Corso di Architettura degli Elaboratori
Modulo di Assembly

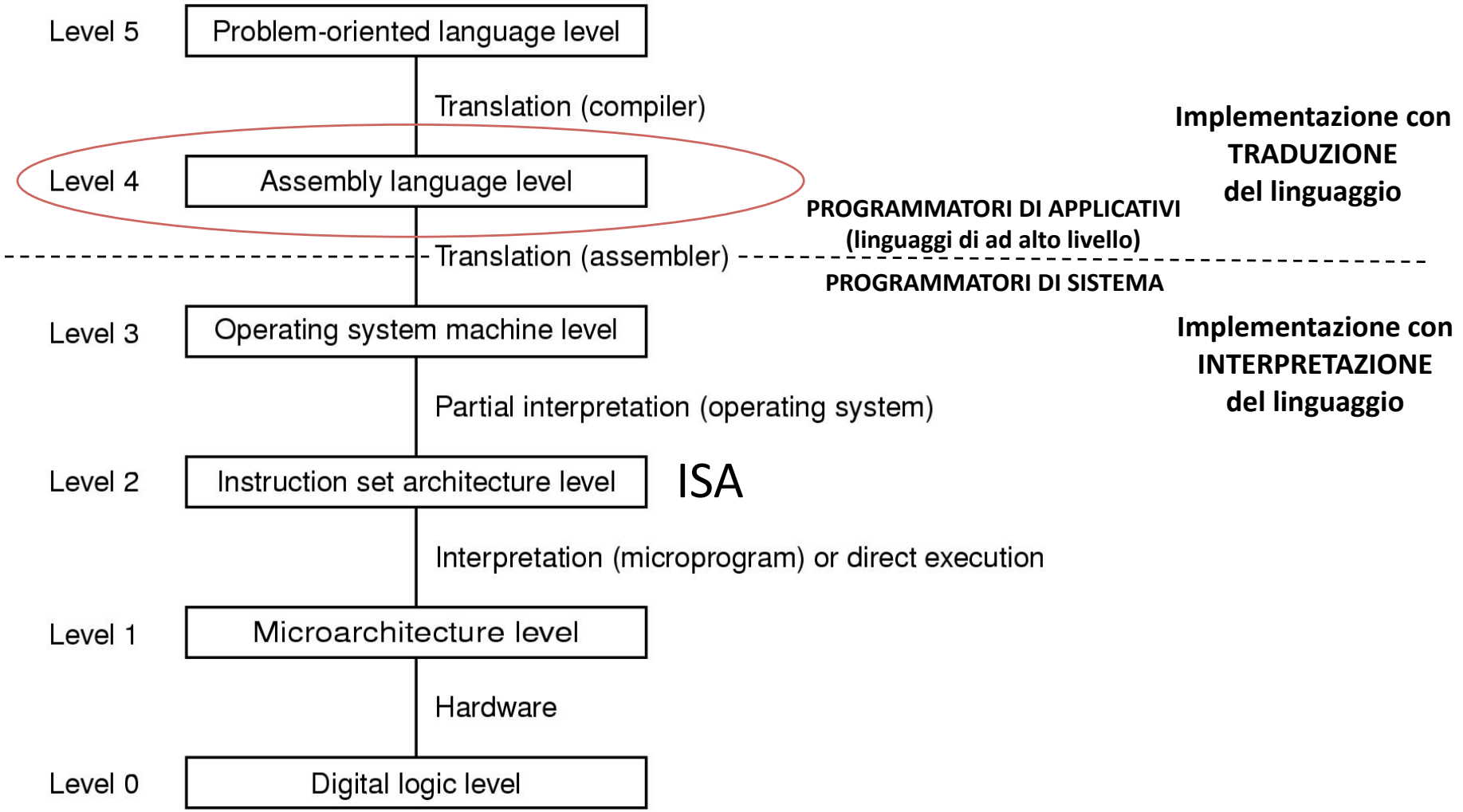
CONCETTI DI BASE

Bruno Iafelice

Università di Bologna

iafelice at cs(dot)unibo(dot)it

LINGUAGGIO ASSEMBLATIVO



```
int void () {
        int i;
        int cont=0;
        for (i=0; i<100; i++)
        cont=cont+i;
        return 1;
}
```

Programma scritto in un Linguaggio Ad Alto Livello (C)

Compilatore

```
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $24, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    $0, -4(%ebp)
        movl    $0, -8(%ebp)
        jmp     .L2
.L3:
        movl    -8(%ebp), %eax
        leal    -4(%ebp), %edx
        addl    %eax, (%edx)
        leal    -8(%ebp), %eax
        incl    (%eax)
.L2:
        cmpl    $99, -8(%ebp)
        jle     .L3
        movl    $0, %eax
        leave
        ret
```

Programma in Linguaggio Assembly (386/NASM)

Assemblatore

```
.....
00 00 00 00 00 00 00 00 1b 00 00 00 01 00 00 00
06 00 00 00 00 00 00 00 34 00 00 00 46 00 00 00
00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
21 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00
7c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
04 00 00 00 00 00 00 00 27 00 00 00 08 00 00 00
03 00 00 00 00 00 00 00 7c 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
2c 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00
7c 00 00 00 2d 00 00 00 00 00 00 00 00 00 00 00
01 00 00 00 00 00 00 00 35 00 00 00 01 00 00 00
00 00 00 00 00 00 00 00 a9 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
11 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00
a9 00 00 00 45 00 00 00 00 00 00 00 00 00 00 00
.....
```

Programma in Linguaggio Macchina

Cortesia: Marco Di Felice

Istruzioni macchina (ISA)

Esempio di possibile istruzione macchina:

1. Il campo Op-code specica il tipo di operazione richiesta
2. Il campo Dest specica l'operando di destinazione
3. Il campo Source specica l'operando sorgente

