

I circuiti aritmetici sono i blocchi costruttivi centrali dei calcolatori. I calcolatori e la logica digitale eseguono molte funzioni aritmetiche: addizioni, sottrazioni, confronti, traslazioni, moltiplicazioni e divisioni. L'addizione è una delle operazioni più comuni nei sistemi digitali.

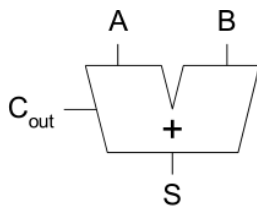
1-Bit Adders

Half Adder (1 bit)

ha due input, A e B, e due uscite, S e Cout.

S rappresenta la somma di A e B. Se sia A sia B hanno valore 1, S è uguale a 2, un valore che non può essere rappresentato con una sola cifra binaria. Di conseguenza, il valore 2 viene rappresentato con un riporto (carry) Cout

Riporta la somma dei due bit dati in input e l'eventuale riporto.



A	B	C _{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

$$C_{out} = AB$$

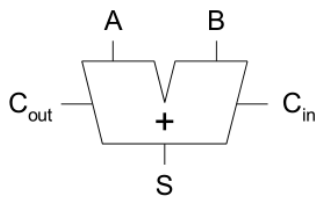
Full adder

E' utilizzato per creare sommatore a più bit, prende in input i due bit di ingresso A e B e l'eventuale riporto C_{in} che è il riporto che proviene dalle cifre meno significative precedenti. Ritorna il valore della somma tra i due bit e un eventuale riporto C_{out} .

somma quindi il riporto eventualmente ottenuto dai due bit di ordine immediatamente inferiore.

$C_{out}=1$ se almeno due dei tre ingressi è uguale a 1

Full Adder



C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

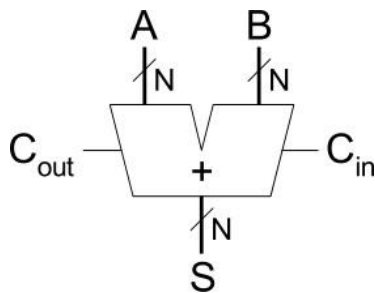
Multibit Adders (CPAs)

Sono sommatore a n bit che si utilizzano quando vogliamo sommare parole di più bit ad esempio quando abbiamo due ingressi A e B a n bit

Hanno in input :

- due parole a n bit A e B
- un riporto C_{in} che si propaga nei bit successivi

In output abbiamo l'eventuale riporto C_{out} e la somma a n bit S degli ingressi A e B



Abbiamo 2 tipologie di multibit adders:

- ripple carry (lento e più ottimizzato per sommare pochi bit)
- carry-lookahead (più veloce e ottimizzato per sommare più bit)

Ripple-Carry Adder

E' formato da una concatenazione di full adder a 1 bit messi in serie dalla cifra meno significativa alla cifra più significativa

Il Cin iniziale è posto a 0. Le prime due cifre binarie meno significative di A e B vengono sommate e nel caso avessimo un Cout esso sarà il Cin delle cifre seguenti, quindi si propaga. Le cifre seguenti saranno sommate con in aggiunta il Cout delle cifre precedenti

Il riporto si propaga lungo la catena

Il principale svantaggio legato a questo sommatore è il progressivo rallentamento all'aumentare di N. Infatti, S₃₁ dipende da C₃₀, che dipende da C₂₉, che dipende a sua volta da C₂₈ e così via fino a risalire a Cin

Si dice quindi che il riporto si propaga a onda attraverso la catena.

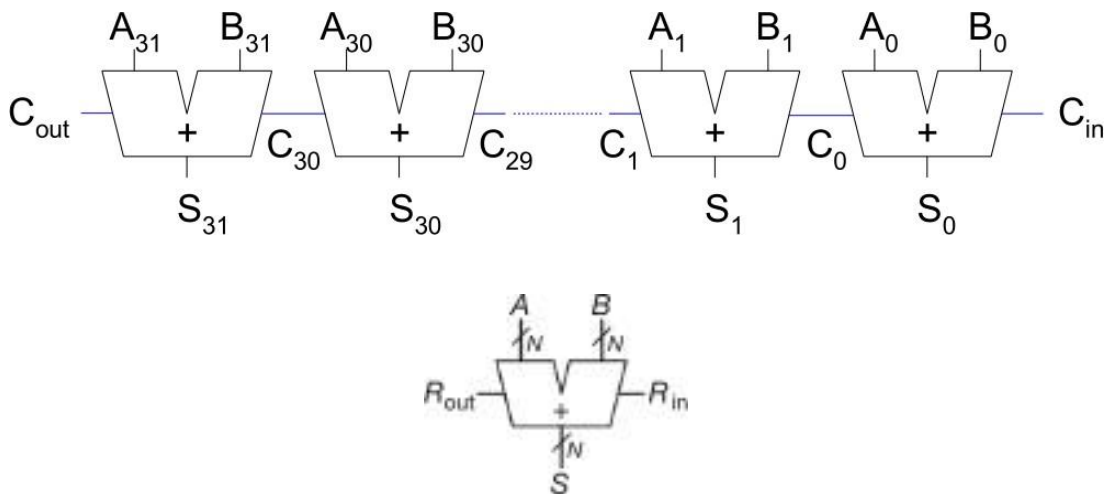
Il ritardo di propagazione nel sommatore, tripple, aumenta all'aumentare del numero di bit coinvolti e quindi dei full adder inseriti.

t_{FA} rappresenta il ritardo di un full adder.

$$t_{\text{ripple}} = N t_{FA}$$

t_{FA} è il ritardo di un singolo 1-bit full adder

N è il numero di 1-bit full adders



Carry lookahead

Questa tipologia di multibit adder minimizza i tempi di ritardo rispetto al ripple carry perché **divide il sommatore in blocchi di k-bit ai quali viene aggiunto un circuito per determinare velocemente il riporto di uscita Cout di ciascun blocco appena è noto il riporto di ingresso Cin**

In questo modo i riporti si propagheranno più velocemente

Per esempio, un sommatore a 32 bit può essere diviso in otto blocchi da 4 bit ciascuno.

I riporti vengono calcolati attraverso le funzioni **Generate G** e **Propagate P**

C_{in}	A_i	B_i	C_{out}
0	0	0	0
1			0
0	0	1	0
1			1
0	1	0	0
1			1
0	1	1	1
1			1

La **configurazione AiBi (11)** viene detta **GENERATE** perchè genera un carry uguale a 1 a prescindere dai valori del Cin

Le **configurazioni AiBi (01,10)** vengono chiamate **PROPAGATE** perchè propagano il valore del Cin sul Cout

La **configurazione AiBi (00)** invece riporta sempre un Cout uguale a 0 a prescindere del Cin

GENERATE

La colonna i di un sommatore genera sicuramente riporto se A_i e B_i sono entrambi uguali a 1. Di conseguenza G_i , cioè il riporto generato dalla colonna i , viene calcolato come

$$G_i = A_i \cdot B_i$$

In generale, per k bit:

$$G_{i:j} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} (\dots G_j) \dots))$$

Un blocco di k bit genera un riporto se la colonna più significativa genera un riporto, oppure se la colonna più significativa propaga un riporto e quella precedente ne genera uno, e così via.

PROPAGATE

Si dice che la colonna i propaga un riporto se produce un riporto di uscita ogniqualvolta ci sia un riporto di ingresso. La colonna i propaga un C_{in} al C_{out} iesimo, **se o A_i o B_i sono uguali a 1**. Di conseguenza,

$$P_i = A_i + B_i$$

In generale, per k bit:

$$P_{i:j} = P_i P_{i-1} P_{i-2} \dots P_j$$

Un blocco propaga un riporto se tutte le colonne del blocco propagano un riporto.

