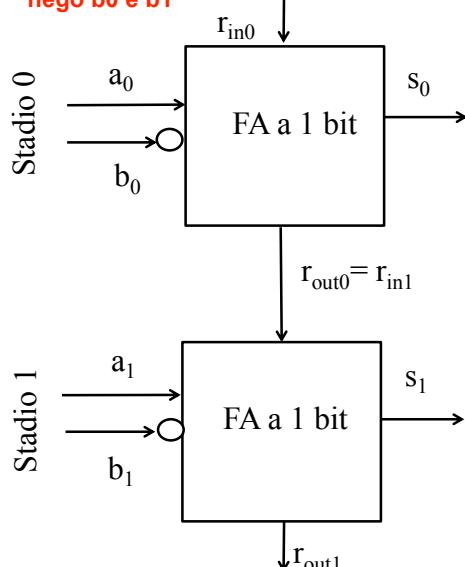
Con questa considerazione, ciascuno stadio aggiungerebbe 2 porte di ritardo. Trascuriamo nel seguito.



Circuito della sottrazione



uguale a prima ma nego b0 e b1



Sommo i seguenti 2 numeri $11 + (-13)$:

$$A = 01011_2 = 11_{10}$$

$$B = 10011_2 = -13_{10}$$

E' equivalente ad effettuare la differenza:
 $S = A - B = 11 - 13$

Calcolo di $-B$:

- $B = +13_{10} = 01101_2 \Rightarrow$
- Complemento a 1 $\Rightarrow 10010 \quad B = \bar{B}$
- Sommo 1 per ottenere il complemento a 2:
 $10010 + 1 = 10011_2 = -13_{10}$

Il complemento a 2 di B , $-B$, si ottiene come:

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

La sottrazione diventa una somma:

$$S = A + (\bar{B} + 1)$$



I problemi del full-adder



Il full adder con propagazione del riporto è lento.

- Il riporto si propaga sequenzialmente
 - caratteristica dell'algoritmo di calcolo
- la commutazione dei circuiti non è istantanea (tempo di commutazione)
 - caratteristica fisica dei dispositivi
- Soluzioni
 - modificare l'algortimo**
 - modificare i dispositivi**



Prima possibilità: forma tabellare



Riscrivo le equazioni del riporto in modo non sequenziale. Come?

$$\{r_{out3}, s_{out3}, s_{out2}, s_{out1}, s_{out0}\} = f(r_{in0}, a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3, \dots)$$

Scrivo la tabella della verità dove in uscita ho il riporto in uscita e I bit di somma e In ingresso 2 * N valori (gli N bit dei 2 addendi).

La tabella della verità ha $2^{(2N+1)}$ righe (per N=32, ...)

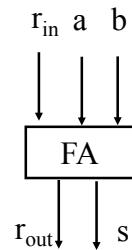


Carry look-ahead (anticipazione di riporto) CLA



Approccio strutturato per diminuire la latenza della somma.

$$r_{out} = ab + (a \oplus b) r_{in}$$



Analisi del singolo stadio.

Quando si genera un riporto in uscita?

Quando ho almeno due 1, in ingresso; cioè almeno
due «1» tra r_{in} , a e b .

11000 riporto

$$\begin{array}{r} 01101 \\ + 00100 \\ \hline \end{array}$$

$$\begin{array}{r} 10001 \\ \hline \end{array}$$



Propagazione e generazione



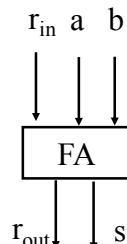
Ho riporto quando ho almeno due 1, in ingresso; cioè tra r_{in} , a e b .

a	b	r_{in}	somma	riporto
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Osservazioni:

- 1) Viene generato un riporto dallo stadio i , qualsiasi sia il riporto in ingresso se $a_i = b_i = 1 \Rightarrow g_i = a_i b_i$.
- 2) Viene generato un riporto allo stadio i , se il riporto in ingresso è = 1 ed una delle due variabili in ingresso è = 1 $\Rightarrow p_i = (a_i \oplus b_i) \Rightarrow$ viene generato riporto se $p_i r_i^{in} = 1$ (p_i propaga il segnale di riporto r_i^{in}).

Quando sia la condizione 1) che la condizione 2) è verificata?
Cosa succede se entrambe le condizioni sono verificate?





Esempio



Sono interessato ad r_4^{out} . Supponiamo anche il riporto in ingresso al primo stadio: $r_0^{\text{in}} = 0$.

$$\begin{array}{r} r_{\text{in}} \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ a \quad 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 + \\ b \quad 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 = \\ \hline \end{array}$$

1 0 1 1 1 1 1 1
ne generazione ne propagazione

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 + \\ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 = \\ \hline \end{array}$$

1 1 1 0 0 1 1 1
ho propagazione

$$\begin{array}{r} 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 + \\ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 = \\ \hline \end{array}$$

1 1 0 1 0 1 0 1
ho generazione perché $a_4 \ b_4 = 1$
e propagazione

$$r_5^{\text{in}} = r_4^{\text{out}} = 0$$

$$r_5^{\text{in}} = r_4^{\text{out}} = 1$$

$$r_5^{\text{in}} = r_4^{\text{out}} = 1$$

Per propagazione

Per generazione

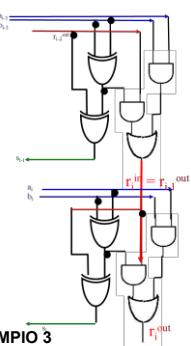
$$p_4 = (a_4 \oplus b_4)r_4^{\text{in}}.$$

$$g_4 = a_4 b_4$$



Sviluppo della funzione logica riporto

ESPRESSIONE RICORSIVA DEL RIPORTO



$$r_i^{\text{out}} = \text{GENERATO} \quad \text{PROPAGATO}$$

$$r_i^{\text{out}} = g_i + p_i r_i^{\text{in}}$$

$$r_{i-1}^{\text{out}} = g_0 + p_0 r_{i-1}^{\text{in}}$$

$$r_{i-1}^{\text{out}} = g_1 + p_1 r_0^{\text{in}} = g_1 + p_1 g_0 + p_1 p_0 r_{i-1}^{\text{in}}$$

ESEMPIO 1

$$\begin{array}{r} r_{i-1}^{\text{out}} \quad 1 \ 1 \ 1 \\ r_{i-1}^{\text{in}} \quad 1 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 + \\ 0 \ 0 \ 1 \ 0 = \\ \hline \end{array}$$

propagato

$$\begin{array}{r} g_0 = 0 \\ p_0 = p_1 = 1 \end{array}$$

ESEMPIO 2

$$\begin{array}{r} r_{i-1}^{\text{out}} \quad 1 \ 1 \ 0 \\ r_{i-1}^{\text{in}} \quad 1 \ 0 \ 0 \ 1 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 + \\ 0 \ 0 \ 1 \ 1 = \\ \hline \end{array}$$

riporto generato nel
primo stadio e poi
propagato

$$\begin{array}{r} g_0 = 1 \\ p_1 = 1 \end{array}$$

ESEMPIO 3

$$\begin{array}{r} r_{i-1}^{\text{out}} \quad 1 \ 0 \\ r_{i-1}^{\text{in}} \quad 1 \ 0 \ 1 \ 0 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 + \\ 0 \ 0 \ 1 \ 1 = \\ \hline \end{array}$$

generato

$$\begin{array}{r} g_1 = 1 \end{array}$$



Sviluppo della funzione logica riporto



$$r_i^{out} = ab + (a \oplus b) r_i^{in}$$

$$r_i^{out} = g_i + p_i r_i^{in}$$



$$r_0 = g_0 + p_0 r_0$$

$$r_1 = g_1 + p_1 r_0 = g_1 + p_1 g_0 + p_1 p_0 r_0$$

$$r_2 = g_2 + p_2 r_1 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 r_0) = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0.$$

$$r_3 = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0.$$

SONO I 5 CASI DOVE POSSO AVERE RIPORTO IN USCITA SE UNO DI QUESTI 5 TERMINI è = 1

Propago il riporto



Determinazione del cammino critico.

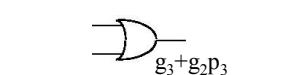
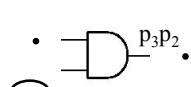
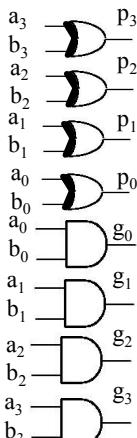


$$r_3 = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0$$

P e G sono creati grazie ad A e B

CALCOLO CON PORTE IN PARALLELO

r_0



$g_3 + g_2 p_3$

ho dimezzato il cammino critico

Cammino critico = 6, senza anticipazione sarebbe $3 * 4 = 12$

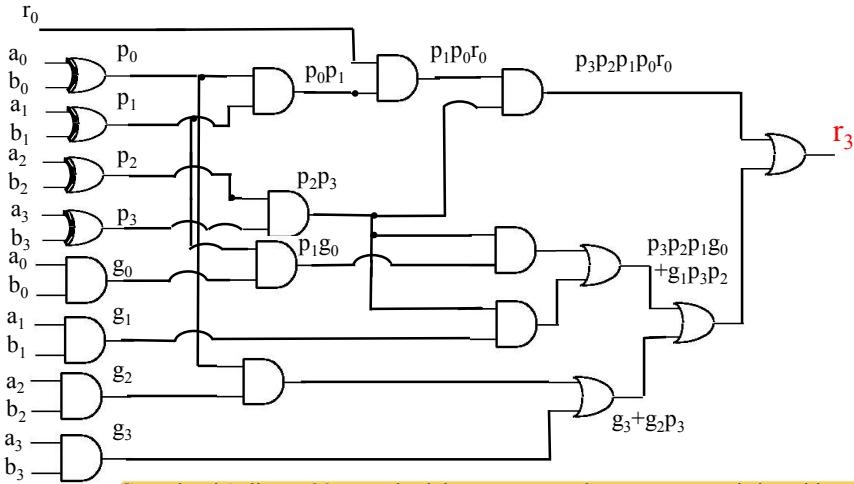
SOMMATORE AD ANTICIPAZIONE DI RIPORTO



Determinazione la complessità.



$$r_3 = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) = \\ g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0.$$



Complessità di $r_3 = 20$ porte logiche + porte per la somma e per i riporti interni



Come determino i bit di somma?



propagazione

$$s_k = (a_k \oplus b_k) \oplus r_{k\text{in}} = p_k \oplus r_{k\text{in}}$$

Occhorre calcolare $r_{1,\text{in}}$, $r_{2,\text{in}}$, $r_{3,\text{in}}$ ($r_{0,\text{in}} = r_0$ dato)

Occhorre riutilizzare più espressioni possibile.



Complessità aggiuntiva per gli altri bit di riporto



$$r_2 = g_2 + p_2 r_1 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 r_0) = \\ g_2 + p_2 g_1 + p_2 \textcolor{red}{p_1 g_0} + p_2 \textcolor{red}{p_1 p_0 r_0}$$

In rosso le porte già presenti nel circuito di r_{out3}

Complessità aggiuntiva pari a **6** porte logiche.

$$r_1 = g_1 + p_1 r_0 = g_1 + \textcolor{red}{p_1 g_0} + \textcolor{red}{p_1 p_0 r_0}$$

In rosso le porte già presenti nel circuito di r_{out3}

Complessità aggiuntiva pari a **2** porte logiche.

Complessità aggiuntiva totale per i riporti: **8** porte logiche.



Complessità aggiuntiva per i bit di somma



PROPAGAZIONE

$$s_k = (a_k \oplus b_k) \oplus r_{kin} = p_k \oplus r_{kin}$$

Ogni bit di somma aggiunge una porta logica XOR =>
La complessità aumenta di $N * 1 = \textcolor{blue}{4}$ porte logiche.

Un CLA su 4 bit ha quindi una complessità di $20 + \textcolor{blue}{6+2+4} = 32$ porte logiche.

Un sommatore a propagazione di riporto ha una complessità di $4*5 = 20$ porte logiche.

AUMENTO COMPLESSITÀ MA DIMEZZO CAMMINO CRITICO



Quanto si guadagna con l'anticipazione del riporto per N stadi?



Cammino critico per le variabili interne:

$$r_0^{\text{out}} \Rightarrow 3$$

$$r_1^{\text{out}} \Rightarrow 4$$

$$r_2^{\text{out}} \Rightarrow 5$$

Cammino critico per le variabili esterne:

$$r_3^{\text{out}} \Rightarrow 6$$

$s_3 \Rightarrow 6$ NB la prima porta XOR è in comune con r_2^{out}

$s_2 \Rightarrow 5$ NB la prima porta XOR è in comune con r_1^{out}

$s_1 \Rightarrow 4$ NB la prima porta XOR è in comune con r_0^{out}

$$s_0 \Rightarrow 2$$

Cammino critico scala come $CC_{(1 \text{ stadio})} * \log(N)$



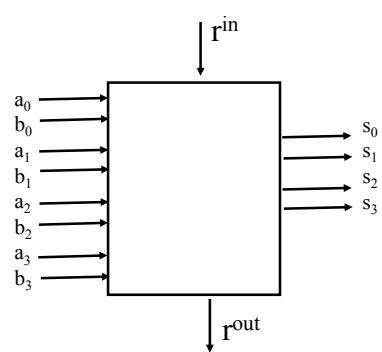
Addizionatori modulari



La complessità del circuito è tollerata per piccoli n.

Circuiti sommatori indipendenti si hanno per 4 bit.

Moduli elementari.



Rout3 diventerà Rin4

Come si ottiene la somma?

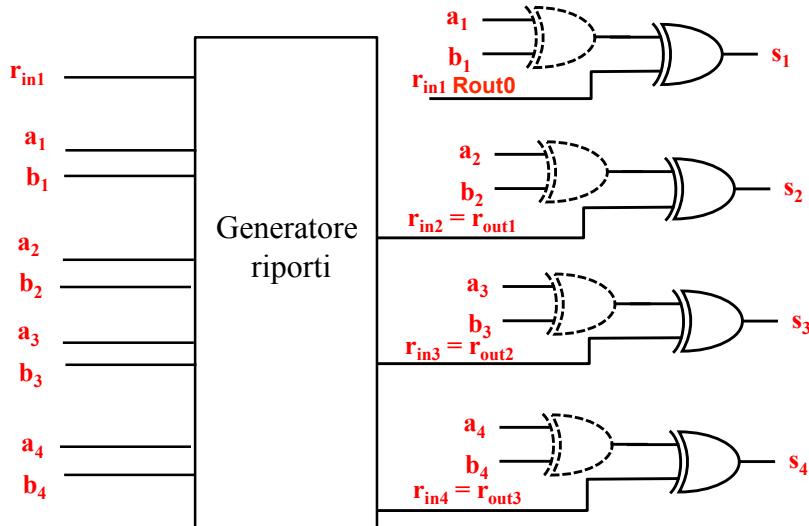
Collegando in cascata i moduli (sommatori elementari).

Cammino critico = 6 (CC di un modulo a 4 bit) * N/4. Per 32 bit, 48 (ciascun modulo dimezza il CC).

Per confronto, senza parallelizzazione, sommatore a propagazione di riporto, per 32 bit, N * 3 = 96.



Architettura interna



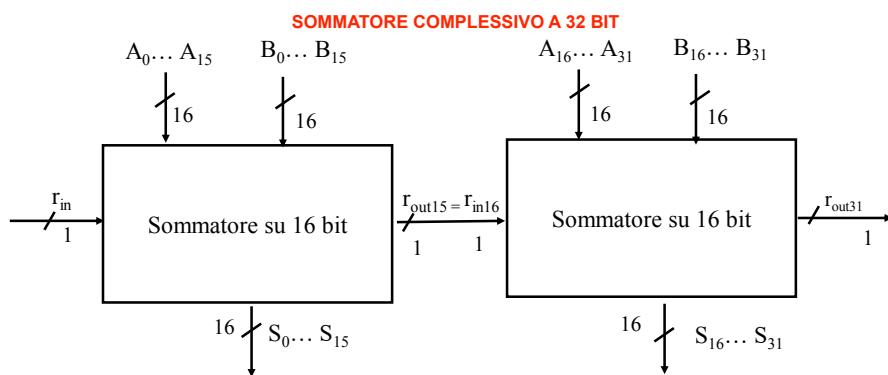
Addizionatori modulari::esempio



Occorre sommare 2 variabili, A e B, su $N = 32$ bit
Ho a disposizione due sommatori su 16 bit.

$$CC = CC_{(1 \text{ stadio})} * \log(N) = 3 * 4 = 12$$

Come si ottiene la somma?
Fondamento delle estensioni architettonali SSE





Sommario



Addizionatori

Addizionatori ad anticipazione di riporto



Moltiplicatori HW e ALU

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti: Appendice B5 prima parte.
Per approfondimenti, Capitolo 7 del Fummi, Sami, Silvano.



Sommario

Moltiplicatori

ALU



Moltiplicazione mediante shift



Lo **shift di un numero a dx**, di k cifre, corrisponde ad una **divisione** per la base elevata alla k-esima potenza.

Lo **shift di un numero a sx**, di k cifre, corrisponde ad una **moltiplicazione** per la base elevata alla k-esima potenza.

Esempio nel caso decimale:

$$213_{10} \times 10 = 2130_{10} \quad \text{shift a sinistra} \quad \text{moltiplicazione}$$

$$213_{10} = (2 \times 10^2 + 1 \times 10^1 + 3 \times 10^0) \times \mathbf{10^1} =$$

$$(2 \times 10^2 + 1 \times 10^1 + 3 \times 10^0) \times \mathbf{10^1} =$$

$$(2 \times 10^2 \times \mathbf{10^1} + 1 \times 10^1 \times \mathbf{10^1} + 3 \times 10^0 \times \mathbf{10^1}) =$$

$$(2 \times 10^3 + 1 \times 10^2 + 3 \times 10^1) = 2130 \text{ cvd.}$$

$$213_{10} / 10 = 21,3_{10} \quad \text{shift a destra} \quad \text{divisione}$$

$$213_{10} = (2 \times 10^2 + 1 \times 10^1 + 3 \times 10^0) / \mathbf{10^1} =$$

$$(2 \times 10^2 + 1 \times 10^1 + 3 \times 10^0) / \mathbf{10^{-1}} =$$

$$(2 \times 10^2 \times \mathbf{10^{-1}} + 1 \times 10^1 \times \mathbf{10^{-1}} + 3 \times 10^0 \times \mathbf{10^{-1}}) =$$

$$(2 \times 10^1 + 1 \times 10^0 + 3 \times 10^{-1}) = 21,3 \text{ cvd.}$$



Moltiplicazione mediante shift



Lo shift di un numero a dx, di k cifre, corrisponde ad una divisione per la base elevata alla k-esima potenza.

Lo shift di un numero a sx, di k cifre, corrisponde ad una moltiplicazione per la base elevata alla k-esima potenza.

Esempio nel caso binario:

$$23 * \mathbf{4} = 92 \Rightarrow 10111 * 100 = 1011100 \quad \text{shift a sinistra}$$

Esprimendo l'operazione in decimale:

$$(1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \times \mathbf{2^2} =$$

$$(1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2) = 64 + 16 + 8 + 4 = 92 \text{ cvd.}$$

$$23 / \mathbf{4} = 5,75 \Rightarrow 10111 / 100 = 101,11 \quad \text{shift a destra}$$

Esprimendo l'operazione in decimale:

$$(1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) \times \mathbf{2^{-2}} =$$

$$(1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}) = 5,75 \text{ cvd.}$$

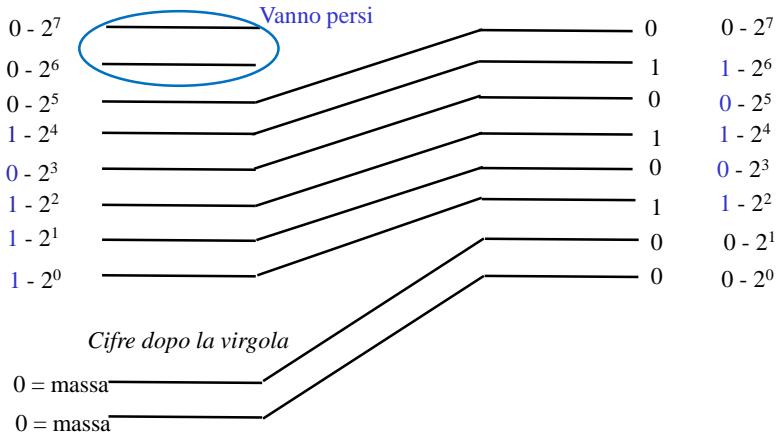


Moltiplicazione mediante shift

Codifica su 8 + 2 cifre: **8 cifre prima della virgola e 2 dopo**

$$23 * 4 = 92 \Rightarrow 10111 * 100 = 1011100$$

$$23_{10} = 00010111,00_2$$

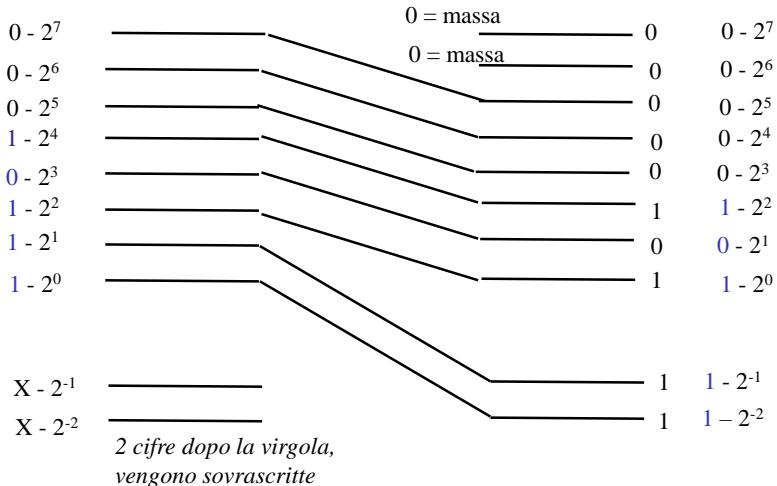


Divisione mediante shift

Codifica su 8 + 2 cifre:

$$23 / 4 = 5,75 \Rightarrow 10111 / 100 = 101,1100$$

$$23_{10} = 00010111,00_2$$





Moltiplicazione decimale



Moltiplicando → $2\ 7\ 8\ x$
 Moltiplicatore → $4\ \textcolor{brown}{2}\ 3 =$

Prodotti parziali → $\begin{array}{r} 3 \times 278 \\ 20 \times 278 \\ 400 \times 278 \end{array}$ $\begin{array}{r} 8\ 3\ 4\ + \\ 5\ 5\ 6\ - \\ 1\ 1\ 1\ 2\ -\ - \end{array}$

prodotto → $1\ 1\ 7\ 5\ 9\ 4$

$$278 \times 423 = 278 \times (4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0) =$$

$$278 \times (4 \times 10^2) + 278 \times (2 \times 10^1) + 278 \times (3 \times 10^0)$$

Somma dei prodotti parziali

ESEMPIO 1

27×7



Moltiplicazione binaria - I



Moltiplicando → $1\ 1\ 0\ 1\ 1\ x$
 Moltiplicatore → $1\ 1\ \textcolor{blue}{1} =$

$1\ 1\ 0\ 1\ 1\ x \quad 27_{10}$ $1^{\circ} \text{ prodotto parziale} \quad 1\ 1\ 0\ 1\ 1\ +$

$1\ 1\ 1\ = \quad 7_{10}$

$1\ 1\ 1\ 1\ 1\ 1$
 $1\ 1\ 0\ 1\ 1\ +$
 $1\ 1\ 0\ 1\ 1\ -$
 $1\ 1\ 0\ 1\ 1\ -\ -$

$1\ 0\ 1\ 1\ 1\ 1\ 0\ 1 \quad 189_{10}$

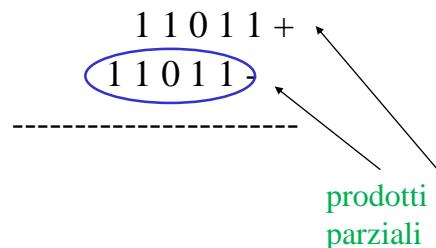


Moltiplicazione binaria - II



Moltiplicando \longrightarrow 1 1 0 1 1 x
 Moltiplicatore \longrightarrow 1 1 1 =

$$\begin{array}{r}
 1 1 0 1 1 x \quad 27_{10} \\
 1 1 1 = \quad 7_{10} \\
 \hline
 1 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 1 1 0 1 1 - - \\
 \hline
 1 0 1 1 1 1 0 1 \quad 189_{10}
 \end{array}$$



Il secondo prodotto parziale è incolonnato alle potenze di 2^1 (la cifra del moltiplicatore ha peso 2^1)

Ho generato 2 addendi (2 prodotti parziali)
 Provvedo subito alla loro somma

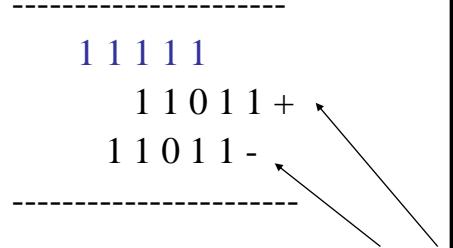


Moltiplicazione binaria - III



Moltiplicando \longrightarrow 1 1 0 1 1 x
 Moltiplicatore \longrightarrow 1 1 1 =

$$\begin{array}{r}
 1 1 0 1 1 x \quad 27_{10} \\
 1 1 1 = \quad 7_{10} \\
 \hline
 1 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 1 1 0 1 1 - - \\
 \hline
 1 0 1 1 1 1 0 1 \quad 189_{10}
 \end{array}$$



1^a Somma parziale \longrightarrow 1 0 1 0 0 0 1 + 2 prodotti parziali

$$\begin{aligned}
 P &= P_0 + P_1 + P_2 = (P_0 + P_1) + P_2 = \\
 &= S_0 + P_2
 \end{aligned}$$



Moltiplicazione binaria - IV



Moltiplicando \longrightarrow 1 1 0 1 1 x
 Moltiplicatore \longrightarrow 1 1 1 =

$$\begin{array}{r}
 1 1 0 1 1 x \quad 27_{10} \\
 1 1 1 = \quad 7_{10} \\
 \hline
 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 1 1 0 1 1 - - \\
 \hline
 1 0 1 1 1 1 0 1 \quad 189_{10}
 \end{array}$$

$$\begin{array}{r}
 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 \hline
 1 0 1 0 0 0 1 + \\
 1 1 0 1 1 - \\
 \hline
 \end{array}$$

1^a Somma parziale → 1 0 1 0 0 0 1 + 1 1 0 1 1 -

prodotti parziali

Il terzo prodotto parziale è incolonnato alle potenze di 2^2 (la cifra del moltiplicatore ha peso 2^2)



Moltiplicazione binaria - V



Moltiplicando \longrightarrow 1 1 0 1 1 x
 Moltiplicatore \longrightarrow 1 1 1 =

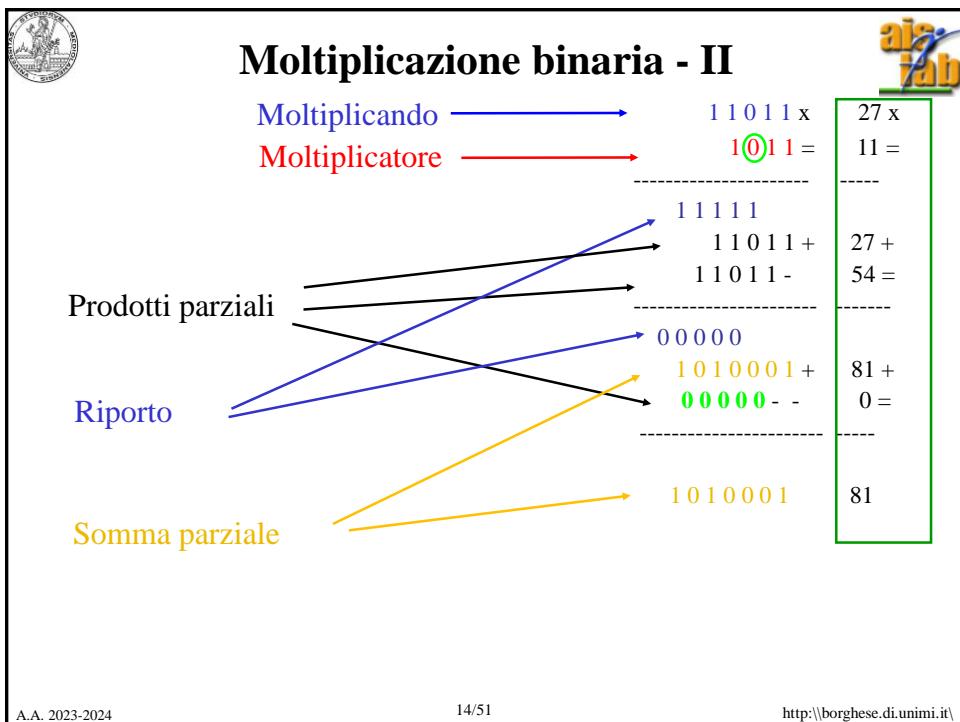
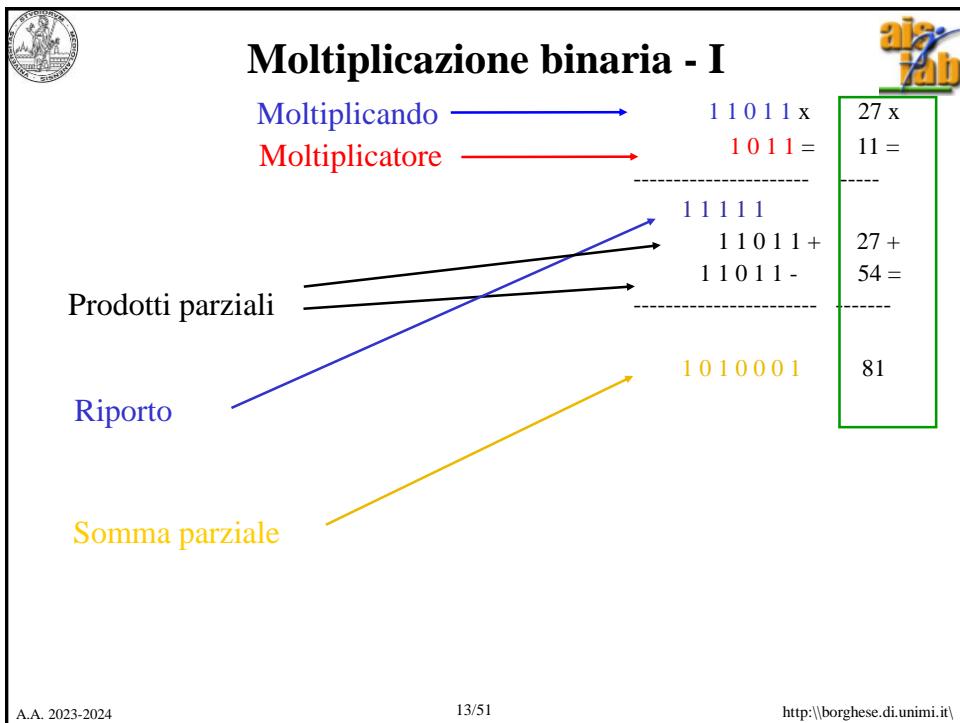
$$\begin{array}{r}
 1 1 0 1 1 x \quad 27_{10} \\
 1 1 1 = \quad 7_{10} \\
 \hline
 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 1 1 0 1 1 - - \\
 \hline
 1 0 1 1 1 1 0 1 \quad 189_{10}
 \end{array}$$

$$\begin{array}{r}
 1 1 1 1 1 \\
 1 1 0 1 1 + \\
 1 1 0 1 1 - \\
 \hline
 1 0 0 0 0 \\
 1 0 1 0 0 0 1 + \\
 1 1 0 1 1 - - \\
 \hline
 \end{array}$$

1^a Somma parziale → 1 0 1 0 0 0 1 + 1 1 0 1 1 - -

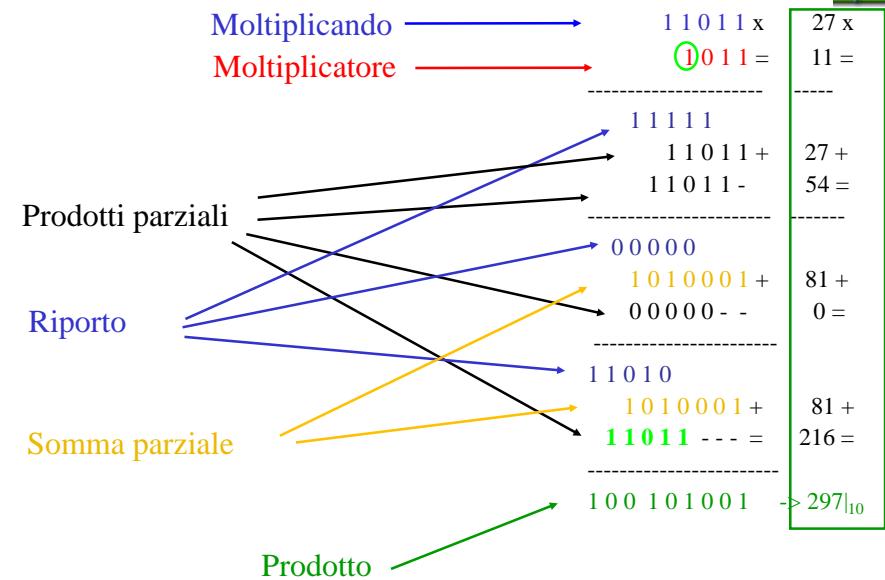
prodotti parziali

Somma finale = prodotto \longrightarrow 1 0 1 1 1 1 0 1

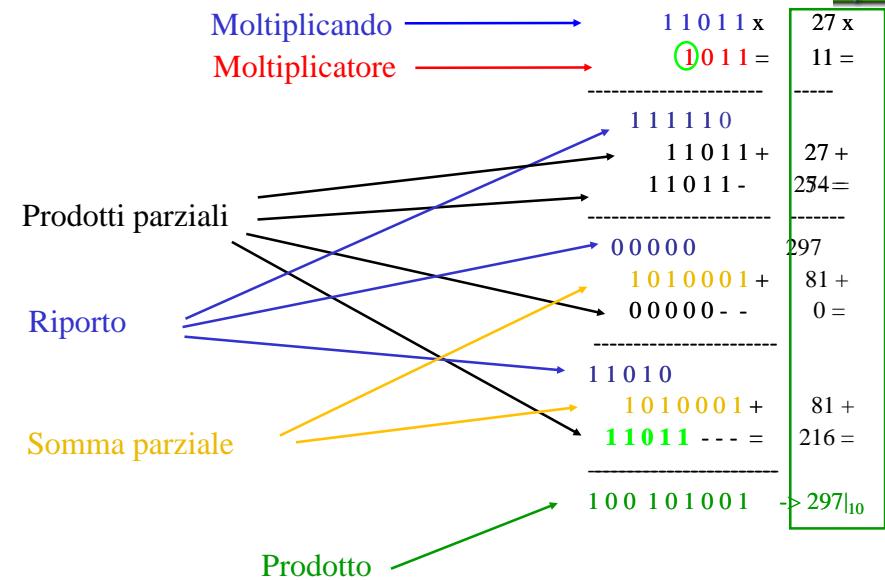
ESEMPIO 2 **27×11** 



Moltiplicazione binaria - III



Moltiplicazione: prodotti e somme



Moltiplicazione binaria (su 4 bit)

The diagram illustrates the step-by-step process of binary multiplication:

- Moltiplicando**: Represented by a blue arrow pointing to the number $1011 \times$.
- Moltiplicatore**: Represented by a red arrow pointing to the number $101 =$.
- OPERAZIONE AND**: Partial products are generated by performing an AND operation between the multiplicando and each bit of the multiplicatore.
- SOMMATORI**: Partial sums (Sommatori) are generated by summing the partial products. The first partial product is $1011 * 1 * 2^0 = 1011$. The second partial product is $1011 * 0 * 2^1 = 0000$.
- Prodotto**: The final result is obtained by summing all partial products: $1011 + 0000 = 110111$, which is 55_{10} .

Il prodotto parziale è = $\begin{cases} \text{Moltiplicando incolonnato opportunamente} \\ 0 \end{cases}$

quando ho cifra moltiplicatore = 0 -> avrò tutti 0
quando ho cifra moltiplicatore = 1 -> avrò cifre del moltiplicando

A.A. 2023-2024 17/51 <http://borgheze.di.unimi.it/>

Moltiplicazione su 1 bit

The diagram shows two configurations of an AND gate:

- Passa cifre moltiplicando**: The inputs are labeled a_k and $b_0 = 1$. The output is labeled $p_k = a_k$. This represents the case where the multiplicator is 1, allowing all bits of the multiplicando to pass through.
- Passa 0**: The inputs are labeled a_k and $b_0 = 0$. The output is labeled $p_k = 0$. This represents the case where the multiplicator is 0, resulting in a 0 output regardless of the multiplicando.

AND non solo semaforo, ma anche moltiplicatore a 1 bit!

A.A. 2023-2024 18/51 <http://borgheze.di.unimi.it/>



La moltiplicazione binaria



Possiamo vederla come:

Un primo stadio in cui si mette in AND ciascun bit del moltiplicatore con il moltiplicando (**prodotto parziale**).

Un secondo stadio in cui si effettuano le **somme dei prodotti parziali** (full adder): somma dei bit su due righe diverse.



La matrice dei prodotti parziali



A e B su 4 bit

Prodotti parziali					a cifre moltiplicando b cifre moltiplicatore	
	a_3	a_2	a_1	a_0		
	$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$	b_0	prodotto parziale associato a b0
	$a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	b_1	prodotto parziale associato a b1
	$a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$	b_2	prodotto parziale associato a b2
	$a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$	b_3	prodotto parziale associato a b3
	2^6	2^5	2^4	2^3	2^2	2^1
						2^0

In binario i **prodotti parziali sono degli AND**.

Sulla linea tanti AND quanto è la lunghezza di A
Tanti prodotti parziali quanto è la lunghezza di B



La matrice dei prodotti parziali



Prodotti parziali	a_3	a_2	a_1	a_0	
	$a_3 b_0$	$a_2 b_0$	$a_1 b_0$	$a_0 b_0$	b_0
	$a_3 b_1$	$a_2 b_1$	$a_1 b_1$	$a_0 b_1$	b_1
	$a_3 b_2$	$a_2 b_2$	$a_1 b_2$	$a_0 b_2$	b_2
	$a_3 b_3$	$a_2 b_3$	$a_1 b_3$	$a_0 b_3$	b_3

Il bit i-esimo del moltiplicatore, b_i , fa passare 0 o il moltiplicando (prodotto parziale).
Il prodotto parziale viene incolonnato opportunamente.

$b_0 (a_3 a_2 a_1 a_0)$ genera P_0 **P = prodotti parziali**

$b_1 (a_3 a_2 a_1 a_0)$ genera P_1

$b_2 (a_3 a_2 a_1 a_0)$ genera P_2

$b_3 (a_3 a_2 a_1 a_0)$ genera P_3

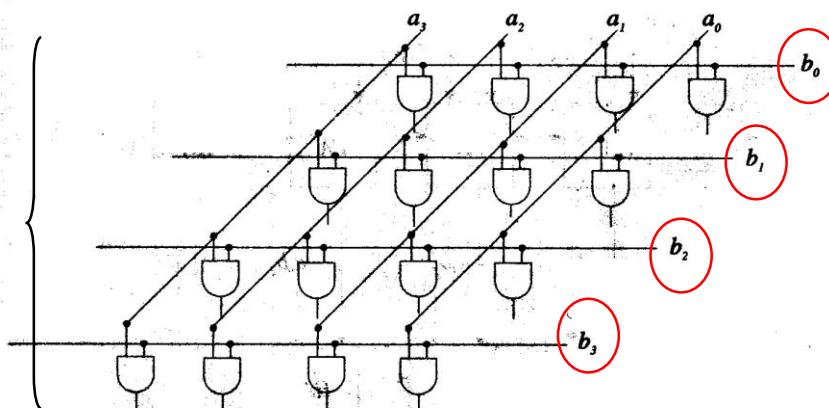
..... per fare moltiplicazione devo fare somma dei prodotti parziali



Il circuito che effettua i prodotti



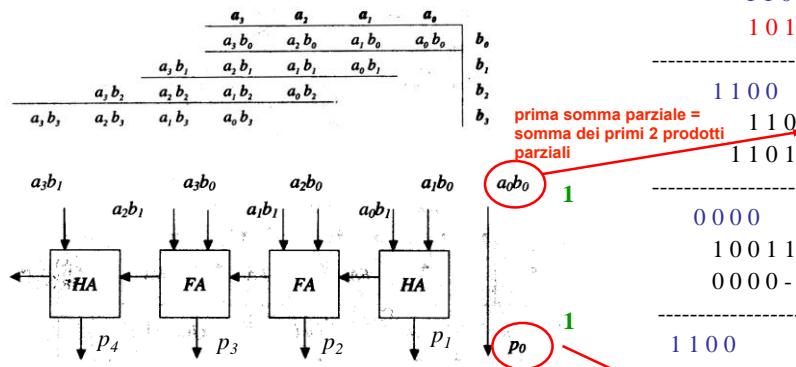
Prodotti
parziali



b_k agisce come interruttore, facendo passare 0 o A



Somma delle prime 2 righe dei prodotti parziali - I



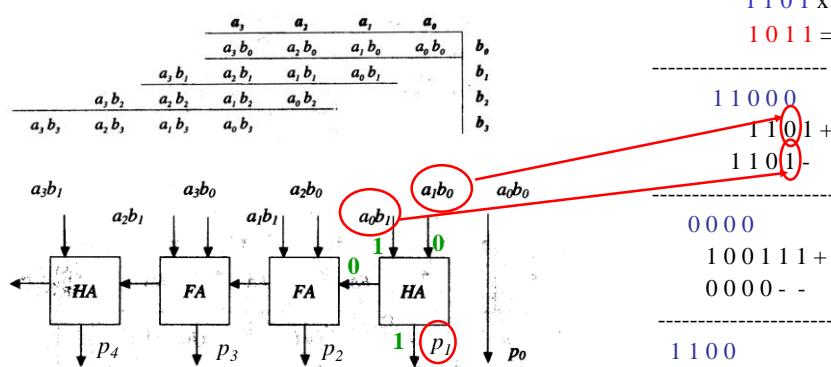
Somma dei primi 2 prodotti parziali

prima somma parziale =
somma dei primi 2 prodotti
parziali

$$\begin{array}{r}
 1101x \\
 1011= \\
 \hline
 1100 \\
 1101+ \\
 1101- \\
 \hline
 0000 \\
 100111+ \\
 0000-- \\
 \hline
 1100 \\
 100111+ \\
 1101----= \\
 \hline
 10001111 \rightarrow 143_{10}
 \end{array}$$



Somma delle prime 2 righe dei prodotti parziali - II

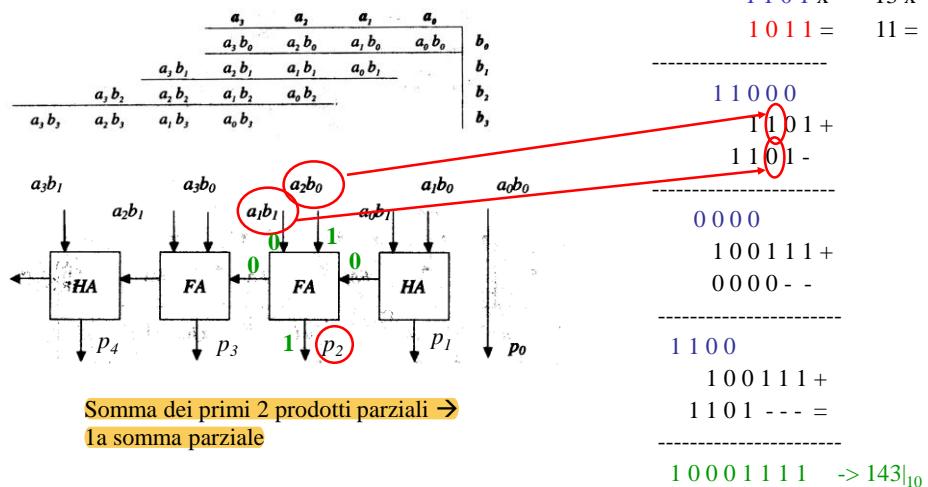


Somma dei primi 2 prodotti parziali \rightarrow
1a somma parziale

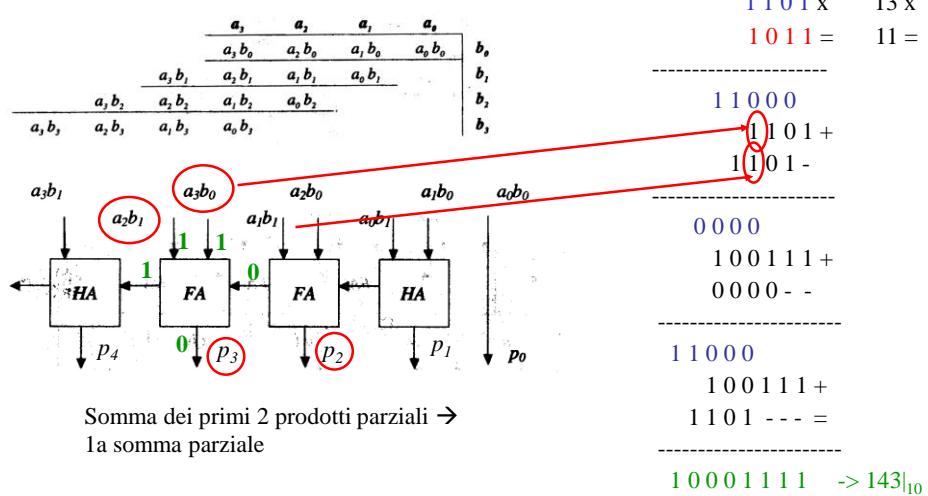
$$\begin{array}{r}
 1101x \\
 1011= \\
 \hline
 11000 \\
 1101+ \\
 1101- \\
 \hline
 0000 \\
 100111+ \\
 0000-- \\
 \hline
 1100 \\
 100111+ \\
 1101----= \\
 \hline
 10001111 \rightarrow 143_{10}
 \end{array}$$



Somma delle prime 2 righe dei prodotti parziali - III



Somma delle prime 2 righe dei prodotti parziali - IV

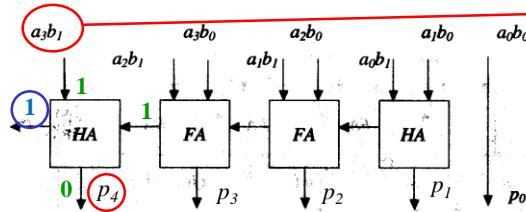




Somma delle prime 2 righe dei prodotti parziali - V



$$\begin{array}{cccc}
 a_3 & a_2 & a_1 & a_0 \\
 \hline
 a_3 b_0 & a_2 b_0 & a_1 b_0 & a_0 b_0 \\
 \hline
 a_3 b_1 & a_2 b_1 & a_1 b_1 & a_0 b_1 \\
 \hline
 a_3 b_2 & a_2 b_2 & a_1 b_2 & a_0 b_2 \\
 \hline
 a_3 b_3 & a_2 b_3 & a_1 b_3 & a_0 b_3
 \end{array}$$



Somma dei primi 2 prodotti parziali → la somma parziale

Dove va il riporto in uscita all'ultimo FA?
Io porto in ingresso all'ultimo HA

$$\begin{array}{r}
 1101x \\
 1011= \\
 \hline
 11000
 \end{array}$$

primo prod parziale
secondo prod parziale

con schema siamo arrivati qui:

$$\begin{array}{r}
 0000 \\
 100111+ \\
 0000-- \\
 \hline
 1100
 \end{array}$$

prima somma parziale

$$\begin{array}{r}
 1100 \\
 100111+ \\
 1101----= \\
 \hline
 10001111 \rightarrow 143_{10}
 \end{array}$$

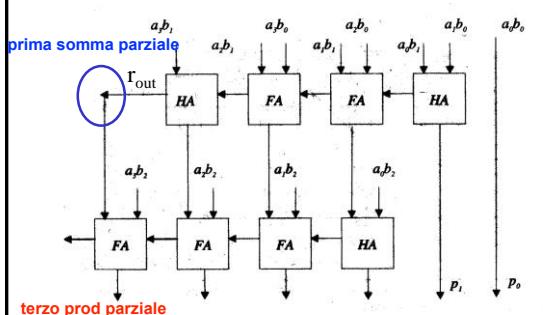


Somma della terza riga



I primi due prodotti parziali sono ottenuti dalla prima batteria di sommatori.

Ogni altro prodotto parziale è sommato da un'ulteriore batteria di sommatori.



$$\begin{array}{r}
 1101x \\
 1011= \\
 \hline
 11000
 \end{array}$$

$$\begin{array}{r}
 1101+ \\
 1101- \\
 \hline
 0000
 \end{array}$$

$$\begin{array}{r}
 100111+ \\
 0000-- \\
 \hline
 1100
 \end{array}$$

terzo prod parziale

$$\begin{array}{r}
 1100 \\
 100111+ \\
 1101----= \\
 \hline
 10001111 \rightarrow 143_{10}
 \end{array}$$

CIRCUITO FINALE MOLTIPLICAZIONE

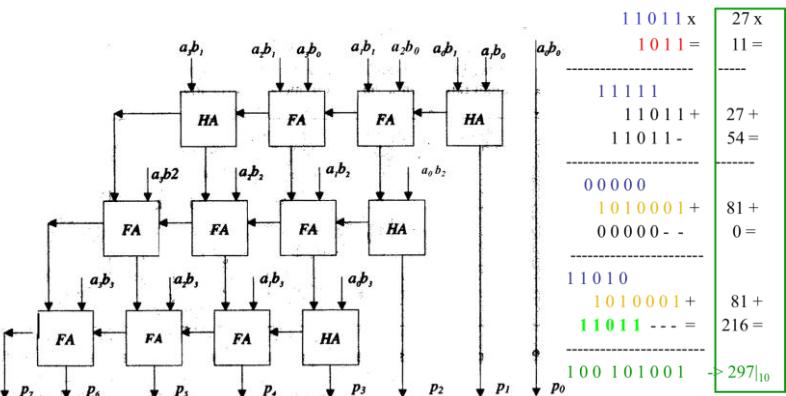
 **Circuito completo della somma dei prodotti parziali** 

N-1 batterie di sommatori

$P_0 + P_1 \rightarrow S_0$

$S_0 + P_2 \rightarrow S_1$

$S_1 + P_3 \rightarrow P$ quarto prod parziale



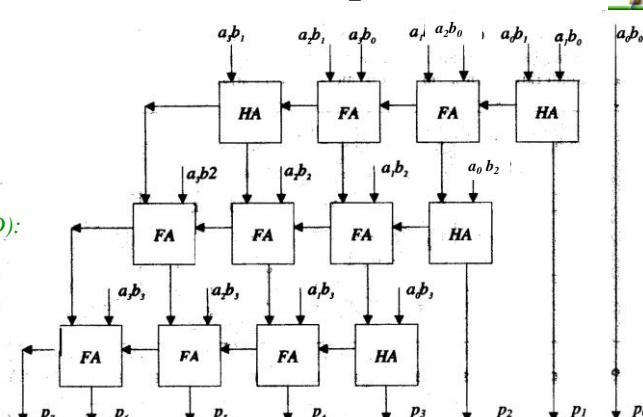
Problema: A e B su 4 bit \Rightarrow P su 8 bit (prodotto su $2N$ bit)

A.A. 2023-2024 29/51 <http://borgheze.di.unimi.it/>

 **Valutazione della complessità** 

Complessità:
 Half Adder: 2 porte
 Full Adder: 5 porte

Stadio prodotti (AND):
 A su N bit
 B su N bit
 $N * N$ porte AND



Stadio Somme:

Se $N = M = 4$ numero totale di porte a 2 ingressi = 5

N sommatori per linea
 N-1 linee
 Numero linee Numero FA per linea Numero HA per linea Complessità Prima linea Prodotti Parziali

$CO_{Tot} =$	$(N-2) * [(N-1) * 5 + 1 * 2]$	$(N-2)*5 + 2*2 +$	$N * N$
--------------	-------------------------------	-------------------	---------

A.A. 2023-2024 30/51 <http://borgheze.di.unimi.it/>



Valutazione del cammino critico



Cammini critici:

Half Adder:

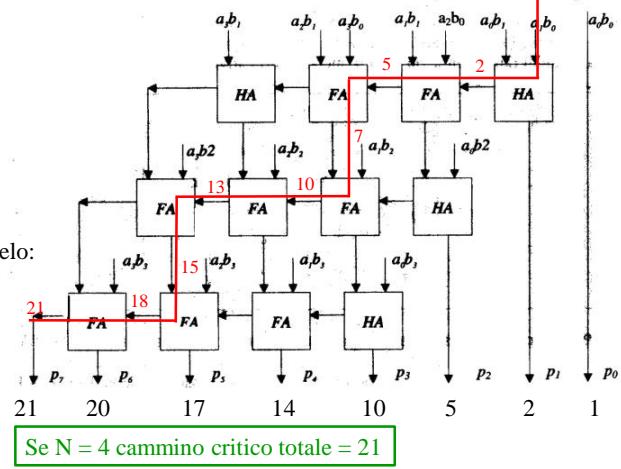
Somma – 1 porta
Riporto – 1 porta

Full Adder:

Somma - 2 porte
Riporto - 3 porte

Gli AND operano in parallelo:
ritardo 1.

HA e FA non sono
equivalenti per il
cammino critico



Osservazioni



Cammini critici:

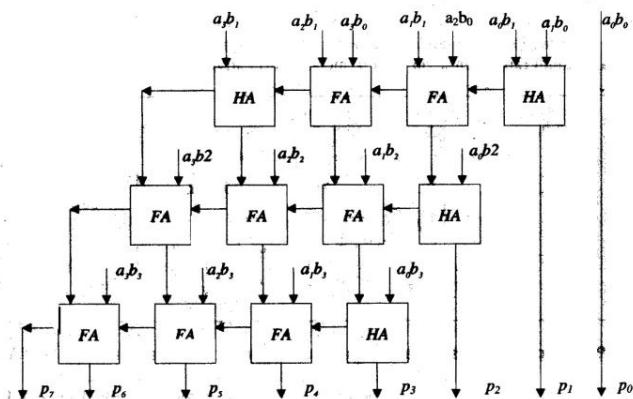
Half Adder:

Somma – 1 porta
Riporto – 1 porta

Full Adder:

Somma - 2 porte
Riporto - 3 porte

Gli AND operano in parallelo:
ritardo 1.



Architettura **modulare**, ogni schiera di sommatori lavora sul risultato della schiera superiore
e fornisce l'input alla schiera inferiore

Quanto si guadagna sostituendo ai sommatori a propagazione di riporto sommatori
ad anticipazione di riporto?



Sommario



Moltiplicatori

ALU



Funzione della ALU



E' integrata nel processore, all'inizio degli anni 90 è stata rivoluzionaria la sua introduzione con il nome di co-processore matematico.

Esegue le operazioni aritmetico-logiche.

Utilizza i blocchi di base già visti.

Opera su parole (MIPS 32 bit / 64 bit).

Le ALU non compaiono solamente nei micro-processori.



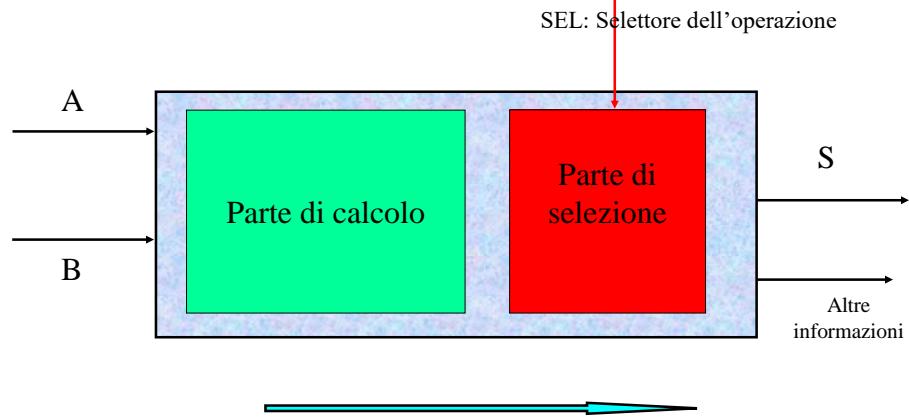
Problematiche di progetto



- Velocità (Riporto).
- Costo.
- Precisione.
- Affidabilità
- Consumo energetico.



Struttura a 2 livelli di una ALU



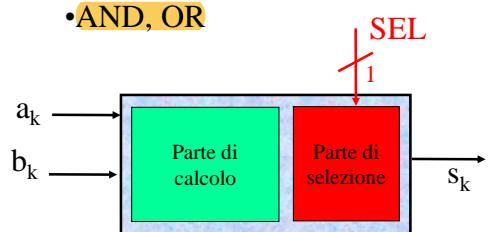
Esempio di esecuzione condizionata.



Progettazione della ALU – 1 bit



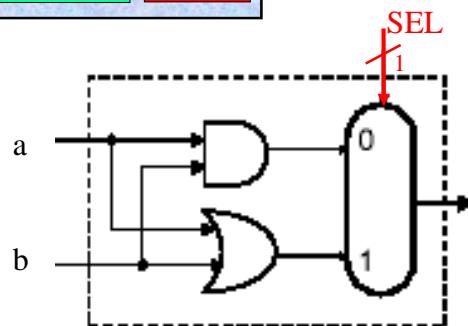
•AND, OR



a	b	a AND b	a OR b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

SEL = 0
s = AND(a,b)

SEL = 1
s = OR(a,b)



selettore per decidere se in uscita avere AND o OR



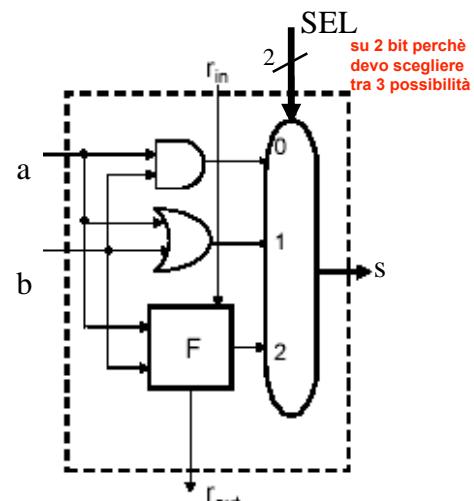
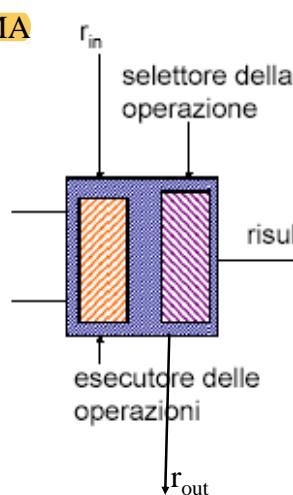
La nuova struttura della ALU – 1 bit



•AND

•OR

•SOMMA



su 2 bit perché devo scegliere tra 3 possibilità

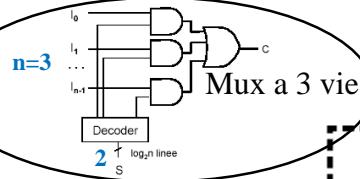
Perchè SEL non viene messo in ingresso?



Valutazione ALU a 1 bit

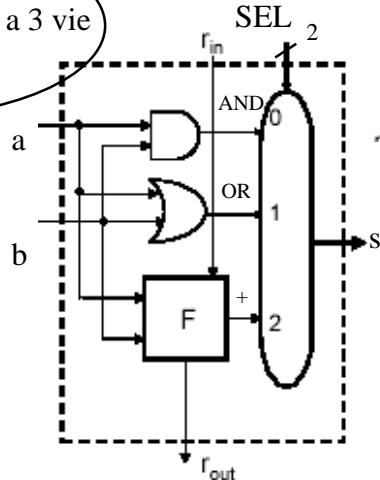


- AND
- OR
- SOMMA



Complessità 1° livello (calcolo): $5+2 = 7$
 Complessità 2° livello (mux): $3*1+(3+2) = 8$
 (Decoder + AND (semaforo) + OR
 (congiunzione))

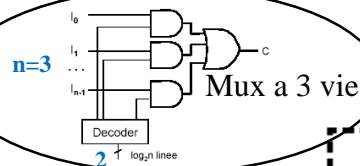
Complessità totale: $7+8 = 15$



Valutazione ALU a 1 bit



- AND
- OR
- SOMMA

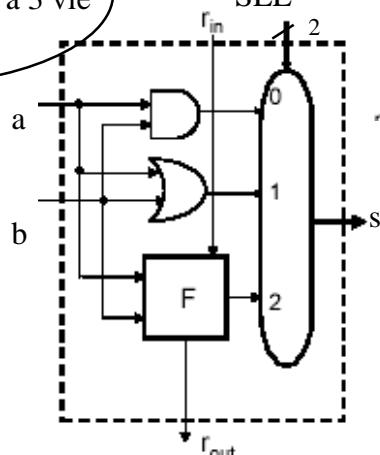


CC 1° livello: 2 per s_{out} , 3 per r_{out}
 CC 2° livello: 4 (1 Decoder + (1 AND - semaforo + 2 OR (congiunzione))

CC complessivo: 2 (calcolo) + [1 AND (semaforo)+ 2 (OR - selezione)] = 5!!

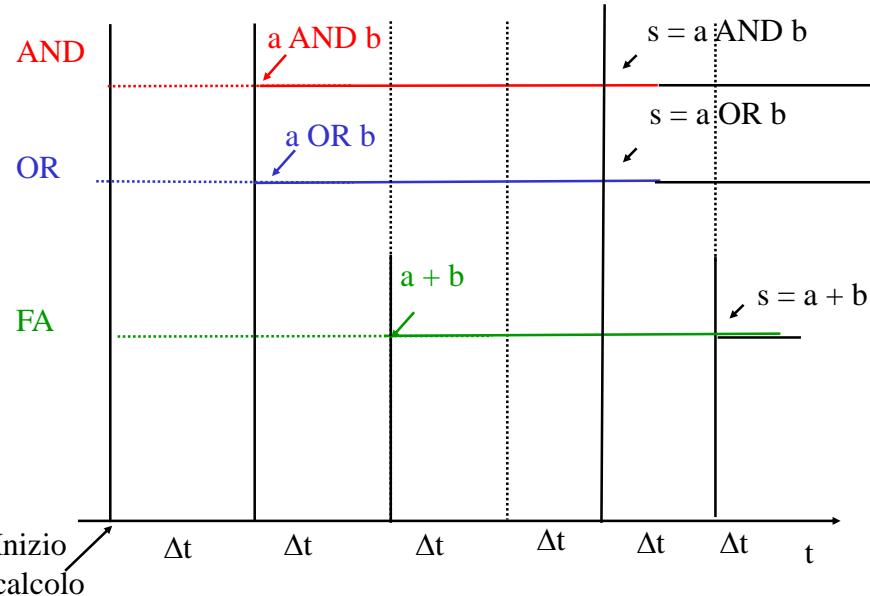
Il CC del decoder non viene contato: gli AND del decoder interni al mux sono attivati in parallelo ai circuiti di calcolo.

Il CC considerato è quello della somma per valutare il tempo necessario perché commuti s.





I cammini critici all'interno della ALU



PASSARE DA ALU 1 BIT AD ALU 32 BIT



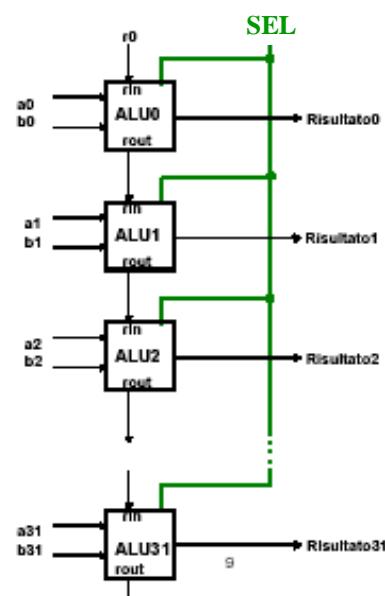
ALU a 32 bit



Come collegare le
ALU ad 1 bit?

Flusso di calcolo

Perchè non si può
parallelizzare?





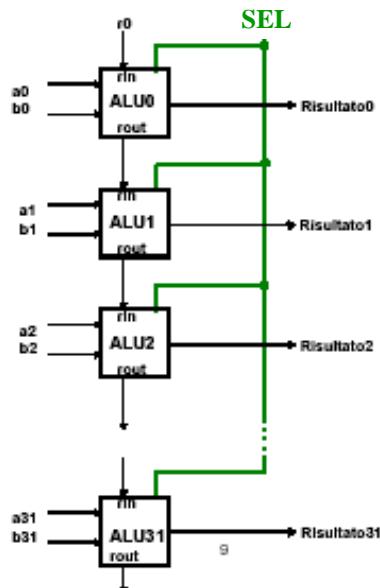
Valutazione ALU a 32 bit



Complessità: $15 \times 32 = 480$ porte logiche

Cammino critico: 3×31 (propagazione riporti) + 3 (semaforo + OR uscita mux di s_{31}) = 96 porte logiche

per 4 operazioni su 32 bit



Sottrazione



In complemento a 2 diventa un'addizione: $a - b = a + \bar{b} + 1 = 1 + a + \bar{b}$

Esempio: $s = 3 - 4$; su 3 bit

3 -> 011	011 +	
-4 -> 100 in complemento a 2	100 =	
-1 -> 111 in complemento a 2	111	

Posso scrivere il numero negativo in complemento a 2 come somma:

Passo I – Complemento a 1	4 -> 100	numero positivo: b
	011+	complemento a 1: $\bar{b} +$
Passo II – Sommo + 1	1=	sommo 1: $1 =$
Risultato – Complemento a 2	100	risultato $-b$

Posso quindi scrivere: $\bar{b} = \bar{b} + 1$



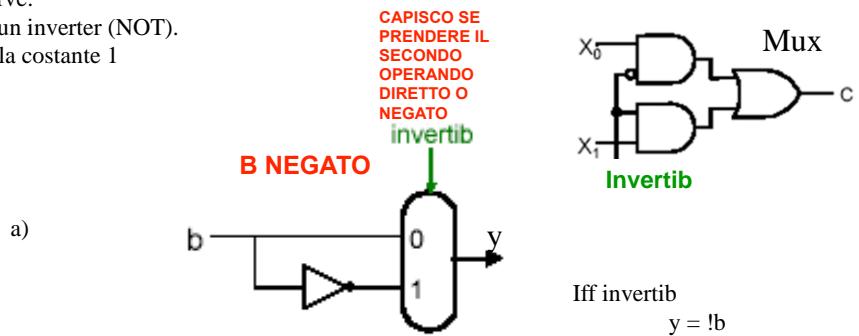
Sottrazione



In complemento a 2 diventa un'addizione: $a - b = a + \bar{b} + 1$

Serve:

- a) un inverter (NOT).
- b) la costante 1



Aggiunge 2 porte logiche al cammino critico di ogni stadio.
Aggiungere 2 porte per ogni stadio.

b) Da dove prendo la costante 1?



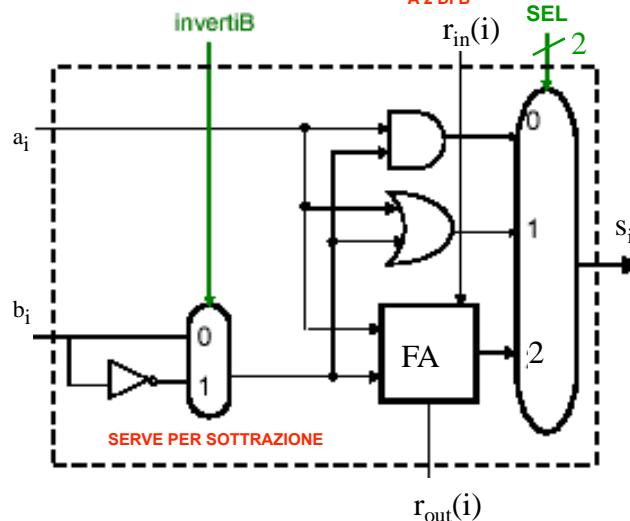
Sottrazione - ALU_i

è IL +1 CHE MI SERVE PER FARE COMPLEMENTO A 2 DI B



Operazioni:

- AND
- OR
- SOMMA
- SOTTRAZIONE

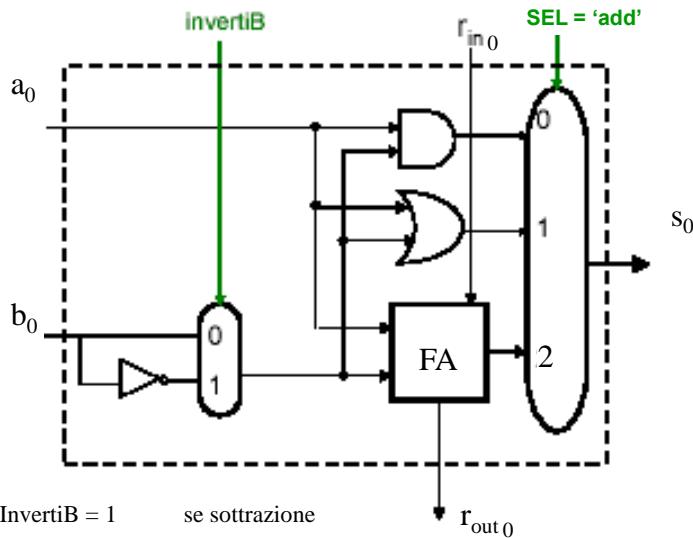


$$r_{in}(i) = r_{out}(i-1) \quad i = 1, 2, 3, \dots, 31 \\ InvertiB = 1$$

$$i \neq 0 \text{ (tranne che nel primo stadio)} \\ \text{se sottrazione}$$



Sottrazione – primo stadio: ALU₀



$$r_{in}(0) = \text{InvertiB} = 1 \quad \text{se sottrazione}$$

(occorre utilizzare un full adder anche per il bit meno significativo con $r_{in0} = 1$).
Effettuo quindi la somma di 1 con la somma della prima coppia di bit.

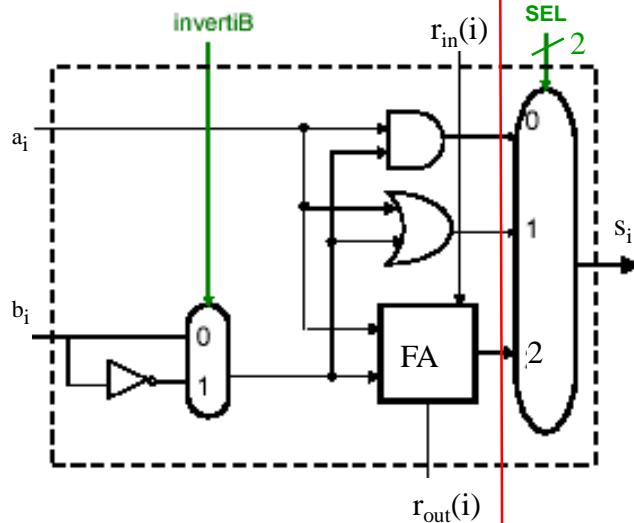


Operazioni - ALU_i



E' possibile programmare questa ALU per eseguire:

- 1) $a \text{ AND } !b$
- 2) $a \text{ OR } !b$
- 3) $a + b$
- 4) $a - b$



La parte di calcolo è comunque separata dalla parte di selezione

ALU COMPLETA



Sottrazione: ALU a 32 bit

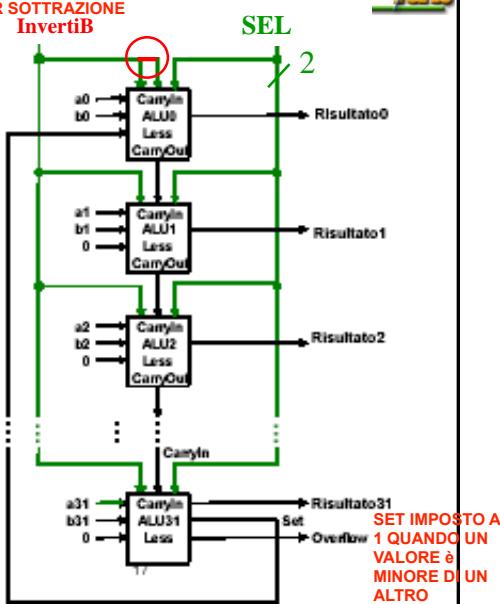
PER SOTTRAZIONE
InvertiB

$r_{in}(0) = \text{InvertiB} = 1$
se sottrazione

- AND
- OR
- SOMMA
- SOTTRAZIONE

From_UC	SEL	r_0	InvertiB
And	And	0	0
Or	Or	0	0
Somma	Add	0	0
Sottr.	Add	1	1

InvertiB e r_0 sono lo stesso segnale, si può ancora ottimizzare.
 $r_{in}(0)$ entra solo in ALU_0
InvertiB entra in tutte le ALU_i



SET IMPOSTO A 1 QUANDO UN VALORE è MINORE DI UN ALTRO

A.A. 49/51 <http://borgheze.di.unimi.it/>



ALU a 32 bit con CLA

• Come realizzare una ALU a 32 bit con:

- Porte OR
- Porte AND
- CLA a 4 bit?

Definire complessità e cammino critico

Notate che l'inverter su b aggiunge complessità e cammino critico.

A.A. 2023-2024 50/51 <http://borgheze.di.unimi.it/>



Sommario



Moltiplicatori

ALU



ALU + Bistabili

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento Patterson: sezioni B.7 & B.8.



Sommario

ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Sottrazione: ALU a 32 bit



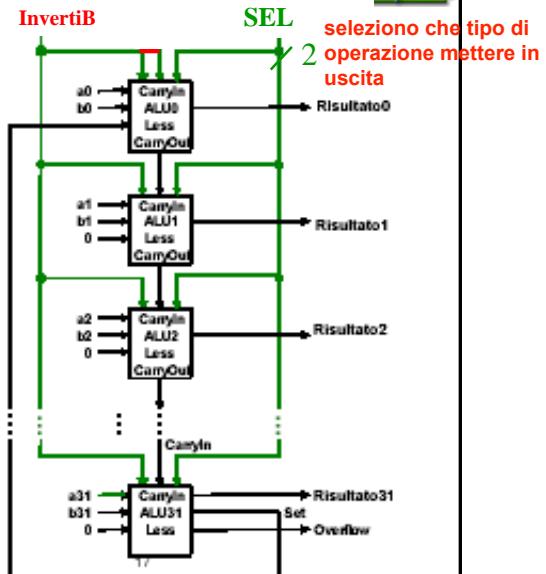
$r_{in}(0) = \text{InvertiB} = 1$
se sottrazione

- AND
- OR
- SOMMA
- SOTTRAZIONE

InvertiB e r_0 sono lo stesso segnale, si può ancora ottimizzare.

$r_{in}(0)$ entra solo in ALU₀

InvertiB entra in tutte le ALU_i



Confronto



controllo di flusso

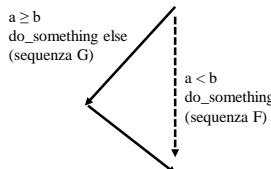
Fondamentale per **dirigere il flusso** di esecuzione (test, cicli....). Costrutto «if»:

if (**Condizione**) then
 Sequenza istruzioni F
else
 Sequenza istruzioni G

istruzione SET

Condizione può essere:

- A = B
- A \neq B
- A < B
- A > B
- A \geq B
- A \leq B



In MIPS solamente le condizioni A = B e A \neq B vengono utilizzate **direttamente** per dirigere il flusso. Corrispondono alle istruzioni di salto condizionato: beq e bne.

Come vengono trattate le condizioni A < B e A > B?



Confronto di minoranza



Il **confronto di minoranza** produce una variabile che assume il valore: VERO o FALSO, flag = {0,1}

```
if (a less_than b) then  
    s = flag = TRUE = 1;  
else  
    s = flag = FALSE = 0;
```

```
if (a < b) then  
    flag = 1;  
else  
    flag = 0;           if (b ≥ a) then  
                        flag = 1;  
                        else  
                        flag = 0;
```

Occorre che la ALU sia attrezzata per eseguire questo confronto di minoranza ($a < b$).
Questa sarà una quinta operazione possibile: AND, OR, Somma, Sottrazione, SLT
(istruzione: *Set on Less Than*).

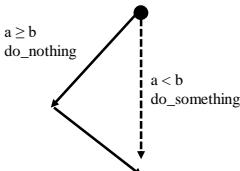


Utilizzo del confronto di minoranza



Linguaggio ad alto livello

```
if (a < b) then  
    do_something;  
end;
```



```
if (a < b) then  
    flag = 1;  
else  
    flag = 0;
```

```
if (s == 0)          // (s = 0) => (a ≥ b) => salta  
    do_something;  
end;  
dopo:
```

Assembler
slt regdest, reg1, reg2
regdest contiene il flag

beq regdest, 0, dopo
do something
dopo:

2 istruzioni assembler



Come sviluppare la comparazione: slt - I?



$a < b \iff a - b < 0$

```

if (a < b) then
    flag = 1
else
    flag = 0

```

mi interessa sapere se il risultato è minore di 0 o no

Si controlla che il primo bit del risultato della sottrazione (bit di segno) sia = 1.

Esempio: consideriamo numeri su 4 bit: 3 per l'ampiezza e 1 per il segno.

$s = 3 - 4$; su 4 bit

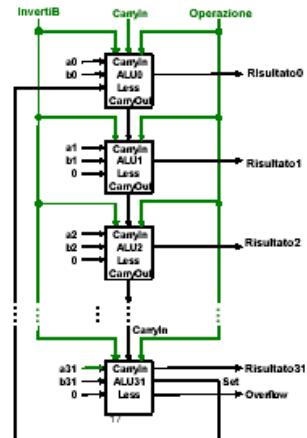
3 -> 0011

-4 -> 1100 in complemento a 2

-1 -> 1111 in complemento a 2

$$\begin{array}{r}
0011 \\
1100 \\
\hline
1111
\end{array}$$

MSB = bit di segno



Come sviluppare la comparazione - II?



```

if ((a - b) < 0) then
    flag = 1
    else
        flag = 0

```

-> Flag (bit singolo che segnala un evento)
assume i valori {Falso, Vero} - {0, 1}

Valori possibili in uscita della ALU: $s = \text{flag} = \{0, 1\}$ ma su 32 bit!!

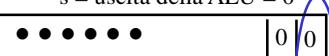
Risultato verso l'esterno

Flag = $s = 0$

$a \geq b$

MSB

$s = \text{uscita della ALU} = 0$



Risultato verso l'esterno

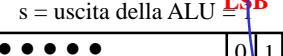
Flag = $s = 1$

$a < b$

MSB

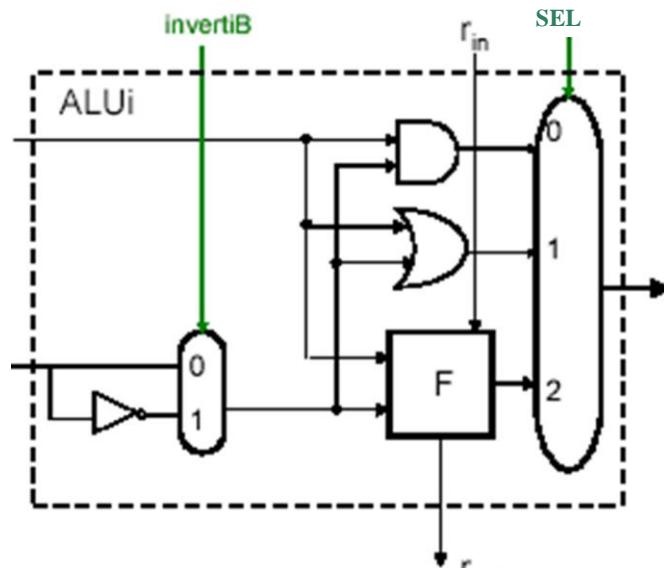
$s = \text{uscita della ALU} = 1$

LSB





Come modificare le ALU a 1 bit?

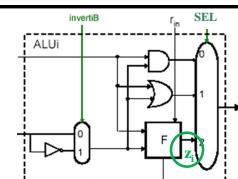


Come sviluppare la comparazione (riassunto)



if $((a - b) < 0)$ then
flag = 1
 else
flag = 0

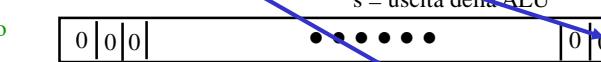
se il bit di segno indica
numero maggiore di 0 vuol
dire che $A > B$



z₁ = uscita dei sommatori



Risultato verso l'esterno
s = Flag = 0



Risultato verso l'esterno
s = Flag = 1





Comparatore – ALUi ($i \neq 0$)



Prevedo una quarta operazione: SLT

AND (0 - 00)

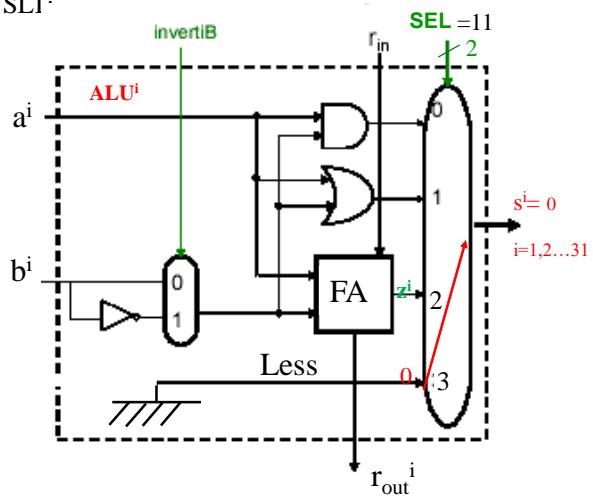
OR (1 - 01)

Somma/Sottrazione (2 - 10)

SLT (3 - 11)

SEL = 11 (ingresso #3 al mux)

InvertiB = 1 (sottrazione)



$$\text{Less}(i) = 0 \quad i = 1, 2, 3, \dots, 31 \quad i \neq 0$$



Comparatore – ALU⁰ : ALU³¹

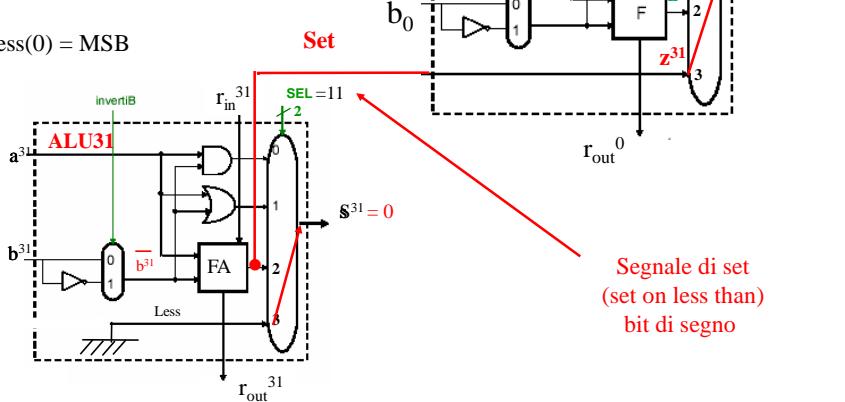


$invertiB = r_{in}^0 = 1$ (sottrazione)

$SEL = 11$ (ingresso #3 al mux)

if $((a - b) < 0)$ then
s flag = 1
 else
s flag = 0

Less(0) = MSB





Comparatore – ALU⁰ : ALU³¹



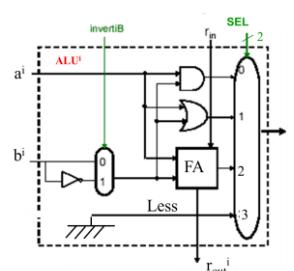
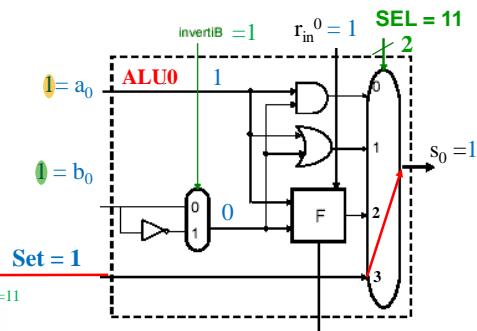
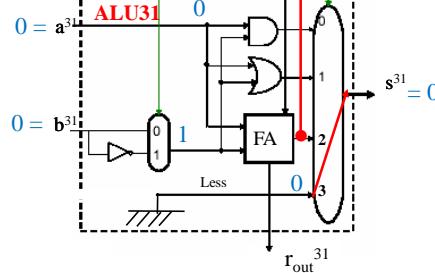
InvertiB = $r_{in}^0 = 1$ (sottrazione)
SEL = 11 (ingresso #3 al mux)

Esempio:

$a = 5 - 0101_2$
 $b = 7 - 1011_2$
 $a - b = -2 - 111\dots110_2$

if $((a - b) < 0)$ then
s flag = 1
else
s flag = 0

essendo numeri positivi $a31 \dots b31$ perché sottrazione
 $r_{in}^0 = 1$ saranno 0



Comparazione - ALU completa a 32 bit

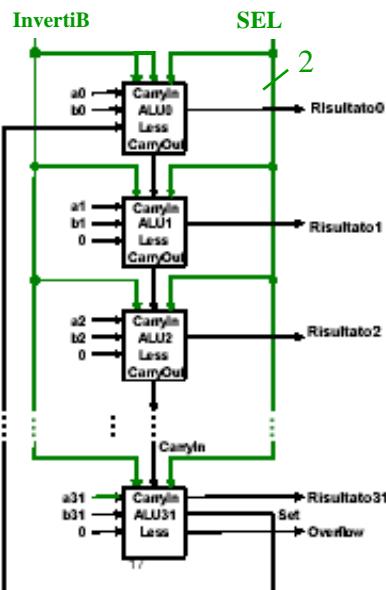


- AND
- OR
- Somma/Sottrazione
- Comparazione ($a < b$)

Come implemento ($a > b$)?
Come implemento ($a \geq b$)?

Io implemento come $B < A$

quindi stessa implementazione





Overflow decimale



$a + b = c$ - codifica su 2 cifre,

$a = 12, b = 83 \Rightarrow$ Not Overflow.

$$\begin{array}{r} 012+ \\ 083= \\ \hline 095 \end{array}$$

$a = 19, b = 83 \Rightarrow$ Overflow

$$\begin{array}{r} 019+ \\ 083= \\ \hline 102 \end{array}$$

condizione che se non gestita provoca degli errori

Si può avere overflow con la differenza?



Overflow binario



$a + b = c$ - codifica su 4 cifre binarie, di cui 1 per il segno.

No overflow

$$\begin{array}{r} 0000 \\ 0010+ \\ 0011= \\ \hline 0101 \end{array}$$

2+ 3= 5

Overflow

0110	$0010+$	$2+$
$0111=$	$0111=$	$7=$ errore
\hline	\hline	\hline
1001	1001	-7 in complemento a 2??
1110	0110	
$1010+$	$1010+$	-6+
$1111=$	$1011=$	-5= errore
\hline	\hline	\hline
1001	0101	+5 in complemento a 2??

