# Architettura degli elaboratori - lezione 9

### Appunti di Davide Vella 2024/2025

Claudio Schifanella
claudio.schifanella@unito.it
Link al moodle :
https://informatica.i-learn.unito.it/course/view.php?id=3106

**25/03/2025**

# Contenuti



## Linking e loading

- Programmi : insieme di procedure (moduli) tradotti separatamente dall'assemblatore (o compilatore). Ogni modulo oggetto ha il suo spazio di indirizzamento separato.

## Linker

È un programma che esegue la funzione di collegamento dei moduli oggetti in modo da formare un unico modulo eseguibile relativo alla macchina specificata. Cosa vuol dire? Su un computer che ha un processore Apple, è possibile creare un eseguibile per RISC-V, tuttavia non è possibile eseguirlo.

## Compiti del linker

Il linker fonde gli spazi di indirizzamento dei moduli oggetto in uno spazio lineare unico nel modo seguente :

- Costruisce una tabella di tutti i moduli oggetto e le loro lunghezze
- Assegna un indirizzo di inizio ad ogni modulo oggetto
- Trova tutte le istruzioni che accedono alla memoria e aggiunge a ciascun indirizzo una relocation constant corrispondente all'indirizzo di partenza del suo modulo
- Trova tutte le istruzioni che fanno riferimento ad altri moduli e le aggiorna con l'indirizzo corretto.
  Se ho un programma che usa solo funzioni interne allo stesso programma (tutte definite dentro ed unico file), quello che viene fatto è semplicemente una chiamata a funzione dove l'assemblatore va a specificare lo spostamento per poter saltare alla funzione nella fase di JAL (jump and link). Invece ci sono altri casi, ovvero quando usiamo qualcosa di una libreria esterna al nostro codice, basta pensare a quando usiamo una printf oppure una variabile globale (ad esempio PI).