LOGICA DI CARRY

logica di riporto di una specifica colonna del sommatore:

la colonna i del sommatore produce un riporto di uscita C_i se c'è un generate, G_i , o se propaga il riporto di ingresso, $P_i C_{i-1}$.

- Carry out: il carry i (C_i) è data da:

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$

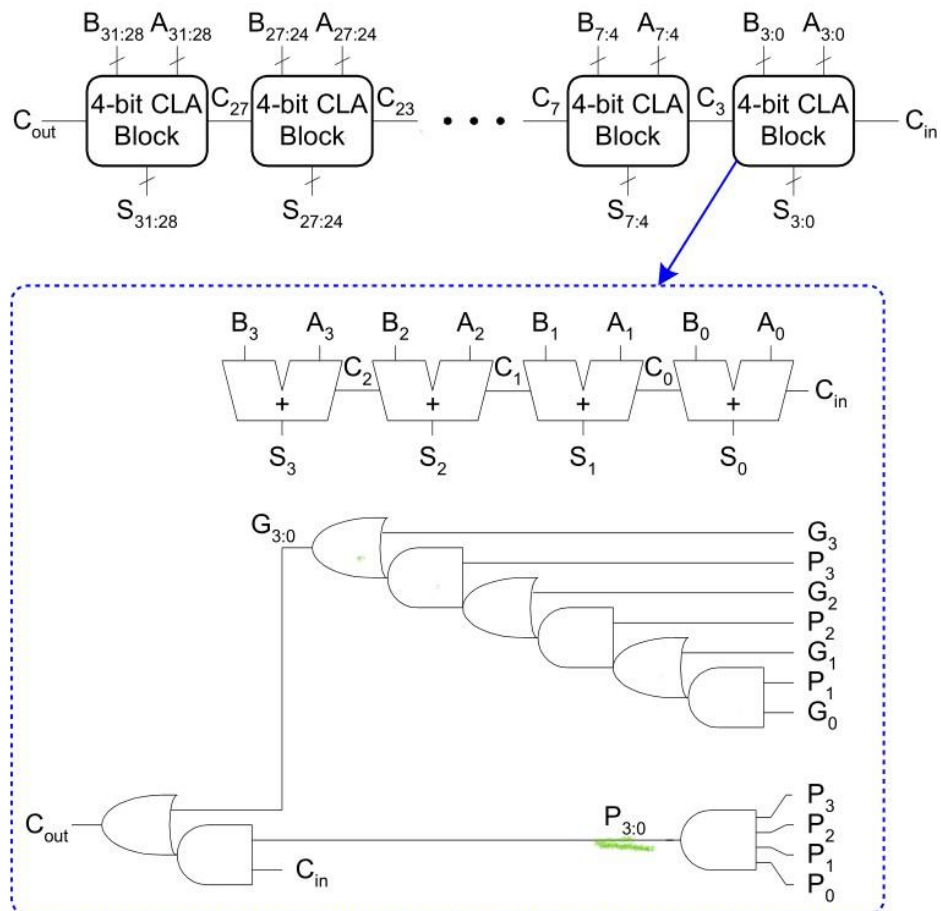
Abbiamo carry out se c'è una configurazione generate o un propagate che si è propagato dal riporto precedente C_{i-1}

$$C_i = G_{i:j} + P_{i:j} C_{j-1}$$

Abbiamo un riporto al blocco se abbiamo configurazioni di Generate o Propagate che propagano il riporto del blocco precedente, oppure configurazioni Generate e Propagate insieme

Com'è fatto un Carry lookahead adder

32-bit CLA with 4-bit Blocks



Step 1: calcola tutti i Gi and Pi

Step 2: calcola G and P per blocchi di k-bit

Step 3: Cin si propaga mediante la logica propagate/generate dei vari blocchi di k-bit

Avremo dei circuiti combinatori che consentono di calcolare in parallelo ognuno dei valori generate e propagate

Si suddividono in questo caso i full adder in blocchi da 4 . Ogni blocco è composto da 4 sommatatori e poi da una logica che consente di trasportare il Cin in ingresso ad un blocco direttamente al Cout del blocco in questione. Poi i riporti si propagano da blocco a blocco I valori di G e P sono calcolati in parallelo su tutti i blocchi

Tutti i blocchi del CLA calcolano i segnali di generazione e di propagazione sia di colonna sia di blocco simultaneamente.

Ritardo del Carry Lookahead Adder

Per un N -bit CLA con blocchi di k bit:

$$t_{CLA} = t_{pg} + t_{pg_block} + (N/k - 1)t_{AND_OR} + kt_{FA}$$

tpg : ritardo per generare P_i , G_i di ogni i -esima colonna

tpg_block: ritardo per calcolare i segnali di generazione e di propagazione $P_{i:j}$ e $G_{i:j}$ per ogni blocco a k bit,

tAND_OR: ritardo delle porte AND/OR a monte della logica propagate/generate, questo ritardo si propaga da C_{in} a C_{out} che si propaga lungo i tutti i blocchi (N num bit totali / $k - 1$ dimensioni blocco)

ktFA: ritardo di un full adder per il numero di bit di cui è costituito un blocco

Per $N > 16$ il sommatore ad anticipazione di riporto è generalmente molto più veloce

Sottrattore

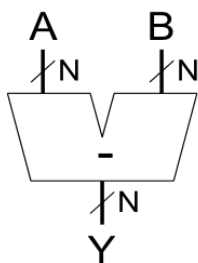
I sommatore sono in grado di sommare numeri sia positivi sia negativi utilizzando la rappresentazione dei numeri in complemento a due. La sottrazione è quindi facile quanto l'addizione: si inverte il segno del secondo numero B e poi si esegue la somma.

Il cambio di segno di un numero in complemento a due si esegue negando tutti i bit e aggiungendo un 1.

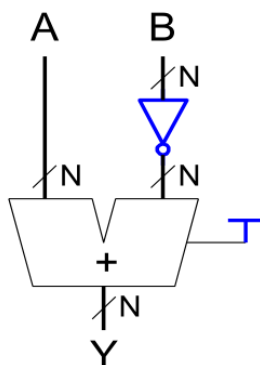
Per calcolare $Y = A - B$, per prima cosa si crea il numero in complemento a due di B : si negano tutti i bit di B per ottenere B^* e si aggiunge 1 per ottenere $-B = B^* + 1$. Questo valore viene aggiunto ad A per ottenere $Y = A + B^* + 1 = A - B$.

Il +1 aggiunto al complemento di B è dato dal C_{in} che è settabile ed è impostato inizialmente a 1 proprio per effettuare il complemento a 2 del numero B .

Symbol



Implementation



Comparatore

Un comparatore determina se due numeri binari sono uguali o se uno dei due è maggiore o minore dell'altro.

Un comparatore riceve due numeri binari a N bit in ingresso, A e B .

Esistono due tipi comuni di comparatori:

comparatori di uguaglianza: produce una singola uscita che indica se A è uguale a B o no ($A == B$) oppure no.

comparatore completo: produce una o più uscite che indicano tutti i valori relativi di A e di B quindi se $A > B$, $A < B$ o $A = B$

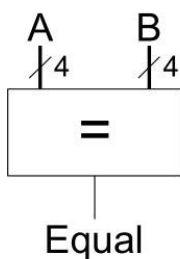
Comparatore di uguaglianza a 4 bit

Confronta 2 ingressi a 4 bit e ritorna 1 se e solo se i due ingressi sono uguali.