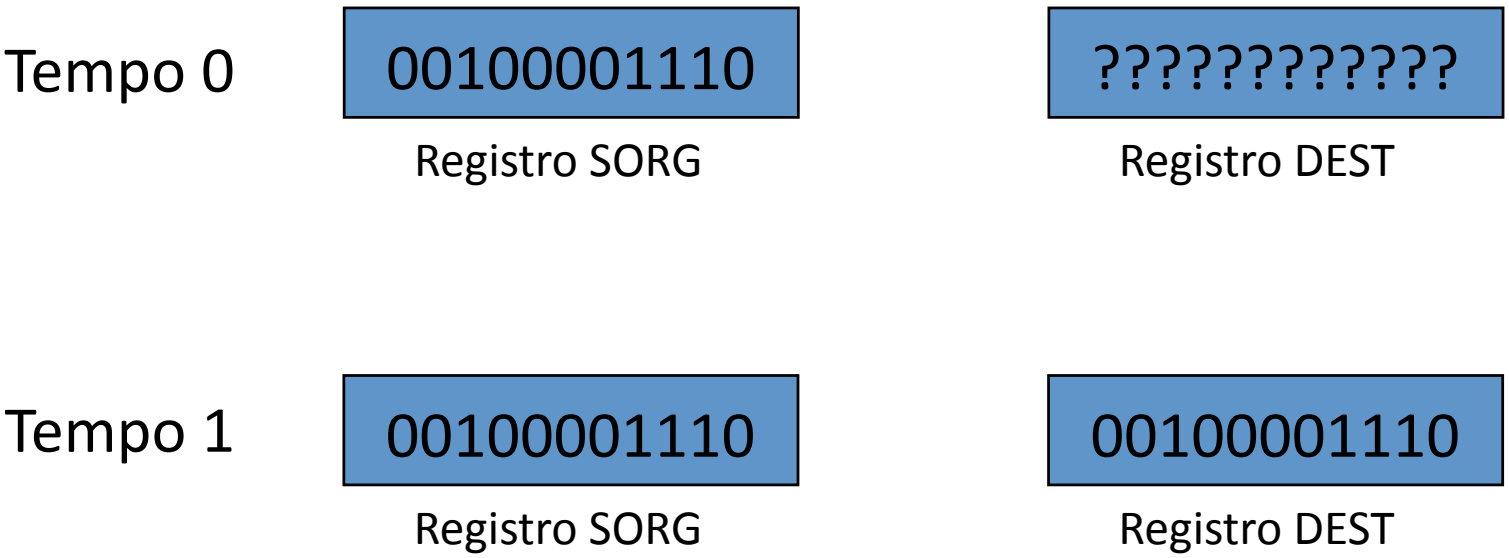
- Un programma a livello ISA e una lunga serie di stringhe binarie che specificano le istruzioni da seguire ed i loro operandi.

- Per scrivere un codice in linguaggio macchina e necessario:
  - Conoscere l'architettura della macchina
  - Conoscere i dettagli relativi alle singole istruzioni:
    - Codici numerici e formato delle istruzioni
    - Rappresentazione degli operandi
    - Gestire direttamente gli indirizzi in memoria per il riferimento ai dati

- Risultato: la programmazione a questo livello risulta estremamente complessa, perciò tutti gli elaboratori dispongono di un linguaggio assemblativo (*assembly language*) e di software che traducono ogni istruzione del linguaggio assemblativo in linguaggio macchina (assemblatore).

Istruzioni Assembler

```
mov DEST,SORG
```



Linguaggio assembler

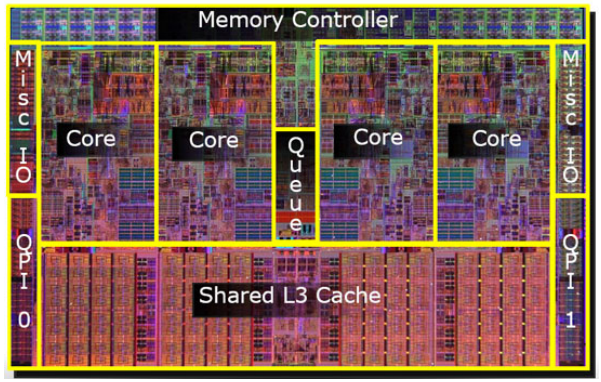
- La relazione tra le istruzioni del linguaggio Assembler e quelle del linguaggio macchina è 1:1, cioè l'assembler è una rappresentazione "umana" del binario della macchina
- Si usano nomi **mnemonici** per facilitare il programmatore a ricordare le istruzioni: ADD MUL MOV
- Il programmatore assembler può controllare TUTTA la macchina hardware controllo completo
- L'Assembler è FORTEMENTE legato al particolare hardware per cui si stà scrivendo il programma: scarsa portabilità tra macchine diverse, solo se appartenenti a generazioni successive di una stessa famiglia, es. da 8088 a 486 o Pentium
- **Quando si programma in Assembler?**

Perchè l'assembly 8088?

- Intel 8088 ha semplice architettura, semplice assembler
- Compatibilità a ritroso nella famiglia di processori Intel: eseguiamo il codice Assembler per 8088 se macchine Pentium
- Cosa serve: conoscenza approfondita di cosa c'è dietro: Architettura
  - Organizzazione della memoria
  - Modalità di indirizzamento
  - Registri
  - Istruzioni a disposizione (risorse *computazionali*)

CONCETTI DI BASE SUI MICROPROCESSORI

Elaboratore elettronico: macchina programmabile in grado di eseguire operazioni logico aritmetiche



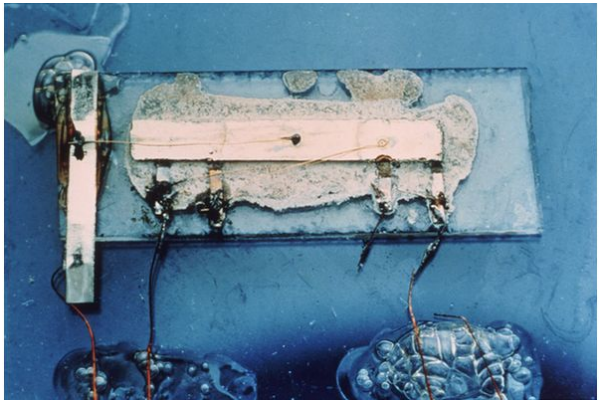
1952 EDVAC, John von Neumann dal progetto ENIAC Electronic Numerical Integrator Automatic Computer del 1943