Quindi si ha overflow nella somma quando $a + b = s$:

- $a \geq 0, b \geq 0$ MSB di a e $b = 0$, MSB di $s = 1$.
- $a < 0, b < 0$ MSB di a e $b = 1$, MSB di $s = 0$.



Circuito di riconoscimento dell'overflow

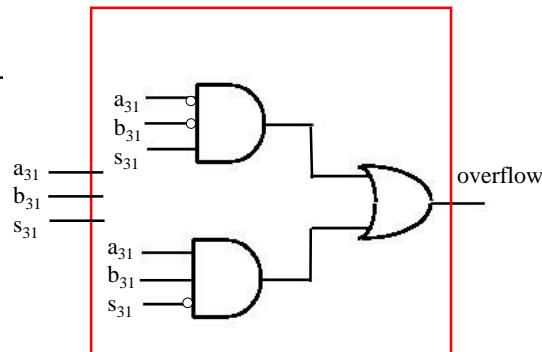


Lavora sui MSB

prim o bit	bit a	bit b	primo bit risultato	overflow
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

da positivi ho risultato negativo

da negativi ho risultato positivo

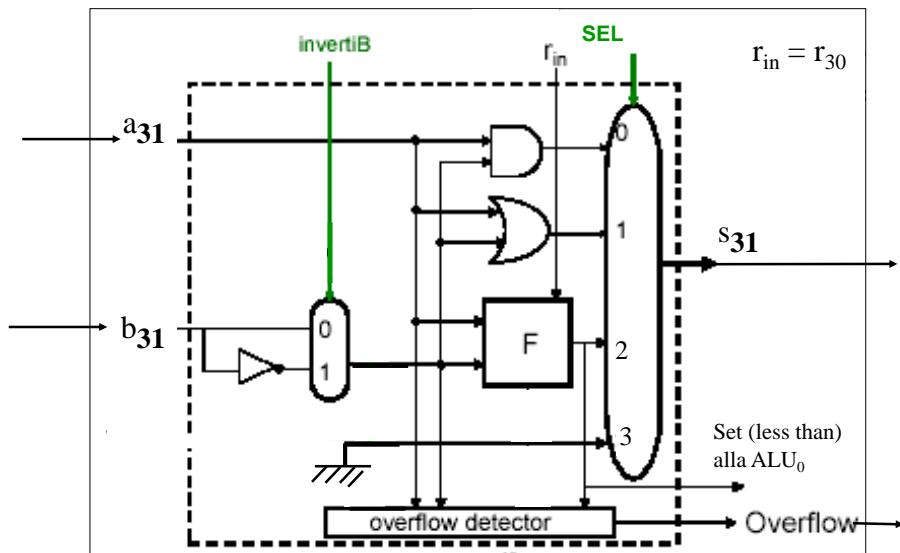


Overflow detector

$$\text{Overflow} = !a_{31}!b_{31}s_{31} + a_{31}b_{31}!s_{31}$$



ALU₃₁





Uguaglianza – prima implementazione



beq reg1, reg2 label

iff (reg1 == reg2), salta.

		$C_k = \overline{a_k \oplus b_k}$			
A_0	B_0	C_0	A_1	B_1	C_1
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	1	1	1

$C = C_0 C_1 \dots C_{n-1}$

...

comparazione di uguaglianza vera quando AND di tutti i bit = 1

Per la ALU:

- Input: a, b su 32 bit
- Output: {0,1} -> segnale di zero.

Per 32 bit occorrono le seguenti porte a 2 ingressi:

32 XOR (già contenute nella ALU)

AND a 32 ingressi

Soluzione per comparatori



Uguaglianza - seconda implementazione



beq reg1, reg2 label

iff (reg1 == reg2),

salta

then

proseguì in sequenza

iff ((rs - rt) == 0),
salta.

**comparazione di uguaglianza
facendo differenza tra A e B e
vedere se risultato = 0**

Occorre quindi:

- Impostare una differenza ($reg1 - reg2$).
- Controllare che la differenza sia = 0 $\rightarrow (reg1 - reg2) = 0$ su 32 bit $\rightarrow 0000 \dots 0000$
- Effettuare l'OR logico di tutti i bit in uscita dai sommatori $\{z_i\}$.
- L'uscita dell'OR logico = 0 \Rightarrow i due numeri sono uguali.

32

Input: a, b su 32 bit

Output è un Flag di zero: {0,1}. Zero = NOR(z_0, z_1, z_{N-1}) a 32 ingressi.

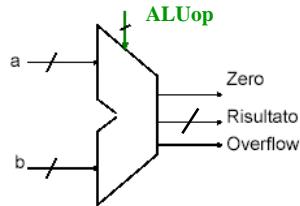


ALU a 32 bit: struttura finale

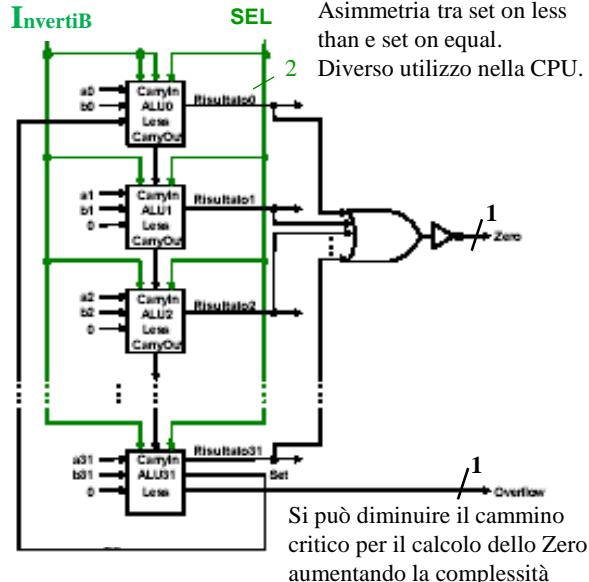


Operazioni possibili:

- AND
- OR
- Somma / Sottrazione
- Comparazione
- Test di uguaglianza



Sono evidenziate solamente le variabili visibili all'esterno.



Approcci tecnologici alla ALU.



Tre approcci tecnologici alla costruzione di una ALU (e di una CPU):

- **Approccio strutturato.** Analizzato in questa lezione.
- **Approccio hardware programmabile (e.g. PAL).** Ad ogni operazione corrisponde un circuito combinatorio specifico.
- **Approccio ROM.** E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabili per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- **Approccio firmware (firm = stabile), o microprogrammato.** Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate a partire dall'algoritmo che le implementa.



Approccio ROM alla ALU

**Input:**

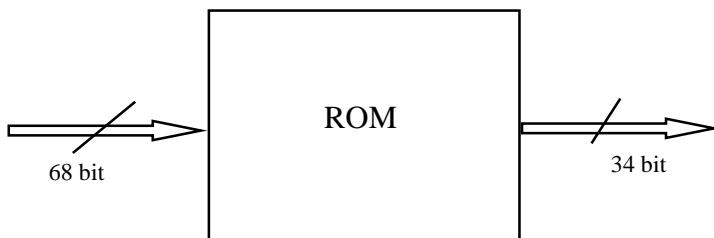
- A n bit
 - B n bit
 - SEL k bit
 - InvertiB: 1 bit
- Totale: $2^*n + k + 1$ bit**

Output:

- s n bit
- zero 1 bit
- overflow 1 bit

Totale: $n + 2$ bit

Per dati su 32 bit, 5 operazioni ($k = 4$):



Capacità della ROM: $2^{68} = 2.95.. \times 10^{20}$ parole di 34 bit!

A.A. 202 (Capacità della ROM per dati su 4 bit, 5 operazioni: $2^{12} = 4,098$ parole di 6 bit) <http://borgheze.di.unimi.it/>



Sommario



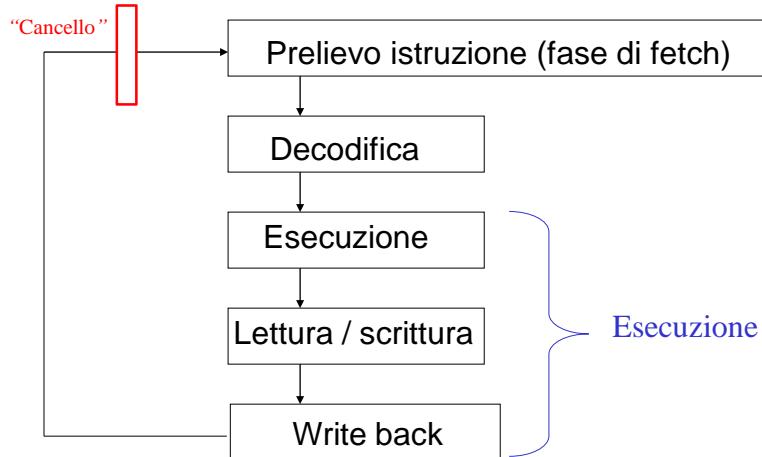
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Dispositivi di sincronizzazione



Circuiti asincroni

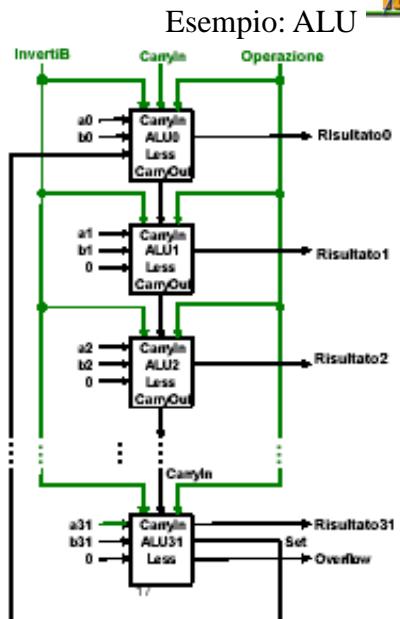


Architettura logica asincrona:

L'elaborazione e propagazione dei segnali avviene in modo incontrollato, secondo le velocità di propagazione dei circuiti.

Progetto asincrono: Devo progettare il circuito in modo che nessun transitorio/cammino critico causi problemi → analisi di tutti i transitori critici possibili.

Improprio per circuiti con feed-back (retroazione).





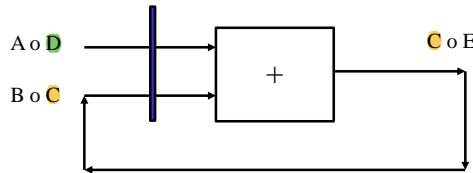
Circuiti retro-azionati e sincronizzazione



Esempio:

$$C = A + B$$

$$E = D + C$$



I problemi dei circuiti retroazionati

- Quando posso calcolare E con lo stesso sommatore?
- Quando devo fare passare A e B oppure A e C?

Soluzione

“Cancello” davanti all’ingresso del sommatore prima della seconda somma.

Occorre una sincronizzazione dell’attività del sommatore.



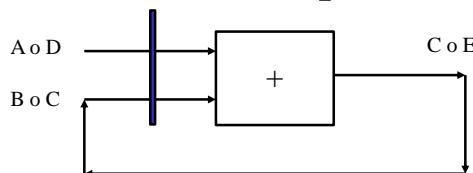
Sincronizzazione temporizzata



Esempio:

$$C = A + B$$

$$E = D + C$$



I problemi dei circuiti retroazionati

- Quando posso calcolare E con lo stesso sommatore?
- Quando devo fare passare A e B oppure A e C?

Occorre una sincronizzazione dell’attività del sommatore.

Dobbiamo essere ragionevolmente sicuri che il risultato sia stato calcolato ed utilizzato.

Soluzione

- “Cancello” davanti all’ingresso del sommatore prima della seconda somma.
- Il cancello viene **temporizzato**: si apre ogni intervallo di tempo prefissato.



Circuiti sincroni



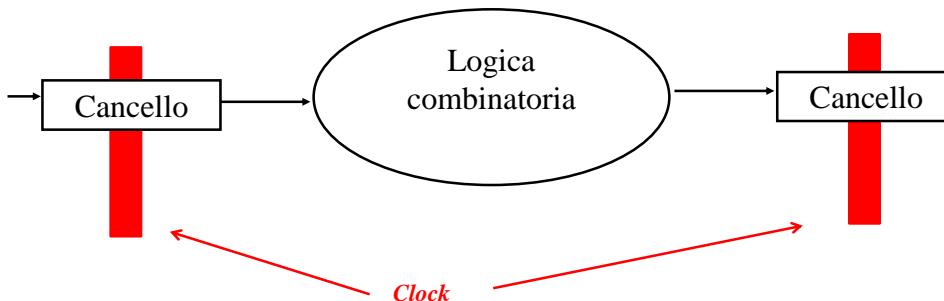
- **Architettura logica sincrona:**

Le fasi di elaborazione sono scandite da un orologio comune a tutto il circuito (**clock**).

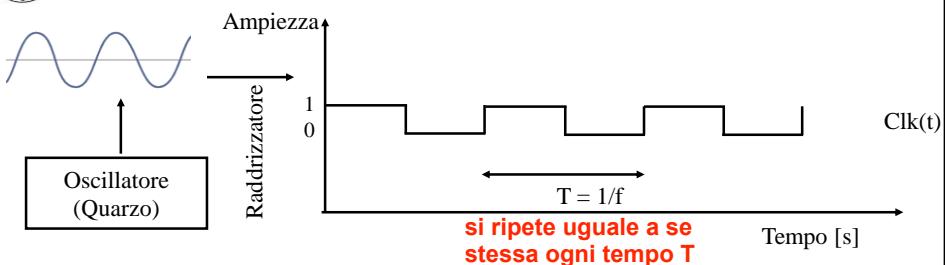
– Ad ogni fase di clock, la parte combinatoria del circuito ha tempo di elaborare (i segnali di percorrere il cammino critico) e quindi il circuito ha il tempo di stabilizzarsi (transistori critici). Questo deve avvenire entro il “**tick**” successivo.

– Progetto sincrono: il controllo dei transistori/cammini critici è limitato alla parte di circuito tra due **cancelli** (porte di **sincronizzazione**)

Esempio: CPU



Clock di sistema



Frequenza: numero di cicli/s Si misura in Hertz, Hz.

Periodo: tempo necessario a completare 1 ciclo Si misura in secondi, s.

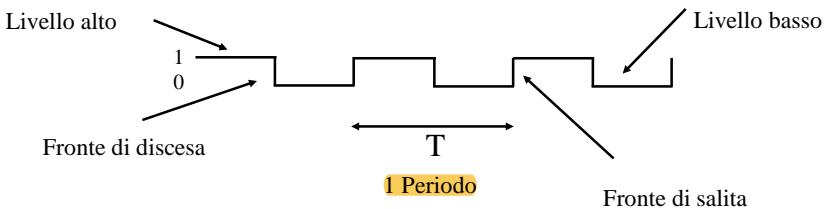
$$T = 1/f$$

$$1\text{ns} = 1/1 \text{ GHz}$$

Tempo di salita e discesa trascurabile rispetto al tempo di commutazione delle porte logiche.



Utilizzo del clock



•Metodologia sensibile ai livelli:

Le variazioni di stato possono avvenire per tutto il periodo in cui il clock è al livello alto (basso).

•Metodologia sensibile ai fronti:

Le variazioni di stato avvengono solamente in corrispondenza di un fronte di clock. Noi adotteremo questa metodologia.



Architetture sequenziali



•I circuiti combinatori **non hanno memoria e non hanno bisogno di sincronizzazione**. Gli output al tempo t dipendono unicamente dagli input al tempo t che provengono dall'esterno: $y^{t+1} = f(u^{t+1})$

1) Sono necessari circuiti **con memoria**, per consentire comportamenti diversi a seconda della situazione dell'architettura. Nella memoria viene memorizzato lo **stato** del sistema che riassume la storia passata (e.g. Register file, memoria, PC sono elementi di stato di una CPU). **Sono alla base delle architetture retro-azionate**.

• Sono necessari dispositivi di **sincronizzazione** (cancelli) per eseguire operazioni **sequenzialmente** e il risultato di un'operazione è l'input dell'operazione successiva (architetture con feed-back (e.g. CPU, Distributore di bibite)).

Memoria e sincronizzazione sono assolte nei circuiti digitali dai **bistabili**.



Bistabili: latch e flip-flop



Elemento base della memoria è il **bistabile**: dispositivo in grado di mantenere *indefinitamente* uno dei due possibili valori di output: (0 o 1).

Il suo valore di uscita coincide con lo stato. L'uscita al tempo t, dipende:

- dallo stato (uscita) al tempo t-1
- dal valore presente agli input.

Tipi di bistabili:

- Bistabili non temporizzati (asincroni, **latch asincroni**) / temporizzati (sincroni).
- Bistabili sincroni che commutano sul livello del clock (**latch**) o sul fronte (**flip-flop**).



Sommario



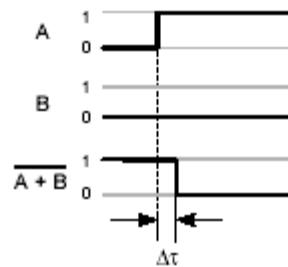
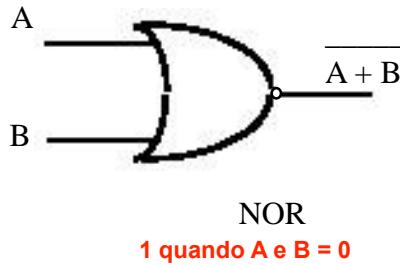
ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Principio di funzionamento



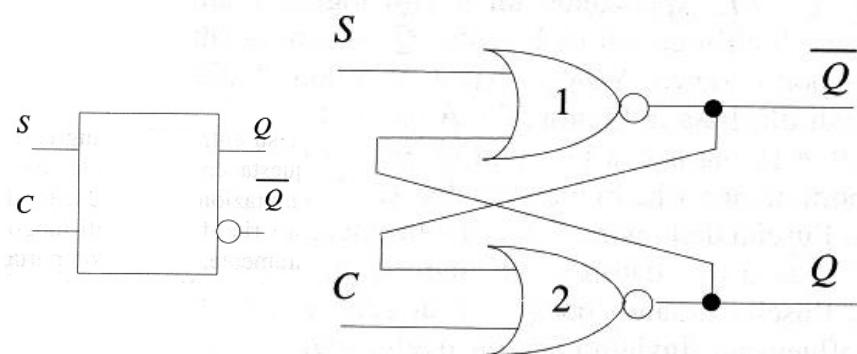
Il **ritardo**, $\Delta\tau$, introdotto tra la commutazione dell'input e la commutazione dell'output è alla base del funzionamento di un bistabile.



Latch asincrono SC (o SR)



set clear set reset

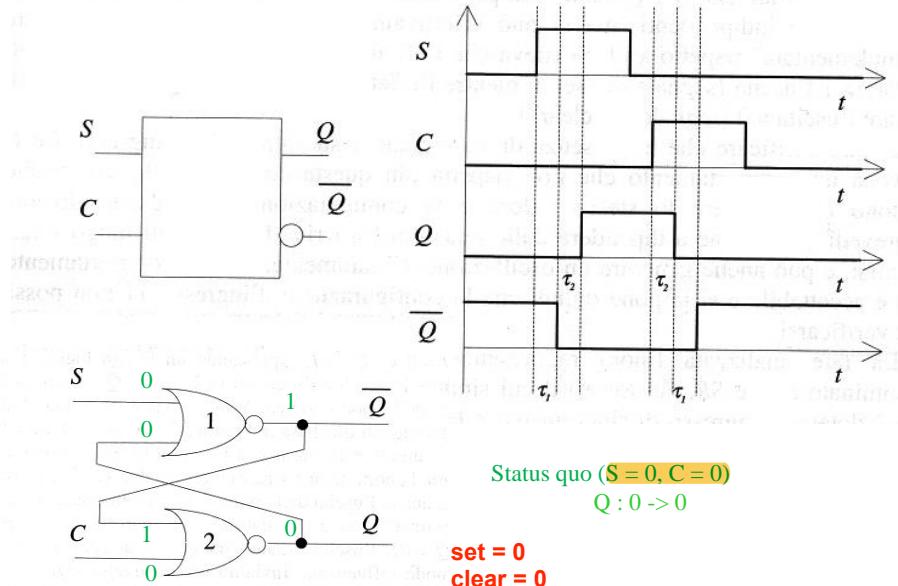


Una coppia di porte NOR retro-azionate può memorizzare un bit!

Cioè se in ingresso arriva $S = C = 0$, l'uscita non commuta e rimane al valore attuale ($Q = 0$; oppure $Q = 1$)



Funzionamento del circuito SC – Status quo



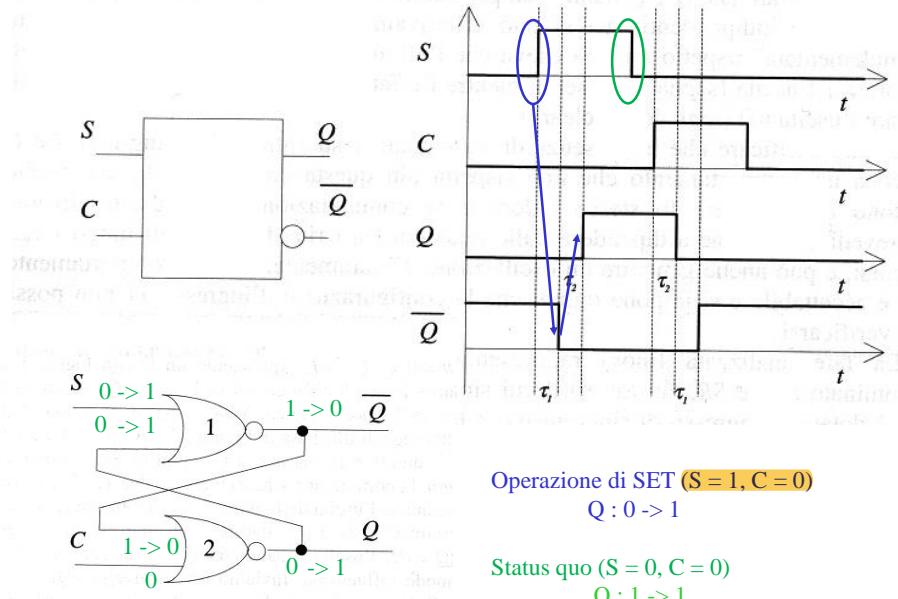
A.A. 2023-2024

37/48

<http://borghese.di.unimi.it/>



Funzionamento del circuito SC - set



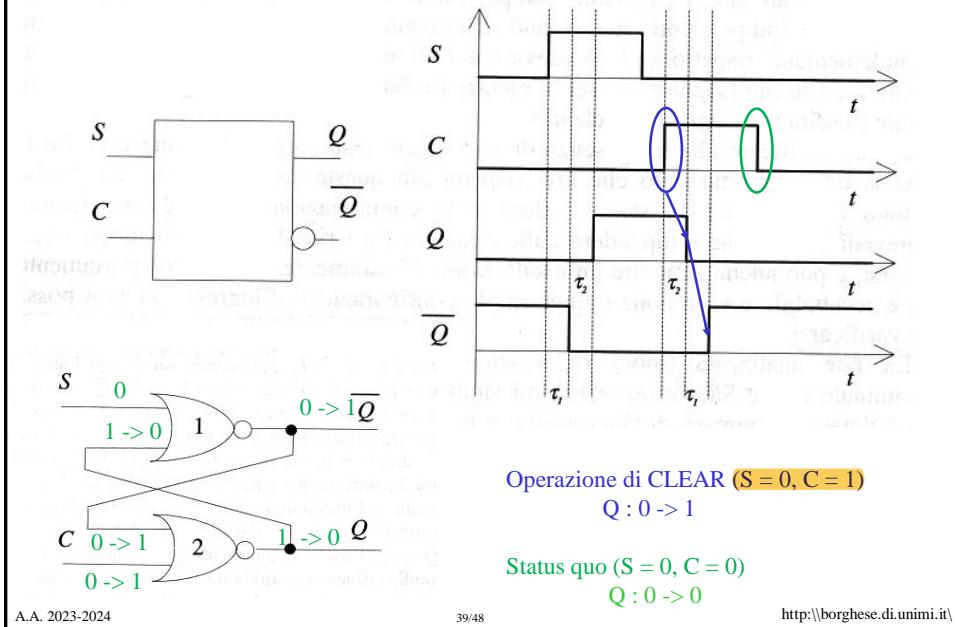
A.A. 2023-2024

38/48

<http://borghese.di.unimi.it/>



Funzionamento del circuito SC - clear



Osservazioni



Il circuito è molto semplice, costituito da 2 NOR retroazionati

Accetta in ingresso le configurazioni

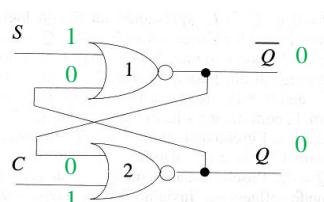
SC

00 (status quo)

01 (clear)

10 (set)

Non accetta la configurazione 11
 \rightarrow in uscita avessimo 00 \rightarrow errore



Come posso evitare una condizione di errore in uscita per ingresso S=C=1?



Tabella della verità del latch



Se considero Q (**lo stato**) e S e C ingressi, ottengo la **tabella della verità di Q***:

$$\xrightarrow{\hspace{1cm}} Q^* = f(Q, S, C)$$

Q è l'uscita del latch: **stato presente**, Q_t
 Q* è il valore dell'uscita al tempo successivo:
stato prossimo, Q_{t+1}

S e C variabili esterne,
Q variabile interna

set = 1
clear = 1
non si può verificare

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

x è uscita indifferente



Tabella delle transizioni



$$Q^* = f(Q, S, C)$$

Variabile di Stato (interna)

Variabili di Ingresso (esterne)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

ingressi interni - stato Q	ingressi esterni			
	SC = 00	SC = 01	SC = 10	SC = 11
0	0	0	1	X
1	1	No change ($Q^* = Q$) Memory	0	1

No change
($Q^* = Q$)
Memory

Clear
Reset
Write 0

Set
Write 1



Tabella della verità del latch ($X = \{0,0\}$)



Se considero Q (**lo stato**) e S e C ingressi, ottengo la **tabella della verità di Q^*** :

$$\xrightarrow{\hspace{1cm}} Q^* = f(Q, S, C)$$

Q è l'uscita del latch: **stato presente**, Q_t
 Q^* è il valore dell'uscita al tempo successivo:
stato prossimo, Q_{t+1}

$$Q^* = \overline{SC}Q + \overline{SC}Q + \overline{SC}Q = \overline{SC}Q + \overline{SC}(Q+Q) =$$

$$\overline{SC}Q + \overline{SC}$$

Status
Quo Set

If ($S=C=1$) then
 $Q^* = 0$ (non dipende da Q)

S	C	Q	Q^*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



Confronto tra i 2 circuiti

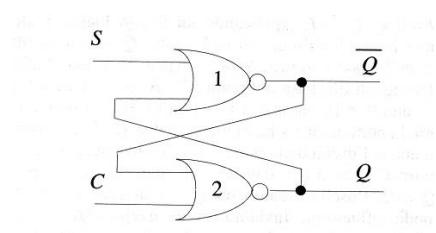
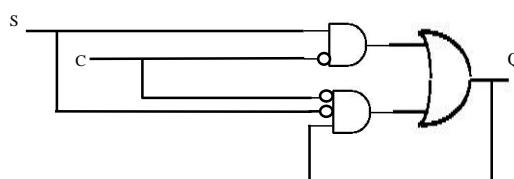




Tabella della verità del latch (X= {1,1})



Impostando X = 1 1, si ottiene:

$$\begin{aligned}
 Q^* &= \bar{\bar{S}}\bar{\bar{C}}Q + \bar{\bar{S}}\bar{\bar{C}}\bar{Q} + \bar{\bar{S}}\bar{C}\bar{Q} + \bar{S}\bar{C}\bar{Q} + S\bar{C}Q = \\
 &= \bar{\bar{S}}\bar{\bar{C}}Q + SC(Q+Q) + SC(\bar{Q}+\bar{Q}) = \\
 &= \bar{\bar{S}}\bar{\bar{C}}Q + S(C+C) = \\
 &= \bar{\bar{S}}\bar{\bar{C}}Q + S \quad (\text{per assorbimento di } \bar{S})
 \end{aligned}$$

↑ ↓
Status quo Set

(SC = 11 -> Q* = 1 -> Set)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	X=1
1	1	1	X=1



Tabella della verità del latch (X= {0,1})



Impostando X = {0 1}, si ottiene:

$$\begin{aligned}
 Q^* &= \bar{\bar{S}}\bar{\bar{C}}Q + \bar{\bar{S}}\bar{\bar{C}}\bar{Q} + \bar{\bar{S}}\bar{C}\bar{Q} + S\bar{C}Q = \\
 &= \bar{\bar{S}}\bar{\bar{C}}Q + S C Q + S \bar{\bar{C}}(Q + \bar{Q}) = \\
 &= (\bar{S}\bar{\bar{C}})Q + S \bar{\bar{C}}
 \end{aligned}$$

↑ ↓
Status quo Set

(SC = 11 -> Q* = Q -> Status quo)

S	C	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	X=0
1	1	1	X=1



Tabella delle eccitazioni



Q	Q*	S	C
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Data la transizione $Q \rightarrow Q^*$, qual'è la coppia di valori di ingresso che la determina?

$$(Q, Q^*) = f(S, C)$$



Sommario



ALU: Comparazione, Overflow, Test di uguaglianza

Bistabili

Latch SC



Latch sincroni e flip-flop

Prof. Alberto Borghese
Dipartimento Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimento Patterson: sezioni B.7 & B.8.



Sommario



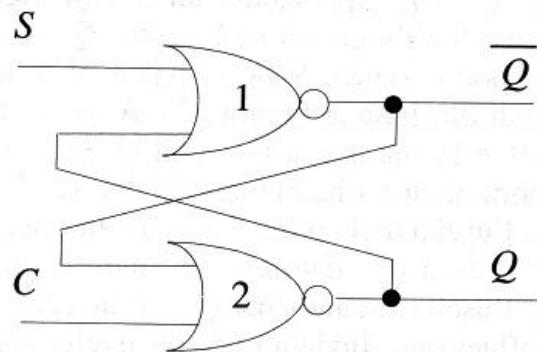
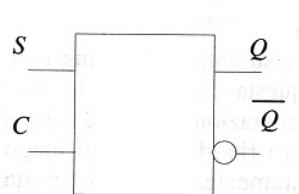
Latch sincroni SR

Latch sincroni D

Flip-flop



Latch asincrono SC (o SR)



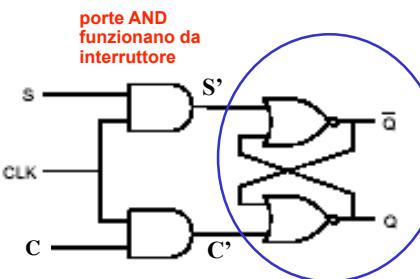
S = set

C = clear (R = reset)

Una coppia di porte NOR retro-azionate può memorizzare un bit.



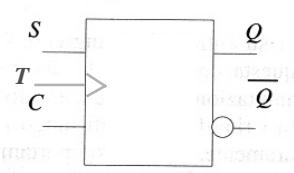
Il latch SC sincrono



Si inserisce tra il clock e gli ingressi un AND che funge da interruttore.

If (**CLK** = H = 1) then
 $S' = S; C' = C$

If (**CLK** = L = 0) then **cancelli chiusi**
 $S' = C' = 0$ **interruttore spento**

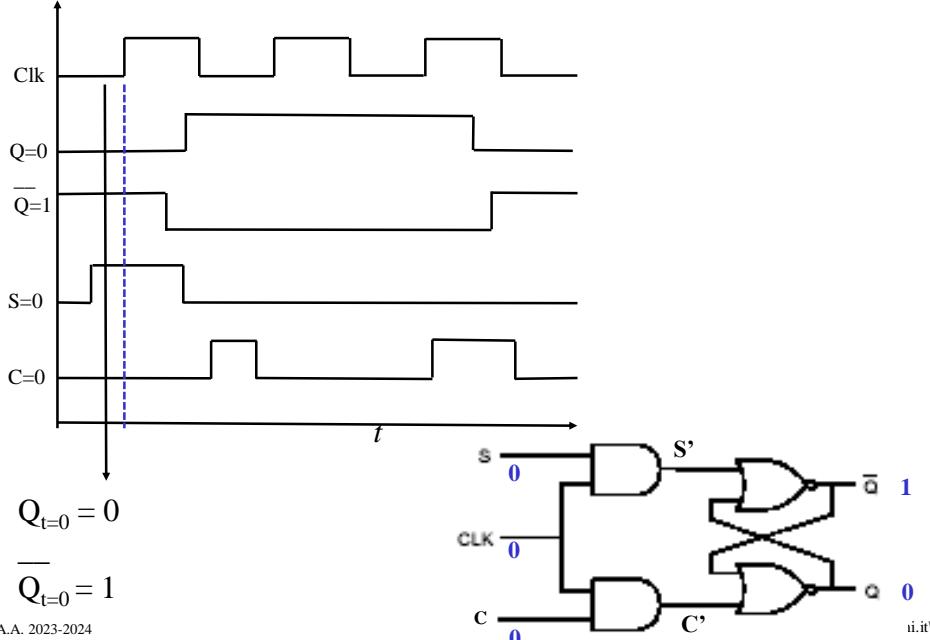


Solo quando il clock è alto i “cancelli” rappresentati dagli AND fanno passare gli input (collegano l’altro ingresso dell’AND con l’uscita). Cancelli di «abilitazione» del latch.

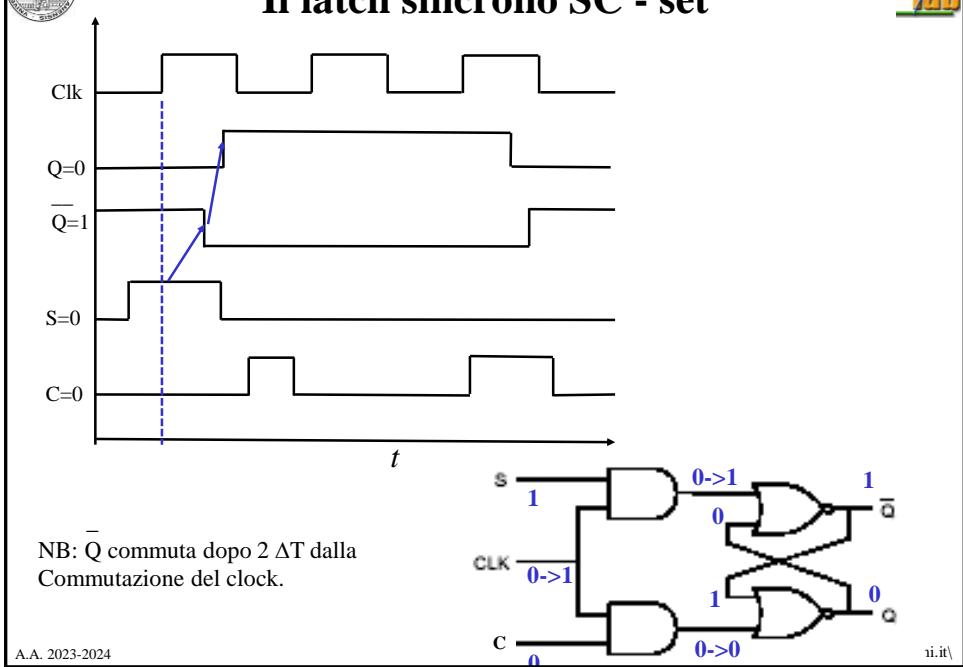
Latch asincrono, sincronizzato.



Il latch sincrono nel tempo: clock basso

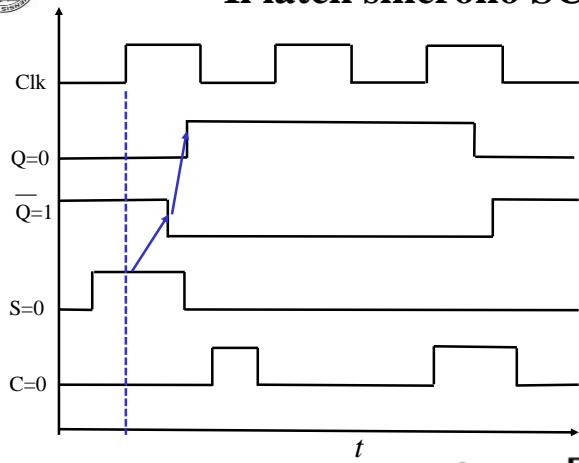


Il latch sincrono SC - set



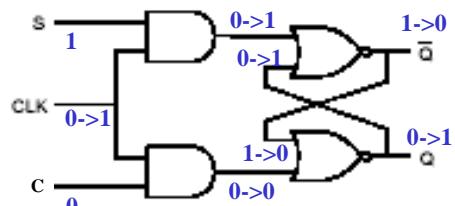


Il latch sincrono SC - set



NB: \bar{Q} commuta dopo $2 \Delta T$ dalla
Commutazione del clock.

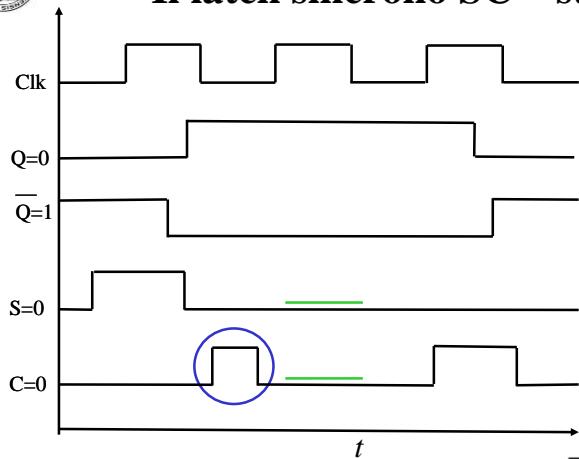
A.A. 2023-2024



ui.it\



Il latch sincrono SC – status quo



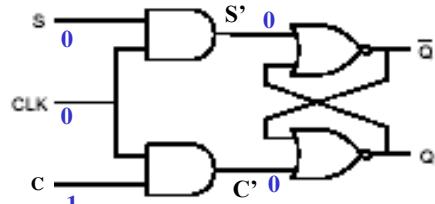
Status quo quando $S'=C'=0$:

$S = C = 0$

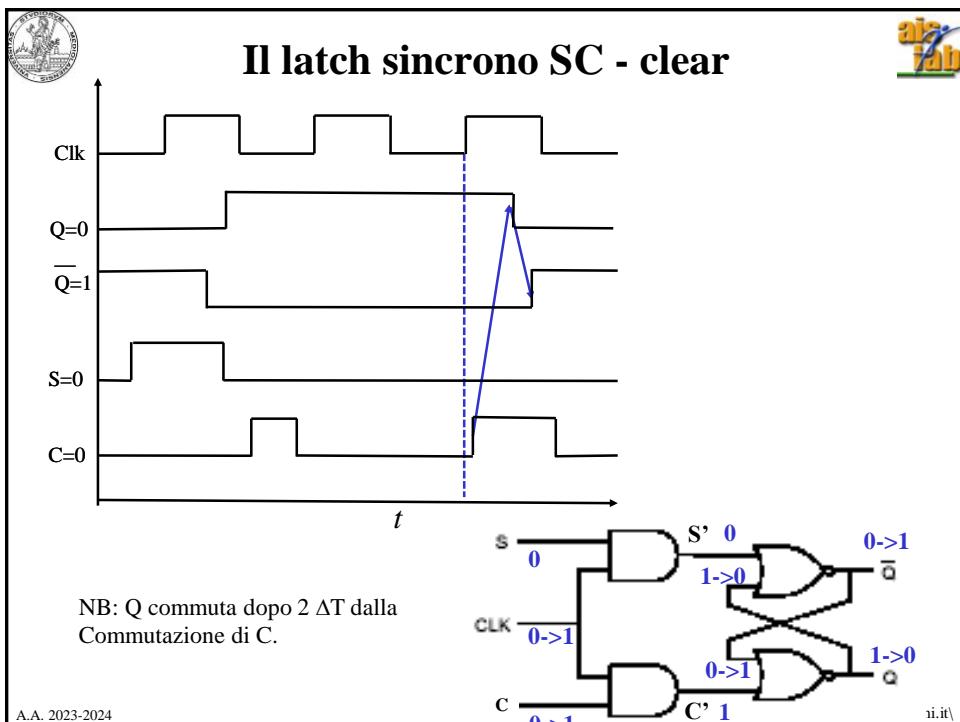
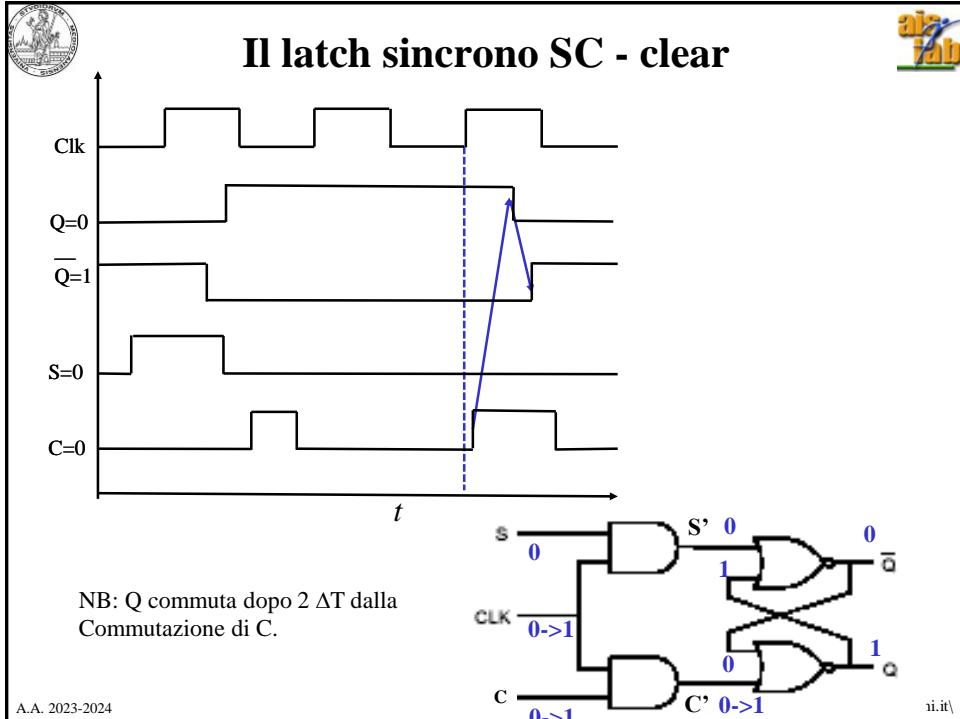
or

$CLK = 0$

A.A. 2023-2024



ui.it\

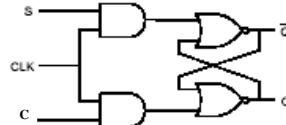




T	Q	S	C	Q*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	X

A.A. 2023-2024

Tabella della verità e tabella di transizione



TQ	SC = 00	SC = 01	SC = 10	SC = 11
00	0	0	0	0
01	1	1	1	1
10	0	0	1	X
11	1	0	1	X

$$Q^* = f(S, C, Q, T)$$

Q è l'uscita del latch: **stato presente**.

Q* è il valore dell'uscita al tempo successivo: **stato prossimo**.

11/47

<http://borgheze.di.unimi.it/>



T	Q	S	C	Q*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X=0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	X=0

A.A. 2023-2024

Tabella della verità - I con X = 0



$$Q^* = f(S, C, Q, T)$$

TQ	SC = 00	SC = 01	SC = 10	SC = 11
00	0	0	0	0
01	1	1	1	1
10	0	0	1	X=0
11	1	0	1	X=0

SOP

$$Q^* = TQSC + TQSC + TQSC + TQSC + TQSC + TQSC +$$

$$\bar{T}Q\bar{S}C =$$

$$= \bar{T}Q\bar{C} + \bar{T}Q\bar{S}C + \bar{T}Q\bar{C} + \bar{T}S\bar{C} =$$

$$= \bar{T}Q + \bar{T}Q\bar{S}C + \bar{T}S\bar{C} =$$

$$TSC = 1 \rightarrow Q^* = 0$$

$$\begin{array}{l} \text{Status quo} \\ \text{Set} \end{array}$$

<http://borgheze.di.unimi.it/>

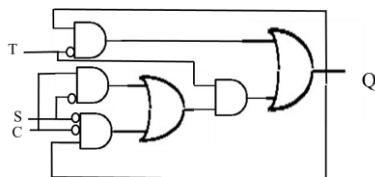


Circuito SOP semplificata

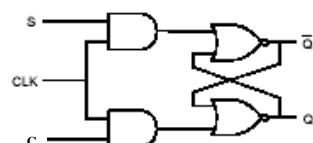


TQ	SC = 00	SC = 01	SC = 10	SC = 11
00	0	0	0	0
01	1	1	1	1
10	0	0	1	X=0
11	1	0	1	X=0

$$Q^* = \bar{T}Q + T(QSC + SC)$$



CC = 5 CO = 7



CC = 3 CO = 4
calcoliamo anche !Q



T	Q	S	C	Q*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X=1
1	1	0	0	1
1	1	1	0	1
1	1	1	1	X=1

Tabella della verità - II

con X = 1



$$Q^* = f(S, C, Q, T)$$

TQ	SC = 00	SC = 01	SC = 10	SC = 11
00	0	0	0	0
01	1	1	1	1
10	0	0	1	X=1
11	1	0	1	X=1

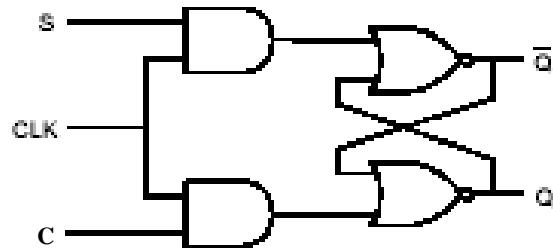
$$\begin{aligned}
 Q^* &= \overline{T}QSC + \overline{T}QSC + \overline{T}QSC + \overline{T}QSC + \overline{T}QSC + \overline{T}QSC + \\
 &+ \overline{T}QSC + \overline{T}QSC + \overline{T}QSC = \\
 &= \overline{T}QC + \overline{T}QSC + \overline{T}QC + \overline{T}SC + \overline{T}SC = \\
 &= \overline{T}Q + \overline{T}QSC + TS = \overline{T}Q + T(QSC + S)
 \end{aligned}$$

Status quo
(Memory)

Cf. Latch
asincrono



Latch sincrono



Latch attivo alto (**commuta sul periodo alto del clock**)



Sommario



Latch sincroni SR

Latch sincroni D

Flip-flop



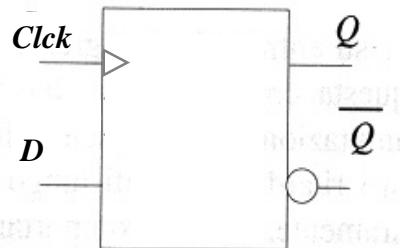
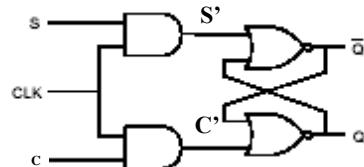
Latch D sincrono



Memorizza il valore presente all'ingresso dati quando il clock è alto.

if (CLK = 1)
then **clock alto -> uscita = ingresso dati**
 $Q^* = D$ **uscita = ingresso dati**

If (CLK = 0)
then **clock basso -> uscita = status quo**
 $Q^* = Q$ **uscita = stato attuale**



Latch trasparente sincrono



La struttura del latch D



If (CLK==1)
 $S'=D$; $C' = !D$
 $Q^*=D$

If (CLK = 0)
 $S'=C'=0$
 $Q^*=Q$

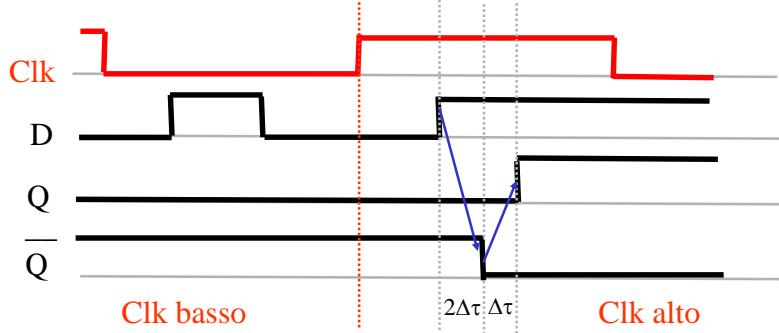
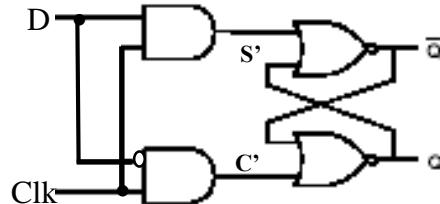




Tabella delle transizioni



$$Q^* = f(T, Q, D)$$

TQ	D = 0	D = 1
00	0	0
01	1	1
11	0	1
10	0	1

Q è l'uscita del latch: **stato presente**.

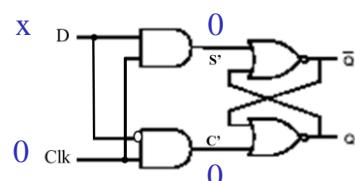
Q* è il valore dell'uscita al tempo successivo:
stato prossimo.

La funzione logica corrispondente è:

$$Q^* = TD + \bar{T}Q$$

$$Q^* = D$$

Status quo



$$Q^* = Q$$



Tabella delle transizioni



$$Q^* = f(T, Q, D)$$

TQ	D = 0	D = 1
00	0	0
01	1	1
11	0	1
10	0	1

Q è l'uscita del latch: **stato presente**.

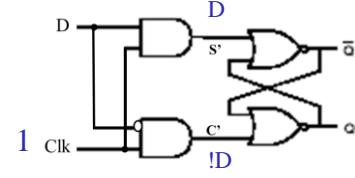
Q* è il valore dell'uscita al tempo successivo:
stato prossimo.

La funzione logica corrispondente è:

$$Q^* = TD + \bar{T}Q$$

$$Q^* = D$$

Status quo



$$Q^* = D$$

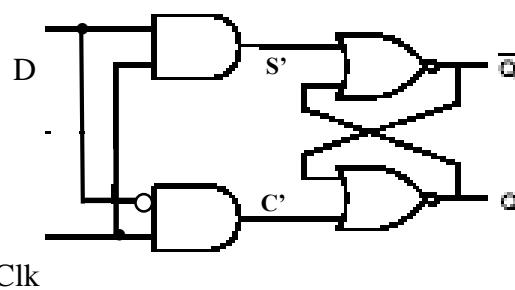
Come mai qui non si verifica la situazione S'=C'=1?
**perche non sono due linee indipendenti ma una
il reciproco dell'altra, provengono dalla linea D**



Tabella della verità



$$Q^* = f(T, Q, D)$$



T	D	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$Q^* = \bar{T} \bar{D} Q + \bar{T} D \bar{Q} + T \bar{D} \bar{Q} + T D Q =$$

$$= \bar{T} Q + T D$$

Status quo
 $Q^* = D$

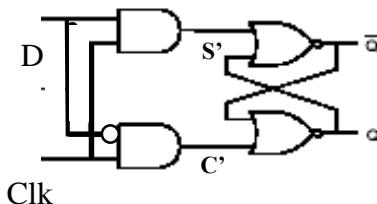
A.A. 2023-2024

21/47

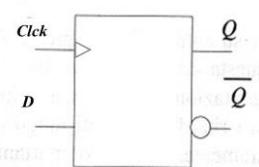
<http://borghese.di.unimi.it/>



Ottimizzazione del circuito

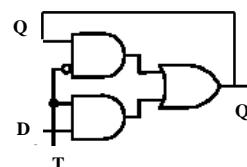


Complessità 4
Cammino critico 3

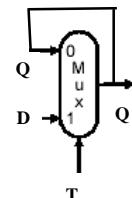


Clk come interruttore che pilota un mux:

$$Q^* = \bar{T} Q + T D$$



Complessità 3
Cammino critico 2



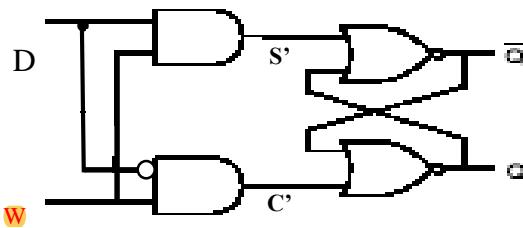
A.A. 2023-2024

22/47

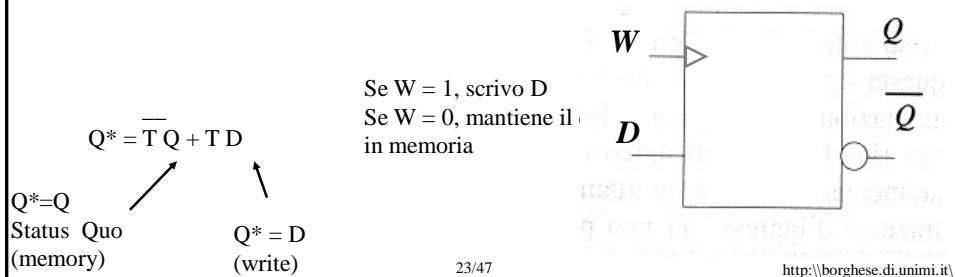
<http://borghese.di.unimi.it/>



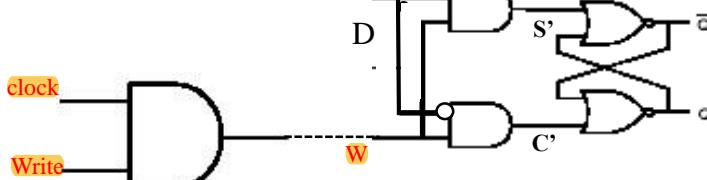
Elemento di memoria



T = segnale di scrittura -> **segnale Write** – attivo alto



Elemento di memoria



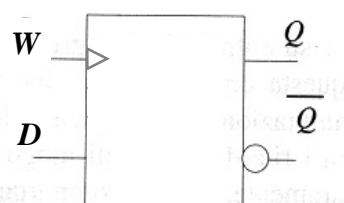
Write può essere sincronizzato dal clock

$$Q^* = \overline{W} Q + W D$$

$Q^* = Q$ $Q^* = D$

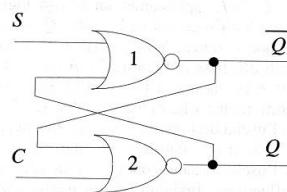
Status Quo (memory) (write)

Clk if (w=1) $Q^* = D$

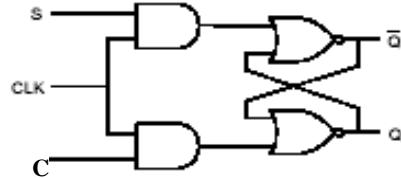




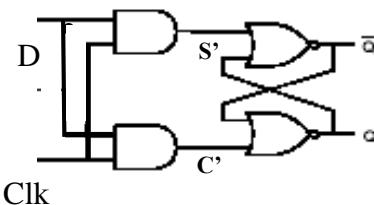
I latch



Operazioni di Set/Reset



Operazioni di Set/Reset sincronizzate



Elemento di memoria



Sommario



Latch sincroni SR

Latch sincroni D

Flip-flop



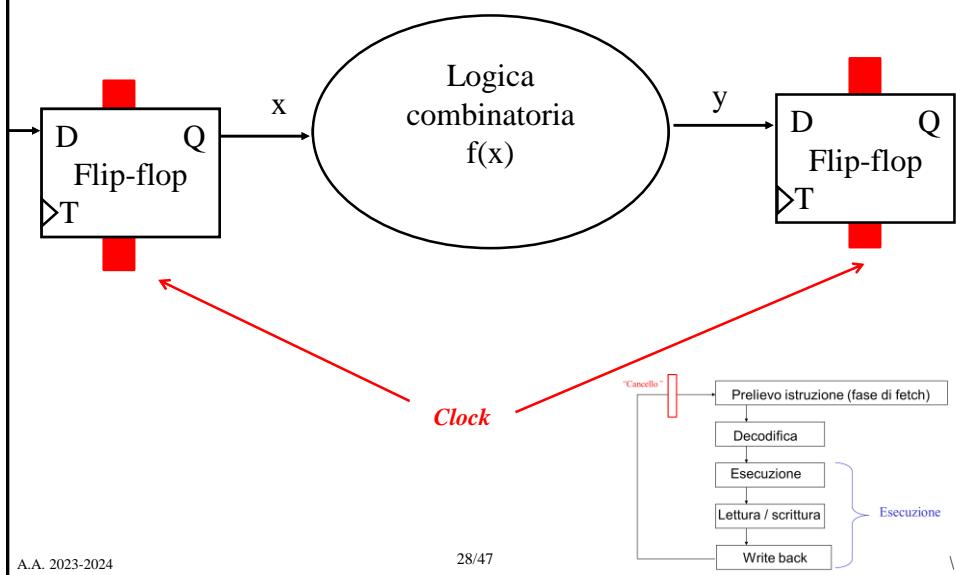
I bistabili



- Elementi di memoria (latch)
 - Sincroni
 - A-sincroni
- “Cancelli” (flip-flop)

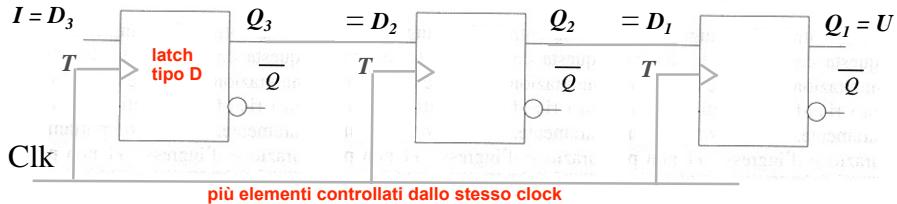


Struttura di un circuito sequenziale



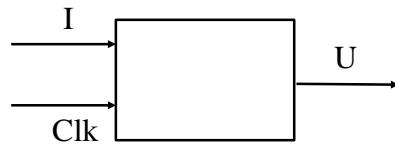


Shift register

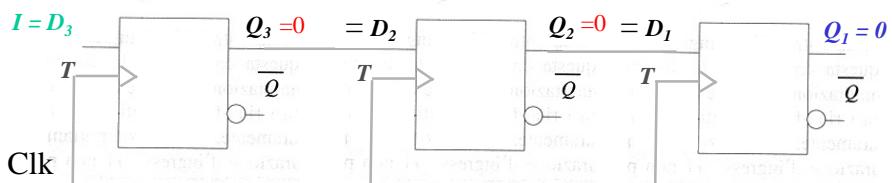


Registro a scorrimento (shift register o barrel shifter).

- Un unico ingresso I e un'unica uscita U.
- In presenza di un segnale attivo (clock alto), il contenuto viene spostato verso dx **di una posizione** (e.g. operazione di shift).
- Il valore contenuto nell'elemento più a dx dove va?
- Qual'è il problema con l'utilizzo dei latch sincroni?



Shift register con i latch (i problemi)



Fotografiamo la situazione iniziale:

- Clock basso
- $Q_3 = Q_2 = Q_1 = 0$
- $D_3 = 0$

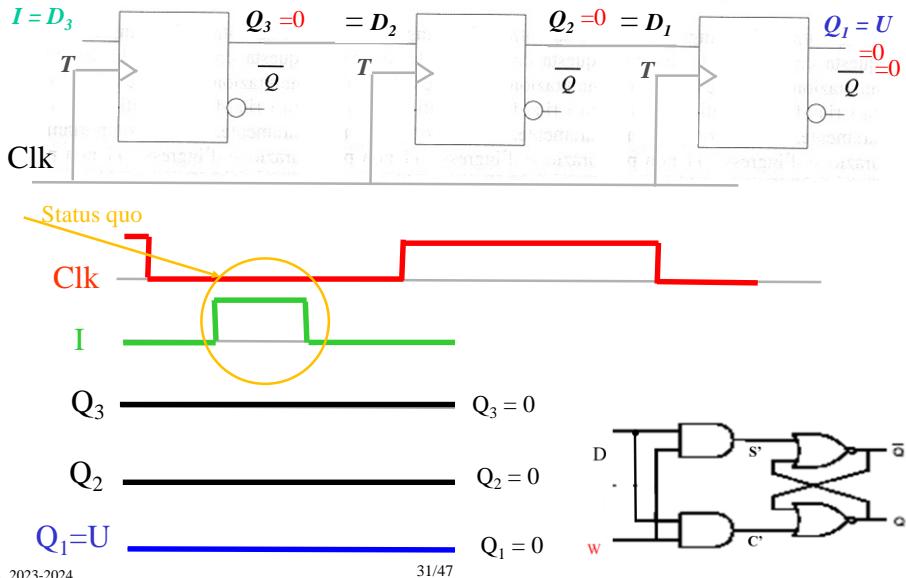
Shift di 1 posizione:

$$D_2 = Q_3$$

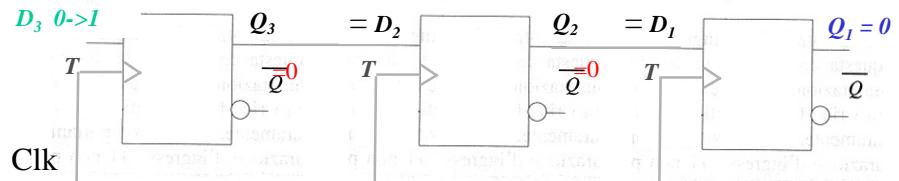
$$D_1 = Q_2$$



Shift register con i latch (status quo)



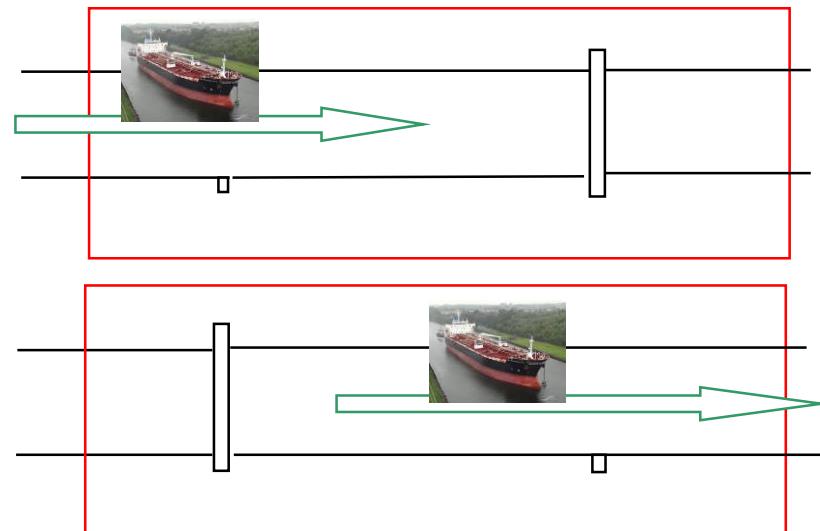
Shift register con i latch (il problema)



L'ingresso $I = D_3$ va a 1 – vorrei ottenere $\{0\ 0\ 0\} \rightarrow \{1\ 0\ 0\}$. Invece ottengo: $\{1\ 1\ 1\}$



Dispositivo di sincronizzazione



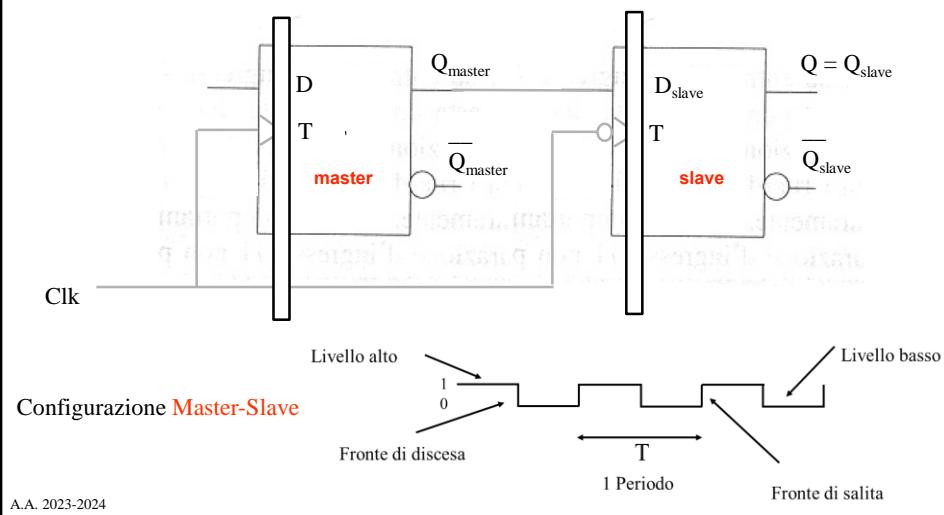
Sistema di “chiuse”

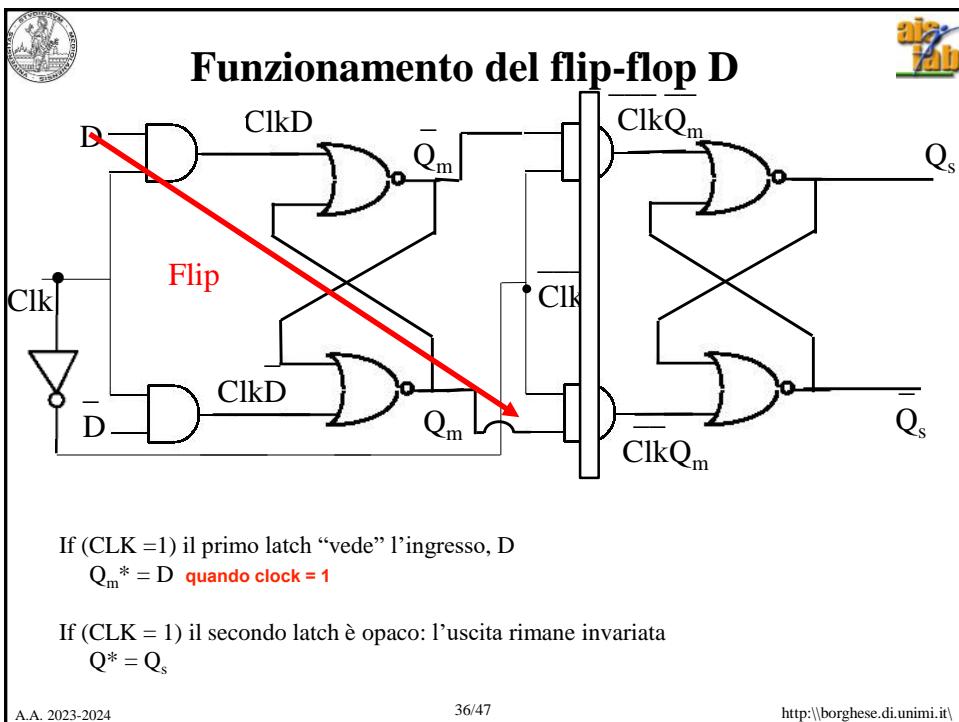
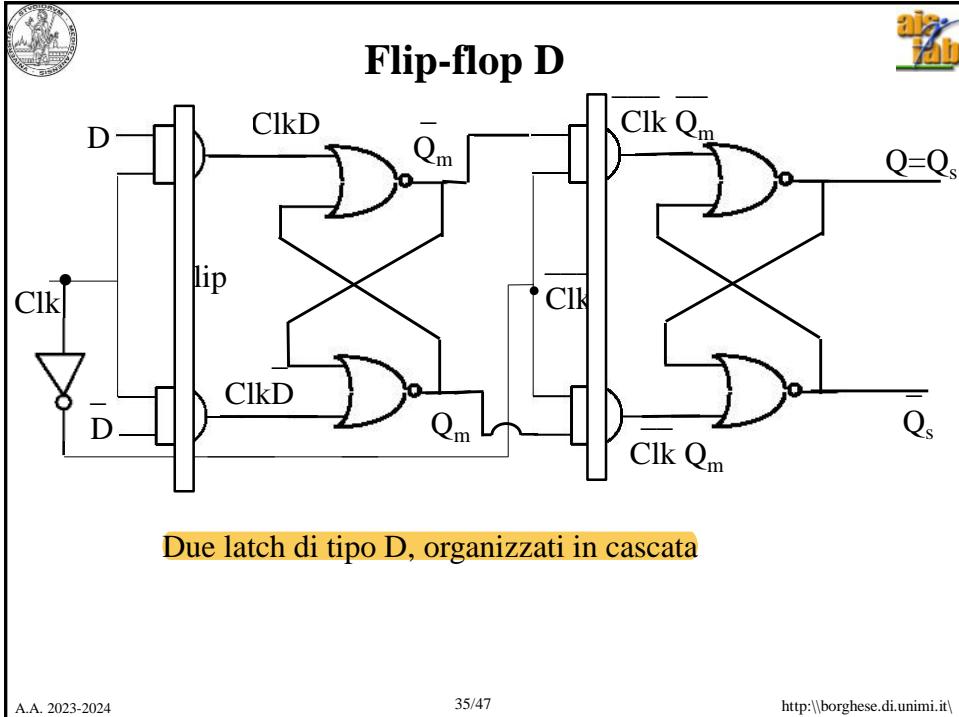


Flip-flop



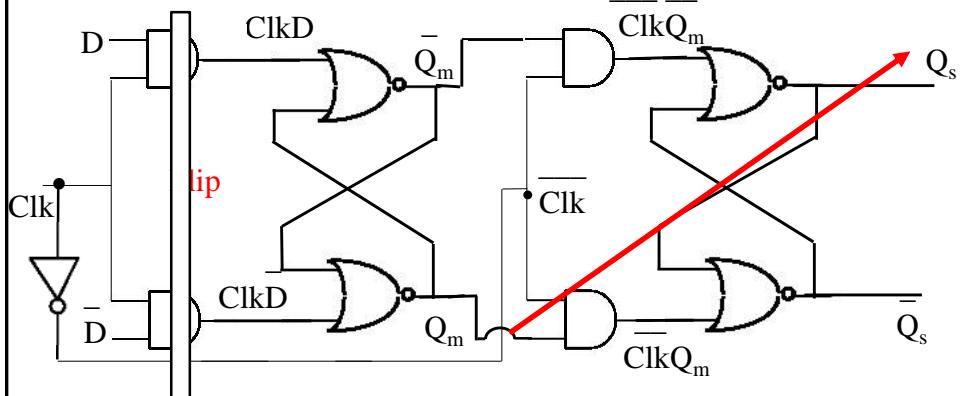
Dispositivi attivi sul fronte (di salita o discesa) del clock (edge sensitive): il loro stato (uscita) può commutare solo in corrispondenza della transizione alto->basso o basso->alto del clock.







Funzionamento del flip-flop D



If (CLK = 0) il primo latch è opaco: l'uscita rimane invariata

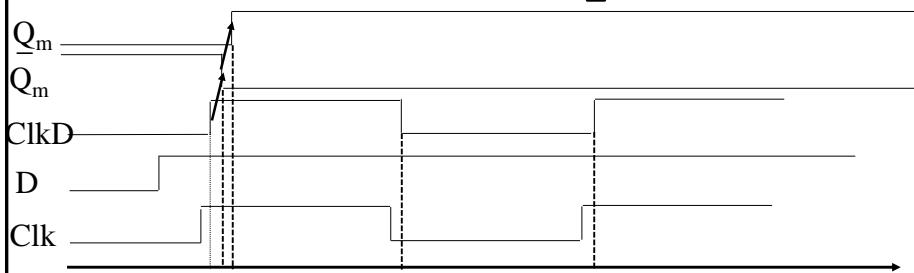
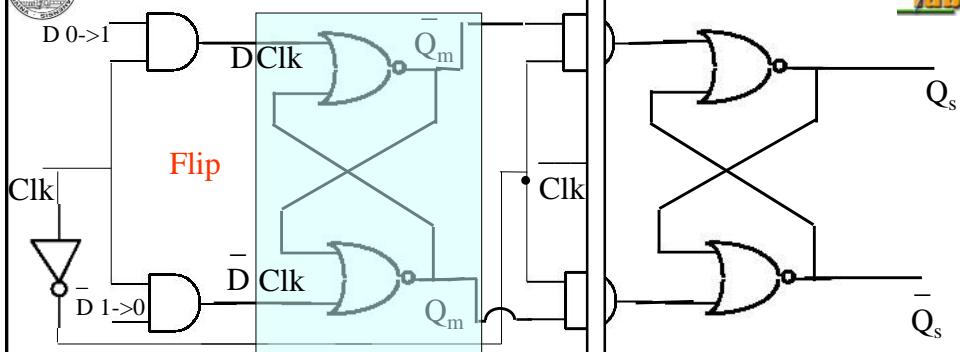
$Q_m^* = Q_m$ **master mantiene il proprio valore**

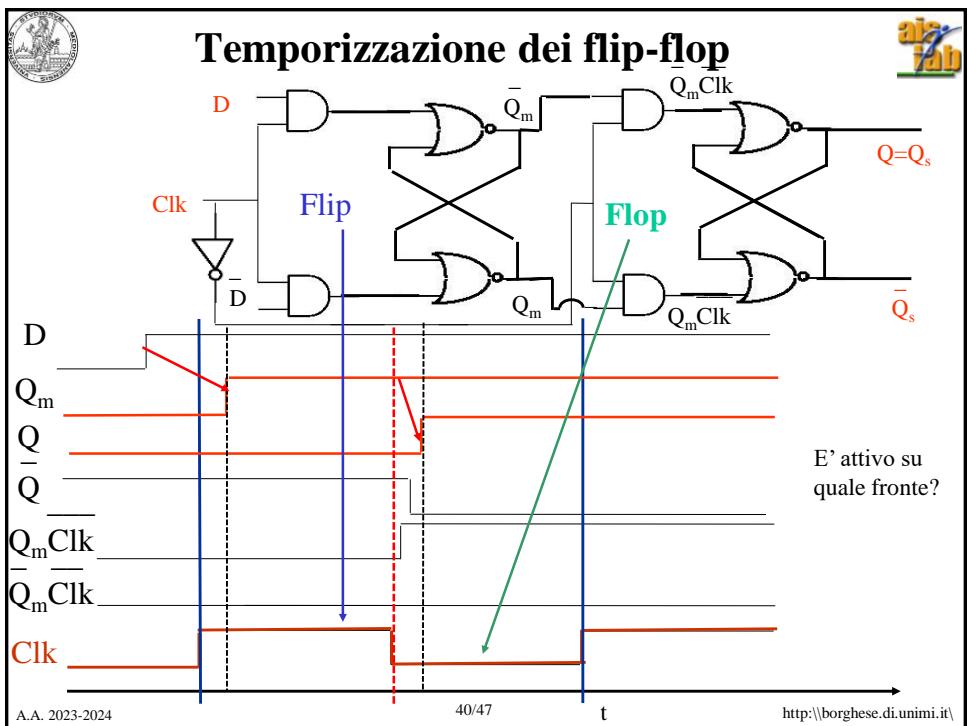
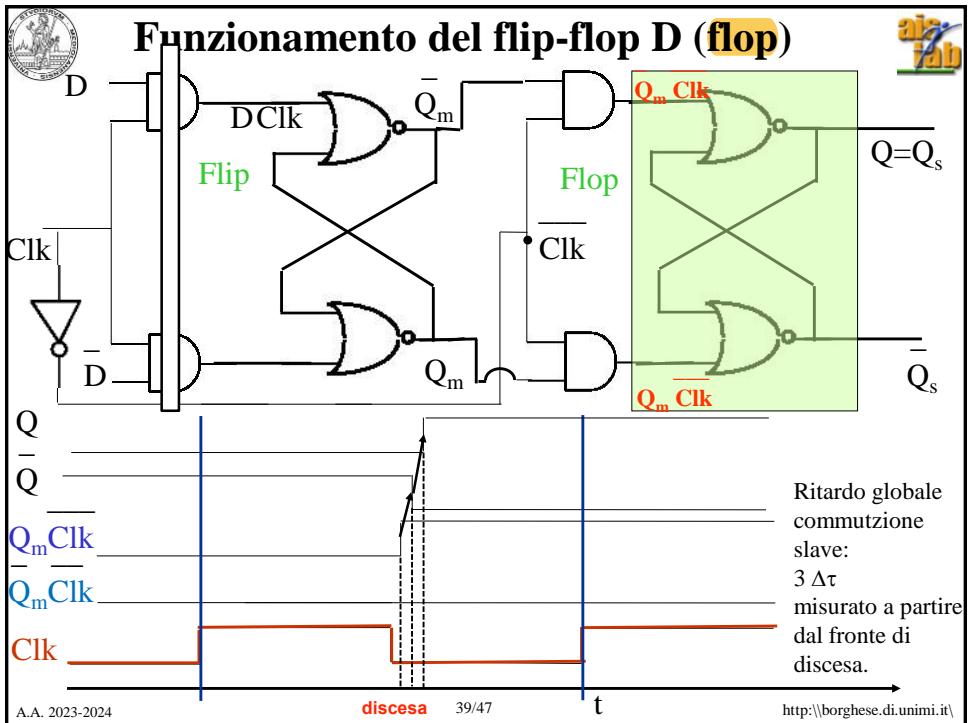
If (CLK = 0) il secondo latch porta l'uscita del master, Q_m , in uscita al dispositivo.

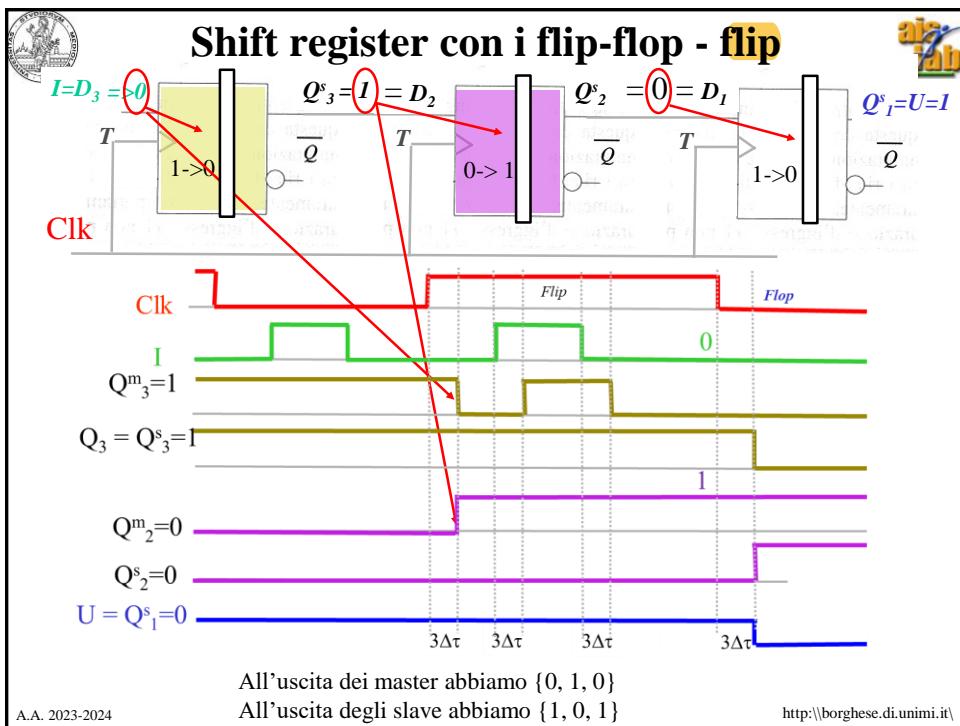
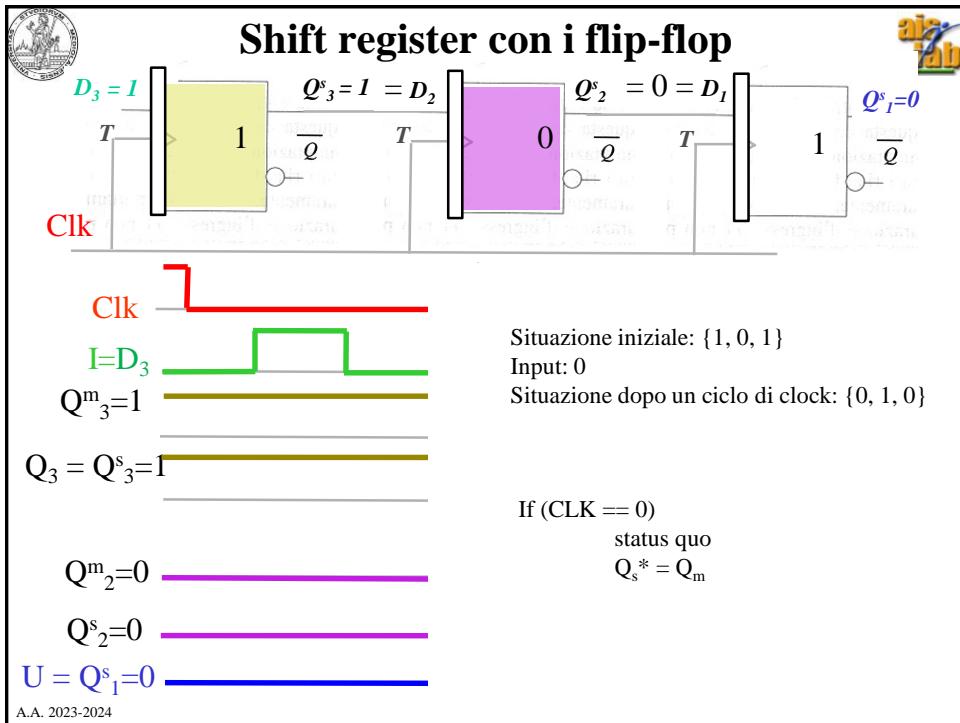
$Q = Q_s^* = Q_m$

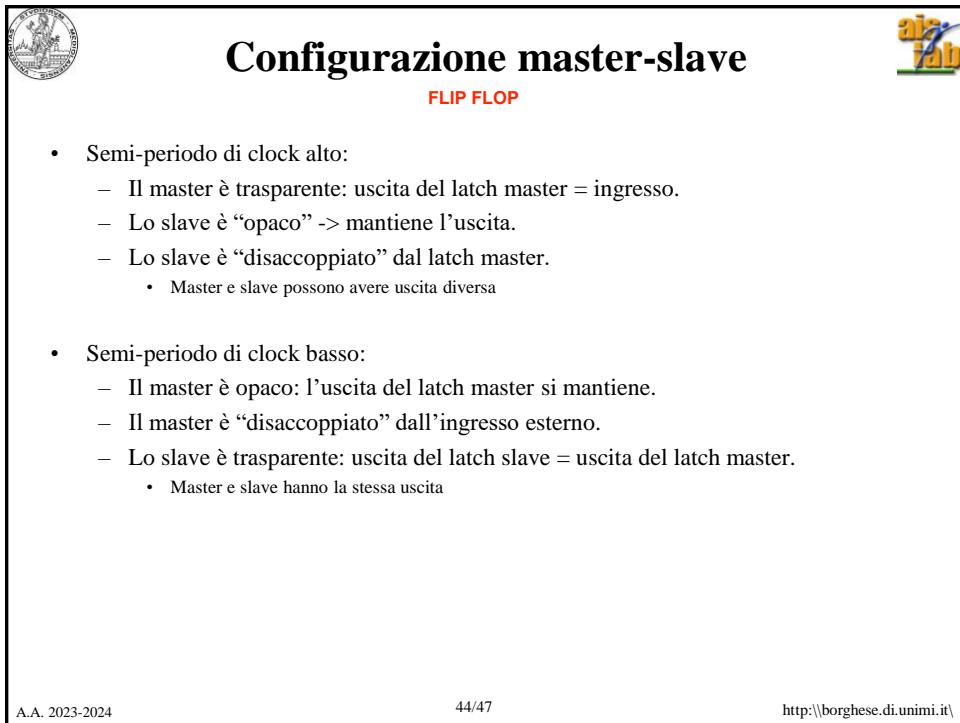
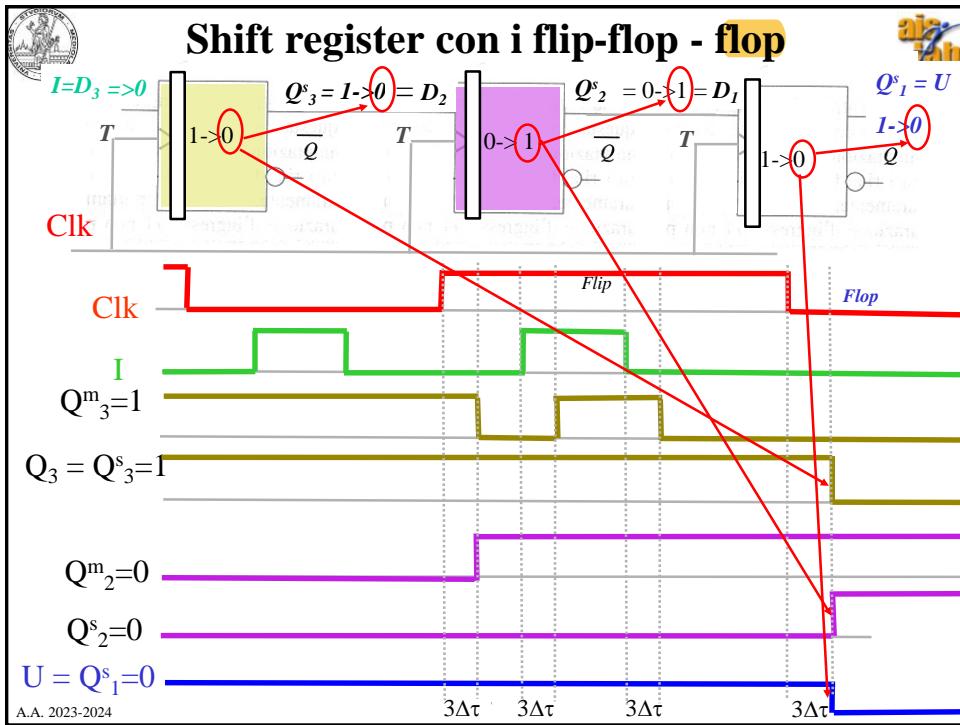


Funzionamento del flip-flop D (flip)



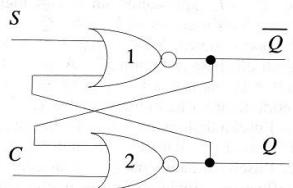




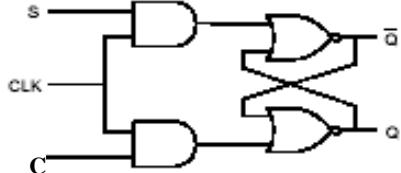




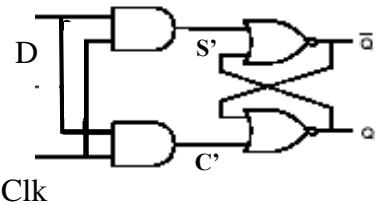
I bistabili



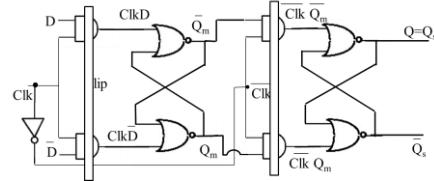
Latch SC asincrono (Set/Reset)



Latch SC sincrono (Set/Reset sincronizzato)



Latch D sincrono (Elemento di memoria)



Flip-flop di tipo D («cancello»)

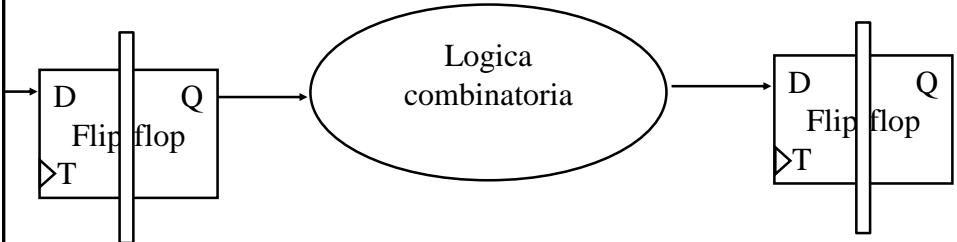
A.A. 2023-2024

45/47

<http://borghese.di.unimi.it/>



Struttura di un circuito sequenziale



Pone dei problemi di sincronizzazione: la logica combinatoria deve terminare la commutazione in tempo utile.