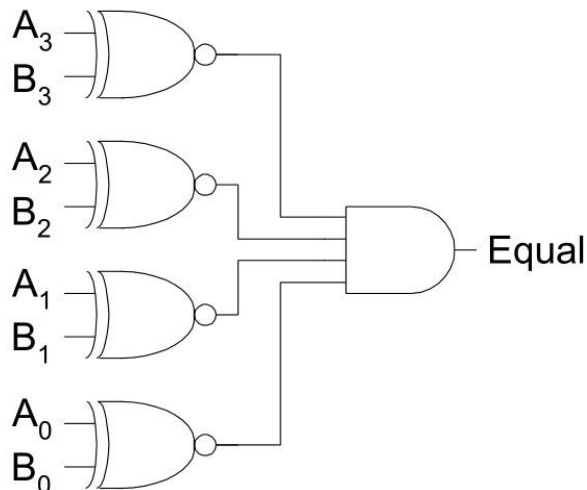
A e B sono uguali quando bit per bit di ogni colonna sono uguali

il comparatore determina se i bit corrispondenti a ogni colonna (bit a bit) sono uguali utilizzando delle porte XNOR. I due numeri sono uguali se tutte le colonne sono uguali, quindi alla fine avremo un AND a 4 ingressi di tutti gli output delle porte XNOR

Symbol

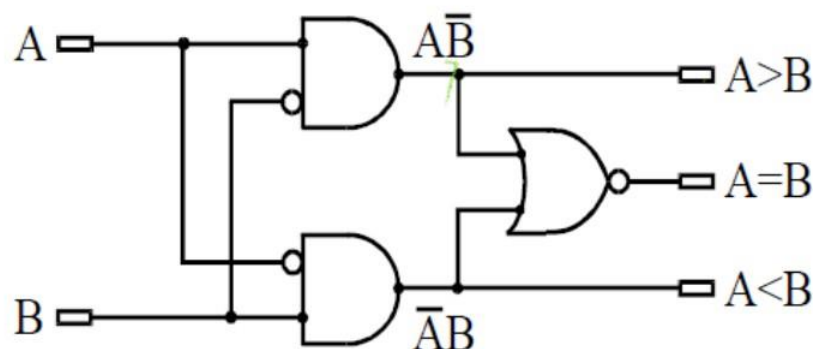


Implementation



Comparatore completo a 2 bit

A	B	$A \wedge B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0



Ha 3 uscite che indicano

- se $A = B$ (ossia $A=0$ e $B=0$ oppure $A=1$ e $B=1$ quindi $AB^* \text{ NOR } A^*B$)

- se $A > B$ (ossia quando $A=1$ AND $B=0$, AB^*)

- se $A < B$ (ossia quando $A=0$ AND $B=1$, A^*B)

Per ogni valore di A e B una sola delle 3 uscite sarà uguale a 1

Un comparatore completo a 4 bit o 12 bit suddivide le parole in blocchi e ottiene i risultati del confronto per ogni blocco. I valori di un blocco di uscita diventano i valori di ingresso del blocco successivo. I valori si propagano dall'inizio alla fine di un blocco a meno che i bit non siano uguali e si passa al blocco successivo

La comparazione di valore dei numeri con segno viene solitamente effettuata calcolando $A - B$ e guardando il segno (cioè il bit più significativo) del risultato dell'operazione.

Se il risultato è negativo (cioè se il bit del segno è uguale a 1) allora A è minore di B. Al contrario, se il risultato è positivo, A è maggiore o uguale a B. Tuttavia, questo comparatore non lavora correttamente in caso di traboccamento (overflow).

ALU

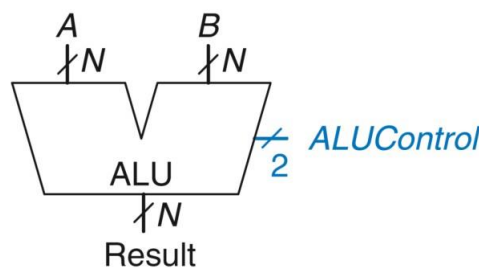
E' l'unità logico aritmetica che esegue i calcoli principali del calcolatore

- Addizione
- Sottrazione
- AND (bit a bit)
- OR (bit a bit)

Una ALU a N bit ha:

- 2 ingressi a N bit A e B
- un output (Result) a N bit
- un segnale di controllo a 2 bit **ALUControl** che specifica quale funzione debba eseguire l'ALU

ALUControl _{1:0}	Function
00	Add
01	Subtract
10	AND
11	OR



Schema circuitale

La ALU contiene un sommatore a N bit per la somma e un numero N di porte AND o OR a due ingressi per le operazioni logiche di AND e OR

Contiene un inverter a N bit e un multiplexer per invertire l'ingresso B quando il segnale di controllo **ALUControl₀ = 1** ossia quando dobbiamo eseguire una sottrazione per complementare il numero

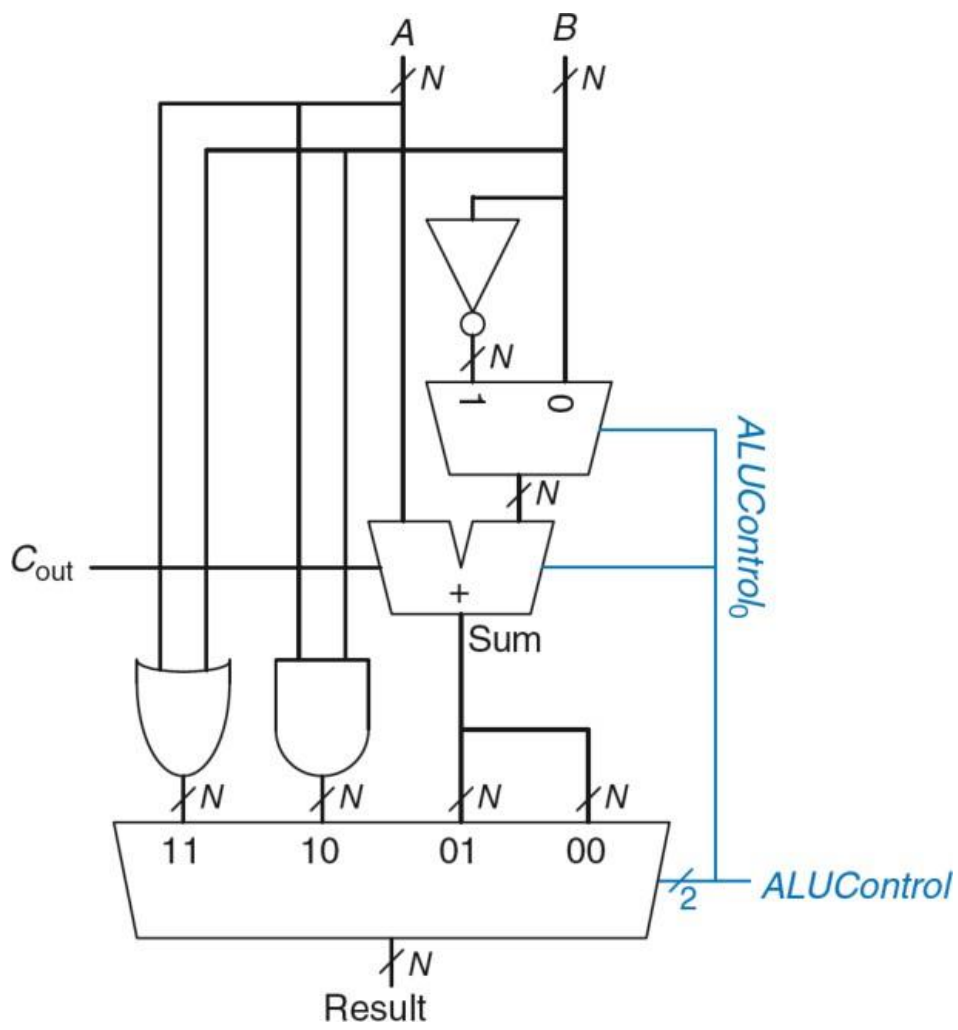
Infine un multiplexer 4:1 sceglie l'operazione desiderata in output sulla base del segnale ALUControl.

se ALUControl = 00, il multiplexer di uscita sceglie **A + B**.

se ALUControl = 01, la ALU calcola $A - B$ (dal momento che ALUControl₀ è uguale a 1, il sommatore riceve gli ingressi A e B* e un riporto di ingresso a 1, il che fa sì che il sommatore esegua la sottrazione col complemento a 2: $A + B^* + 1 = A - B$).

se ALUControl = 10 l'ALU esegue **A AND B**

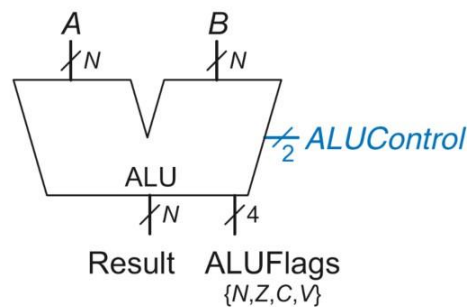
se ALUControl = 11, l'ALU esegue **A OR B**.