2010 INTEL Core i7 (4 core)

Cosa è cambiato? La tecnologia

Valvola termoionica (o tubo a vuoto) 1940

Transistor 1947



Circuito integrato 1958

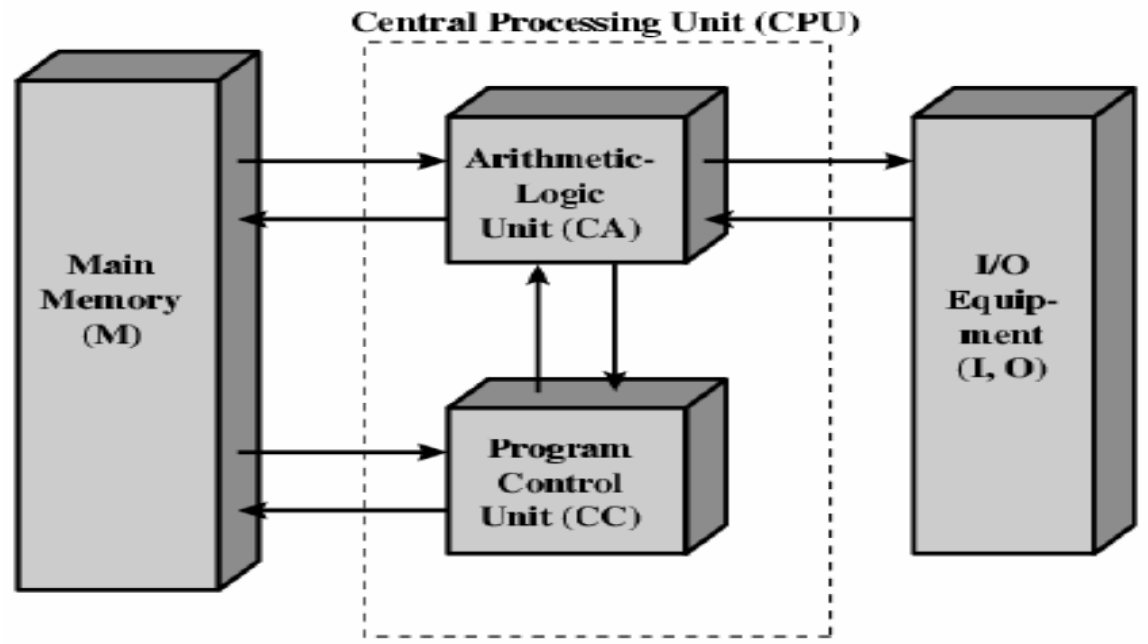
Integrazione su larga scala

VLSI 100K-100M transistor su chip 1978

1969 primo microcomputer Intel 4004

1968 viene fondata Intel

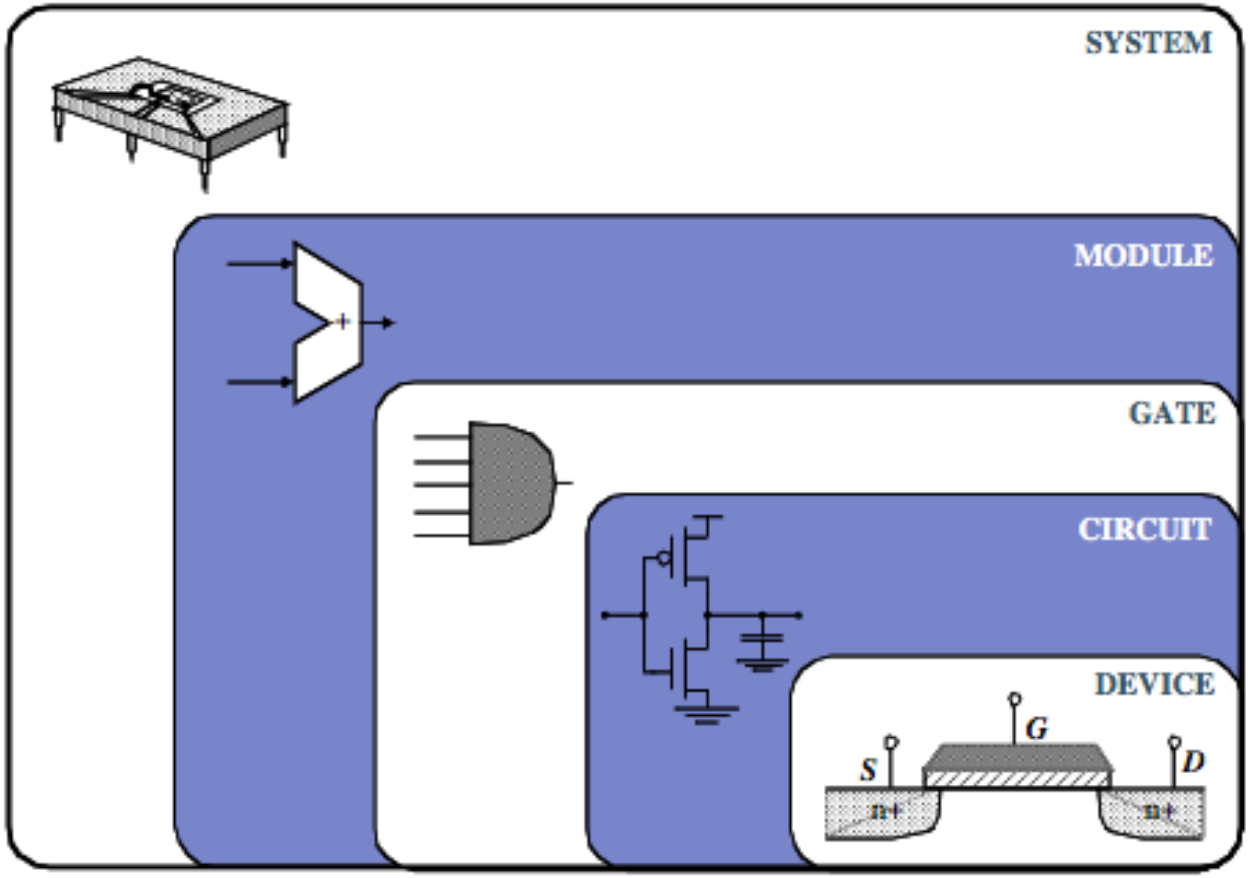
Cosa non è cambiato? L'architettura



von Neumann architecture

- Code and data stored together
- Arithmetic unit
- Load and control unit
- I/O

Come è fatto un micro computer?



Adapted from "Digital Integrated Circuits, Rabaey, Chandrakasan and Nikolic, Copyright 2003 Prentice Hall/Pearson" for classroom presentations and handouts only.

Definizione: Sistema digitale

- E’ un qualsiasi sistema (elettronico) in cui le informazioni vengono rappresentate in forma binaria (numerica), utilizzando cioè dei simboli (0 e 1) e l’elaborazione si basa sull’algebra di Boole

Algebra Booleana

- "cattura l'essenza" degli operatori logici AND, OR e NOT
- consente di trattare in termini esclusivamente algebrici le operazioni insiemistiche dell'intersezione, dell'unione e della complementazione
- Gli operatori dell'algebra booleana sono AND, OR e NOT. Nella descrizione dei circuiti NAND (NOT AND), NOR (NOT OR) e XOR (OR esclusivo)
- Definizione

Se B è un insieme formato da almeno 2 elementi, diciamo **algebra booleana**, avente B come supporto, la struttura algebrica costituita da B, da due operazioni binarie su B, OR e AND, da un'operazione unaria NOT su B e da un elemento particolare di B che indichiamo **0** i quali godono delle seguenti proprietà.

$\forall a, b \in B : a\ AND\ b = b\ AND\ a$. AND è simmetrica

$\forall a, b \in B : a\ OR\ b = b\ OR\ a$. OR è simmetrica

$\forall a \in B : NOT(NOT(a)) = a$. NOT è un'involuzione

$\forall a, b \in B : NOT(a\ OR\ b) = NOT(a)\ AND\ NOT(b)$ valgono le leggi di De Morgan

$\forall a \in B : a\ AND\ \mathbf{0} = \mathbf{0};\ a\ OR\ \mathbf{0} = a$ l'insieme B è *limitato inferiormente*

Per ogni algebra booleana si definisce l'elemento **1** come la negazione, o il *complementare*, dello **0**: **1** := NOT(**0**). Per esso si deducono le proprietà