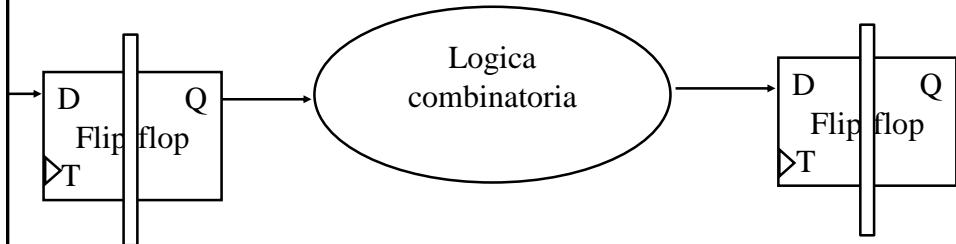
A.A. 2023-2024

46/47

<http://borghese.di.unimi.it/>



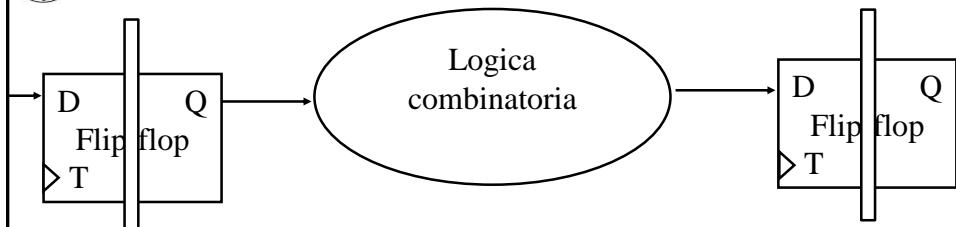
Temporizzazione di un circuito sequenziale



- La logica ha tempo sufficiente per completare la commutazione.
- Il periodo di clock è tale, per cui la commutazione del clock avviene dopo che la logica combinatoria ha terminato tutte le commutazioni.
- Il tempo necessario alla logica combinatoria per commutare è \leq tempo associato al cammino critico.
- Il clock arriva contemporaneamente a tutti i dispositivi sincronizzati.

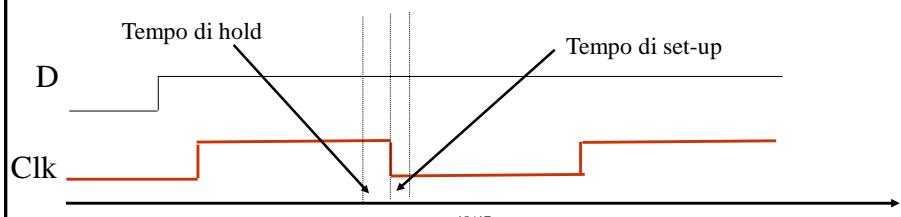


Temporizzazione: problemi



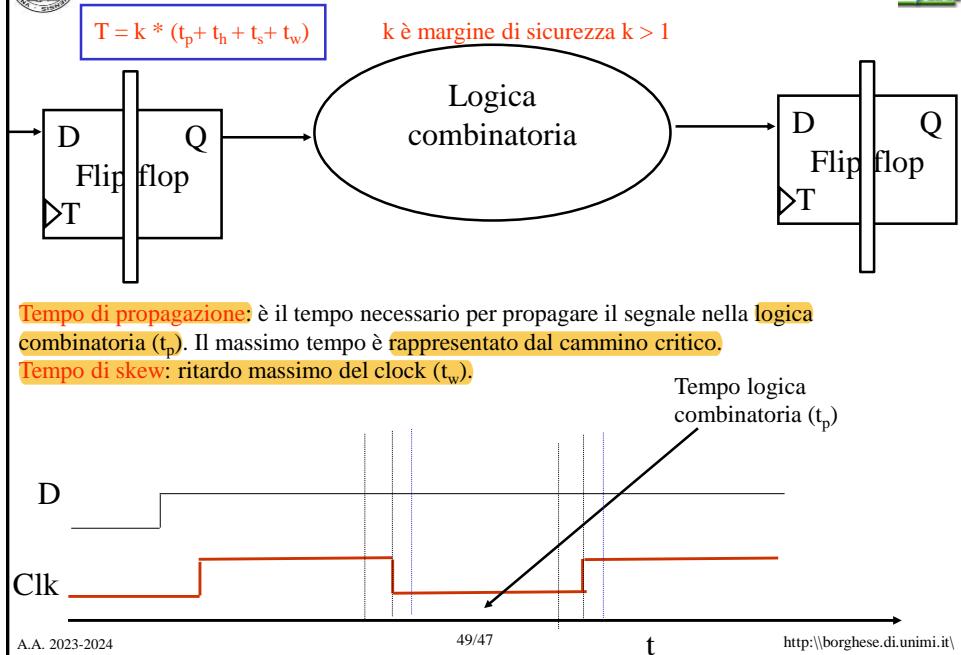
L'input D deve essere stabile intorno alla commutazione del clock:

- **Tempo di hold (t_h)**: è il tempo minimo per cui deve rimanere stabile l'input D prima del fronte di clock (tempo di attraversamento delle porte del master).
- **Tempo di set-up (t_u)**: è il tempo minimo per cui deve rimanere stabile l'input D dopo il fronte di clock (tempo di attraversamento delle porte dello slave).





Temporizzazione: Come si dimensiona il clock



Sommario



Latch sincroni SR

Latch sincroni D

Flip-flop



Macchine a Stati finiti

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson: Sezione B.10
(per approfondimenti, D.3 e D.4)



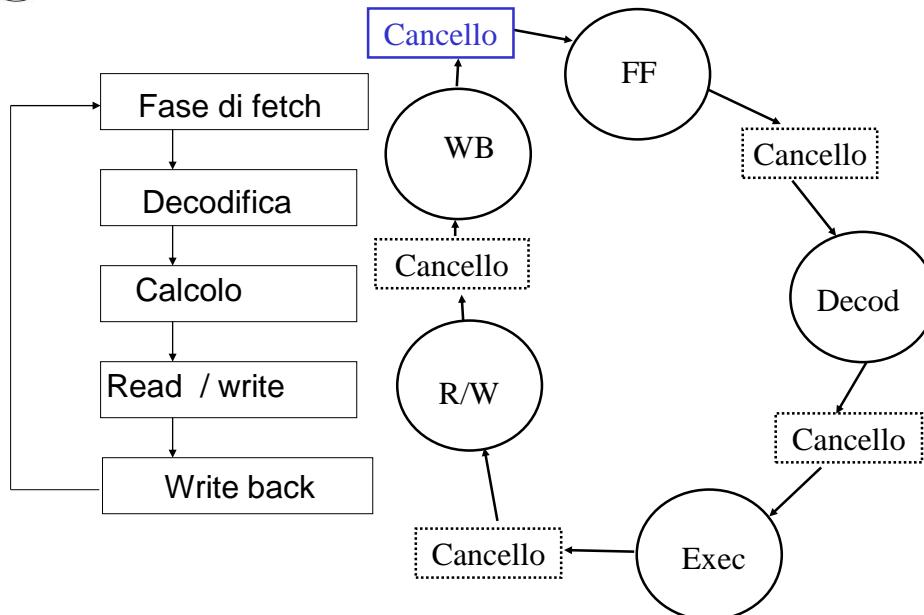
Sommario

Macchine a stati finiti

Esempio: sintesi di un controllore per venditore di bibite.



La CPU come macchina sequenziale



Controllore di una macchina venditrice di bibite



Voglio costruire un controllore di una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta in **input**: {0c, 10c, 20c} e non dà resto.

Questo controllore produrrà in **output** 2 informazioni:

- N = No Caffè
- C = Caffè.

Noi vediamo solamente i dati di input e di output.

NB L'uscita non dipende dalla moneta inserita al tempo t, ma dipende dal totale!
Il totale richiede di tenere conto anche delle monete inserite in precedenza.

Il totale è la **situazione** della pancia della macchina -> **stato della macchina**

Lo stato è interno alla macchina.

Come?



Una macchina venditrice di bibite - I



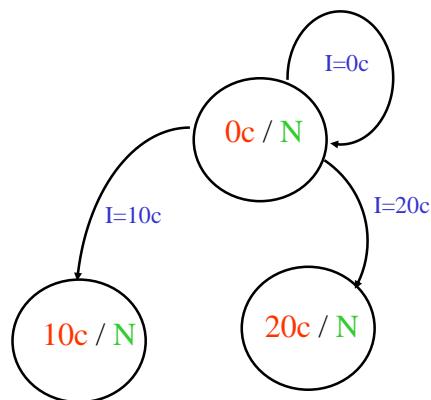
Voglio costruire un controllore di una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta: {0c, 10c, 20c} e non dà resto.

N = No Caffè

C = Caffè.

Approccio costruttivo.



Una macchina venditrice di bibite - II



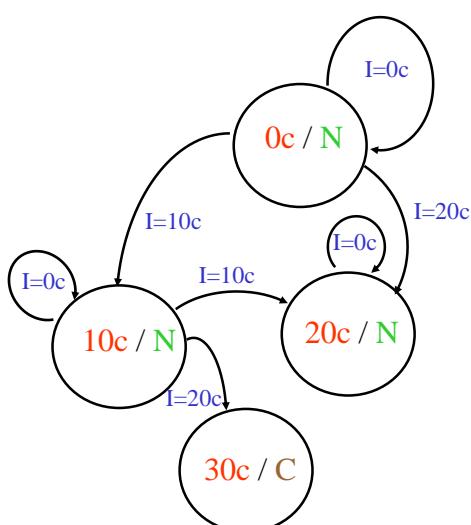
Voglio costruire un controllore di una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta: {0c, 10c, 20c} e non dà resto.

N = No Caffè

C = Caffè.

Approccio costruttivo.





Una macchina venditrice di bibite - III

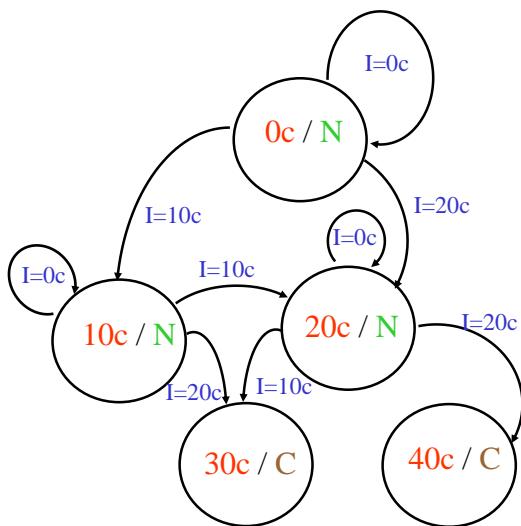


Voglio costruire una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta: {0c, 10c, 20c} e non dà resto.

N = No Caffè
C = Caffè.

Approccio costruttivo.



Una macchina venditrice di bibite - IV

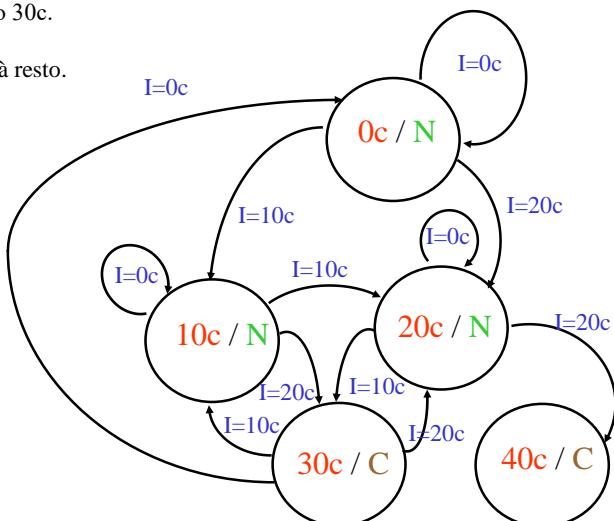


Voglio costruire una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta: {0c, 10c, 20c} e non dà resto.

N = No Caffè
C = Caffè.

Approccio costruttivo.





Una macchina venditrice di bibite - V



Voglio costruire una macchinetta che eroga caffè quando l'utente ha inserito 30c.

Accetta: {0c, 10c, 20c} e non dà resto.

N = No Caffè
C = Caffè.

Approccio costruttivo.

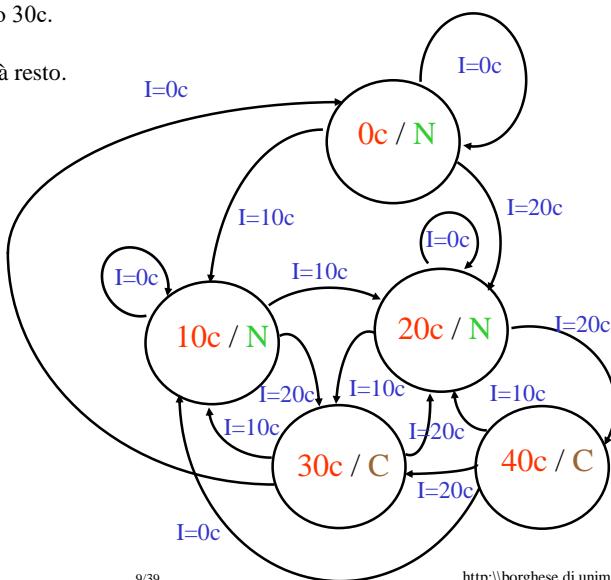
Si suppone che la cifra in eccesso rimanga nella pancia della macchina.

A.A. 2023-2024

9/39

<http://borghese.di.unimi.it/>

CI SONO
ERRORI -
STATO 40
CENT SI
COMPORTA
COME
STATO 30
CENT



State transition graph



Un grafo è un insieme di elementi detti **nodi** che possono essere collegati fra loro da linee chiamate **archi**.

Il totale è la situazione della pancia della macchina -> **stato della macchina**, i **nodi** del grafo.

La moneta inserita correntemente nella macchina rappresenta **l'input alla macchina**, un **arco** del grafo.

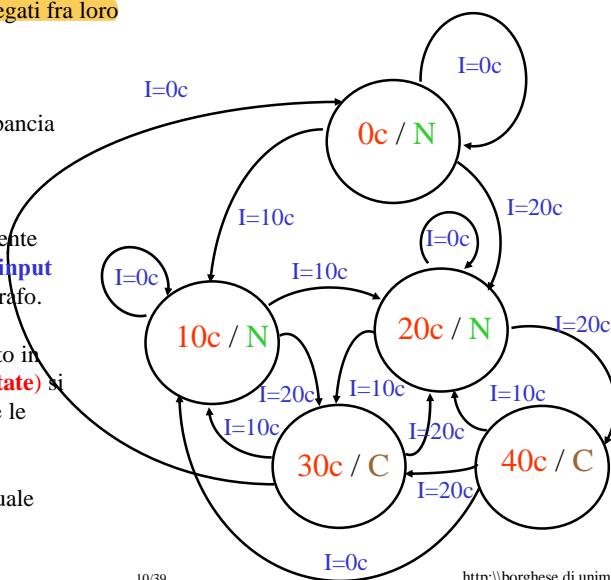
Per ogni stato, viene esaminato in quale **stato prossimo (next state)** si troverà la macchina. Descrivono le **transizioni di stato**.

Per ogni **stato** si determina quale sarà l'**uscita** della macchina.

A.A. 2023-2024

10/39

<http://borghese.di.unimi.it/>





Macchina a Stati Finiti (di Moore)



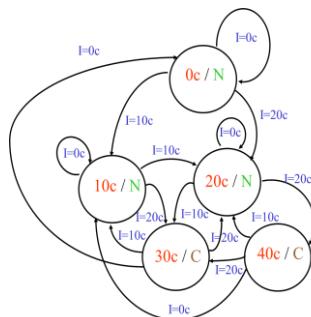
La Macchina di (Edward) Moore è definita, in teoria degli automi, dalla sestupla:

$$\langle X, I, Y, f(\cdot), g(\cdot), X_{\text{ini}} \rangle$$

X: insieme degli stati (in numero finito). **0, 10, 20, 30, 40**

I: insieme di ingresso: tutti i simboli che si possono presentare in ingresso. **0, 10, 20**

Y: insieme di uscita: tutti i simboli che si possono produrre in uscita. **caffè / no caffè**



funzione stato presente e ingresso

f(·): funzione stato prossimo: $X^* = f(X, I)$. Definisce l'evoluzione della macchina nel tempo. L'evoluzione è deterministica.

g(·): funzione di uscita: $Y = g(X)$, è **funzione solamente dello stato attuale** nelle macchine di Moore. Non è funzione dello stato prossimo, X^* .

Stato iniziale: X_{ini} . Per il buon funzionamento della macchina è previsto uno stato iniziale, al quale la macchina può essere portata mediante un comando di reset.



La nostra macchina di Moore



La Macchina di Moore è definita, in teoria degli automi, dalla sestupla:

$$\langle X, I, Y, f(\cdot), g(\cdot), X_{\text{ini}} \rangle$$

X: insieme degli stati (in numero finito): $\{0c, 10c, 20c, 30c, 40c\}$

I: insieme di ingresso: tutti i simboli che si possono presentare in ingresso: $\{0c, 10c, 20c\}$

Y: insieme di uscita: tutti i simboli che si possono generare in uscita: $\{\text{Niente}, \text{Caffè}\}$.

f(·): funzione stato prossimo: $X^* = f(X, I)$. Definisce l'evoluzione della macchina nel tempo. L'evoluzione è deterministica.

g(·): funzione di uscita: $Y = g(X)$, è **funzione solamente dello stato attuale** nelle macchine di Moore. Non è funzione dello stato prossimo, X^* .

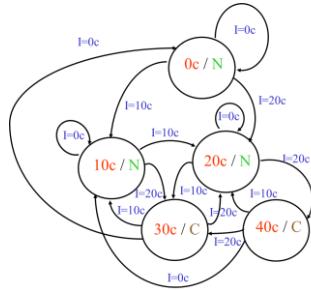
Stato iniziale: X_{ini} . Per il buon funzionamento della macchina è previsto uno stato iniziale, al quale la macchina può essere portata mediante un comando di reset.



Descrizione di una macchina di Moore



STG: State Transition Graph (Diagramma degli stati o Grafo delle transizioni). Ad ogni nodo è associato uno stato. Un arco orientato da uno stato x_i ad uno stato x_j , contrassegnato da un simbolo (di ingresso) α , rappresenta una transizione (passaggio di stato) che si verifica quando la macchina, essendo nello stato x_i , riceve come ingresso il simbolo α .



STT: State Transition Table (Tabella degli Stati). Per ogni coppia, (Stato presente – Ingresso), si definisce l’Uscita e lo Stato Prossimo. La forma è tabellare e ricorda le tabelle della verità da cui è derivata.



Sommario



Macchine a stati finiti

Esempio: sintesi di un controllore per venditore di bibite.



STG di una macchina venditrice di bibite (Semplificata)



Voglio costruire una macchinetta che **eroga caffè** quando l'utente ha inserito **30c**.
Accetta solamente monete da **10c**.

$$I = \{0c, 10c\}$$

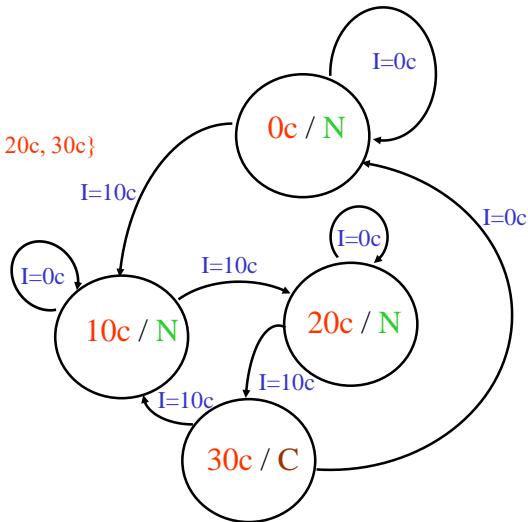
$$Y = \{\text{Nulla, Caffè}\}$$

$$X = \text{"Monete accumulate"} = \{0, 10c, 20c, 30c\}$$

$$X^* = f(X, I)$$

$$Y = g(X)$$

$$X_o = 0c$$



La Macchina di Moore è definita,
in teoria degli automi, dalla
sestupla: $< X, I, Y, f(.), g(.), X_{ini} >$

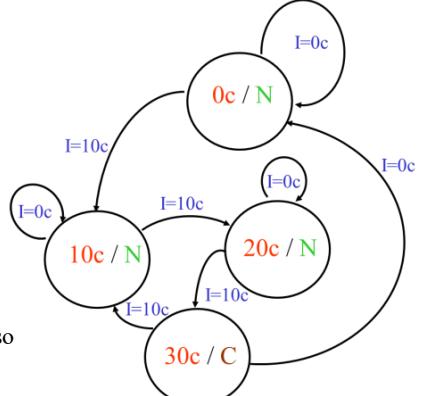


STT della vendor machine - I



STATI <i>X</i>	I INPUT	
	0c	10c
0c	0c	10c
10c	10c	20c
20c	20c	30c
30c	0c	10c

$$X^*$$



Il controllore controlla a ogni iterazione l'ingresso
e a ogni iterazione aggiorna lo **stato prossimo**,
 $X^* = f(X, I)$.

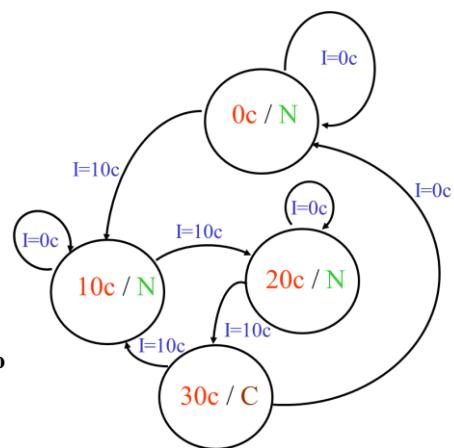
FUNZIONE F



STT della vendor machine - II



X	Y
0c	Nulla
10c	Nulla
20c	Nulla
30c	Caffè



Il controllore controlla a ogni iterazione lo **stato attuale** aggiorna l'**uscita**, $Y = g(X)$

FUNZIONE G

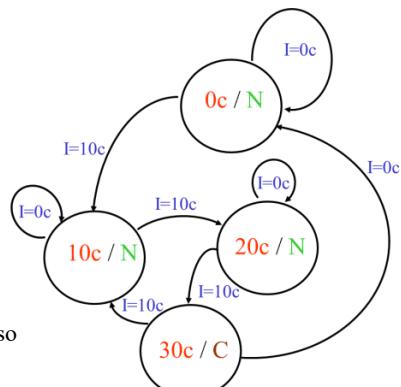


STT della vendor machine - III



I X	0c	10c	Y
0c	0c	10c	Nulla
10c	10c	20c	Nulla
20c	20c	30c	Nulla
30c	0c	10c	Caffè

X*



Il controllore controlla a ogni iterazione l'**ingresso** e a ogni iterazione aggiorna lo **stato prossimo**,
 $X^* = f(X, I)$ e l'**uscita**, $Y = g(X)$



Dal linguistico al numerico



X	I	I	Y
	0c	10c	
0c	0c	10c	Nulla
10c	10c	20c	Nulla
20c	20c	30c	Nulla
30c	0c	10c	Caffè

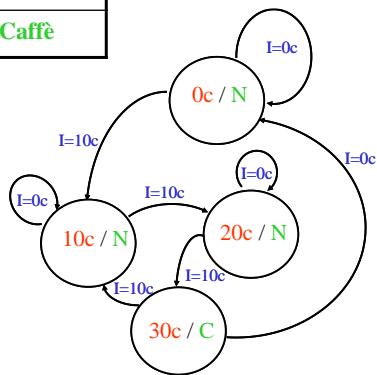
$I = \{0c, 10c\}$
 $Y = \{\text{Nulla}, \text{Caffè}\}$
 $X = \text{"Monete accumulate"} = \{0, 10c, 20c, 30c\}$

Occorre codificare ingresso, stato ed uscita.

=

Assegnare un **codice numerico** ad ogni possibile stato, ingresso, uscita.

Abbiamo definito: $\langle X, I, Y, f(\cdot), g(\cdot), X_{\text{ini}} \rangle$



Codifica della STT della vendor machine



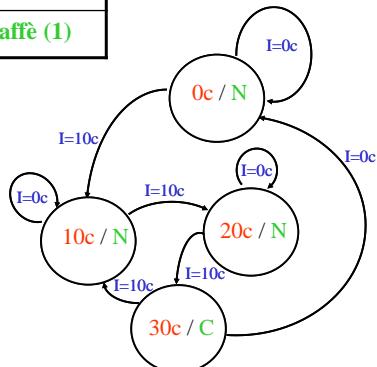
X	I	I	Y
	0c (0)	10c (1)	
0c (0)	0c (0)	10c (1)	Nulla (0)
10c (1)	10c (1)	20c (2)	Nulla (0)
20c (2)	20c (2)	30c (3)	Nulla (0)
30c (3)	0c (0)	10c (1)	Caffè (1)

$$\begin{aligned}
 I &= \{0c, 10c\} = \{0, 1\} \\
 Y &= \{\text{Nulla}, \text{Caffè}\} = \{0, 1\} \\
 X &= \{0, 10c, 20c, 30c\} = \{0, 1, 2, 3\}
 \end{aligned}$$

$$\begin{aligned}
 X^* &= f(X, I) \\
 Y &= g(X)
 \end{aligned}$$

da sintetizzare
da sintetizzare

Abbiamo definito: $\langle X, I, Y, f(\cdot), g(\cdot), X_{\text{ini}} \rangle$





Significato della codifica



X	I	I	Y
	0c (0)	10c (1)	
0c (0)	0c (0)	10c (1)	Nulla (0)
10c (1)	10c (1)	20c (2)	Nulla (0)
20c (2)	20c (2)	30c (3)	Nulla (0)
30c (3)	0c (0)	10c (1)	Caffè (1)

$$I = \{0c, 10c\} = \{0, 1\}$$

$$Y = \{\text{Nulla}, \text{Caffè}\} = \{0, 1\}$$

$$X = \{0, 10c, 20c, 30c\} = \{0, 1, 2, 3\}$$

if (stato == 3) the Y = 1

if (stato = 0 and I = 1) then stato_prossimo = 1

if (stato = 0 and I = 0) then stato_prossimo = 0

if (stato = 1 and I = 1) then stato_prossimo = 2

if (stato = 1 and I = 0) then stato_prossimo = 1

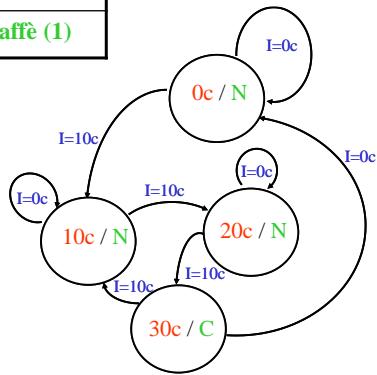
if (stato = 2 and I = 1) then stato_prossimo = 3

if (stato = 2 and I = 0) then stato_prossimo = 2

if (stato = 3 and I = 1) then stato_Prossimo = 1

if (stato = 3 and I = 0) then stato_Prossimo = 0

21/39



<http://borghese.di.unimi.it/>



Codifica binaria della STT



X	I	I	Y
	0c (00)	10c (01)	
0c (00)	0c (00)	10c (01)	Nulla (0)
10c (01)	10c (01)	20c (10)	Nulla (0)
20c (10)	20c (10)	30c (11)	Nulla (0)
30c (11)	0c (00)	10c (01)	Caffè (1)

$$I = \{0c, 10c\} =$$

$$\{0, 1\}$$

$$Y = \{\text{Nulla}, \text{Caffè}\} =$$

$$\{0, 1\}$$

$$X = \{0, 10c, 20c, 30c\} =$$

$$\{00, 01, 10, 11\}$$

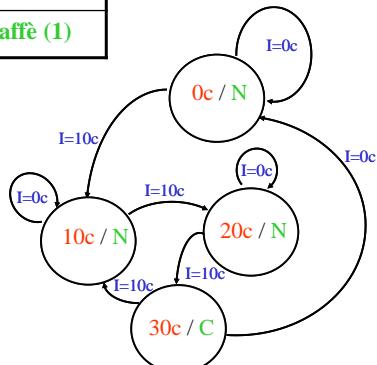
$$X^* = f(X, I)$$

da sintetizzare

$$Y = g(X)$$

da sintetizzare

Abbiamo definito: $\langle X, I, Y, f(\cdot), g(\cdot), X_{\text{ini}} \rangle$





Significato della Codifica binaria della STT -

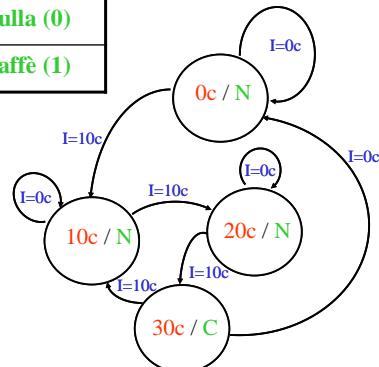
X	I		Y
	0c (0)	10c (1)	
0c (00)	0c (00)	10c (01)	Nulla (0)
10c (01)	10c (01)	20c (10)	Nulla (0)
20c (10)	20c (10)	30c (11)	Nulla (0)
30c (11)	0c (00)	10c (01)	Caffè (1)

$$I = \{0c, 10c\} = \{0, 1\}$$

$$Y = \{\text{Nulla, Caffè}\} = \{0, 1\}$$

$$X = \{0, 10c, 20c, 30c\} = \{00, 01, 10, 11\}$$

X*



if ($X_1 == 1$ and $X_0 == 1$) the $Y = 1$

A.A. 2023-2024

23/39

<http://borghese.di.unimi.it/>



Significato della Codifica della STT - X_0^*

X	I		Y
	0c (0)	10c (1)	
0c (00)	0c (00)	10c (01)	Nulla (0)
10c (01)	10c (01)	20c (10)	Nulla (0)
20c (10)	20c (10)	30c (11)	Nulla (0)
30c (11)	0c (00)	10c (01)	Caffè (1)

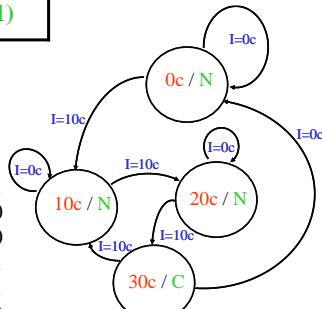
$$I = \{0c, 10c\} = \{0, 1\}$$

$$Y = \{\text{Nulla, Caffè}\} = \{0, 1\}$$

$$X = \{0, 10c, 20c, 30c\} = \{00, 01, 10, 11\}$$

X*

if ($(X_1 == 0$ and $X_0 == 0)$ and $I = 1$) then LSB_stato_prossimo = $X_0^* = 1$
 if ($(X_1 == 0$ and $X_0 == 0)$ and $I = 0$) then LSB_stato_prossimo = $X_0^* = 0$
 if ($(X_1 == 0$ and $X_0 == 1)$ and $I = 1$) then LSB_stato_prossimo = $X_0^* = 0$
 if ($(X_1 == 0$ and $X_0 == 1)$ and $I = 0$) then LSB_stato_prossimo = $X_0^* = 1$
 if ($(X_1 == 1$ and $X_0 == 0)$ and $I = 1$) then LSB_stato_prossimo = $X_0^* = 1$
 if ($(X_1 == 1$ and $X_0 == 0)$ and $I = 0$) then LSB_stato_prossimo = $X_0^* = 0$
 if ($(X_1 == 1$ and $X_0 == 1)$ and $I = 1$) then LSB_stato_Prossimo = $X_0^* = 1$
 if ($(X_1 == 1$ and $X_0 == 1)$ and $I = 0$) then LSB_stato_Prossimo = $X_0^* = 0$



<http://borghese.di.unimi.it/>



Significato della Codifica della STT - S_1^*



X	I		Y
	0c (0)	10c (1)	
0c (00)	0c (00)	10c (01)	Nulla (0)
10c (01)	10c (01)	20c (10)	Nulla (0)
20c (10)	20c (10)	30c (11)	Nulla (0)
30c (11)	0c (00)	10c (01)	Caffè (1)

I = {0c, 10c} = {0, 1}

Y = {Nulla, Caffè} = {0, 1}

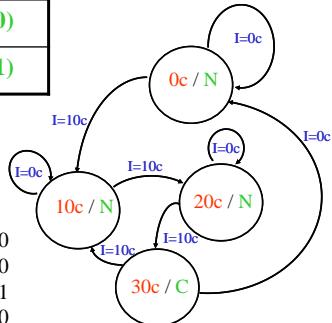
X = {0, 10c, 20c, 30c} = {00, 01, 10, 11}

X*

```

if ((X1 == 0 and X0 == 0) and I = 1) then MSB_stato_prossimo = X1* = 0
if ((X1 == 0 and X0 == 0) and I = 0) then MSB_stato_prossimo = X1* = 0
if ((X1 == 0 and X0 == 1) and I = 1) then MSB_stato_prossimo = X1* = 1
if ((X1 == 0 and X0 == 1) and I = 0) then MSB_stato_prossimo = X1* = 0
if ((X1 == 1 and X0 == 0) and I = 1) then MSB_stato_prossimo = X1* = 1
if ((X1 == 1 and X0 == 0) and I = 0) then MSB_stato_prossimo = X1* = 1
if ((X1 == 1 and X0 == 1) and I = 1) then MSB_stato_Prossimo = X1* = 0
if ((X1 == 1 and X0 == 1) and I = 0) then MSB_stato_Prossimo = X1* = 0

```



<http://borgheze.di.unimi.it/>



Macchina a Stati Finiti (di Moore)



La Macchina di Moore è definita, in teoria degli automi, dalla sestupla:

< X, I, Y, f(.), g(.), X_{ini} >

X: insieme degli stati (in numero finito).

I: insieme di ingresso: tutti i simboli che si possono presentare in ingresso. In caso di codifica binaria, se abbiamo n linee in ingresso (variabili binarie), avremo 2ⁿ possibili simboli da leggere in ingresso (configurazioni).

Y: insieme di uscita: tutti i simboli che si possono generare in uscita. In caso di codifica binaria, se abbiamo m linee in uscita (variabili binarie), avremo 2^m possibili simboli in uscita (configurazioni).

f(.): funzione stato prossimo: $X' = f(X, I)$. Definisce l'evoluzione della macchina nel tempo. L'evoluzione è deterministica. La funzione è una funzione logica.

g(.): funzione di uscita: $Y = g(X)$ nelle macchine di Moore. E' una funzione logica.

Stato iniziale: X_{ini} . Per il buon funzionamento della macchina è previsto uno stato iniziale, al quale la macchina può essere portata mediante un comando di reset.



Codifica binaria della FSM della Vendor Machine



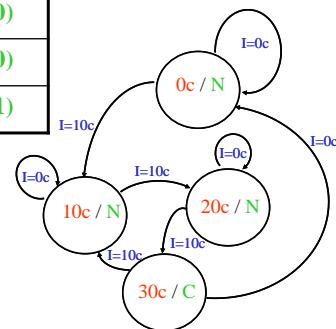
I	I	Y
X - X₁X₀	0c (0)	10c (1)
0c (00)	0c (00)	10c (01)
10c (01)	10c (01)	20c (10)
20c (10)	20c (10)	30c (11)
30c (11)	0c (00)	10c (01)

X è su 2 cifre =>
2 bit X₁ e X₀

$$\begin{aligned} I &= \{0c, 10c\} = & \{0, 1\} \\ Y &= \{\text{Nulla, Caffè}\} = & \{0, 1\} \\ X &= \{0, 10c, 20c, 30c\} = & \{00, 01, 10, 11\} \end{aligned}$$

$$\begin{aligned} X^* &= f(X, I) && \text{da sintetizzare} \\ Y &= g(X) && \text{da sintetizzare} \end{aligned}$$

Abbiamo definito: < X, I, Y, f(.), g(.), X_{ini} >



Sintesi della funzione di uscita della FSM della Vendor Machine

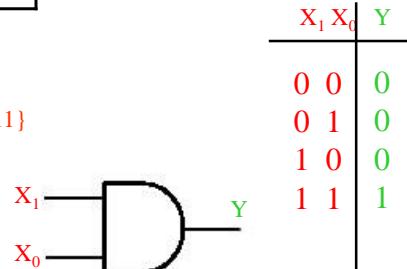


I	Y
X - X₁X₀	
(0c) 00	(Nulla) 0
(10c) 01	(Nulla) 0
(20c) 10	(Nulla) 0
(30c) 11	(Caffè) 1

X è su 2 cifre =>
2 bit X₁ e X₀

$$\begin{aligned} I &= \{0c, 10c\} = & \{0, 1\} \\ Y &= \{\text{Nulla, Caffè}\} = & \{0, 1\} \\ X &= \{0, 10c, 20c, 30c\} = & \{00, 01, 10, 11\} \end{aligned}$$

$$\begin{aligned} X^* &= f(X, I) && \text{da sintetizzare} \\ Y &= g(X) && \text{da sintetizzare} \end{aligned}$$





Sintesi della funzione stato prossimo della FSM della Vendor Machine



I	I	
$X - X_1 X_0$	0c (0)	10c (1)
0c (00)	0c (00)	10c (01)
10c (01)	10c (01)	20c (10)
20c (10)	20c (10)	30c (11)
30c (11)	0c (00)	10c (01)

$$I = \{0, 1\}$$

$$Y = \{0, 1\}$$

$$X = \{00, 01, 10, 11\}$$

$$X^* = f(X, I)$$

$$Y = g(X) = X_1 X_0$$

I 2 bit di stato prossimo vengono sintetizzati separatamente. Sono entrambi funzione dei 2 bit di stato all'istante attuale e del bit di ingresso

X è su 2 cifre => 2 bit: X_1 e X_0

Devo sintetizzare:

$$X_1^{t+1} = f_1(X_0^t, X_1^t, I)$$

$$X_0^{t+1} = f_0(X_0^t, X_1^t, I)$$

$X_1 X_0 I$	$X^*_1 X^*_0$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	1 0
1 0 1	1 1
1 1 0	0 0
1 1 1	0 1

<http://borghese.di.unimi.it/>



Sintesi della funzione stato prossimo della FSM della Vendor Machine



I	I		Y
$X_1 X_0$	0	1	
0 0	00	01	0
0 1	01	10	0
1 0	10	11	0
1 1	00	01	1

$$I = \{0, 1\}$$

$$Y = \{0, 1\}$$

$$X = \{00, 01, 10, 11\}$$

$$X^* = f(X, I) \Rightarrow X^*_0 = \overline{I} \overline{X}_1 \overline{X}_0 + \overline{I} \overline{X}_1 X_0 + I \overline{X}_1 \overline{X}_0 + I X_1 \overline{X}_0 =$$

$$I(X_1 + X_0) + I X_1 X_0$$

$$X^*_1 = \dots$$

I 2 bit di stato prossimo vengono sintetizzati separatamente. Sono entrambi funzione dei 2 bit di stato all'istante attuale e del bit di ingresso

X è su 2 cifre => 2 bit: X_1 e X_0

Devo sintetizzare:

$$X_1^{t+1} = f_1(X_0^t, X_1^t, I)$$

$$X_0^{t+1} = f_0(X_0^t, X_1^t, I)$$

$X_1 X_0 I$	$X^*_1 X^*_0$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	1 0
1 0 1	1 1
1 1 0	0 0
1 1 1	0 1

<http://borghese.di.unimi.it/>



Sintesi della funzione stato prossimo della FSM della Vendor Machine



$X_1 X_0$	I	I
	0	1
0 0	00	01
0 1	01	10
1 0	10	11
1 1	00	01

I = {0, 1}

Y = {0, 1}

X = {00, 01, 10, 11}

$$X^* = f(X, I) \Rightarrow X_0^* = I \bar{X}_1 \bar{X}_0 + \bar{I} \bar{X}_1 X_0 + I \bar{X}_1 X_0 + I X_1 X_0 =$$

$$I(X_1 + X_0) + I X_1 X_0$$

$$X_0^* = X_1 \bar{X}_0 + \bar{X}_1 X_0 I$$

$$X^* = \{X_0^* \ X_1^*\}$$

$X_1 \ X_0 \ I$	$X_0^* \ X_1^*$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	1 0
1 0 1	1 1
1 1 0	0 0
1 1 1	0 1

I 2 bit di stato prossimo vengono sintetizzati separatamente. Sono entrambi funzione dei 2 bit di stato all'istante attuale e del bit di ingresso

<http://borgheze.di.unimi.it/>

X è su 2 cifre => 2 bit: X_1 e X_0

Devo sintetizzare:

$$X_1^{t+1} = f_1(X_0^t, X_1^t, I)$$

$$X_0^{t+1} = f_0(X_0^t, X_1^t, I)$$



Sintesi della funzione stato prossimo della FSM della Vendor Machine



$X_1 X_0$	I	I	Y
	0	1	
0 0	00	01	0
0 1	01	10	0
1 0	10	11	0
1 1	00	01	1

I = {0, 1}

Y = {0, 1}

X = {00, 01, 10, 11}

$$X^* = f(X, I) \Rightarrow X_0^* = I (X_1 + \bar{X}_1 \bar{X}_0) + \bar{I} \bar{X}_1 X_0 = \\ -I(X_1 + X_0) + I X_1 X_0$$

$$X_0^* = X_1 X_0 + X_1 X_0 I$$

$$Y = g(X) = X_1 X_0$$

X è su 2 cifre => 2 bit: X_1 e X_0

Devo sintetizzare:

$$X_1^{t+1} = f_1(X_0^t, X_1^t, I)$$

$$X_0^{t+1} = f_0(X_0^t, X_1^t, I)$$

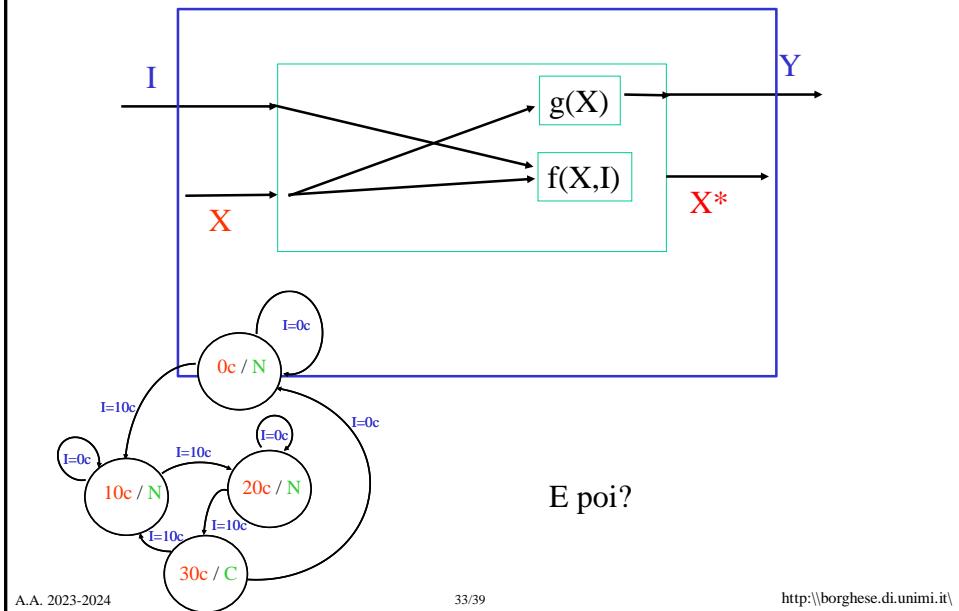
$X_1 \ X_0 \ I$	$X_0^* \ X_1^*$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	1 0
1 0 1	1 1
1 1 0	0 0
1 1 1	0 1

I 2 bit di stato prossimo vengono sintetizzati separatamente. Sono entrambi funzione dei 2 bit di stato all'istante attuale e del bit di ingresso

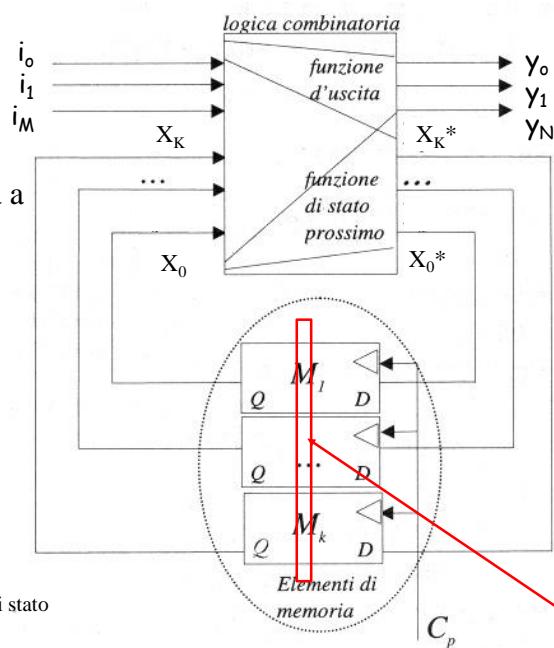
<http://borgheze.di.unimi.it/>



La situazione



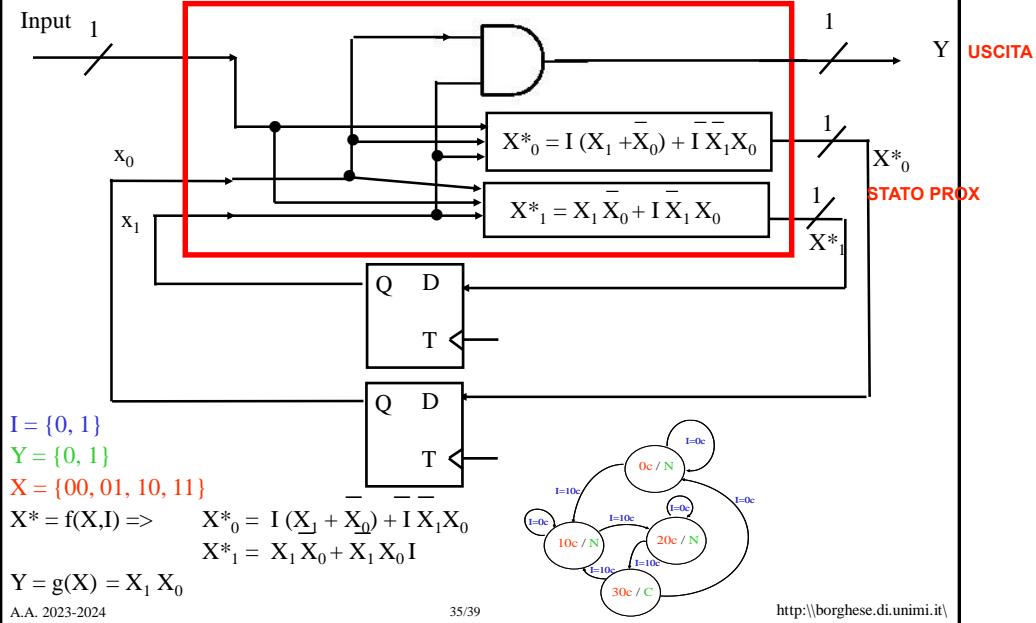
Macchina a
stati finiti
binaria



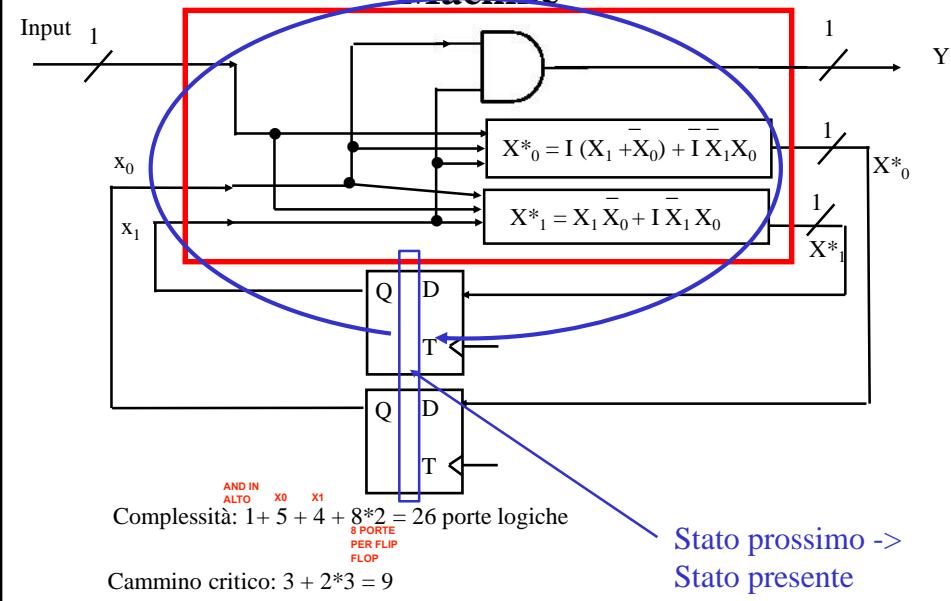
Macchina di Huffman



Sintesi del circuito della FSM della Vendor Machine



Sintesi del circuito della FSM della Vendor Machine

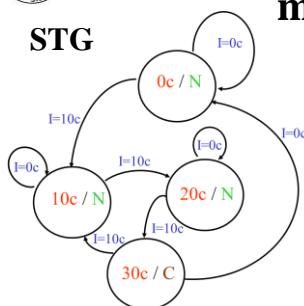




I passi per la progettazione di una macchina di Huffman



STG



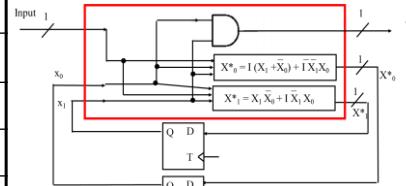
I	Y	
X	0c	10c
0c	0c	10c
10c	10c	20c
20c	20c	30c
30c	0c	10c

STT

STT codificata

I	Y	
X - $X_1 X_0$	0c (0)	10c (1)
0c (00)	0c (00)	10c (01)
10c (01)	10c (01)	20c (10)
20c (10)	20c (10)	30c (11)
30c (11)	0c (00)	10c (01)

Macchina di Huffman



Una vendor machine più completa.



Monete diverse dai 10c.

Scelta di bevande diverse.

Bevande diversi con costi diversi.

Periodo di refrattarietà nella quale non si possono inserire monete (periodo di preparazione del caffè).

.....



Sommario



Macchine a stati finiti.

Esempio: sintesi di un controllore per venditore di bibite.



Firmware Multiplier

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti sul Patterson 5a ed.: B.6 & 3.4



Sommario

Il moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware



L'approccio firmware



Nell'approccio firmware, viene inserita nella ALU una micro-architettura costituita da una unità di controllo, dei componenti di calcolo e dei registri.

L'unità di controllo attiva opportunamente le unità di calcolo e il trasferimento da/verso i registri. Approccio “*controllore-datapath*” in piccolo.

Viene inserito un microcalcolatore dentro la ALU.

Il primo micropogramma era presente nell'IBM 360 (1964).



L'approccio firmware vs hardware



La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti. Inoltre è rigida («hard») e non si può adattare a implementare funzioni diverse.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. È meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure, modificando solamente l'unità di controllo.

La soluzione HW viene utilizzata per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).



Approcci tecnologici alla ALU

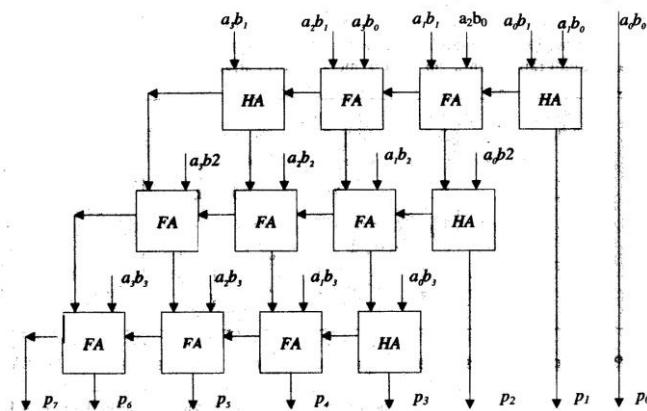


Quattro approcci tecnologici alla costruzione di una ALU (e di una CPU):

- **Approccio porte logiche.** E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabile per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- **Approccio hardware programmabile** (e.g. PLA, FPGA). Ad ogni operazione corrisponde un circuito combinatorio specifico.
- **Approccio firmware** (firm = stabile), o microprogrammato. Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate collegando opportunamente i componenti, a partire dall'algoritmo che le implementa.



Circuito hardware della moltiplicazione



$$\text{Complessità} = (N-2) * [(N-1) * 5 + 1 * 2] + (N-2)*5 + 2 * 2 + N * N$$

$$\text{Complessità per parole su 4 bit} = 2 * (3 * 5 + 2) + 14 + 16 = 64 \text{ porte a due ingressi}$$

Come possiamo renderlo più flessibile? Come arrivare a un circuito che serva anche la divisione intera?



Algoritmo firmware per la moltiplicazione



Il rationale degli algoritmi firmware della moltiplicazione è il seguente.

Si analizzano sequenzialmente i bit del moltiplicatore e si creano i **prodotti parziali**:

- 1) Si mette **0** (su n bit) nella posizione opportuna (se il bit analizzato del moltiplicatore = 0).
 - 2) Si mette una **copia del moltiplicando** (su n bit) nella posizione opportuna (se il bit analizzato del moltiplicatore è = 1).

$$27 \times 5 = 135$$

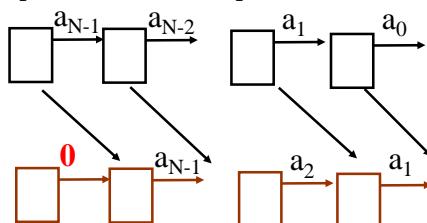


Shift (scalamento)



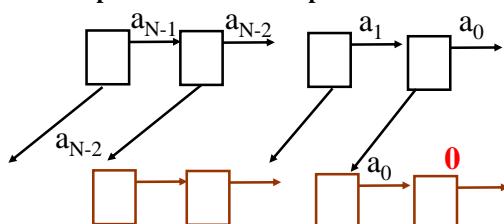
Dato A su 32 bit: $a_i = a_{i-k}$ k shift amount ($>$, $=$, < 0).

Esempio. shift dx 1 di una parola su N bit:



Il bit a_0 si "perde".
 Il bit $a_{N-1} = 0$.
 $a_k = a_{k+1}$ per $k=0,1,2,\dots,N-2$

Esempio. shift sx 1 di una parola su N bit:



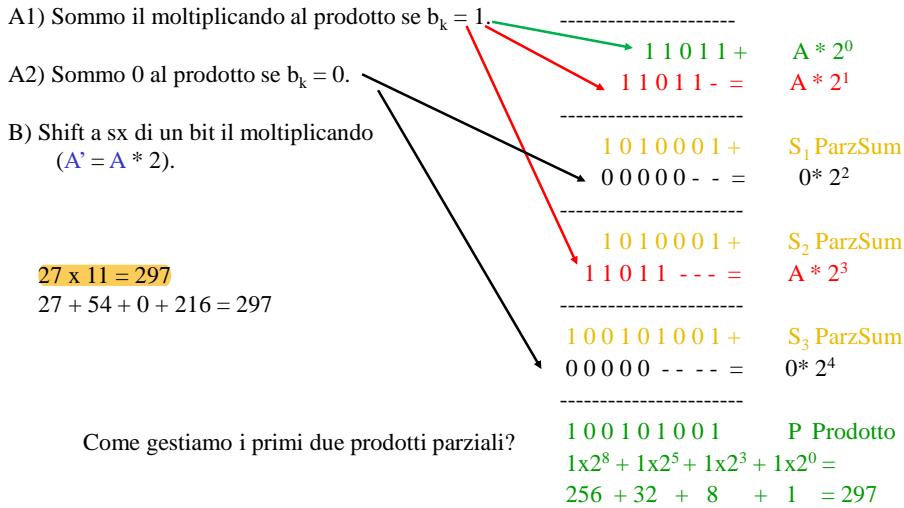
Il bit a_{N-1} si "perde".
 Il bit $a_0 = 0$.
 $a_{k+1} = a_k$ per $k=0,1,2,\dots, N-2$



Moltiplicazione utilizzando somma e shift



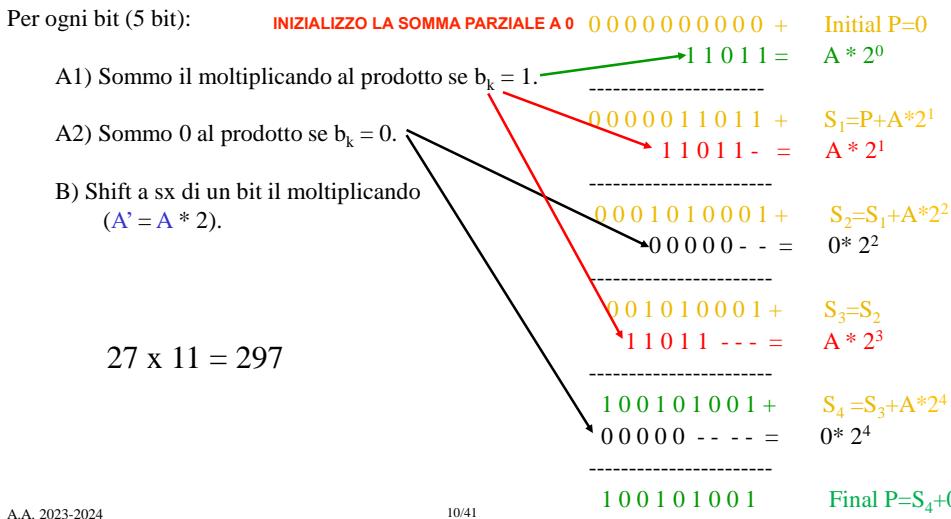
Analizzo sequenzialmente **ogni bit b_k del moltiplicatore**:



Moltiplicazione utilizzando somma e shift



Analizzo sequenzialmente **ogni bit b_k del moltiplicatore**
e applico ad ogni passo le stesse operazioni.





Moltiplicazione utilizzando somma e shift



Analizzo sequenzialmente **ogni bit b_k del moltiplicatore**
e applico ad ogni passo le stesse operazioni.

Per ogni bit (5 bit):

A1) Sommo il moltiplicando al prodotto se $b_k = 1$.

A2) Sommo 0 al prodotto se $b_k = 0$.

B) Shift a sx di un bit il moltiplicando
($A' = A * 2$).

$$27 \times 11 = 297$$

P contiene le somme parziali, al termine conterrà la somma totale, cioè il risultato del prodotto.

$$\begin{array}{r} 11011 \\ \times \quad \quad \quad A \\ 01011 = \quad B \end{array}$$

$$\begin{array}{r} 0000000000 + \quad \text{Initial } P=0 \\ 11011 = \quad A * 2^0 \end{array}$$

$$\begin{array}{r} 0000011011 + \quad S_1 = P + A * 2^1 \\ 11011 - = \quad A * 2^1 \end{array}$$

$$\begin{array}{r} 0001010001 + \quad S_2 = S_1 + A * 2^2 \\ 00000 - - = \quad 0 * 2^2 \end{array}$$

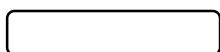
$$\begin{array}{r} 001010001 + \quad S_3 = S_2 \\ 11011 - - = \quad A * 2^3 \end{array}$$

$$\begin{array}{r} 100101001 + \quad S_4 = S_3 + A * 2^4 \\ 00000 - - = \quad 0 * 2^4 \end{array}$$

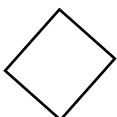
$$\begin{array}{r} 100101001 \quad \text{Final } P = S_4 + 0 \\ \hline \end{array}$$



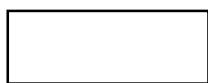
Diagrammi di flusso (flow chart)



Inizio / terminazione



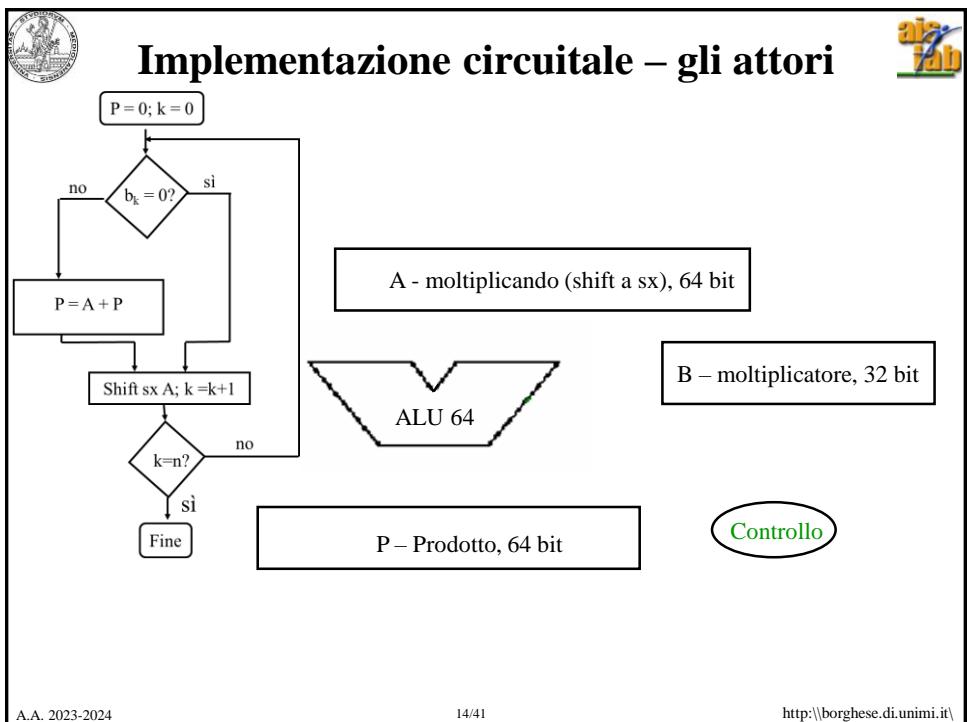
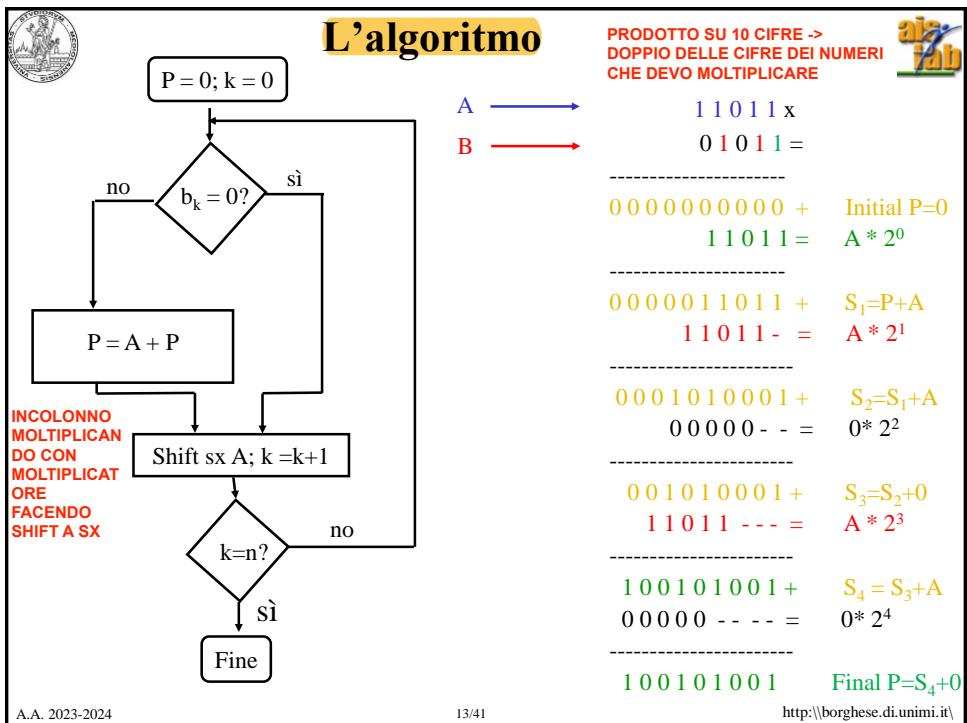
Test



Processo (esecuzione)

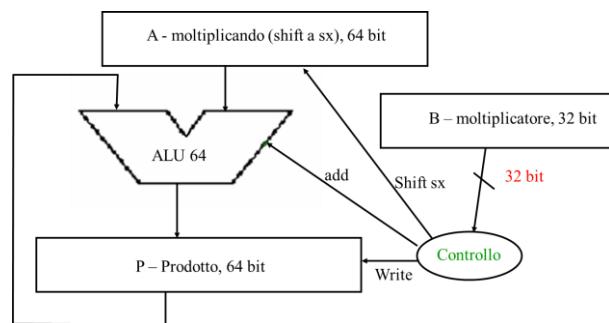
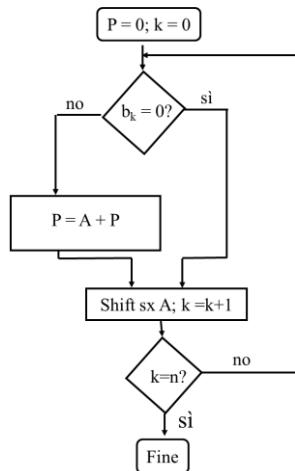
P INIZIALIZZATO A 0
è ACCUMULATORE DI SOMME PARZIALI

K CONTA LE CIFRE DEL MOLTIPLICATORE





Implementazione circuitale



Qual'è il problema?



Esempio su 4 bit

INIZIALIZZO VALORI



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000

$$\begin{array}{l} A \longrightarrow 1010x \\ B \longrightarrow 1011 = \end{array}$$

$$\begin{array}{r} 00000000 + \\ 1010 = \end{array}$$

$$A = A * 2^0 \qquad \qquad \qquad 1010 = A^0$$

$$\begin{array}{r} 00001010 + \\ 1010 - \end{array}$$

$$\begin{array}{r} 00011110 + \\ 0000 - - \end{array}$$

$$A^1 = A * 2^1 \qquad \qquad \qquad S_1 \qquad \qquad \qquad A^1$$

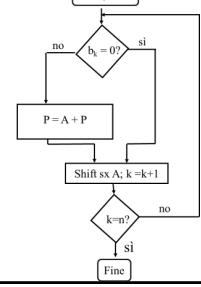
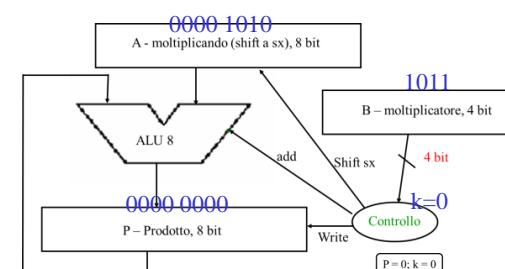
$$\begin{array}{r} 00011110 + \\ 1010 - - - \end{array}$$

$$A^2 = A * 2^2 \qquad \qquad \qquad S_2 \qquad \qquad \qquad A^2$$

$$\begin{array}{r} 00011110 + \\ 1010 - - - \end{array}$$

$$A^3 = A * 2^3 \qquad \qquad \qquad S_3 \qquad \qquad \qquad A^3$$

$$P \longrightarrow 01101110 \qquad \qquad \qquad 110_{10}$$



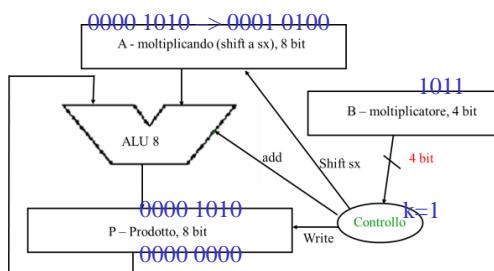
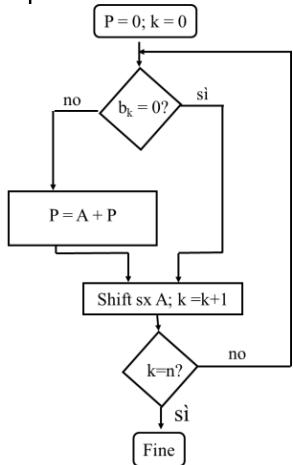


Esempio – passo 1



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010

$$10*1 + 0 \\ = 10$$



A.A. 2023-2024

17/41

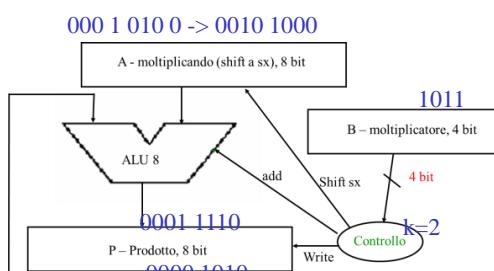
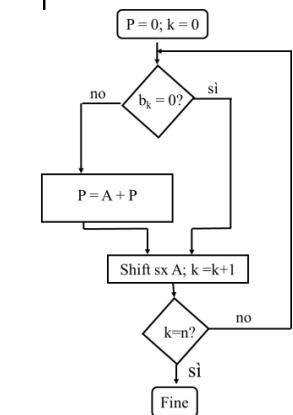
<http://borghese.di.unimi.it/>

Esempio – passo 2



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110

$$10*2+10 \\ = 30$$



A.A. 2023-2024

18/41

<http://borghese.di.unimi.it/>

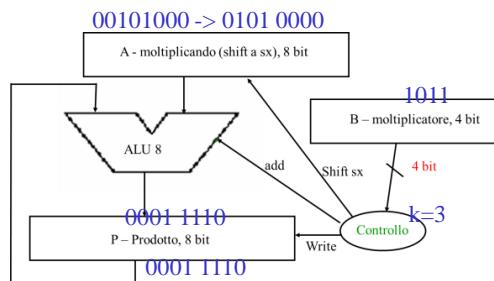
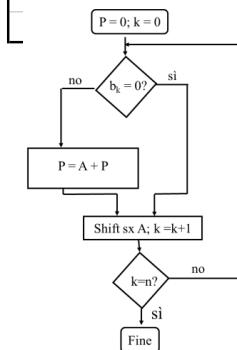


Esempio – passo 3



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	$b_1=1 \rightarrow P=P+A$	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	$b_2=0 \rightarrow \text{Nulla}$	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110

$$0 + 30 = 30$$



A.A. 2023-2024

19/41

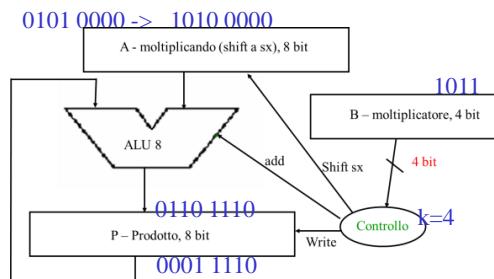
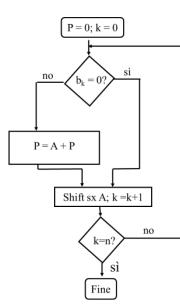
<http://borghese.di.unimi.it/>

Esempio – passo 4



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	$b_1=1 \rightarrow P=P+A$	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	$b_2=0 \rightarrow \text{Nulla}$	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	$b_3=1 \rightarrow P=P+A$	1011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110

$$10*8 + 30 = 110$$



A.A. 2023-2024

20/41

<http://borghese.di.unimi.it/>

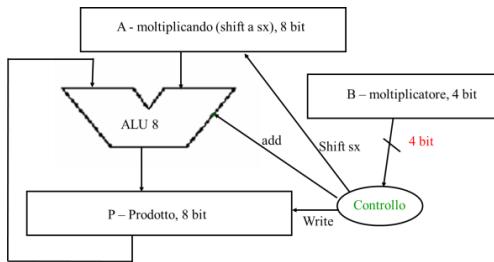
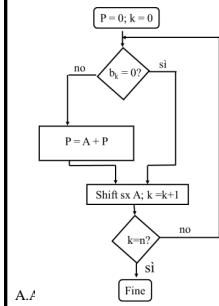
0000 1010 X

1011



Esempio – riassunto

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1010	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1001	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	b3=1->P=P+A	01011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110



21/41

<http://borghese.di.unimi.it/>



Esercizi



Costruire il circuito HW che esegui la moltiplicazione 7 x 9 in base 2 su 4 bit.

Eseguire la stessa moltiplicazione secondo l'algoritmo visto, indicando passo per passo il contenuto dei 3 registri: A che contiene il moltiplicando, B che contiene il moltiplicatore e P che contiene somme parziali ed il risultato finale.



Sommario



I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1010	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1001	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	b3=1->P=P+A	0011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110



Ottimizzazione

- Inizializzo B (ho tutti i bit di B)
- A ogni passo leggo B, ma utilizzo solo 1 bit, b_k
- Utilizzo b_k a ogni iterazione, **poi non serve più**.
- Non è necessario conservare tutti i bit di B per tutta la durata dell'operazione.

Situazione del tipo: **produttore-consumatore**.

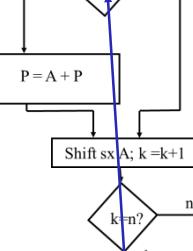
Produco dei dati: la parola B.

Consumo i dati: 1 bit della parola B a ogni iterazione (consume = utilizzo una tantum e non riutilizzo in seguito).

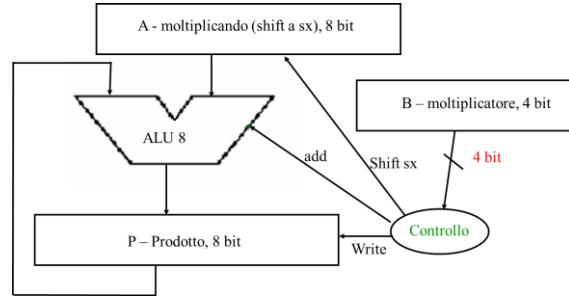


P = 0; k = 0

no $b_k = 0?$ sì



Razionale - I



Per scegliere, b_k , a ogni passo k, serve un mux all'interno dell'unità di controllo.

Posso ottenere lo stesso effetto in altro modo:

- Leggo b_0
- Shift a dx di una posizione di B così leggo solo il bit di interesse
una volta letto posso buttare via il bit facendo shift a destra

Espongo così all'unità di controllo b_k a ogni iterazione.



Implementazione circuitale ottimizzata - I

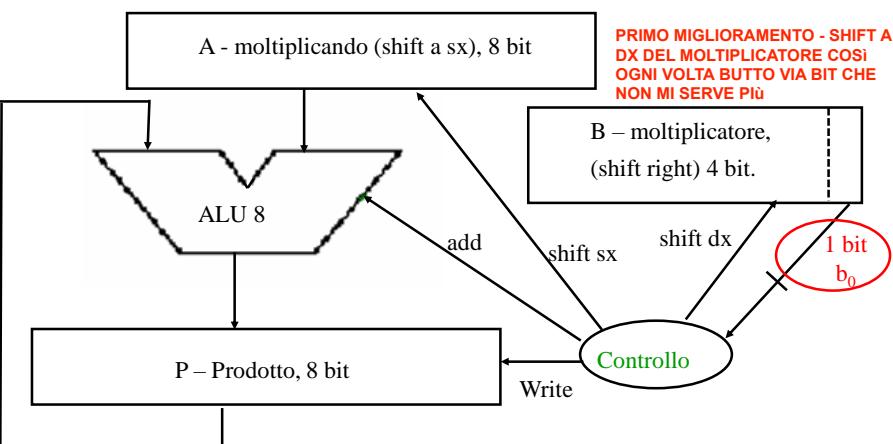


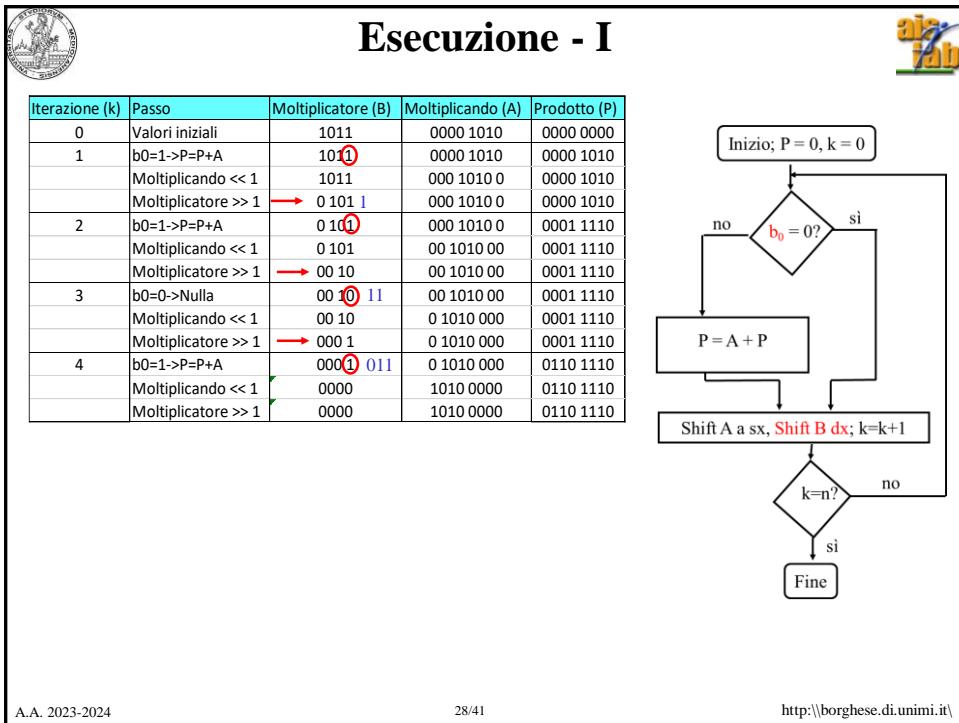
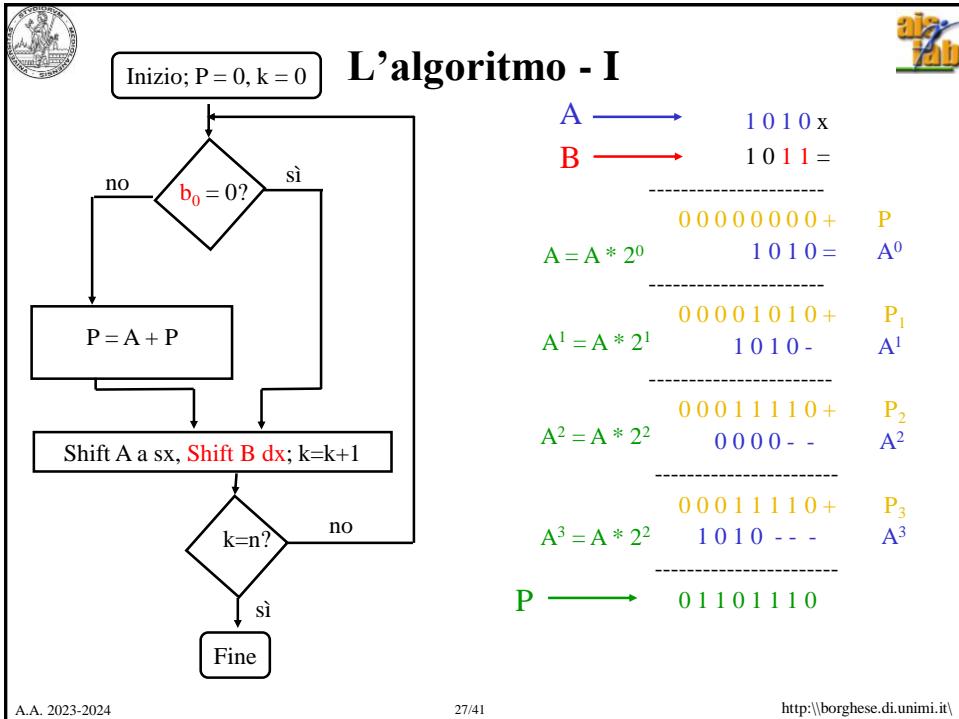
A - moltiplicando (shift a sx), 8 bit

PRIMO MIGLIORAMENTO - SHIFT A DX DEL MOLTIPLICATORE COSÌ OGNI VOLTA BUTTO VIA BIT CHE NON MI SERVE Più

B - moltiplicatore,
(shift right) 4 bit.

1 bit
 b_0







Razionale per una seconda implementazione



Metà dei bit del registro moltiplicando vengono utilizzati a ogni iterazione.

Gli N bit del moltiplicando sommati al registro prodotto vengono incolonnati di una posizione più a sinistra a ogni iterazione. Occupano N bit.

Ad ogni iterazione 1 bit del registro prodotto viene calcolato definitivamente.

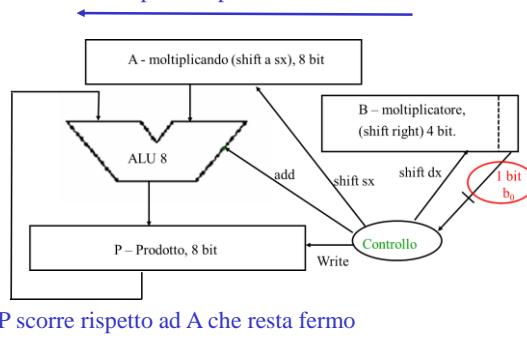
Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	b0=1->P=P+A	0 101	000 1010 0	000 1111 0
	Moltiplicando << 1	0 101	00 1010 00	0001 1110
	Moltiplicatore >> 1	00 10	00 1010 00	0001 1110
3	b0=0->Nulla	00 10	00 1010 00	00 01 1110
	Moltiplicando << 1	00 10	0 1010 000	0001 1110
	Moltiplicatore >> 1	000 1	0 1010 000	0001 1110
4	b0=1->P=P+A	000 1	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110



Analisi dello shift

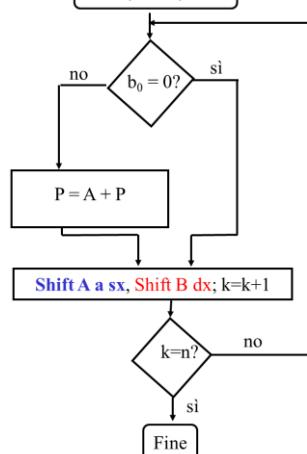


A scorre rispetto al prodotto P che resta fermo



P scorre rispetto ad A che resta fermo

Inizio; $P = 0, k = 0$





Razionale per una seconda implementazione



Ad ogni iterazione sommo N cifre (pari al numero di cifre del moltiplicando).

Spostamento di A a sx rispetto al registro prodotto, P.

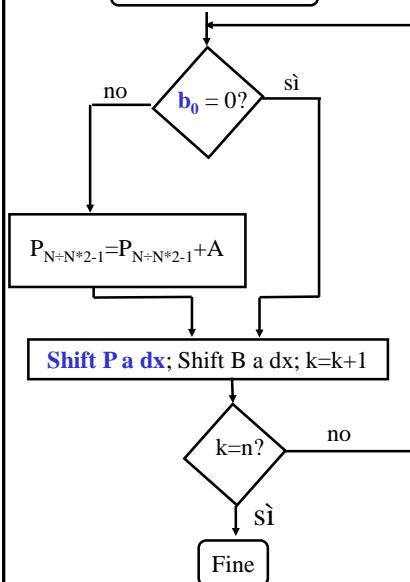
$$\begin{array}{r}
 1010x \\
 1011 = \\
 \hline
 00000000+ \\
 1010 = \\
 \hline
 00001010+ \\
 1010 - \\
 \hline
 00011110+ \\
 0000 - - \\
 \hline
 00011110+ \\
 1010 - - - \\
 \hline
 01101110
 \end{array}
 \quad \begin{array}{l} P \\ A^0 \\ P_1 \\ A^1 \\ P_2 \\ A^2 \\ P_3 \\ A^3 \end{array}$$

Spostamento di P a dx rispetto al registro moltiplicando, A

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1010	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	b0=1->P=P+A	0 100	000 1010 0	0001 1110
	Moltiplicando << 1	0 101	00 1010 00	0001 1110
	Moltiplicatore >> 1	00 10	00 1010 00	0001 1110
3	b0=0->Nulla	00 10	00 1010 00	0001 1110
	Moltiplicando << 1	00 10	0 1010 000	0001 1110
	Moltiplicatore >> 1	000 1	0 1010 000	0001 1110
4	b0=1->P=P+A	0000	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110



L'algoritmo ottimizzato - II



$$\begin{array}{r}
 A \longrightarrow 1010x \\
 B \longrightarrow 1011 = \\
 \hline
 00000000+ \\
 1010 = \\
 \hline
 00001010+ \\
 1010 - \\
 \hline
 00011110+ \\
 0000 - - \\
 \hline
 00011110+ \\
 1010 - - - \\
 \hline
 01101110
 \end{array}
 \quad \begin{array}{l} P \\ A^0 \\ P_1 \\ A^1 \\ P_2 \\ A^2 \\ P_3 \\ A^3 \end{array}$$



Implementazione ottimizzata - II



1 0 1 0 x

1 0 1 1 =

0 0 0 0 0 0 0 +	P
1 0 1 0 =	A ⁰

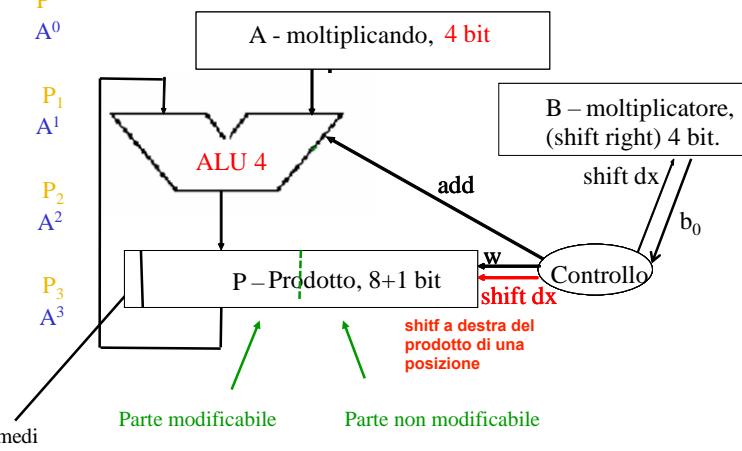
0 0 0 0 1 0 1 +	P ₁
1 0 1 0 -	A ¹

0 0 0 1 1 1 1 0 +	P ₂
0 0 0 0 - -	A ²

0 0 0 1 1 1 1 0 +	P ₃
1 0 1 0 - - -	A ³

0 1 1 0 1 1 1 0	

Riporto bit intermedi



A.A. 2023-2024

33/41

<http://borgheze.di.unimi.it/>

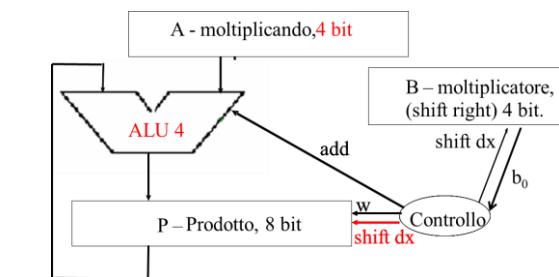
Esecuzione - II



Iterazione	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	1010	0000 0000
1	b0=1->P=P+A	1011	1010	1010 0000
	Prodotto >> 1	1011	1010	0 1010 0000
	Moltiplicatore >> 1	0 101	1010	0 1010 0000
2	b0=1->P=P+A	0 101	1010	0 1010 0000
	Prodotto >> 1	0 101	1010	0 1110 0000
	Moltiplicatore >> 1	0 0 10	1010	0 1110 0000
3	b0=0->Nulla	00 10	1010	0 1110 0000
	Prodotto >> 1	00 10	1010	0 0111 0000
	Moltiplicatore >> 1	000 1	1010	0 0111 0000
4	b0=1->P=P+A	000 1	1010	1 1011 0000
	Prodotto >> 1	0000	1010	0 110 1110
	Moltiplicatore >> 1	0000	1010	0111 1110

A.A. 2023-2024

34/41

<http://borgheze.di.unimi.it/>



Razionale dell'implementazione - III



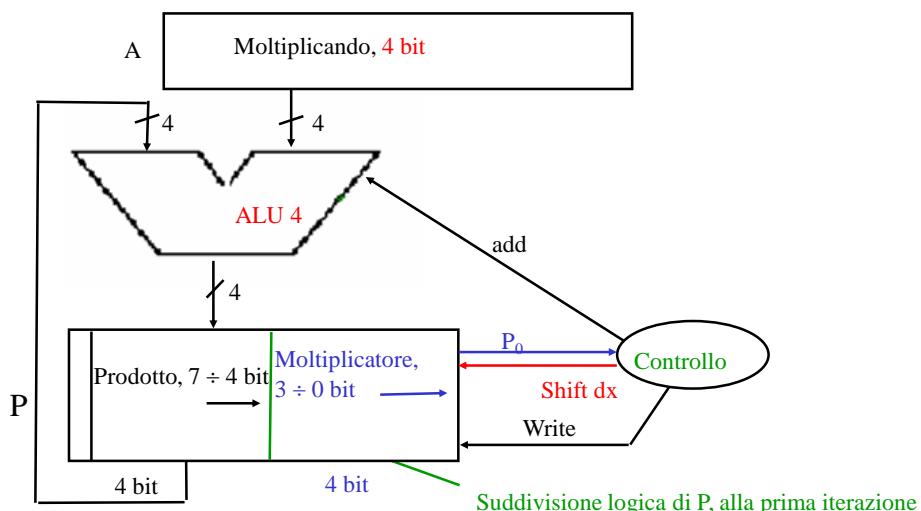
Il numero di bit del registro **prodotto** corrente (somma dei prodotti parziali) più il numero di bit da esaminare nel registro **moltiplicatore** rimane **costante** ad ogni iterazione (pari a 8 bit).