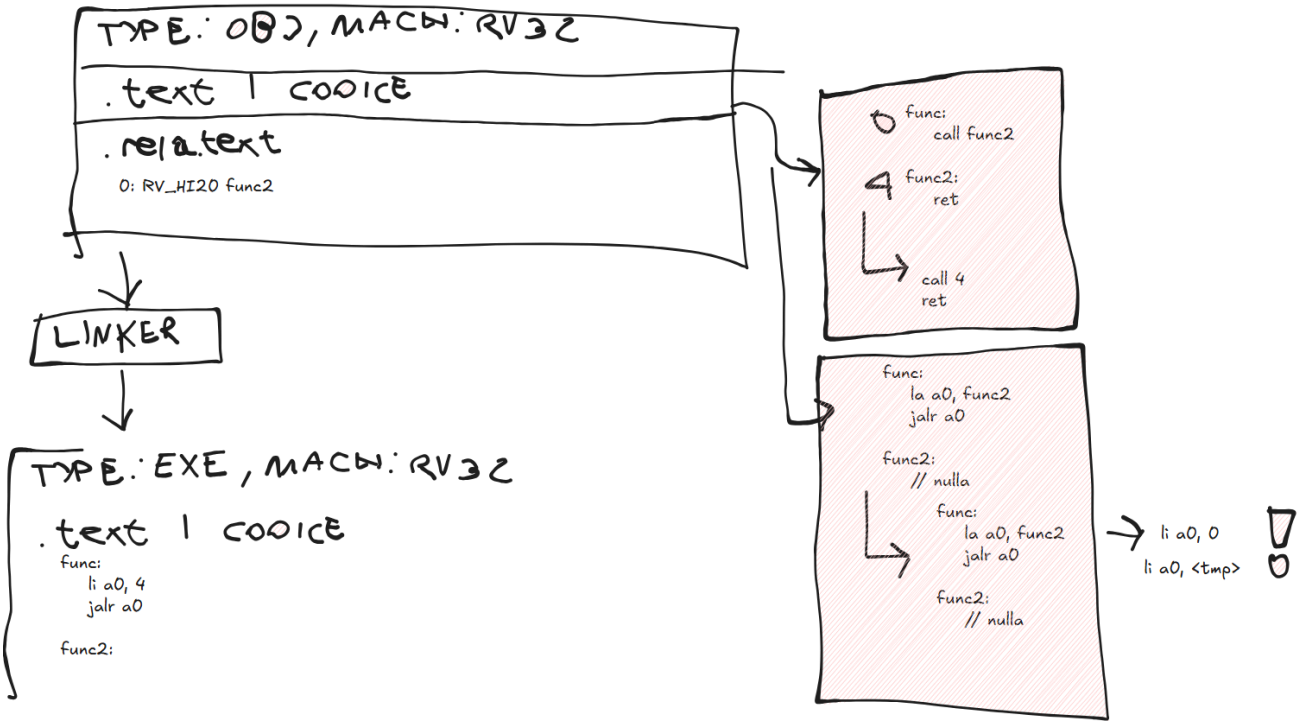
# Veloce riassunto

Abbiamo il compilatore. Questo prende in input un file sorgente C e produce un file sorgente assembly. Preso questo file assembly, lo si passa per l'assembler il quale produce un file oggetto. Questo file oggetto viene poi passato al linker

## Composizione file oggetto

- header (es. "type : obj, macchina : rv32")

- sezioni/segmenti (la sezione .text ad esempio dove abbiamo il codice)




# Loader

- Una volta creato l'eseguibile (ad opera del linker) esso viene memorizzato su un supporto di memoria secondaria
- Al momento dell'esecuzione il sistema operativo lo carica in memoria centrale e ne avvia l'esecuzione
- Il loader (che è un programma del sistema operativo) si occupa di:
  1. Leggere l'intestazione per determinare la dimensione del programma e dei dati
  2. Riservare uno spazio in memoria sufficiente per contenerli
  3. Copiare programma e dati nello spazio riservato
  4. Copiare nello stack i parametri (se presenti) passati al main
  5. Inizializzare tutti i registri e lo stack pointer
     (ma anche gli altri del modello di memoria)
  6. Saltare ad una procedura che copia i parametri dallo stack ai registri e che poi invoca il main

# Binding e allocazione dinamica

Se si ***spostano*** in memoria programmi per cui è già stato fatto il collegamento e il calcolo degli indirizzi, tutti gli ***indirizzi*** di memoria risultano sbagliati e le informazioni di rilocazioni sono state scartate da tempo. Il problema di spostare in memoria programmi è connesso con la scelta del momento in cui effettuare il collegamento (**binding**) tra nomi simbolici e indirizzi fisici (**binding time**).

Quando fare il collegamento? Ci sono alcune scelte possibili:

- Al momento della ***scrittura*** del programma
- Al momento della ***traduzione*** del programma
- Al momento del ***linking*** (ma prima del loading)
- Al momento del ***loading***
- Al momento dell'***esecuzione*** (uso di un registro di base)

**Collegamento statico:**

- **Le funzioni di libreria diventano parte del codice eseguibile**
- Se viene rilasciata una nuova versione, un programma che carica staticamente le librerie continua a utilizzare la vecchia versione
- La libreria può essere molto più grande del programma; i file binari diventano eccessivamente grossi

**Collegamento dinamico:**

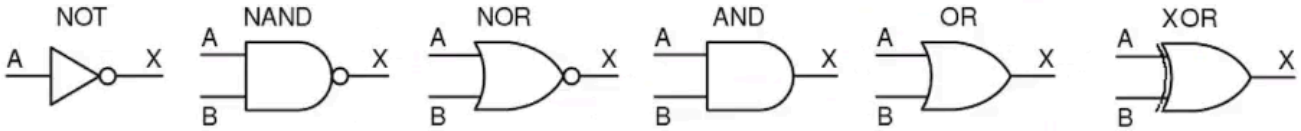
- DLL, Dynamically Linked Libraries
- Le funzioni di libreria non vengono collegate e caricate finché non si inizia l'esecuzione del programma
- DLL con collegamento lazy: ogni procedura viene caricata solo dopo la sua prima chiamata

# Circuiti Digitali

I circuiti digitali sono gli elementi di base con cui si sono costruiti i calcolatori. Sono dispositivi che utilizzano solo due valori logici 0 e 1 (oppure rispettivamente un segnale tra 0 e 1 volt e uno tra 2 e 5. Questi valori non sono universali, possono essere anche diversi).
Un circuito digitale trasforma segnali (binari) di ingresso $x_1, x_2, x_3, \ldots, x_n$ nei segnali (binari) di uscita $z_1, z_2, z_3, \ldots, z_n$.