ALU con flags di stato

E' un alu che come output ha anche delle uscite chiamate flag, che danno informazioni aggiuntive sul risultato dell'operazione effettuata dall'ALU.

Flag	Description
N	Result is N egative
Z	Result is Z ero
C	Adder produces C arry out
V	Adder o V erflowed



L'uscita ALUFlags è a 4 bit ed è composta dalle flag

N (negative) : indica che il risultato dell'ALU è negativo, ossia il bit più significativo del risultato $\text{Result}_{31}=1$

Z (zero): indica che il risultato dell'ALU è uguale a zero quindi se tutti i bit di Result sono uguali a 0. Si prendono tutti i bit di $\text{result}_{31:0}$, si commutano (porta NOT) e si mettono in AND. La porta AND restituisce 1 se tutti i 32 ingressi commutati sono 1 e quindi sono tutti 0.

C (carry): indica che il sommatore ha prodotto un riporto CarryOut

La flag $C=1$ quando il sommatore produce un riporto e l'ALU sta eseguendo una somma o una sottrazione ($\text{ALUControl}_1 = 0$). ALUControl_1 viene commutato (quindi se è 0 diventa 1 e messo in AND con il $\text{Cout} = 1$) se abbiamo 1 dalla porta AND allora il Carry flag si attiva

V (overflow): indica che il sommatore ha prodotto un Overflow. esso si verifica quando la somma di due numeri di egual segno produce come risultato un numero di segno opposto.

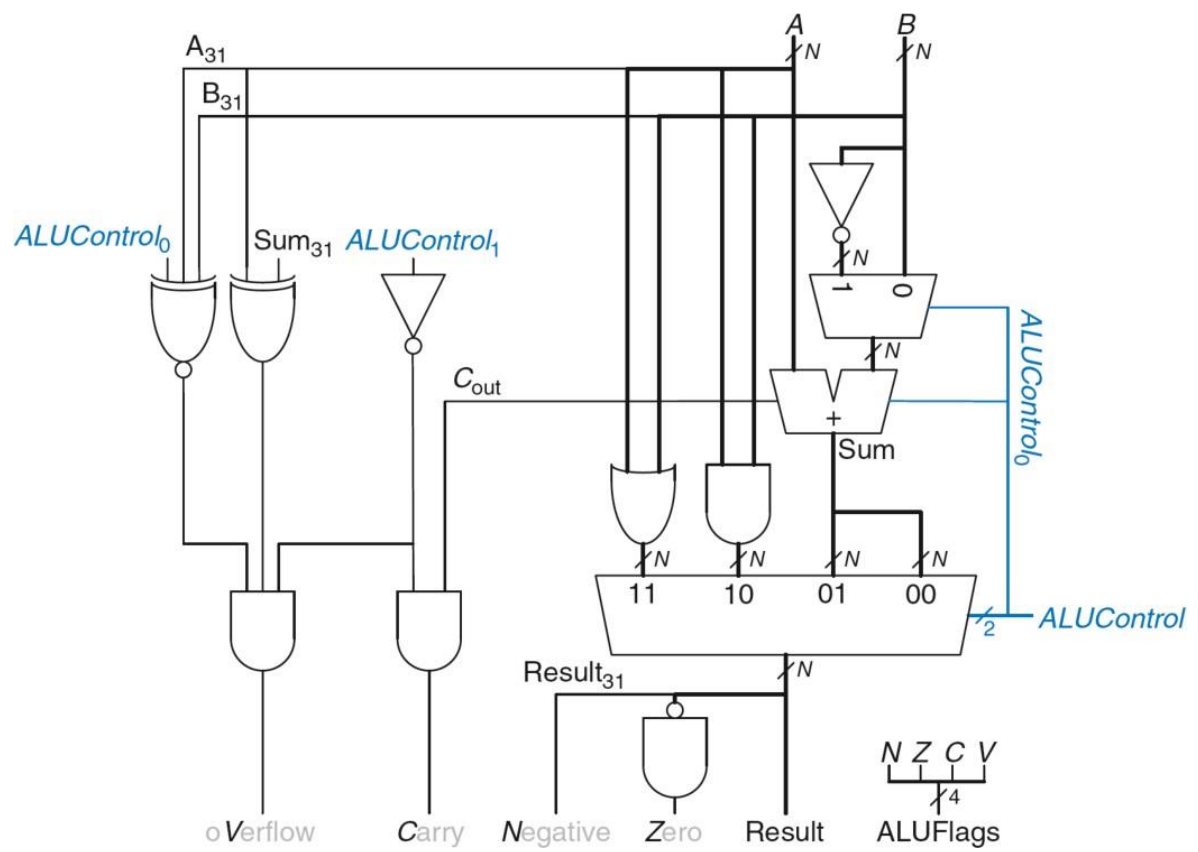
V è attiva nel caso in cui si verificano le 3 seguenti condizioni:

(1) la ALU sta eseguendo una **somma o una sottrazione ($\text{ALUControl}_1 = 0$ commutato 1)**

(2) il bit A₃₁ e quello della somma Sum₃₁ hanno segni opposti, come identificato dalla porta XOR

(3) come identificato dalla porta XNOR i bit A₃₁ e B₃₁ sono uguali, hanno lo stesso segno e il sommatore sta seguendo un'addizione ($\text{ControlloALU}_0 = 0$), oppure A₃₁ e B₃₁ hanno segno opposto e il sommatore sta eseguendo una sottrazione ($\text{ControlloALU}_0 = 1$).

La porta AND a tre ingressi riconosce quando tutte e tre le condizioni si avverano e quindi attiva V



Shifters

Sono circuiti che traslano o ruotano i bit ed eseguono la moltiplicazione o la divisione per potenze di 2.

Logical shifter: trasla i bit a sinistra a (LSL, Logical Shift Left) o a destra (LSR, Logical Shift Right) e riempie gli spazi vuoti con degli 0

Arithmetic shifter: come il logical shifter a sinistra, nella traslazione a destra (ARS, Arithmetic Shift Right) invece riempie gli spazi vuoti con il bit più significativo (msb)

Rotator: ruota i bits circolarmente verso sinistra (ROL, Rotate Left) o destra (ROR, Rotate Right)

i bits che "escono" da un lato rientrano dall' "altro"

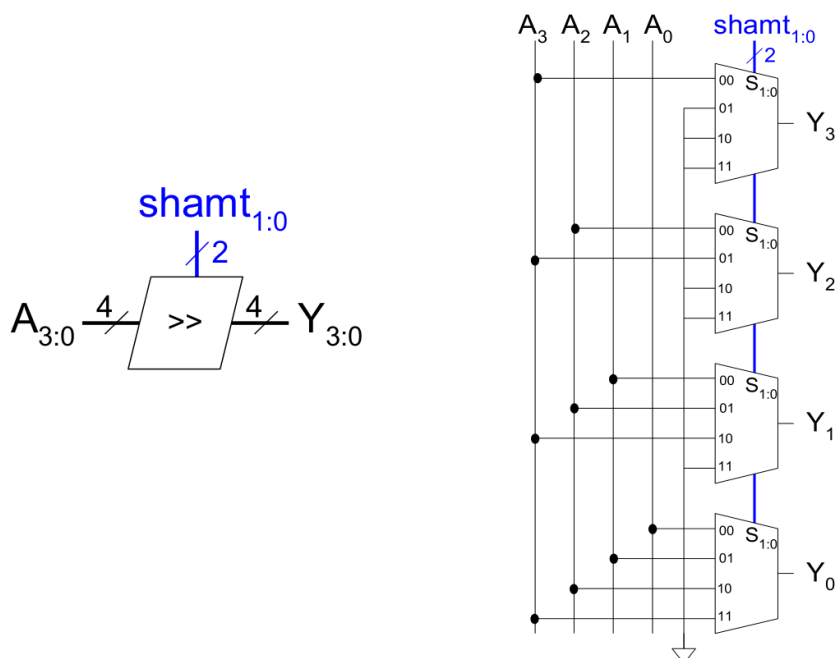
Shifter Design

Uno shifter a N bit può essere costruito con un numero N di multiplexer N:1.