Si può perciò eliminare il registro moltiplicatore.

Iterazione	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	1010	1010 0000
	Prodotto >> 1	1011	1010	0 1010 000
	Moltiplicatore >> 1	0 101	1010	0 1010 000
2	$b_0=1 \rightarrow P=P+A$	0 101	1010	1 1110 000
	Prodotto >> 1	0 101	1010	0 1111 000
	Moltiplicatore >> 1	00 10	1010	0 1111 000
3	$b_0=0 \rightarrow$ Nulla	00 10	1010	0 1111 000
	Prodotto >> 1	00 10	1010	0 0111 100
	Moltiplicatore >> 1	000 1	1010	0 0111 100
4	$b_0=1 \rightarrow P=P+A$	000 1	1010	1 1011 100
	Prodotto >> 1	0000	1010	0 110 1110
	Moltiplicatore >> 1	0000	1010	0111 1110



Circuito ottimizzato - III

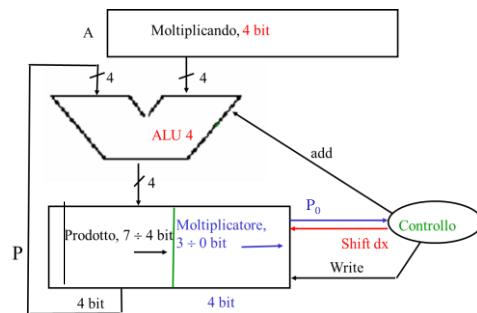
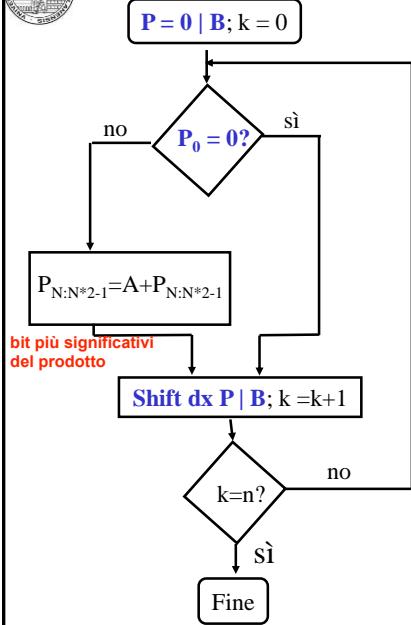


Il moltiplicando è allineato sempre ai 4 bit più significativi del prodotto.

Ad ogni iterazione, il prodotto si allarga, il moltiplicatore si restringe.



Algoritmo ottimizzato

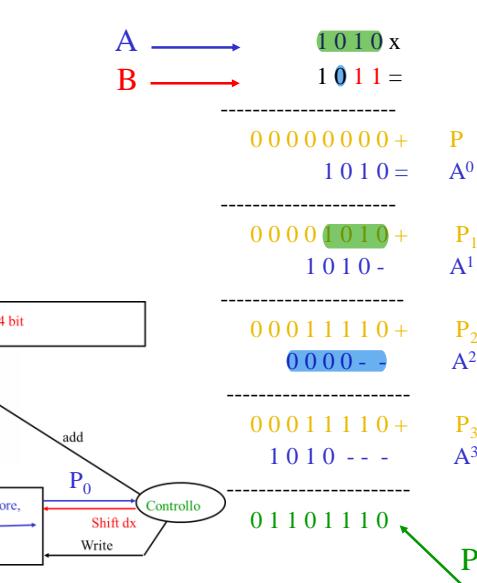
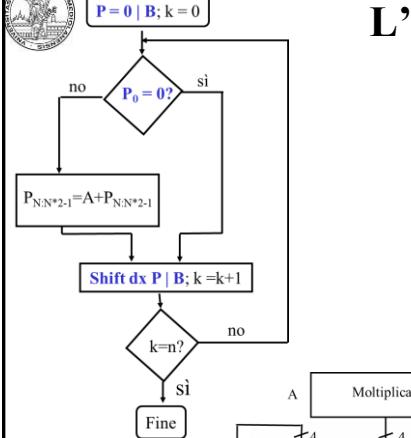


A.A. 2023-2024

37/41

<http://borghese.di.unimi.it/>

L'algoritmo ottimizzato - III



A.A. 2023-2024

38/41

<http://borghese.di.unimi.it/>



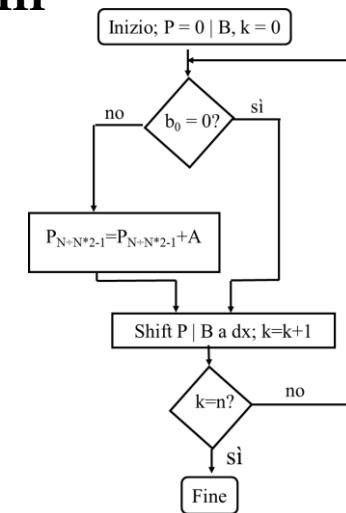
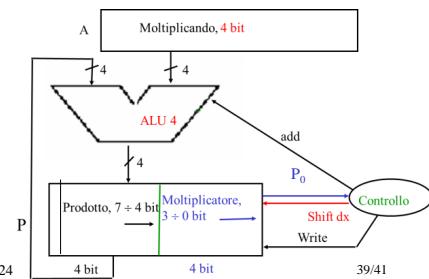
Esempio di esecuzione dell'algoritmo ottimizzato - III



Iterazione	Passo	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1010	0000 1011
1	p0=1->P=P+A	1010	1010 1011
	Prodotto >> 1	1010	0 1010 101
2	p0=1->P=P+A	1010	1 1110 101
	Prodotto >> 1	1010	0 11110 10
3	p0=0->Nulla	1010	0 11110 10
	Prodotto >> 1	1010	0 011110 1
4	p0=1->P=P+A	1010	1 101110 0
	Prodotto >> 1	1010	0111 1110

A.A. 2023-2024

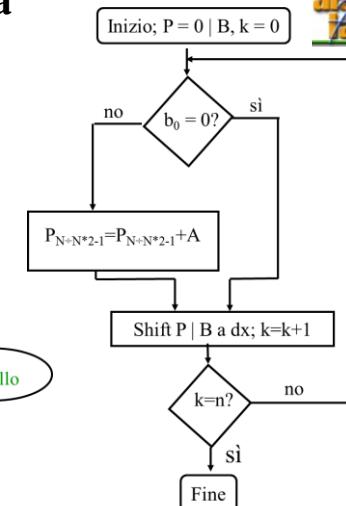
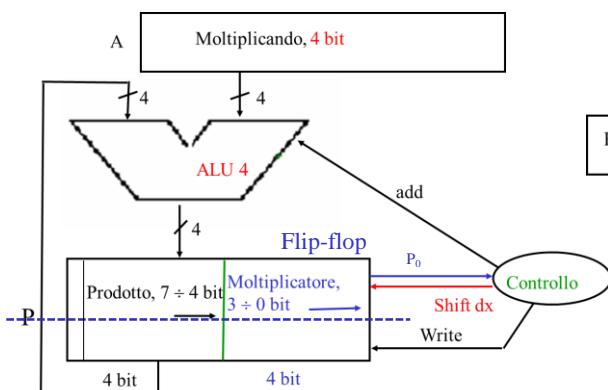
39/41

<http://borghese.di.unimi.it/>

Complessità



Latch



- Registro moltiplicando ($4 \times 4 = 16$) – latch. Viene solo letto.
- Registro Prodotto ($(8+1) \times 8 = 72$) – Flip flop perché registro a scorrimento e perché il suo contenuto viene letto e scritto.
- ALU4 ($5 \times 4 = 20$)
- UC ?

(Moltiplicatore HW aveva complessità 65 porte logiche), ma questo circuito può essere utilizzato anche per la divisione...

mi.it\



Sommario



I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware



Firmware Division

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti sul Patterson 5a ed.: 3.4, 3.5



Sommario

- **Divisione intera**
- Circuiti divisione intera
- Divisione e moltiplicazione



Moltiplicazione



$$\begin{array}{r}
 128 \times \\
 214 = \\
 \hline
 512 + \quad x10^0 \\
 128 - \quad x10^1 \\
 256 - = \quad x10^2 \\
 \hline
 27392
 \end{array}
 \quad
 \begin{array}{l}
 128x(4x10^0) = \text{unità} \\
 128x(1x10^1) = \text{decine} \\
 128x(2x10^2) = \text{centinaia} \\
 27392
 \end{array}$$

Prodotto come **somma dei prodotti parziali**. Ognuno dei prodotti parziali aggiunge il **moltiplicando pesato con il peso della cifra analizzata del moltiplicatore** x un numero di volte pari alla cifra analizzata del moltiplicatore.

Nel caso binario le cifre sono solo 0,1 ==>



Algoritmi per la moltiplicazione



Il rationale degli algoritmi firmware della moltiplicazione è il seguente.

Si analizzano sequenzialmente i bit del moltiplicatore e:

- 1) Si mette 0 nella posizione opportuna (se il bit analizzato del moltiplicatore = 0).
- 2) Si mette una copia del moltiplicando nella posizione opportuna (se il bit analizzato del moltiplicatore è = 1).

$$\begin{array}{r}
 \text{Moltiplicando} \quad 11011 \times \\
 \text{Moltiplicatore} \quad 101 = \\
 \hline
 11011 + \\
 00000 - \\
 \hline
 11011 \\
 11011 - - \\
 \hline
 \text{Prodotto} \quad 10000111
 \end{array}$$

La moltiplicazione viene effettuata come somme successive, con peso crescente, di uno tra i 2 valori: {moltiplicando, 0}



La divisione decimale



Dividendo Divisore

$$3716 : 12 = 309$$

36--

11-

0-

116

108

8

Resto

Dividendo = Divisore * Quoziente (quoto) + Resto

A.A. 2023-2024

5/74

<http://borghese.di.unimi.it/>

Razionale della divisione



1. Analizzo il dividendo

$$3716 : 12 = 309$$

36--

11-

0-

116

108

8

2. Identifico quale potenza di 10 iniziare a considerare. Questa è la **massima potenza, p**, tale per cui 10^p moltiplicata per il divisore mi dà un valore inferiore al dividendo.

In questo caso: **12 divisore**

$$p = 1 \rightarrow 10 * 12 = 120$$

$$\text{p} = 2 \rightarrow 100 * 12 = 1200 \quad \text{cifra massima che riesco a togliere dal dividendo}$$

$$p = 3 - 1000 * 12 = 12000.$$

3. Determino qual è il **numero massimo di volte**, q_2 , per cui il divisore $\times 10^p$ può essere moltiplicato per ottenere un numero inferiore al dividendo.

In questo caso:

$$q_2 = 1 \rightarrow 1 * 1200 = 1200$$

$$q_2 = 2 \rightarrow 2 * 1200 = 2400$$

$$q_2 = 3 \rightarrow 3 * 1200 = 3600 \quad \text{da sottrarre al dividendo}$$

$$q_2 = 4 \rightarrow 4 * 1200 = 4800$$

NB la moltiplicazione $\times 10^p$ si sottoindende, allineando il $q_2 \times$ divisore alla cifra opportuna.
In questo caso $3 \times 12 = 36$ allineato alle centinaia.

i.it\



Razionale della divisione

$$\begin{array}{r} \overline{3716 : 12 = 309} \\ 3600 \\ \hline \end{array}$$

$$\begin{array}{r} \overline{3716 : 12 = 309} \\ 36-- \\ \hline \end{array}$$

air
lab

Centinaia

$$\begin{array}{r} \overline{116} \\ 0000 \\ \hline \end{array} \quad \begin{array}{r} 11- \\ 0- \quad \text{Decine} \\ \hline \end{array}$$

$$\begin{array}{r} 116 \\ 108 \\ \hline \end{array} \quad \begin{array}{r} 116 \\ 108 \quad \text{Unità} \\ \hline \end{array}$$

$$\begin{array}{r} 8 \\ \hline \end{array} \quad \begin{array}{r} 8 \\ \hline \end{array}$$

4. Ho stabilito $p = 2$, $q_2 = 3 \rightarrow$ posso erodere una quantità pari a: $3 * 12 * 10^2 = 3600$

$$\begin{array}{r} 3716 - \\ 3600 = \\ \hline \end{array} \quad q_2 = 3 \text{ è la cifra del quoziente cioè di quanto stiamo erodendo}$$

$p = 2 \rightarrow 10^2$ è il peso del divisore

$$\begin{array}{r} 116 \\ \hline \end{array} \quad \text{Resto parziale}$$

Procedo ricorsivamente ripetendo i passi 2-4 sul resto fino a quando non ottengo un resto inferiore al divisore.

I passi successive al primo sono leggermente diversi.



Sviluppo della divisione - I

$$\begin{array}{r} \overline{3716 : 12 = 309} \\ 36-- \\ \hline \end{array}$$

air
lab

$$\begin{array}{r} 11- \\ 0- \\ \hline \end{array}$$

1. Analizzo il dividendo

2. Considero la potenza $p = \mathbf{p-1}$. La potenza 1 in questo caso.

$$p = 1 \rightarrow 10^1 * 12 = 120$$

$$\begin{array}{r} 116 \\ 108 \\ \hline \end{array}$$

$$8$$

3. Determino qual è il **numero massimo di volte**, q_1 , per cui il divisore $x 10^p$ può essere moltiplicato per ottenere un numero inferiore al dividendo.

In questo caso:

$$q_1 = 0 \rightarrow 0 * 120 = 0$$

$$q_1 = 1 \rightarrow 1 * 120 = 120$$

4. Ho stabilito $p = 1$, $q_1 = 0 \rightarrow$ posso erodere una quantità pari a: $0 * 12 * 10^1 = 0$

$$\begin{array}{r} 116 - \\ 0- = \\ \hline \end{array} \quad q_1 = 0 \text{ è la cifra del quoziente cioè di quanto stiamo erodendo}$$

$p = 1 \rightarrow 10^1$ è il peso del divisore

$$\begin{array}{r} 116 \\ \hline \end{array} \quad \text{Resto parziale}$$



Sviluppo della divisione - II

$$\begin{array}{r} 3716 : 12 = 309 \\ 36 \\ \hline 11 \\ 0 \\ \hline 116 \\ 108 \\ \hline 8 \end{array}$$

**air
lab**

1. Analizzo il moltiplicatore

2. Considero la potenza $p=p-1, 0$ in questo caso.

$$p = 0 \rightarrow 1 * 12 = 12$$

3. Determino qual è il **numero massimo di volte**, q_0 , per cui il divisore $x 10^p$ può essere moltiplicato per ottenere un numero inferiore al dividendo.

In questo caso:

$$q_0 = 8 \rightarrow 8 * 12 = 96$$

$$q_0 = 9 \rightarrow 9 * 12 = 108$$

4. Ho stabilito $p = 0$, $q_0 = 9 \rightarrow$ posso **erodere una quantità pari a: $9 * 12 * 10^0 = 108$**

$$\begin{array}{r} 116 \\ 108 \\ \hline 8 \end{array} \quad \begin{array}{l} q_0 = 9 \text{ è la cifra del quoziente cioè di quanto stiamo erodendo} \\ p = 0 \rightarrow 10^0 \text{ è il peso del divisore} \\ \text{Resto totale} \end{array}$$



Razionale della divisione decimale

**air
lab**

La divisione è l'operazione inversa del prodotto

$$2629 : 12 = 219 \quad (\text{resto } = 1) \quad \rightarrow \quad 219 \times 12 + 1 = 2629$$

$$[(2 \times 10^2) + (1 \times 10^1) + (9 \times 10^0)] \times 12 + 1 = 2629$$

Da cui segue che ad ogni passo della divisione erodiamo (**sottraiamo**) una quantità via **via decrescente**. Al primo passo sottraiamo a entrambi i membri:

$$1. [(1 \times 10^1) + (9 \times 10^0)] \times 12 + 1 = 2629 - (2 \times 10^2) \times 12 = 2629 - 2400 = 229 \rightarrow \text{resto parziale.}$$

Abbiamo cioè eroso 24 centinaia al dividendo 2629.

2. Al passo successivo eroderemo al resto parziale 12 decine:

$$[(9 \times 10^0)] \times 12 + 1 = 229 - (1 \times 10^1) \times 12 = 229 - 120 = 109 \rightarrow \text{resto parziale}$$

3. E al passo finale eroderemo $12 \times 9 = 108$ unità:

$$[1 = 109 - (9 \times 10^0) \times 12 = 1] \rightarrow \text{resto finale.}$$

Ad ogni passo si cerca di erodere il più possibile!



Confronto con la moltiplicazione



La divisione è l'operazione inversa del prodotto

$$2629 : 12 = 219 \quad (\text{resto} = 1) \quad \rightarrow \quad 219 \times 12 + 1 = 2629$$

Divisione: Ad ogni passo erodiamo (sottraiamo) un certo numero di volte il divisore moltiplicato per la massima cifra possibile (cifra del quoziente), moltiplicato anche per la potenza associata alla posizione della cifra del dividendo, che stiamo considerando.

Opera per sottrazioni successive di quantità sempre più piccole proporzionali al divisore, producendo via via i resti parziali.

Moltiplicazione: a ogni passo sommiamo il moltiplicando moltiplicato per il numero di volte indicato dalla cifra del moltiplicatore considerata e per la potenza associata alla posizione della cifra del moltiplicatore, che stiamo considerando.

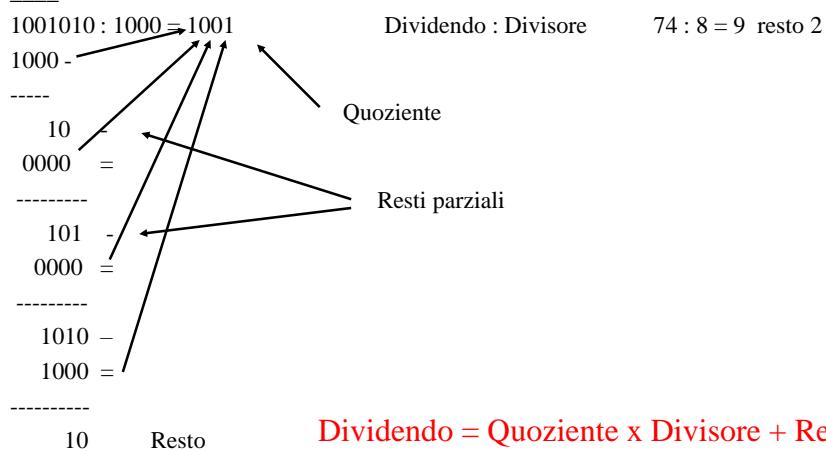
Opera per somme successive di quantità sempre più grandi proporzionali al moltiplicatore, producendo via via le somme parziali.



La divisione tra numeri binari



Divisione decimale fra i numeri $a = 1.001.010$ e $b = 1.000$ $a : b = ?$





Confronto tra divisione tra numeri binari e decimali



In DECIMALE:

Ad ogni passo devo verificare QUANTE VOLTE il resto parziale contiene il divisore. Il risultato è un numero che va da 0 a 9 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. 0 = il divisore non è contenuto nel resto.

In BINARIO:

Ad ogni passo devo verificare SE il resto parziale contiene il divisore. Ovverosia se lo contiene 0 o 1 volta. Il risultato è un numero che può valere {0, 1}.

$$\begin{array}{r} \text{---} \\ 3716 : 12 = 309 \\ \text{---} \\ 3600 \\ \text{---} \\ 116 \\ \text{---} \\ 00 \\ \text{---} \\ 116 \\ \text{---} \\ 108 \\ \text{---} \\ 8 \end{array} \qquad \begin{array}{r} \text{---} \\ 1001010 : 1000 = 1001 \\ \text{---} \\ 1010 \\ \text{---} \\ 000000 \\ \text{---} \\ 1010 \\ \text{---} \\ 00000 \\ \text{---} \\ 1010 \\ \text{---} \\ 1000 \\ \text{---} \end{array}$$



Peculiarità della divisione



- Come rappresento la condizione “il divisore è contenuto nel resto parziale”?

Esempi:

- 10 (resto parziale) non contiene 1000 (divisore)
1001 (resto parziale) contiene 1000 (divisore)

Per tentativi.

Eseguo la sottrazione.

differenza positiva

Risultato $\geq 0 \rightarrow$ il resto parziale è maggiore del divisore.

Risultato $< 0 \rightarrow$ il resto parziale è minore del divisore (non contiene il divisore)

differenza negativa

Osserviamo che nel secondo caso abbiamo fatto in realtà una sottrazione che non avremmo dovuto effettuare.

NB il calcolatore non può sapere se il resto parziale contiene il divisore fino a quando non ha effettuato la sottrazione.



La divisione tra numeri binari



Divisione decimale fra i numeri su 7 bit: $a = \textcolor{red}{100\ 1010}$ e $b = \textcolor{blue}{000\ 1000}$ $a : b = ?$ $74 : 8 = ?$

Nel primo passaggio allineo il divisore alla sinistra della prima cifra (MSB) del dividendo.

Allocazione di 14 bit. Sottrazione -> somma in complemento a 2:

$$\begin{array}{r}
 000\ 0000\ \textcolor{red}{100\ 1010} - \\
 \textcolor{blue}{000\ 1000}\ 000\ 0000 = \\
 \hline
 111\ 1000\ 100\ 1010
 \end{array}
 \quad
 \begin{array}{r}
 000\ 0000\ \textcolor{red}{100\ 1010} + \\
 \textcolor{blue}{111\ 1000\ 000\ 0000} = \\
 \hline
 111\ 1000\ 100\ 1010
 \end{array}
 \quad
 \begin{array}{l}
 \text{Resto parziale} = \text{dividendo} - \text{divisore} * 2^7 \\
 = 74 - 8 * 2^7 = 74 - 1024 = -950 \\
 \quad (= -950|_{10}) \\
 \quad \text{Nuovo resto parziale} \\
 \quad \text{provvisorio}
 \end{array}$$

Ripristino il resto parziale precedente: 0000 000 100 1010



Note e strategia di implementazione



All'inizio il divisore è allineato alla sinistra del dividendo: le cifre del dividendo sono allineate agli 0 del divisore e gli 0 del dividendo sono allineati alle cifre del divisore.

Il divisore viene spostato verso dx di 1 bit ad ogni passo.

Il quoziente cresce dal bit più significativo verso il bit meno significativo. Cresce verso dx.

Ci sono $N+1$ passi di divisione, il primo darà sempre 0 e si potrebbe omettere.

Occorre quindi effettuare a ogni passo:

Resto parziale = Resto parziale – divisore * base corrente

Ripristino del resto parziale (se sottrazione < 0)

Shift quoziante a sx.

Scrittura di 1 o 0 nel registro quoziante.

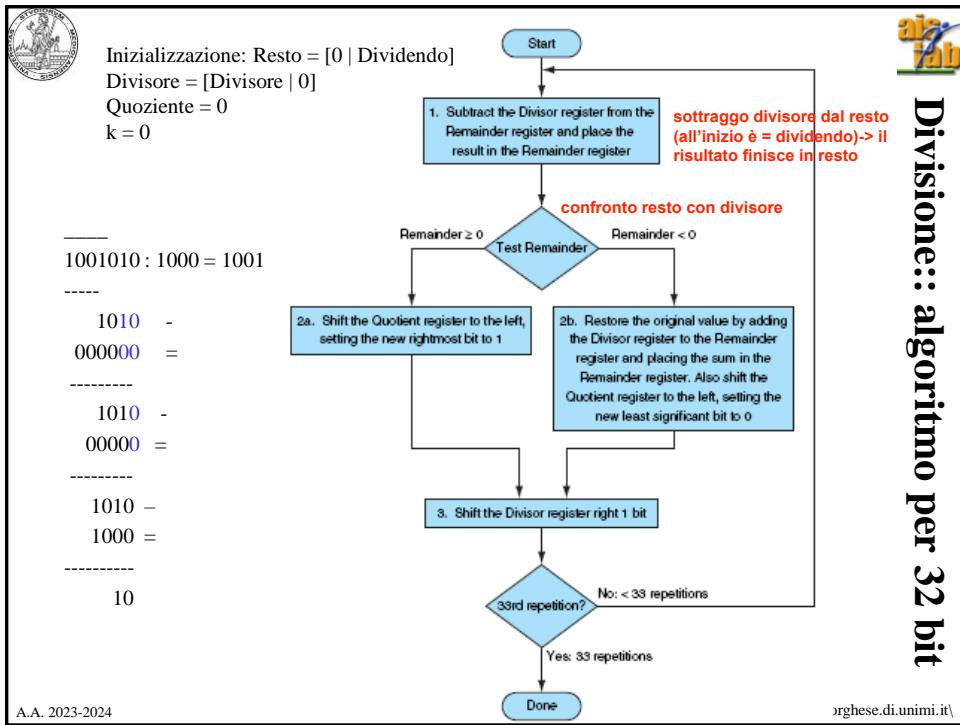
Shift del divisore verso dx.

Utilizzo un unico registro per dividendo e resto (il resto si ottiene per erosione del dividendo).

Considero il primo resto parziale uguale al dividendo (inizializzazione).

Il primo passo sarà "a vuoto" perché produrrà sempre quoziante 0.

Divisione:: algoritmo per 32 bit



Esempio - 1

Divisione decimale fra i numeri a = 7 e b = 2 su 4 bit. a : b = ?

Inizializzo il divisore alla sinistra delle quattro cifre significative. La prima cifra del quoziente sarà sempre 0.

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem - Div	0000	0010 0000	110 0111-25
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	000 0111 7
	3: Shift Div right	0000	0001 0000	0000 0111

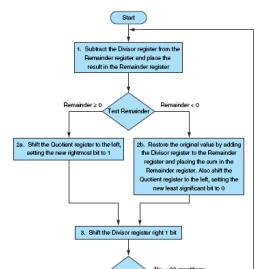
sottraggo 32
risomma 32

$$\text{Divisore} = 1 \times 2^4 \times 2^1 = 32$$

$$\text{Resto} = 7$$

$$7 - 32 = -25 < 0$$

$$\text{ripristino } -25+32 = 7$$





Esempio - 2



Divisione decimale fra i numeri a = 7 e b = 2 a : b = ?

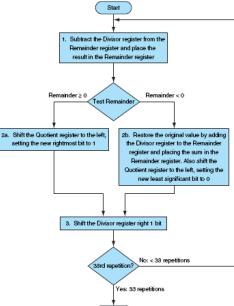
Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	110 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	110 0111 -9
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111 7
	3: Shift Div right	0000	0000 1000	0000 0111

sottraggo 16
risommo 16

$$\text{Divisore} = 1 \times 2^3 \times 2 = 16$$

$$\text{Resto} = 7$$

$$7 - 16 = -9 < 0$$



A.A. 2023-2024

19/74



Esempio - 3



Divisione decimale fra i numeri a = 7 e b = 2 a : b = ?

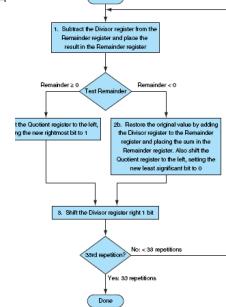
Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	110 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	110 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	111 1111 -1
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111 7
	3: Shift Div right	0000	0000 0100	0000 0111

sottraggo 8
risommo 8

$$\text{Divisore} = 1 \times 2^2 \times 2 = 8$$

$$\text{Resto} = 7$$

$$7 - 8 = -1 < 0$$



A.A. 2023-2024



Esempio - 4



Iteration	Step	Quotient	Divisor	Reminder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem - Div	0000	0010 0000	①110 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem - Div	0000	0001 0000	①111 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem - Div	0000	0000 1000	①111 1111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem - Div	0000	0000 0100	①000 0011 3 sottraggo 4
	2a: Rem $\geq 0 \Rightarrow$ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011

Divisore = $1 \times 2^1 \times 2 = 4$

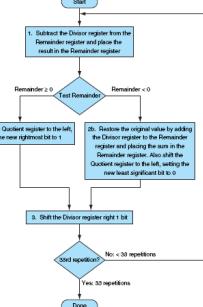
Resto = 7

$7 - 4 = +3 > 0$ nuovo resto

numero positivo \rightarrow
non devo ripristinare
resto precedente

inserisco 1 nel bit
meno significativo del
quoziente

shift a dx del divisore



Esempio - 5



Iteration	Step	Quotient	Divisor	Reminder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem - Div	0000	0010 0000	①110 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem - Div	0000	0001 0000	①111 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem - Div	0000	0000 1000	①111 1111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem - Div	0000	0000 0100	①000 0011
	2a: Rem $\geq 0 \Rightarrow$ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem - Div	0001	0000 0010	①000 0001
	2a: Rem $\geq 0 \Rightarrow$ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Divisore = $1 \times 2^0 \times 2 = 2$

Resto = 3

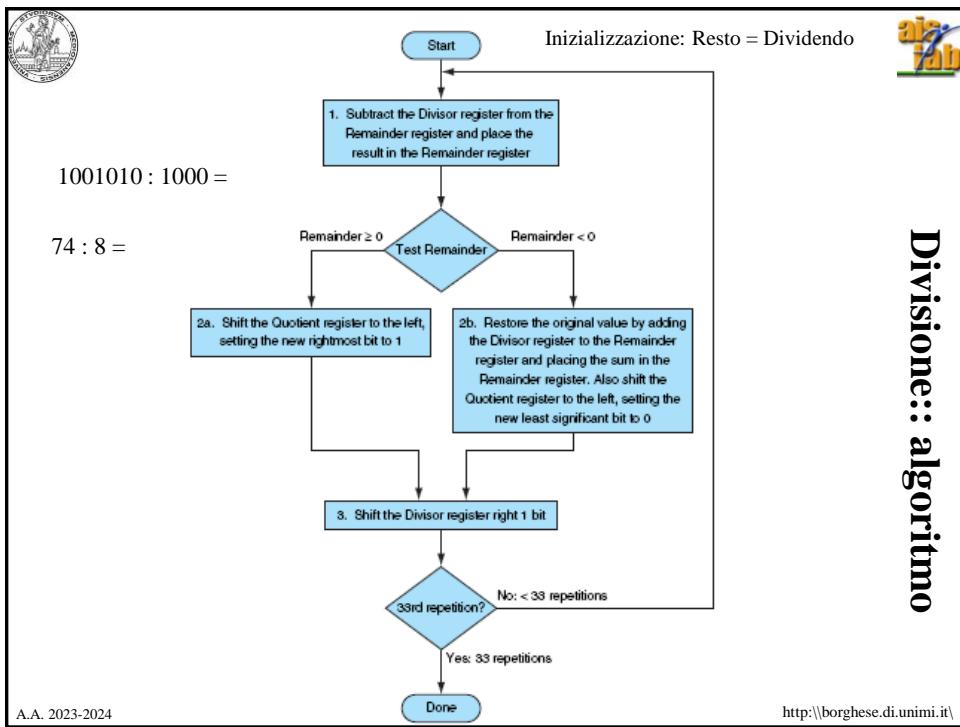
$3 - 2 = +1 > 0$ nuovo resto = resto finale

3-2=1
numero positivo \rightarrow
non devo ripristinare
resto precedente

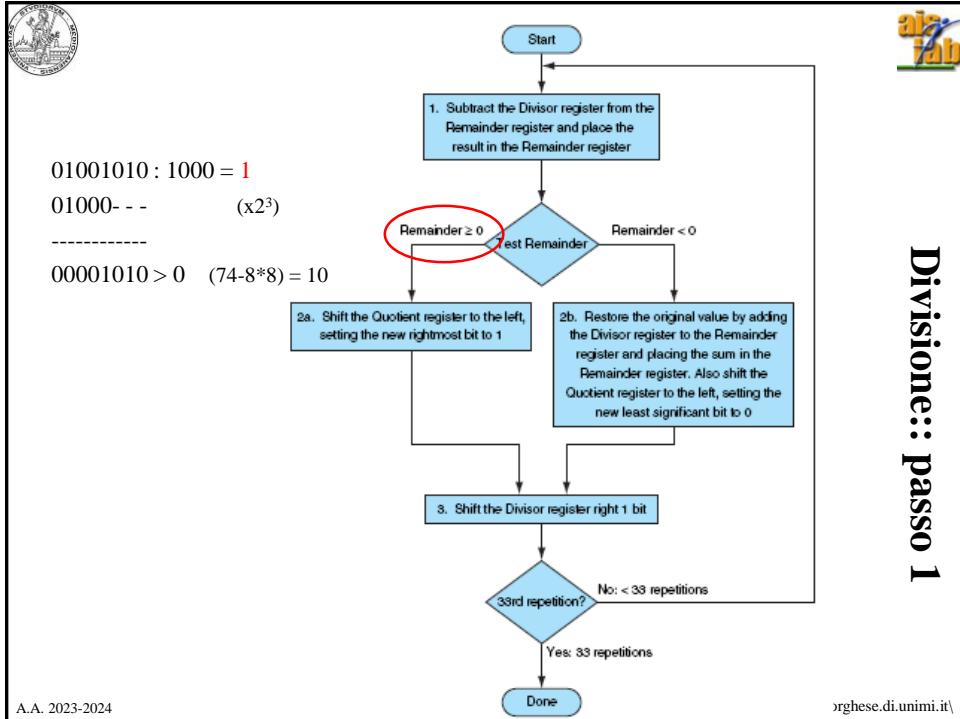
inserisco 1 nel bit
meno significativo del
quoziente

shift a dx del divisore

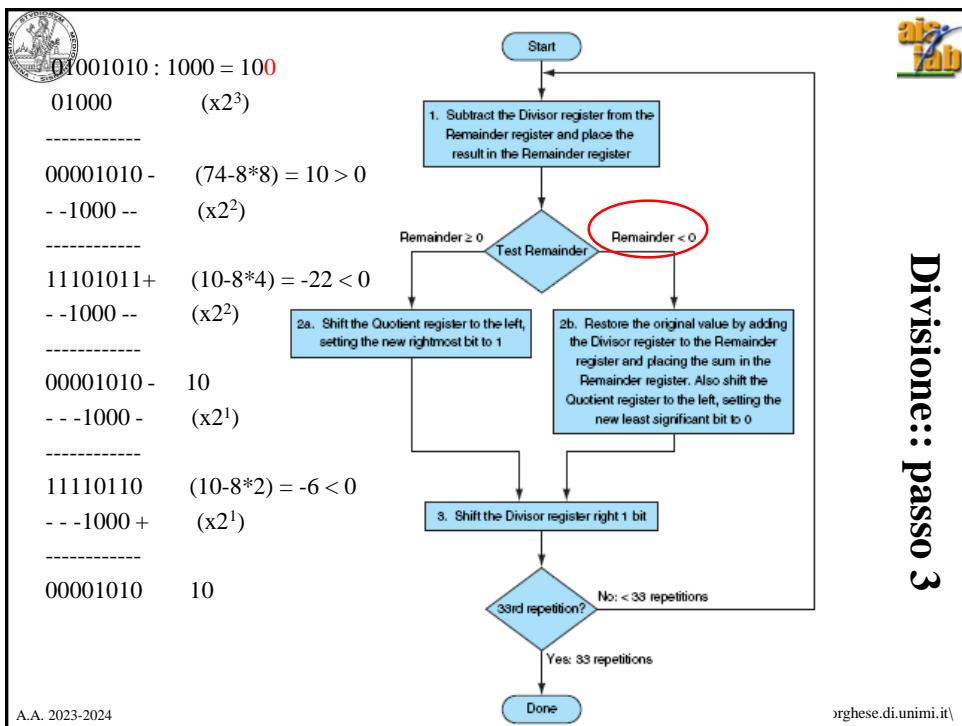
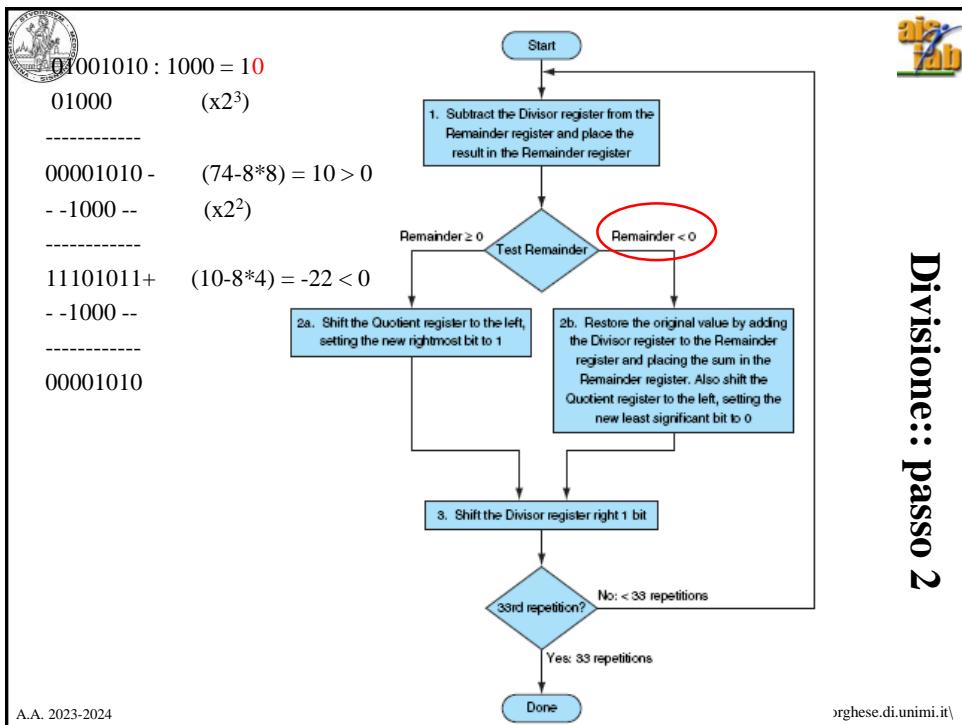
Divisione:: algoritmo



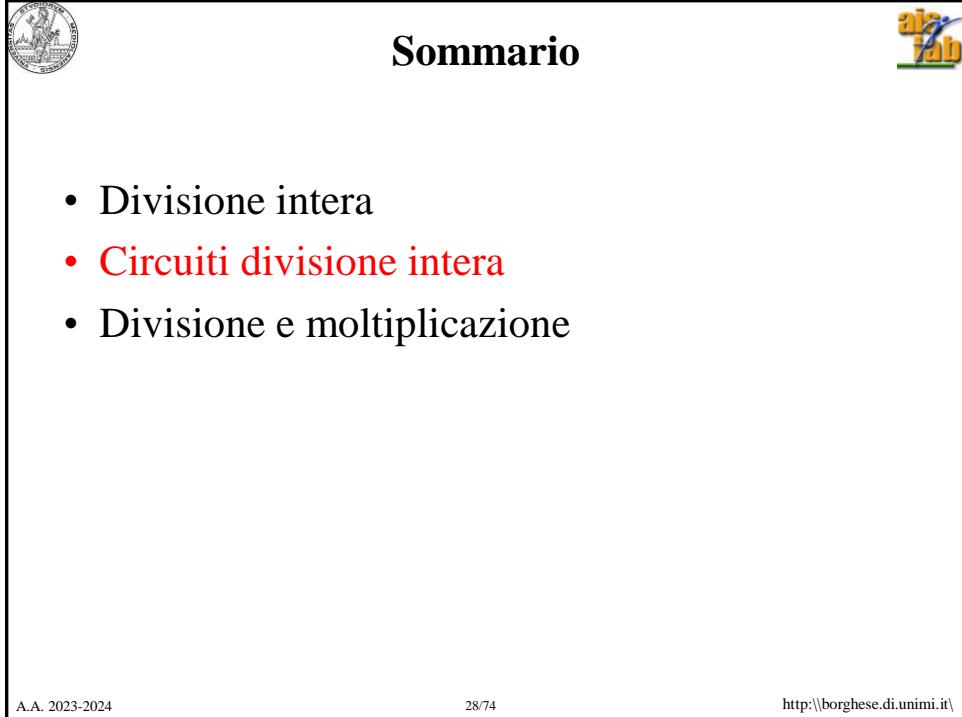
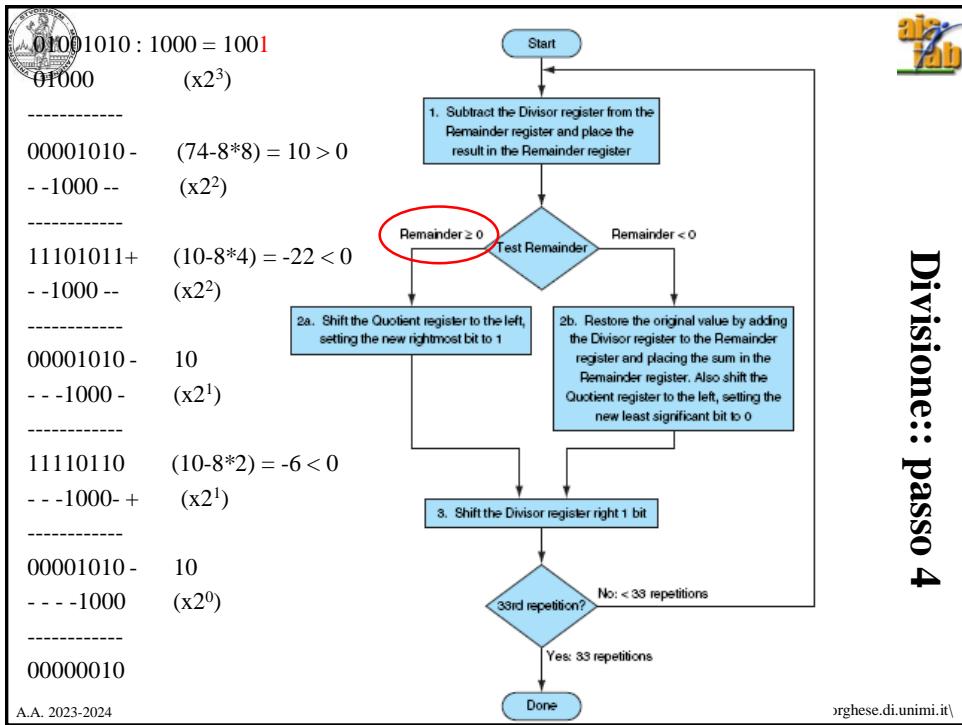
Divisione:: passo 1



Divisione:: passo 2



Divisione:: passo 4

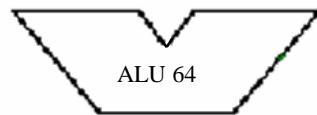




Implementazione circuitale gli stessi attori della moltiplicazione



A - moltiplicando (shift a sx), 2N bit



B – moltiplicatore, N bit

P – Prodotto, 2N bit

Controllo



Il circuito firmware della divisione

Inizializzazione:

- Resto = 0 | Dividendo
- Divisore | 0

moltiplicando diventa divisore

Divisore

Shift right

2N bit



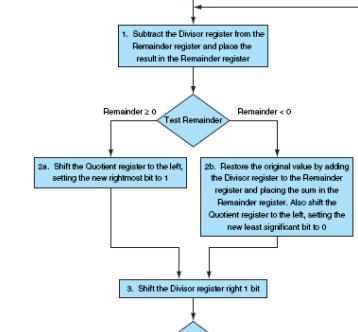
sub, add

2N bit ALU

Remainder

prodotto diventa resto

Write



Quotient

Shift left

N bit

Shift left

Write {0, 1}

moltiplicatore diventa quoziente

Control test

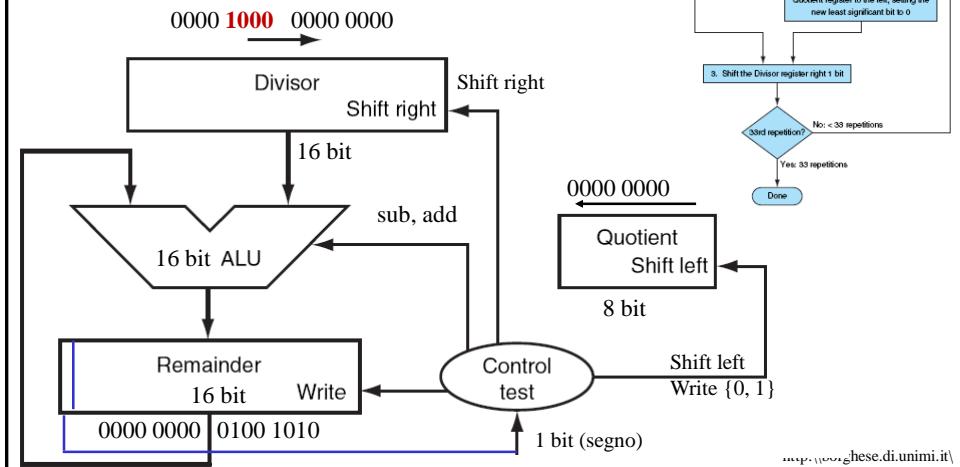
1 bit (segno)



Il circuito firmware della divisione

Inizializzazione:
 -- Resto = 0 | Dividendo
 -- Divisore | 0

$$1001010 : 1000 = \\ 74 : 8 \text{ su } 8 \text{ bit}$$



<http://www.ehese.di.unimi.it/>

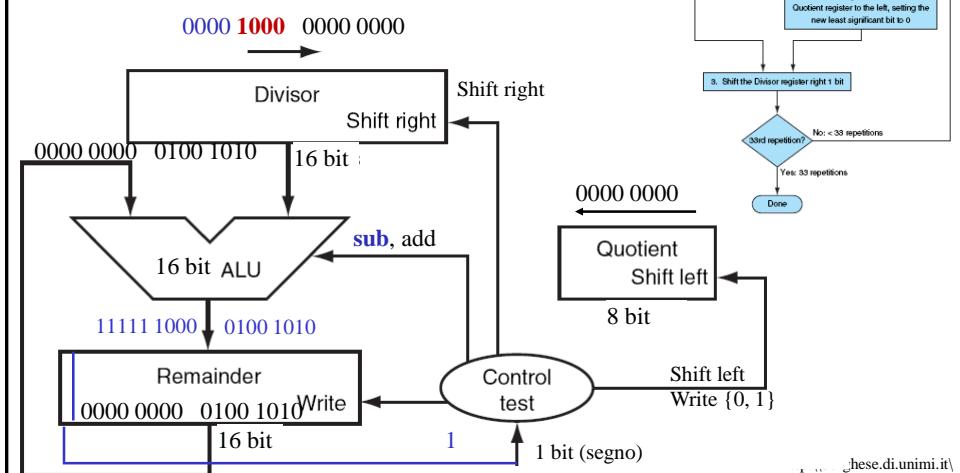


Il circuito firmware della divisione – passo 1a

Inizializzazione:
 -- Resto = 0 | Dividendo
 -- Divisore | 0

$$1001010 : 1000 =$$

$$74 - 8 \times 2^8 = -1974 < 0$$



<http://www.ehese.di.unimi.it/>



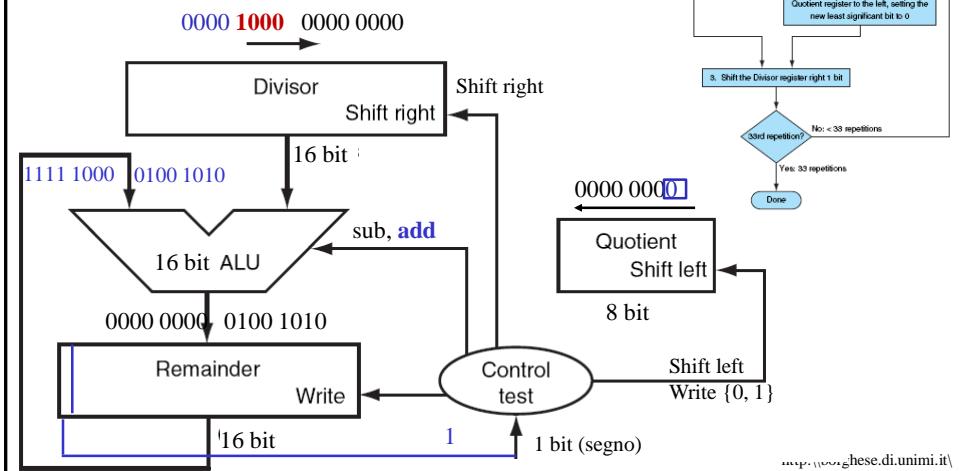
Il circuito firmware della divisione – passo 1b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo

-- Divisore | 0

$$-1974 + 8 \times 2^8 = +74 \text{ -- } Q_k = 0$$



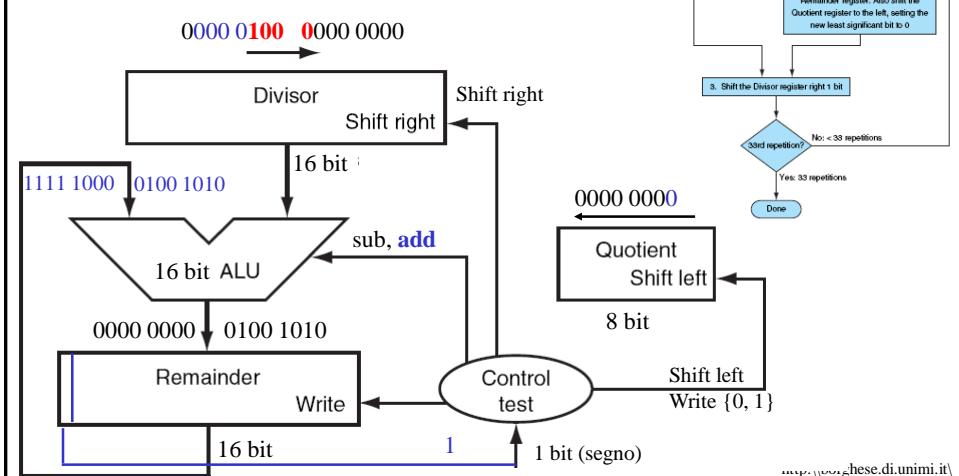
Il circuito firmware della divisione – passo 1c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo

-- Divisore | 0

$$\text{Divisore: } 2048_{10} \text{ ShiftDx 1 } \rightarrow 1024_{10}$$



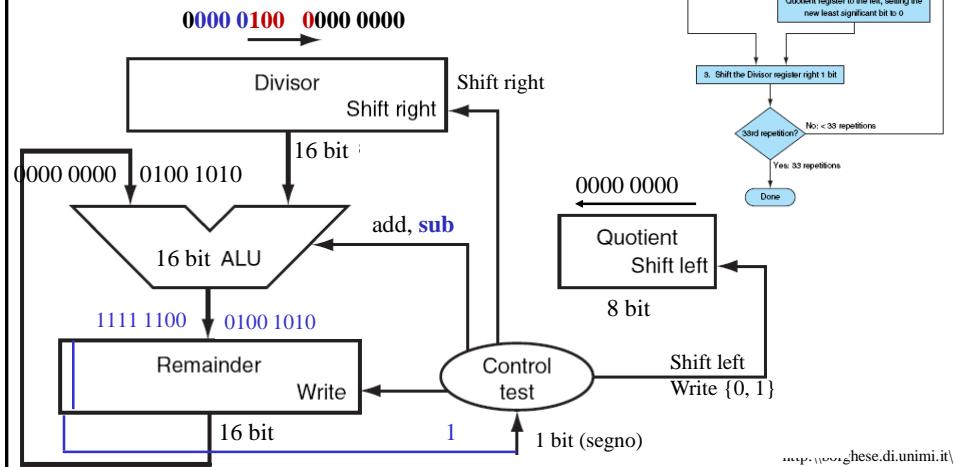


Il circuito firmware della divisione – passo 2a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^7 = -950 < 0$$



<http://www.eheze.di.unimi.it/>

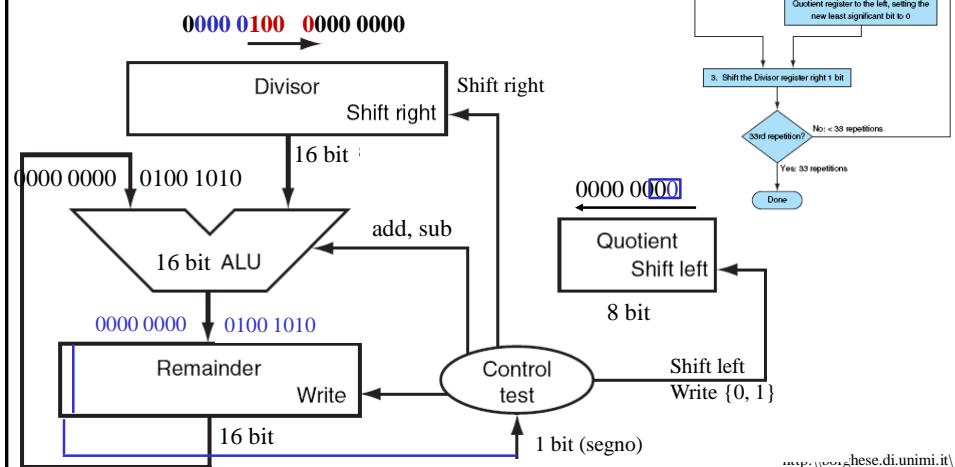


Il circuito firmware della divisione – passo 2b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^7 = -950 < 0 \quad \text{-- } Q_k = 0$$



<http://www.eheze.di.unimi.it/>

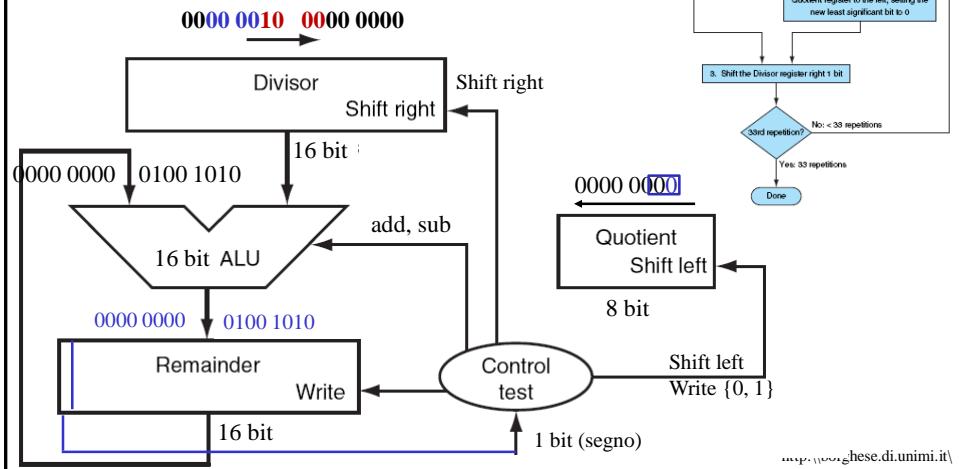


Il circuito firmware della divisione – passo 2c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

Divisore: 1024_{10} ShiftDx 1 -> 512_{10}

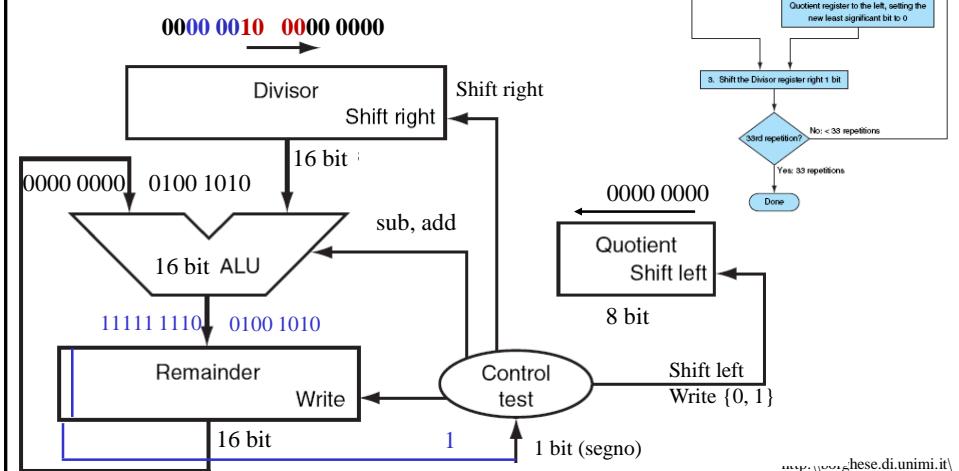


Il circuito firmware della divisione – passo 3a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$74 - 8 \times 2^6 = -438 < 0$



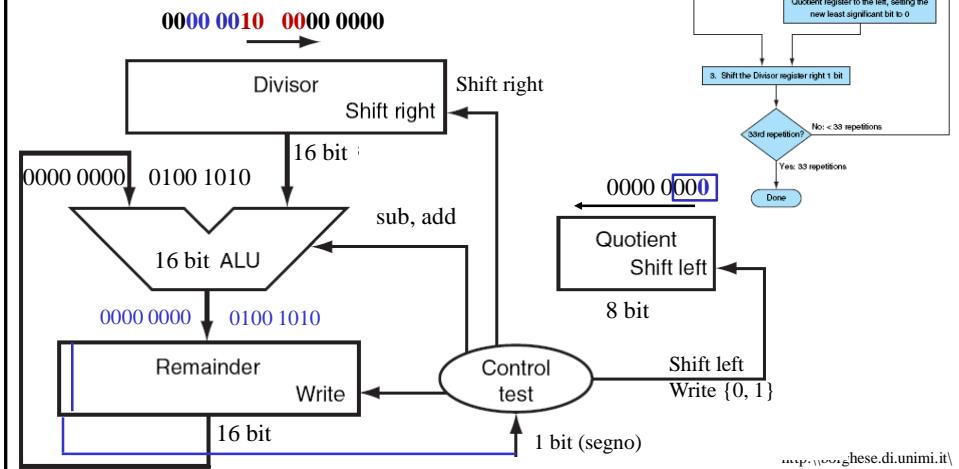


Il circuito firmware della divisione – passo 3b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^6 = -438 < 0 \rightarrow Q_k = 0$$



http://www.ehese.di.unimi.it/

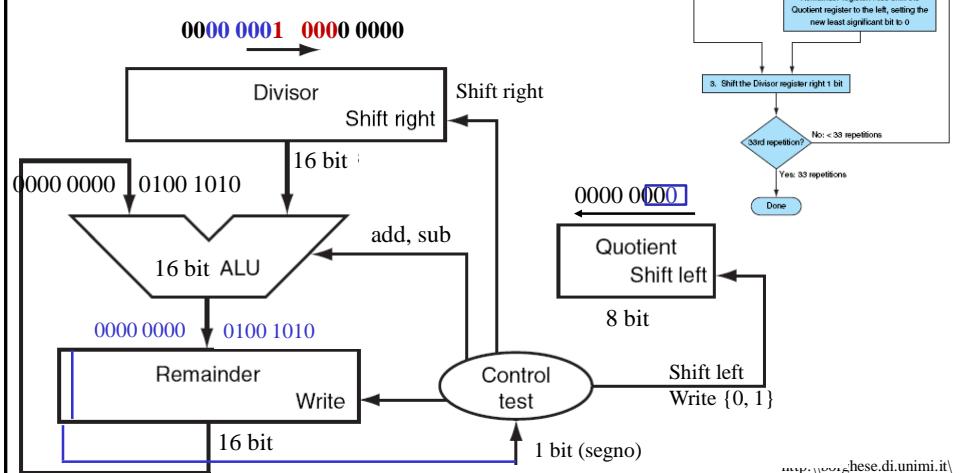


Il circuito firmware della divisione – passo 3c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$\text{Divisore: } 512_{10} \text{ ShiftDx 1} \rightarrow 256_{10}$$



http://www.ehese.di.unimi.it/

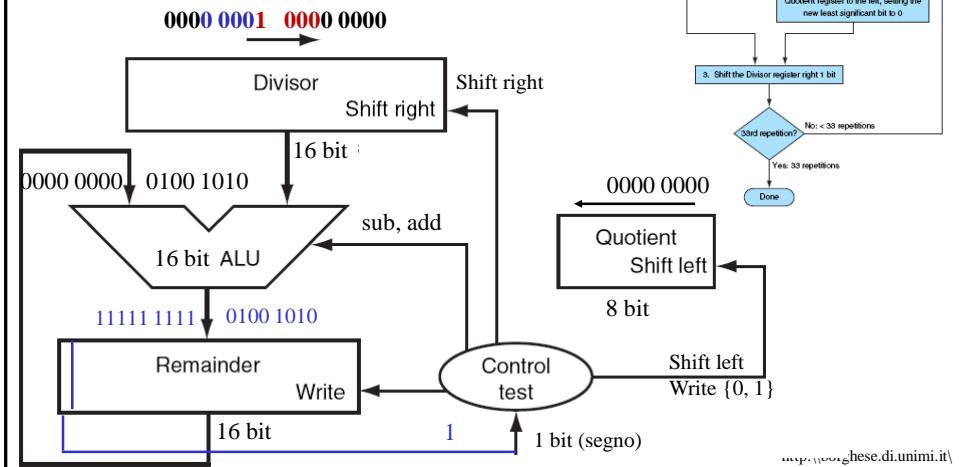


Il circuito firmware della divisione – passo 4a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^5 = -182 < 0$$

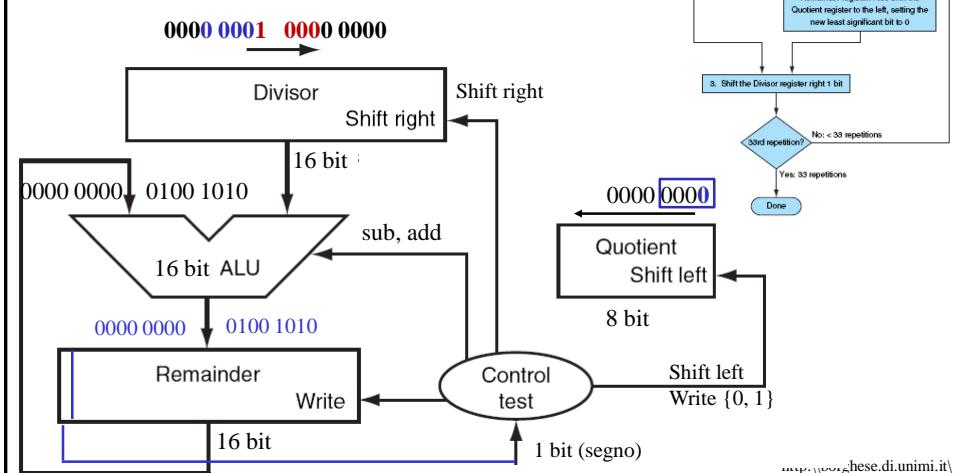


Il circuito firmware della divisione – passo 4b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^5 = -182 < 0 \quad Q_k = 0$$





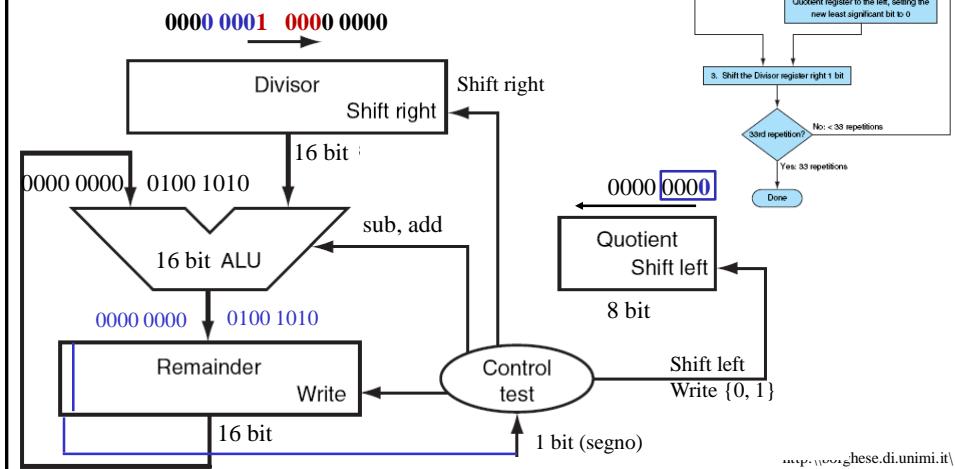
Il circuito firmware della divisione – passo 4c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo

-- Divisore | 0

$$74 - 8 \times 2^5 = -182 < 0 \quad Q_k = 0$$



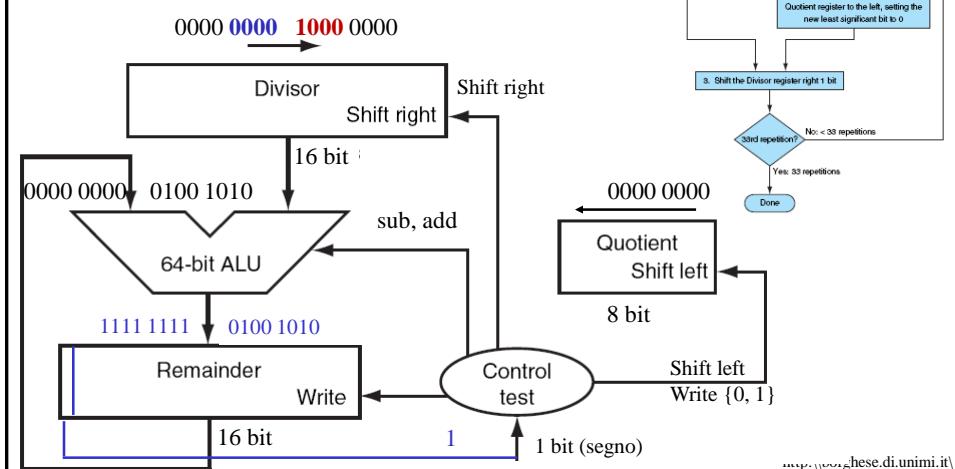
Il circuito firmware della divisione – passo 5a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo

-- Divisore | 0

$$74 - 8 \times 2^4 = -54 < 0$$



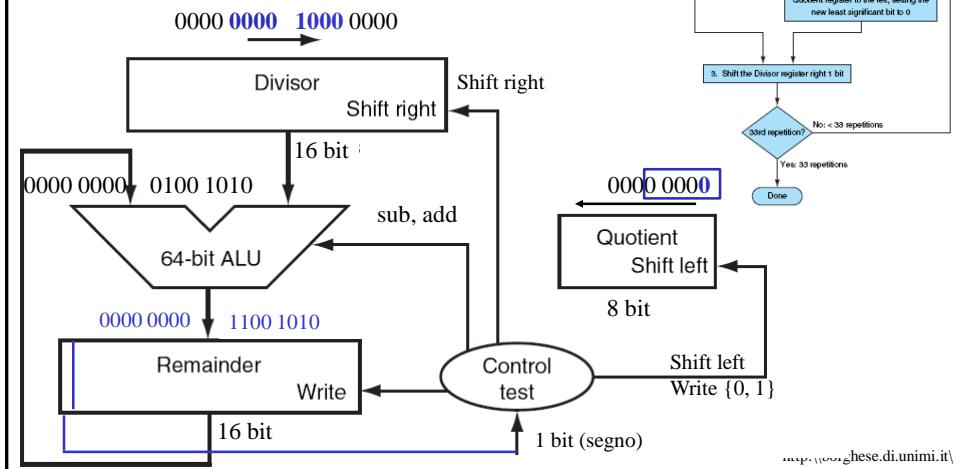


Il circuito firmware della divisione – passo 5b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^4 = -54 < 0$$



<http://www.hese.di.unimi.it/>

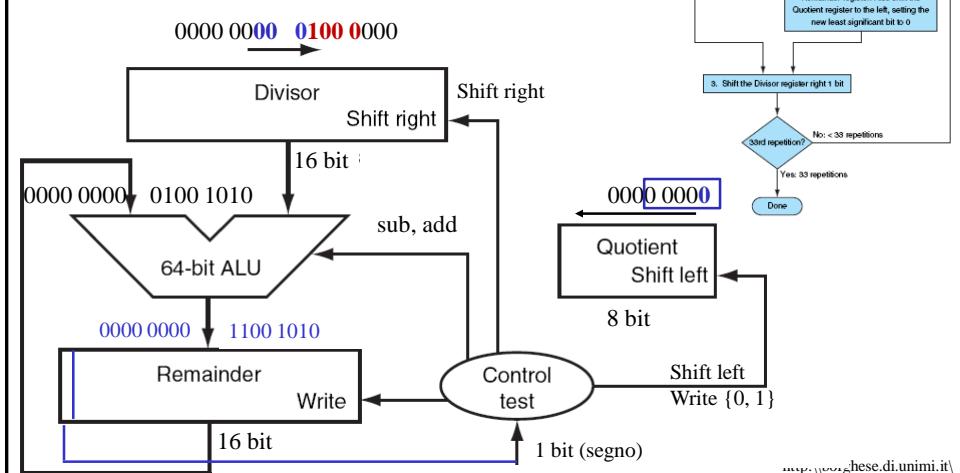


Il circuito firmware della divisione – passo 5c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^4 = -54 < 0$$



<http://www.hese.di.unimi.it/>

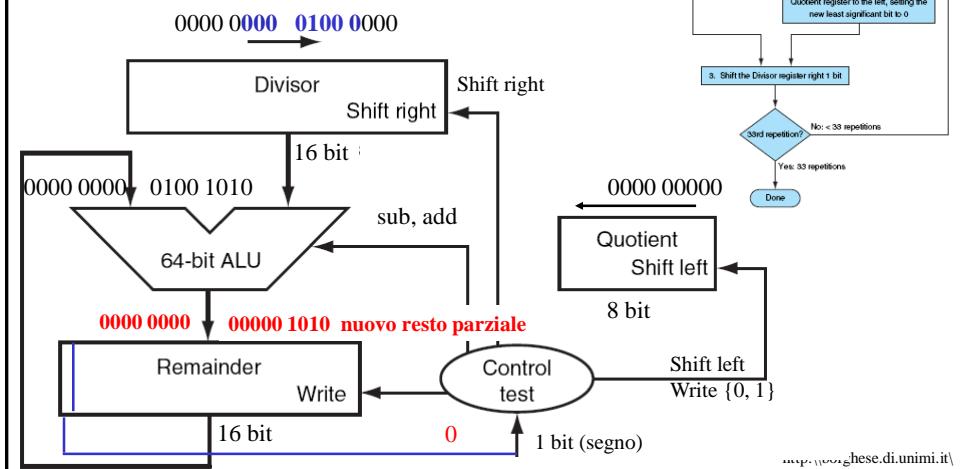


Il circuito firmware della divisione – passo 6a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^3 = +10 \geq 0$$

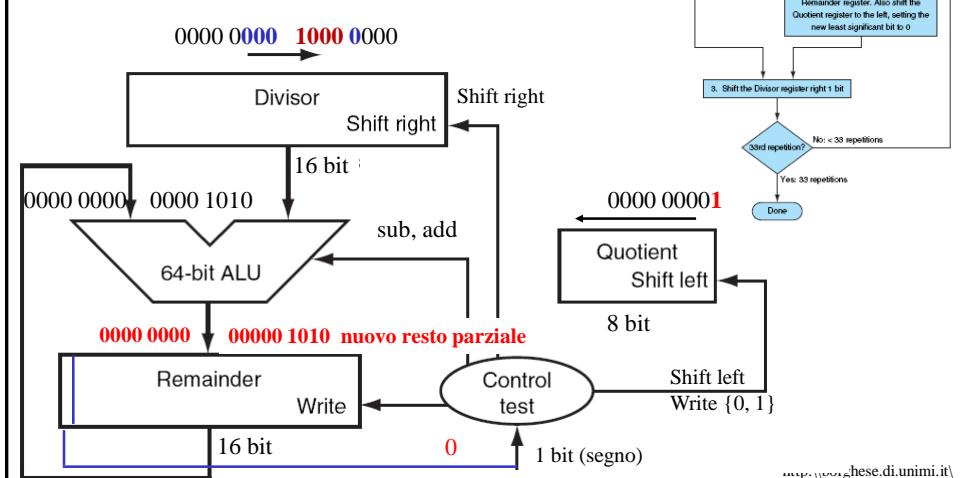


Il circuito firmware della divisione – passo 6b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^3 = +10 \geq 0$$



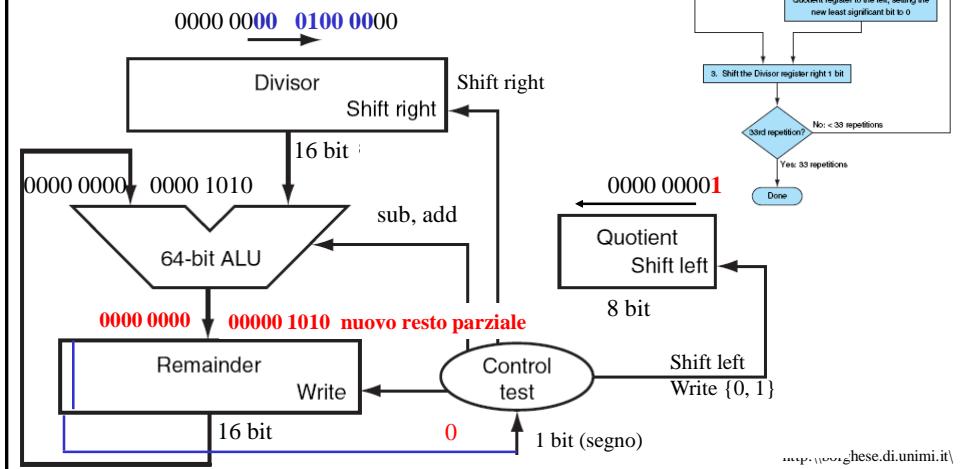


Il circuito firmware della divisione – passo 6c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$74 - 8 \times 2^3 = +10 \geq 0$$

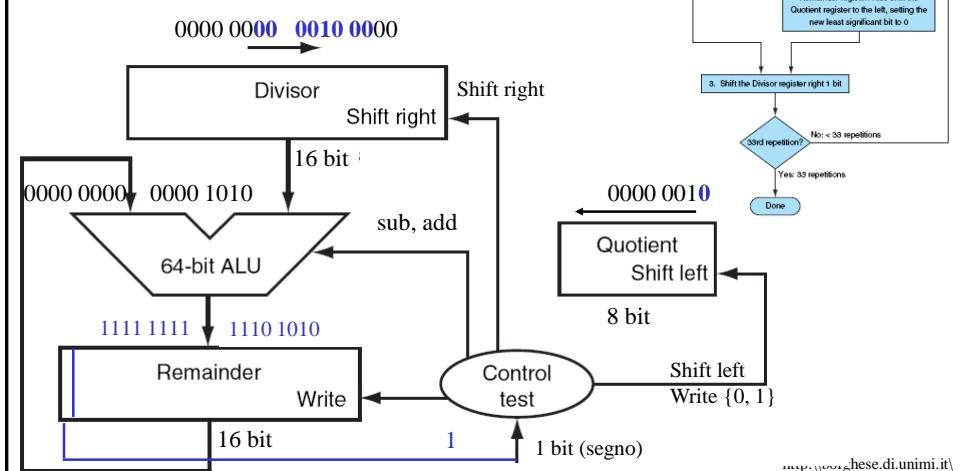


Il circuito firmware della divisione – passo 7a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^2 = -22 < 0$$



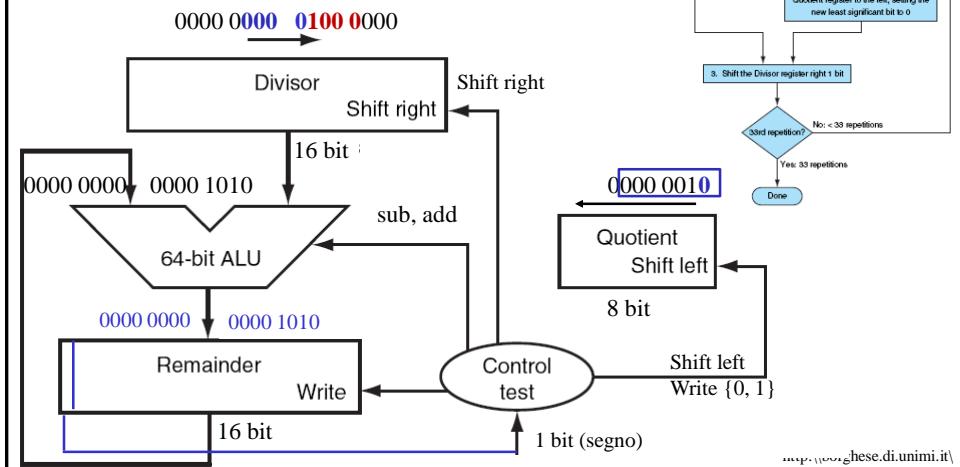


Il circuito firmware della divisione – passo 7b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^2 = -22 < 0$$



<http://www.hese.di.unimi.it/>

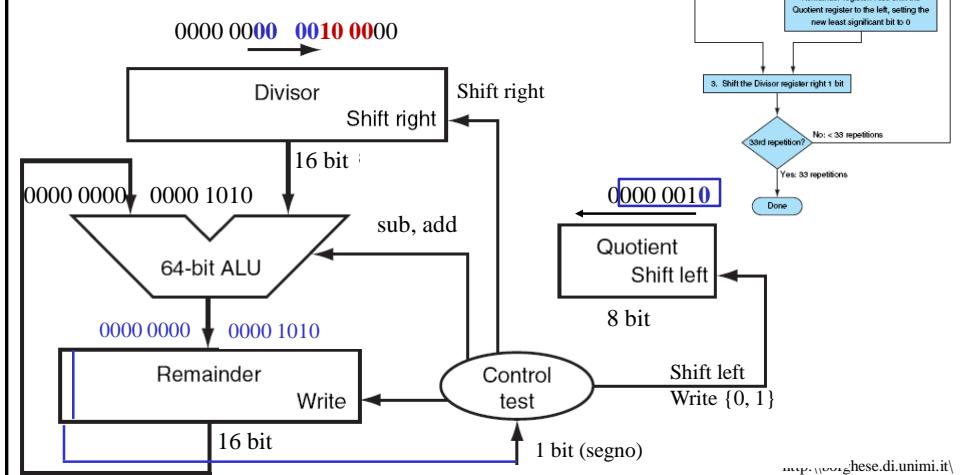


Il circuito firmware della divisione – passo 7c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^2 = -22 < 0$$



<http://www.hese.di.unimi.it/>

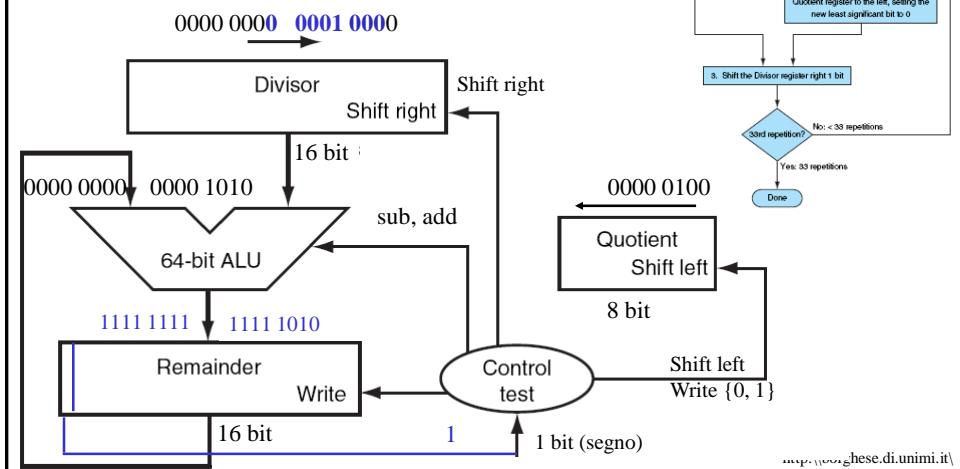


Il circuito firmware della divisione – passo 8a

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^1 = -6 < 0$$

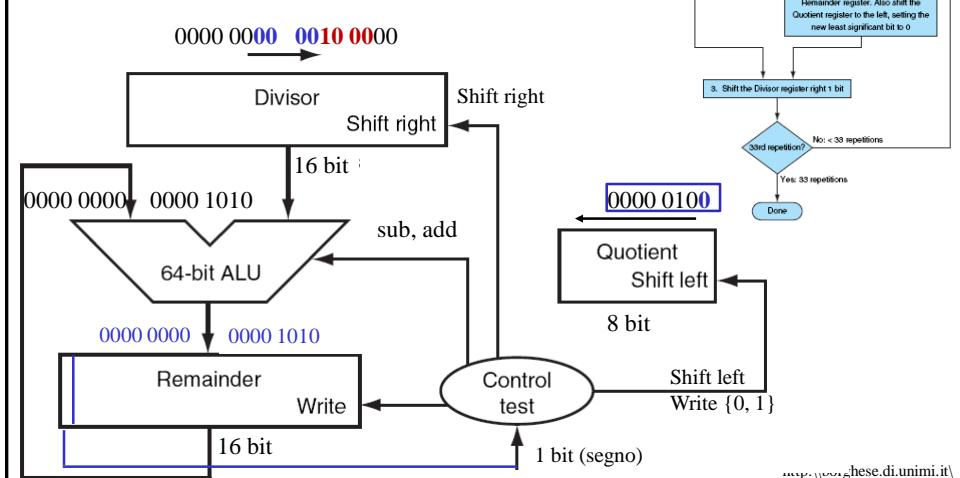


Il circuito firmware della divisione – passo 8b

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^1 = -6 < 0$$



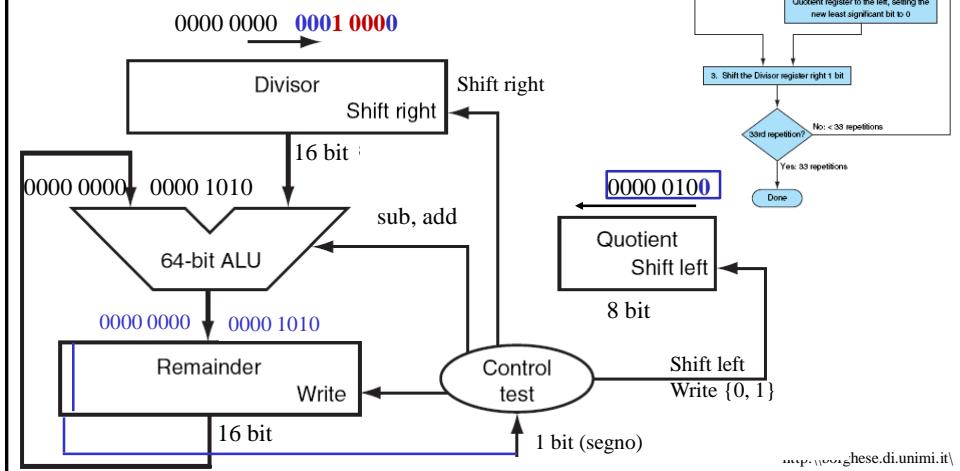


Il circuito firmware della divisione – passo 8c

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^1 = -6 < 0$$

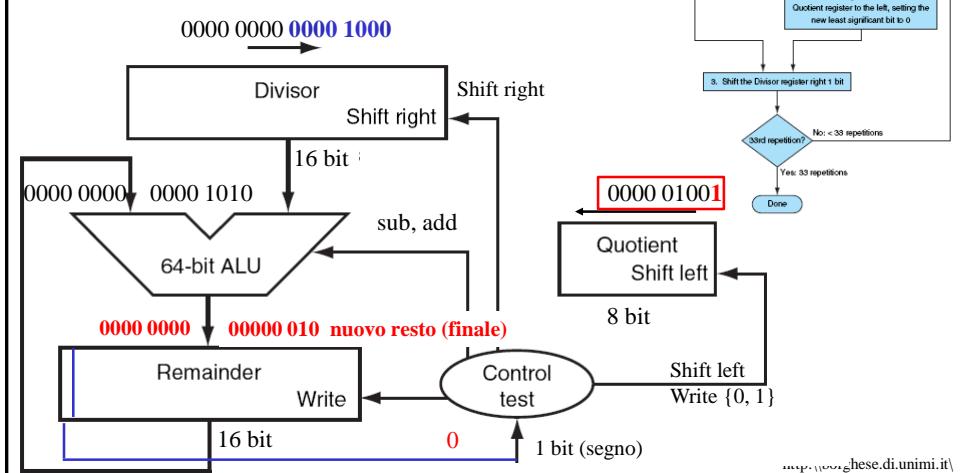


Il circuito firmware della divisione – passo 9

Inizializzazione: $1001010 : 1000 =$

-- Resto = 0 | Dividendo
-- Divisore | 0

$$10 - 8 \times 2^0 = +2 \geq 0$$





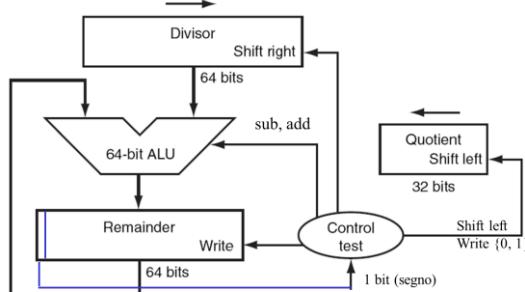
Come ottimizzare il circuito della divisione



Il divisore si sposta verso dx di un bit ad ogni passo e viene sottratto al resto parziale.
Otteniamo lo stesso risultato se **spostiamo il resto parziale a sx di un bit ad ogni passo.**

Inizializziamo il resto come RESTO = 0 | DIVIDENDO.