# Porte logiche

Sono circuiti costituiti da transistor, etc. Questo è il livello più "basso" che vedremo.




| A | X |
|---|---|
| 0 | 1 |
| 1 | 0 |

| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

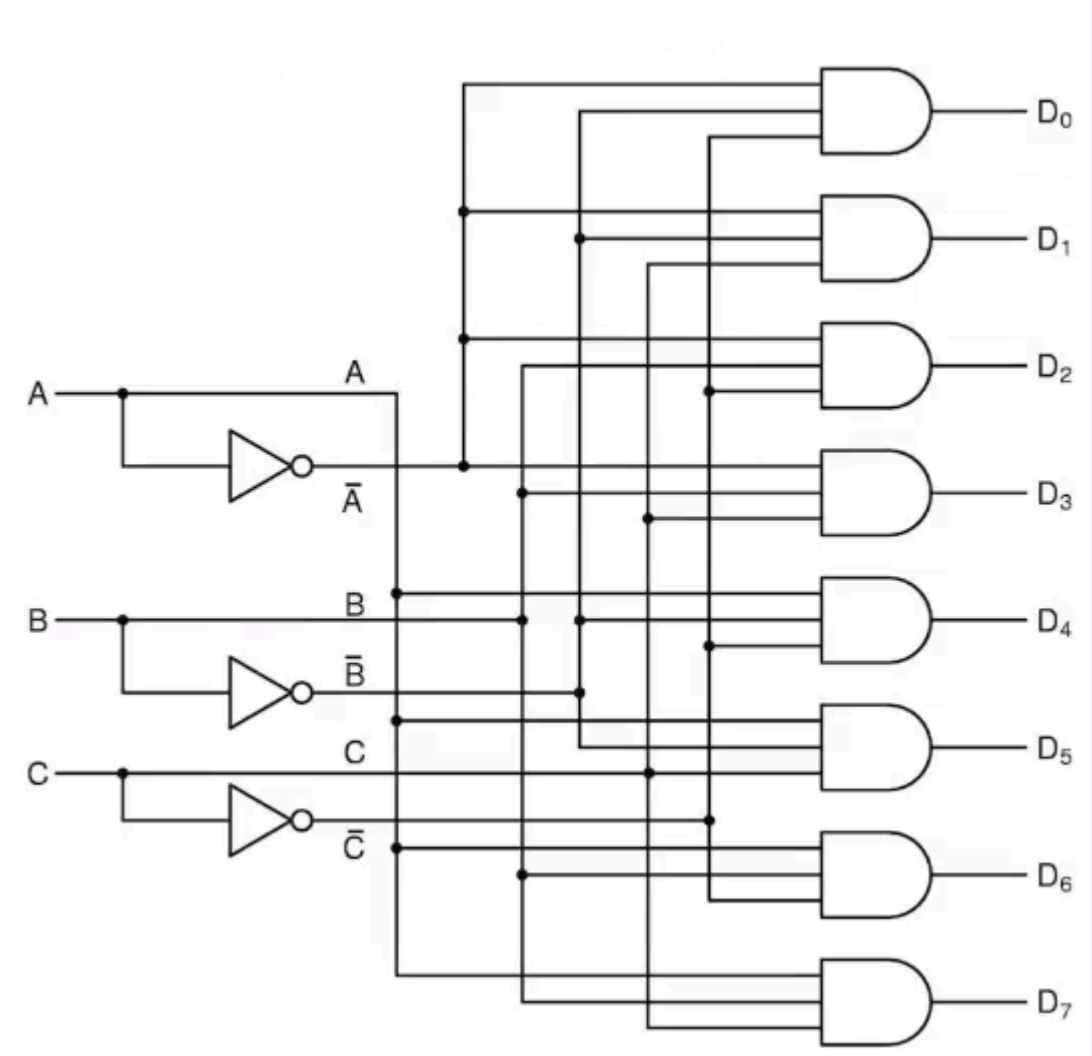
| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Circuiti combinatori

Nei circuiti combinatori l'output viene determinato solo dagli input. Per convenzione, l'incrocio tra due linee non implica alcuna connessione a meno che non sia presente il simbolo • nel punto di intersezione.