L'ingresso viene traslato da 0 a N – 1 posizioni, a seconda dei valori presenti sulle $\log_2 N$ linee di selezione

Es. Logical Shift Right

In output Y abbiamo la parola di 4 bit shiftata verso dx di una quantità che viene indicata dai due selettori che prendono il nome di $shamt_{1:0}$ (shift amount) ed indicano la quantità di cui deve shiftare la parola. Essendo la parola in ingresso A di 4 bit è possibile shiftare fino a 4 posizioni, quindi avremo bisogno di $\log_2 4 = 2$ selettori per lo shamt



Shifters as Multipliers, Dividers

Gli arithmetic shifter sono detti aritmetici perchè le operazioni di asr e asl corrispondono alla moltiplicazione di un numero per 2^N dove N è la quantità della traslazione (shift a sx lsl o asl)

oppure alla divisione di un numero per 2^N dove N è la quantità della traslazione (shift a dx asr)

- $A \lll N = A \times 2^N$

- **Example:** $00001 \ll 2 = 00100 \quad (1 \times 2^2 = 4)$

- **Example:** $11101 \ll 2 = 10100 \quad (-3 \times 2^2 = -12)$

- $A \ggg N = A \div 2^N$

- **Example:** $01000 \ggg 2 = 00010 \quad (8 \div 2^2 = 2)$

- **Example:** $10000 \ggg 2 = 11100 \quad (-16 \div 2^2 = -4)$

Counters

Il contatore è un dispositivo che conta , quindi incrementa il suo valore di output di 1 ad ogni ciclo di clock sul fronte alto. Viene utilizzato in particolare dal program counter per tenere traccia dell'istruzione corrente da eseguire o come orologio digitale.

Un contatore binario a N bit, è circuito sequenziale aritmetico

Ha come ingresso il segnale di clock che pilota il contatore

Il segnale RESET permette di resettare l'output, inizialmente è resettato a 0

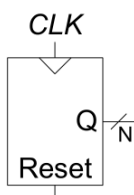
In output abbiamo N uscite, Q che rappresenta di volta in volta il numero corrente

C'è un flip flop D resettabile che tiene memoria del valore corrente del contatore, è resettabile quindi è possibile resettare il valore del contatore.

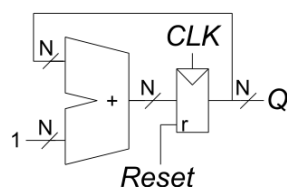
Il valore corrente in output Q viene retroazionato in un sommatore che prende in input come argomenti il valore corrente dell'output Q e il valore 1. Quindi il sommatore di volta in volta somma al valore corrente 1 . **Per ogni clock $Q = Q_{prev} + 1$**

Genera 2^N valori

Symbol



Implementation



Shift Registers

Lo shift register esegue una traslazione di un **ingresso seriale Sin** di 1 bit per ogni battito del clock.

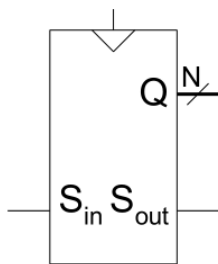
Ha un **ingresso seriale Sin** cioè 1 alla volta, un'uscita seriale **Sout**, e **N** uscite parallele **QN-1:0**

A ogni fronte di salita del clock, un nuovo bit viene inserito dall'ingresso Sin e tutti i bit seguenti vengono traslati in avanti. L'ultimo bit nel registro diventa quindi disponibile all'uscita Sout.

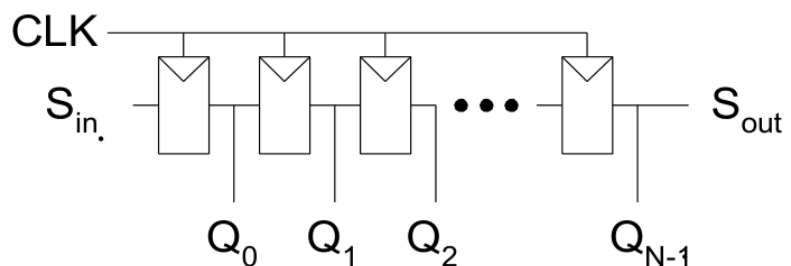
E' implementato da una serie di flip flop d cioè registri che memorizzano 1 bit in sequenza tra loro. Alla fine di ogni flip flop abbiamo l'uscita Q0, Q1, Qn-1 alla fine tutte le uscite compongono l'uscita finale Sout. Tutti i flip flop sono pilotati dallo stesso clock

Viene detto **convertitore seriale parallelo** : l'ingresso viene infatti ricevuto in serie (un bit alla volta) da Sin. Dopo N cicli, gli N ingressi ricevuti sono disponibili in parallelo su Q. Quindi in parallelo si legge quella che è la storia di Sin negli scorsi n clock, a partire dal meno recente (N clock fa) fino all'ultimo valore che è il più recente dell'ultimo clock

Symbol:



Implementation:



Q0 = 1

Q1 = 0

Q2 = 1

...

QN-1 = 1

Q = 1011

Sout mostra l'output uno alla volta

Shift Register con load parallelo

E' una versione più complessa dello shift register.

Esegue sia la conversione seriale-parallelo sia quella parallelo-seriale, aggiungendo un ingresso parallelo D a N bit e un segnale di controllo LOAD che pilota un mux di scelta

Quando LOAD=1 -> parallelo seriale

i flip-flop vengono caricati in parallelo a partire dall'ingresso D0 fino a Dn-1. funziona come un usuale registro a N-bit, leggendo poi le uscite in serie una alla volta su Sout

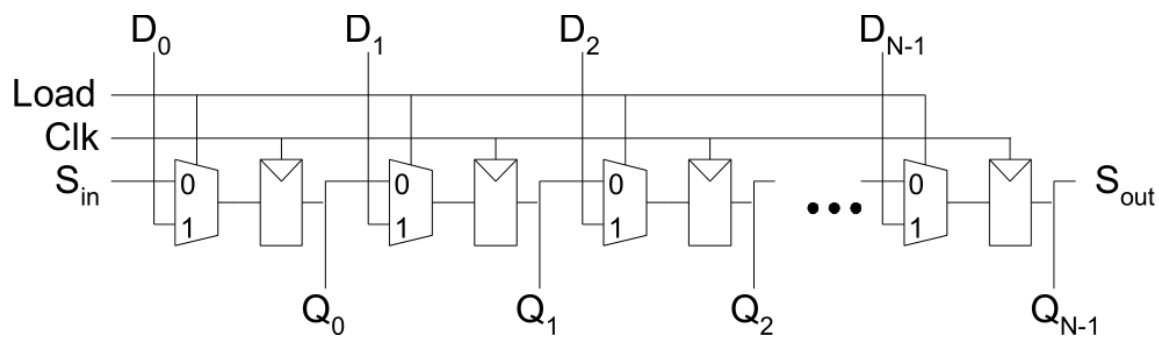
Sin disabilitato grazie al mux

Quando LOAD=0 -> seriale parallelo

il registro effettua la normale traslazione seriale-parallelo

l'input viene letto un bit alla volta e poi alla fine le uscite vengono mostrate in parallelo