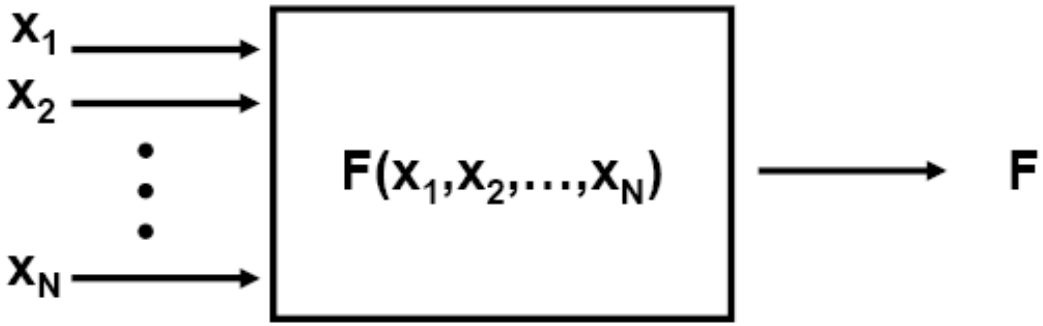
$\forall a \in B : a\ OR\ \mathbf{1} = \mathbf{1}; a\ AND\ \mathbf{1} = a$ (l'insieme B è *limitato superiormente*)

e in particolare

0 AND **1** = **0** ; **0** OR **1** = **1**.

Definizione: Funzione logica

- Relazione che lega N ingressi ad un'uscita
- Oppure che correla un valore ad altri N

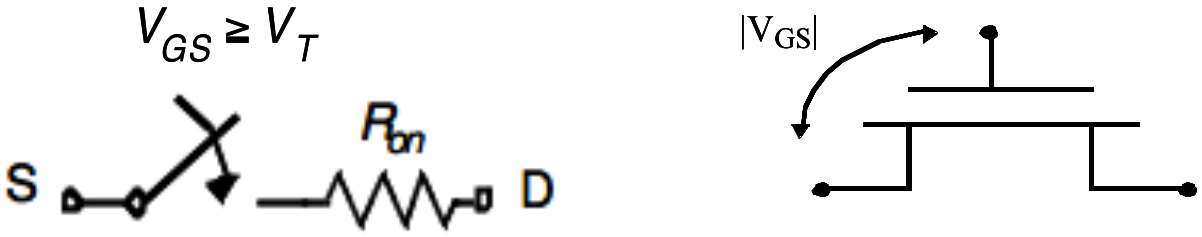


- Es. AND e OR sono funzioni logiche a 2 ingressi

Il transistore MOS

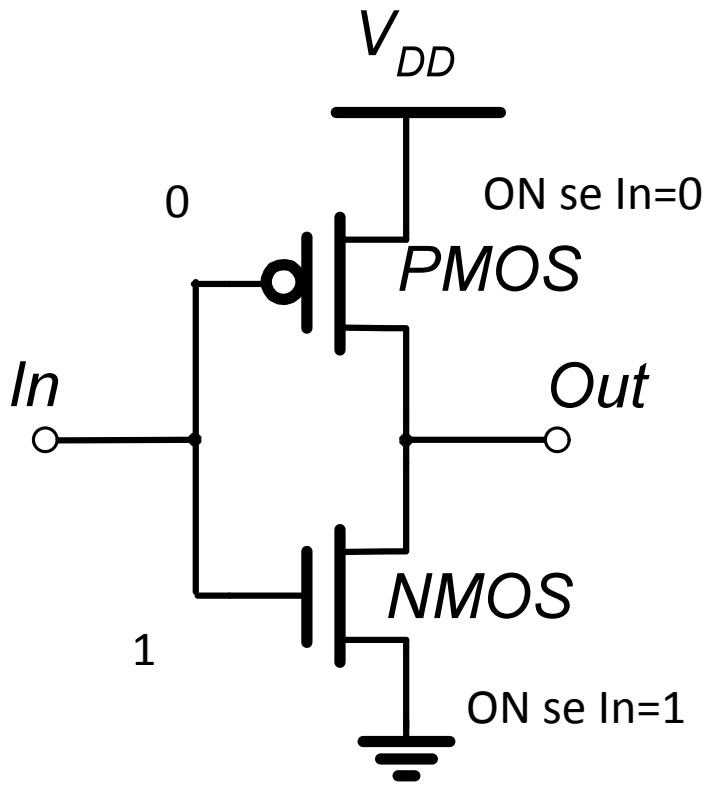
- MOSFET o MOS (metal–oxide–semiconductor field-effect transistor)
- Idea del 1925 realizzata a fine anni 60’
- Incredibile capacità di scalare, cioè di diminuire in dimensioni, e aumentare le prestazioni
- Come funziona?

A Switch! ⟷ An MOS Transistor



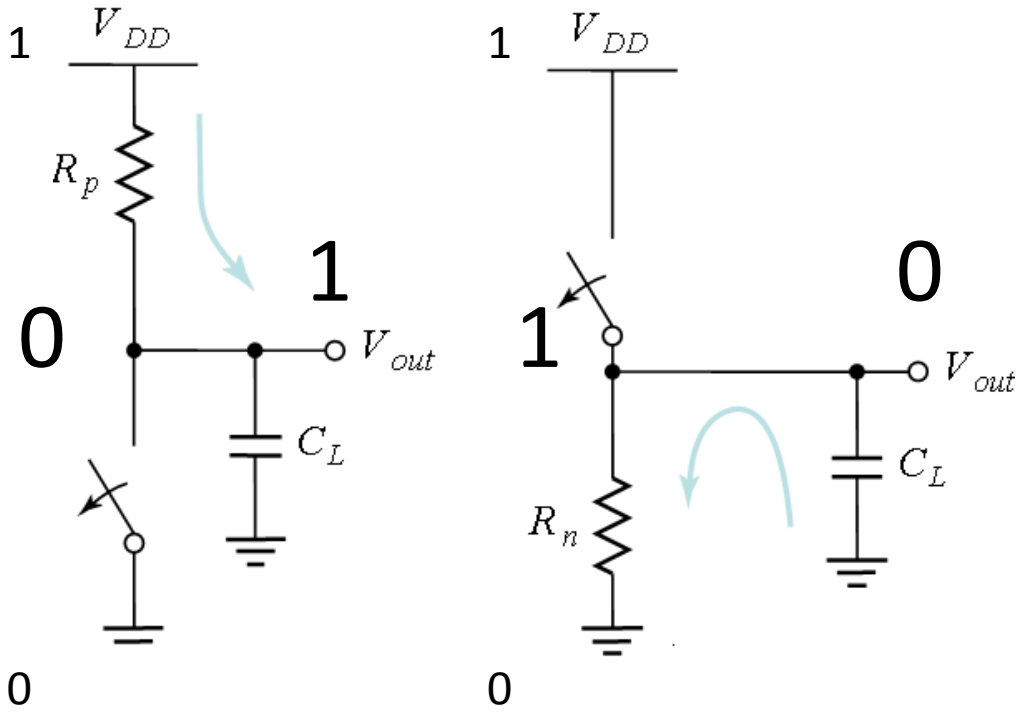
Circuiti CMOS

- CMOS: Complementary MOS
- Basato due tipi di transistori MOS con comportamento complementare
  - p-MOS (attivo con ingresso **basso**)
  - n-MOS (attivo con ingresso **alto**)