Ad ogni passo sposto il dividendo alias resto parziale di una posizione a sx ed inserisco un bit del quozi



A.A. 2023-2024

<http://borghese.di.unimi.it/>



Esempio - 1



Divisione decimale fra i numeri a = 7 e b = 2 su 4 bit. a : b = ?

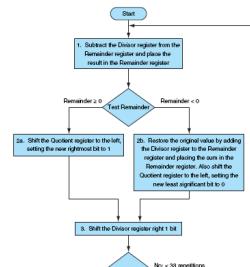
Inizializzo il divisore alla sinistra delle quattro cifre significative. La prima cifra del quoziante sarà sempre 0.

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem - Div	0000	0010 0000	①10 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0010 0000	000 0111
	3: Shift Div right	0000	0001 0000	0000 0111

$$\text{Divisore} = 1 \times 2^4 \times 2^1 = 32$$

$$\text{Resto} = 7$$

$$7 - 32 = -25 < 0$$



A.A. 2023-2024

58/74



Esempio - 2



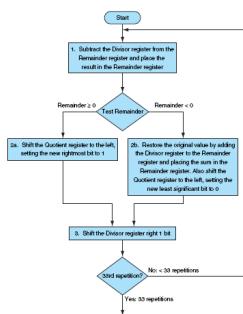
Divisione decimale fra i numeri $a = 7$ e $b = 2$ $a : b = ?$

Iteration	Step	Quotient	Divisor	Reminder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	110 0111
	2b: Rem < 0 $\Rightarrow +\text{Div}$, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	110 0111
	2b: Rem < 0 $\Rightarrow +\text{Div}$, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111

$$\text{Divisore} = 1 \times 2^3 \times 2 = 16$$

$$\text{Resto} = 7$$

$$7 - 16 = -9 < 0$$



Effetto dello spostamento relativo - I



Iteration	Step	Quotient	Divisor	Reminder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	110 0111
	2b: Rem < 0 $\Rightarrow +\text{Div}$, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111

Iterazione	Passo	Quoziente	Divisore	Dividendo/Resto
0	Valori iniziali	0000	0010	0 0000 0111
1	Resto = Resto - div	0000	0010	1 1110 0111
	Resto<0 ->Resto+=div;	0000	0010	0 0000 0111
	Resto << 1 - Q ₀ =0	0000	0010	0 0000 1110

$$0111 : 10 \Rightarrow$$

$$\begin{array}{r} 0111 \\ - 0010 \\ \hline \end{array} =$$



Effetto dello spostamento relativo - II



Iteration	Step	Quotient	Divisor	Reminder
2	1: Rem = Rem - Div	0000	0001 0000	0000 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0001 0000	1111 0111
	3: Shift Div right	0000	0001 0000	0000 0111

Iterazione	Passo	Quoziente	Divisore	Dividendo/Resto
2	Resto = Resto - div	0000	0010	0 0000 1110
	Resto<0 \rightarrow Resto+=div;	0000	0010	1 1110 1110
	Resto << 1 - Q0=0	0000	0010	0 0000 1110

0111 : 10 \Rightarrow

0111 -

0010 =



Effetto dello spostamento relative - III



Iteration	Step	Quotient	Divisor	Reminder
3	1: Rem = Rem - Div	0000	0000 1000	1111 0111
	2b: Rem < 0 \Rightarrow +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111

Iterazione	Passo	Quoziente	Divisore	Dividendo/Resto
3	Resto = Resto - div	0000	0010	0 0001 1100
	Resto<0 \rightarrow Resto+=div;	0000	0010	1 1111 1100
	Resto << 1 - Q0=0	0000	0010	0 0001 1100

0111 : 10 \Rightarrow

0111 -

0010 =



Effetto dello spostamento relativo - IV



Iteration	Step	Quotient	Divisor	Reminder
				0000 0111
4	1: Rem = Rem - Div	0000	0000 0100	0000 0011
	2a: Rem $\geq 0 \Rightarrow$ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011

Iterazione	Passo	Quoziente	Divisore	Dividendo/Resto
		0000	0010	0 0011 1000
4	Resto = Resto - div	0000	0010	0 0001 1000
	Resto > 0	0000	0010	0 0001 1000
	Resto $<< 1 - Q_0=1$	0001	0010	0 0011 0000

0111 : 10 \Rightarrow 0111 –
0010 =



Effetto dello spostamento relativo - V



Iteration	Step	Quotient	Divisor	Reminder
				0000 0011
5	1: Rem = Rem - Div	0001	0000 0010	0000 0001
	2a: Rem $\geq 0 \Rightarrow$ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Iterazione	Passo	Quoziente	Divisore	Dividendo/Resto
		0001	0010	0 0011 0000
5	Resto = Resto - div	0001	0010	0 0001 0000
	Resto > 0	0000	0010	0 0001 0000
	Resto $<< 1 - Q_0=1$	0011	0010	0 0010 0000

0111 : 10 \Rightarrow 0011 –
0010 =

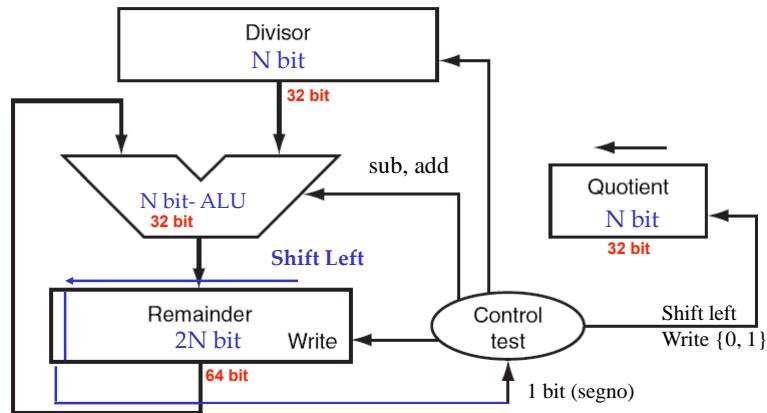
Il resto finale va fatto scorrere a dx di 1 posizione



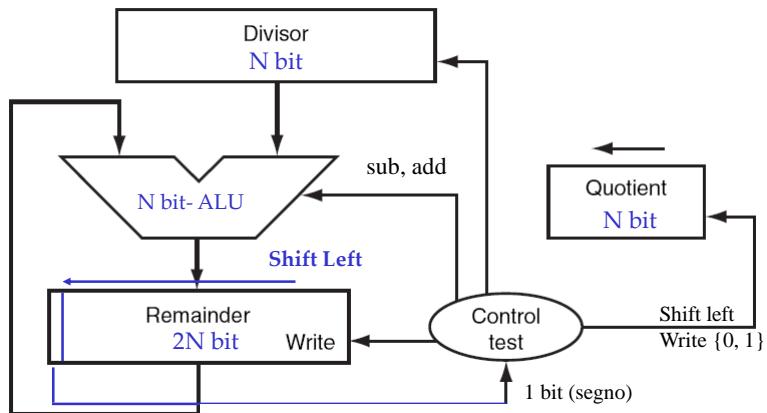
Il circuito firmware con un'ottimizzazione



Inizializzazione: Resto = 0 | Dividendo



Razionale per un'ulteriore ottimizzazione



Il quoziente viene riempito un bit alla volta da dx a sx
Il dividendo viene spostato da dx a sx di una posizione

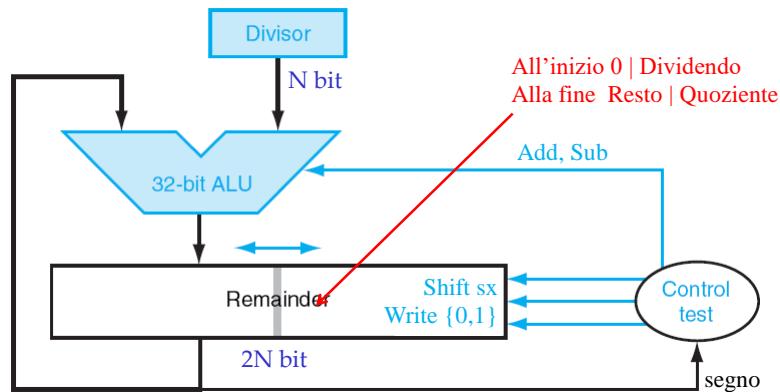


Il circuito firmware ottimizzato della divisione

identico a quello per la moltiplicazione ottimizzata



Inizializzazione: Resto = 0 | Dividendo

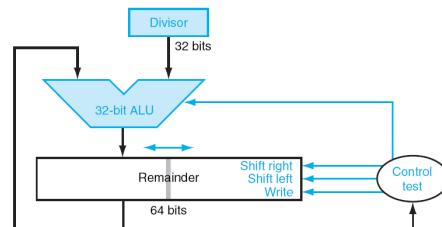


Effetto sul circuito

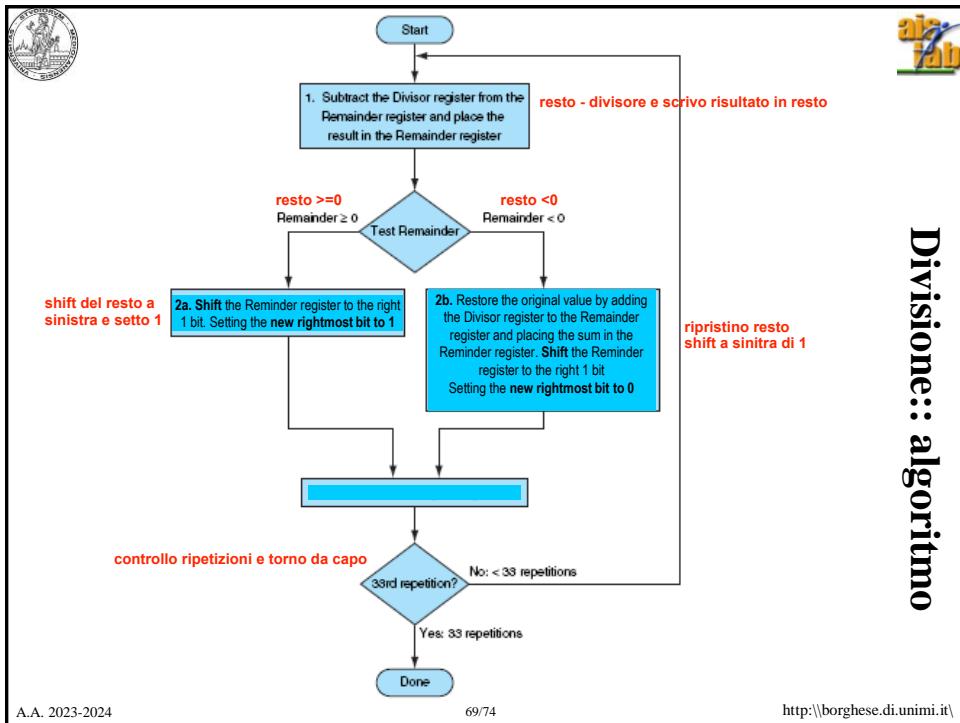


Iteration	Step	Quotient	Divisor	Reminder
2	1: Rem = Rem - Div	0000	001 0000	0000 0111
	2b: Rem < 0 => +Div, sll Q, Q0 = 0	0000	001 0000	1111 0111
	3: Shift Div right	0000	001 0000	0000 0111

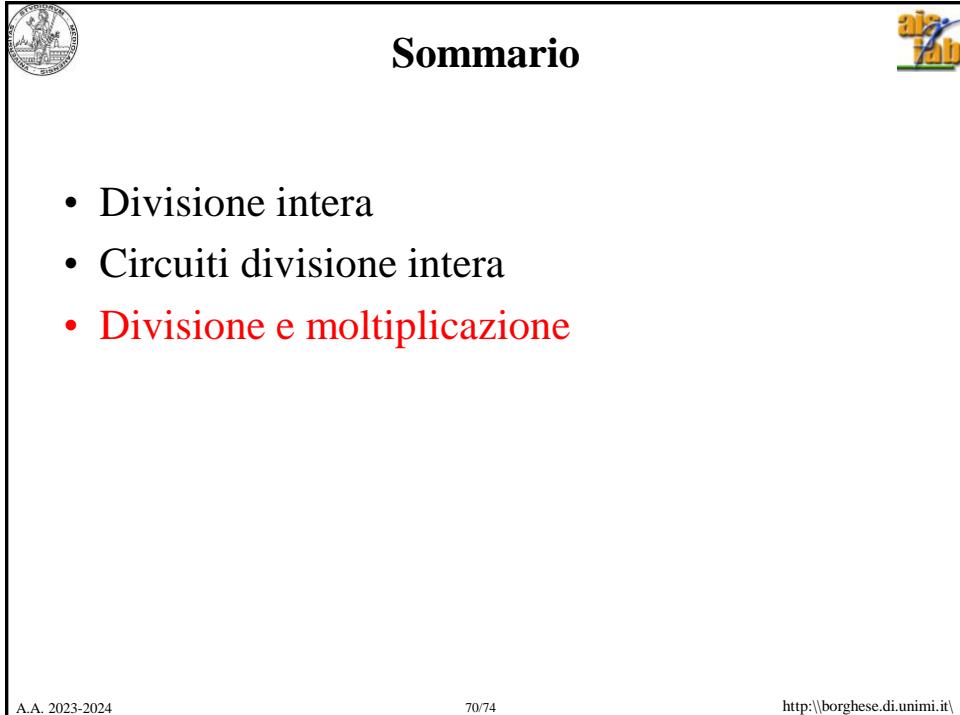
Iterazione	Passo	Divisore	Dividendo/Resto/Quoziente
2	Resto = Resto - div	0010	0 0000 1110
	Resto<0 ->Resto+=div;	0010	1 1110 1110
	Resto << 1 - Q ₀ =0	0010	0 0000 1110



algoritmo semplificato

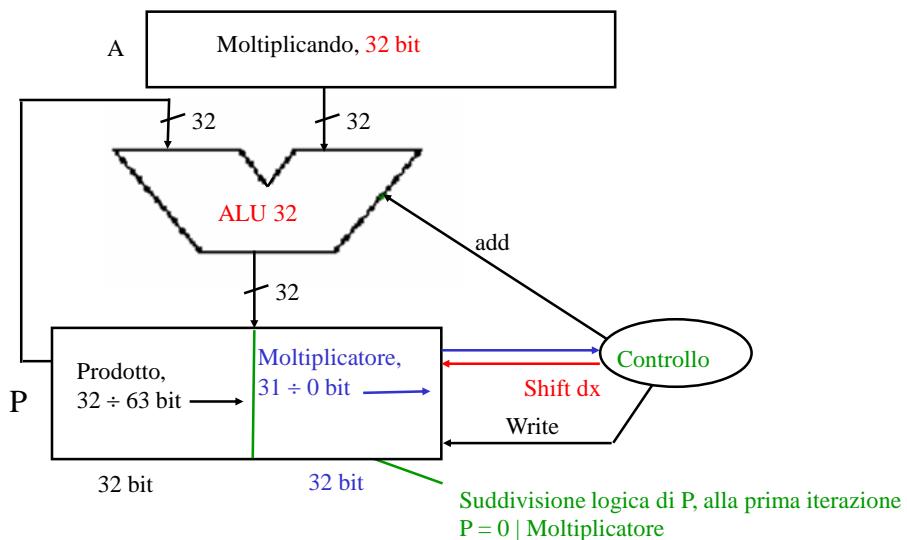


Divisione:: algoritmo





Circuito ottimizzato della moltiplicazione (su 32 bit)

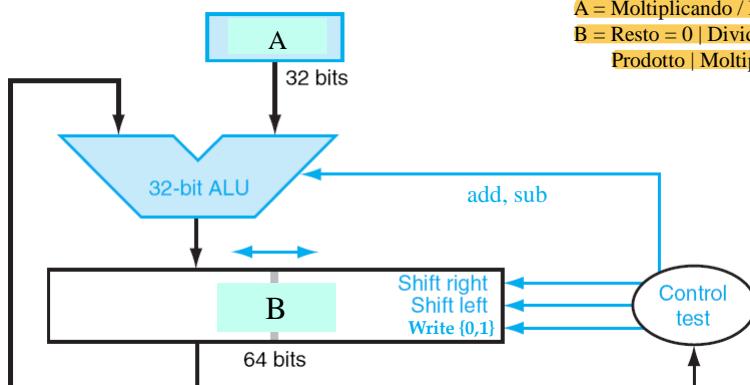


Un unico circuito per moltiplicazione e divisione



Inizializzazione:

A = Moltiplicando / Divisore
B = Resto = 0 | Dividendo /
 Prodotto | Moltiplicatore





Segno divisione e moltiplicazione



Moltiplicazione e divisione di numeri positivi -> viene aggiunto il segno alla fine.

Moltiplicazione: XOR dei bit di segno di moltiplicando e moltiplicatore

Se XOR positivo -> prodotto negativo

Divisione: $a : b = q + r$

XOR dei bit di segno di dividendo e divisore. Se XOR = 1 -> quoziante negativo.

E il resto?

$$b*q = a - r$$

$$\begin{array}{ll} a > 0 \ b > 0 \rightarrow q > 0, r \geq 0 & 7 : (+3) = +2 +1 \\ a > 0 \ b < 0 \rightarrow q < 0, r \geq 0 & 7 : (-3) = -2 +1 \end{array}$$

$$\begin{array}{ll} a < 0 \ b > 0 \rightarrow q < 0, r \leq 0 & (-7) : (+3) = -2 -1 \\ a < 0 \ b < 0 \rightarrow q > 0, r \leq 0 & (-7) : (-3) = +2 -1 \end{array}$$

Il dividendo e il resto hanno sempre lo stesso segno



Sommario



- Divisione intera
- Circuiti divisione intera
- Divisione e moltiplicazione



UC firmware moltiplicazione Floating pointer adder

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

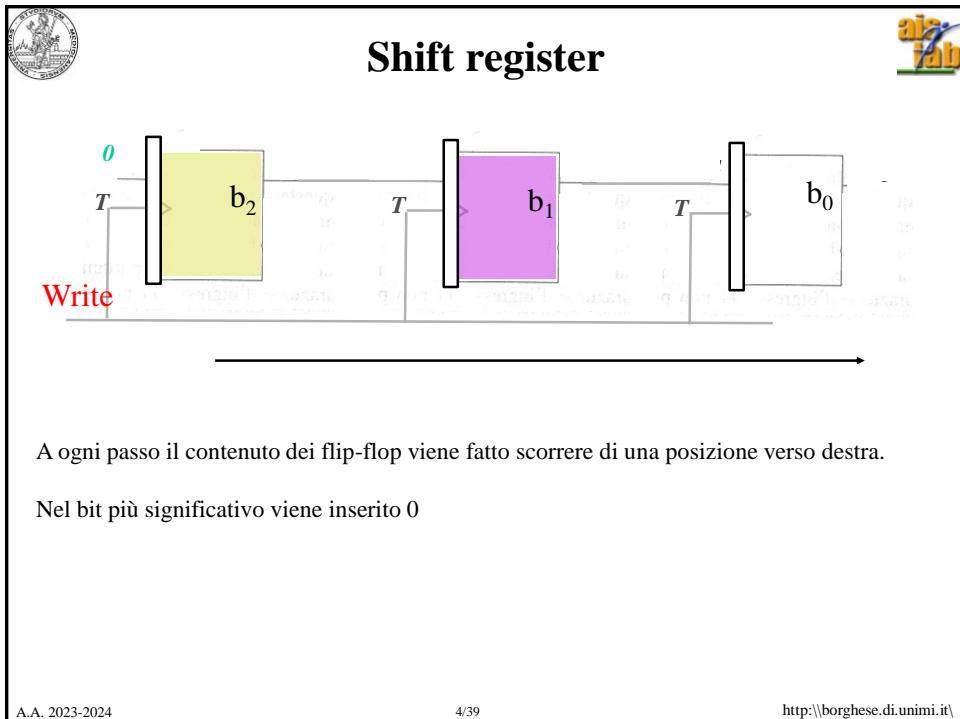
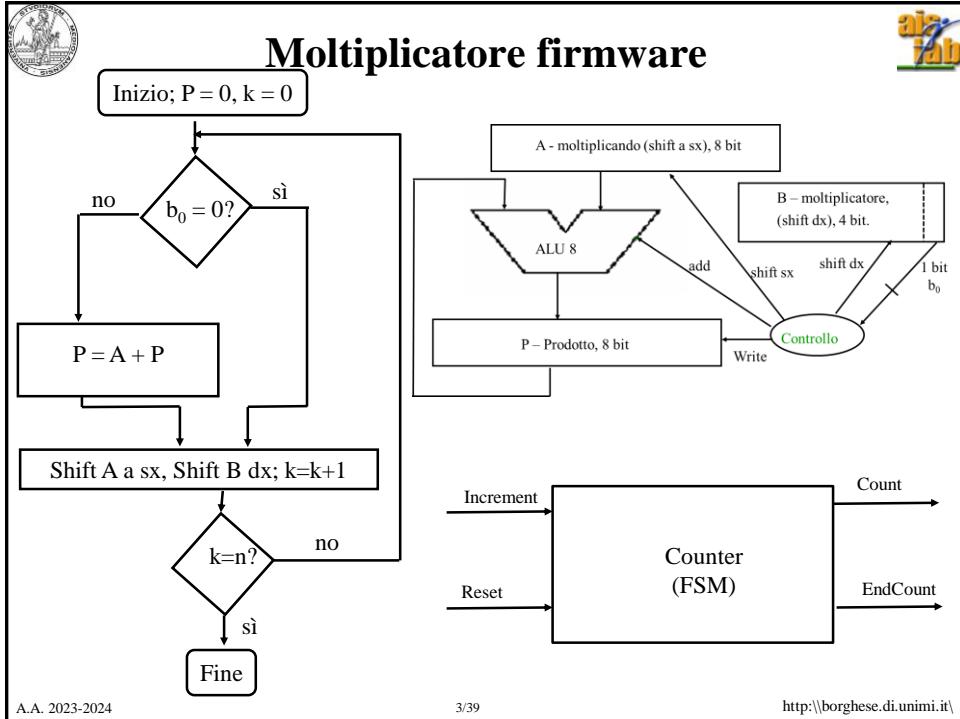
Università degli Studi di Milano
Riferimenti sul Patterson, 6a Ed.: 3.4, 3.5, 4.2

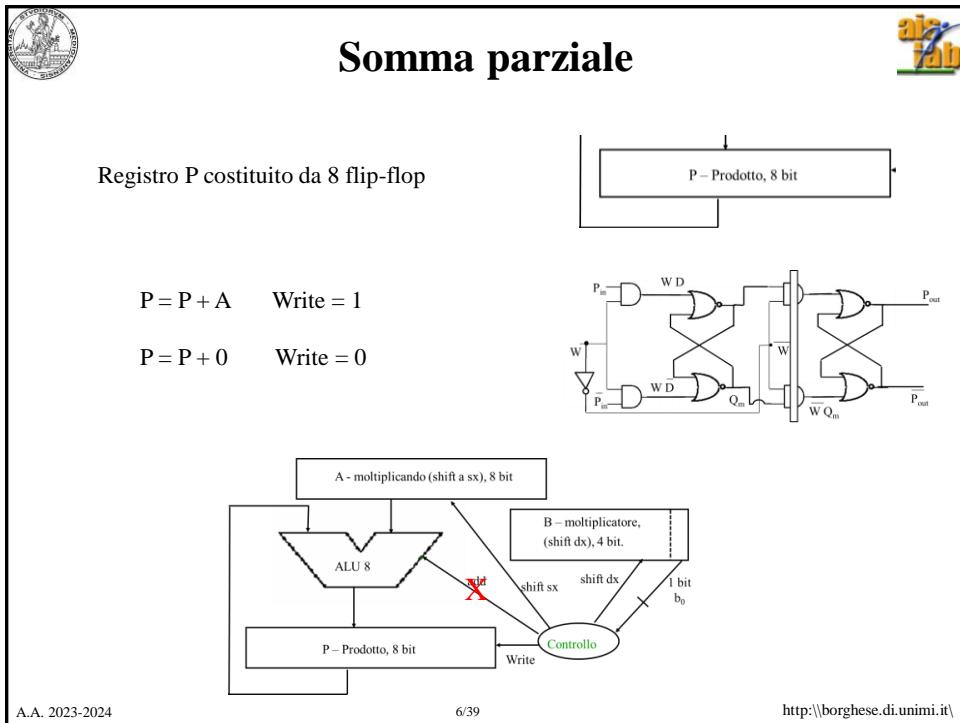
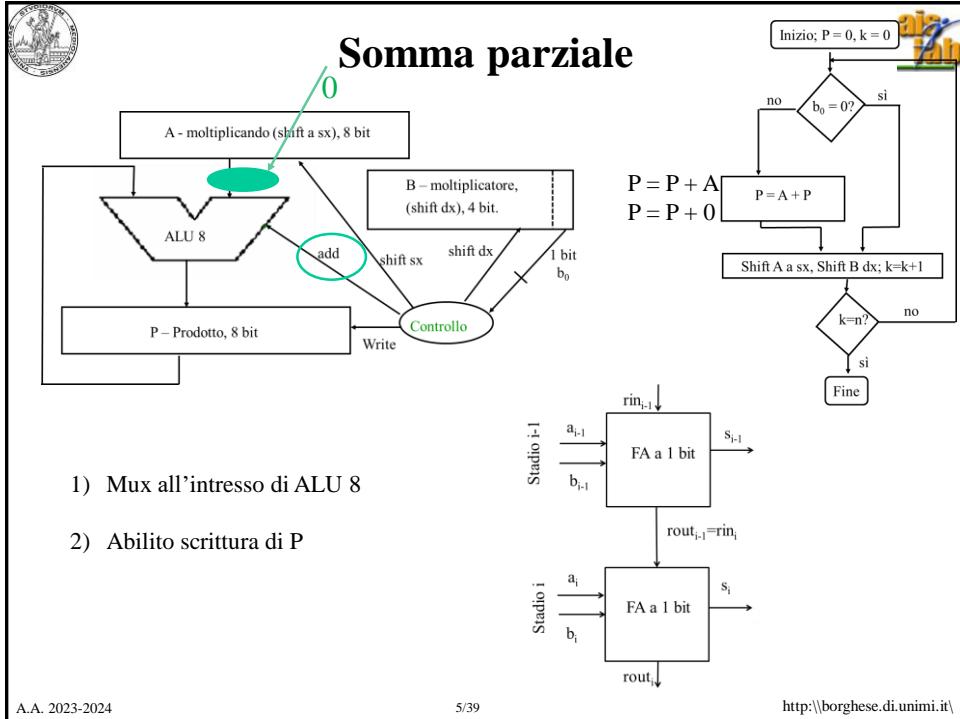


Sommario



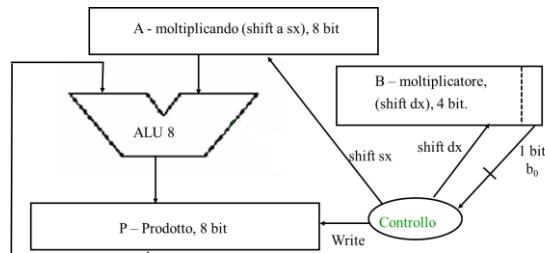
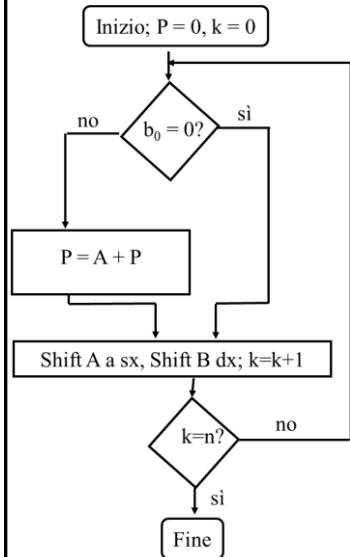
- Unità di controllo del firmware
- Somma in virgola mobile







Operazioni elementari



Da eseguire in sequenza:

- Somma (write P)
- Shift sx A; Shift dx B



STG – S₀

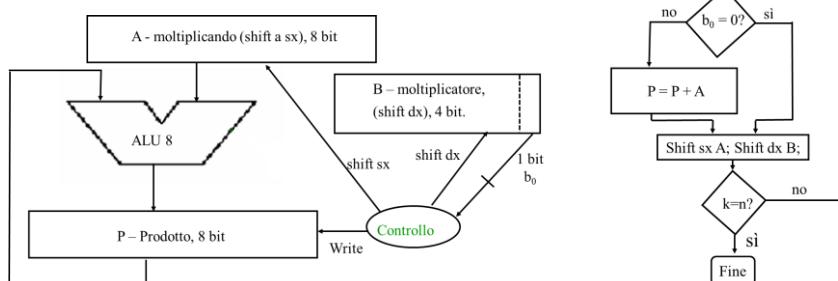


Macchina a stati finiti:

Write [0 | A], Write B
ResetCount, Clear P

S₀ / InitCount

{X} = {initCount,
{I} =
{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
X₀ = Stato iniziale = InitCount





STG – S₁



Macchina a stati finiti:

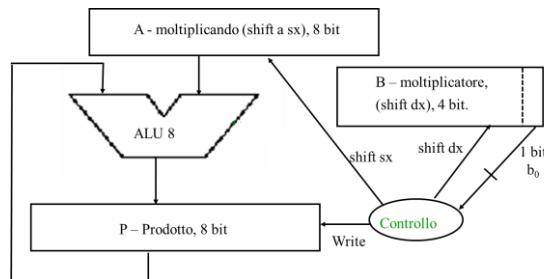
Write [0 | A], Write B
ResetCount, Clear P

{X} = {initCount, Activate},

{I} = {

{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
IncCount,

X₀ = Stato iniziale = InitCount



A.A. 2023-2024

9/49

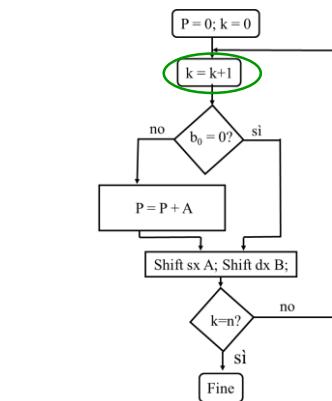
<http://borgheze.di.unimi.it/>



S₀ / InitCount

S₁ / Activate

IncCount



STG – S₂



Macchina a stati finiti:

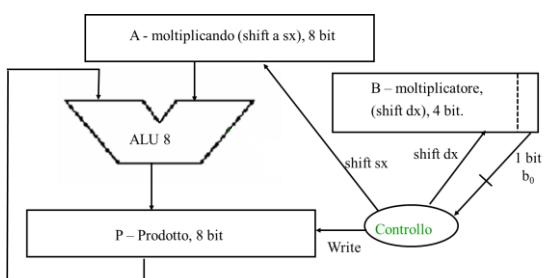
Write [0 | A], Write B
ResetCount, Clear P

{X} = {initCount, Activate, Add},

{I} = {b₀},

{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
IncCount, Write P,

X₀ = Stato iniziale = InitCount



A.A. 2023-2024

10/49

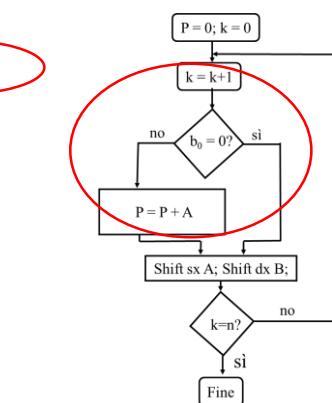
<http://borgheze.di.unimi.it/>

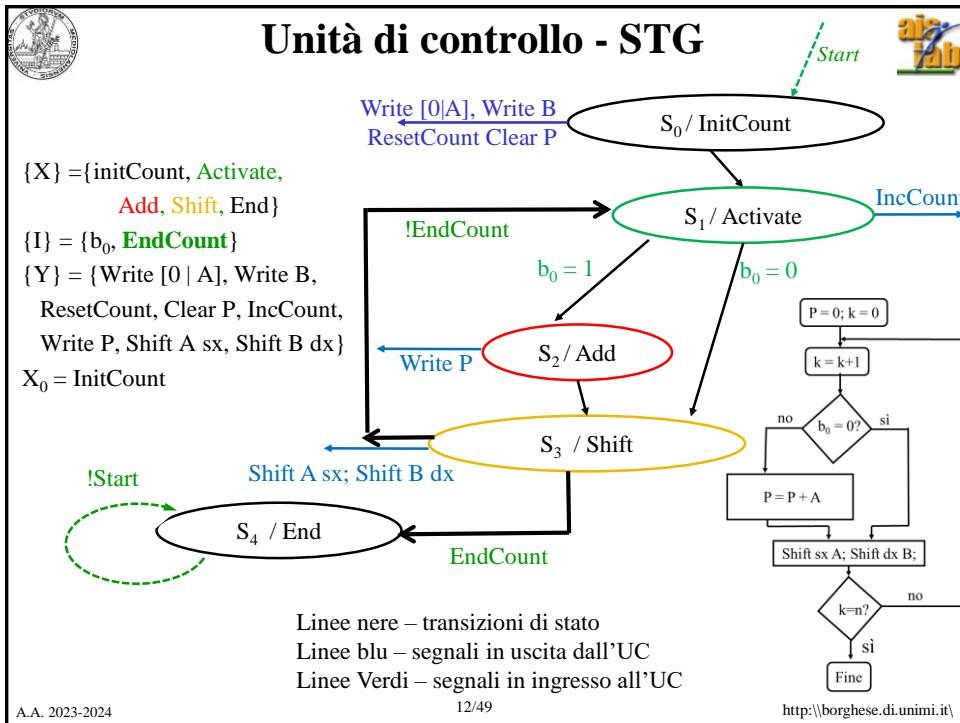
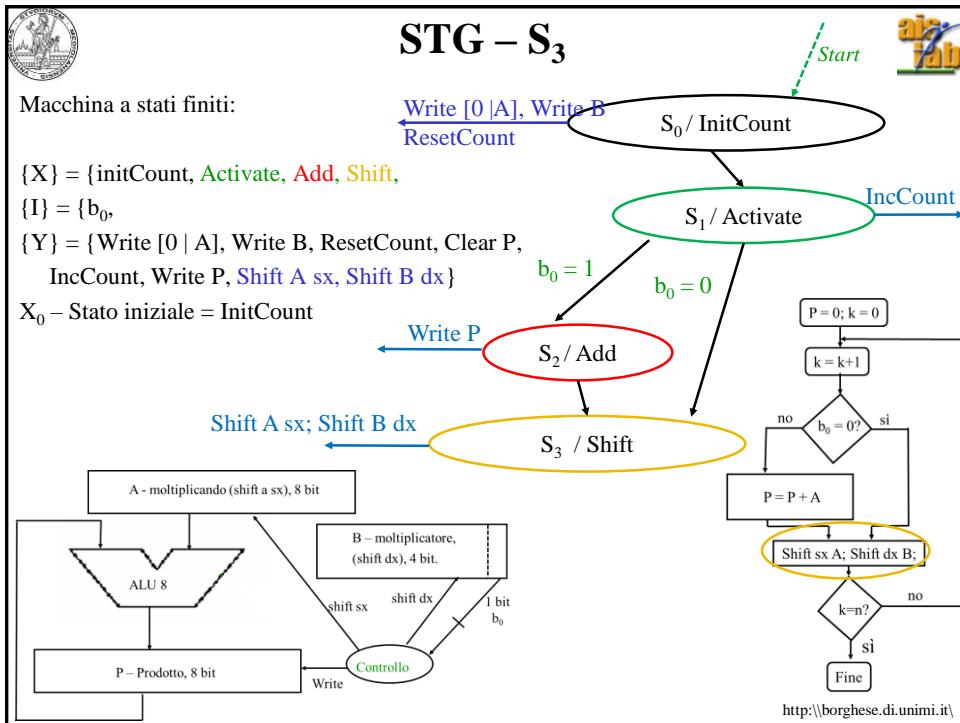


S₀ / InitCount

S₁ / Activate

IncCount







Macchina di Huffman

$\{X\} = \{\text{initCount, Activate, }$

$\text{Add, Shift, End}\}$

$\{I\} = \{b_0, \text{EndCount}\}$

$\{Y\} = \{\text{Write } [0 | A], \text{ Write B, }$
 $\text{ResetCount, Clear P, IncCount, }$
 $\text{Write P, Shift A sx, Shift B dx}\}$

$X_0 = \text{InitCount}$

$M = 2$ - 2 input binary (yes/no)

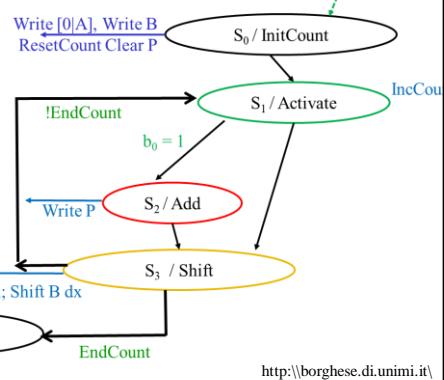
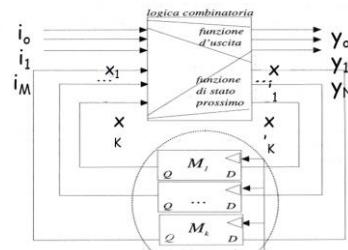
$N = 8$ - 8 uscite binarie

(yes/no - write / no write)

$K = 3$ - 5 stati

NB Sono riportati solo i segnali in uscita (Y) quando vengono asseriti (=1)

A.A. 2023-2024



13/49

<http://borghese.di.unimi.it/>



Unità di controllo - STT



$\{X\} = \{\text{initCount, Activate, Add, Shift, End}\}$

$\{I\} = \{b_0, \text{EndCount}\}$

$\{Y\} = \{\text{Write } [0 | A], \text{ Write B, Clear P, ResetCount, }$
 $\text{IncCount, Write P, Shift A sx, Shift B dx}\}$

$X_0 = \text{InitCount}$

$f(X, I)$ – Funzione stato prossimo

$g(X)$ – Funzione di uscita



13/49

Uscita

	!EndCount $b_0 = 0$	EndCount $b_0 = 0$!EndCount $b_0 = 1$	EndCount $b_0 = 1$	Uscita
InitCount	Activate	Activate	Activate	Activate	Clear P, Write A, Write B, Reset counter, Clear P
Activate	Shift	Shift	Add	Add	Inc counter
Add	Shift	Shift	Shift	Shift	Write P
Shift	Activate	End	Activate	End	Shift A sx, Shift B dx
End	End	End	End	End	

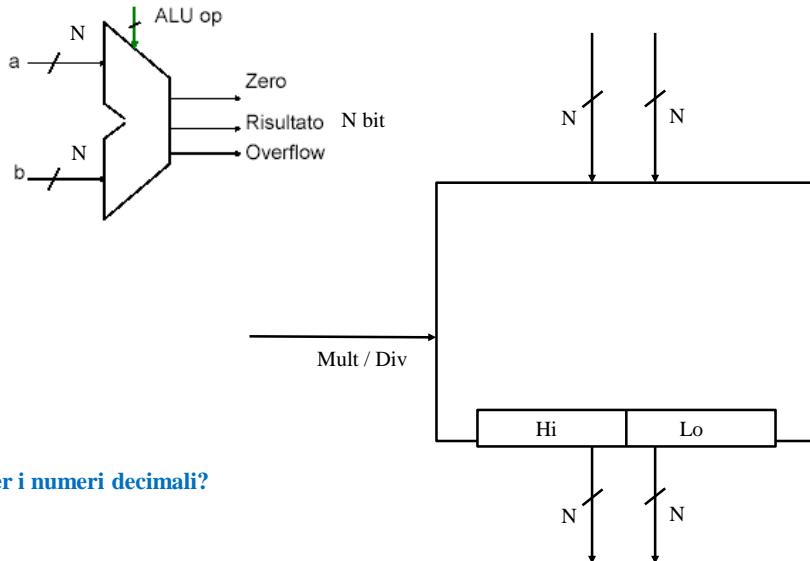
A.A. 2023-2024

14/49

<http://borghese.di.unimi.it/>



Circuiti operazioni tra numeri interi



A.A. 2023-2024

15/39

<http://borghese.di.unimi.it/>



Sommario



- Unità di controllo del firmware
- Somma in virgola mobile

A.A. 2023-2024

16/39

<http://borghese.di.unimi.it/>



Codifica in virgola mobile Standard IEEE 754 (1980)

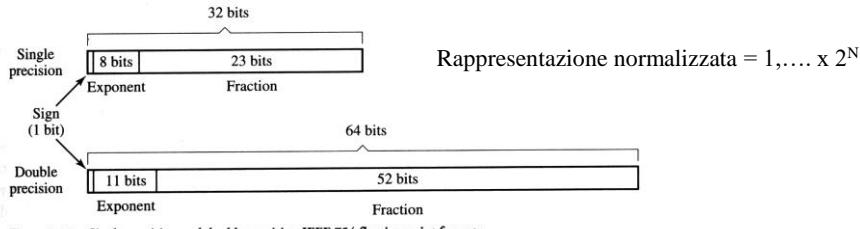


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Rappresentazione polarizzata dell'esponente:

Polarizzazione pari a 127 per singola precisione =>
1 viene codificato come 1000 0000.

Polarizzazione pari a 1023 in doppia precisione.
1 viene codificato come 1000 0000 000.



Esempio di somma in virgola mobile



$$a = 7,999 \times 10^1 \quad b = 1,61 \times 10^{-1} \quad a + b = ?$$

NB I numeri decimali sono normalizzati -> vanno riportati alla stessa base (incolonnati correttamente):

Una possibilità è: **incolonna correttamente i 2 numeri**

$$\begin{array}{r} 79,99 \\ + \\ 0,161 \\ \hline \end{array} \quad \begin{array}{l} a = 7,999 \times 10^1 = 79,99 \times 10^0 \\ b = 1,61 \times 10^{-1} = 0,161 \times 10^0 \end{array}$$

$$80,151 \times 10^0 = 80,151 = \mathbf{8,0151 \times 10^1} \text{ in forma normalizzata}$$

Altre possibilità sono:

$$\begin{array}{r} 799,9 \\ + \\ 1,61 \\ \hline \end{array} \quad \begin{array}{l} 7,999 \\ + \\ 0,0161 \\ \hline \end{array}$$

$$801,51 \times 10^{-1} = \mathbf{8,0151 \times 10^1}$$

in forma normalizzata = **8,0151 x 10¹**



Quale forma conviene utilizzare?



$$a = 7,999 \times 10^1 \quad b = 1,61 \times 10^{-1} \quad a + b = ?$$

Supponiamo di avere 4 cifre in tutto per il risultato del prodotto: 1 per la parte intera e 3 per la parte decimale:

$$\begin{array}{r} 79,99 \\ + \\ 0,161 \\ \hline 80,151 \end{array}$$

$$\begin{array}{r} 799,9 \\ + \\ 1,61 \\ \hline 801,51 \end{array}$$

$$\begin{array}{r} 7,999 \\ + \\ 0,0161 \\ \hline 8,0154 \end{array}$$

La rappresentazione **migliore** è:

$$\begin{array}{r} 7,999 \\ + \\ 0,0161 \\ \hline \end{array}$$

Risultato normalizzato

$$8,0154 \times 10^1$$

Con la quale posso scrivere: 1 cifra prima della virgola (8) e 3 cifre dopo la virgola (015), 1 va **perso**, ma è la **cifra che pesa di meno**.

Con la rappresentazione più a sinistra, perdo le decine, con quella in mezzo decine e centinaia commettendo un errore grande sulla rappresentazione.

Allineo al numero con esponente maggiore (perdo cifre di peso minore).

A.A. 2023-2024

19/39

<http://borghese.di.unimi.it/>



Approssimazione



Interi -> risultato esatto (o overflow)

Numeri decimali -> Spesso occorrono delle **approssimazioni**

- **Troncamento** (floor): $8,0151 \rightarrow 8,015$
- **Arrotondamento alla cifra superiore** (ceil): $8,0151 \rightarrow 8,016$
- **Arrotondamento alla cifra più vicina**: (round) $8,0151 \rightarrow 8,015$

IEEE754 prevede 2 bit aggiuntivi nei calcoli per mantenere l'accuratezza.

bit di guardia (guard)
bit di arrotondamento (round)

Invece di approssimare gli operandi, i bit di guardia e arrotondamento consentono di approssimare il risultato finale.

A.A. 2023-2024

20/39

<http://borghese.di.unimi.it/>



Esempio: aritmetica in floating point accurata



$a = 2,34$

$a + b = ?$

Codifica su 3 cifre decimali totali (1 prima e 2 dopo la virgola).

Approssimazione mediante **rounding**.

Senza cifre di arrotondamento e utilizzando il **troncamento**, devo scrivere:

$2,34 +$

$0,02 =$ ho troncato il secondo addendo per rimanere nella capacità

2,36

Con le cifre di guardia e di **arrotondamento** posso scrivere:

$2,3400 +$

$0,0256 =$

2,3656

L'arrotondamento finale (round) viene effettuato **sul risultato** per rientrare in 3 cifre decimali fornisce: **2,37**



L'effetto perverso del troncamento



$C = A + B$

if (C > A) then

(a)...

else

(b)....

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = (7,999 + 0,0161) \times 10^1 = 8,0151 \times 10^1$$

*Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:*

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = (7,999 + 0,0161) \times 10^1 = 8,015 \\ \Rightarrow C > A \text{ correttamente}$$

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = 7,999161$$

*Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:*

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = (7,999 + 0,0000161) \times 10^1 = 7,999 \\ \Rightarrow C = A \quad \text{errore sull'istruzione di test!!!} \quad \text{secondo numero non viene visto} \\ \text{perché non sta nella rappresentazione}$$

Questo è un errore molto comune quando si considera l'aritmetica con i numeri decimali



Non vale la proprietà associativa della somma



$$Z = A + (B + C)$$

$$A = -10^{38} \quad B = 10^{38} \quad C = 1$$

$$(B + C) = 10^{38} \Rightarrow Z = A + 10^{38} = 0 \quad \text{Risultato sbagliato}$$

$$Z = (A + B) + C$$

$$(A + B) = 0 \Rightarrow Z = 0 + 1 = 1 \quad \text{Risultato corretto}$$

Risultati molto diversi.

Non vale la proprietà associativa!



Problemi di arrotondamento – IEEE 754



$$A = 4$$

$$B = 1,0000003576278686523438 \times 10^0$$

In IEEE754:

$$A = 1 \times 2^2 = 4$$

$$B = 1,00000\ 00000\ 00000\ 00000\ 011 \times 2^0 \quad \text{parte frazionaria su 23 bit}$$

$$A = 0\ 1000\ 0001\ 00000\ 00000\ 00000\ 000 \quad \text{codifica IEEE754 su 32 bit}$$

$$B = 0\ 1111\ 1111\ 00000\ 00000\ 00000\ 011 \quad \text{codifica IEEE754 su 32 bit}$$

$$\text{Allineo } B \text{ ad } A: \Rightarrow B = 0,01000\ 00000\ 00000\ 00000\ 000\overline{1} \times 2^2 \text{ parte frazionaria su 23 bit}$$

$$\text{Segue che: } \Rightarrow C = A + B = 1,01000\ 00000\ 00000\ 00000\ 000 \times 2^2 \text{ su 23 bit}$$

$$\begin{array}{rcl} 0,01000\ 00000\ 00000\ 000\overline{1} & + & \text{su 23 bit} \\ 1,00000\ 00000\ 00000\ 00000\ 000 & = & \text{su 23 bit} \end{array} \quad \text{Base} = 2^2$$

$$\begin{array}{rcl} \hline 1,01000\ 00000\ 00000\ 00000\ 000 & & \text{su 23 bit} \\ C = A+B = 5 \Rightarrow \text{errore!} & & \text{Base} = 2^2 \end{array}$$

$$C = A+B = 5 \Rightarrow \text{errore!}_{24/39}$$



Problemi di troncamento – IEEE 754



A = 4

B = 1,0000003576278686523438 x 10⁰

In IEEE754:

A = 1 x 2²

B = 1, 00000 00000 00000 00000 011 x 2⁰ parte frazionaria su 23 bit

A = 0 1000 0001 00000 00000 00000 00000 000 codifica IEEE754 su 32 bit

B = 0 1111 1111 00000 00000 00000 00000 011 codifica IEEE754 su 32 bit

Senza aggiungere i bit di guardia e arrotondamento quando allineo B ad A (Potenza : 2²):

$$\begin{array}{rcl} 0,01000 \ 00000 \ 00000 \ 00000 \ 000 + & \text{su 23 bit} & \text{Base} = 2^2 \\ 1,00000 \ 00000 \ 00000 \ 00000 \ 000 = & \text{su 23 bit} & \\ \hline \end{array}$$

Somma $\overline{1,01000 \ 00000 \ 00000 \ 00000 \ 000}$ su 23 bit Base = 2²

Per rounding, C = 1,01000 00000 00000 00000 000 su 23 bit Base = 2²

C = A+B = A+1 = 5 **errore!** può essere pericoloso.



Problemi di troncamento – IEEE 754



A = 4

B = 1,0000003576278686523438 x 10⁰

In IEEE754:

A = 1 x 2²

B = 1, 00000 00000 00000 00000 011 x 2⁰ parte frazionaria su 23 bit

A = 0 1000 0001 00000 00000 00000 00000 000 codifica IEEE754 su 32 bit

B = 0 1111 1111 00000 00000 00000 00000 011 codifica IEEE754 su 32 bit

Se aggiungo i bit di **guardia e arrotondamento** quando allineo B ad A (Potenza : 2²):

$$\begin{array}{rcl} 0,01000 \ 00000 \ 00000 \ 00000 \ 00011 + & \text{su 25 bit} & \text{Base} = 2^2 \\ 1,00000 \ 00000 \ 00000 \ 00000 \ 00000 = & \text{su 25 bit} & \\ \hline \end{array}$$

Somma $\overline{1,01000 \ 00000 \ 00000 \ 00000 \ 00011}$ su 25 bit Base = 2²

Per rounding, C = 1,01000 00000 00000 00000 001 su 23 bit Base = 2²

C = A+B = 5 + 2⁻²³2² = 5+0.000000476837158203125

molto più vicino al valore vero!



Algoritmo di somma in virgola mobile - I



- 1) Trasformare **uno dei due numeri (normalizzati)** in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola. Si allinea all'esponente più alto (denormalizzo il numero più piccolo).

$$\begin{array}{l} a = 9,12 \times 10^0 \\ \Downarrow \\ a = 9,12 \times 10^0 \end{array} \quad \begin{array}{l} b = 8,99 \times 10^{-1} \\ \Downarrow \\ b = 0,899 \times 10^0 \end{array}$$

- 2) Effettuare la somma delle mantisse.

$$\begin{array}{r} 9,12 + \\ 0,899 = \\ \hline 10,019 \times 10^0 \end{array}$$

Se il numero risultante è normalizzato termino qui. Altrimenti:

- 3) Normalizzare il risultato.

$$10,019 \times 10^0 \rightarrow 1,0019 \times 10^1$$



Esempio di somma in virgola mobile - II



$$a = 9,999 \times 10^1 \quad b = 1,61 \times 10^{-1} \quad a + b = ?$$

Supponiamo di avere a disposizione 4 cifre per la mantissa e due per l'esponente.

- 1) Esprimo entrambi i numeri con la base 10^1

$$1,61 \times 10^{-1} = 0,0161 \times 10^1$$

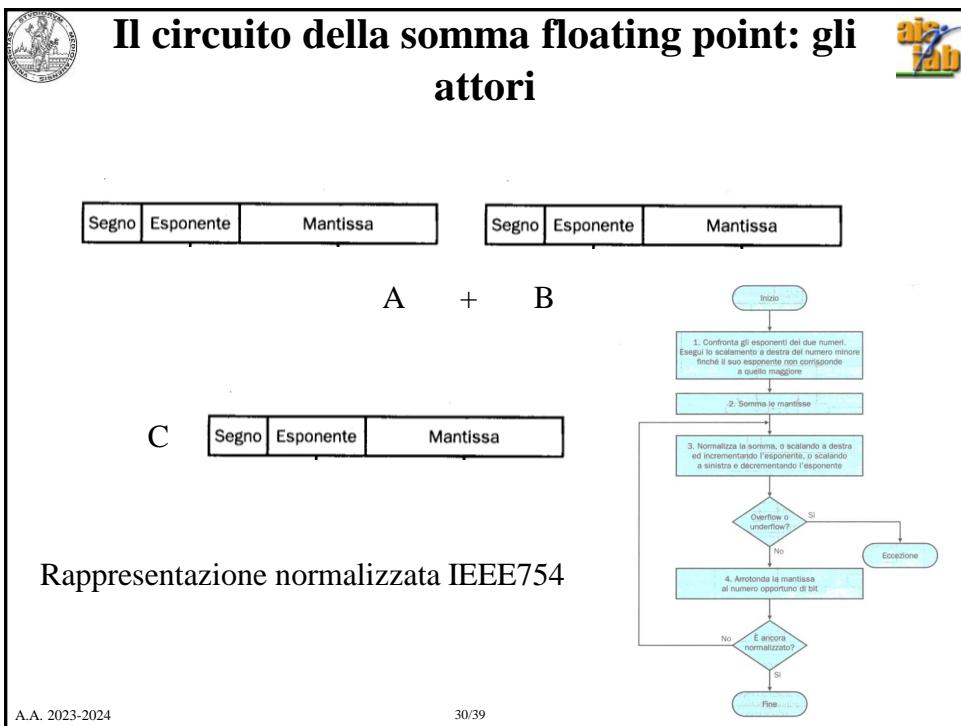
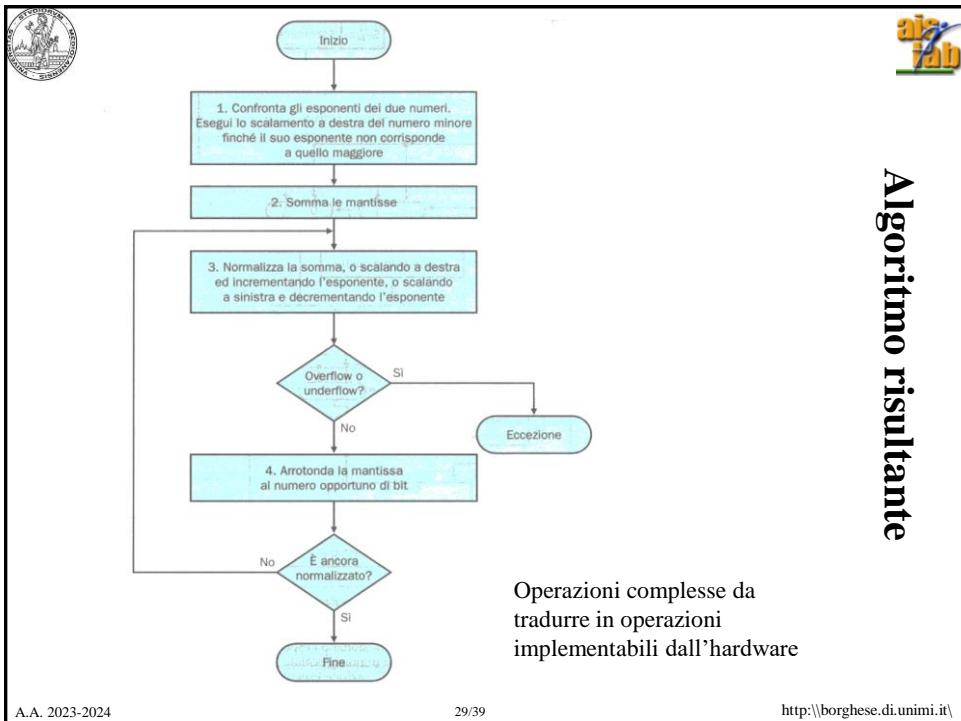
- 2) Somma delle mantisse:

$$\begin{array}{r} 9,999 + \\ 0,016\cancel{1} = \quad \text{Perdo una cifra perchè non rientra nella capacità della mantissa (troncamento)} \\ \hline 10,015 \times 10^1 \quad \text{Il risultato non è più normalizzato, anche se i due addendi sono normalizzati.} \end{array}$$

NB: In questa fase si può generare la necessità di rinormalizzare il numero (passo 3):

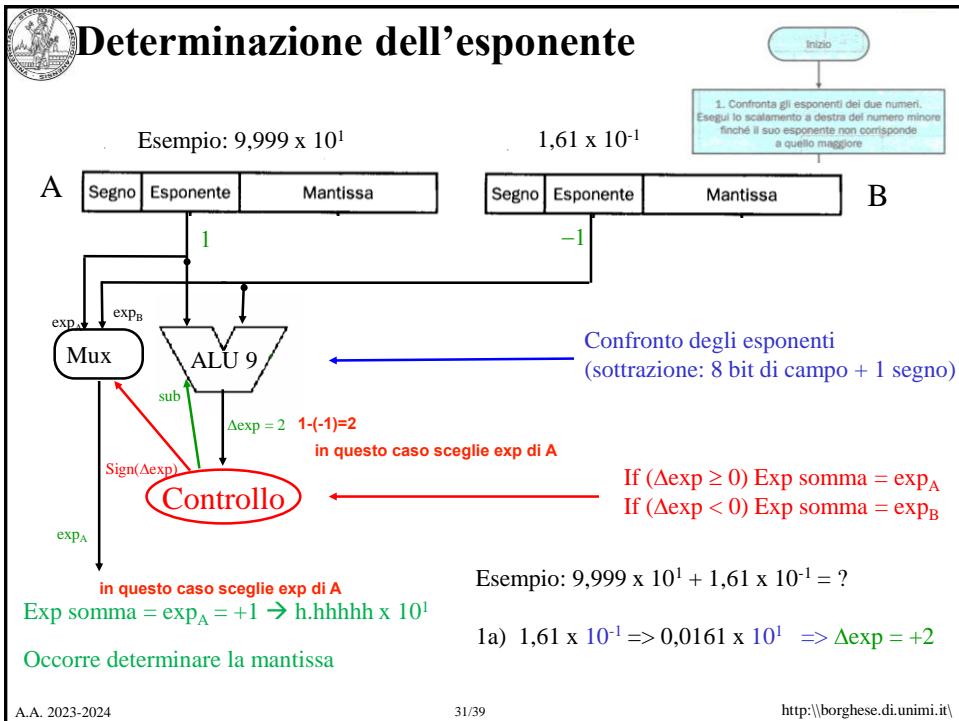
$10,015 \times 10^1 = 1,0015 \times 10^2$ in forma normalizzata (per una cifra per effetto del troncamento)

Algoritmo risultante

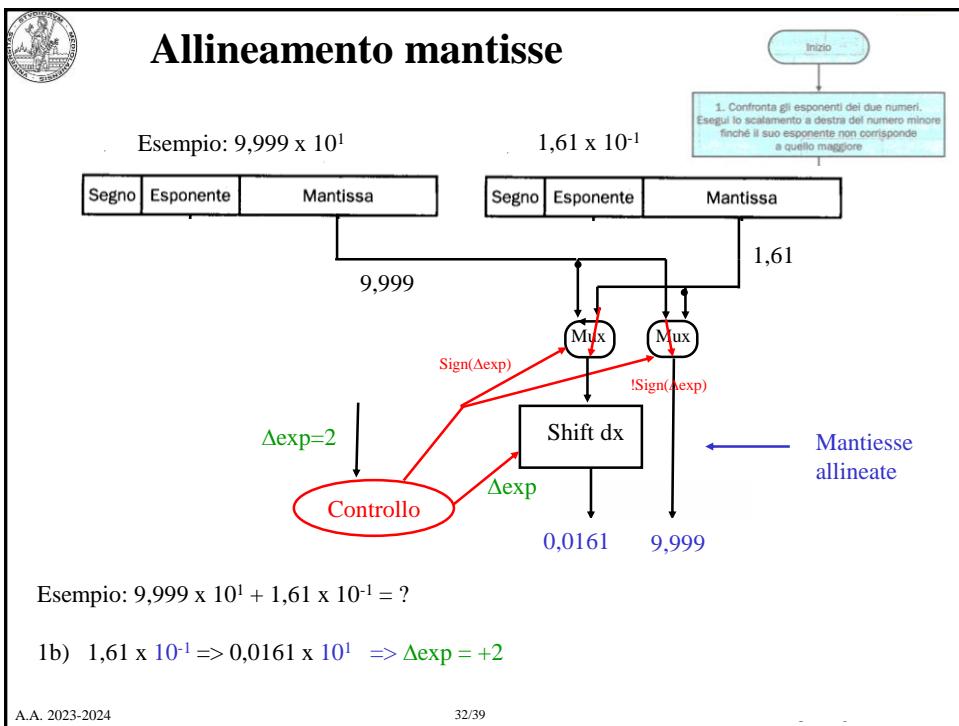




Determinazione dell'esponente



Allineamento mantisse



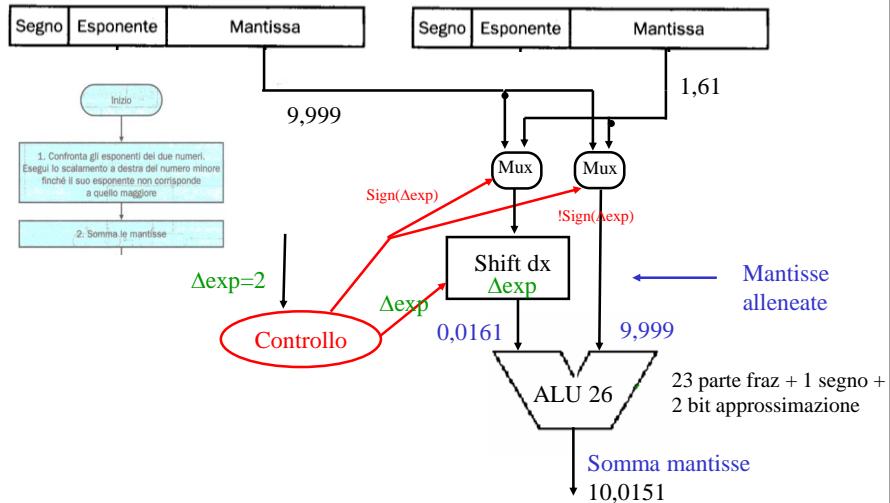


Somma delle mantisse



Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$



2) Somma delle mantisse: $0,0161 \times 10^1 + 9,999 \times 10^1 = 10,0151 \times 10^1$

<http://borghese.di.unimi.it/>

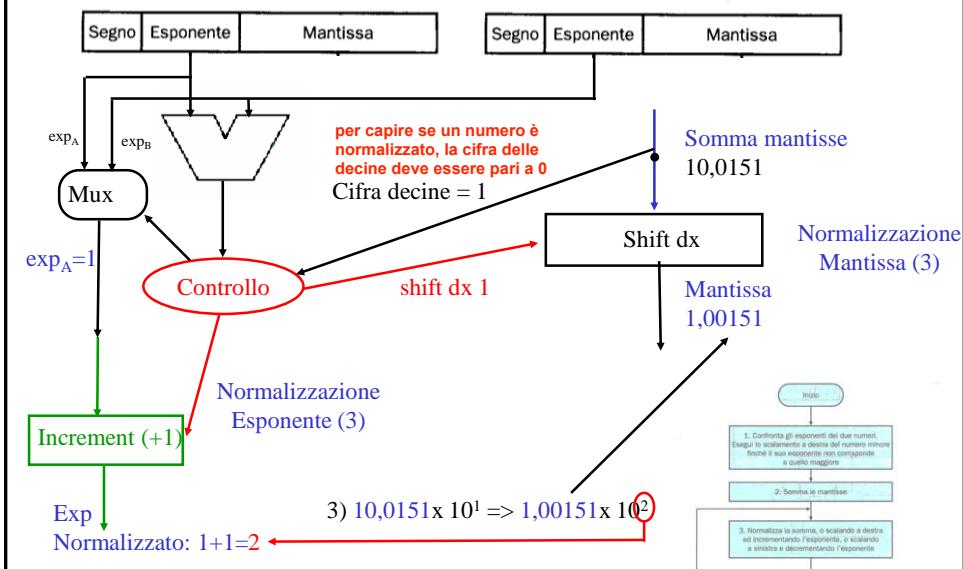


Normalizzazione



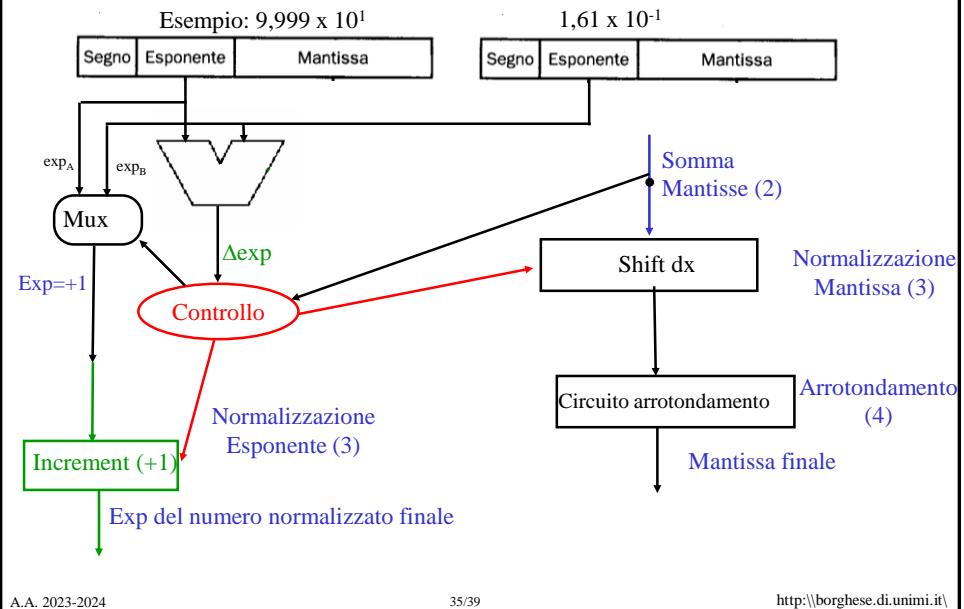
Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$





Il circuito della somma floating point: arrotondamento e normalizzazione



A.A. 2023-2024

35/39

<http://borghese.di.unimi.it/>

Circuito della somma floating point

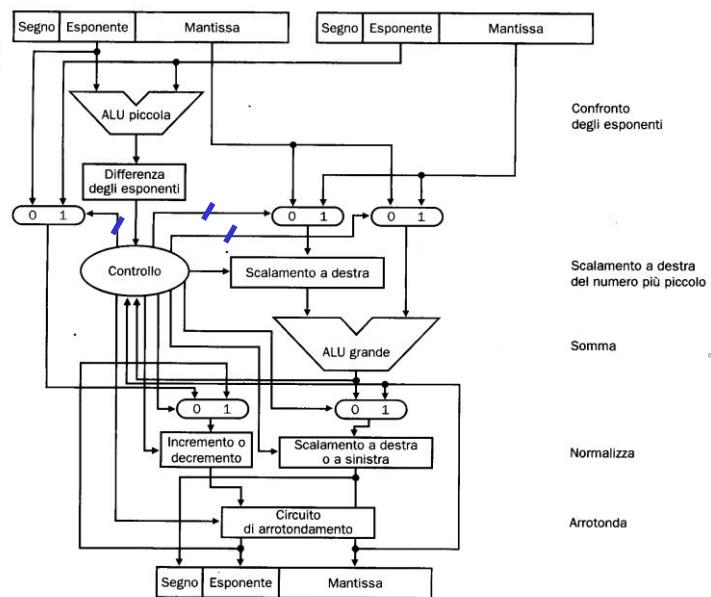


Le tre linee in blu contengono lo stesso segnale di controllo (funzione di Δ_{exp}).

Gestisce anche la rinormalizzazione:
 $9,9999 \times 10^2 = 10,00 \times 10^1$

Gestisce anche i numeri negativi.

Problemi?



A.A. 2023-2024



Circuito della somma floating point con bit di arrotondamento



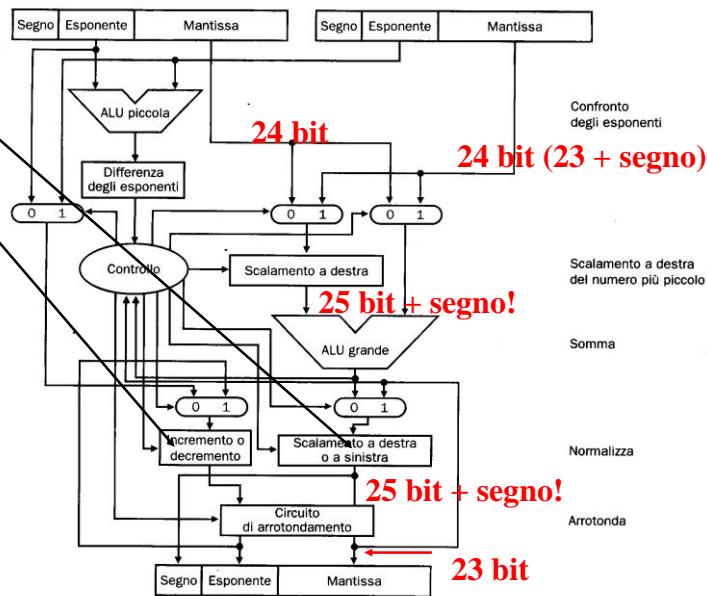
In quale caso la mantissa viene scalata a sx?

In quale caso l'esponente viene decrementato?

La rappresentazione interna, secondo IEEE 754, prevede 2 bit aggiuntivi: **bit di guardia** e **bit di arrotondamento**.

Mantissa 1,...

A.A. 2023-2024



Prodotto e divisione in virgola mobile



- Prodotto delle mantisse
- Somma degli esponenti
- Normalizzazione
- Divisione in virgola mobile = Prodotto di un numero per il suo inverso.



Sommario



- UC del firmware
- Somma in virgola mobile



ISA e linguaggio assembler

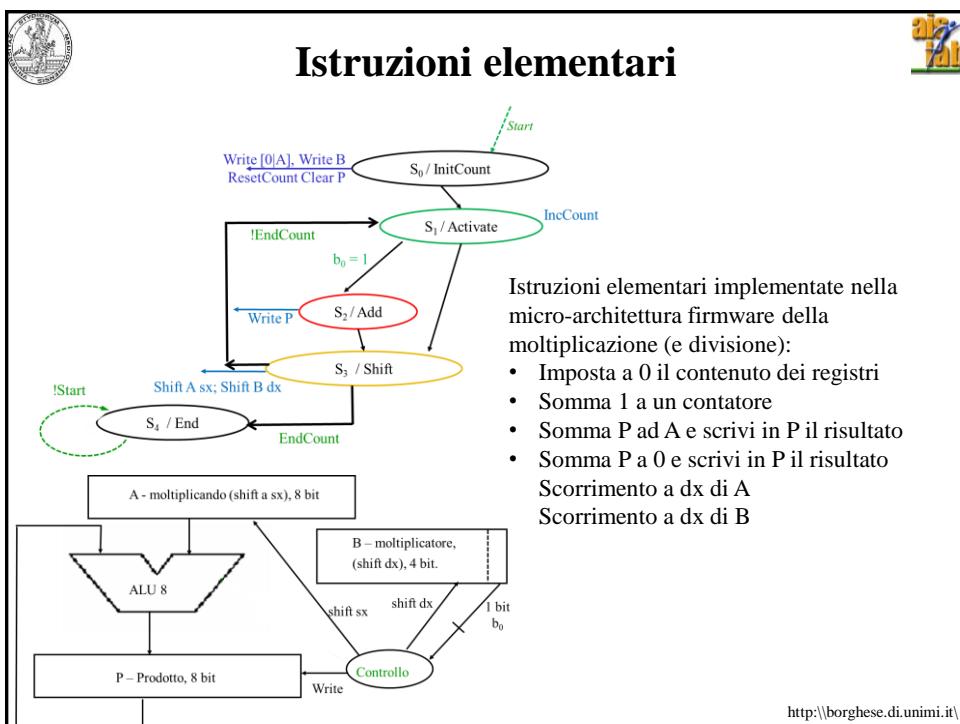
Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano
Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.



Sommario

- ISA
- Istruzioni aritmetico-logiche
- Istruzioni di accesso alla memoria
- Istruzioni di salto





Definizione di un'ISA (Instruction Set Architecture)

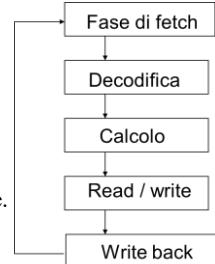


ISA: definisce le istruzioni messe a disposizione dalla macchina (in linguaggio macchina).

IS = Instruction Set = Insieme delle istruzioni. Non solo elenco ma anche:

Definizione del funzionamento: ISA è interfaccia verso i linguaggi ad alto livello.

- Tipologia di istruzioni.
- Meccanismo di funzionamento delle istruzioni.



Definizione del formato: codifica delle istruzioni (ISA è interfaccia verso l'HW).

- Formato delle istruzioni.
- Suddivisione in gruppi omogenei dei bit che costituiscono l'istruzione.
- Formato dei dati.

Le istruzioni devono contenere tutte le informazioni necessarie ad eseguire il ciclo di esecuzione dell'istruzione: registri, comandi,



ISA - IPR



ARM (Advanced RISC Machine and originally Acorn RISC Machine) è una famiglia di architetture di istruzioni.

Acorn, poi diventata RISC Limited, vende le licenze sull' ISA a società che poi realizzano i loro processori RISC. Tra i più diffusi i processori Cortex, alcuni realizzati come SoC – Systems on Chip), FPGA che comprendono memorie, interfacce, radio, ecc..

IPR – Intellectual Property Rights (proprietà intellettuale).

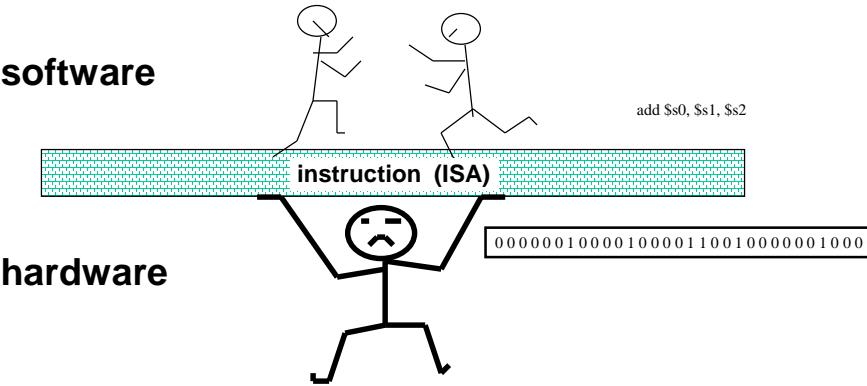
Nel 2019, RISC concede la licenza per lo sviluppo con pagamento delle royalties solo a partire dal primo prototipo delle CPU.

Non solo insieme di istruzioni elementari messe a disposizione dalla macchina (in linguaggio macchina).

L'architettura delle istruzioni, specifica come devono essere strutturate le istruzioni in modo tale che siano comprensibili alla macchina (in linguaggio macchina).

Insieme delle istruzioni



software

hardware

instruction (ISA)

add \$s0, \$s1, \$s2

00000010000100001100100000010000

Quale è più facile modificare?

A.A. 2023-2024 7/58 <http://borgheze.di.unimi.it/>

Tipi di istruzioni

Il linguaggio macchina (di una calcolatrice) è costituito da un insieme di istruzioni che possono essere eseguite dalla macchina.

Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:

- Istruzioni aritmetico-logiche;
- Istruzioni di trasferimento da/verso la memoria (*load/store*);
- Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;
- Istruzioni di trasferimento in ingresso/uscita (I/O).

• Le istruzioni e la loro codifica costituiscono l'ISA di un calcolatore.

A.A. 2023-2024 8/58 <http://borgheze.di.unimi.it/>



Tipi di istruzioni



```

for (i=0; i<N; i++)
    elem = i*N + j;           // Istruzioni di controllo
    s = v[elem];               // Istruzioni aritmetico-logiche
    z[elem] = s;               // Istruzioni di accesso a memoria
}
    
```



Le istruzioni in linguaggio macchina



- Linguaggio di programmazione direttamente comprensibile dalla macchina
 - Le parole di memoria sono interpretate come *istruzioni*
 - Vocabolario è l'*insieme delle istruzioni* (*instruction set*)

Codice in linguaggio ad alto livello (C)

a = a + c

b = b + a

var = m[a]

Codice in linguaggio macchina

00000001...0101010
00000010...1000111
10001000...0001000
00000010...0010000

Linguaggio Assembler

- Le istruzioni assembler sono una **rappresentazione simbolica del linguaggio macchina** comprensibile dall'HW.
- Rappresentazione simbolica del linguaggio macchina
 - Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit
- Rispetto ai linguaggi ad alto livello, l'assembler fornisce limitate forme di controllo del flusso e non prevede articolate strutture dati

Codice in linguaggio ad alto livello (C)	a = a + c b = b + a var = m[a]
---	---

• Linguaggio usato come linguaggio target nella fase di compilazione di un programma scritto in un linguaggio ad alto livello (es: C, Pascal, ecc.)

Codice Assembler	<pre>add \$t0,\$s0,\$s1 add \$s2,\$s2,\$t0 muli \$t1,\$s4,4 add \$t2,\$s5,\$t1 lw \$s3,0(\$t2)</pre>
-------------------------	--

• Vero e proprio linguaggio di programmazione che fornisce la visibilità diretta sull'hardware.

Codice in linguaggio macchina	00000000...0100000 00000010...0100000 10001000...0001000 00000010...0010000
--	--

A.A. 2023-2024 11/58

Assembler come linguaggio di programmazione

- Principali *svantaggi* della programmazione in linguaggio assembler:
 - Mancanza di portabilità dei programmi su macchine diverse
 - Maggiore lunghezza, difficoltà di comprensione, facilità d'errore rispetto ai programmi scritti in un linguaggio ad alto livello
- Principali *vantaggi* della programmazione in linguaggio assembler:
 - Ottimizzazione delle prestazioni.
 - Massimo sfruttamento delle potenzialità dell'hardware sottostante.
- Le strutture di controllo hanno forme limitate
- Non esistono tipi di dati all'infuori di interi, virgola mobile e caratteri.
- La gestione delle strutture dati e delle chiamate a procedura deve essere fatta in modo esplicito dal programmatore.

A.A. 2023-2024 12/58 <http://borghese.di.unimi.it>



Assembler come linguaggio di programmazione



- Alcune applicazioni richiedono un approccio *ibrido* nel quale le parti più critiche del programma sono scritte in assembly (per massimizzare le prestazioni) e le altre parti sono scritte in un linguaggio ad alto livello (le prestazioni dipendono dalle capacità di ottimizzazione del compilatore).

Esempio: Sistemi embedded o dedicati

Sistemi “automatici” di traduzione da linguaggio ad alto livello (linguaggio C) ad Assembly e codice binario ed implementazione circuitale (e.g. sistemi di sviluppo per FPGA).



I registri



- Un registro è un insieme di celle di memoria che vengono lette / scritte in parallelo.
- I registri sono associati alle variabili di un programma dal compilatore. Contengono i **dati**.
- Un processore possiede un numero limitato di registri: ad esempio il processore MIPS possiede **32 registri composti da 32-bit (word), register file**.
- I registri possono essere direttamente indirizzati mediante il loro numero progressivo (0, ..., 31) preceduto da \$: ad es.
\$0, \$1, ..., \$31
- Per convenzione di utilizzo, sono stati introdotti nomi simbolici significativi. Sono preceduti da \$, ad esempio:

\$s0, \$s1, ..., \$s7 (\$s8) Per indicare variabili in C

\$t0, \$t1, ..., \$t9 Per indicare variabili temporanee



I registri del register file



Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno

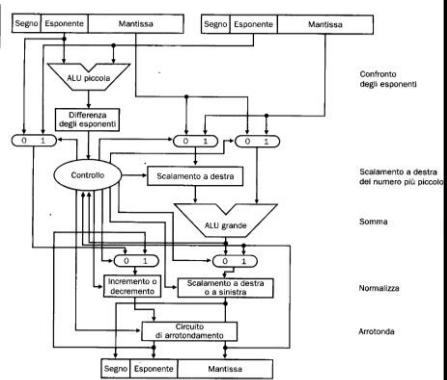


I registri per le operazioni floating point



- Esistono 32 registri utilizzati per l'esecuzione delle istruzioni.
- Esistono **32** registri per le operazioni floating point (virgola mobile) indicati come

- \$f0, ..., \$f31**
- Per le operazioni in doppia precisione si usano i registri contigui **\$f0, \$f2, \$f4, ...**





Linguaggio C: somma dei primi 100 numeri al quadrato



```
main()
{
    int i;
    int sum = 0;
    for (i = 0; i <= 100; i = i + 1)
        sum = sum + i*i;
    printf("La somma da 0 a 100 è
%d\n",sum);
}
```



Linguaggio Assembler: somma dei primi 100 numeri al quadrato



```
main()
{
    int i;
    int sum = 0;
    for (i = 0; i <= 100; i = i + 1)
        sum = sum + i*i;
    printf("La somma da 0 a 100 è
%d\n",sum);
}

.text
.align 2
.globl main
main:
    add $t6,$zero,$zero          // $t6 = 0 ind ciclo
    add $s0,$zero,$zero          // $s0 variabile da aggiornare
    add $s1,$a0,$zero            // $s1 Indice di fine ciclo, $a0 proviene dall'esterno
loop:   beq $t6,$s1,exit         // termine ciclo quando $t6 = $s1
        mult $t4,$t6,$t6           // $t4 = indice*indice
        addu $s0,$s0,$t4            // sommo $t4 a $s0 - $s0 è risultato parziale
        addu $t6,$t6,1              // incremento ind ciclo
        j loop                      // vai all'inizio del ciclo
exit:   .....
```



Sommario



- ISA
- Istruzioni aritmetico-logiche
- Istruzioni di accesso alla memoria
- Istruzioni di salto



Tipi di istruzioni



- Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:
 - Istruzioni aritmetico-logiche;
 - Istruzioni di trasferimento da/verso la memoria (*load/store*);
 - Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;
 - Istruzioni di trasferimento in ingresso/uscita (I/O).



Istruzioni aritmetico-logiche



- In MIPS, un'istruzione aritmetico-logica possiede in generale *tre* operandi: i due registri contenenti i valori da elaborare (*registri sorgente*) e il registro contenente il risultato (*registro destinazione*).
- L'ordine degli operandi è **fisso**: prima il registro contenente il **risultato** dell'operazione e poi i due operandi.

OPCODE DEST, SORG1, SORG2

```

OPCODE rd, rs, rt
rd = registro destinazione (DEST)
rs = registro source (SORG1)
rt = registro target (SORG2)

```

- La codifica prevede il codice operativo e tre campi relativi ai tre operandi:

Le operazioni vengono eseguite esclusivamente su dati presenti nella CPU, non su dati residenti nella memoria.



Esempi: istruzioni add e sub



Codice C:

$$R = A + B;$$

```

add $s6, $s7, $s8
add rd, rs, rt

```

mette la somma del contenuto di rs e rt in rd:

```

add rd, rs, rt      # rd ← rs + rt
add $s6, $s7, $s8    # $s6 ← $s7 + $s8

```

Nella traduzione da linguaggio ad alto livello a linguaggio assembler, le variabili sono associate ai registri dal compilatore

sub serve per sottrarre il contenuto di due registri sorgente rs e rt:

sub rd rs rt

e mettere la differenza del contenuto di rs e rt in rd

```

sub rd, rs, rt      # rd ← rs - rt
sub $s6, $s7, $s8    # $s6 ← $s7 - $s8

```



Istruzioni aritmetico-logiche in sequenza



Il fatto che ogni istruzione aritmetica ha tre operandi sempre nella stessa posizione consente di semplificare l'hw, ma complica alcune cose...

Codice C:

$$\begin{aligned} Z &= A - (B + C + D); \Rightarrow \\ E &= B + C + D; \quad Z = A - E; \end{aligned}$$

Suppongo che le variabili siano contenute nei seguenti registri:

A -> \$s0 B -> \$s1 C -> \$s2 D -> \$s3 Z -> \$s5

Occorre spezzare la catena di operazioni in tante operazioni su 2 operandi. Codice MIPS:

```
add $t0, $s1, $s2
add $t1, $t0, $s3
sub $s5, $s0, $t1
```



Istruzioni aritmetico-logiche



- Operazioni con un numero di operandi maggiore di tre possono essere effettuate scomponendole in operazioni più semplici.
- Ad esempio, per eseguire la somma e sottrazione delle variabili A...D nella variabile Z servono tre istruzioni che eseguono le operazioni in sequenza da sinistra a destra:

Codice C:

$$Z = A + B - C + D;$$

Codice MIPS:

$$\begin{aligned} &\text{add } \$t0, \$s0, \$s1 \\ &\text{sub } \$t1, \$t0, \$s2 \\ &\text{add } \$s5, \$t1, \$s3 \end{aligned}$$

Suppongo che le variabili siano contenute nei seguenti registri:

A -> \$s0 B -> \$s1 C -> \$s2 D -> \$s3 Z -> \$s5



Implementazione alternativa



- Operazioni con un numero di operandi maggiore di tre possono essere effettuate scomponendole in operazioni più semplici.
- Ad esempio, per eseguire la somma e sottrazione delle variabili A.. D nella variabile Z servono tre istruzioni :

Codice C: $Z = A + B - C + D;$

Può essere riscritta con il seguente codice C: $Z = (A + B) - (C - D);$

Suppongo che le variabili siano contenute nei seguenti registri:

A -> \$s0 B -> \$s1 C -> \$s2 D -> \$s3 Z -> \$s5

Codice MIPS:

<code>add \$t0, \$s0, \$s1</code> <code>sub \$t1, \$s2, \$s3</code> <code>sub \$s5, \$t0, \$t1</code>	<code>add \$t0, \$s0, \$s1</code> <code>sub \$t1, \$t0, \$s2</code> <code>add \$s5, \$t1, \$s3</code>
---	---

Sono implementazioni equivalenti. Quale implementazione è la migliore? Sceglierà il compilatore il quale cerca di massimizzare la parallelizzazione del codice.



add: varianti



- `add $s0, $s1, $s2` **#add: \$s0 = \$s1+\$s2**
- `addi $s1, $s2, 100` **#add immediate: \$s1 = \$s2+100**
 - Somma una costante: il valore del secondo operando è presente nell'istruzione come costante e sommata estesa in segno.
 $rt \leftarrow rs + costante$
- `addiu $s0, $s1, 100` **#add immediate unsigned: \$s0 = \$s1+100**
 - Somma una costante ed evita overflow.
- `addu $s0, $s1, $s2` **#add unsigned: \$s0 = \$s1+\$s2**
 - Evita overflow: la somma viene eseguita considerando gli addendi sempre positivi. Il bit più significativo è parte del numero e non è bit di segno.