Q0:N-1



Memory arrays

Lo scopo delle memorie è quello di memorizzare efficacemente una grossa quantità di dati
Esistono 3 tipologie di memorie che immagazzinano grandi quantità di dati:

- **Dynamic random access memory (DRAM)**
- **Static random access memory (SRAM)**
- **Read only memory (ROM)**

Ognuna di queste memorie è differente nella modalità di immagazzinamento dei dati

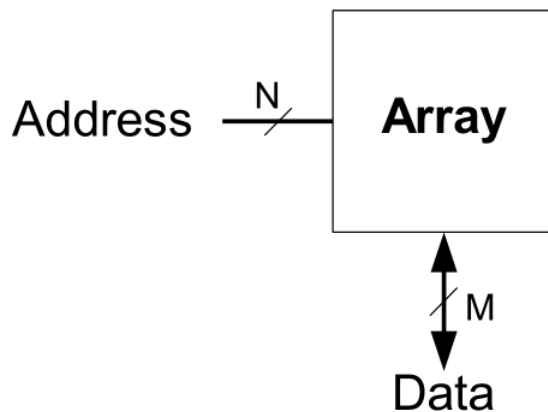
La memoria è organizzata come array bidimensionale di celle di memoria

In ingresso abbiamo n bit che identificano un certo **indirizzo** di una locazione di memoria nella quale si può leggere o scrivere il contenuto di una riga dell array bidimensionale di memoria detta parola (tipicamente di 8 bit=1 byte).

Il valore letto o scritto nella memoria viene chiamato data ed è una parola di m bit.

Tipicamente $m = 8$ ossia 1 byte.

N (indirizzo) = 32

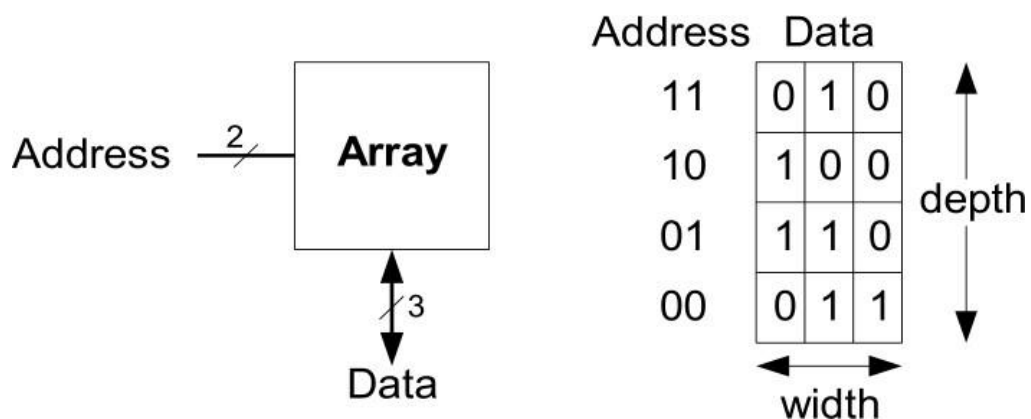


Ogni cella memorizza un bit

Le memorie sono byte addressable perchè ogni byte ha un suo specifico indirizzo

Con N bit di indirizzo e M bits data: 2^N righe e M colonne

- **Depth:** numero di righe (ossia numero di parole)
- **Width:** numero di colonne (lunghezza di una parola)
- **Dimensione Array :** depth \times width = $2^N \times M$



- $2^2 \times 3$ -bit array
- Numero parole: 4
- Lunghezza parole: 3-bits
- All'indirizzo 10 corrisponde la parola 100

Bitline e Wordline

Le memorie vengono realizzate come matrici, ogni elemento della matrice è una cella di memoria che può contenere un bit di dato.

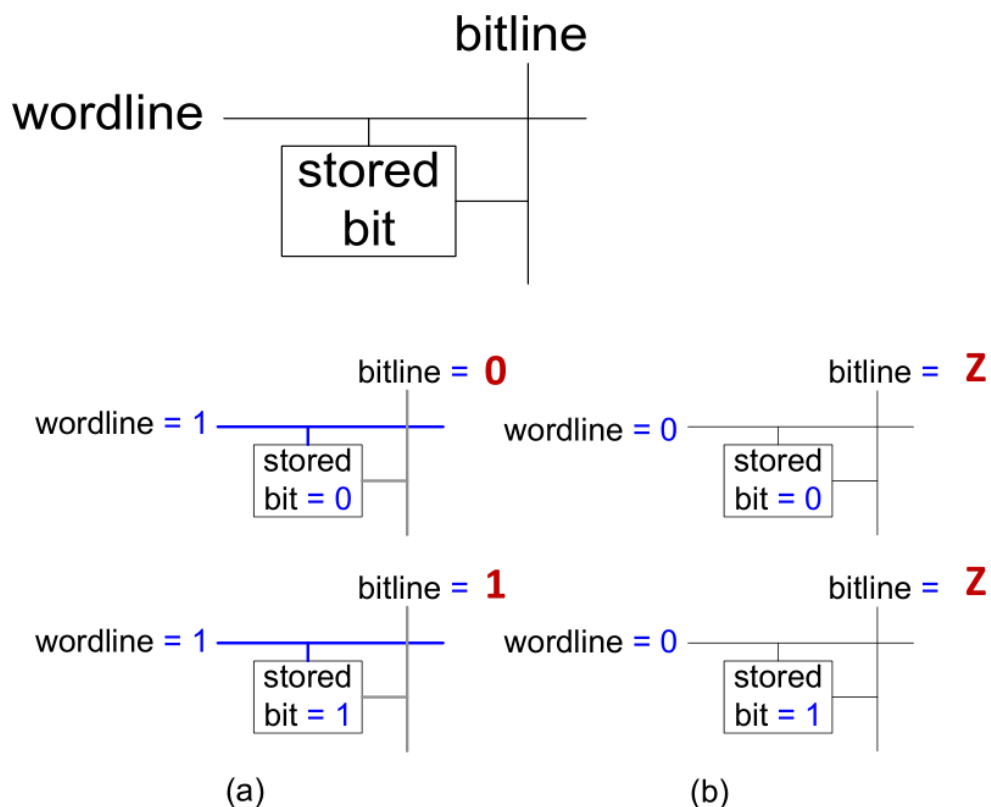
La cella di memoria è un elemento circuitale realizzato in modo diverso a seconda delle memorie utilizzate. Ogni cella di memoria è connessa ad una bitline (il cui insieme costituisce i valori di "data") ed è collegata ad una wordline che consente l'attivazione di una data parola. Ad ogni parola corrisponde una wordline

Per ogni configurazione dei bit di indirizzo, la memoria attiva una wordline che, a sua volta, attiva le bitline presenti nella riga corrispondente.

Quando la wordline = 1, il bit memorizzato viene inviato alla bitline o prelevato dalla stessa. Altrimenti, la bitline è disconnessa dalla cella in cui è memorizzato il bit.