Invertitore CMOS

Invertitore CMOS

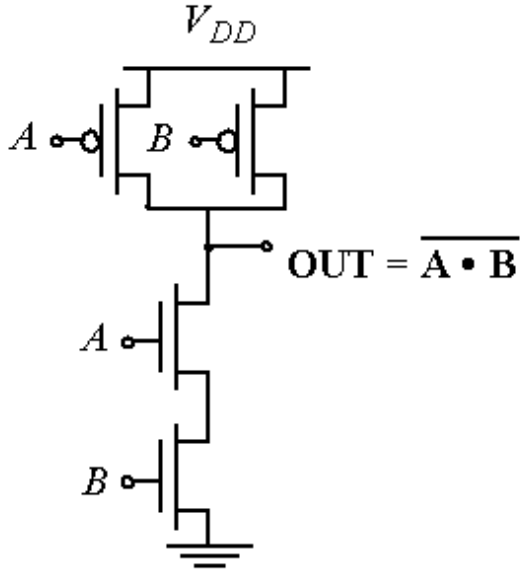


Porte logiche in tecnologia CMOS

Gate NAND

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

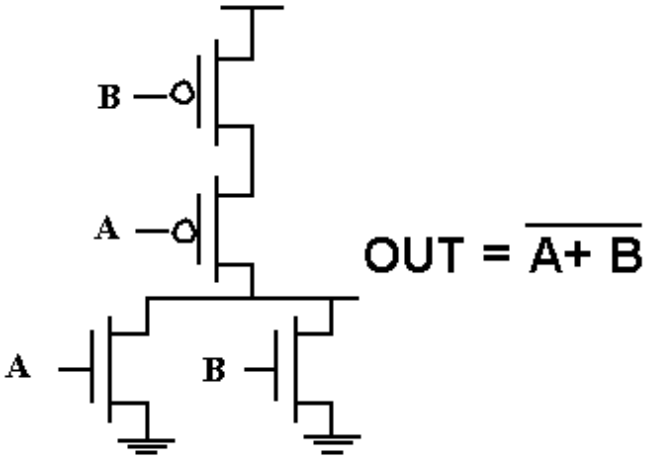
Truth Table of a 2 input NAND gate



Gate NOR

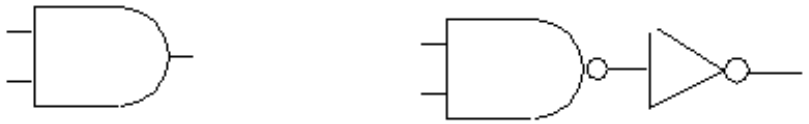
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table of a 2 input NOR gate



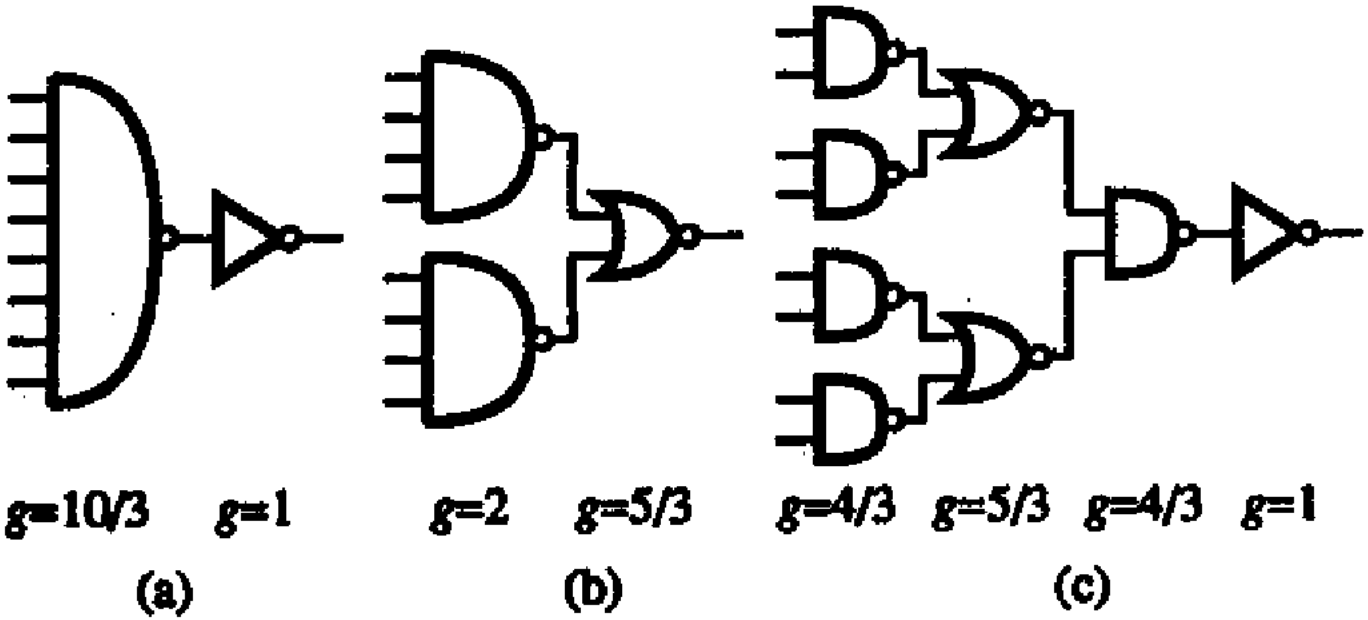
Porte logiche

AND



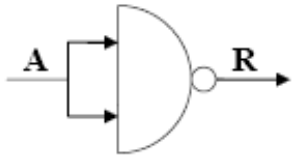
AND = NAND + INV

8-input AND

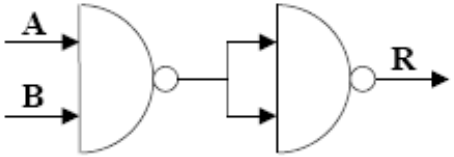


Completezza di NAND (e di NOR)