Non esiste un'istruzione di subi. Perchè?
perchè sarebbe la somma in complemento a 2



Sottrazione immediata



$$\begin{aligned} B &= A - 100 \\ B &= A - (-100) \end{aligned}$$

Come si possono implementare utilizzando solo la somma?

$$\begin{aligned} B &= A + (-100) \\ B &= A + 100 \end{aligned}$$

Diventano la somma del reciproco.

```
addi $s1, $s2, -20 #add immediate: $s1 = $s2 - 20
```



Moltiplicazione

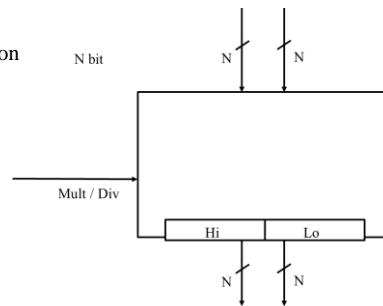


- Due istruzioni:

- mult rs rt	mult \$s0, \$s1
- multu rs rt	multu \$s0, \$s1 # unsigned

- Il registro destinazione è *Implicito*. Il risultato della moltiplicazione viene posto sempre in due registri dedicati di una parola (special purpose) denominati *Hi (High order word)* e *Lo (Low order word)*

- La moltiplicazione di due numeri rappresentabili con 32 bit può dare come risultato un numero non rappresentabile in 32 bit.





Moltiplicazione



- Il risultato della moltiplicazione si può elaborare prelevando il contenuto del registro **Hi** e del registro **Lo** utilizzando le due istruzioni:

```
- mfhi rd1          mfhi $t0      # move from Hi
  • Sposta il contenuto del registro hi nel registro rd

- mflo rd2          mflo $t1      # move from Lo
  • Sposta il contenuto del registro lo nel registro rd
```

Test sull'overflow

Risultato del prodotto

Produzione di overflow o conversione in virgola mobile



Sommario



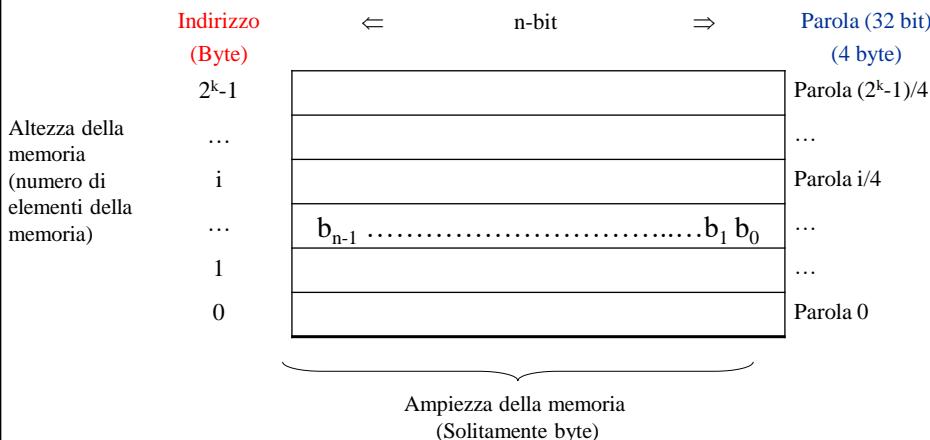
- ISA
- Istruzioni aritmetico-logiche
- Istruzioni di accesso alla memoria**
- Istruzioni di salto



La memoria principale (main memory)



- La memoria è vista come un unico grande array uni-dimensionale.
- Un **indirizzo di memoria** costituisce un **indice** all'interno dell'array.



Indirizzi nella memoria principale

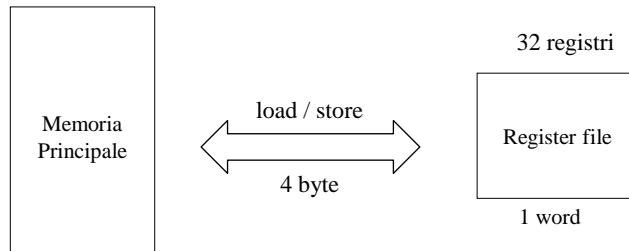


- La memoria è organizzata in **parole di memoria** composte da n -bit che possono essere indirizzate come un unicum (tipicamente 1 Byte)
- Ogni **parola** di memoria è associata ad un **indirizzo** composto da k -bit.
- I 2^k indirizzi costituiscono lo **spazio di indirizzamento** del calcolatore. Ad esempio un indirizzo di memoria composto da 32-bit genera uno spazio di indirizzamento di 2^{32} Byte o 4Gbyte.

Memoria Principale e parole

- In genere, la dimensione della parola di memoria (1 Byte) non coincide con la dimensione della parola della CPU e dei registri contenuti all'interno della *CPU* (1 word)
- Supponiamo che a ogni trasferimento tra Memoria Principale e Registri, venga trasferito contemporaneamente in parallelo un numero di Byte pari alla dimensione dei registri dell'architettura (1 word).
 - ⇒ l'operazione di *load/store* di una parola avviene in un singolo ciclo di clock del bus.
 - ⇒ Vengono trasferiti in parallelo 4 Byte
- Le parole hanno quindi generalmente indirizzo in memoria che è multiplo del numero di byte di una parola (32 bit => indirizzi spaziati di 4, 64 bit => indirizzi spaziati di 8).
- Alcuni dati possono essere rappresentati su singolo Byte (e.g. caratteri) o su coppie di Byte (e.g. audio). Può nascere un problema di allineamento dei dati.

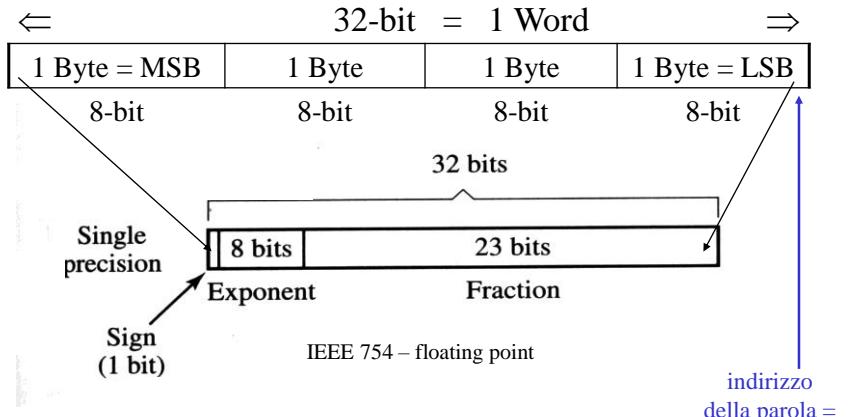


A.A. 2023-2024 33/58 <http://borgheze.di.unimi.it/>

Organizzazione dei Byte in una parola

Una parola viene costruita «montando» assieme 4 byte nelle architetture a 32 bit. MIPS utilizza un **indirizzamento al byte**, cioè l'indice punta ad un byte di memoria, byte consecutivi hanno indirizzi di memoria consecutivi. Ad esempio:



32-bit = 1 Word

1 Byte = MSB 1 Byte 1 Byte 1 Byte = LSB

8-bit 8-bit 8-bit 8-bit

32 bits

Single precision

Sign (1 bit)

Exponent

Fraction

IEEE 754 – floating point

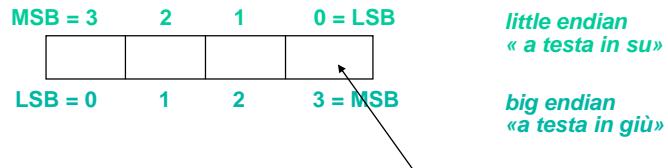
indirizzo della parola = indirizzo del LSB

A.A. 2023-2024 34/58 <http://borgheze.di.unimi.it/>



Addressing Objects: Endianess

- **LittleEndian:** address of least significant byte = word address
(xx00 = Little End of word)
 - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)
- **BigEndian:** address of most significant byte = word address
(xx00 = Big End of word)
 - IBM 360/370, Motorola 68k, MIPS, Sparc, HP

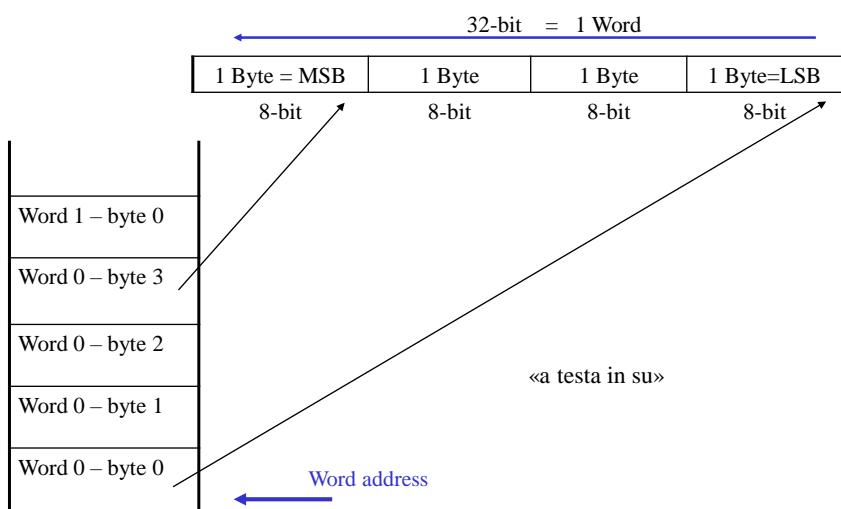


Ispirato dai "I viaggi di Gulliver" di Jonhathan Swift

Indirizzo della parola
in memoria principale

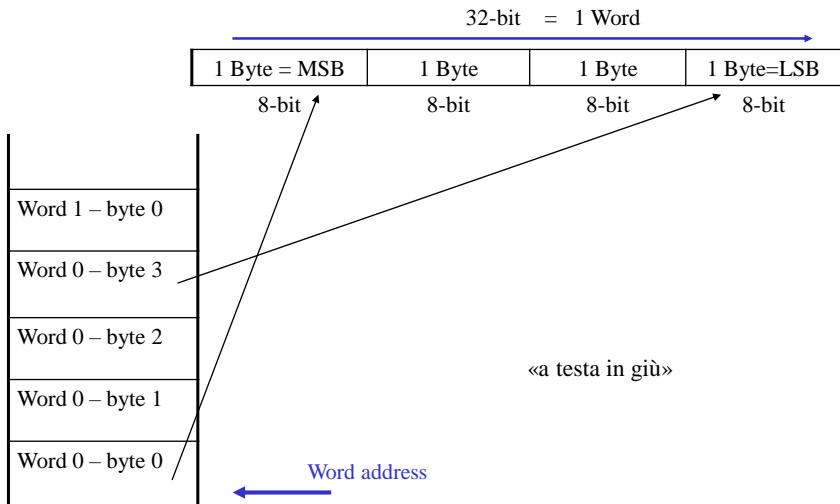


Disposizione in memoria::little endian





Disposizione in memoria::big endian



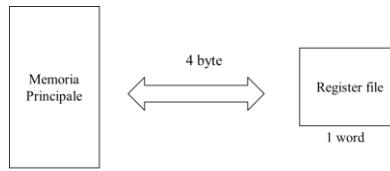
Istruzioni di trasferimento dati



- Gli operandi di una istruzione aritmetica **devono risiedere nei registri (architettura load/store)** che sono in numero limitato (32 nel MIPS). I programmi in genere richiedono un numero maggiore di variabili.
- Cosa succede ai programmi i cui dati richiedono più di 32 registri (32 variabili)?
Alcuni dati risiederanno in memoria.
- La tecnica di trasferire le variabili meno usate (o usate successivamente) in memoria viene chiamata **Register Spilling**.



Servono istruzioni apposite per trasferire dati da memoria a registri e viceversa





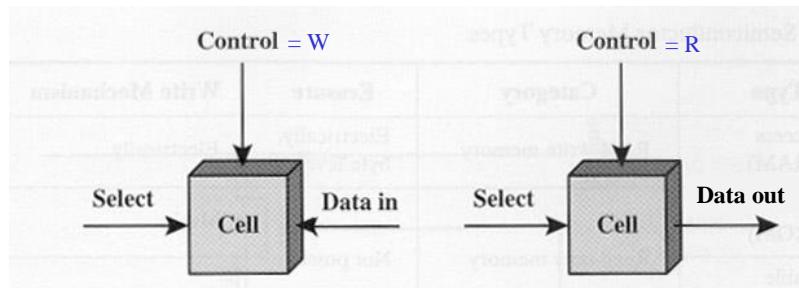
Cella di memoria



La memoria è suddivisa in celle, ciascuna delle quali assume un valore binario stabile.

Si può scrivere il valore 0/1 in una cella.

Si può leggere il valore di ciascuna cella.



Control (lettura – scrittura)
Select (selezione)
Data in oppure Data out (sense)

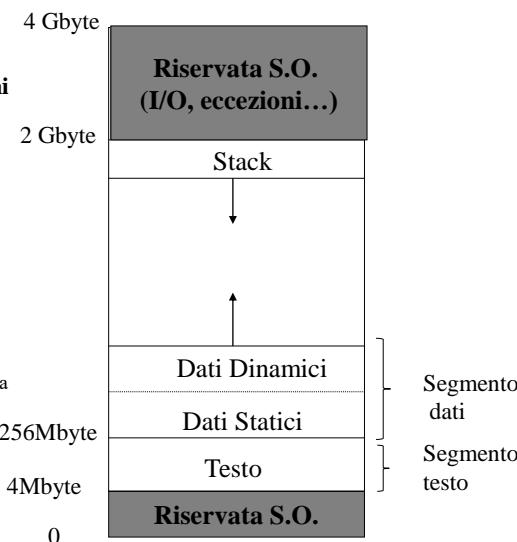


Organizzazione logica della memoria



Nei sistemi basati su processore MIPS (e Intel) la memoria è solitamente divisa in tre parti:

- **Segmento testo:** contiene le **istruzioni** del programma
- **Segmento dati:** ulteriormente suddiviso in:
 - **dati statici:** contiene dati la cui dimensione è conosciuta al momento della compilazione e il cui intervallo di vita coincide con l'esecuzione del programma
 - **dati dinamici:** contiene dati ai quali lo spazio è allocato dinamicamente al momento dell'esecuzione del programma su richiesta del programma stesso.
- **Segmento stack:** contiene lo stack allocato automaticamente da un programma durante l'esecuzione.



Convenzione di utilizzo!

Indirizzamento della memoria dati

Indirizzo della memoria su cui operare

Indirizzo base su 32 bit
Spiazzamento con un'ampiezza inferiore (**principio di località**).

Analogo all'indirizzamento degli elementi di un vettore:
Indirizzo di Vett[i] = indirizzo di Vett[0] + i*4

Indirizzo = Base + Offset

A.A. 2023-2024 41/58 <http://borgheze.di.unimi.it/>

Indirizzamento della memoria dati

Base +

Spiazzamento

MIPS fornisce due operazioni base per il trasferimento dei dati:

lw (load word) per trasferire una parola di memoria in un registro della CPU

sw (store word) per trasferire il contenuto di un registro della CPU in una parola di memoria

lw e sw richiedono come argomento l'indirizzo della locazione di memoria che contiene il primo byte sul quale devono operare (leggono / scrivono 4 byte)

A.A. 2023-2024 42/58 <http://borgheze.di.unimi.it/>



Indirizzamento della memoria dati

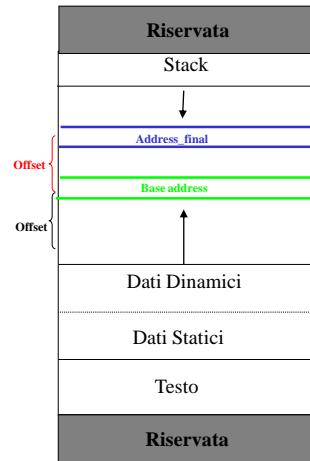


Base + spiazzamento

- Come base può essere utilizzato il registro \$gp
- Offset positivo / negativo

$$\text{Address_final} = \text{Base_address} + \text{Offset}$$

Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno



A volte come base address si considera il registro \$gp (Global Pointer)



Istruzione load



- L'istruzione di *load* trasferisce una copia di un dato/istruzione, contenuto in una specifica locazione di memoria, a un registro della *CPU*, lasciando inalterata la parola di memoria:

load LOC, reg

reg ← [LOC]

- La *CPU* invia l'indirizzo della locazione desiderata alla memoria e richiede un'operazione di lettura del suo contenuto.
- La memoria effettua la lettura del dato memorizzato all'indirizzo specificato e lo invia alla *CPU*.



Implementazione MIPS



Nel MIPS l'istruzione di caricamento di un dato dalla memoria è: "load word" (lw):

- Nel MIPS, l'istruzione lw ha tre argomenti:
 - un registro base (*base register*) che contiene il valore dell'indirizzo base (*base address*) da sommare all'offset.
 - una costante o *spiazzamento* (*offset*)
 - il *registro destinazione* in cui caricare la parola letta dalla memoria

```
lw rt, costante(rs) # rt ← M[ [rs] + costante ]
lw $s1, 100($s2)    # $s1 ← M[ [$s2] + 100 ]
```

Al registro destinazione \$s1 è assegnato il valore contenuto all'indirizzo della memoria principale: (\$s2 + 100). L'indirizzo è espresso **in byte**.

Questo spiega la semantica di «registro target» per il registro SORG2



Istruzione di sw



- L'istruzione di *store* trasferisce una parola di dato/istruzione da un registro della CPU in una specifica locazione di memoria, sovrascrivendo il contenuto precedente di quella locazione:

```
store reg, LOC          # [LOC] ← reg
```

- La CPU invia l'indirizzo della locazione di memoria, assieme con i dati che vi devono essere scritti e richiede un'operazione di scrittura.
- La memoria effettua la scrittura dei dati all'indirizzo specificato.

L'istruzione MIPS per la scrittura di un registro in memoria è la sw (store word). Essa possiede argomenti analoghi alla lw

Esempio:

```
sw rt, costante(rs) # M[ [rs] + costante ] ← rt
sw $s1, 100($s2)   # M[ [$s2] + 100 ] ← $s1
```



lw & sw: esempio



Elaborazione di dati di un vettore A.

Codice C: $A[12] = h + A[8];$

- Si suppone che:
 - la variabile **h** sia associata al registro **\$s2**
 - l'indirizzo del primo elemento dell'array (*base address*) sia contenuto nel registro **\$s3 (A[0])**

Codice MIPS:

```
lw $t0, 32($s3)          # $t0 ← M[[$s3] + 32]
add $t0, $s2, $t0         # $t0 ← $s2 + $t0
sw $t0, 48($s3)          # M[[$s3] + 48] ← $t0
```

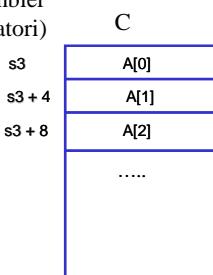


Memorizzazione di un vettore



- L'elemento **i-esimo** di un array di N elementi, si troverà nella locazione **br + 4 * i** dove:
 - **br** è il registro base;
 - **i** è l'indice del vettore (e.g. codice C);
 - il fattore **4** dipende dall'indirizzamento al byte della memoria nel MIPS e si riferisce ad architetture a 32 bit

Assembler
(puntatori)



A[0]	3	2	1	0
A[1]	7	6	5	4
A[2]	11	10	9	8
A[N-1]	$2^{N \cdot 4} - 1$	$2^{N \cdot 4} - 2$	$2^{N \cdot 4} - 3$	$2^{(N-1) \cdot 4}$

0x40000

0x40004

0x40008

.....



Frammento di gestione di un vettore



- Sia A un array di N word. **Realizziamo l'istruzione C:** $g = h + A[i]$
- Si suppone che:
 - le variabili **g, h, i** siano associate rispettivamente ai registri **\$s1, \$s2, ed \$s4**
 - l'indirizzo del primo elemento dell'array (*base address*) sia contenuto nel registro **\$s3**
- L'elemento **i-esimo** dell'array si trova nella locazione di memoria di indirizzo (**$\$s3 + 4 * i$**)
- Caricamento dell'indirizzo di $A[i]$ nel registro temporaneo **\$t1**:


```
muli $t1, $s4, 4      # $t1 ← 4 * i
add $t1, $t1, $s3      # $t1 ← address of A[i]
# that is ($s3 + 4 * i)
```
- Per trasferire $A[i]$ nel registro temporaneo **\$t0**:


```
lw $t0, 0($t1)      # $t0 ← A[i]
```
- Per sommare h e $A[i]$ e mettere il risultato in g :


```
add $s1, $s2, $t0      # g = h + A[i]
```

A.A. 2023-2024

49/58

<http://borgheze.di.unimi.it/>

Vettori: aritmetica dei puntatori



Codice C:

```
for (i=0; i<N; i+=2)
    g = h + A[i];
```

Supponiamo che l'indirizzo del primo elemento dell'array A (*base address*) sia contenuto nel registro **\$s3**

Codice Assembler:

First iteration:

```
lw $t0, 0($s3)      # Carico l'indirizzo dell'elemento 0
# di A (base address) = &A
```

All the other iterations:

```
addi $s3, $s3, 8      # Carico l'elemento successivo (+=2) &A +=8
lw $t0, 0($s3)
...
```

- Increment of the address of the location of $A[i]$, inside $\$s3$, by adding the proper offset (here 4 Byte * 2 elements = 8 Byte, as we supposed a 32 bit architecture)

A.A. 2023-2024

50/58

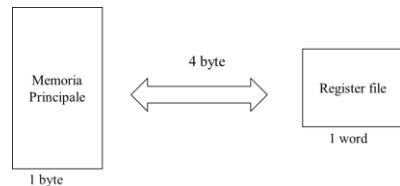
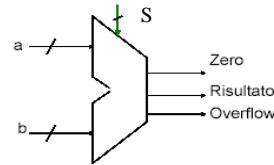
<http://borgheze.di.unimi.it/>



Istruzioni aritmetiche vs. load/store



- Le istruzioni aritmetiche leggono il contenuto di due registri (operandi), eseguono una computazione e scrivono il risultato in un terzo registro (destinazione o risultato)
- Le operazioni di trasferimento dati leggono e scrivono un solo operando senza effettuare nessuna computazione. Tuttavia utilizzano 2 registri, di cui uno viene utilizzato per costruire l'indirizzo.
- Le operazioni di trasferimento dati sono necessarie per eseguire le istruzioni aritmetiche!! (cf. Roof model)



Sommario



- ISA
- Istruzioni aritmetico-logiche
- Istruzioni di accesso alla memoria
- Istruzioni di salto**



Istruzioni di salto in ciclo for



Ciclo a condizione iniziale di uscita (può essere eseguito 0 volte)

```
for (i=0; i<N; i++)
{
    elem = i*N + j;
    s = v[elem];
    z[elem] = s;
}
```

```
inizia: 0x400000 beq $t0, $s0, esci           // $s0 conteggio fine ciclo
        0x400004 ..
        ..
        ..
        0x40068 j inizia                         ; torna in ciclo
esci:   0x4006C ...
```



Istruzioni di salto condizionato



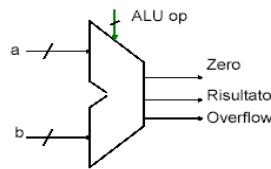
- Salti condizionati relativi:

- **beq rs, rt, Etichetta** (*branch on equal*)
- **bne rs, rt, Etichetta** (*branch on not equal*)

- Salti condizionati relativi:

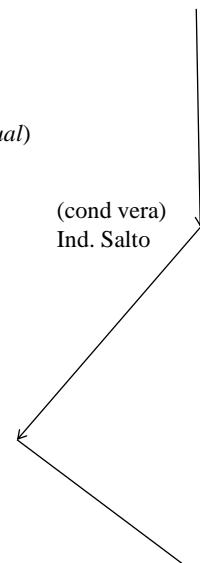
- Il flusso sequenziale di controllo cambia solo se la condizione è vera. (beq)

```
beq $s1, $s0, esci # if (s1 == s0) esci
bne $s1, $s0, esci # if (s0 ≠ s0) esci
```



(cond vera)
Ind. Salto

(cond falsa)
Ind. =+4
(procedi in sequenza)





Condizioni di minoranza



```
blt $s1, $s0, esci # if (s1 < s0) esci
```

blt è una **pseudo-istruzione**:

- Non fa parte dell'ISA
- E' un'istruzione molto utilizzata
- Una psudo-istruzione equivale a due o più istruzioni dell'ISA

```
slt $t0, $s1, $s0      # if (s1 < s0) t0 = 1
bne $t0, $zero, esci    # if (t0 ≠ 0) esci
```



I salti incondizionati

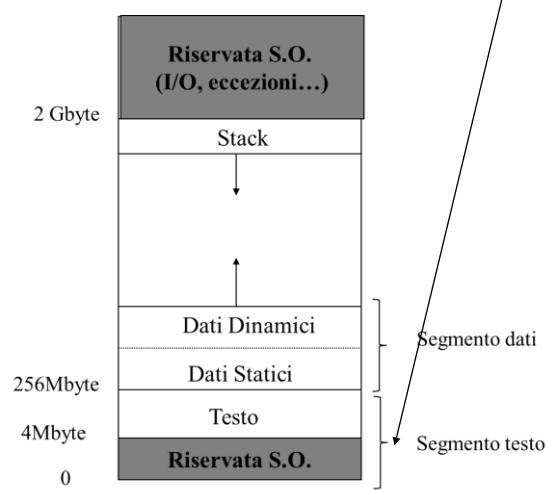


Salti incondizionati assoluti (j, jal...) – j Etichetta

Il salto viene sempre eseguito.

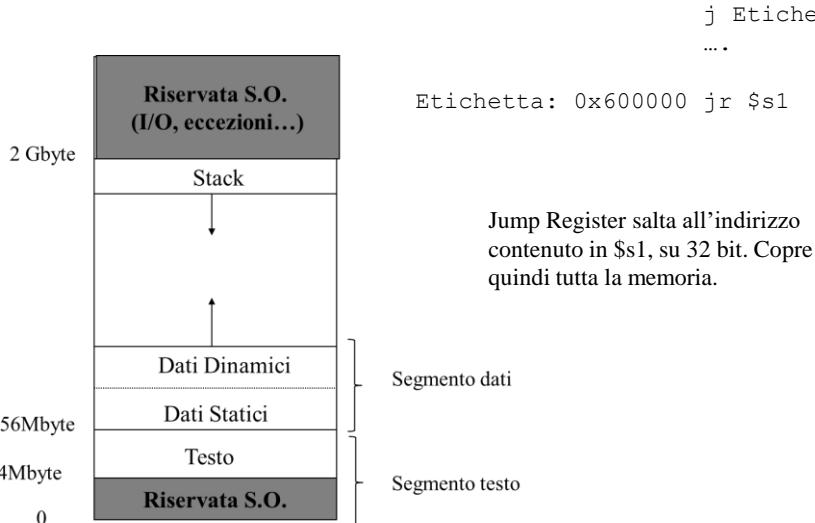
L'indirizzo di destinazione del salto è un indirizzo assoluto di memoria.

L'indirizzo di destinazione del salto è un numero sempre positivo.





Salti più ampi



Sommario



- ISA
- Istruzioni aritmetico-logiche
- Istruzioni di accesso alla memoria
- Istruzioni di salto



Architettura degli elaboratori Linguaggio Macchina e Register File

Prof. Alberto Borghese
Dipartimento di Informatica
Alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.

A.A. 2023-2024

1/48

<http://borgheze.di.unimi.it/>



Sommario

- Linguaggio macchina
- Formato R
- Formato I
- Formato J
- Il register file

A.A. 2023-2024

2/48

<http://borgheze.di.unimi.it/>

Linguaggio Macchina

- Le istruzioni assembler sono una **rappresentazione simbolica del linguaggio macchina** comprensibile dall'HW.
- Rappresentazione simbolica del linguaggio macchina
 - Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit
- Rispetto ai linguaggi ad alto livello, l'assembler fornisce limitate forme di controllo del flusso e non prevede articolate strutture dati

<p style="text-align: center;">Codice in linguaggio ad alto livello (C)</p> <pre>a = a + c b = b + a var = m[a]</pre>	 Codice Assembler <pre>add \$t0,\$s0,\$s1 add \$s2,\$s2,\$t0 mul \$t1,\$s4,4 add \$t2,\$s5,\$t1 lw \$s3,0(\$t2)</pre>  Codice in linguaggio macchina <pre>01110001010101010 000110101000111 000010000010000 001000100010000</pre>
--	---

A.A. 2023-2024 3/48 <http://borgheze.di.unimi.it/>

Definizione di un'ISA (Instruction Set Architecture)

ISA: definisce le istruzioni messe a disposizione dalla macchina (in linguaggio macchina).

IS = Instruction Set = Insieme delle istruzioni. Non solo elenco ma anche:

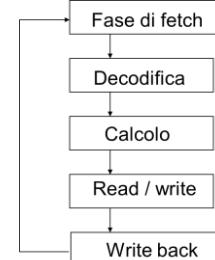
Definizione del funzionamento: ISA è interfaccia verso i linguaggi ad alto livello.

- Tipologia di istruzioni.
- Meccanismo di funzionamento delle istruzioni.

Definizione del formato: codifica delle istruzioni (ISA è interfaccia verso l'HW).

- Formato delle istruzioni.
- Suddivisione in gruppi omogenei dei bit che costituiscono l'istruzione.
- Formato dei dati.

Le istruzioni devono contenere tutte le informazioni necessarie ad eseguire il ciclo di esecuzione dell'istruzione: registri, comandi,



```

graph TD
    A[Fase di fetch] --> B[Decodifica]
    B --> C[Calcolo]
    C --> D[Read / write]
    D --> E[Write back]
  
```

A.A. 2023-2024 4/48 <http://borgheze.di.unimi.it/>



Codifica delle istruzioni in linguaggio macchina



- Tutte le istruzioni MIPS hanno la stessa dimensione (**32 bit = 1 parola**) – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
 - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di **3 tipi** (formati):
 - Tipo R (register)** – **Lavorano prevalentemente su 3 registri.**
 - Istruzioni aritmetico-logiche.
 - Tipo I (immediate)** – **Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
 - Istruzioni di accesso alla memoria o operazioni con una costante.
 - Tipo J (jump)** – **Lavora senza registri: codice operativo + indirizzo di salto.**
 - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt			Indirizzo / costante
J	op				Indirizzo / costante	

A.A. 2023-2024

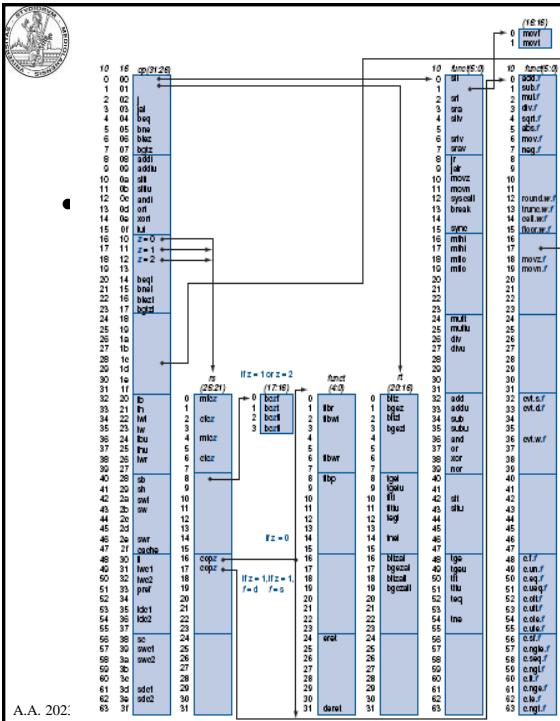
5/48

http://borghese.di.unimi.it/



La mappa dei codici operativi

FIGURE A.10.2
del Patterson Hennessy





Sommario



- Linguaggio macchina
- **Formato R**
- Formato I
- Formato J
- Il register file

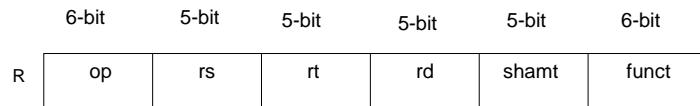


Formato delle istruzioni di tipo R



Contiene:

- Un codice operativo su 6 bit
- Un registro source, rs, su 5 bit (32 registri)
- Un registro target, rt, su 5 bit (32 registri)
- Un registro destinazione, rd, su 5 bit (32 registri)
- Un numero di posizioni di shift (shift amount, shampt), su 5 bit
- Un codice di funzione (cf. selettore ALU), su 6 bit

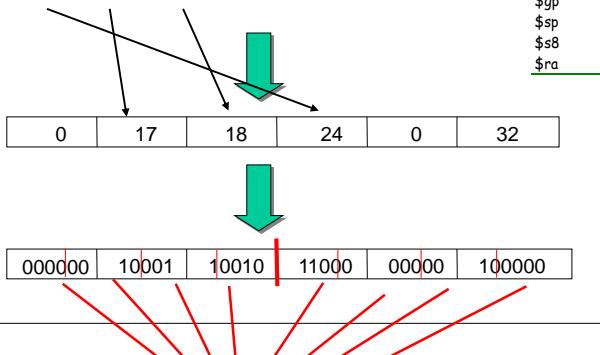


Istruzioni di tipo R: esempio

Nome	Numero	Utilizzo
\$zero	0	costante zero
\$at	1	riservato per l'assemblatore
\$v0-\$v1	2-3	valori di ritorno di una procedura
\$a0-\$a3	4-7	argomenti di una procedura
\$t0-\$t7	8-15	registri temporanei (non salvati)
\$s0-\$s7	16-23	registri salvati
\$t8-\$t9	24-25	registri temporanei (non salvati)
\$k0-\$k1	26-27	gestione delle eccezioni
\$gp	28	puntatore alla global area (dati)
\$sp	29	stack pointer
\$s8	30	registro salvato (fp)
\$ra	31	indirizzo di ritorno

add \$t8, \$s1, \$s2



0 17 18 24 0 32

000000 10001 10010 11000 00000 100000

0x0232A020

A.A. 2023-2024 9/48 <http://borgheze.di.unimi.it/>

Istruzioni di tipo R: esempi

Nome campo	Op	Rs	rt	rd	Shamt	Funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
add \$t8, \$s1, \$s2	000000	10001	10010	11000	00000	100000

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
sub \$t8, \$s1, \$s2	000000	10001	10010	11000	00000	100010

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
and \$s1, \$s2, \$s3	000000	10010	10011	10001	00000	100100

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
sll \$s1, \$s2, 3	000000	X	10010	10001	00011	000000
<i>s1 = s2 * 2³ Se s2 contiene 20 (0000....0010100) => s1 conterrà = 20*2³ = 160 (0000....0010100000)</i>						

Nome campo	op	rs	rt	rd	shamt	funct
Dimensione	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
srl \$s1, \$s2, 3	000000	X	10010	10001	00011	000010
<i>s1 = s2 * 2⁻³</i>						

A.A. 2023-2024 10/48 <http://borgheze.di.unimi.it/>



Sommario



- Linguaggio macchina
- Formato R
- **Formato I**
- Formato J
- Il register file



Formato istruzioni di tipo I



I	op	rs	rt	costante / indirizzo
	6 bit	5 bit	5 bit	16 bit

- In questo caso, i campi hanno il seguente significato:
 - **op** identifica il tipo di istruzione;
 - **rs** indica il registro sorgente. Nel caso di una lw contiene il registro base;
 - **rt** indica il registro target. Nel caso di una lw, contiene il registro destinazione dell'istruzione di caricamento;
 - **Costante / indirizzo**. Nel caso di una lw riporta lo spiazzamento (offset) nel segmento dati, nel caso di una beq riporta lo spiazzamento nel segmento testo.

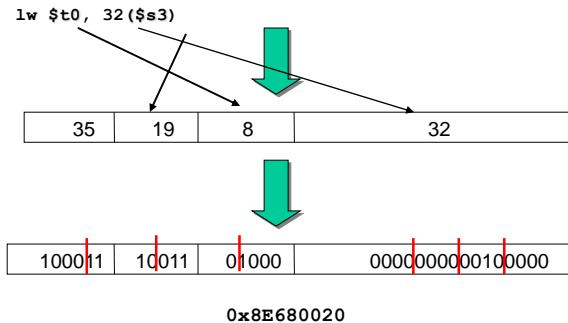


Istruzioni di tipo I: accesso a memoria



Con questo formato una istruzione **lw (sw)** può indirizzare byte nell'intervallo -2^{15} (-32K) ÷ $+2^{15}-1$ (32K -1) rispetto all'indirizzo base:

$$\text{indirizzo} = \text{indirizzo_base} + \text{offset} (= \text{costante})$$



La costante può essere positiva o negativa

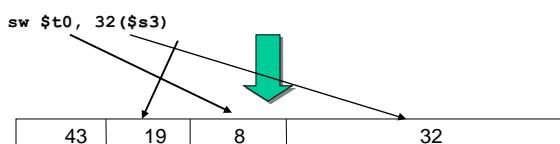


Istruzioni di tipo I: accesso memoria



Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
lw \$t0, 32(\$s3)	100011	10011	01000	0000 0000 0010 0000

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
sw \$t0, 32(\$s3)	101011	10011	01000	0000 0000 0010 0000



Differenza di 1 bit (distanza di Manhattan pari a 1) ->
cambia la direzione del trasferimento con la memoria.



Istruzioni di tipo I: aritmetico-logiche



Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
addi \$t0, \$s3, 64	001000	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
andi \$t0, \$s3, 64	001100	10011	01000	0000 0000 0100 0000

Nome campo	op	rs	rt	costante
Dimensione	6-bit	5-bit	5-bit	16-bit
slti \$t0, \$s2, 8	001010	10010	01000	0000 0000 0000 1000

\$t0 = 1 if \$s2 < 8

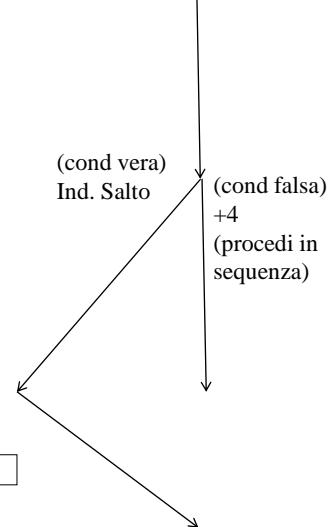
NB ruolo del registro target: nelle istruzioni di addi e di lw rappresenta **dove nel register file vado a scrivere**. Nelle istruzioni di tipo R e di sw contiene **uno dei dati sorgente**.



Istruzioni di tipo I: salto condizionato



- Salti condizionati relativi:
 - **beq r1, r2, L1** (*branch on equal*)
 - **bne r1, r2, L1** (*branch on not equal*)
- Salti condizionati relativi:
 - Il flusso sequenziale di controllo cambia solo se la condizione è vera (**beq**)
 - Il calcolo del valore dell'etichetta **L1** (**indirizzo di destinazione del salto**) avviene a partire dal Program Counter (PC).
 - Indirizzamento del tipo Base (PC) + Spiazzamento.



I	op	rs	rt	indirizzo
	6 bit	5 bit	5 bit	16 bit



Allargamento dello spazio di indirizzamento relativo



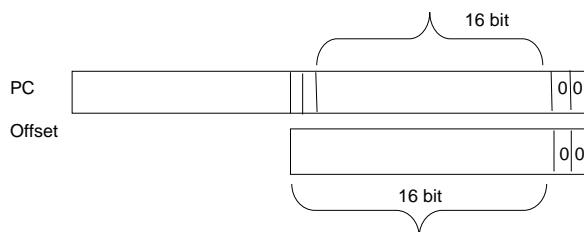
0000	0	0
0100	1	4
1000	2	8
1100	3	12

Gli offset in Byte avranno sempre **00 come ultimi 2 bit** perché gli indirizzi sono multipli di 4.

Calcolo lo spiazzamento in numero di parole invece che di Byte.

Considero 64 Kword (64K istruzioni) invece di 64Kbyte. Lo spazio indirizzabile all'interno del segmento di testo è di $64\text{Kword} * 4 = 256\text{Kbyte}$.

Moltiplicare per 4 vuol dire operare uno shift a sinistra di due posizioni dell'offset



La costante su 16 bit, contenuta nel codice, rappresenta l'offset in termini di numero di istruzioni



Calcolo dell'indirizzo di salto



t1 = 10; t0 = 0;	0x0000	addi \$t1, \$zero, 10 # t1=10
	0x0004	addi \$t0,\$zero,0 # t0 =0;
Repeat	0x0008	loop: addi \$t0, \$t0,1 # t0++
{ t0++;	0x000C
...		.
}	0x0020	until (t0 == t1) bne \$t0, \$t1, loop

Quanto vale il campo costante da inserire nella stringa dell'istruzione bne?

Ind	Ind	Istruzione	# istruzione
Dec	Exadec		
000	0x0000	addi \$t1, \$zero, 10 # N=10	1
004	0x0004	addi \$t0,\$zero,0 # i =0;	2
008	0x0008	loop: addi \$t0, \$t0,1 # i++	3
012	0x000C	4
036	0x0024	bne \$t0, \$t1, loop	
040	0x0028		9

L'indirizzo di destinazione è 0x8 (indirizzo dell'etichetta loop)

Lo spiazzamento del salto in byte è pari a: $(0x8 - 0x24) = (8-36) = -28$ Byte
 Lo spiazzamento del salto in numero di istruzioni è pari a -7 istruzioni

Prova: Indirizzo di salto = Indirizzo PC+4 + Offset (#istruzioni) * 4
 loop = 8 → costante = 36 + $\boxed{-7}$ * 4

5	8	9	-7 = 1111 1111 1111 1001
---	---	---	--------------------------

A.A. 2023-2024 19/48 <http://borgheze.di.unimi.it/>

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
beq \$s1, \$s2, L1	000100	10001	10010	0000 0000 0001 1001

L1 = **PC+4 + 100** (byte) Codifica su 18 bit: **0000 0000 0001 1001(00)** in binario.

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
beq \$s1, \$s2, L1	000100	10001	10010	1111 1111 1110 0111

L1 = **PC+4 - 100** (byte) Codifica su 18 bit: **1111 1111 1110 0111(00)** in binario.

A.A. 2023-2024 20/48 <http://borgheze.di.unimi.it/>



Osservazioni



Nome campo	op	rs	rt	Indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
<code>beq \$s1, \$s2, Label</code>	000100	10001	10010	0000	0000	0001	1000
Nome campo	op	rs	rt	indirizzo			
Dimensione	6-bit	5-bit	5-bit	16-bit			
<code>lw \$s2, 24(\$s1)</code>	100011	10001	10010	0000	0000	0001	1000
Nome campo	op	rs	rt	costante			
Dimensione	6-bit	5-bit	5-bit	16-bit			
<code>addi \$s2,\$s1,24</code>	001000	10001	10010	0000	0000	0001	1000

Salto di 24 istruzioni in avanti.

Istruzioni molto diverse possono distare pochi bit una dall'altra.



Formato R e operazioni logico-matematiche



Non tutte le operazioni logico-matematico, sono di tipo R.

Le operazioni logico-matematiche di tipo R hanno codice operativo 0.

Non tutte le operazioni con codice operativo 0 sono logico-matematiche (ad esempio ci sono le istruzioni di *jr*, *syscall*...).

Occorre distinguere il funzionamento dell'istruzione elementare dalla sua codifica.

- Codifiche simili (e.g. Tipo R) possono essere condivise da istruzioni di tipo diverso (e.g. aritmetico-logiche, salto).
- Codifiche diverse (e.g. Tipo I e Tipo R) possono essere condivise da istruzioni dello stesso tipo (e.g. add ed addi)

Non c'è corrispondenza 1 a 1, tra tipi strutturali e tipi funzionali.



Sommario



- Linguaggio macchina
- Formato R
- Formato I
- **Formato J**
- Il register file



Formato istruzioni di tipo J



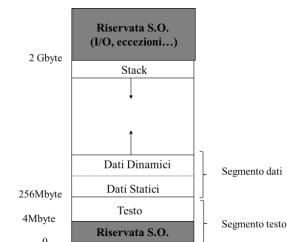
- E' il formato usato per le istruzioni di salto incondizionato (*jump*):

j	indirizzo
op	indirizzo

6 bit

26 bit

- In questo caso, i campi hanno il seguente significato:
 - **op** indica il tipo di operazione;
 - **indirizzo** (composto da **26-bit**) riporta una parte (26 bit su 32) dell'indirizzo **assoluto** di destinazione del salto.
- I 26-bit del campo **indirizzo** rappresentano un indirizzo di parola (**word address**) 2^{26} parole = 256 Mbyte (segmento testo).



PC



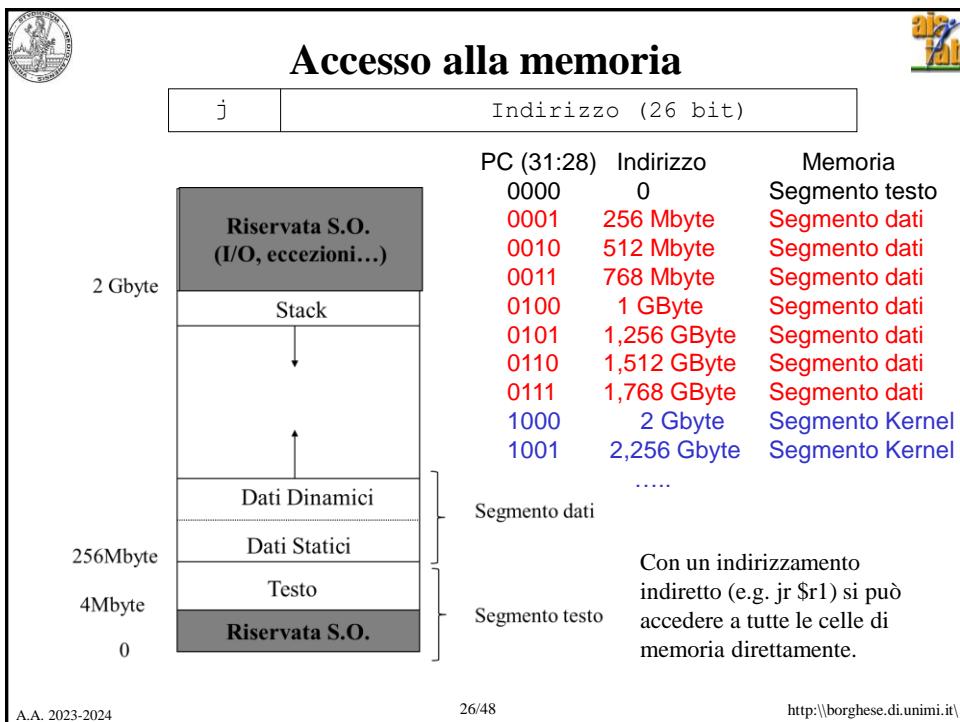
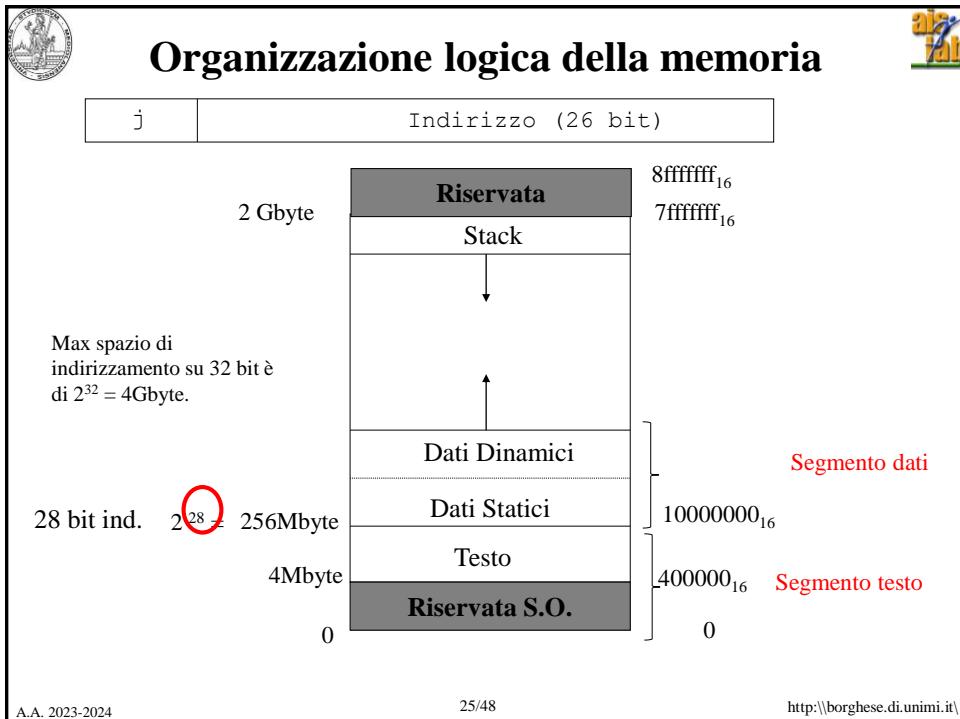
4 bit
(invariati)

26 bit



2 bit

$$0 \leq \text{indirizzo} < 2^{28} = 256 \text{ Mbyte (segmento testo!)}$$





Codifica delle istruzioni



- Tutte le istruzioni MIPS hanno la stessa dimensione (**32 bit**) – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
 - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di **3** tipi (formati):
 - **Tipo R (register)** – **Lavorano su 3 registri**.
 - Istruzioni aritmetico-logiche.
 - **Tipo I (immediate)** – **Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
 - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
 - **Tipo J (jump)** – **Lavora senza registri: codice operativo + indirizzo di salto.**
 - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt		Indirizzo / costante	
J	op			Indirizzo / costante		



Formati e tipi di istruzioni



Tipi di istruzioni di un'ISA

- Istruzioni aritmetiche
- Istruzioni di accesso a memoria
- Istruzioni di controllo di flusso
- (Istruzioni di I/O)

Formati

- R - Registro
- I – Immediato
- J - Jump (salto)

Non c'è corrispondenza 1 a 1 tra tipi di istruzioni e formati



Sommario



- Linguaggio macchina
- Formato R
- Formato I
- Formato J
- Il register file



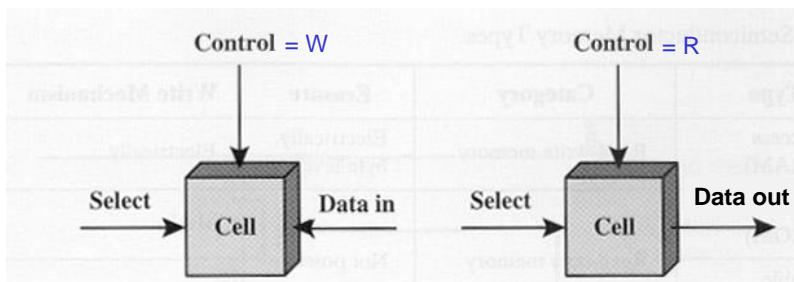
Cella di memoria



La memoria è suddivisa in celle, ciascuna delle quali assume un valore binario stabile.

Si può scrivere il valore 0/1 in una cella.

Si può leggere il valore di ciascuna cella.



Control (lettura – scrittura)

Select (selezione)

Data in oppure Data out (sense)

Base



Latch sincrono come elemento di memoria

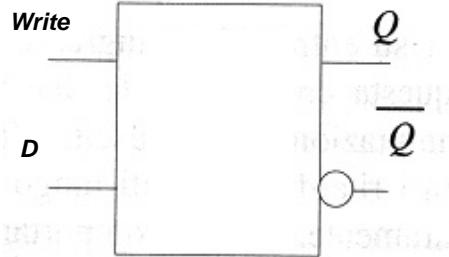
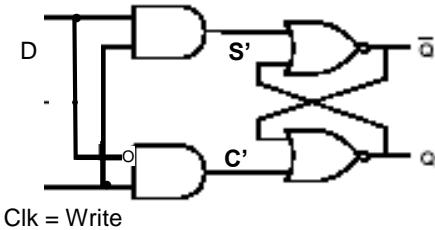


L'ingresso "clock" del bistabile viene utilizzato come segnale di write

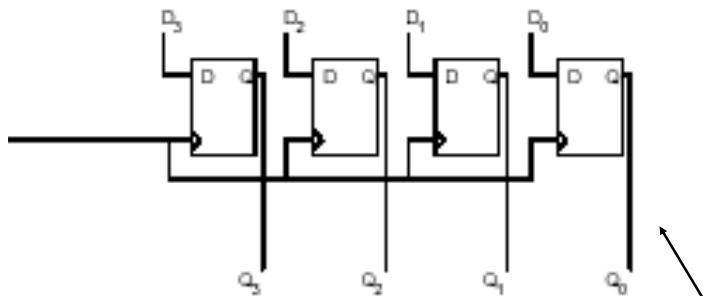
E' trasparente quando Write = 1

Se Write = 1 $Q_{t+1} = D$

Se Write = 0 $Q_{t+1} = Q_t$



Registri



Un registro a 4 bit.

Memorizza 4 bit.

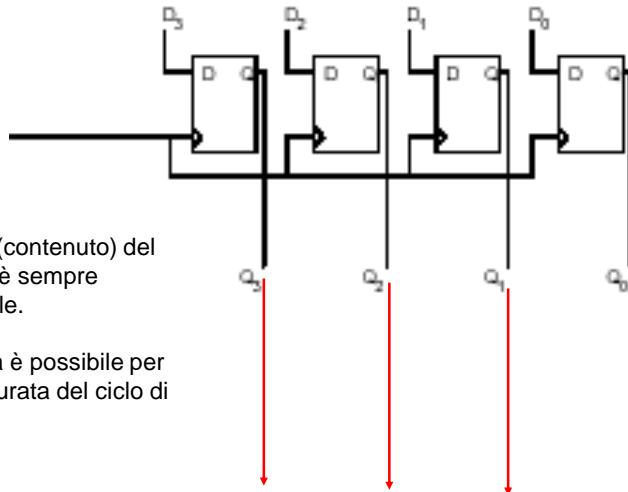
Tutti gli elementi di memoria ricevono lo stesso segnale di write

Latch o flip-flop
di tipo D

NB Non è un registro a scorrimento (shift register!)



Lettura di un registro



A.A. 2023-2024

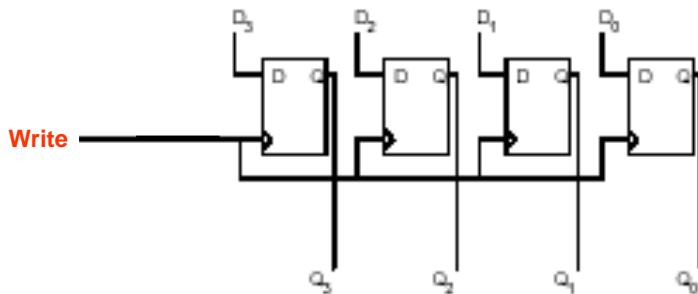
33/48

<http://borghese.di.unimi.it/>

Scrittura di un registro



Ad ogni colpo di clock lo stato del registro assume il valore dell'ingresso dati.



Cosa occorre modificare perché il registro venga scritto quando serve?

Introdurre una sorta di “*apertura del cancello (chiusura circuito)*”.

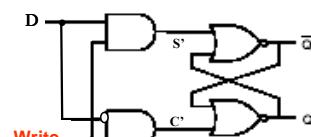
Può essere sincronizzata o meno con il clock.

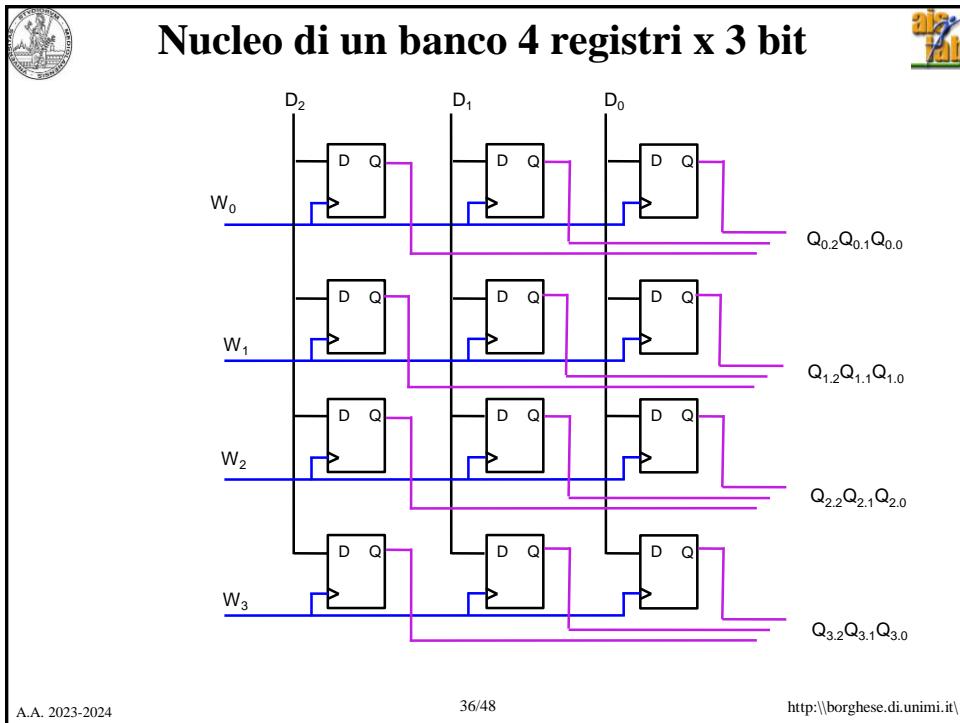
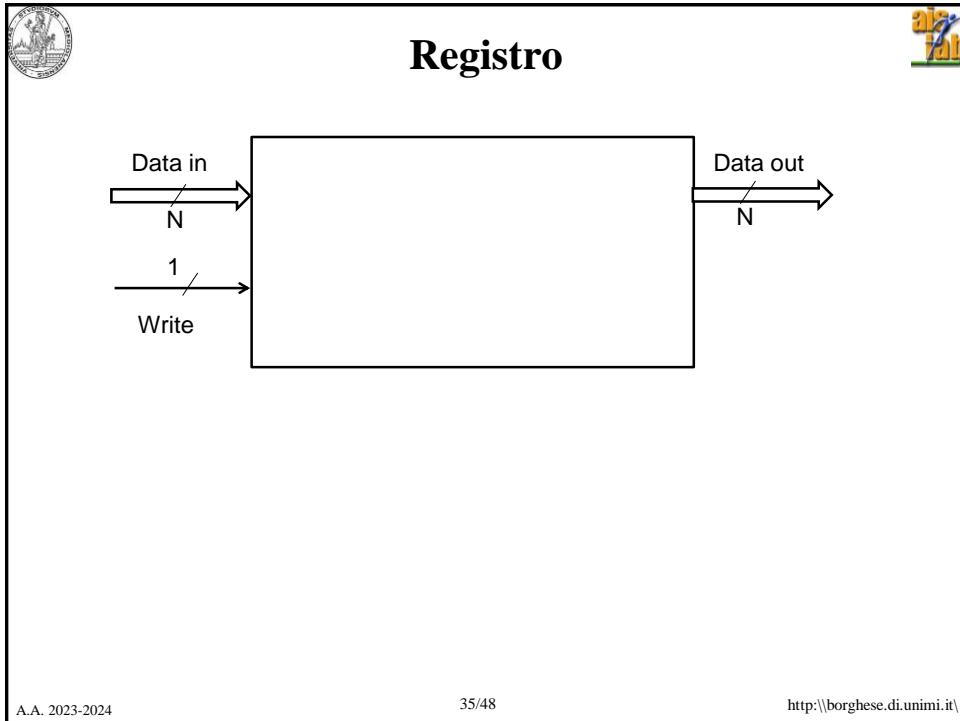
3

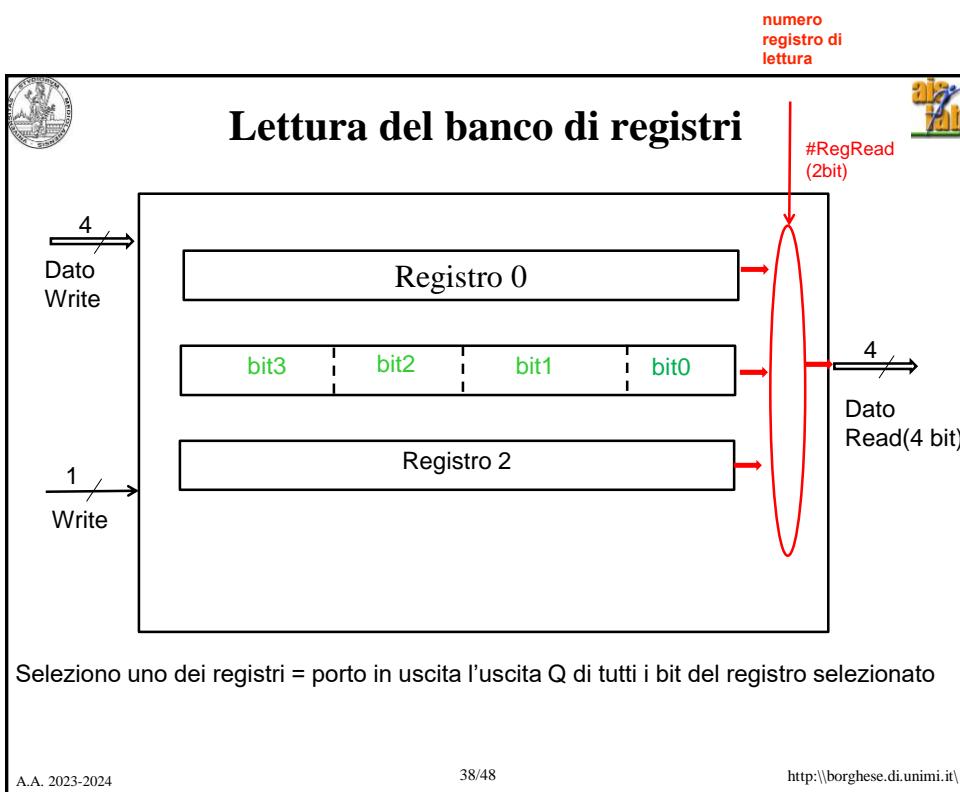
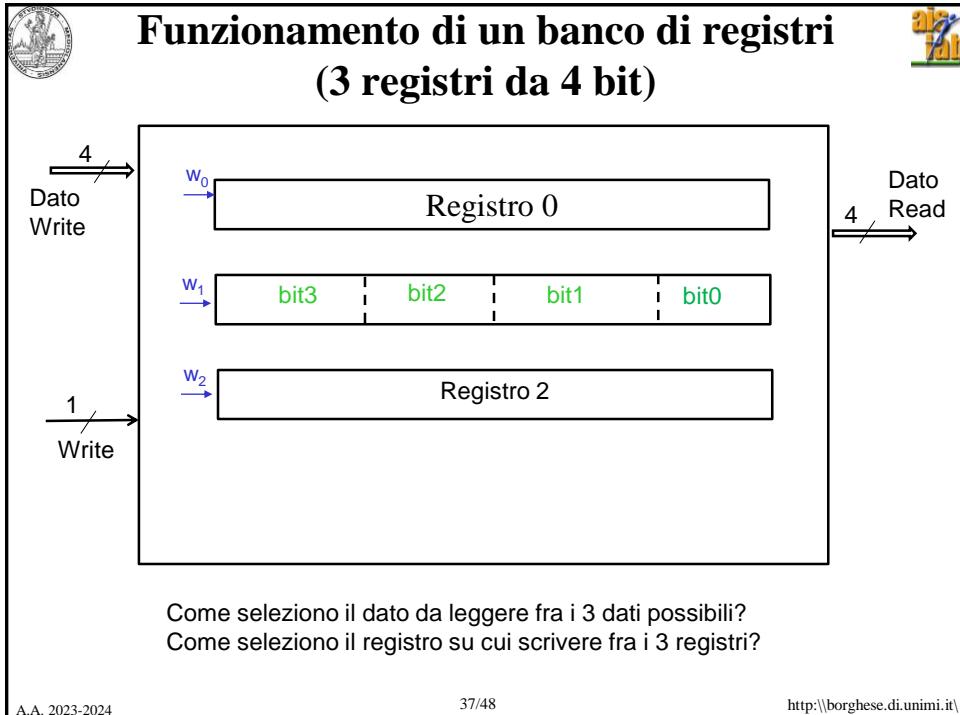
Il segnale di write apre il passaggio al contenuto di D attraverso il latch. Quando il segnale di Write è a zero, lo stato non varia.

A.A. 2023-2024

34/48

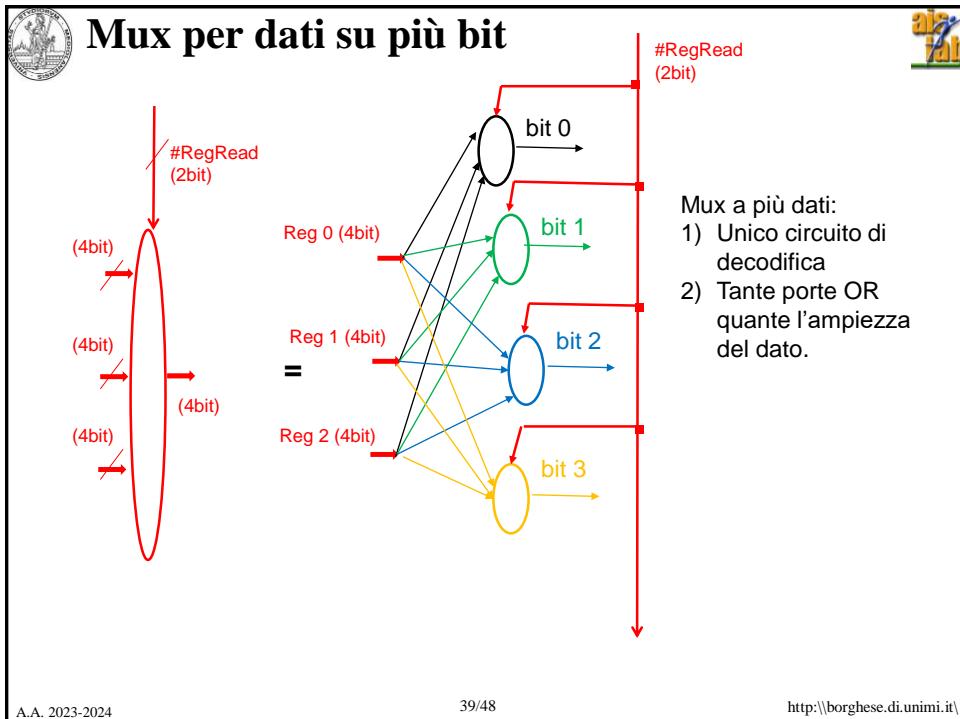






LETTURA

sono 4 multiplexer ad 1 bit

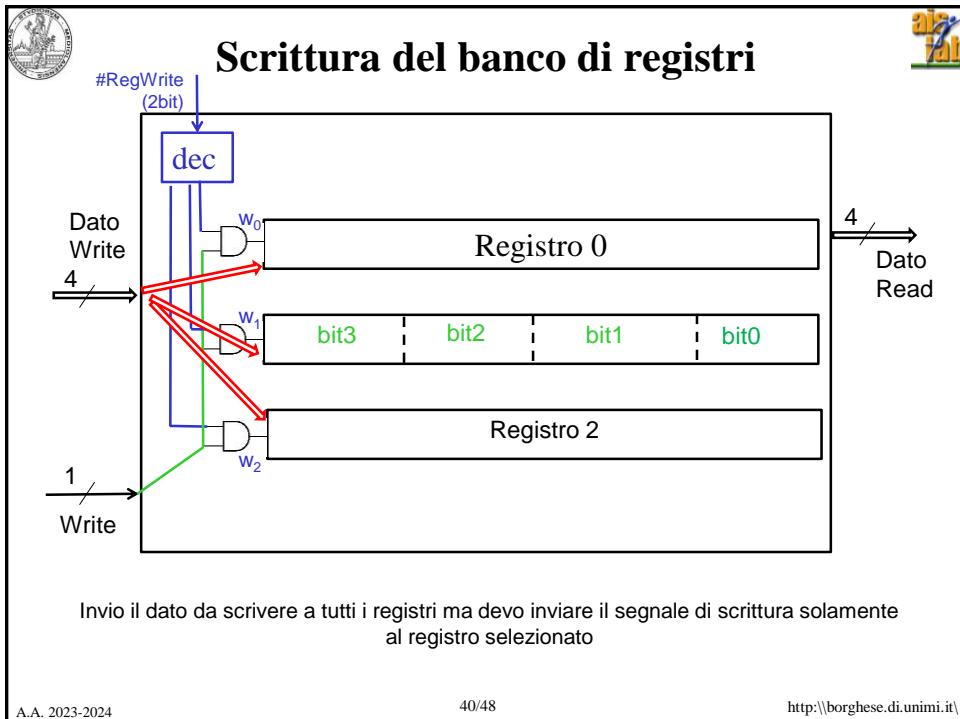


A.A. 2023-2024

39/48

<http://borghese.di.unimi.it/>

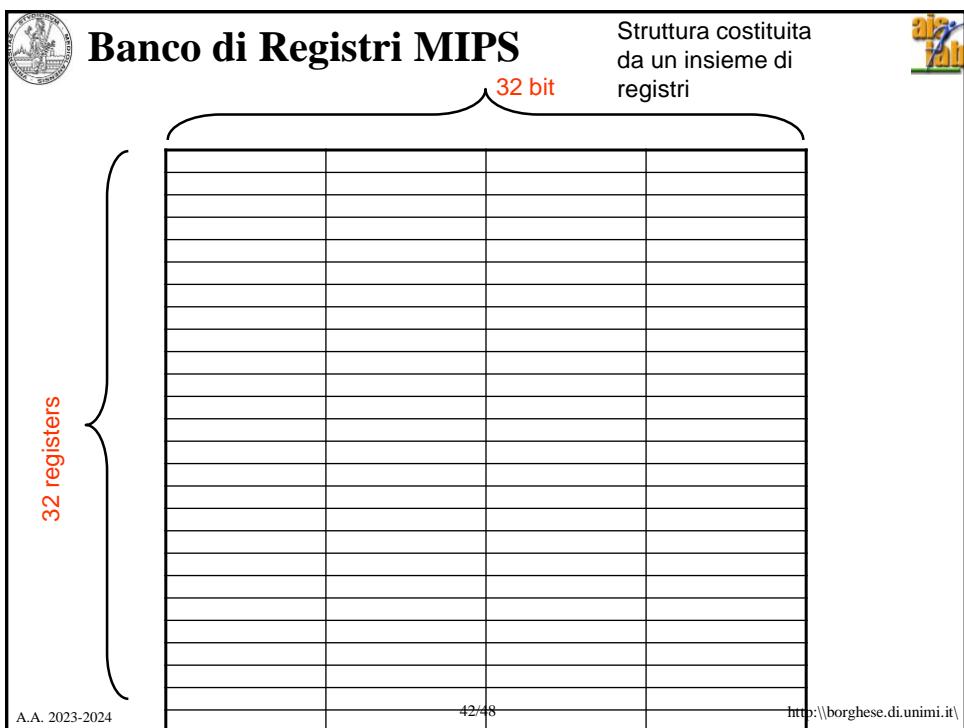
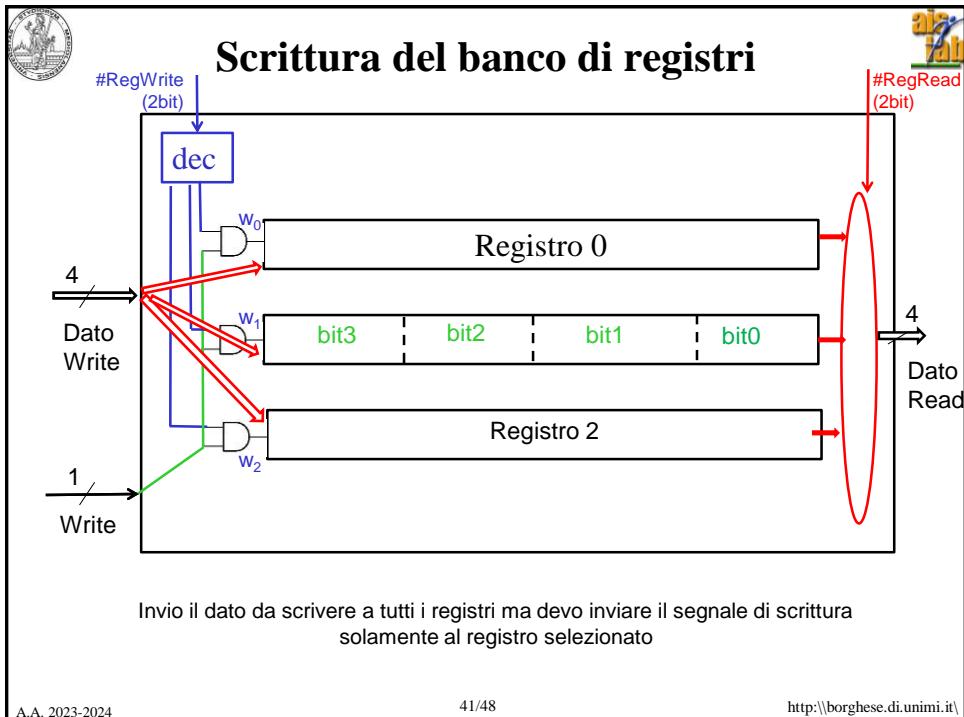
SCRITTURA

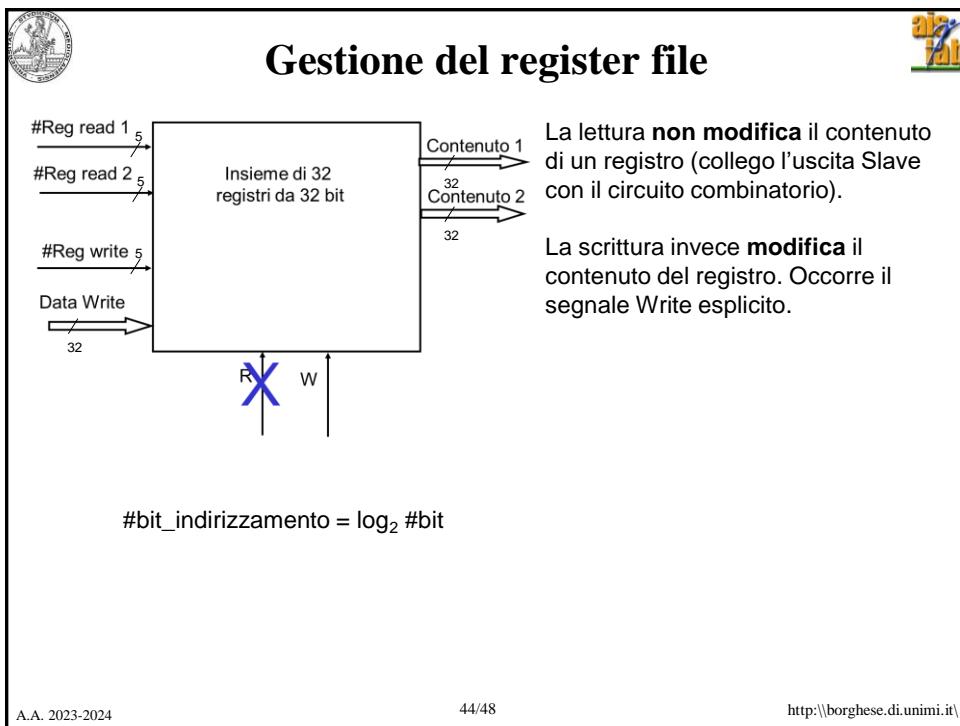
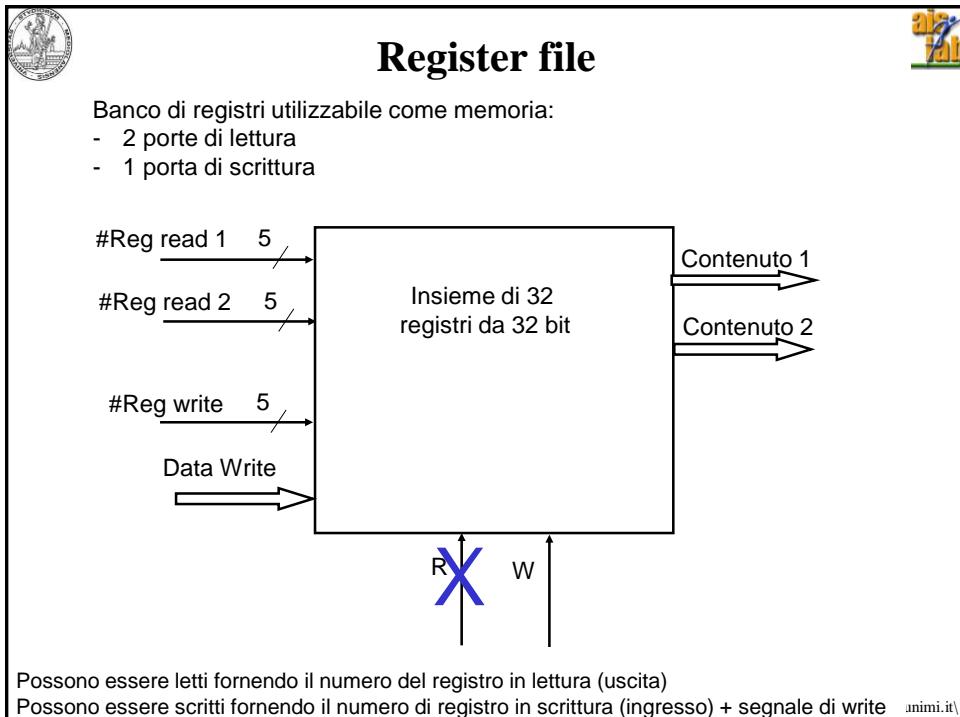


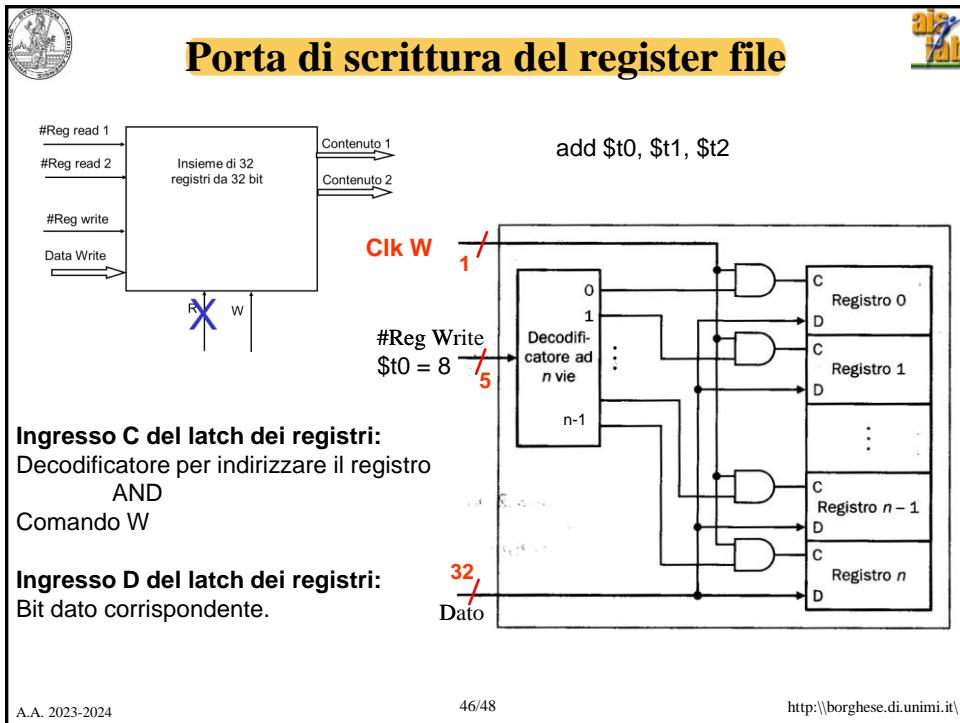
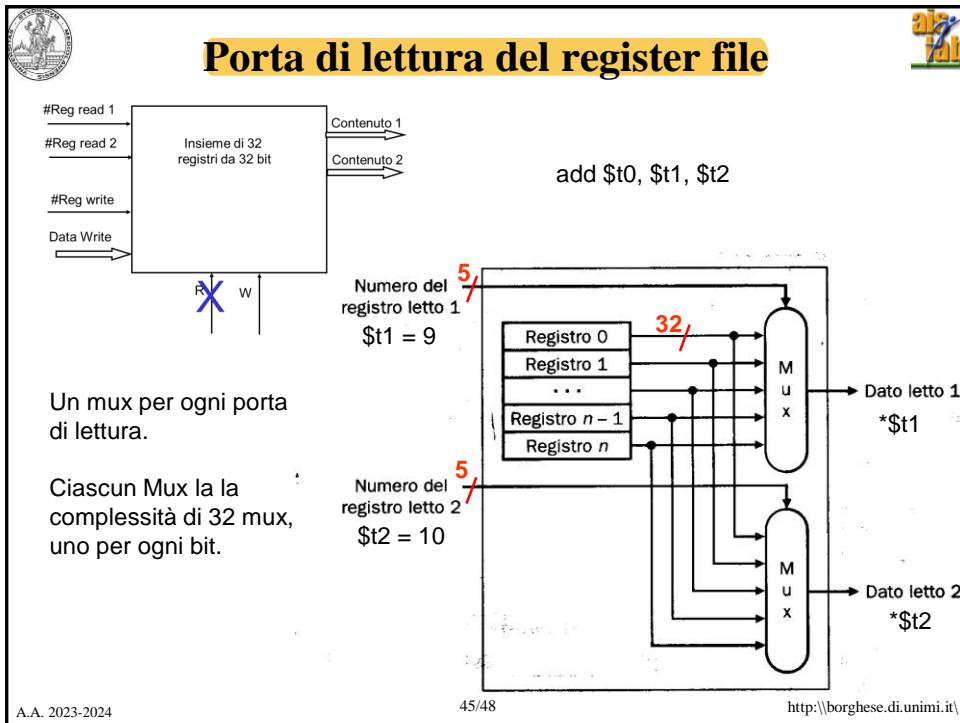
A.A. 2023-2024

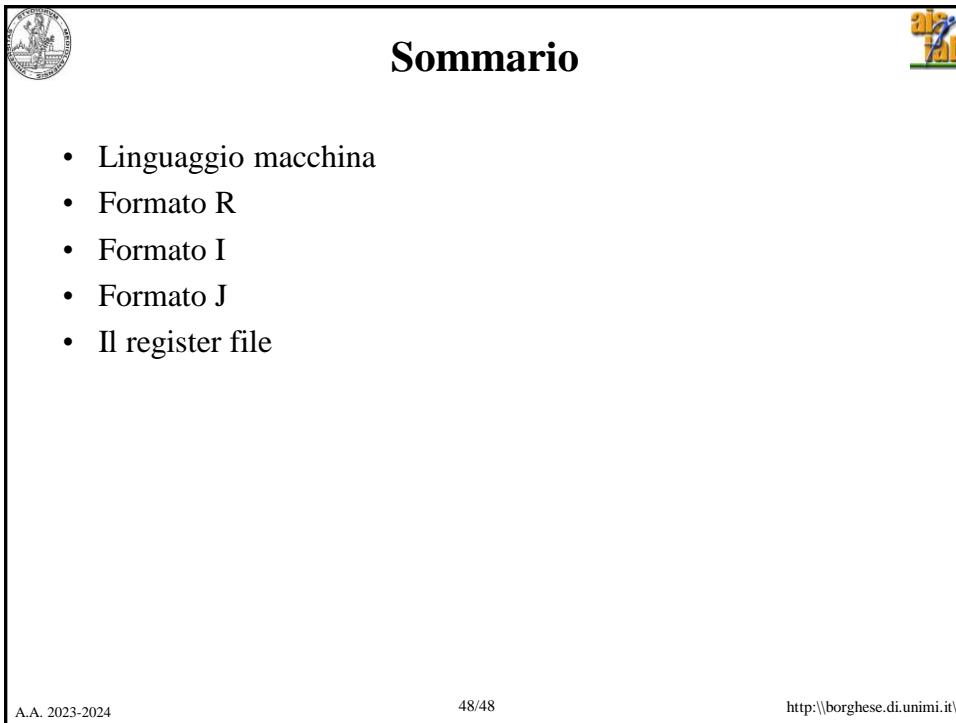
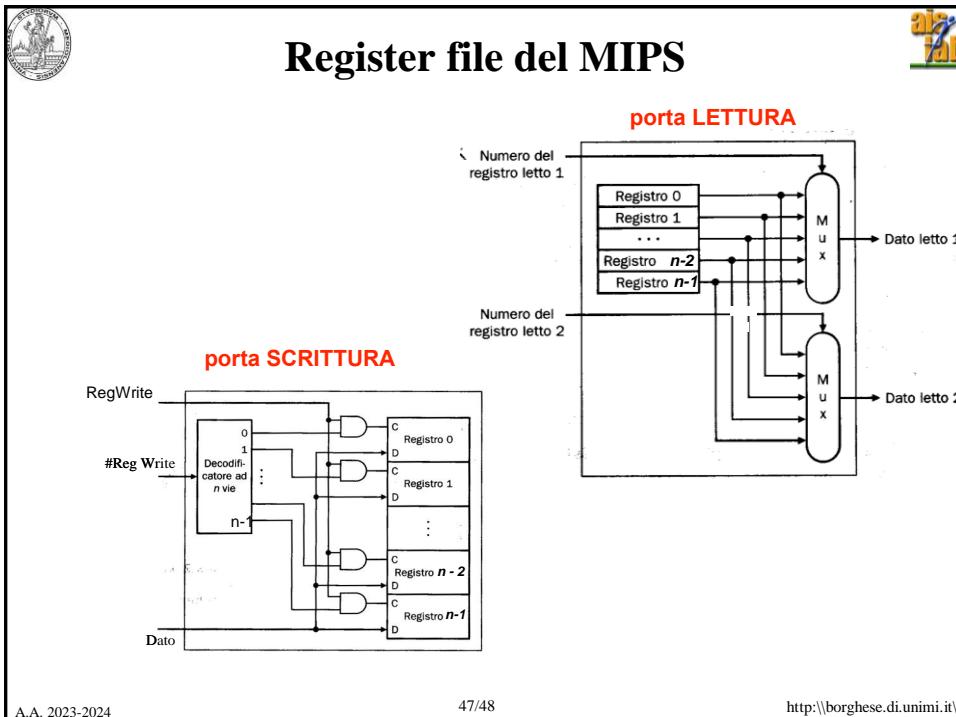
40/48

<http://borghese.di.unimi.it/>











CPU a singolo ciclo

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.