**Example**

Esempio di circuito combinatorio :



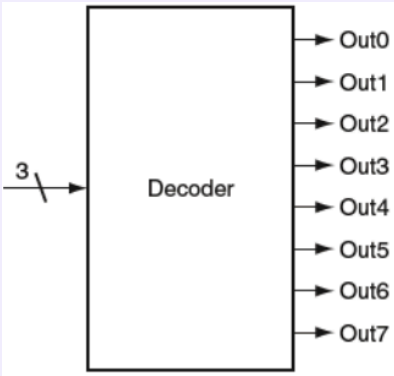

# Decoder

Il decoder è un circuito combinatorio che permette di scegliere uno di $2^n$ output, avendo $n$ input. I decoder sono utili per selezionare qualcosa. Vengono usati ad esempio per scrivere in un registro anziché in un altro.

**Example**

Nell'immagine sotto, abbiamo 3 bit di input che permettono di selezionare uno di 8 bit ni output. Interpretiamo gli ingressi A B C (o I2 I1 I0) come le cifre di un numero in base 2

con A (I2) quella più significativa.




a. A 3-bit decoder

| Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 11 | 10 | Out7 | Out6 | Out5 | Out4 | Out3 | Out2 | Out1 | Out0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

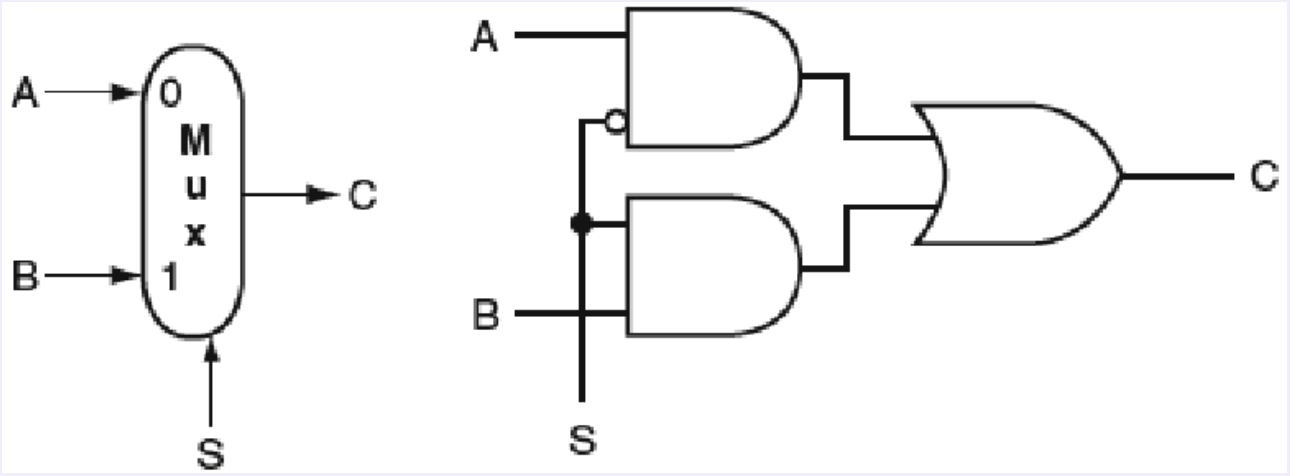
b. The truth table for a 3-bit decoder

# Multiplexer

Il multiplexer è un circuito combinatorio che dati $2^n$ input, permette di selezionare in uscita 1 solo di questi input avendo $n$ ingressi di controllo. La linea di controllo viene chiamata anche "selettore". I multiplexer sono utili per selezionare qualcosa. Vengono usati ad esempio per prendere qualcosa da uno specifico registro anziché da un altro (magari una lw o una add di due registri). Per convenzione, quando il valore di ingresso è 0, viene scelto A.

**Example**

Nell'esempio sotto, abbiamo $n = 1,\ 2^n = 2$. A destra anche il multiplexer sulla sinistra costruita con le porte logiche.