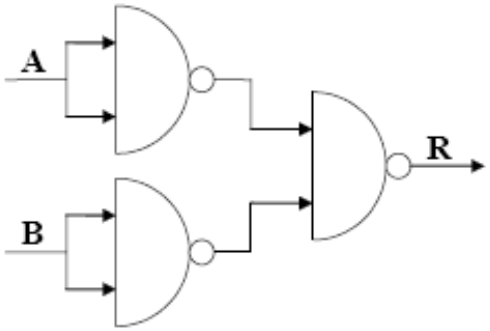
NOT



AND

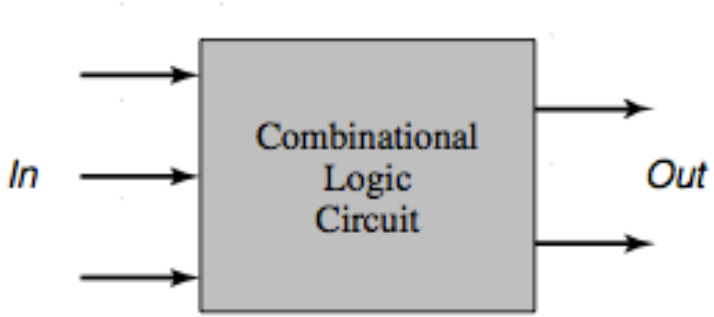


OR



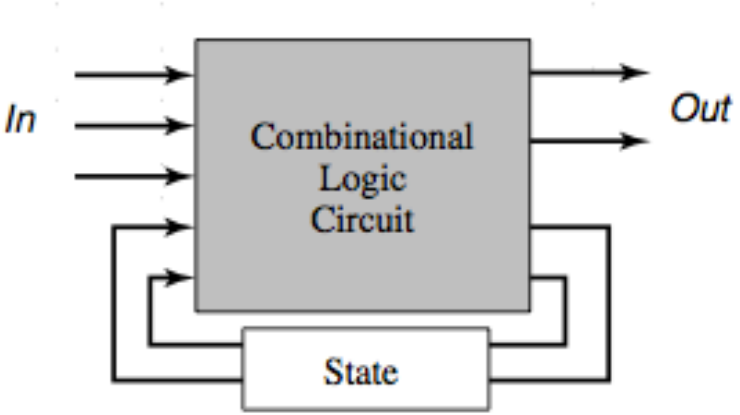
Quindi NAND o NOR sono complete ➔ circuiti con solo porte NAND o solo porte NOR.