Sommario

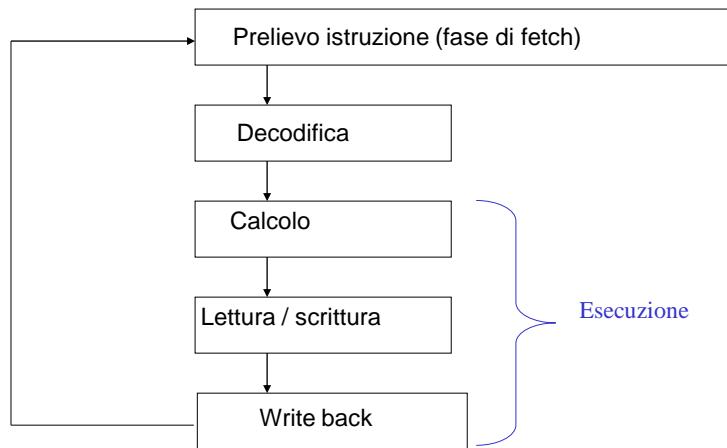
Introduzione alla CPU

CPU per le istruzioni di tipo R

CPU per le istruzioni di tipo I



Ciclo di esecuzione di un'istruzione MIPS



Obiettivo



Costruzione di una CPU completa che sia in grado di eseguire:

- Istruzioni logico-matematiche di tipo R (e.g. add, sub, and....). e.g. add \$t0, \$t1, \$t2).
- Istruzioni logico-matematiche di tipo I (e.g. addi, ori...) e.g. addi \$t0, \$t1, 24.
- Accesso alla memoria in lettura (lw) o scrittura (sw). e.g. lw \$t0, 24(\$t1)
- Istruzioni di salto condizionato (branch). e.g. beq \$t0, \$t1, etichetta
- Istruzioni di salto incondizionato (jump). e.g. j etichetta



Codifica delle istruzioni



- Tutte le istruzioni MIPS hanno la stessa dimensione (**32 bit**) – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
 - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di **3** tipi (formati):
 - **Tipo R (register)** – **Lavorano prevalentemente su 3 registri.**
 - Istruzioni aritmetico-logiche.
 - **Tipo I (immediate)** – **Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante.**
 - Istruzioni di accesso alla memoria o operazioni con una costante.
 - **Tipo J (jump)** – **Lavora senza registri: codice operativo + indirizzo di salto.**
 - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt			Indirizzo / costante
J	op				Indirizzo / costante	



I componenti di un'architettura



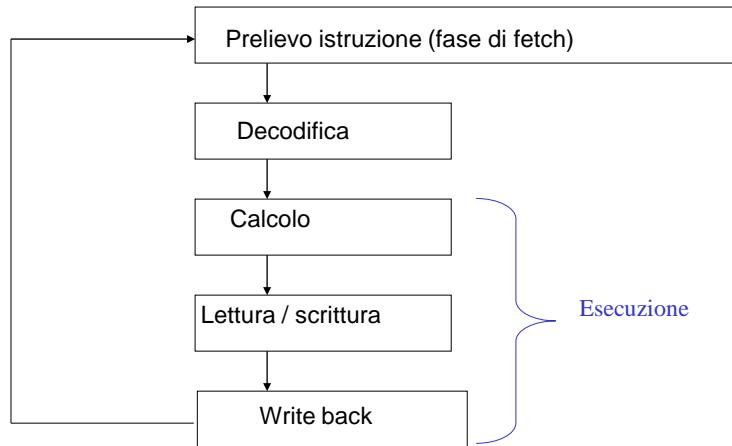
CPU

- Banco di registri (*Register File*) ad accesso rapido, in cui memorizzare i dati di utilizzo più frequente. Il tempo di accesso ai registri è circa 10 volte più veloce del tempo di accesso alla memoria principale.
- Registro *Program counter (PC)*. Contiene l'indirizzo dell'istruzione corrente da aggiornare durante l'evoluzione del programma, in modo da prelevare dalla memoria la corretta sequenza di istruzioni;
- Registro *Instruction Register (IR)*. Contiene l'istruzione in corso di esecuzione. Questo registro verrà utilizzato più avanti nelle architetture dotate di pipe-line.
- Unità per l'esecuzione delle operazioni aritmetico-logiche (*Arithmetic Logic Unit - ALU*). I dati forniti all'*ALU* possono provenire da registri oppure direttamente dalla memoria, a seconda delle modalità di indirizzamento previste;
- Unità aggiuntive per elaborazioni particolari come unità aritmetiche per dati in virgola mobile (*Floating Point Unit - FPU*), sommatori ausiliari, ecc.;
- *Unità di controllo*. Controlla il flusso e determina le operazioni di ciascun blocco.

MEMORIA PRINCIPALE



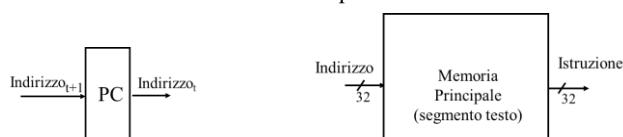
Ciclo di esecuzione di un'istruzione MIPS



Lettura dell'istruzione (fetch)



- Istruzioni e dati risiedono nella memoria principale, dove sono stati caricati attraverso un'unità di ingresso.
- L'esecuzione di un programma inizia quando il registro PC punta alla (contiene l'indirizzo della) prima istruzione del programma in memoria (segmento testo).
- Il segnale di controllo per la lettura (READ) viene inviato alla memoria.
- Trascorso il tempo necessario alla lettura dalla memoria, la parola indirizzata (in questo caso la prima istruzione del programma) si troverà nel registro IR.
- Il contenuto del PC viene incrementato in modo da puntare all'istruzione successiva.



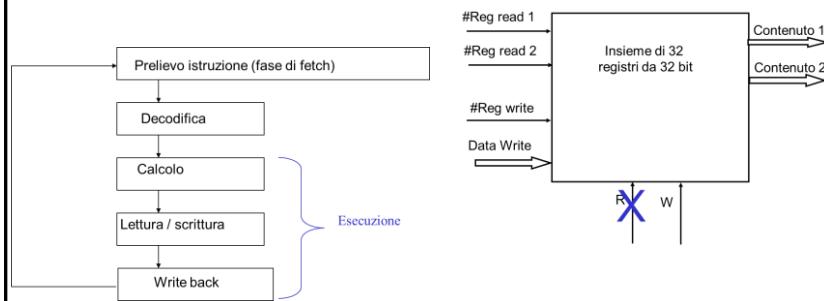


Decodifica dell'istruzione



- L'istruzione contenuta nel registro IR viene decodificata per essere eseguita. Alla fase di decodifica corrisponde la predisposizione della CPU (apertura delle vie di comunicazione appropriate) all'esecuzione dell'istruzione.
- In questa fase vengono anche recuperati gli operandi. Nelle architetture MIPS gli operandi possono essere solamente nel Register File oppure letti dalla memoria.

Architetture LOAD/STORE: Le istruzioni di caricamento dalla memoria sono separate da quelle aritmetico/logiche.

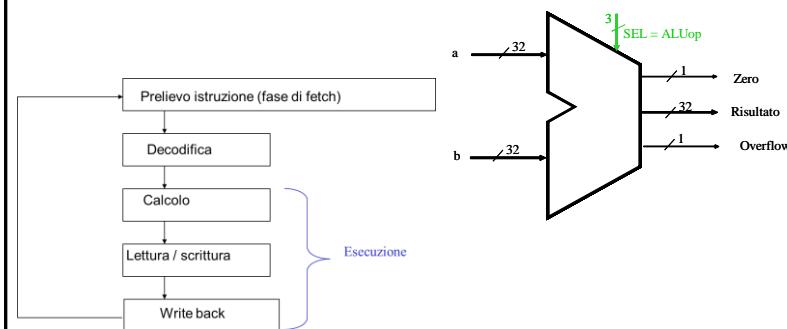


Calcolo dell'istruzione (execute - calcolo)



Viene selezionata all'interno della ALU l'operazione prevista dall'istruzione e determinata in fase di decodifica dell'istruzione.

Tra le operazioni previste, c'è anche la formazione dell'indirizzo di memoria da cui leggere o su cui scrivere un dato.





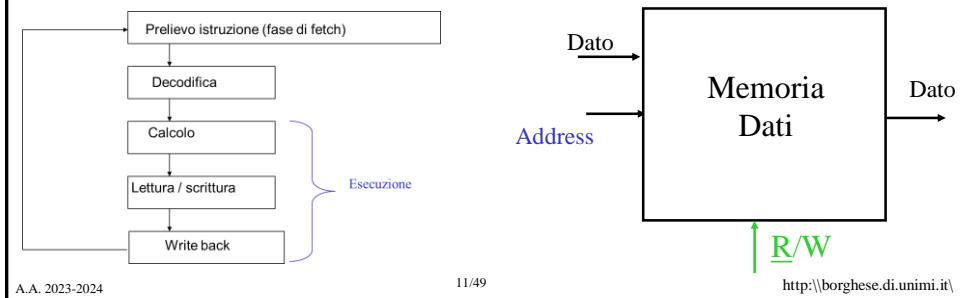
Lettura / Scrittura in memoria dati



In questa fase il dato presente in un registro, viene scritto in memoria oppure viene letto dalla memoria un dato e trasferito ad un registro.

Questa fase non è richiesta da tutte le istruzioni

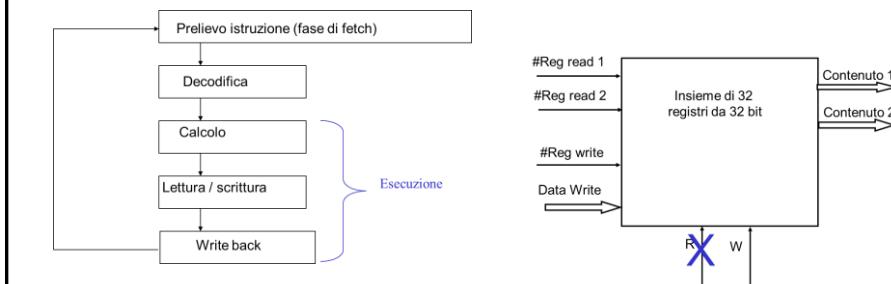
Nel caso particolare di Architetture LOAD/STORE, quali MIPS, le istruzioni di caricamento dalla memoria sono separate da quelle aritmetico/logiche. Se effettuo una Lettura / Scrittura, **non** eseguo operazioni aritmetico logiche sui dati.



Scrittura in register file (write-back)



- Il risultato dell'operazione può essere memorizzato nei registri ad uso generale oppure in memoria.
- Non appena è terminato il ciclo di esecuzione dell'istruzione corrente (termina la fase di Write Back), si preleva l'istruzione successiva dalla memoria.





Come funziona una CPU?



- Usa un registro, il Program Counter (PC) per ottenere l'indirizzo dell'istruzione.
- Preleva l'istruzione dalla memoria e la inserisce nell'IR.
- L'UC capisce di che tipo di istruzione si tratta (decodifica).
 - usa l'istruzione stessa per decidere cosa fare esattamente.
- Legge il contenuto dei registri.

Da qui le istruzioni si differenziano.

- Calcolo: utilizzo dell'ALU dopo aver letto i registri:
 - per calcolare l'indirizzo in memoria.
 - per eseguire un'operazione logico-arithmetica.
 - per effettuare test (uguaglianza, disuguaglianza, <...).
- Accesso alla memoria (se richiesto). **LOAD-STORE**
- Scrittura del risultato nel register file (se richiesto).



Sommario



Introduzione alla CPU

CPU per le istruzioni di tipo R

CPU per le istruzioni di tipo I

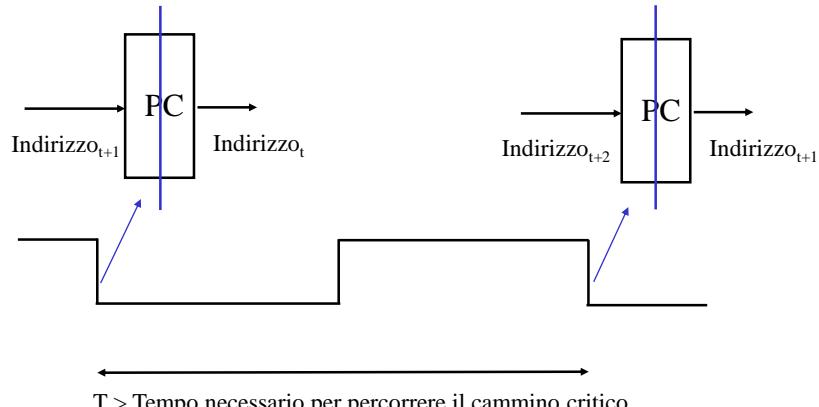
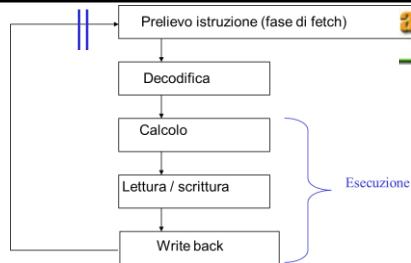


Temporizzazione



1 istruzione per ciclo di clock.

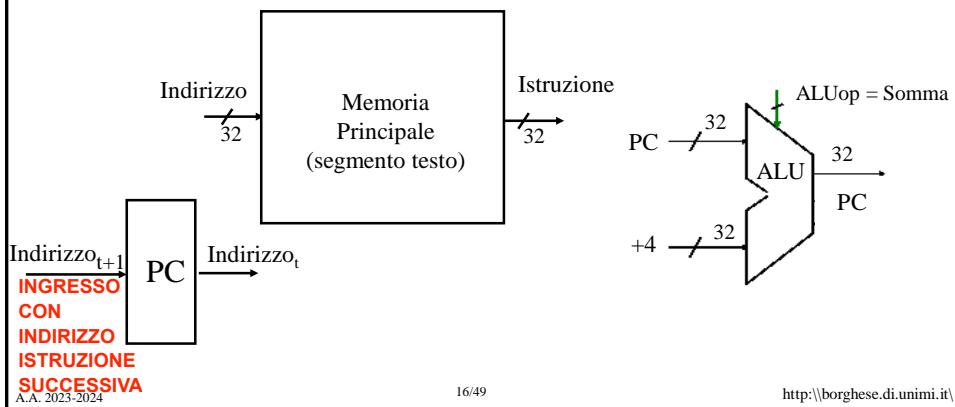
Temporizzazione del PC.



Fase di fetch



- 1) Memorizzare l'indirizzo dell'istruzione nel PC.
- 2) Leggere l'istruzione dalla memoria.
- 3) Aggiornare l'indirizzo in modo che in PC sia contenuto l'indirizzo dell'istruzione successiva.



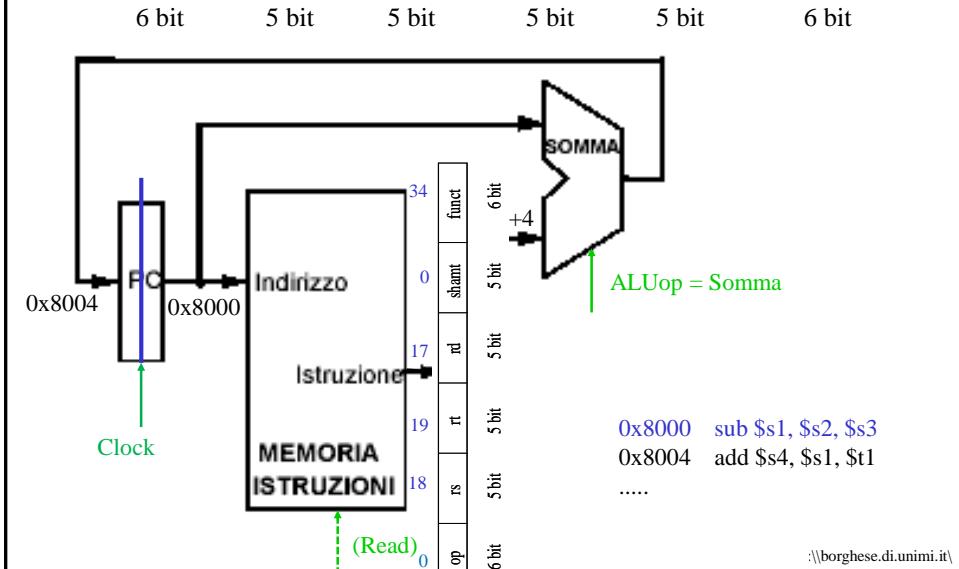


Circuito della fase di fetch



R

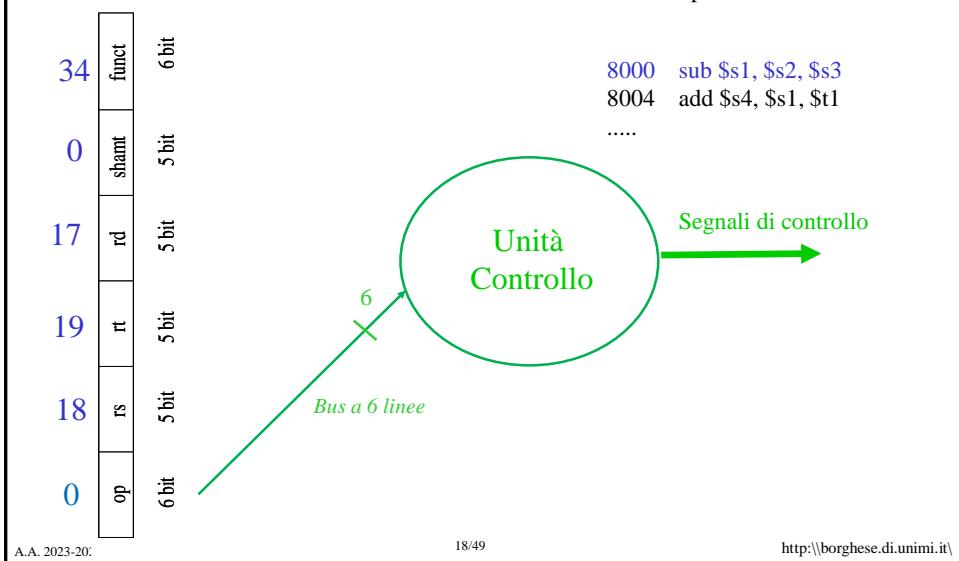
op = 0	rs = 18	rt = 19	rd = 17	Shamt=0	funct=34
--------	---------	---------	---------	---------	----------



Fase di decodifica



- Leggo l'istruzione e genero i segnali di controllo opportuni.
Bus che connette i 6 bit del CodOp con la UC

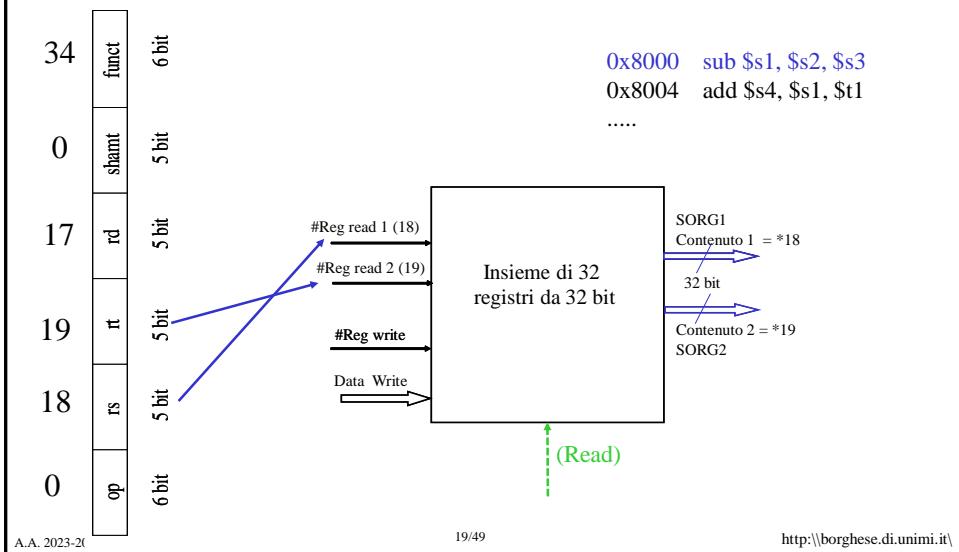




Lettura dei registri (istruzioni di tipo R)



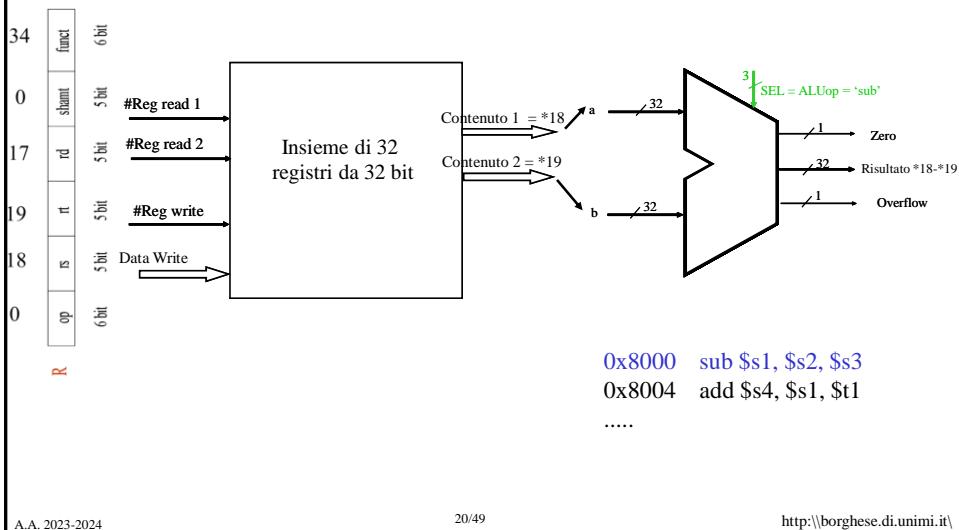
2) Leggo il contenuto dei registri. 2 bus che collegato i campi rs e rt con gli input corrispondenti del register file



Fase di Calcolo (tipo R)



18-19





Fase di Write back (tipo R)



R

op = 0	rs = 18	rt = 19	rd = 17	Shamt=0	funct=34
--------	---------	---------	---------	---------	----------

6 bit 5 bit 5 bit 5 bit 5 bit 6 bit

#Reg read 1 (18)

#Reg read 2 (19)

#Reg write (17)

Data Write

Insieme di 32 registri da 32 bit

Contenuto 1 = *18

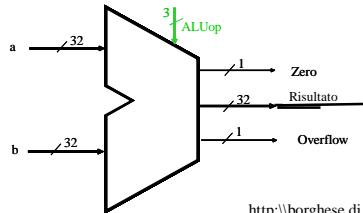
Contenuto 2 = *19

0x8000 sub \$s1, \$s2, \$s3
0x8004 add \$s4, \$s1, \$t1

.....

W

Risultato *18-*19



A.A. 2023-2024

21/49

<http://borghese.di.unimi.it/>

Porta di scrittura del register file

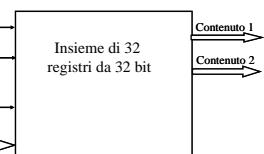


#Reg read 1

#Reg read 2

#Reg write

Contenuto Write



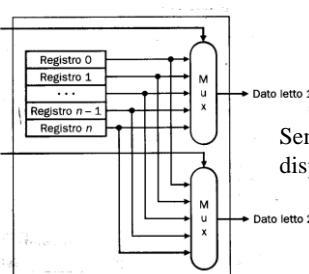
R

X

W

Numero del registro letto 1

Numero del registro letto 2

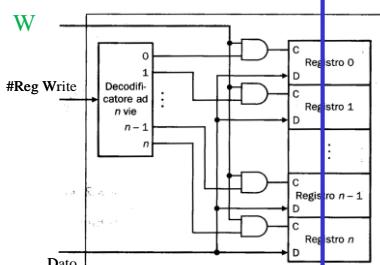


Sempre disponibili

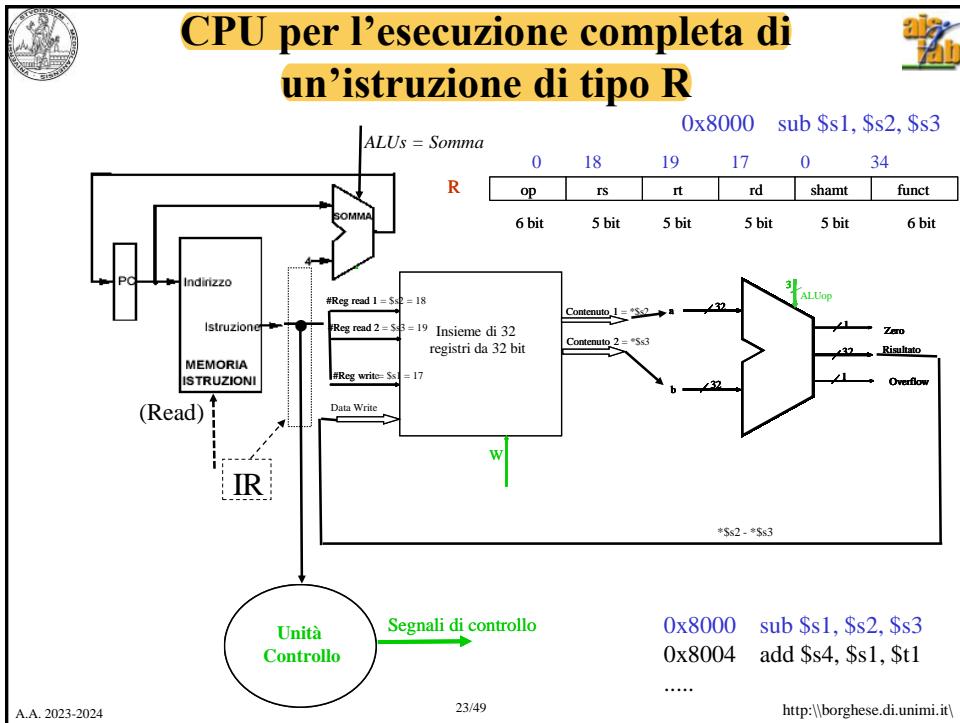
$$A' = A + B$$

A'

A



NB Utilizzo registri flip-flop in modo da potere leggere / scrivere nello stesso ciclo di clock (scrivo nel master nella fase di WB e leggo dallo slave in fase di decodifica. La commutazione da master a slave è pilotata dal clock.



Sommario

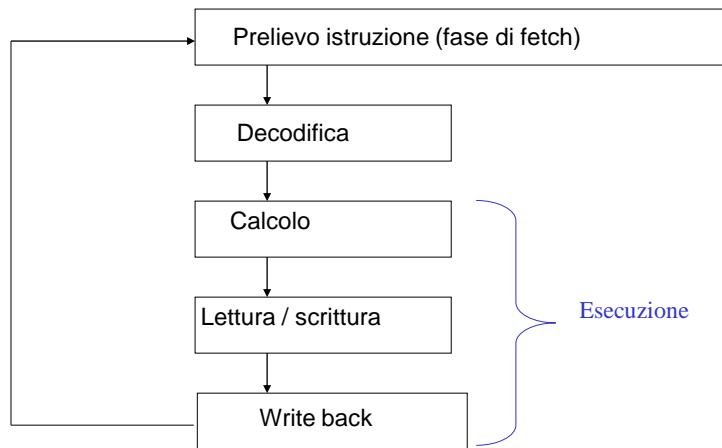
Introduzione alla CPU

CPU per le istruzioni di tipo R

CPU per le istruzioni di tipo I



Ciclo di esecuzione di un'istruzione MIPS



Codifica delle istruzioni



- Tutte le istruzioni MIPS hanno la **stessa dimensione (32 bit)** – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
 - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di 3 tipi (formati):
 - Tipo R (register)** – Lavorano su 3 registri.
 - Istruzioni aritmetico-logiche.
 - Tipo I (immediate)** – Lavorano su 2 registri. L'istruzione è suddivisa in un gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante **indirizzo**
 - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
 - Tipo J (jump)** – Lavora senza registri: codice operativo + indirizzo di salto.
 - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	Indirizzo / costante		
J	op	indirizzo				

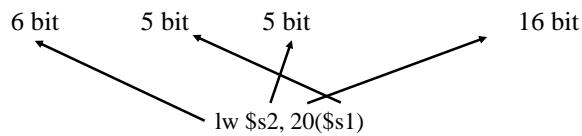


Istruzioni di tipo I: lw/sw

OP
35SOURCE
17TARGET
18COSTANTE
20

I

100011	10001	10010	0000 0000 0001 0100
--------	-------	-------	---------------------



L'indirizzo della memoria (dati) sarà:

Base = *\$s1
Offset = 20

0000 1000 0011 0001 1011 1011 1011 1011 +
0000 0000 0001 0100 +

Indirizzo finale
*\$s1 + 20

0000 1000 0011 0001 1011 1011 1100 1111



Organizzazione logica della memoria

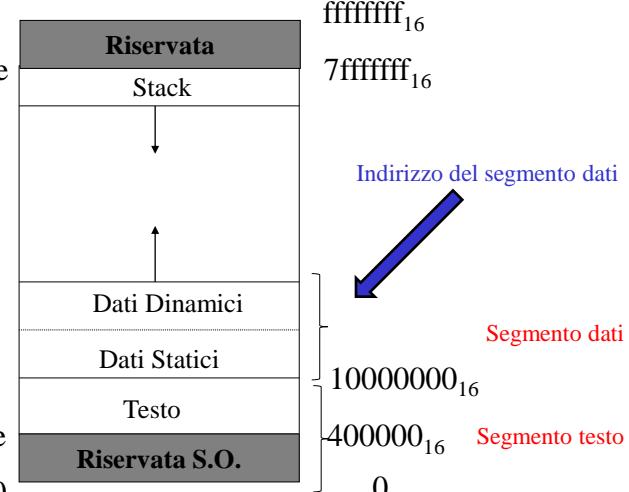


Max spazio di
indirizzamento su 32 bit è
di $2^{32} = 4\text{Gbyte}$.

$2^{28} = 256\text{Mbyte}$

4Mbyte

0



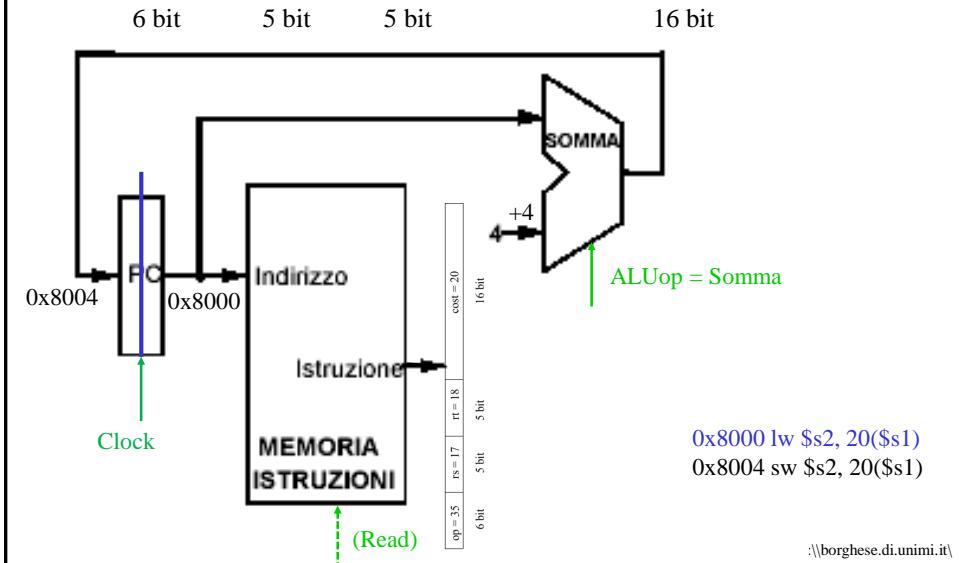


Circuito della fase di fetch



R

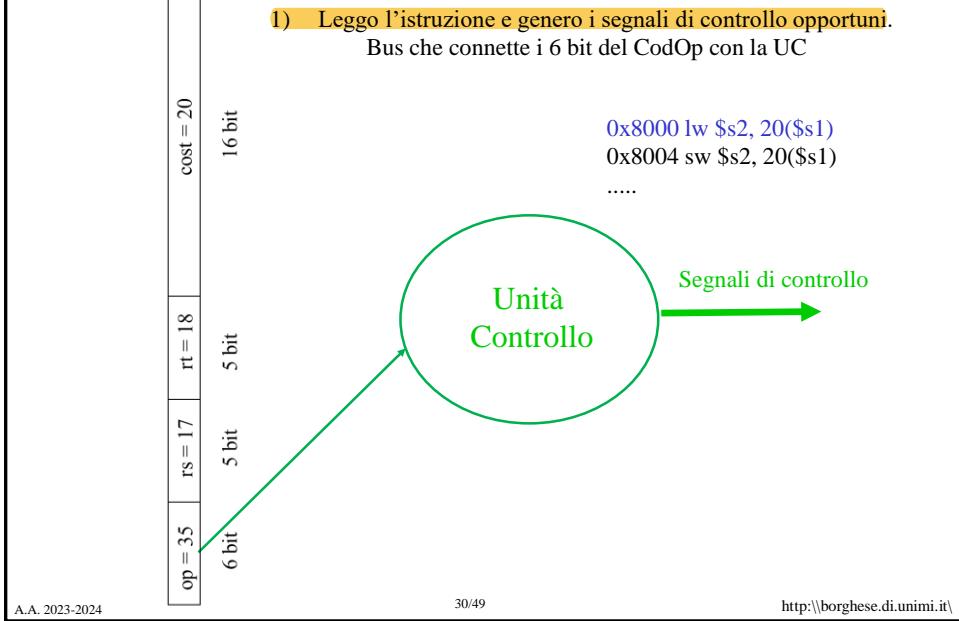
op = 35	rs = 17	rt = 18	cost = 20
---------	---------	---------	-----------



Fase di decodifica



- Leggo l'istruzione e genero i segnali di controllo opportuni.
Bus che connette i 6 bit del CodOp con la UC

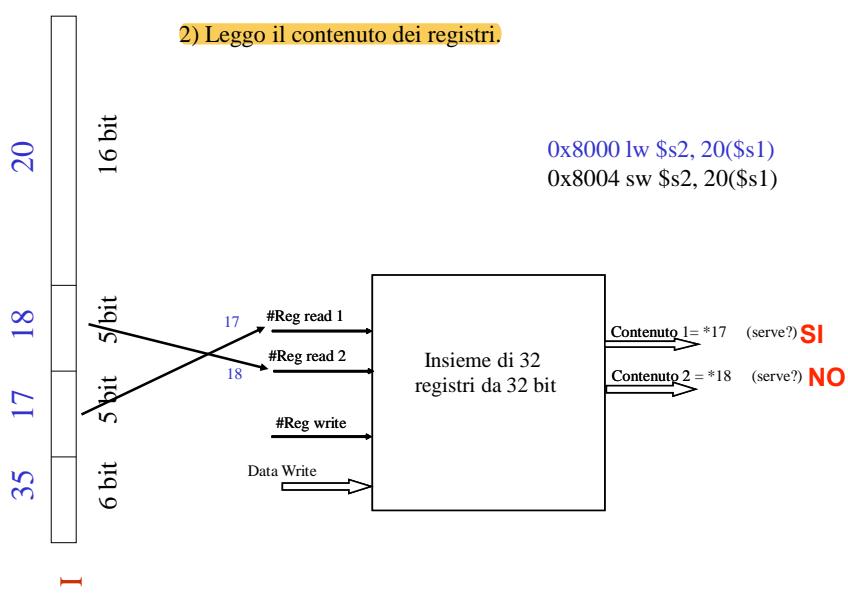




Lettura dei registri (istruzioni di tipo I)



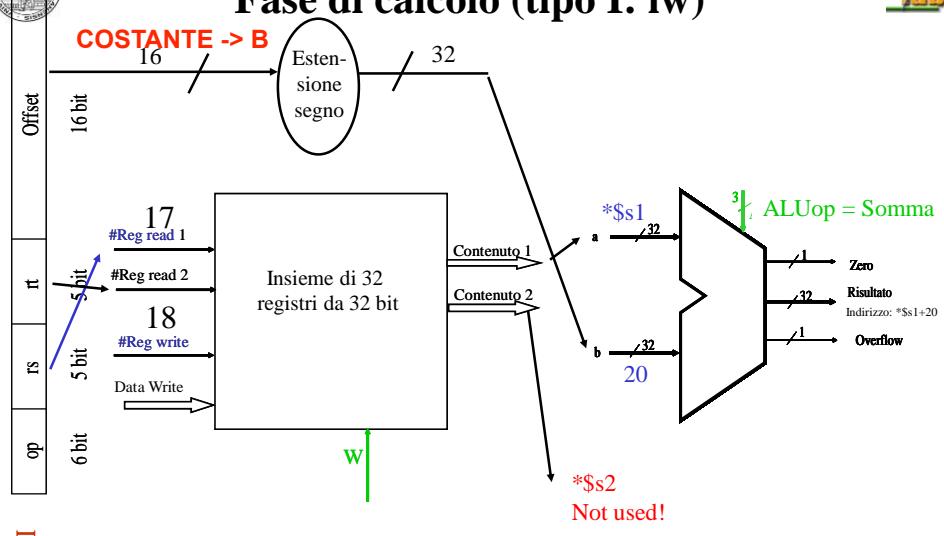
2) Leggo il contenuto dei registri



Fase di calcolo (tipo I: lw)

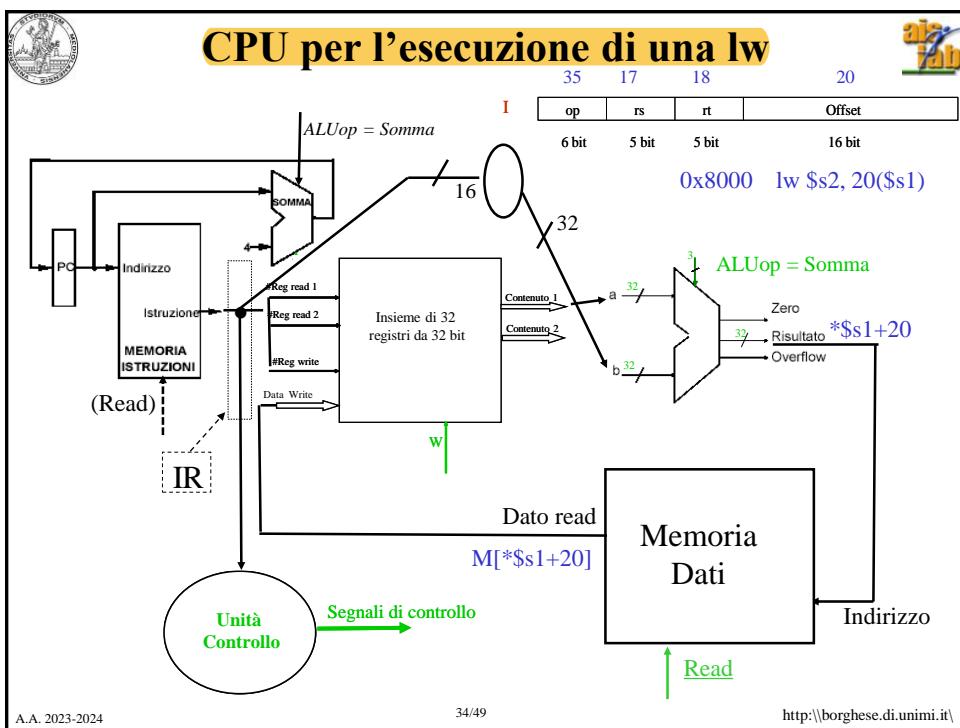
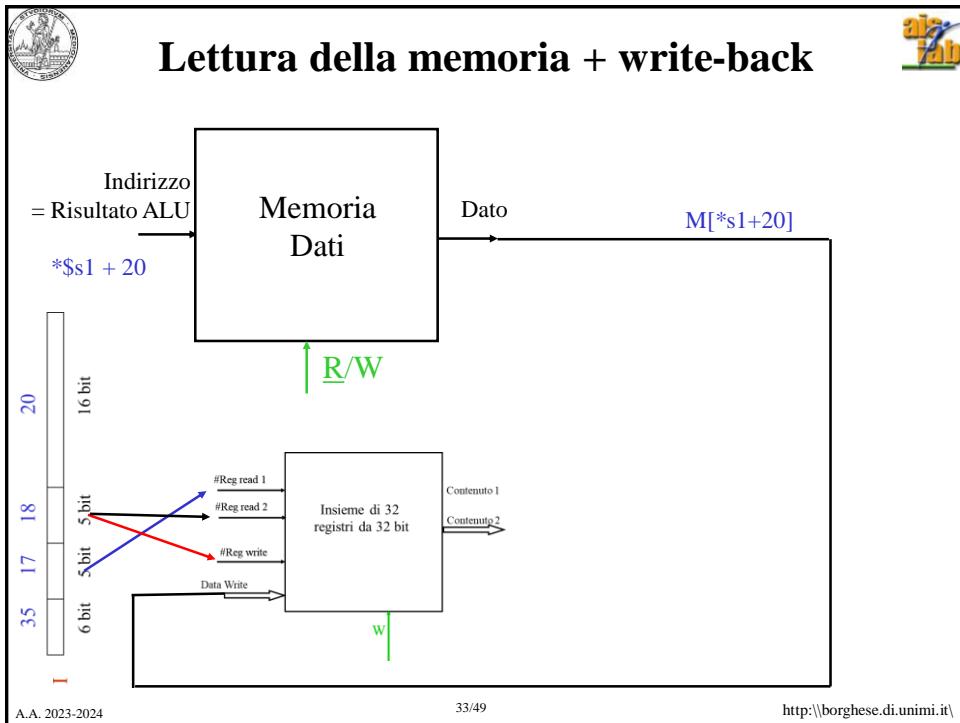


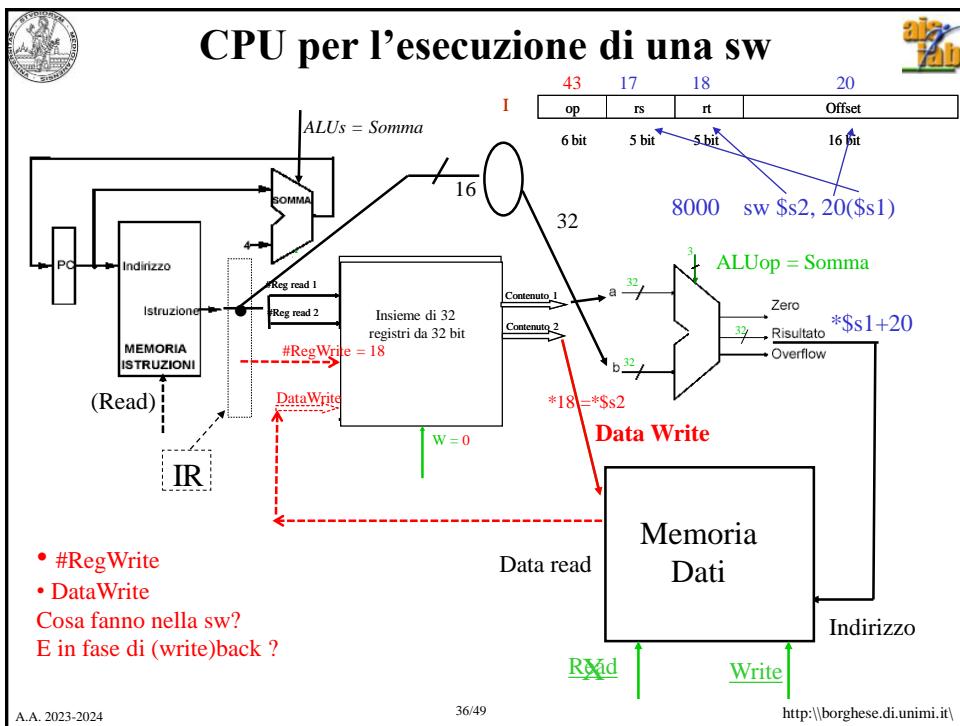
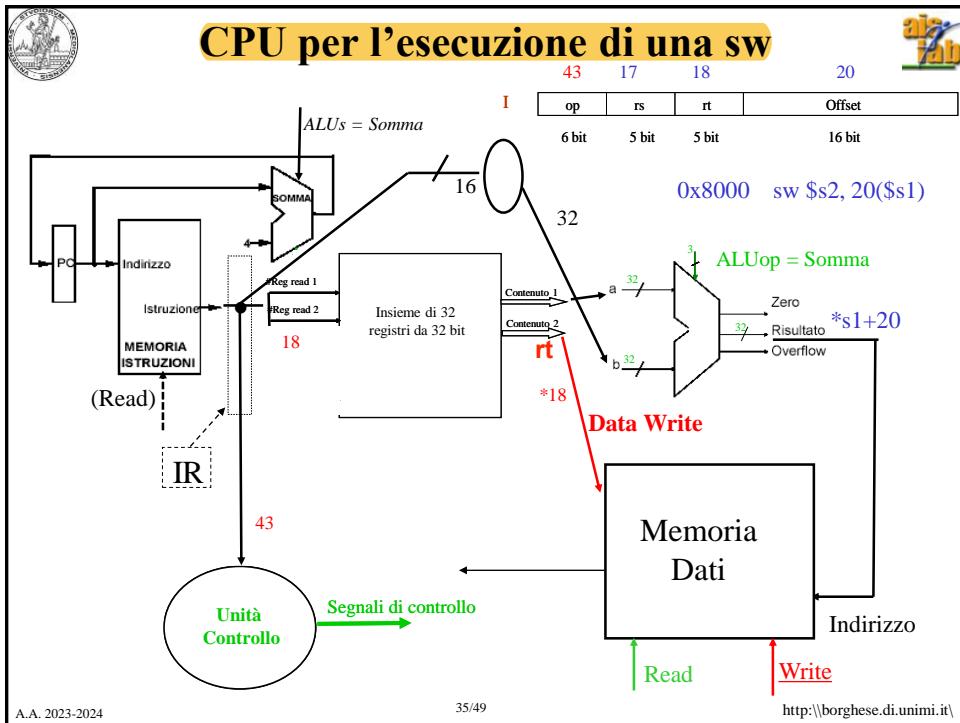
COSTANTE -> B



Il Risultato è un indirizzo della memoria: $*s1 + 20$

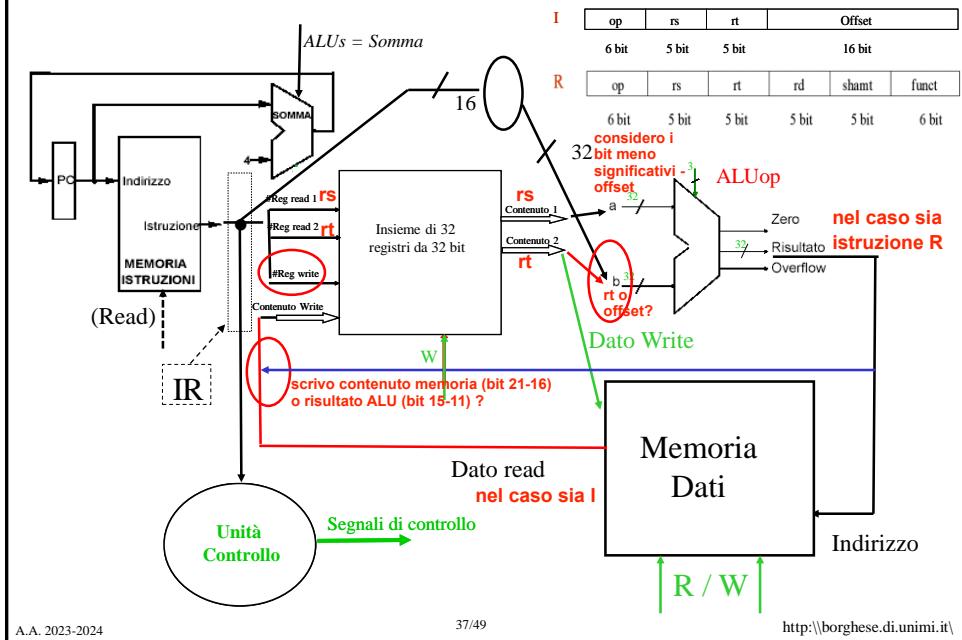
0x8000 lw \$s2, 20(\$s1)







CPU per l'esecuzione di una R, lw, sw



Commenti



Questa CPU può eseguire istruzioni di tipo R, lw, sw

Fase di fetch e decodifica è uguale per tutte e 3 i tipi di istruzioni

Le fasi di Exe, Memoria e WriteBack sono diverse:

- Le istruzioni di tipo R hanno solo Exe e WriteBack.
 - Le istruzioni di sw hanno Exe e Mem
 - Le istruzioni di lw hanno Exe, Mem e Write Back.
- Nelle istruzioni di tipo R i due operandi vengono prelevati dal RF e il risultato scritto nel RF proviene dall'uscita della ALU
 - Nelle istruzioni di sw e lw il secondo operando è la costante estesa di segno.
 - Nelle istruzioni di sw rt contiene il dato da scrivere in memoria, nelle istruzioni di lw rt conterrà il dato letto dalla memoria.
 - Nelle istruzioni di lw viene scritto nel RF il dato letto dalla memoria dati.

L'UC guida le scelte dei cammini

L'UC impone i segnali di controllo per le unità funzionali.



Istruzioni di salto condizionato



- Salti condizionati relativi:
 - **beq r1, r2, L1** (*branch on equal*)
 - **bne r1, r2, L1** (*branch on not equal*)

```
beq $s1, $s0, esci # if (s1 == s0) esci
bne $s1, $s0, esci # if (s0 ≠ s0) esci
```
- Salti condizionati relativi:
 - Il flusso sequenziale di controllo cambia solo se la condizione è vera (**beq**)
 - Il calcolo del valore dell'etichetta **L1** (**indirizzo di destinazione del salto**) avviene a partire dal **Program Counter** (PC).
 - Indirizzamento del tipo Base (PC) + Spiazzamento. **costante L1 * 4**
- Indirizzo destinazione del salto:
 - $PC_{dest} = (PC + 4) + offset * 4$

Nome campo	op	rs	rt	indirizzo
Dimensione	6-bit	5-bit	5-bit	16-bit
beq \$s1, \$s2, L1	000100	10001	10010	0000 0000 0000 1001

A.A. 2023-2024

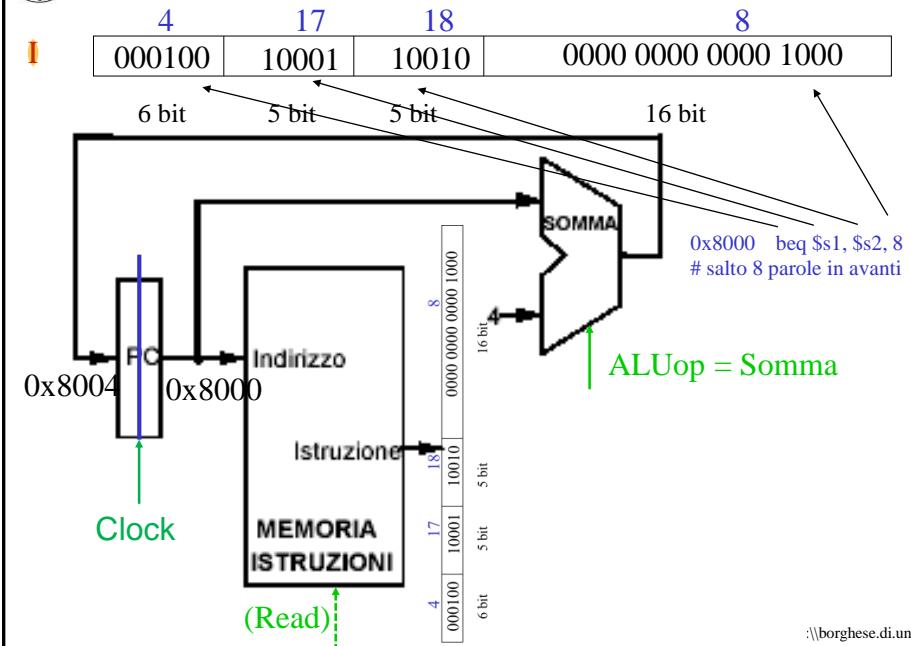
39/49

<http://borghese.di.unimi.it/>

(cond vera)
Ind. Salto
↓
(cond falsa)
+4
(procedi in sequenza)



Circuito della fase di fetch

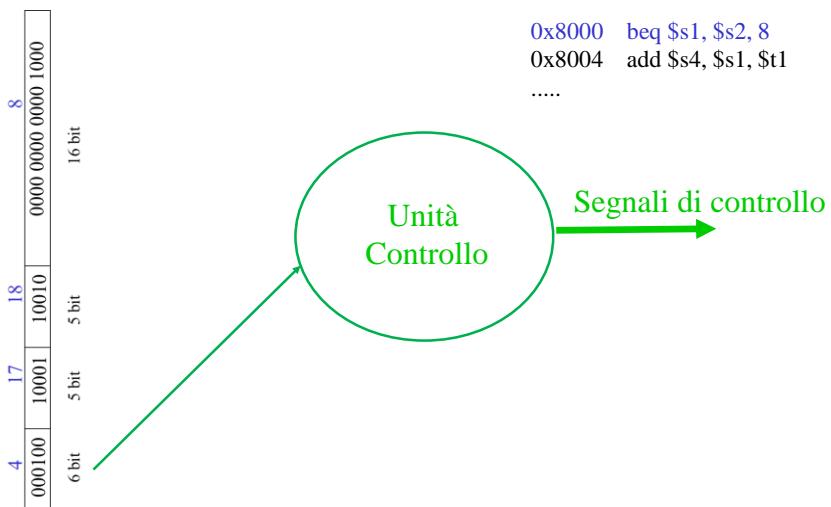
<http://borghese.di.unimi.it/>



Fase di decodifica



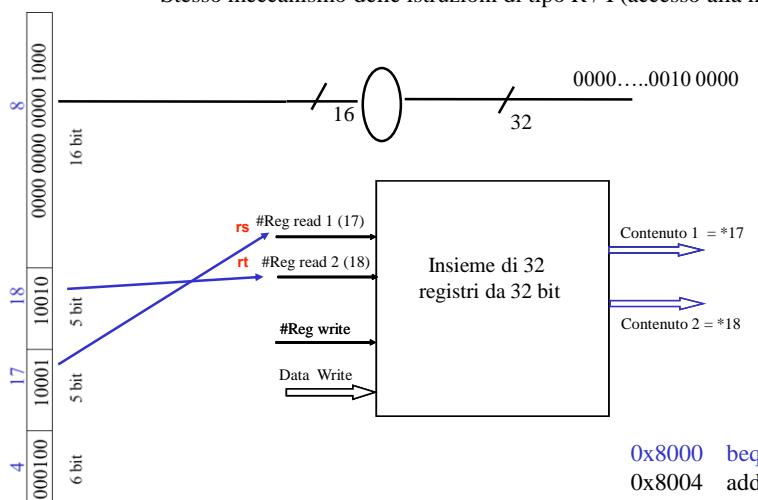
- 1) Leggo l'istruzione e genero i segnali di controllo opportuni.
Bus che connette i 6 bit del CodOp con la UC

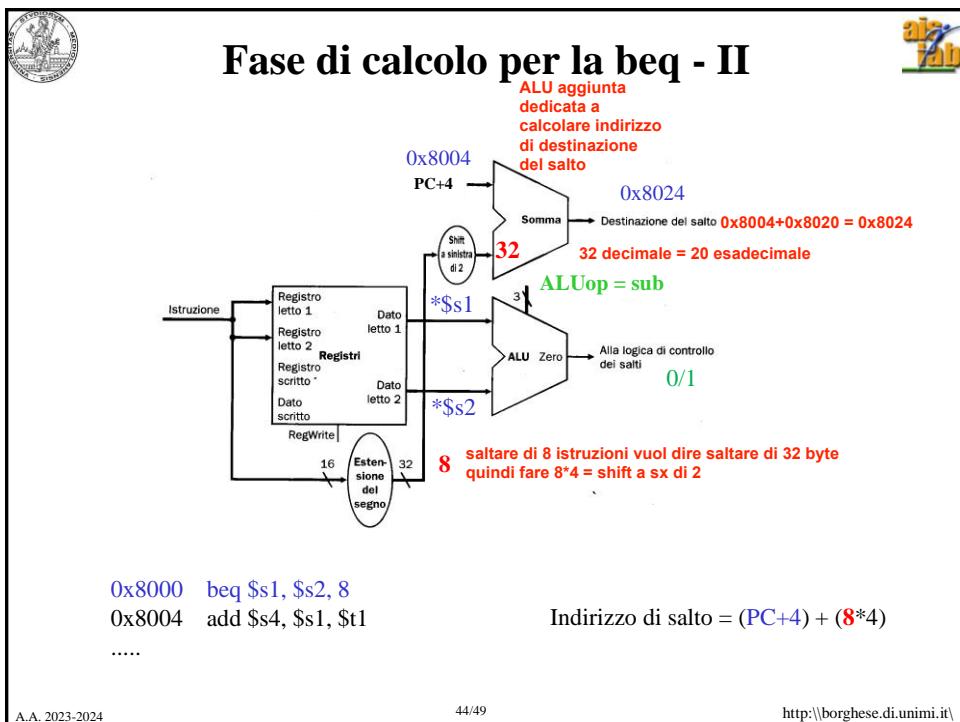
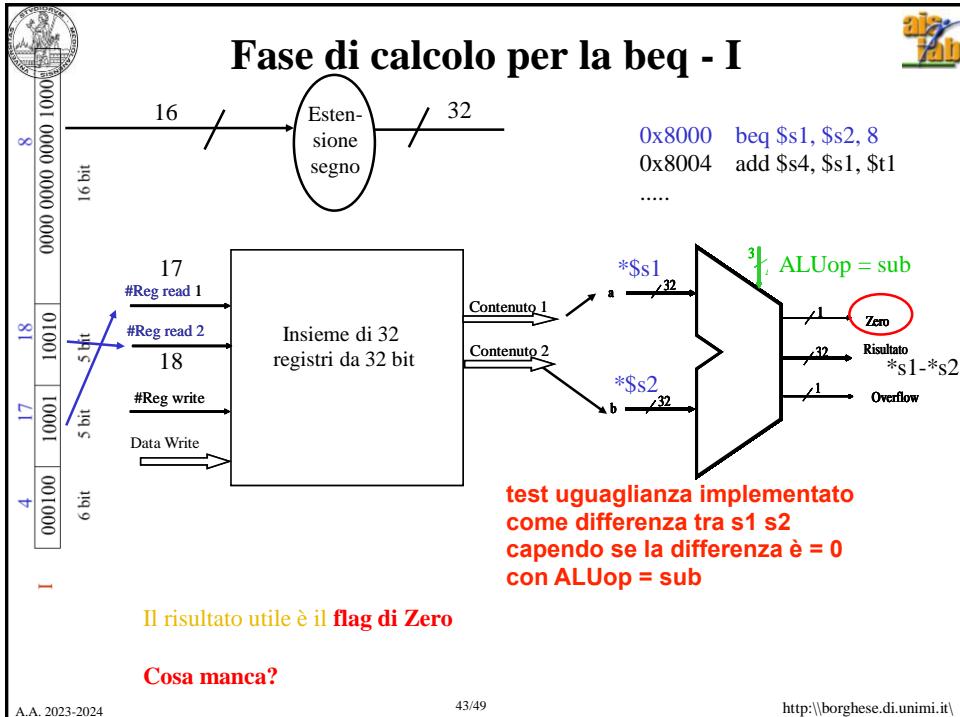


Lettura dei registri



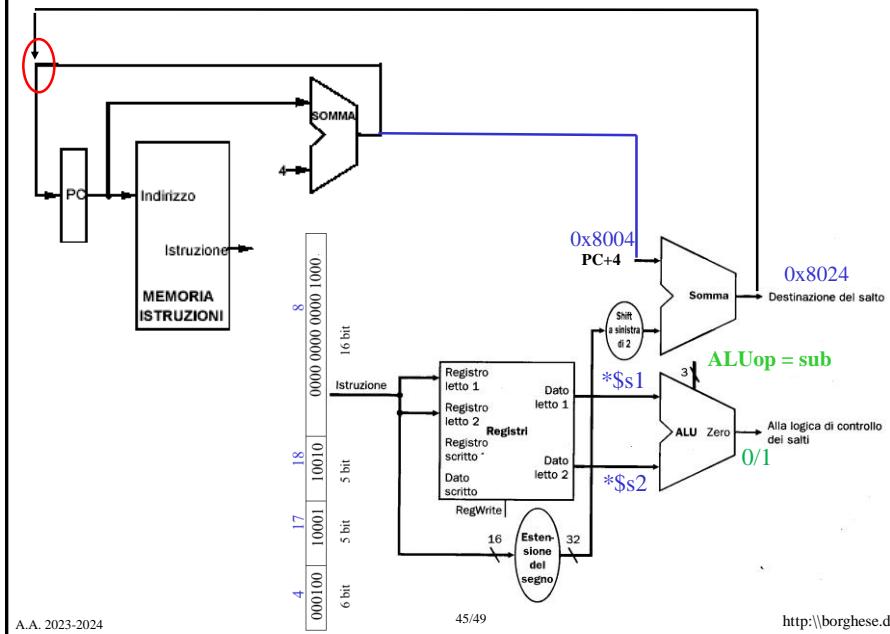
- 2) Leggo il contenuto dei registri. 2 bus che collegano i campi rs e rt con gli input corrispondenti del register file.
Stesso meccanismo delle istruzioni di tipo R / I (accesso alla memoria)







Implementazione del salto



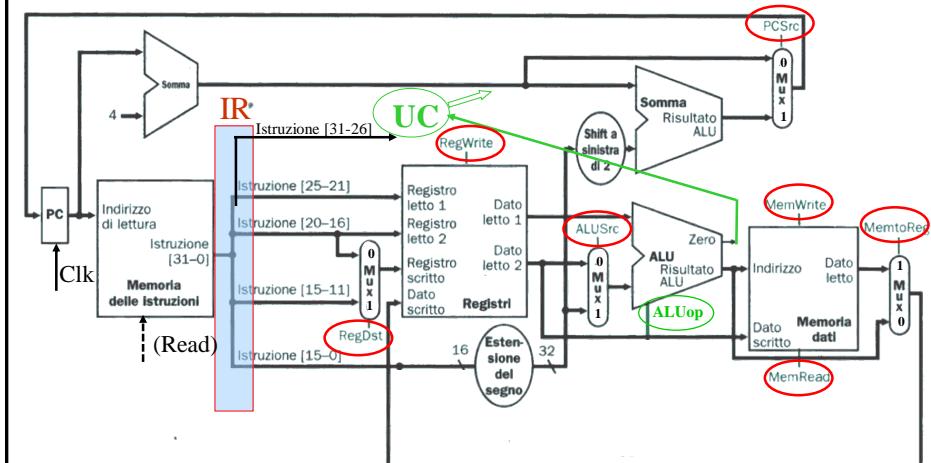
A.A. 2023-2024

<http://borghese.di.unimi.it/>

45/49



Schema generale CPU



{R, lw, sw, addi, beq}

A.A. 2023-2024

46/49

<http://borghese.di.unimi.it/>



Osservazioni



Il ciclo di esecuzione di un'istruzione si compie in un **unico** ciclo di clock.



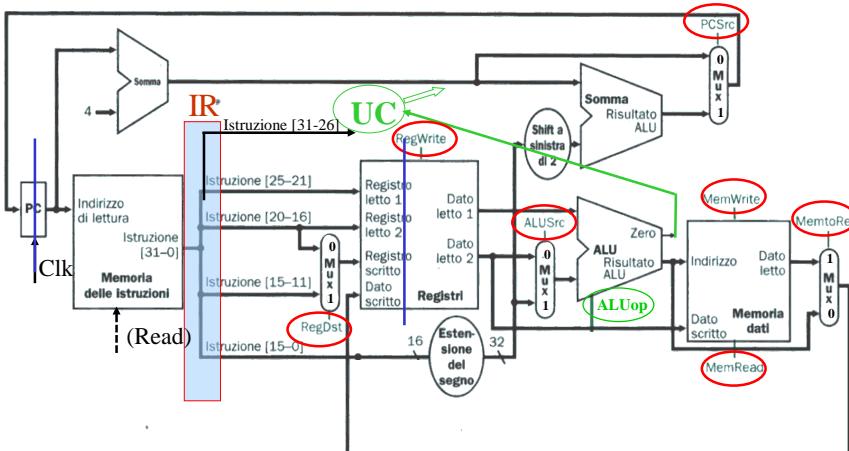
Ogni unità funzionale può essere utilizzata 1 sola volta.



Duplicazione Memoria: Memoria dati e memoria istruzioni (split memory).
Tripli cazione ALU: 2 sommatori + 1 ALU general purpose.

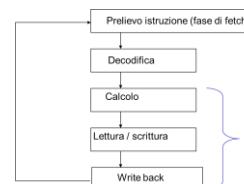


Schema generale CPU



E' un'architettura retroazionata sincronizzata dal clock

Sincronizzazione del controllo (sul PC) **program counter**
Sincronizzazione sui dati (sul RF) **register file**





Sommario



Introduzione alla CPU

CPU per le istruzioni di tipo R

CPU per le istruzioni di tipo I



UC della CPU a singolo ciclo

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento sul Patterson: capitolo 4.2 , 4.4, D1, D2.



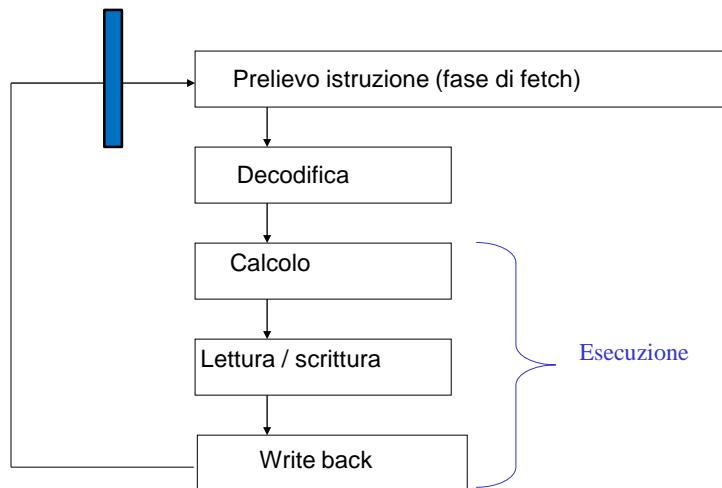
Sommario

UC della CPU

Control and Data path



Ciclo di esecuzione di un'istruzione MIPS



Codifica delle istruzioni

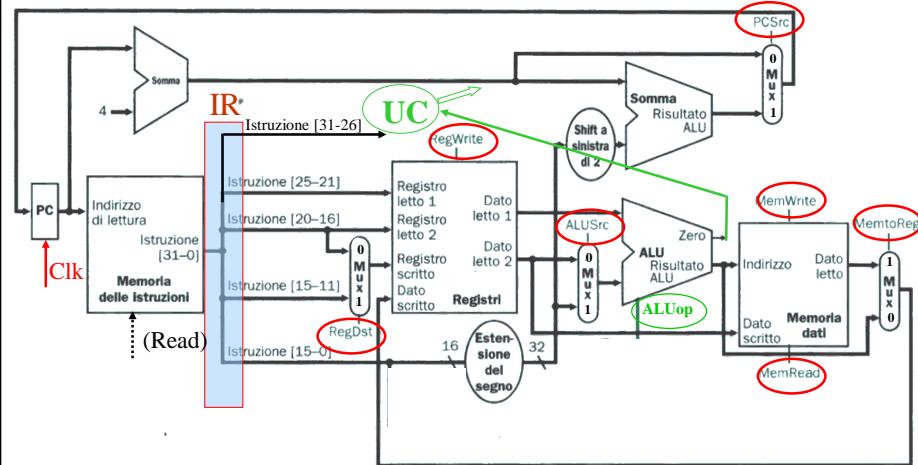


- Tutte le istruzioni MIPS hanno la **stessa dimensione (32 bit)** – **Architettura RISC**.
- I 32 bit hanno un significato diverso a seconda del formato (o tipo) di istruzione
 - il tipo di istruzione è riconosciuto in base al valore di alcuni bit (**6 bit**) più significativi (**codice operativo - OPCODE**)
- Le istruzioni MIPS sono di 3 tipi (formati):
 - Tipo R (register)** – Lavorano su 3 registri.
 - Istruzioni aritmetico-logiche.
 - Tipo I (immediate)** – Lavorano su 2 registri. L'istruzione è suddivisa in un **gruppo di 16 bit contenenti informazioni + 16 bit riservati ad una costante**.
 - Istruzioni di accesso alla memoria o operazioni contenenti delle costanti.
 - Tipo J (jump)** – Lavora senza registri: codice operativo + indirizzo di salto.
 - Istruzioni di salto incondizionato.

	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
R	op	rs	rt	rd	shamt	funct
I	op	rs	rt		Indirizzo / costante	
J	op			indirizzo		



Schema generale CPU



UC: Segnali_Controllo = f(OpCode)

- Selezione di cammini
 - Attivazione di unità funzionali
- {R, lw, sw, addi, beq}

A.A. 2023-202

5/39

<http://borgheze.di.unimi.it/>

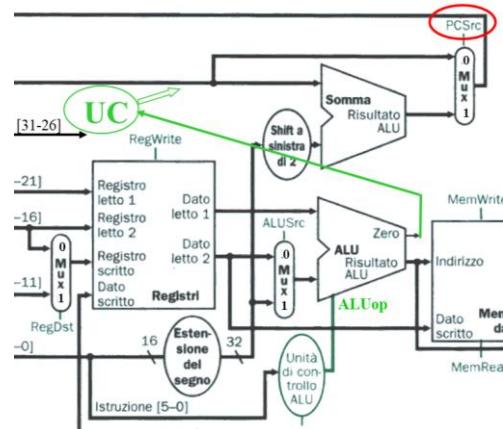
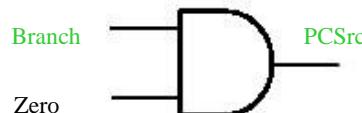


Da dove esce PCSrc?



PCSrc = 1 if

- Branch (Branch = 1) # Prodotto dalla UC = f(OpCode)
- AND
- Zero (Zero = 1) # Prodotto dalla ALU



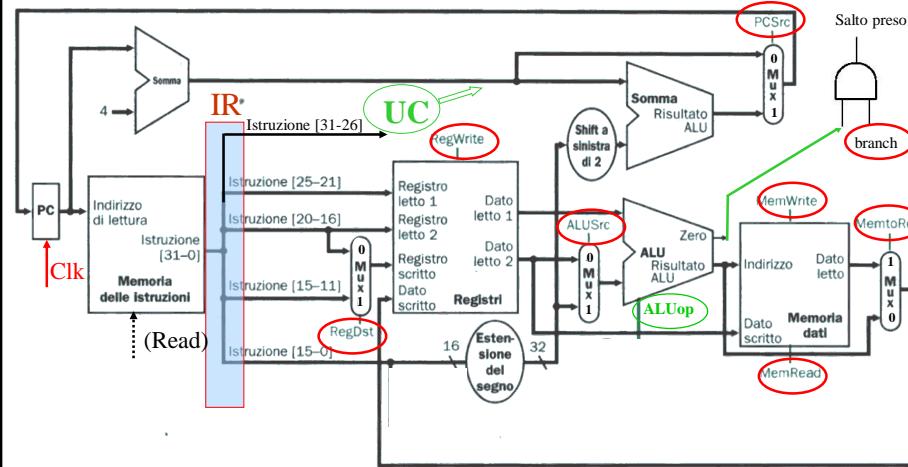
A.A. 2023-202

9/39

<http://borgheze.di.unimi.it/>



Schema generale CPU



UC: Segnali_Controllo = f(OpCode)

- Selezione di cammini
- Attivazione di unità funzionali

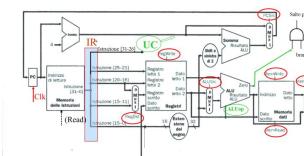
Codici operativi: considerati: {R, lw, sw, addi, beq}



Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst		
RegWrite		
ALUSrc		
Branch		
MemRead		
MemWrite		
MemtoReg		

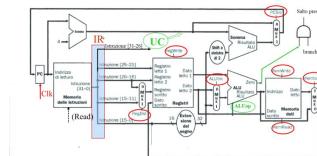




Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite		
ALUSrc		
Branch		
MemRead		
MemWrite		
MemtoReg		



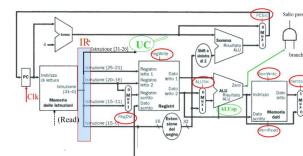
A.A. 2023-202

<http://borghese.di.unimi.it/>

Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc		
Branch		
MemRead		
MemWrite		
MemtoReg		



A.A. 2023-202

10/39

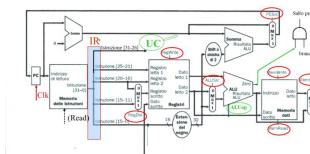
<http://borghese.di.unimi.it/>



Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene preso dall'uscita del sommatore che calcola PC+4 (se il segnale di Zero della ALU = 0)	Il valore del PC viene preso dall'uscita del sommatore che calcola la destinazione del salto (se il segnale di Zero della ALU = 1)
MemRead		
MemWrite		
MemtoReg		



A.A. 2023-202

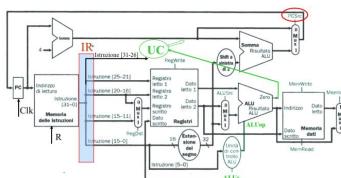
11/39

<http://borghese.di.unimi.it/>

Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene preso dall'uscita del sommatore che calcola PC+4 (se il segnale di Zero della ALU = 0)	Il valore del PC viene preso dall'uscita del sommatore che calcola la destinazione del salto (se il segnale di Zero della ALU = 1)
MemRead	Nessuno	Il contenuto della cella di memoria dati indirizzata dal MAR è posto nel MDR
MemWrite		
MemtoReg		



A.A. 2023-202

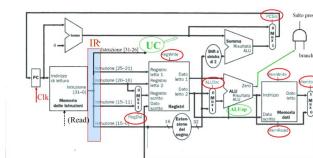
<http://borghese.di.unimi.it/>



Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene preso dall'uscita del sommatore che calcola PC+4 (se il segnale di Zero della ALU = 0)	Il valore del PC viene preso dall'uscita del sommatore che calcola la destinazione del salto (se il segnale di Zero della ALU = 1)
MemRead	Nessuno	Il contenuto della cella di memoria dati indirizzata dal MAR è posto nel MDR
MemWrite	Nessuno	Il contenuto in ingresso al MDR, viene memorizzato nella cella il cui indirizzo è caricato nel MAR
MemtoReg		



A.A. 2023-202

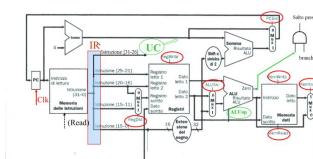
13/39

<http://borghese.di.unimi.it/>

Segnali di controllo su 1 bit



Nome del segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene preso dall'uscita del sommatore che calcola PC+4 (se il segnale di Zero della ALU = 0)	Il valore del PC viene preso dall'uscita del sommatore che calcola la destinazione del salto (se il segnale di Zero della ALU = 1)
MemRead	Nessuno	Il contenuto della cella di memoria dati indirizzata dal MAR è posto nel MDR
MemWrite	Nessuno	Il contenuto in ingresso al MDR, viene memorizzato nella cella il cui indirizzo è caricato nel MAR
MemtoReg	Il valore inviato all'ingresso Dato al Register File proviene dalla ALU	Il valore inviato all'ingresso Dato al Register File proviene dalla memoria



A.A. 2023-202

14/39

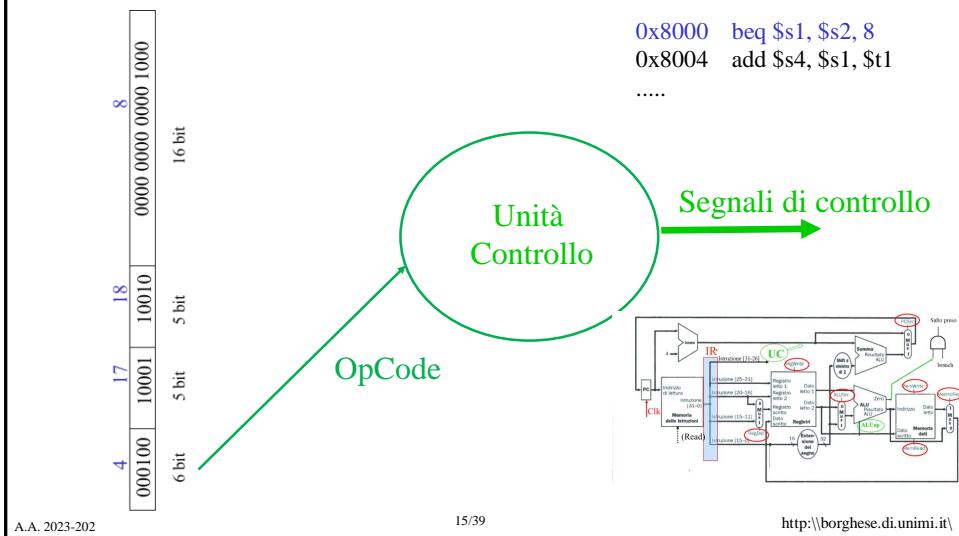
<http://borghese.di.unimi.it/>



Fase di decodifica



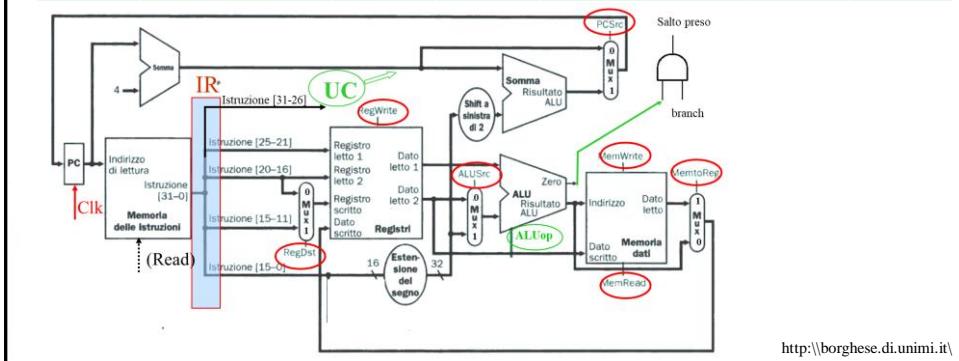
- 1) Leggo l'istruzione e genero i segnali di controllo opportuni.
Bus che connette i 6 bit del CodOp con la UC



Codifica dei segnali di controllo



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)								
Iw (100011)								
sw (101011)								
beq (000100)								
addi(001000)								

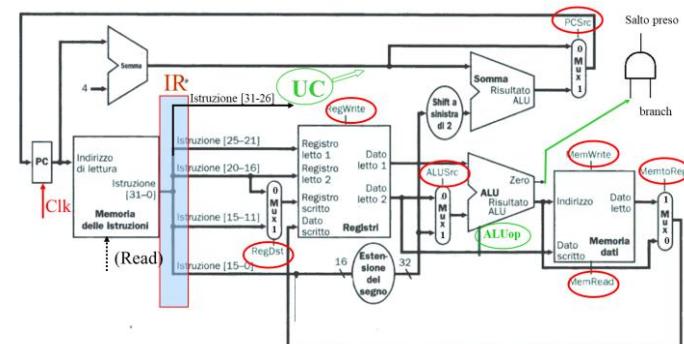




Controllo del data-path



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10 (R)
Iw (100011)								
sw (101011)								
beq (000100)								
addi(001000)								



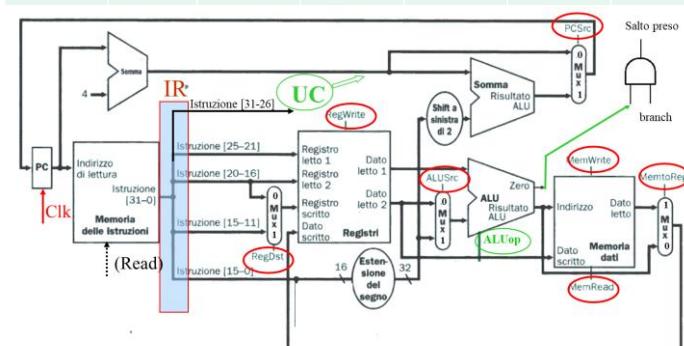
<http://borgheze.di.unimi.it/>



Controllo del data-path



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10 (R)
Iw (100011)	0	1	1	1	1	0	0	00 (add)
sw (101011)								
beq (000100)								
addi(001000)								



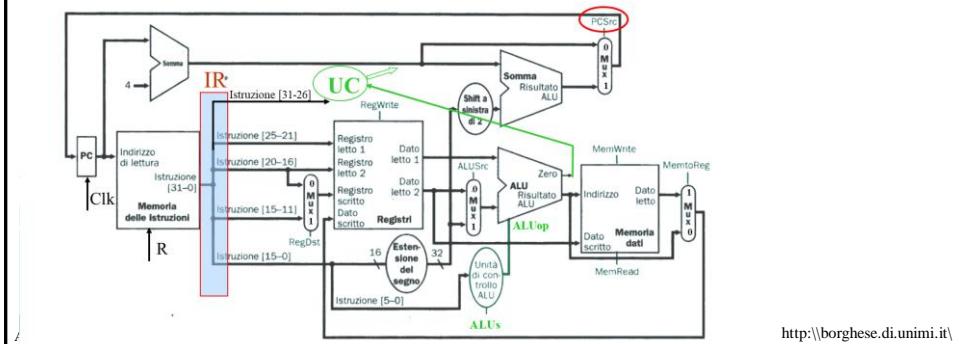
<http://borgheze.di.unimi.it/>



Controllo del data-path



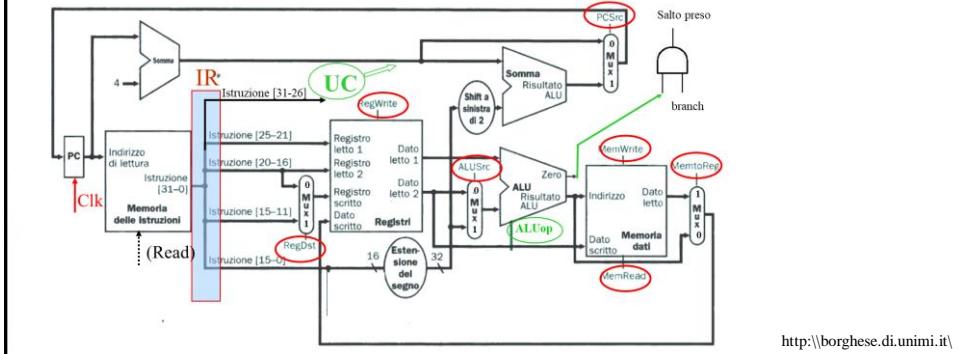
Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10 (R)
lw (100011)	0	1	1	1	1	0	0	00 (add)
sw (101011)	x	1	x	0	0	1	0	00 (add)
beq (000100)								
addi(001000)								



Controllo del data-path



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10 (R)
lw (100011)	0	1	1	1	1	0	0	00 (add)
sw (101011)	x	1	x	0	0	1	0	00 (add)
beq (000100)	x	0	x	0	0	0	1	01 (sub)
addi(001000)								

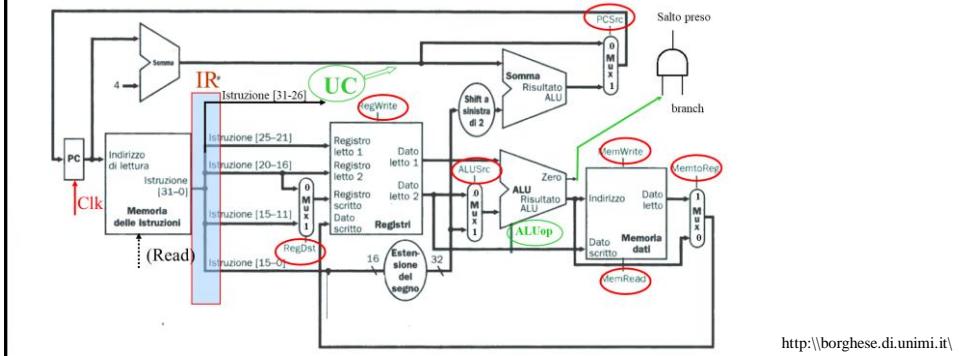




Controllo del data-path



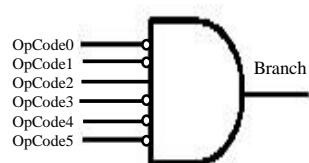
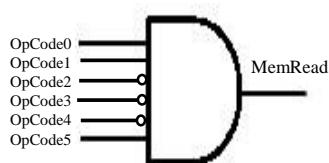
Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10 (R)
lw (100011)	0	1	1	1	1	0	0	00 (add)
sw (101011)	x	1	x	0	0	1	0	00 (add)
beq (000100)	x	0	x	0	0	0	1	01 (sub)
addi(001000)	0	1	0	1	0	0	0	00 (add)



Controllo del data-path



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10
lw (100011)	0	1	1	1	1	0	0	00
sw (101011)	x	1	x	0	0	1	0	00
beq (000100)	x	0	x	0	0	0	1	01
addi(001000)	0	1	0	1	0	0	0	00

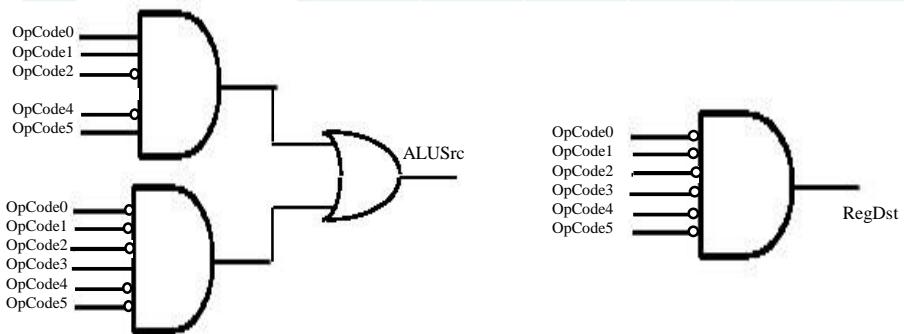




Controllo del data-path



Istruzione (OpCode)	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUs
R (000000)	1	0	0	1	0	0	0	10
Iw (100011)	0	1	1	1	1	0	0	00
sw (101011)	x	1	x	0	0	1	0	00
beq (000100)	x	0	x	0	0	0	1	01
addi(001000)	0	1	0	1	0	0	0	00



A.A. 2023-2

23/39

<http://borghese.di.unimi.it/>

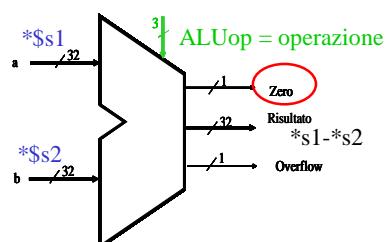
Controllo della ALU



Unità
Controllo

Segnali di controllo dalla UC
{ALUs, ...}

ALUs = {R, add, sub,}



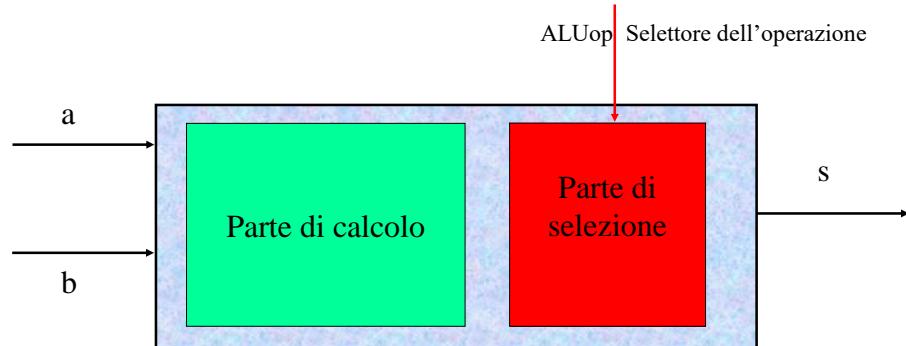
A.A. 2023-202

24/39

<http://borghese.di.unimi.it/>



Struttura a 2 livelli di una ALU



Eseguire un'operazione vuol dire scegliere di portare in uscita un certo risultato prodotto dalla Parte di calcolo oppure un altro



Segnali di UC e ALU



Visione della UC

R	6-bit	5-bit	5-bit	5-bit	5-bit	6-bit
I	op	rs	rt	rd	shamt	funct
Operazione? (ALUs = 10 = R) sub \$s0, \$s1,\$s2	op	rs	rt	Indirizzo / costante		
Operazione = 00 = somma addi \$0, \$1, 20	op = 0	rs = 18	rt = 19	rd = 17	Shamt=0	funct=34
	op = 8	rs = 17	rt = 18			cost = 20

Visione della ALU

Le operazioni consentite dalla ALU (**selezionate tramite ALUop**):

Tipo	Aluop
and	000
or	001
add	010
sub	110
slt	111



UC e ALU

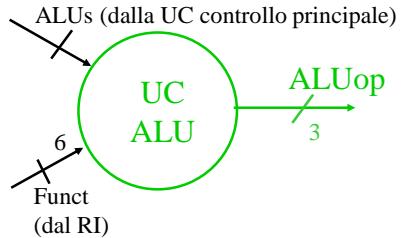


Data l'istruzione, l'UC deve inviare il comando opportuno alla ALU.