Quando la wordline=1 e il bit memorizzato è 0 nella cella, leggeremo sulla bitline 0, altrimenti se il bitline memorizzato è 1 leggeremo 1 sulla bitline

Quando la wordline=0 la wordline non è attivata quindi è come se il bit fosse scollegato dalla sua bitline che si trova relativamente a questo bit in uno stato di alta impedenza

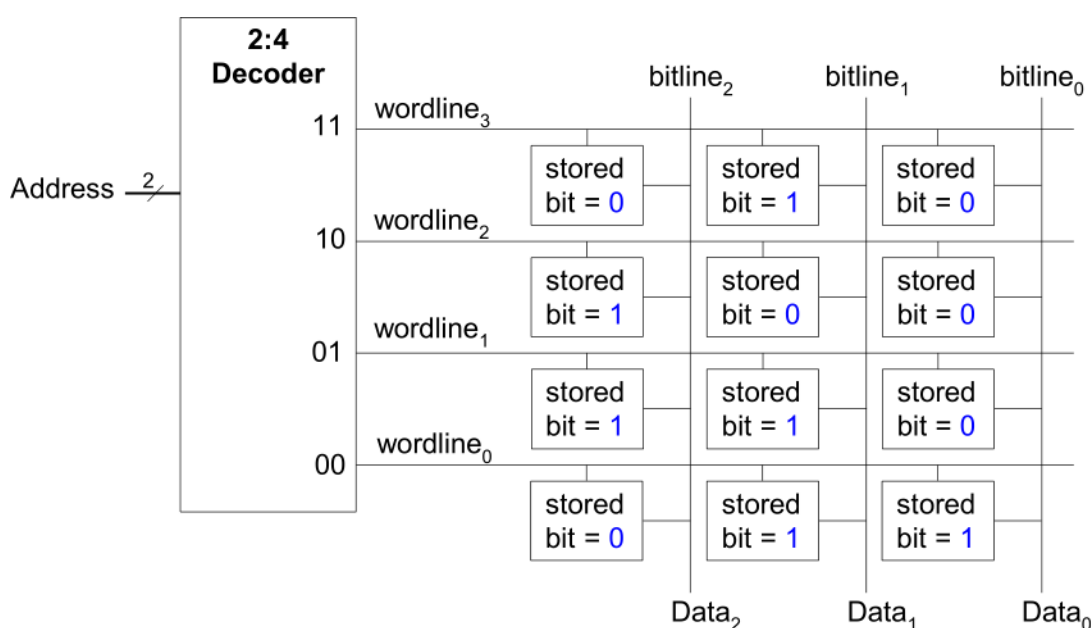


Organizzazione della memoria

Un array di memoria dal punto di vista circuitale è formato da una serie di N ingressi relativi all'indirizzo, poi vi è un decoder che ad ogni valore dell'indirizzo in input attiva la wordline corrispondente, che ci consente di leggere/scrivere alla wordline selezionata con le relative bitline. Le altre wordline sono scollegate temporaneamente

Wordline:

- Agisce come un enable
- Seleziona una riga nella memoria
- Corrisponde ad un unico indirizzo
- Solo una wordline per volta è attivata



Tipi di memoria

- Random access memory (RAM): volatile
- Read only memory (ROM): non volatile

RAM

La RAM è una memoria volatile nel senso che i dati sono persi allo spegnimento del computer. Quindi perderemo i valori delle parole all'interno della memoria.

Caratteristica della RAM è che le operazioni di lettura e scrittura sono più veloci rispetto alle ROM. Di fatto la memoria RAM è la memoria principale di un computer.

Per eseguire un programma si carica dalla memoria di massa il programma nella memoria ram, insieme ai dati relativi all'esecuzione e poi il programma viene eseguito

Viene chiamata random access memory perchè il tempo per accedere ad una parola ad un certo indirizzo è uguale al tempo di accesso di altre parole locate ad indirizzi diversi. Il tempo di accesso ad ogni singola parola è costante indipendentemente dall'indirizzo

Le operazioni di lettura e scrittura sono più veloci (rispetto alle ROM)

ROM

Per ROM si intende una memoria non volatile, cioè una memoria che mantiene i dati contenuti al suo interno anche quando il pc è spento.

Inizialmente venivano chiamate di sola lettura perchè scritte attraverso dei fusibili cioè elementi circuitali che una volta bruciati non era più possibile modificare i contenuti della memoria

La lettura è relativamente veloce (meno della ram), la scrittura meno

Tipi di RAM: DRAM e SRAM

Le ram le dividiamo in 2 grandi categorie che si differenziano per le componenti usate per realizzare le singole celle di memoria e quindi memorizzare i dati:

- **DRAM (Dynamic random access memory) usano dei CONDENSATORI/CAPACITÀ**
- **SRAM (Static random access memory) usano INVERTITORI**

DRAM

I bit di una cella di memoria sono memorizzati in delle capacità/condensatori

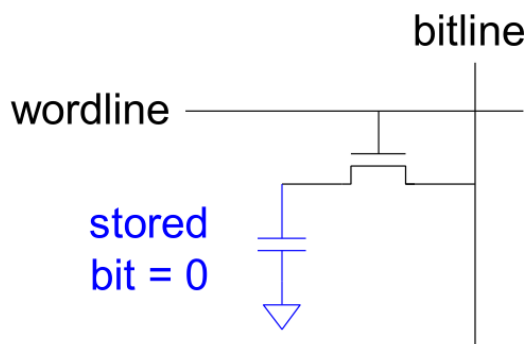
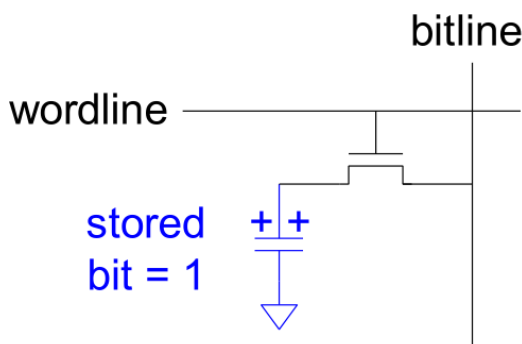
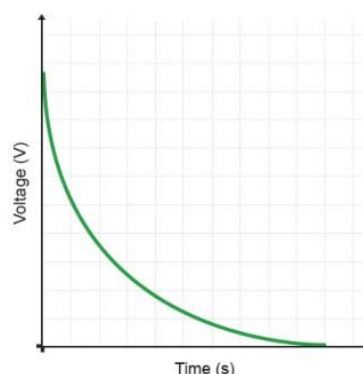
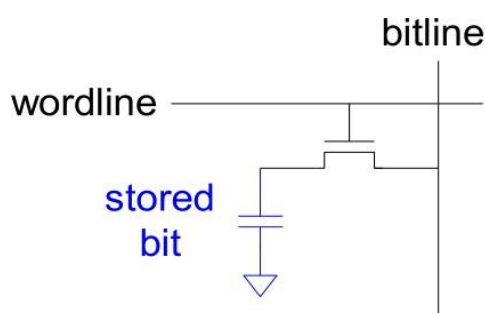
I bit sono quindi memorizzati come presenza o assenza di carica nel condensatore

Il valore del bit viene memorizzato in un condensatore.

C'è un transistor che si comporta come un interruttore che connette o disconnette il condensatore dalla bitline. Quando wordline è attiva, il transistor si accende e il valore del bit immagazzinato viene trasferito alla o dalla bitline.

quando il condensatore viene caricato a VDD, il bit immagazzinato è 1; quando invece viene scaricato fino a GND, il bit immagazzinato è 0.

Dynamic perché il valore di un bit deve essere “refreshed” (riscritto) periodicamente e anche dopo che è stato letto: ciò è dovuto al fatto che la perdita di carica di una capacità degrada il valore memorizzato e che la lettura distrugge il valore letto



Un condensatore possiamo paragonarlo ad una vasca cioè un elemento che ha la capacità di immagazzinare le cariche elettriche. La carica elettrica è il fluido all'interno della vasca. Il livello del fluido corrisponde al potenziale (alto o basso). Quando abbiamo un valore di

potenziale alto VDD è perchè vi è una differenza di potenziale che genera corrente quindi un movimento di elettroni e cariche elettriche che riempiono il condensatore "la vasca" fino ad un certo livello di potenziale VDD.

Quando il livello è uguale a VDD non vi è più flusso di elettroni e la corrente cessa.