Circuiti combinatori e circuiti sequenziali



Combinatorio

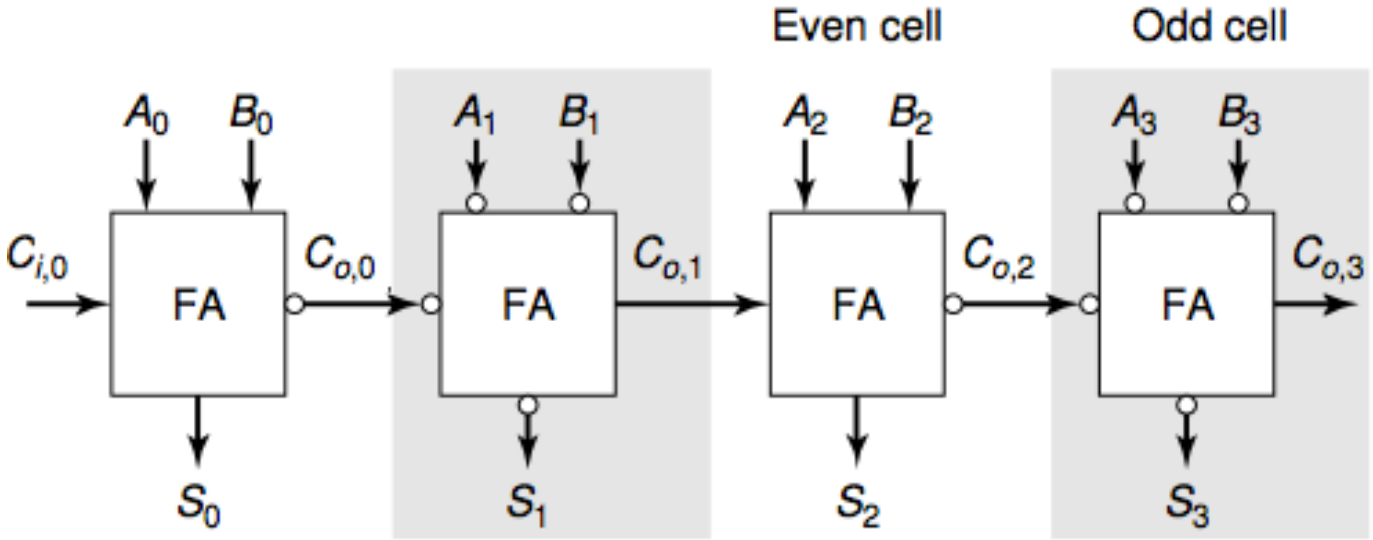
Output = $f(In)$

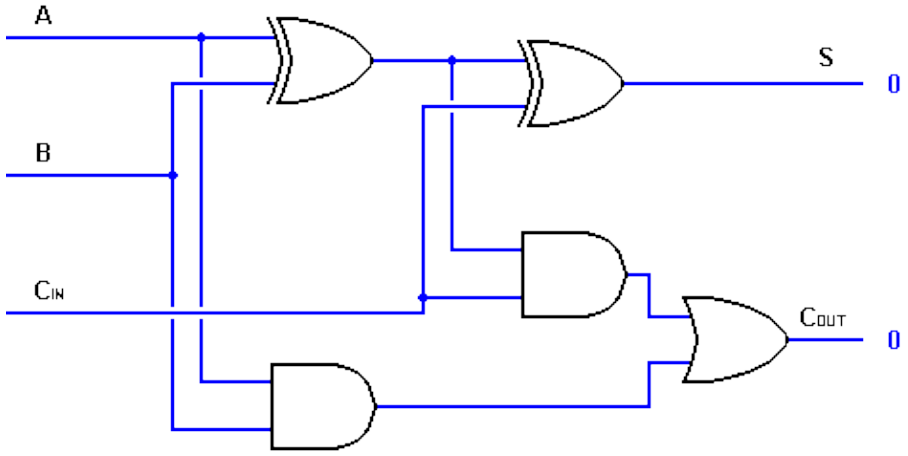


Sequenziale

Output = $f(In, In\_precedente)$

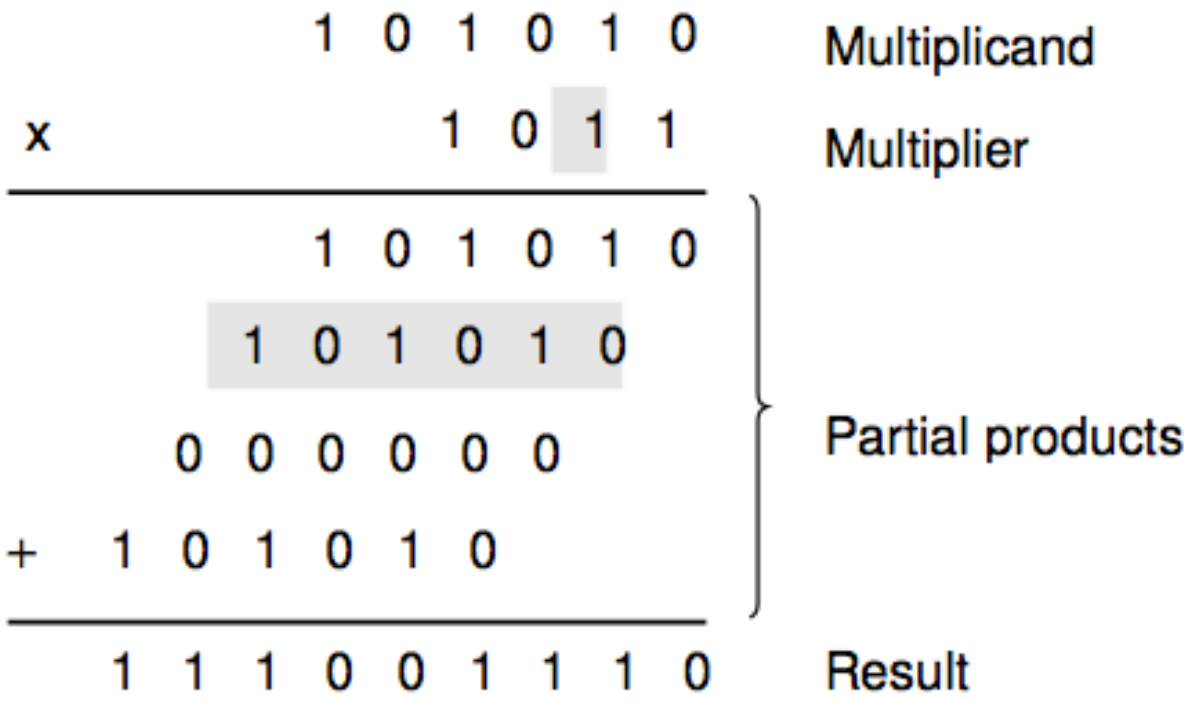
Adder



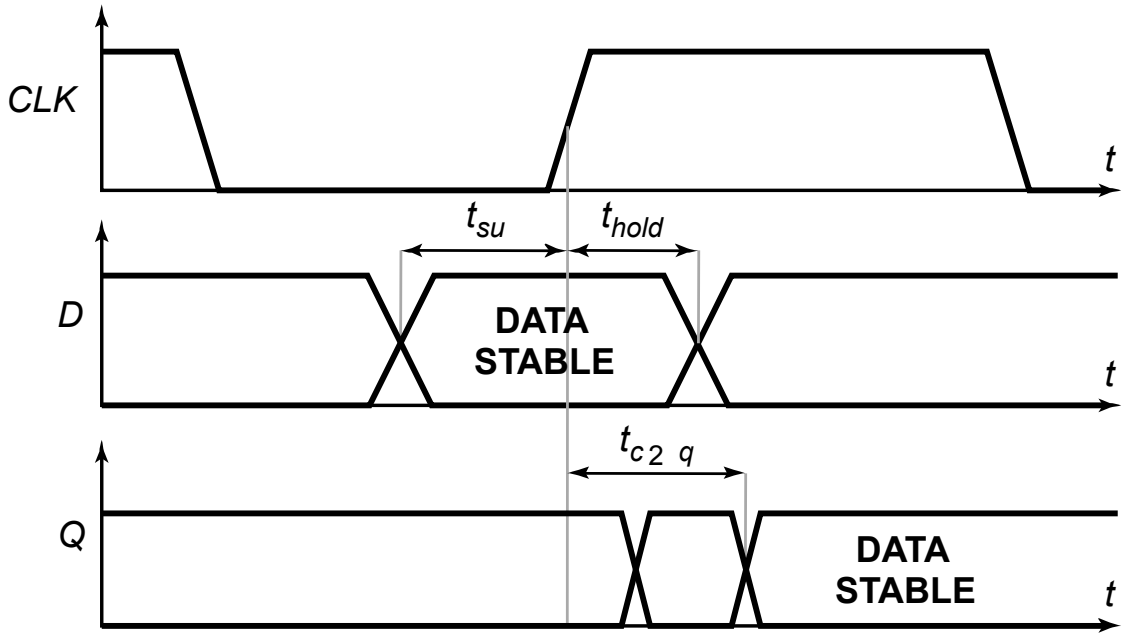


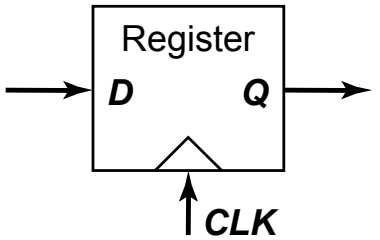
Full Adder (FA)