Campo Op Code
Campo Funct

Le operazioni consentite dalla ALU:

- and 000
- or 001
- add 010
- sub 110
- slt 111



Quali operazioni devono essere eseguite per le diverse istruzioni (ALUs):

- R -> Dipende dal campo funct
- lw -> Somma
- sw -> Somma
- beq -> Differenza

27/39

Output (ALUop): 3 bit

<http://borghese.di.unimi.it/>

A.A. 2023-202

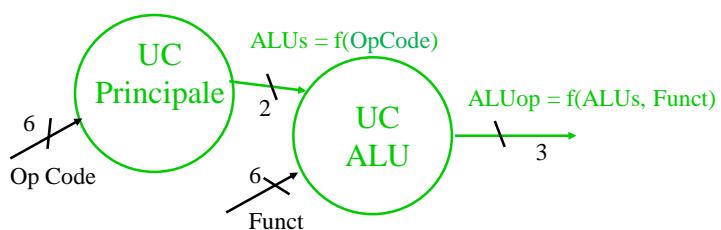


Controllo gerarchico



Le operazioni consentite dalla ALU:

- and 000
- or 001
- add 010
- sub 110
- slt 111

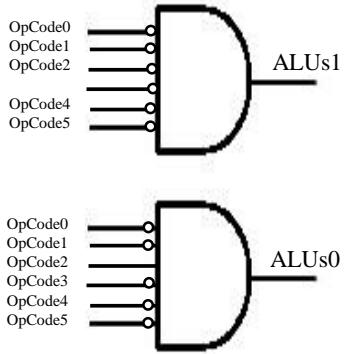


If (OpCode == R) then
 Funct → ALUop
 Else
 OpCode → ALUop

A.A. 2023-202

28/39

<http://borghese.di.unimi.it/>



Sintetizzo i 2 bit come SOP

$$\text{ALUs} = f(\text{OpCode})$$

Istr	OpCode				ALUs		Funct						ALUop			
lw	1	0	0	0	1	1	0	0	x	x	x	x	x	0	1	0
sw	1	0	1	0	1	1	0	0	x	x	x	x	x	0	1	0
beq	0	0	0	1	0	0	0	1	x	x	x	x	x	1	1	0
addi	0	0	1	0	0	0	0	0	x	x	x	x	x	0	1	0
add	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0
sub	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0
and	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0
or	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	1
slt	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1

UC principale

ALU control

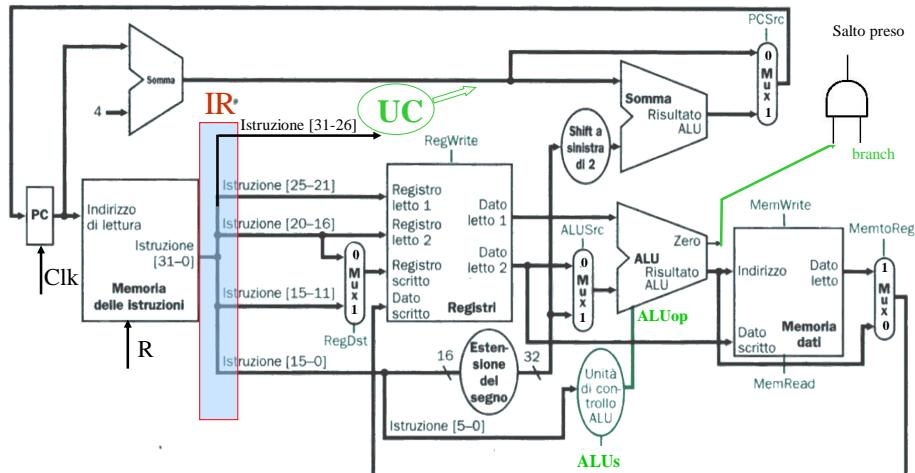
$$\text{ALUop} = f(\text{ALUs}, \text{Funct})$$



Schema generale CPU



{R, lw, sw, addi, beq}



Sommario



UC della CPU

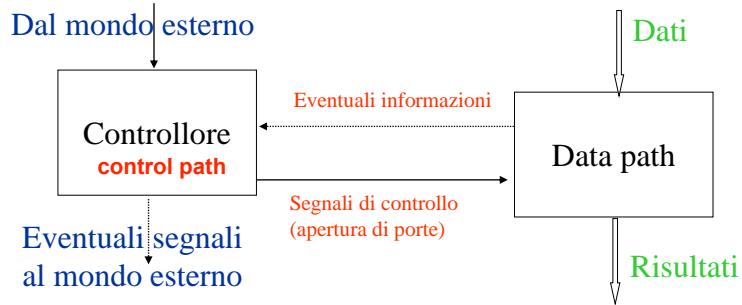
Control and Data path



Rapporto UC - Dati



La CPU è un'architettura del tipo: Controllore - Data-path



Fase comune nel ciclo di esecuzione:

- Fase di fetch
- Decodifica (generazione dei segnali di controllo)

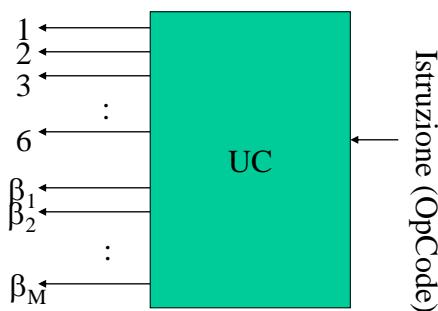
Fase diversa: Esecuzione (Calcolo, Memoria, WriteBack)



L'unità di controllo



- Unità di controllo coordina i flussi di informazione (è il “cervello” della CPU):
- 1) abilitando le vie di comunicazione opportune a seconda dell’istruzione in corso di esecuzione.
- 2) selezionando l’operazione opportuna delle ALU.
- 3) modificando un elemento di stato (memoria o register file)





L'istruzione jump



J

Op Code = 2

Indirizzo

6 bit

26 bit

salto incondizionato

L'indirizzo di salto sarà determinato in due passi:

0x8000 j 80040

A) Calcolo di Indirizzo = Indirizzo * 4.

B) Determinazione dell'indirizzo di destinazione del salto come:

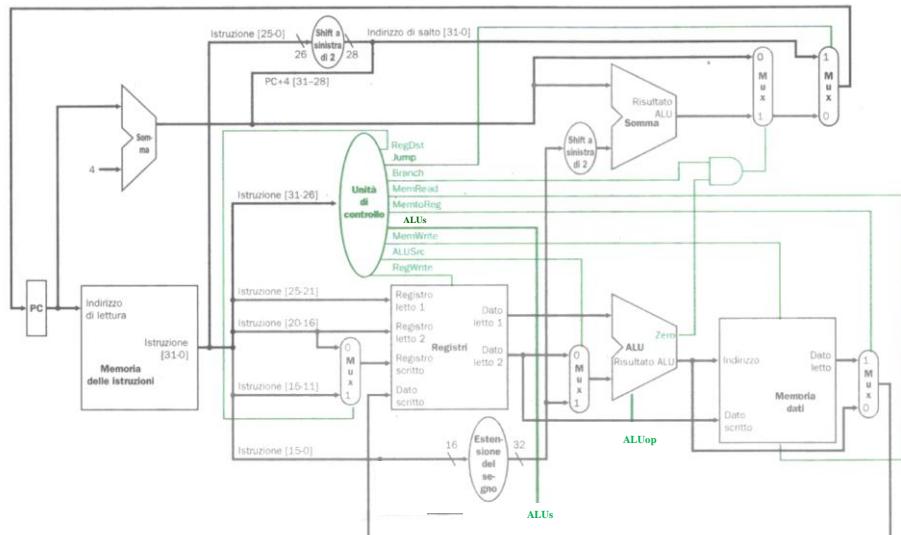
Base (PC) 0100 1000 0011 0001 1011 1011 1011 10 00
 Nuovo indirizzo 0000 0110 0111 0000 0000 0001 00 00 =

Indirizzo salto 0100 0000 0110 0111 0000 0000 0001 00 00

L'indirizzo è un numero positivo (posizione in memoria assoluta).



CPU + UC completa (aggiunta di jump)





Controllo del data-path

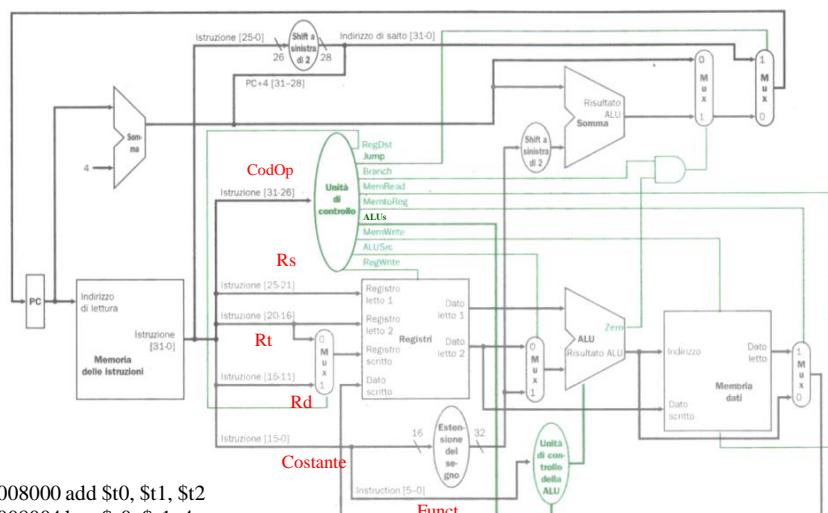


Istruzione (OpCode)	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUs	Jump
R (0000000)	1	0	0	1	0	0	0	10	0
lw (100011)	0	1	1	1	1	0	0	00	0
sw (101011)	x	1	x	0	0	1	0	00	0
addi (001000)	0	1	0	1	0	0	0	00	0
beq (000100)	x	0	x	0	0	0	1	01	0
j (000010)	x	x	x	0	0	0	0	xx	1

La lettura della memoria non è indolare soprattutto quando sono presenti dei livelli (di cache).



Contenuto della CPU per l'esecuzione di istruzioni diverse



0x00008000 add \$t0, \$t1, \$t2
 0x00008004 beq \$s0, \$s1, 4
 0x00008008 lw \$t1, 32(\$s0)
 0x0000800C sw \$t1, 32(\$s0)
 0x00008010 addi \$t1, \$t1, 16



Sommario



UC della CPU

Control and Data path



Codifica dell'informazione Esercitazione

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimenti al testo: Paragrafi 2.4, 2.9, 3.1, 3.2, 3.5 (codifica IEEE754)



Sommario

Esercizi sulla codifica dei numeri binari

Esercizi sulle operazioni con i numeri binari

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



Conversione da base n a base 10



Consideriamo un numero in base 2 (2 cifre: 0, 1)

Convertiamo in decimale il numero 1101 in base binaria.

$$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13_{10}$$

Consideriamo un numero in base 3 (3 cifre: 0, 1, 2)

Convertiamo in decimale il numero 2102 in base ternaria.

$$2 \cdot 3^3 + 1 \cdot 3^2 + 0 \cdot 3^1 + 2 \cdot 3^0 = 54 + 9 + 0 + 2 = 65_{10}$$



Conversione da base Hex a base decimale



Consideriamo un numero in base 16 (16 cifre: da 0 a F)

Convertiamo in decimale il numero 1BC8 in base 16.

$$\begin{aligned} 1 \cdot 16^3 + 11 \cdot 16^2 + 12 \cdot 16^1 + 8 \cdot 16^0 &= \\ 1 \cdot 4,096 + 11 \cdot 256 + 12 \cdot 16 + 8 \cdot 1 &= \\ 4,096 + 2,816 + 192 + 8 &= 7,112 \end{aligned}$$

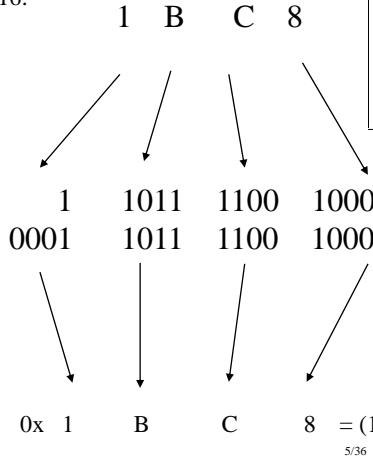


Conversione da base Hex a binario



Consideriamo un numero in base 16
(16 cifre: da 0 a F)

Convertiamo in binario il numero 0x1BC8
in base 16.



Cifre esadecimali	Valori decimali	Equivalenti binari	Valori posizionali esadecimali	Valori decimali
0	0	0000	16^{-5}	$1/4096=0.000\ 244\ 140\ 625$
1	1	0001	16^{-4}	$1/256=0.003\ 906\ 25$
2	2	0010	16^{-3}	$1/16=0.062\ 5$
3	3	0011	16^0	1
4	4	0100	16^1	16
5	5	0101	16^2	256
6	6	0110	16^3	4096
7	7	0111	16^4	65\ 536
8	8	1000	16^5	1\ 048\ 576
9	9	1001		
A	10	1010		
B	11	1011		
C	12	1100		
D	13	1101		
E	14	1110		
F	15	1111		

16 cifre binarie (4x4)

A.A. 2023-2024

$5/36$

<http://borghese.di.unimi.it/>



Conversione da base 10 a base 2 - I



$$N = 528$$

$$528 : 2 = 264$$

$$R = 0 \text{ (peso } 2^0\text{)}$$

$$M = \dots .0$$

A.A. 2023-2024

6/36

<http://borghese.di.unimi.it/>



Conversione da base 10 a base 2 - II



N = 528

$$528 : 2 = 264$$

R = 0 (peso 2^0)

$$264 : 2 = 132$$

R = 0 (peso 2^1)

M = 00



Conversione da base 10 a base 2 - III



N = 528

$$528 : 2 = 264$$

R = 0 (peso 2^0)

$$264 : 2 = 132$$

R = 0 (peso 2^1)

$$132 : 2 = 66$$

R = 0 (peso 2^2)

M = 000



Conversione da base 10 a base 2 - IV



N = 528

$$\begin{array}{ll} 528 : 2 = 264 & R = 0 \text{ (peso } 2^0) \\ 264 : 2 = 132 & R = 0 \text{ (peso } 2^1) \\ 132 : 2 = 66 & R = 0 \text{ (peso } 2^2) \\ 66 : 2 = 33 & R = 0 \text{ (peso } 2^3) \end{array}$$

M = 0000



Conversione da base 10 a base 2 - V



N = 528

$$\begin{array}{ll} 528 : 2 = 264 & R = 0 \text{ (peso } 2^0) \\ 264 : 2 = 132 & R = 0 \text{ (peso } 2^1) \\ 132 : 2 = 66 & R = 0 \text{ (peso } 2^2) \\ 66 : 2 = 33 & R = 0 \text{ (peso } 2^3) \\ 33 : 2 = 16 & R = 1 \text{ (peso } 2^4) \end{array}$$

M = 10000



Conversione da base 10 a base 2 - VI



N = 528

$$\begin{array}{ll} 528 : 2 = 264 & R = 0 \text{ (peso } 2^0) \\ 264 : 2 = 132 & R = 0 \text{ (peso } 2^1) \\ 132 : 2 = 66 & R = 0 \text{ (peso } 2^2) \\ 66 : 2 = 33 & R = 0 \text{ (peso } 2^3) \\ 33 : 2 = 16 & R = 1 \text{ (peso } 2^4) \\ 16 : 2 = 8 & R = 0 \text{ (peso } 2^5) \end{array}$$

M = 010000



Conversione da base 10 a base 2 - VII



N = 528

$$\begin{array}{ll} 528 : 2 = 264 & R = 0 \text{ (peso } 2^0) \\ 264 : 2 = 132 & R = 0 \text{ (peso } 2^1) \\ 132 : 2 = 66 & R = 0 \text{ (peso } 2^2) \\ 66 : 2 = 33 & R = 0 \text{ (peso } 2^3) \\ 33 : 2 = 16 & R = 1 \text{ (peso } 2^4) \\ 16 : 2 = 8 & R = 0 \text{ (peso } 2^5) \\ 8 : 2 = 4 & R = 0 \text{ (peso } 2^6) \end{array}$$

M = 0010000



Conversione da base 10 a base 2 - VIII



N = 528

528 : 2 = 264	R = 0 (peso 2^0)
264 : 2 = 132	R = 0 (peso 2^1)
132 : 2 = 66	R = 0 (peso 2^2)
66 : 2 = 33	R = 0 (peso 2^3)
33 : 2 = 16	R = 1 (peso 2^4)
16 : 2 = 8	R = 0 (peso 2^5)
8 : 2 = 4	R = 0 (peso 2^6)
4 : 2 = 2	R = 0 (peso 2^7)

M = 00010000



Conversione da base 10 a base 2 - IX



N = 528

528 : 2 = 264	R = 0 (peso 2^0)
264 : 2 = 132	R = 0 (peso 2^1)
132 : 2 = 66	R = 0 (peso 2^2)
66 : 2 = 33	R = 0 (peso 2^3)
33 : 2 = 16	R = 1 (peso 2^4)
16 : 2 = 8	R = 0 (peso 2^5)
8 : 2 = 4	R = 0 (peso 2^6)
4 : 2 = 2	R = 0 (peso 2^7)
2 : 2 = 1	R = 0 (peso 2^8)

M = 000010000



Conversione da base 10 a base 2 - X



N = 528

$528 : 2 = 264$	R = 0 (peso 2^0)
$264 : 2 = 132$	R = 0 (peso 2^1)
$132 : 2 = 66$	R = 0 (peso 2^2)
$66 : 2 = 33$	R = 0 (peso 2^3)
$33 : 2 = 16$	R = 1 (peso 2^4)
$16 : 2 = 8$	R = 0 (peso 2^5)
$8 : 2 = 4$	R = 0 (peso 2^6)
$4 : 2 = 2$	R = 0 (peso 2^7)
$2 : 2 = 1$	R = 0 (peso 2^8)
$1 : 2 = \textcolor{red}{0}$	R = 1 (peso 2^9)

$$M = 10\ 0001\ 0000 = 1 \times 2^4 + 1 \times 2^9 = 16 + 512 = 528$$



Conversione da base 10 a base 3



- N = 528

$528 : 3 = 176$	R = 0 (peso 3^0)
$176 : 3 = 58$	R = 2 (peso 3^1)
$58 : 3 = 19$	R = 1 (peso 3^2)
$19 : 3 = 6$	R = 1 (peso 3^3)
$6 : 3 = 2$	R = 0 (peso 3^4)
$2 : 3 = \textcolor{red}{0}$	R = 2 (peso 3^5)

$$M = 201120 = 2 \times 3^5 + 1 \times 3^3 + 1 \times 3^2 + 2 \times 3^1 = \\ 2 \times 243 + 1 \times 27 + 1 \times 9 + 2 \times 3 = 486 + 27 + 9 + 6 = 528$$



Conversione numeri decimali



I numeri decimali sono i numeri che hanno una parte frazionaria. Sono un (piccolissimo) sottoinsieme dei numeri reali

Viene convertita separatamente la parte intera e la parte frazionaria.



Conversione da base 10 a base 2 – parte intera



$$N = 528,125$$

$528 : 2 = 264$	$R = 0$ (peso 2^0)
$264 : 2 = 132$	$R = 0$ (peso 2^1)
$132 : 2 = 66$	$R = 0$ (peso 2^2)
$66 : 2 = 33$	$R = 0$ (peso 2^3)
$33 : 2 = 16$	$R = 1$ (peso 2^4)
$16 : 2 = 8$	$R = 0$ (peso 2^5)
$8 : 2 = 4$	$R = 0$ (peso 2^6)
$4 : 2 = 2$	$R = 0$ (peso 2^7)
$2 : 2 = 1$	$R = 0$ (peso 2^8)
$1 : 2 = 0$	$R = 1$ (peso 2^9)

$$M = 10\ 0001\ 0000 = 1 \times 2^4 + 1 \times 2^9 = 16 + 512 = 528$$



Conversione da base 10 a base 2 – parte decimale



$N = 528,125$

$$0,125 * 2 = 0 + 0.25 \quad 0 \text{ (peso } 2^{-1} \text{)}$$

$$0,25 * 2 = 0 + 0.50 \quad 0 \text{ (peso } 2^{-2} \text{)}$$

$$0,5 * 2 = 1 + 0 \quad 1 \text{ (peso } 2^{-3} \text{)}$$

Non c'è più nulla da codificare

$M = 10\ 0001\ 0000,001$



Errori di approssimazione



Esempio: $10,75_{10} = 1010,11_2$

Esempio: $10,76_{10} = 1010,1100001..._2$

$$10 : 2 \Rightarrow 0$$

$$0,75 * 2 \Rightarrow 1$$

$$0,76 * 2 \Rightarrow 1x2^{-1}$$

$$5 : 2 \Rightarrow 1$$

$$(1),50 * 2 \Rightarrow 1$$

$$(1),52 * 2 \Rightarrow 1x2^{-2}$$

$$2 : 2 \Rightarrow 0$$

$$(1),00 \Rightarrow$$

$$(1),04 * 2 \Rightarrow 0x2^{-3}$$

$$1 : 2 \Rightarrow 1$$

$$\# \# \# \# \# 11$$

$$(0),08 * 2 \Rightarrow 0x2^{-4}$$

$$\# \# \# \# \# 1010,$$

$$(0),16 * 2 \Rightarrow 0x2^{-5}$$

$$(0),32 * 2 \Rightarrow 0x2^{-6}$$

$$(0),64 * 2 \Rightarrow 1x2^{-7} \quad (2^{-7} = 0,0078125)$$

$$(1),28 \dots \dots \dots$$

Errori di approssimazione:
arrotondamento e troncamento.

Con 7 bit di parte fraz, rappresento:

$$0,5 + 0,25 + 0,0078125 = 0,7578125$$

$$\text{Errore} = 0,0011875$$



Sottrazione di numeri in complemento a 2



$N = (a-b) = (18-21)$ in base 10 su 8 bit

$$a = 10010$$

$$18 : 2 = 9 \quad R = 0$$

$$9 : 2 = 4 \quad R = 1$$

$$4 : 2 = 2 \quad R = 0$$

$$2 : 2 = 1 \quad R = 0$$

$$1 : 2 = 0 \quad R = 1$$

$$b = 10101$$

Scrivo a e b su 8 bit. Sono numeri positivi, copro le cifre alla sinistra del numero con l'estensione del segno (0) e ottengo:

$$a = 10010 \Rightarrow \text{su 8 bit} \Rightarrow 0001\ 0010$$

$$b = 10101 \Rightarrow \text{su 8 bit} \Rightarrow 0001\ 0101$$



Sottrazione di numeri in complemento a 2



$N = (a-b) = (18-21)$ in base 10 su 8 bit

$N = (a-b) = a + (-b) \Rightarrow$ calcolo $-b$ in complemento a 2:

- $a = 21$, su 8 bit $\Rightarrow 0001\ 0101$

- Complemento a 1 $1110\ 1010$

- Complemento a 2 $1110\ 1011 = -21_{10}$

Eseguo la somma:

$$0001\ 0010 +$$

$$1110\ 1011 =$$

$$1111\ 1101 = -3$$

$1111\ 1101 \Rightarrow$ compl a 1 = 0000 0010 \Rightarrow compl a 2: $+1 = 0000\ 0011$

$$21 + (-21) = 0$$

$$0001\ 0101 + 1110\ 1011 = (1) 0000\ 0000 = 2^n = 2^8 = 256$$



Sommario



Esercizi sulla codifica dei numeri binari

Esercizi sulle operazioni con i numeri binary

Codifica IEEE754 dei numeri reali anche in forma denormalizzata



Numeri decimali rappresentazione in fixed point



Numeri reali per il computer non sono i numeri reali per la matematica!!
E' meglio chiamarli float (numeri decimali), sono in numero finito.

Dato un certo numero di bit (stringa) per codificare un numero float, esistono due tipi di codifiche possibili:

Rappresentazione in fixed point.
La virgola è in posizione fissa all'interno della stringa.

Supponiamo di avere una stringa di 8 cifre, con virgola in 3a posizione:

27,35 = + | 27,35000

-18,7 = - | 18,70000

0,001456 = + | 00,00145(6)

928 = + | 28,00000 - perdo il 9 delle centianai



Numeri decimali rappresentazione floating point

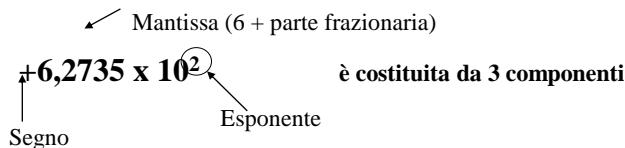


Rappresentazione come mantissa + esponente. $E = \sum_k c_k b_k = \sum_{k=-M}^N c_k B^k$

Esempio di **rappresentazioni equivalenti**:

$$62,735 = 62,735 \times 10^1 = \mathbf{6,2735 \times 10^2} = 0,62735 \times 10^3 = \\ 10^2 \times (\mathbf{6 \times 10^0 + 2 \times 10^{-1} + 7 \times 10^{-2} + 3 \times 10^{-3} + 5 \times 10^{-4}})$$

In grassetto viene evidenziata la rappresentazione normalizzata (una cifra prima della virgola).



Vengono rappresentati numeri molto grandi e molto piccoli.



Standard IEEE 754 (1980)



<http://stevehollasch.com/cgindex/coding/ieefloat.html>

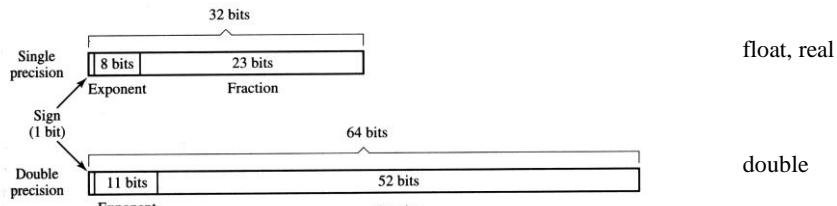


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Rappresentazione **polarizzata** dell'esponente:

Polarizzazione pari a 127 per singola precisione =>
1 viene codificato come 1000 0000.

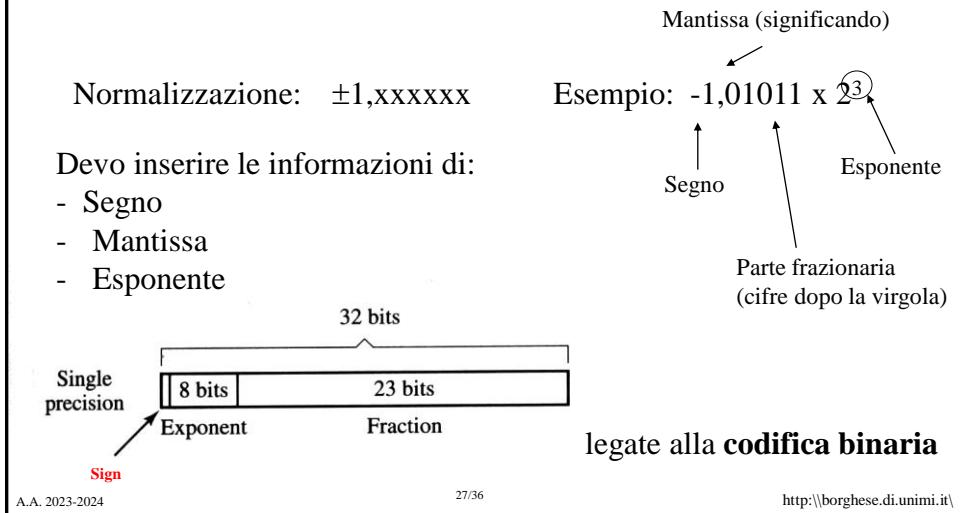
Polarizzazione pari a 1023 in doppia precisione.
1 viene codificato come 1000 0000 000.



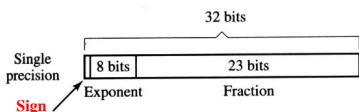
Codifica mediante lo standard IEEE 754



Esempio: $N = -10,75_{10} = -1010,11_2$



Codifica mediante lo standard IEEE 754



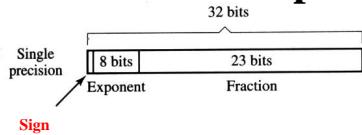
Esempio: $N = -10,75_{10} = -1010,11_2$

- 1) Normalizzazione: $\pm 1,xxxxxx$ Esempio: $-1,01011 \times 2^3$
- 2) Codifica del segno $1 = -$ $0 = +$
- 3) Calcolo dell'esponente, $exp.$
- 4) Completamento della scrittura della parte frazionaria su 23 bit

s	Exp+127	Parte Frazionaria
1	130	01011000000000000000000



Calcolo dell'esponente in notazione polarizzata



Esempio: $N = -10,75_{10} = -1010,11_2$
 $= -1,01011_2 \times 2^3$

Sign

Calcolo dell'esponente, **e**, in rappresentazione polarizzata (si considerano solo 254 esponenti sui 256 possibili, compreso tra -126 e +127):

Codifica

1111 1111 = 255 →

1111 1110 = 254 →

Exp effettivo del numero

Codifica riservata

+127

1000 0010 = 130

+3

1000 0001 = 129 →

+2

1000 0000 = 128 →

+1

Polarizzazione:

0111 1111 = 127 →

0

0111 1110 = 126 →

-1

0000 0001 = 1 →

-126

0000 0000 = 0 →

Codifica riservata

A.A. 2023-20 N = 1 | 1000 0010 | 0101 1000 0000 0000 0000 000 http://borgheze.di.unimi.it/



Configurazioni notevoli nello Standard IEEE 754

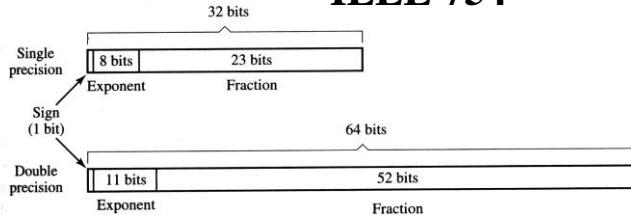


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Configurazioni notevoli:

0 Mantissa: 0 Esponente: 00000000

$+\infty$ Mantissa: 0 Esponente: 11111111.

NaN Mantissa: $\neq 0$. Esponente: 11111111.

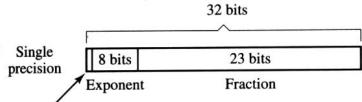
NB Non esiste uno «0» che si possa codificare come $1,yyyy \times b^{-zzz}$

Range degli esponenti (8 bit): $1-254 \Rightarrow -126 \leq \text{exp} \leq +127$.

Numeri float: $1.0 \times 2^{-126} = 1.175494351 \times 10^{-38} \div 3.402823466 \times 10^{38} = 1.1\dots11 \times 2^{127}$



Capacità dello Standard IEEE 754



Range degli esponenti (8 bit): $1--254 \Rightarrow -126 \leq \text{exp} \leq +127$.

Minimo float (in valore assoluto!): 1.0×2^{-126}

Massimo float: $1.1111\ 1111\ 1111\ 1111\ 1111\ 111 \times 2^{+127}$

Capacità di rappresentazione di una variabile float in decimale:

Minimo: $1.175494350822288 \times 10^{-38}$ ($1.175494350822288e-038$)

Massimo: $3.402823466385289x 10^{38}$ ($3.402823466385289e+038$)

Distanza tra Minimo_float e il float successivo è 2^{-149}

Distanza tra Minimo_float e 0 è 2^{-126} ($1.175494350822288 \times 10^{-38}$)

Discontinuità tra Minimo_float e zero.

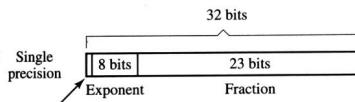
Si può fare di meglio?

Numero denormalizzato Mantissa: $\neq 0$ Esponente: 00000000



Denormalizzazione nello Standard IEEE 754

Esempio di numero denormalizzato: $0,000001 \times 2^{-126}$



Range degli esponenti (8 bit): $1--254 \Rightarrow -126 \leq \text{exp} \leq +127$.

Minimo float (in valore assoluto!): $1.0 \times 2^{-126} = 1.175494350822288 \times 10^{-38}$

Tuttavia abbiamo anche la mantissa a disposizione. Se troviamo una codifica per cui possiamo scrivere 0,0000 0000 0000 0000 0001, otteniamo un numero più piccolo (in valore assoluto!) pari a $2^{(-23-126)} = 2^{-149}$ ($1.401298464324817 \times 10^{-45}$)

Discontinuità tra Minimo_float e 0 diventa 2^{-149} , la stessa che è presente tra Minimo_float e il float successivo.

Configurazioni notevoli:

0 Mantissa: 0 Esponente: 00000000

$+\infty$ Mantissa: 0 Esponente: 11111111.

NaN Mantissa: $\neq 0$ Esponente: 11111111.

Numero denormalizzato Mantissa: $\neq 0$ Esponente: 00000000



Risoluzione della codifica dei reali

Distanza tra due numeri vicini.



Fixed point: Risoluzione fissa, pari al peso del bit meno significativo.

Esempio su 8 bit: +1111,101 la risoluzione per tutti i numeri sarà: $1 \times 2^{-3} = 0,125$

Floating point: Risoluzione *relativa* fissa, pari al peso del bit meno significativo.

Il bit meno significativo è in 23a posizione in singola precisione => 2^{-23} , ne consegue che la risoluzione sarà 2^{-23} volte il numero descritto.

Esempi:

$$100,\dots = 1,000 \times 2^2 \Rightarrow \text{La risoluzione sarà } 2^{-23} \times 2^2 = 2^{-21}$$

$$1.0 \times 2^{-126} \Rightarrow \text{La risoluzione sarà } 2^{-23} \times 2^{-126} = 2^{-149}$$

.....



Conversione in IEEE754



$$N = 140,25$$

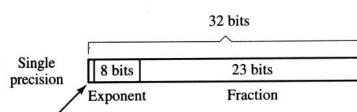
Converto in binario: 1000 1100,01

Normalizzo: $1,000110001 \times 2^7$

Detemino il segno (+) -> bit di segno = 0

Calcolo l'esponente polarizzato su 9 bit ($127+7$) = 1000 0100

Determino la parte frazionaria su 23 bit = 000110001 0000000





Altri formati



Aggiornamento di IEEE754 del 2008:

16 bit (mezza precisione): 1 bit di segno + 5 bit di esponente (con polarizzazione di 15) e 10 bit per la parte frazionaria.

128 bit (quadrupla precisione): 1 bit di segno + 15 bit di esponente (con una polarizzazione di 262.143) + 112 bit per la parte frazionaria.

Operazioni di machine learning non richiedono una precisione di 23 bit sulla parte frazionaria.

- L'output della moltiplicazione di matrici e le somme interne devono rimanere su 32 bit (fp32).
- L'esponente su 5 bit dei numeri su 16 bit (fp16) in input alla moltiplicazione di matrici induce ad errori di calcolo, producendo numeri al di fuori dell'intervallo codificato, che è stretto; questo verrebbe evitato utilizzando il formato fp32.
- La riduzione della dimensione della mantissa dei numeri in input alla moltiplicazione di matrici dai 23 bit del formato fp32 a 7 bit non riduce significativamente l'accuratezza.

Il risultato è stata la definizione del formato **Brain floating (bf16) proposto da Google**, il quale mantiene la stessa codifica del formato fp32 per l'esponente, su 8 bit, ma taglia la mantissa a 7 bit. Quindi: 1 bit di segno + 8 bit di esponente (con polarizzazione di 127) + 7 bit di parte frazionaria.



Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



Macchine a Stati finiti (esercizi)

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

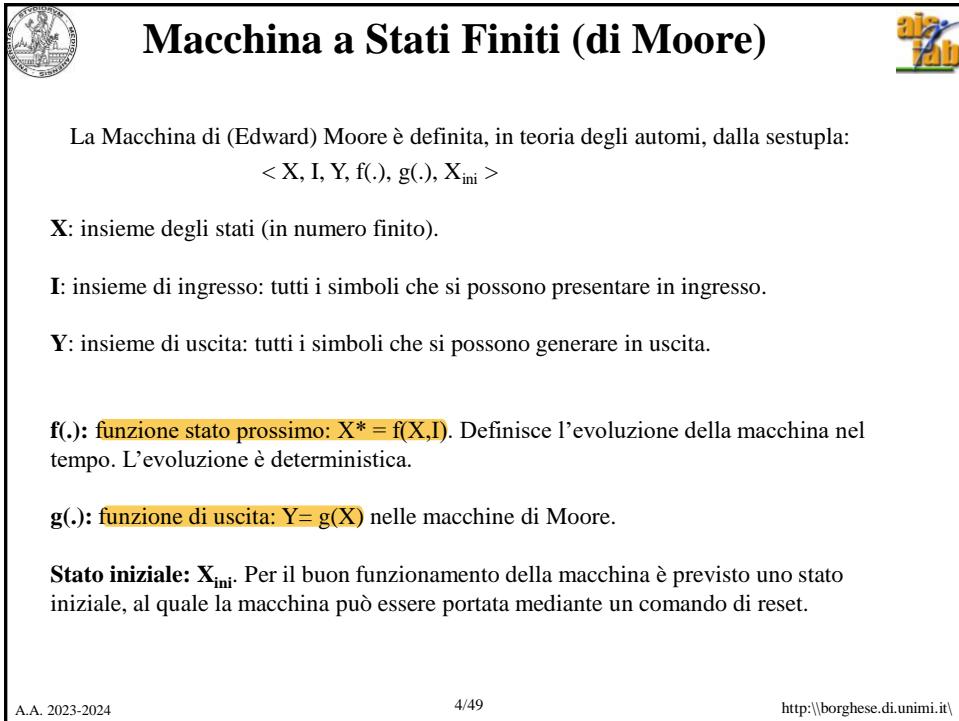
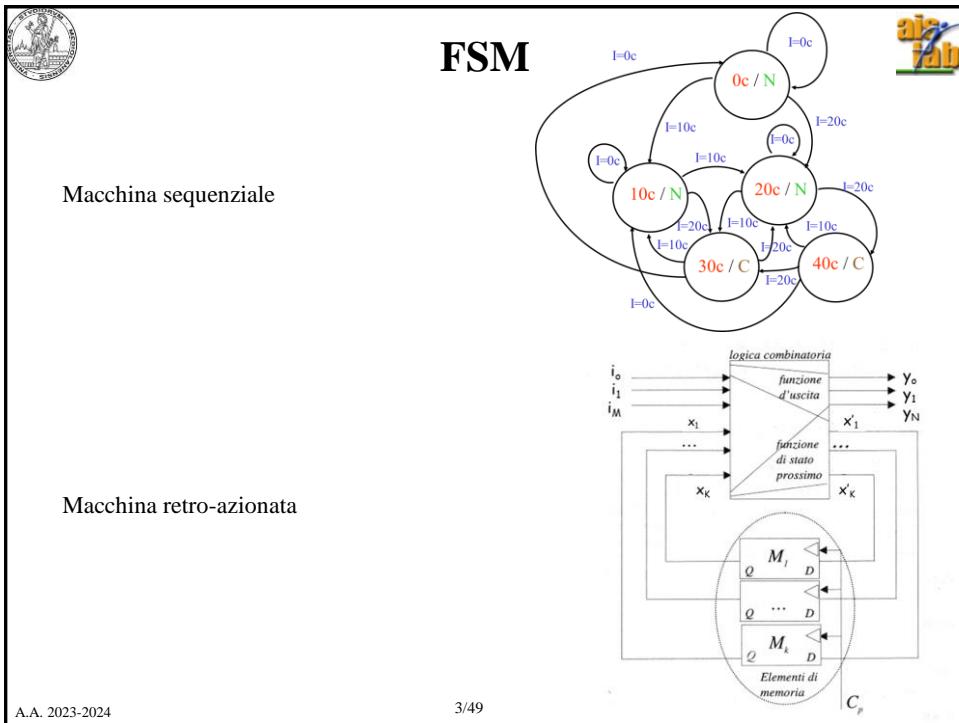
Università degli Studi di Milano



Sommario



- **FSM**
- Controllore di un semaforo
- Riconoscitore di stringhe
- Latch come FSM





Descrizione di una macchina di Moore



STG: State Transition Graph (Diagramma degli stati o Grafo delle transizioni). Ad ogni nodo è associato uno stato. Un arco orientato da uno stato x_i ad uno stato x_j , contrassegnato da un simbolo (di ingresso) α , rappresenta una transizione (passaggio di stato) che si verifica quando la macchina, essendo nello stato x_i , riceve come ingresso il simbolo α .

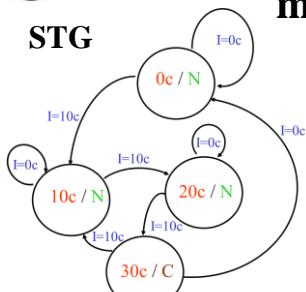
STT: State Transition Table (Tabella degli Stati). Per ogni coppia, (Stato presente – Ingresso), si definisce l’Uscita e lo Stato Prossimo. La forma è tabellare e ricorda le tabelle della verità da cui è derivata.



I passi per la progettazione di una macchina di Huffman



STG



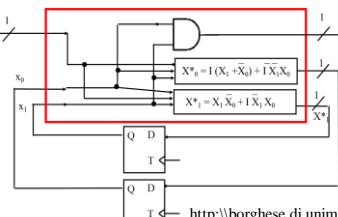
I X	I		Y
0c	0c	10c	Nulla
10c	10c	20c	Nulla
20c	20c	30c	Nulla
30c	0c	10c	Caffè

STT

STT codificata codifico in binario
stati e ingressi

I X - $X_1 X_0$	I		Y
	0c (0)	10c (1)	
0c (00)	0c (00)	10c (01)	Nulla (0)
10c (01)	10c (01)	20c (10)	Nulla (0)
20c (10)	20c (10)	30c (11)	Nulla (0)
30c (11)	0c (00)	10c (01)	Caffè (1)

Macchina di Huffman





Sommario



- FSM
- **Controllore di un semaforo**
- Riconoscitore di stringhe
- Latch come FSM



Controllore di un semaforo Intelligente



Si vuole realizzare una macchina intelligente che controlli il traffico lungo due direttive: Nord-Sud ed Est-Ovest, attraverso un semaforo.

Si supponga che il semaforo possa essere solamente: verde per la direttrice nord-sud (rosso per la direttrice est-ovest) o rosso per la direttrice nord-sud (verde per la direttrice est-ovest).

Alla macchina che controlla il semaforo è associato un sistema di video-sorveglianza che fornisce in ogni intervallo di tempo le seguenti informazioni: 1) non ci sono auto nelle due direzioni, 2) ci sono solo auto in una delle due direzioni, 3) ci sono auto in entrambe le direzioni.

Il sistema di controllo, non cambia la luce del semaforo se non passano auto o passano auto solo lungo la direzione in cui il semaforo è verde.

Il sistema di controllo accende il semaforo lungo una direttrice EO (NS) quando passano auto solo in quella direzione EO (NS).

Quando passano auto in entrambe le direzioni, la macchina che controlla il semaforo commuta.



Stato, Input, Output del semaforo



Input: {Nulla, Auto_{NS}, Auto_{EO}, Auto_{Both}}

Output: {Verde_{NS}, Verde_{EO}}

Stato: ?

Stato iniziale:

f(X,I) = ?

g(X) = ?



STG del semaforo



Input: {Nulla, Auto_{NS}, Auto_{EO}, Auto_{Both}}

Output: {Verde_{NS}, Verde_{EO}}

Stato: ?

Stato iniziale: ?

f(X,I) = ?

g(X) = ?





STG del semaforo



Input: {Nulla, Auto_{NS}, Auto_{EO}, Auto_{Both}}

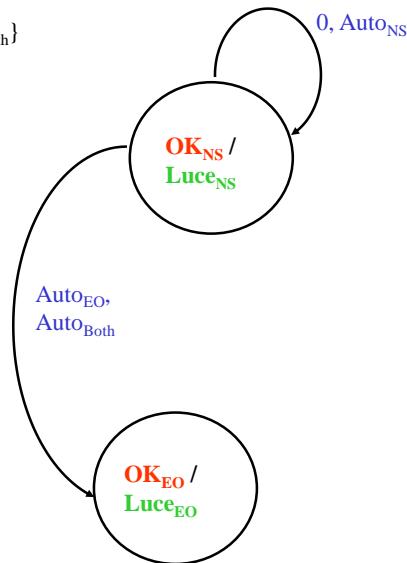
Output: {Verde_{NS}, Verde_{EO}}

Stato: {OK_{NS}, OK_{EO}}

Stato iniziale: OK_{NS}

f(X,I) = ?

g(X) = ?



STG del semaforo



Ingresso: {Nulla, Auto_{NS}, Auto_{EO}, Auto_{Both}}

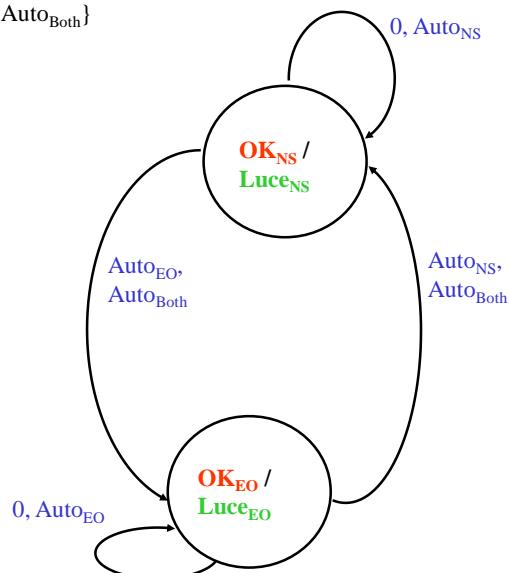
Uscita: {Verde_{EO}, Verde_{NS}}

Stato: {OK_{NS}, OK_{EO}}

Stato iniziale: X₀ = OK_{NS}

f(X,I) = ?

g(X) = ?

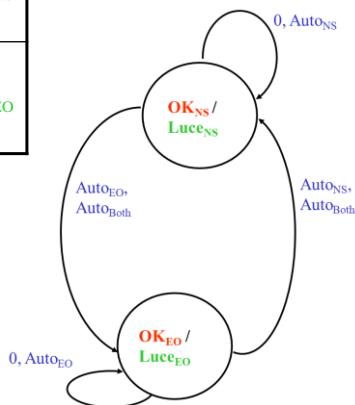




STT del semaforo



I X	Input				Uscita
	Nulla	Auto _{NS}	Auto _{EO}	Auto _{Both}	
OK _{NS}	OK _{NS}	OK _{NS}	OK _{EO}	OK _{EO}	Luce _{NS}
OK _{EO}	OK _{EO}	OK _{NS}	OK _{EO}	OK _{NS}	Luce _{EO}



A.A. 2023-2024

13/49

<http://borghese.di.unimi.it/>

STT del semaforo codificata



I X	Input					Output
	Nulla = 00	Auto _{NS} = 01	Auto _{EO} = 10	Auto _{Both} = 11		
OK _{NS} = 0	OK _{NS} = 0	OK _{NS} = 0	OK _{EO} = 1	OK _{EO} = 1	Verde _{NS} = 1	
OK _{EO} = 1	OK _{EO} = 1	OK _{NS} = 0	OK _{EO} = 1	OK _{NS} = 0	Verde _{EO} = 0	

Ingresso: {Nulla, Auto_{NS}, Auto_{EO}, Auto_{Both}} = {00, 01, 10, 11}Uscita: {Verde_{EO}, Verde_{NS}} = {0, 1}Stato: {OK_{NS}, OK_{EO}} = {0, 1}Stato iniziale: X₀ = OK_{NS} 0

f(X,I) = ?

g(X) = ?

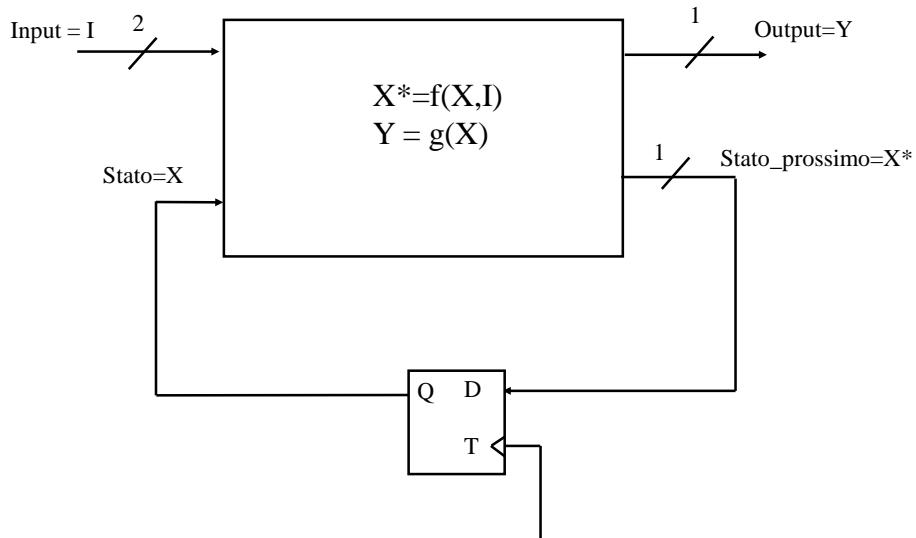
A.A. 2023-2024

14/49

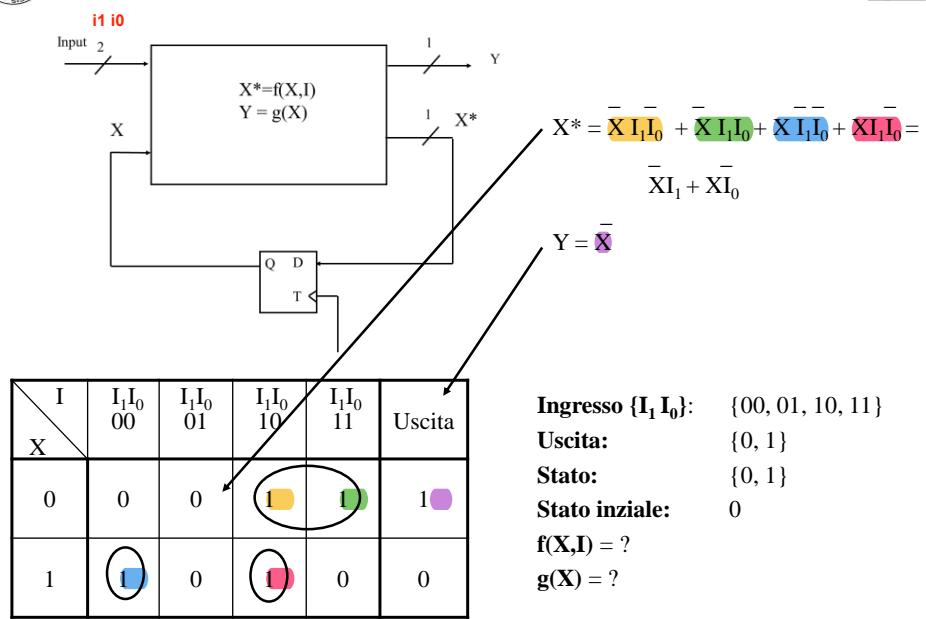
<http://borghese.di.unimi.it/>



Dimensionamento della MSF del semaforo

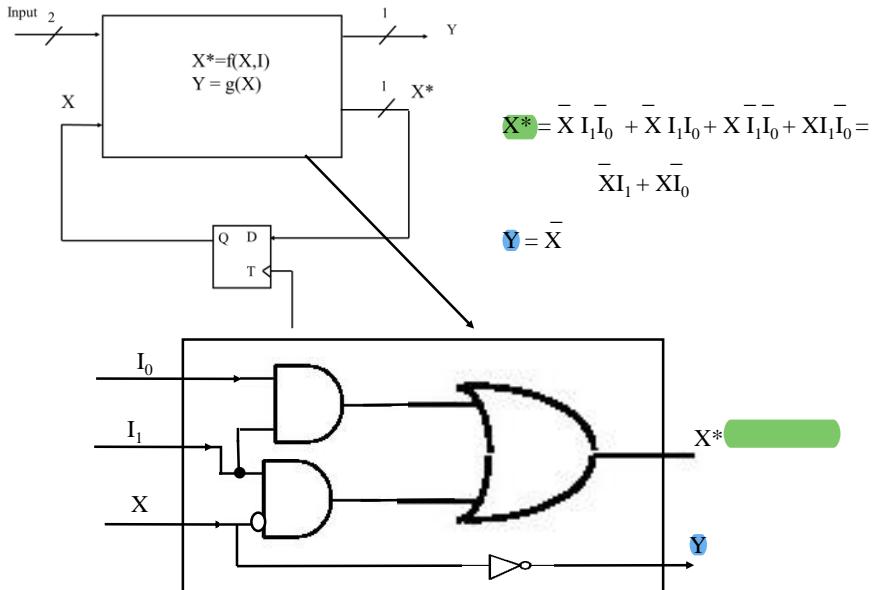


Sintesi della MSF del semaforo





Sintesi del circuito della MSF del semaforo

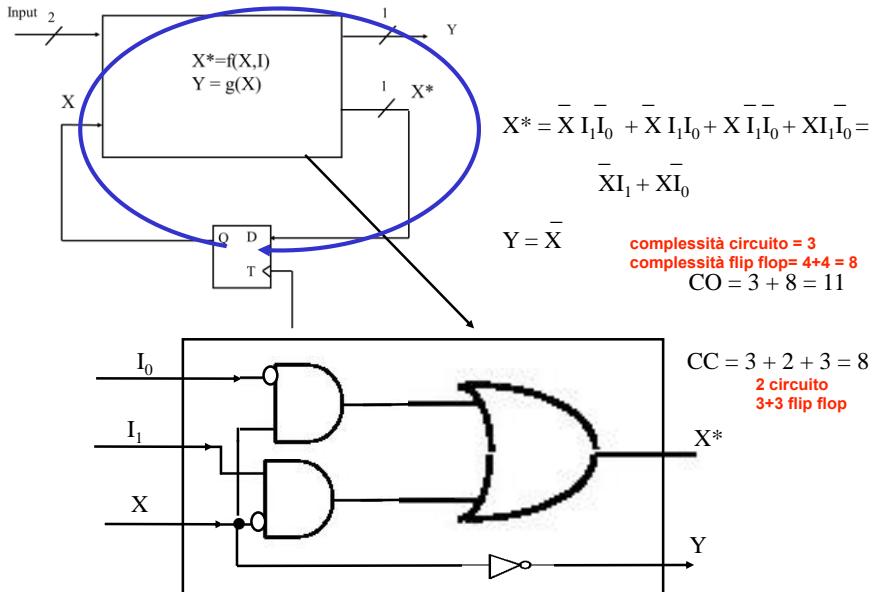


A.A. 2023-2024

17/49

<http://borghese.di.unimi.it/>

Valutazione del circuito



A.A. 2023-2024

18/49

<http://borghese.di.unimi.it/>



I passi della progettazione di una MSF



Il committente fornisce le specifiche di funzionamento.

Definizione delle variabili di Input, Stato e Output. Definizione degli insiemi di simboli che possono essere assunti dalle variabili di Input e di Output.

Costruzione dello STG => Definizione dell'insieme di simboli che possono essere assunti dallo stato.

Costruzione della STT => Definizione implicita delle funzioni stato prossimo ed uscita.

Codifica della STT => Definizione del numero di bit per Input, Stato e Output.

STT Codificata => Circuiti combinatori che sintetizzano le funzioni $f(X,I)$ e $g(X)$.

Realizzazione della macchina di Huffman => Aggiunta degli elementi di sincronizzazione (stato prossimo -> stato presente).



Sommario



- FSM
- Controllore di un semaforo
- **Riconoscitore di stringhe**
- Latch come FSM



Riconoscitore di stringhe



La macchina analizza una stringa che viene composta inserendo un carattere alla volta alla destra della stringa precedentemente formata.

Quando la parte terminale della stringa contiene i caratteri “AAB”, la macchina invia un segnale di stringa riconosciuta.

La macchina accetta in ingresso un carattere alfabetico alla volta (A-Z + nulla).

La macchina è sincronizzata, ed in ogni periodo di tempo accetta al massimo un carattere in ingresso. Se non riceve nulla, la stringa costruita fino a quel momento rimane la stessa (carattere «null»).

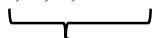
Sintetizzare la macchina di Huffman.



Stato, Input, Output



Input: {Null, A, B, C, D, E, Z} è poco efficiente



non partecipano alla stringa “AAB”

Input: {Null, A, B, Altro}

Output: {Niente, Stringa_AAB}

Stato: {Nulla,}

Stato iniziale: Nulla = “”

$X^* = f(X, I) = ?$

$Y = g(X) = ?$



STG del riconoscitore di stringhe



Input: {Null, A, B, Altro}

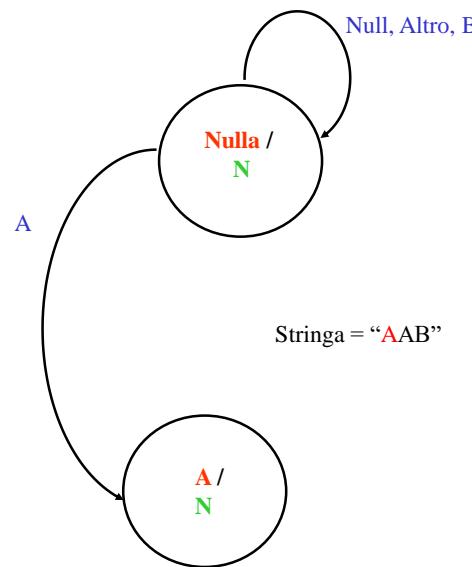
Output: {N, S}

Stato: {Nulla, A,}

Stato iniziale: Nulla

$X^* = f(X, I) = ?$

$Y = g(X) = ?$



Stringa = “AAB”



STG del riconoscitore di stringhe



Input: {Null, A, B, Altro}

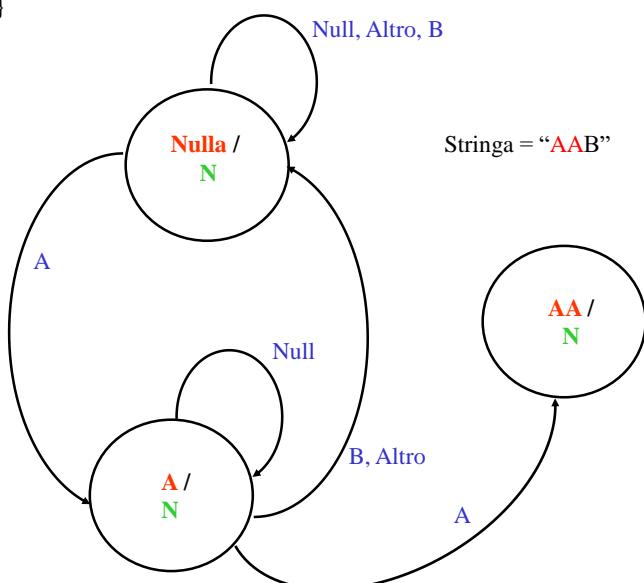
Output: {N, S}

Stato: {Nulla, A, AA,}

Stato iniziale: Nulla

$X^* = f(X, I) = ?$

$Y = g(X) = ?$



Stringa = “AAB”



STG del riconoscitore di stringhe



Input: {Null, A, B, Altro}

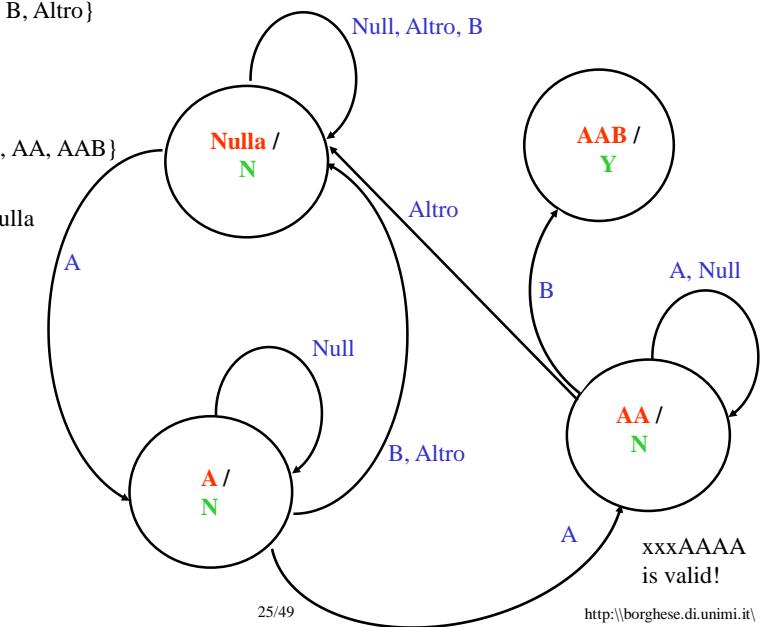
Output: {N, S}

Stato: {Nulla, A, AA, AAB}

Stato iniziale: Nulla

$X^* = f(X, I) = ?$

$Y = g(X) = ?$



STG del riconoscitore di stringhe



Input: {Null, A, B, Altro}

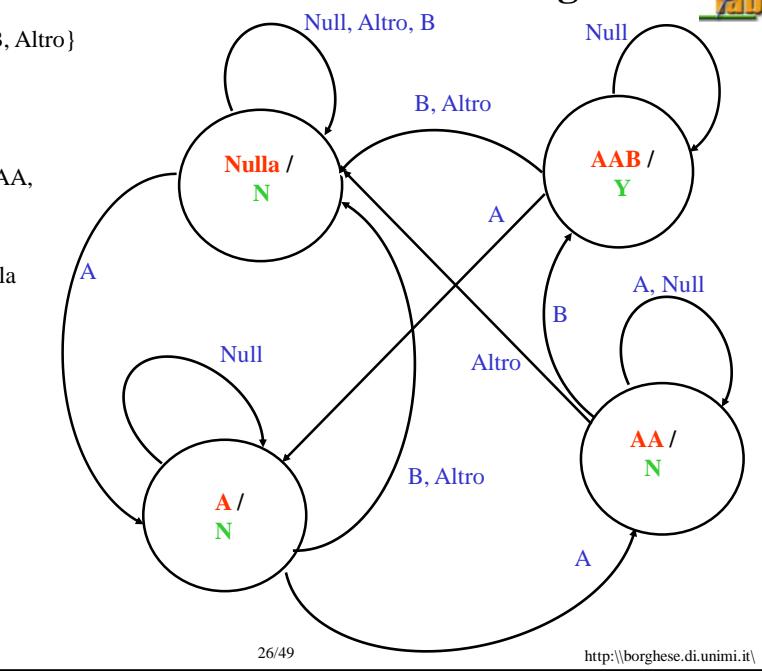
Output: {N, S}

Stato: {Nulla, A, AA, AAB}

Stato iniziale: Nulla

$X^* = f(X, I) = ?$

$Y = g(X) = ?$





Stato, Input, Output



Ingresso: {Null, A, B, Altro}

Uscita: {Niente, Stringa_AAB}

Stato: {Nulla, A, AA, AAB}

Stato iniziale: Nulla

$X^* = f(X, I) = ?$

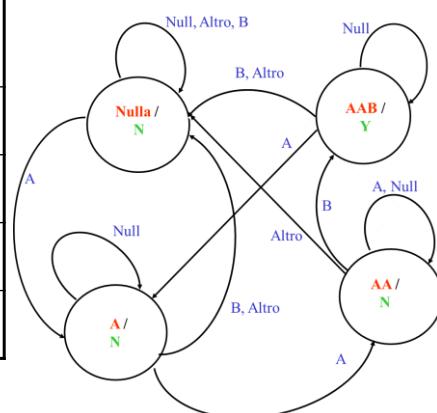
$Y = g(X) = ?$



STT Semantica



I X \ I	Null (0)	"A" (01)	"B" (10)	"Altro"(11)	Y
Nulla	Nulla	"A"	Nulla	Nulla	No
"A"	"A"	"AA"	Nulla	Nulla	No
"AA"	"AA"	"AA"	"AAB"	Nulla	No
"AAB"	"AAB"	"A"	Nulla	Nulla	Si



$$X = \{\text{Nulla, "A", "AB", "AAB"}\} \quad f(X, I) = ?$$

$$I = \{\text{Null, "A", "B", Altro}\} \quad g(X) = ?$$

$$Y = \{\text{Nulla, Stringa_AAB}\}$$

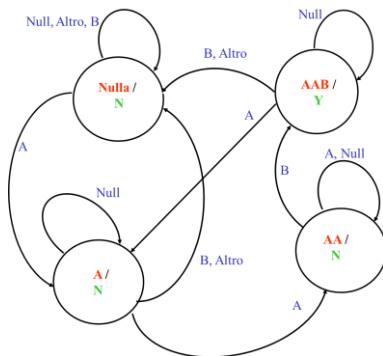
$$X_{\text{ini}} = \text{Nulla}$$



STT Codificata



I X \ I	Null (00)	"A" (01)	"B" (10)	"Altro" (11)	Y
I X	Nulla (00)	"A" (01)	Nulla (00)	Nulla (00)	No (0)
"A" (01)	"A" (01)	"AA" (10)	Nulla (00)	Nulla (00)	No (0)
"AA" (10)	"AA" (10)	"AA" (10)	"AAB" (11)	Nulla (00)	No (0)
"AAB" (11)	"AAB" (11)	"A" (01)	Nulla (00)	Nulla (00)	Si (1)



$$X = \{\text{Nulla, "A", "AA", "AAB"}\} = \{00, 01, 10, 11\}$$

$$f(X, I) = ?$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$g(X) = ?$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

$$X_{\text{ini}} = \text{Altro}$$

A.A. 2023-2024

29/49

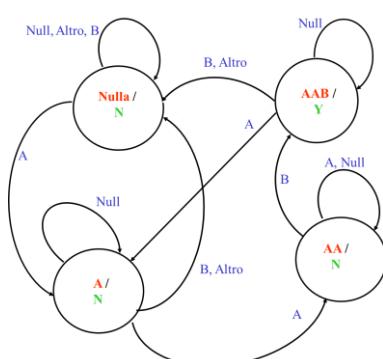
<http://borghese.di.unimi.it/>



STT Codificata - sintesi



I X = $\{X_1 X_0\}$	Null (00)	"A" (01)	"B" (10)	"Altro" (11)	Y	
I X = $\{X_1 X_0\}$	(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(00)	(1)



$$X = \{\text{Nulla, "A", "AB", "ABB"}\} = \{00, 01, 10, 11\}$$

$$f(X, I) = ?$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$g(X) = ?$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

$$X_{\text{ini}} = \text{Altro}$$

A.A. 2023-2024

30/49

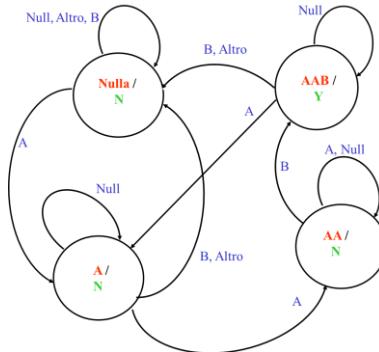
<http://borghese.di.unimi.it/>



STT Codificata - sintesi



I	I = {I₁ I₀}				Y
X= {X₁X₀}	Null	“A”	“B”	“Altro”	
	(00)	(01)	(10)	(11)	
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$X = \{\text{Nulla, "A", "AB", "AAB"}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

X_{ini} = Altro

A.A. 2023-2024

31/49

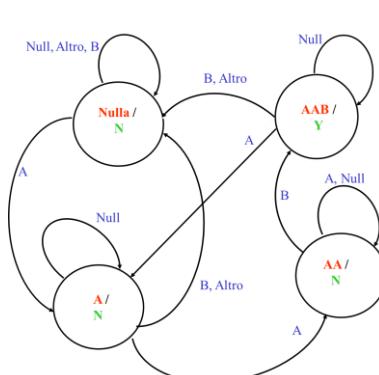
<http://borghese.di.unimi.it/>



STT Codificata - sintesi



I	I = {I₁ I₀}				Y
X= {X₁X₀}	Null	“A”	“B”	“Altro”	
	(00)	(01)	(10)	(11)	
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$X = \{\text{Nulla, "A", "AB", "AAB"}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null, "A", "B", "Altro"}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Si}\} = \{0, 1\}$$

X_{ini} = Altro

A.A. 2023-2024

32/49

<http://borghese.di.unimi.it/>

$$f(X, I) = \begin{cases} X_0^* = f_1(X, I) \\ X_1^* = f_2(X, I) \end{cases}$$

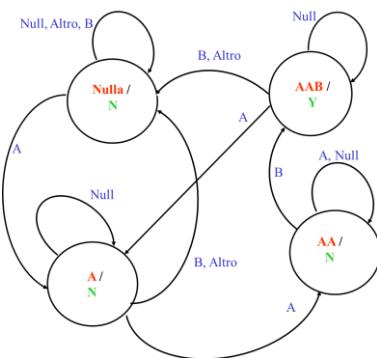
$$g(X) = ?$$



STT Codificata – sintesi X^*_0



I X= {X₁X₀}	Null (00)	I = {I ₁ I ₀ } “A” (01)	“B” (10)	“Altro” (11)	Y
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$X = \{\text{Nulla}, \text{“A”}, \text{“AB”}, \text{“ABB”}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null}, \text{“A”}, \text{“B”}, \text{“Altro”}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Sì}\} = \{0, 1\}$$

X_{ini} = Altro

$$X^*_0 = \bar{X}_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + \bar{X}_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 = \bar{X}_1 \bar{I}_1 (X_0 \oplus I_0) + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0$$

A.A. 2023-2024

33/49

<http://borgheze.di.unimi.it/>

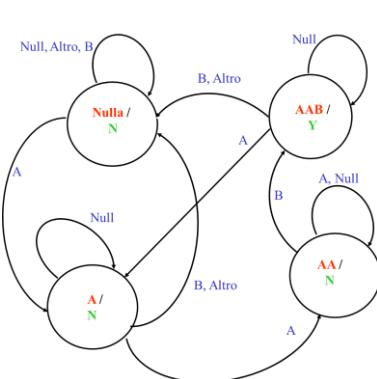
$$\begin{aligned} f(X, I) &= X^*_0 = f_1(X, I) \\ g(X) &=? \end{aligned}$$



STT Codificata – sintesi X^*_1



I X= {X₁X₀}	Null (00)	I = {I ₁ I ₀ } “A” (01)	“B” (10)	“Altro” (11)	Y
(00)	(00)	(01)	(00)	(00)	(0)
(01)	(01)	(10)	(00)	(00)	(0)
(10)	(10)	(10)	(11)	(00)	(0)
(11)	(11)	(01)	(00)	(00)	(1)



$$X = \{\text{Nulla}, \text{“A”}, \text{“AB”}, \text{“ABB”}\} = \{00, 01, 10, 11\}$$

$$I = \{\text{Null}, \text{“A”}, \text{“B”}, \text{“Altro”}\} = \{00, 01, 10, 11\}$$

$$Y = \{\text{No, Sì}\} = \{0, 1\}$$

X_{ini} = Altro

$$X^*_1 = \bar{X}_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 + X_1 \bar{X}_0 \bar{I}_1 \bar{I}_0 = \bar{X}_1 \bar{I}_1 (\bar{X}_0 + I_0)$$

A.A. 2023-2024

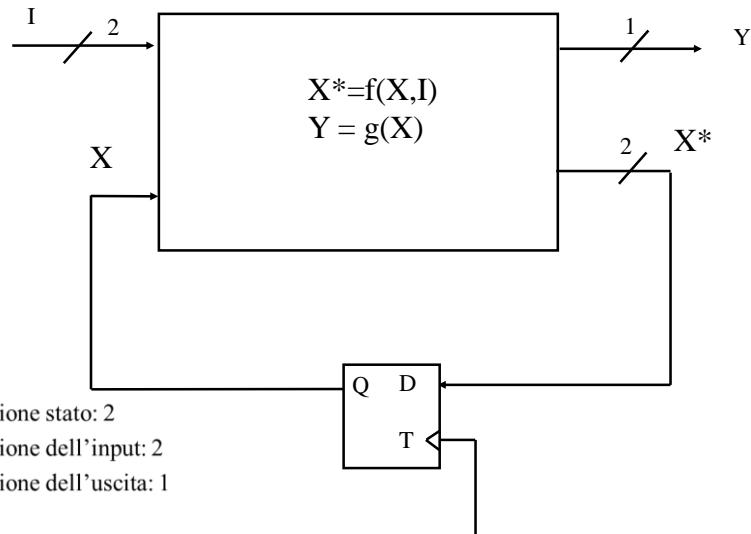
34/49

$$\begin{aligned} f(X, I) &= X^*_1 = f_1(X, I) \\ g(X) &=? \end{aligned}$$

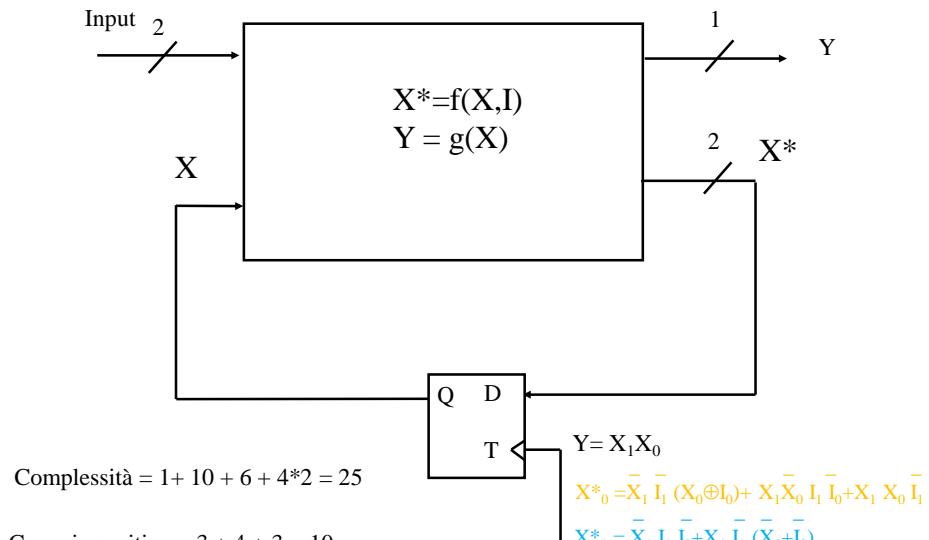
<http://borgheze.di.unimi.it/>



Dimensionamento della MSF



Caratteristiche della FSM





Esercizio: sommatore sequenziale



Si vuole realizzare un sommatore controllato da una macchina a stati finiti. Il sommatore somma due numeri su N bit, commando ad ogni istante una coppia dei bit del numero, da destra a sinistra.

L'uscita, ad ogni istante, rappresenta la somma dei bit presenti all'istante precedente più il riporto all'istante precedente.

Il sistema memorizza ad ogni istante la somma dei bit presenti ed il riporto che va all'istante successivo.



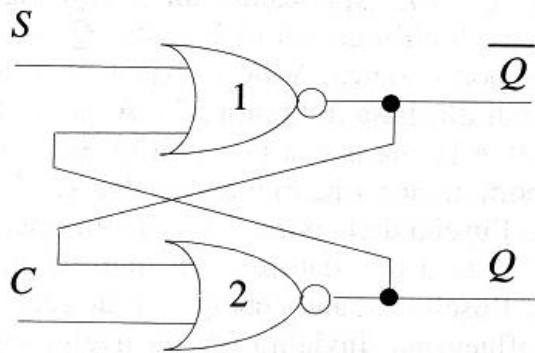
Sommario



- FSM
- Controllore di un semaforo
- Riconoscitore di stringhe
- **Latch come FSM**



Il latch SC asincrono



Specifiche:

if ($S=C=0$) then $Q^*=Q$ status quo

if ($S=1, C=0$) then $Q^*=1$ set

if ($S=0, C=1$) then $Q^*=0$ clear

if ($S=1, C=1$) then ... dipende. Nell'implementazione di cui sopra non si deve presentare.



Verso lo STG del latch SC asincrono



$$I = \{00, 01, 10, 11\} = \{S, C\}$$

$$X = \{0, 1\} = Q$$

$$Y = \{0, 1\} = Q$$

$$X_o = 0$$

$$X^* = Q^* = f(Q, I) = f(X, I) ?$$

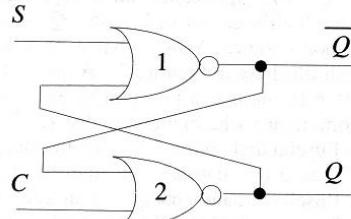
$$Y = X = Q$$

Q è l'uscita del latch: stato presente, $X = Q$.

Q^* è il valore dell'uscita al tempo successivo: stato prossimo, $X^* = Q^*$.

Non viene specificato cosa succeda quando $S = C = 1$.

L'uscita del latch coincide con il suo stato (interno): $Y = Q$





STG del latch SC asincrono



$$I = \{00, 01, 10, 11\} = \{S, C\}$$

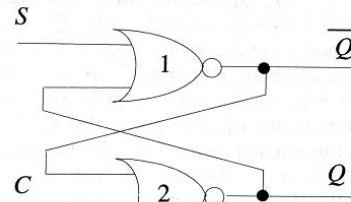
$$X = \{0, 1\} = Q$$

$$Y = \{0, 1\} = Q$$

$$X_o = 0$$

$$X^* = Q^* = f(Q, I) = f(X, I) ?$$

$$Y = Q = X$$



Q è l'uscita del latch: stato presente, X .

Q^* è il valore dell'uscita al tempo successivo: stato prossimo, X^* .

0 / 0

Non viene specificato cosa succeda quando $S = C = 1$.

L'uscita del latch coincide con il suo stato (interno).



STG del latch SC asincrono



$$I = \{00, 01, 10, 11\} = \{S, C\}$$

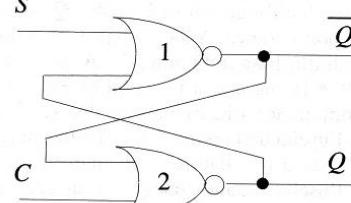
$$X = \{0, 1\} = Q$$

$$Y = \{0, 1\} = Q$$

$$X_o = 0$$

$$X^* = Q^* = f(Q, I) = f(X, I) ?$$

$$Y = Q = X$$



Q è l'uscita del latch: stato presente, X .

Q^* è il valore dell'uscita al tempo successivo: stato prossimo, X^* .

00,
01
0 / 0
 $Y = X = Q$

10

1 / 1

Non viene specificato cosa succeda quando $S = C = 1$.

L'uscita del latch coincide con il suo stato (interno).



STG del latch SC asincrono



$$I = \{00, 01, 10, 11\} = \{S, C\}$$

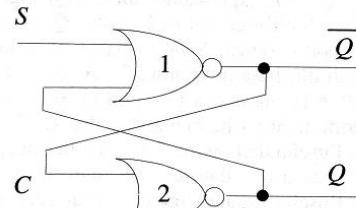
$$X = \{0, 1\} = Q$$

$$Y = \{0, 1\} = Q$$

$$X_o = 0$$

$$X^* = Q^* = f(Q, I) = f(X, I) ?$$

$$Y = Q = X$$

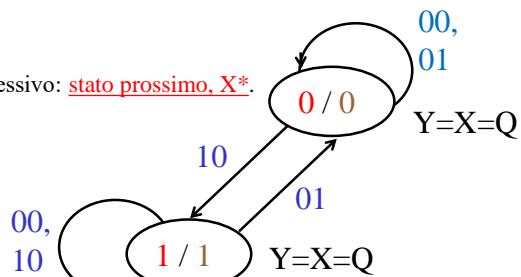


Q è l'uscita del latch: stato presente, X .

Q^* è il valore dell'uscita al tempo successivo: stato prossimo, X^* .

Non viene specificato cosa succeda quando $S = C = 1$.

L'uscita del latch coincide con il suo stato (interno).



STT del latch SC asincrono



$$Q^* = f(Q, S, C)$$

Variabile di
Stato
(interna)

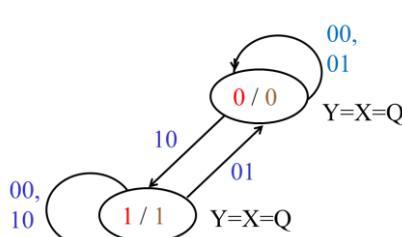
Variabili di
Ingresso
(esterne)

Q	SC =		SC =		SC =	
	00	01	10	11	10	11
0	0	0	1	X	1	X
1	1	0	1	X	0	1

No change
($Q^* = Q$)

Clear
Reset

Set





STT del latch SC asincrono



$$I = \{00, 01, 10, 11\}$$

$$X = \{0, 1\}$$

$$Y = \{0, 1\}$$

$$X_0 = 0$$

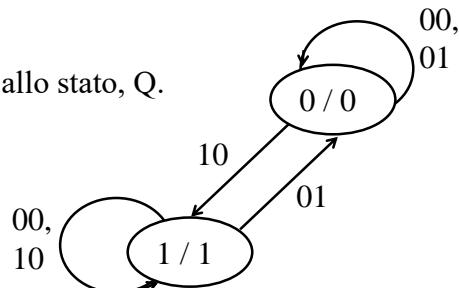
$$f(X, I) = ?$$

$$g(X) = ?$$

I Q	SC = 00	SC = 01	SC = 10	SC = 11	Y = Q
0	0	0	1	X=0	0
1	1	0	1	X=0	1

$Y = g(X) = Q$ L'uscita è uguale allo stato, Q.

$$X^* = f(X, I) = \bar{Q} \bar{S} \bar{C} + \bar{S} \bar{C}$$



A.A. 2023-2024

45/49

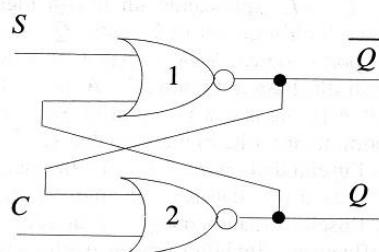
<http://borghese.di.unimi.it/>



Macchina di Huffman per il latch SC asincrono

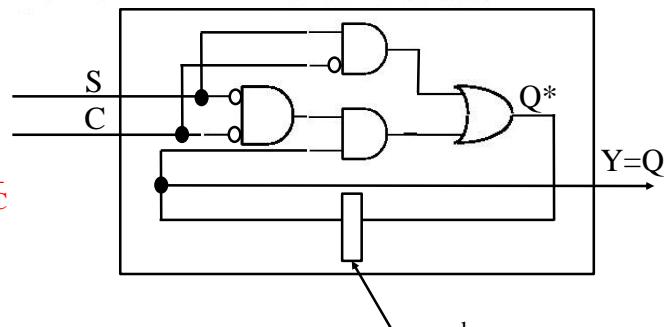


$$S$$



$$Y = g(X) = Q$$

$$Q^* = f(X, I) = \bar{Q} \bar{S} \bar{C} + \bar{S} \bar{C}$$



A.A. 2023-2024

46/49

<http://borghese.di.unimi.it/>



Esercizio: Controllore di un semaforo Intelligente avanzato



Si vuole realizzare una macchina intelligente che controlli il traffico lungo due direttive: Nord-Sud ed Est-Ovest, attraverso un semaforo.

Si supponga che il semaforo possa essere solamente: verde per la direttiva nord-sud (rosso per la direttiva est-ovest) o rosso per la direttiva nord-sud (verde per la direttiva est-ovest).

Alla macchina che controlla il semaforo è associato un sistema di video-sorveglianza che fornisce in ogni intervallo di tempo le seguenti informazioni: 1) non ci sono auto nelle due direzioni, 2) ci sono solo auto in una delle due direzioni, 3) ci sono auto in entrambe le direzioni.

Il sistema di controllo, non cambia la luce del semaforo se non passano auto o passano auto solo lungo la direzione in cui il semaforo è verde.

Il sistema di controllo accende il semaforo lungo una direttiva EO (NS) quando passano auto solo in quella direzione EO (NS).

Quando passano auto in entrambe le direzioni, in un primo periodo di tempo, la macchina che controlla il semaforo non commuta. Se nel periodo di tempo successivo sono presenti auto in entrambe le direzioni, solo allora la macchina che controlla il semaforo commuta.



Altri esercizi



Costruire una macchina a stati finiti (di Moore), in grado di individuare all'interno di una parola di 0 e 1 le seguenti configurazioni: 1010 e 1110. Le configurazioni si possono concatenare (e.g. 101010 da' uscita vera, al secondo e terzo 0). Stato iniziale 00.

Costruire una macchina a stati finiti (di Moore), con due ingressi, x_1 e x_2 , che fornisce 1 quando negli ultimi 3 istanti si è verificata la seguente configurazione:

	$t = -2$	$t = -1$	$t = 0$
x_1	0	X	1
x_2	x	1	0

Stato iniziale $x_1 = 0$ $x_2 = 0$.

Costruire un venditore di bibite che distribuisce una bibita quando si raggiungono i 35 cents inseriti. Non dà resto.

Costruire un contatore che riceva in ingresso i seguenti input:

- Reset

- Conteggio

e che fornisca in uscita il conteggio attuale (costruire un contatore che conti fino a 4 o fino a 8).



Sommario



- FSM
- Controllore di un semaforo
- Riconoscitore di stringhe
- Latch come FSM