Il condensatore si trova in uno stato logico = 1.

Se abbiamo uno stato logico = 0 non vi è alcun flusso di corrente e il condensatore rimane scarico.

I condensatori hanno delle “perdite” nel senso che il valore del potenziale tende a degradarsi (ossia ritornare a 0) e non si mantiene costante a VDD . Quindi se voglio mantenere un bit pari a 1 devo di tanto in tanto refreshare la memoria , cioè ribadirei il valore della cella di memoria in particolare per i bit che hanno un valore di memoria alto.

Altrimenti dopo pochi ms tutti i bit tornerebbero a 0.

Transistor

L’operazione di collegare/scollegare la stored bit (cella) alla bitline dipende da una componente circuitale detta transistor cmos

è un interruttore che quando la wordline è uguale a 1 è chiuso e quindi il condensatore è collegato alla rispettiva bitline

quando il valore della wordline è 0 il transistor corrisponde ad un interruttore aperto e quindi il condensatore è scollegato dalla bitline

Serve allora per controllare l'accesso alla cella

SRAM

Una RAM statica (SRAM, Static RAM) viene chiamata in questo modo perché i bit immagazzinati nella memoria non hanno bisogno di essere refreshati

Viene realizzata attraverso un circuito bistabile costituito da due porte NOT invertite

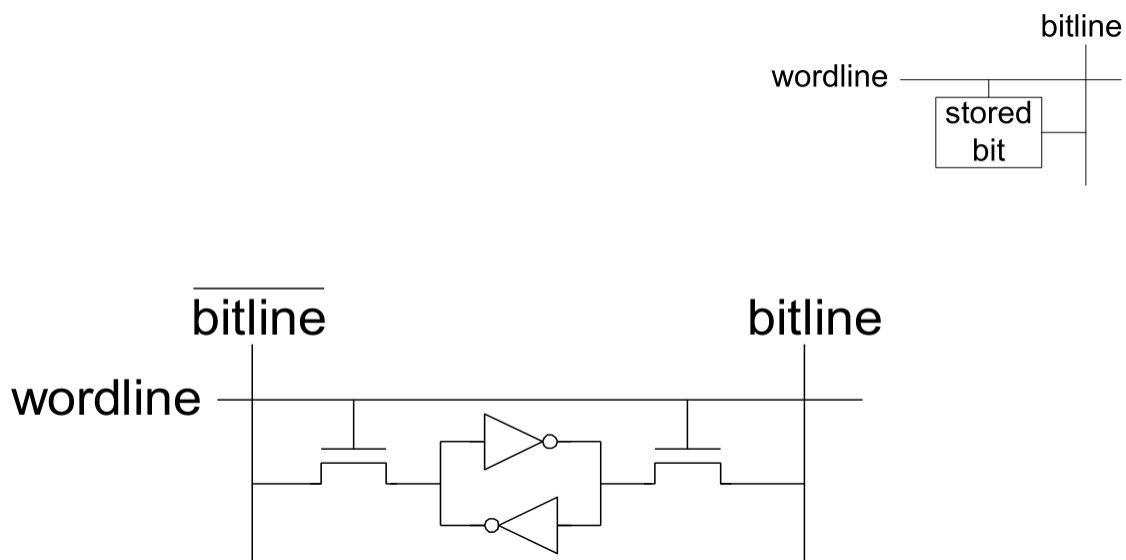
La bitline viene sdoppiata in due linee bitline e *bitline che hanno sempre valori complementari

Quando bitline=1 , *bitline=0 e viceversa

Queste due linee sono collegate alla memoria attraverso 2 transistor che attivano o disattivano la cella di memoria

Quando la wordline viene attivata, entrambi i transistor si accendono e i bit di dato vengono trasferiti da o verso le linee di bit.

Gli invertitori ribadiscono il valore del potenziale in uno dei due punti è come se il dato fosse intrappolato



DRAM vs SRAM confronto

Le DRAM sono più lente perchè il condensatore ci mette un po' di tempo per riempirsi fino ad un certo livello logico o per scaricarsi per passare da un livello logico 1 a 0. Questi tempi di latenza di carica e scarica rendono questo tipo di memorie più lente rispetto alla SRAM realizzata con gli invertitori che sono più veloci e reattivi e quindi le sram sono più reattive e veloci

Le dram devono essere refreshate e riscritte e quindi bisogna fare un refresh ogni pochi ms ciò induce un maggior consumo di energia ma esse sono più economiche delle SRAM perchè necessitano di un solo transistor per costruirle mentre le sram 6 transistor

I flip flop hanno una 20ina di transistor e sono ancora più costosi delle dram e sram. Sono i più veloci come tempi di risposta e latenza poi abbiamo le SRAM e le DRAM che sono le più lente I flip flop sono utilizzati per costruire i registri che usa la CPU

La SRAM viene utilizzata per costruire la memoria cache


La DRAM viene utilizzata per costruire la memoria RAM del PC la main memory

Le DRAM si dicono DDR (double data rate) Synchronous RAM perchè le operazioni di scrittura e lettura sono sincronizzate da un clock e avvengono sia sul fronte alto che sul fronte basso del clock. La Data bandwidth è raddoppiata rispetto alla frequenza di clock = double pumping

DRAM vs SRAM

- DRAM è più lenta
- Scrittura e refresh (millisecondi) inducono un consumo maggiore di energia
- DRAM, più economiche

Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow



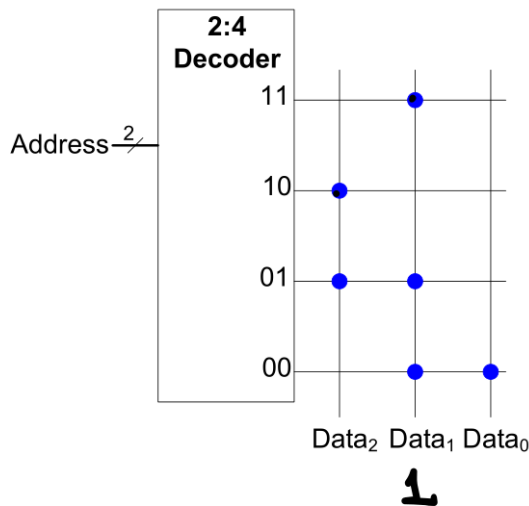
DDR SDRAM

- (DDR) Double data rate (S) synchronous (DRAM) dynamic random access memory
- Le operazioni di lettura e scrittura sono sincronizzate da un clock
- Le trasmissioni avvengono sia nel rising-edge del clock (0→1) che nel che falling-edge (1→0)
- In questo modo il data-bandwidth doppio rispetto alla frequenza di clock (double pumping)

ROM

Il contenuto di una ROM può essere indicato con la notazione a punti.. Un punto posto all'intersezione tra una riga (linea di parola) e una colonna (linea di bit) indica che il bit di dato è uguale a 1. Per esempio, la prima linea di parola ha un unico punto su Dato1, quindi la parola memorizzata all'Indirizzo 11 è 010.

Concettualmente, le ROM possono essere costruite utilizzando la logica a due livelli, con una serie di porte AND seguite da una serie di porte OR. Le porte AND producono tutti i mintermini possibili, quindi vanno a formare un decoder. Ogni punto nella corrisponde a una linea di ingresso a una porta OR. Per linee di bit con un solo punto, in questo caso Dato0, non è necessaria una porta OR. Questa rappresentazione di una ROM è particolarmente interessante perché mostra come la ROM sia in grado di realizzare una qualsiasi funzione logica a due livelli.



Address	Data			
11	0	1	0	<div style="display: flex; align-items: center;"> ↑ depth </div>
10	1	0	0	
01	1	1	0	
00	0	1	1	
				<div style="display: flex; align-items: center;"> ← width </div>

- $X = AB$
- $Y = A + B$
- $Z = A \overline{B}$