The Binary Multiplication

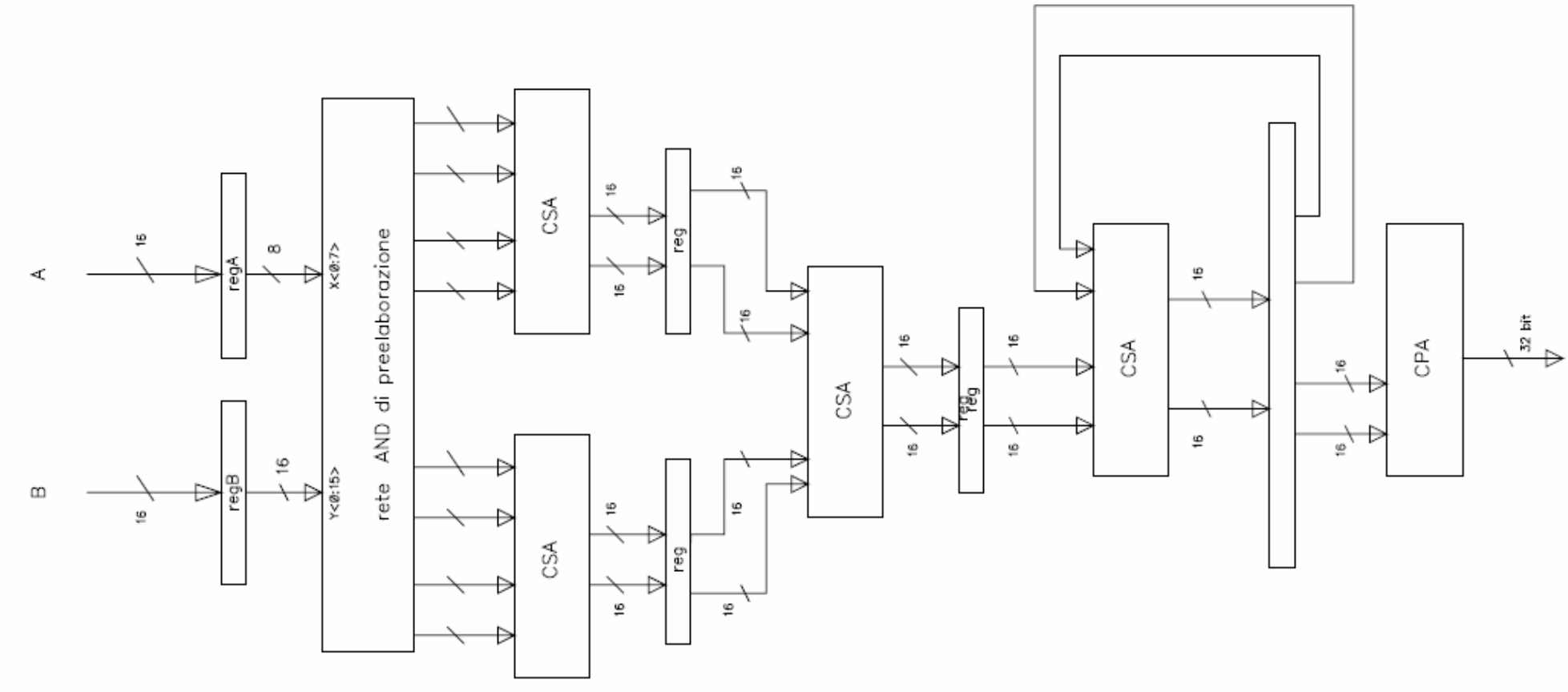


Registri e concetto di sincronia



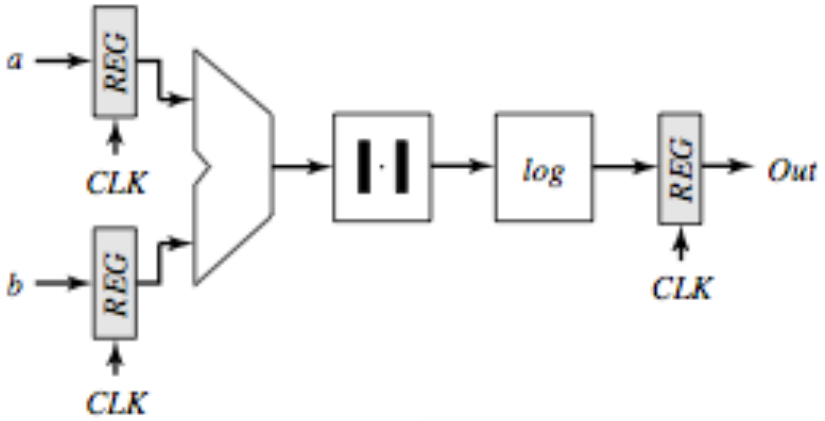


Moltiplicatore

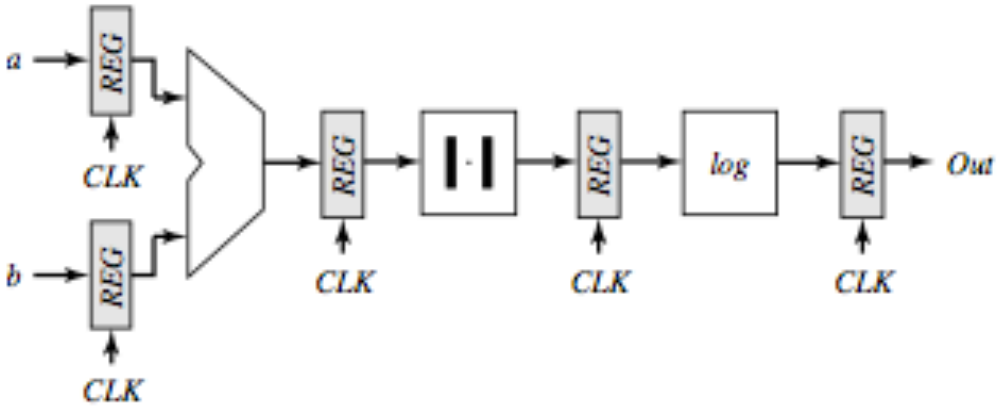


Schema a blocchi del moltiplicatore a 16 bit

Pipelining



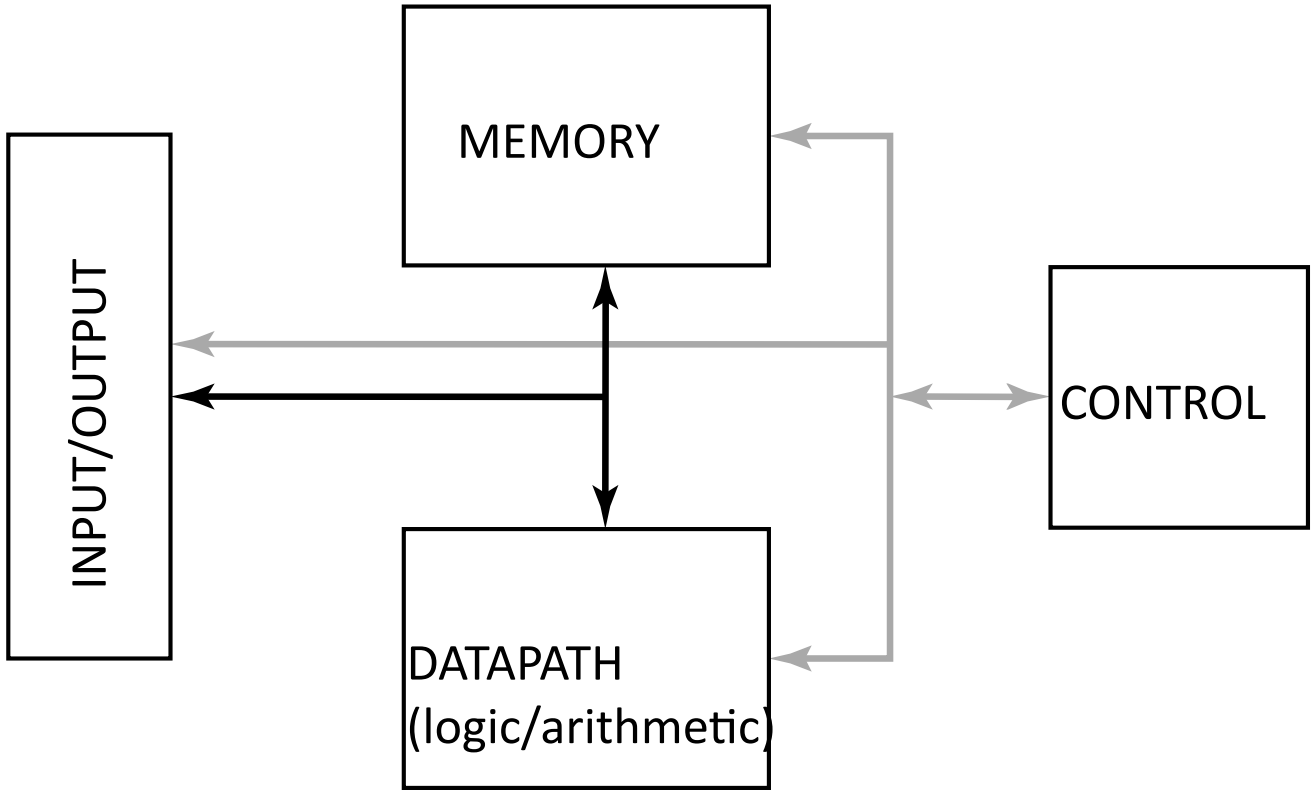
Reference



Pipelined

Clock Period	Adder	Absolute Value	Logarithm
1	$a_1 + b_1$		
2	$a_2 + b_2$	$\lvert a_1 + b_1 \rvert$	
3	$a_3 + b_3$	$\lvert a_2 + b_2 \rvert$	$\log(\lvert a_1 + b_1 \rvert)$
4	$a_4 + b_4$	$\lvert a_3 + b_3 \rvert$	$\log(\lvert a_2 + b_2 \rvert)$
5	$a_5 + b_5$	$\lvert a_4 + b_4 \rvert$	$\log(\lvert a_3 + b_3 \rvert)$

Blocchi architetturali



Arithmetic unit

- adder, multiplier, shifter, comparator, etc.

Memory

- RAM, ROM, Buffers, Shift registers

Control

-Finite state machine

….or microprogram

- Counters

Interconnect

- Switches
- Bus