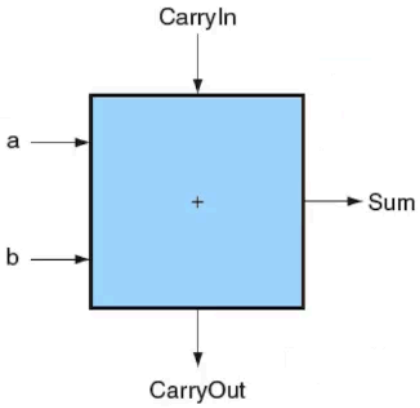


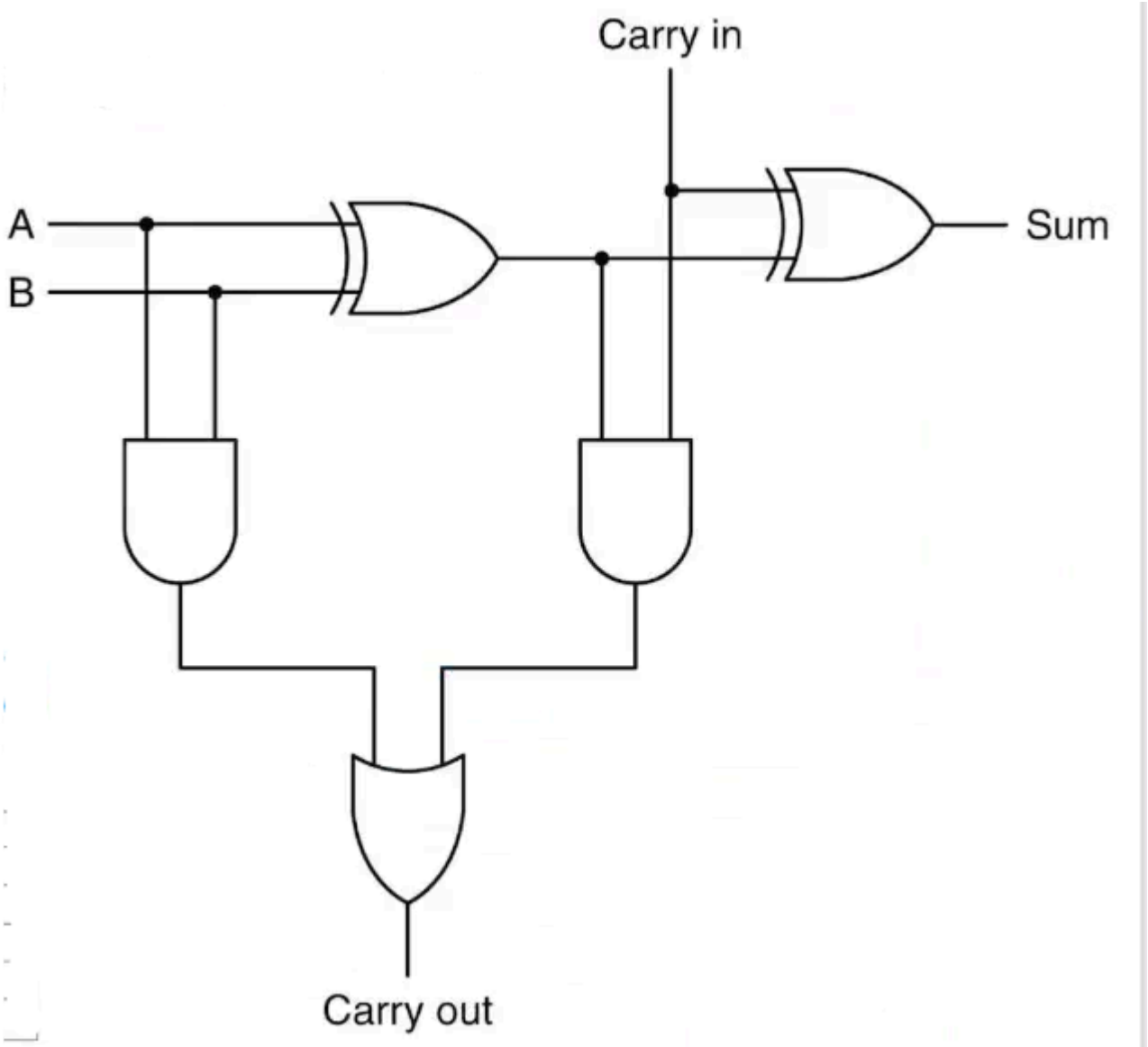

# Addizionatore

- Riceve in ingresso due bit (cifre in base 2) da sommare.
- Riceve in ingresso un bit (cifra in base 2) di riporto, il CarryIn.
- Restituisce un bit in uscita che rappresenta il risultato, Sum.

- Restituisce un bit in uscita che rappresenta il riporto CarryOut nello schema.
  Dato che c'é anche il CarryIn (il riporto), questo è un full adder

| Inputs | | | Outputs | |
|---|---|---|---|---|
| a | b | CarryIn | CarryOut | Sum |
| 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 |



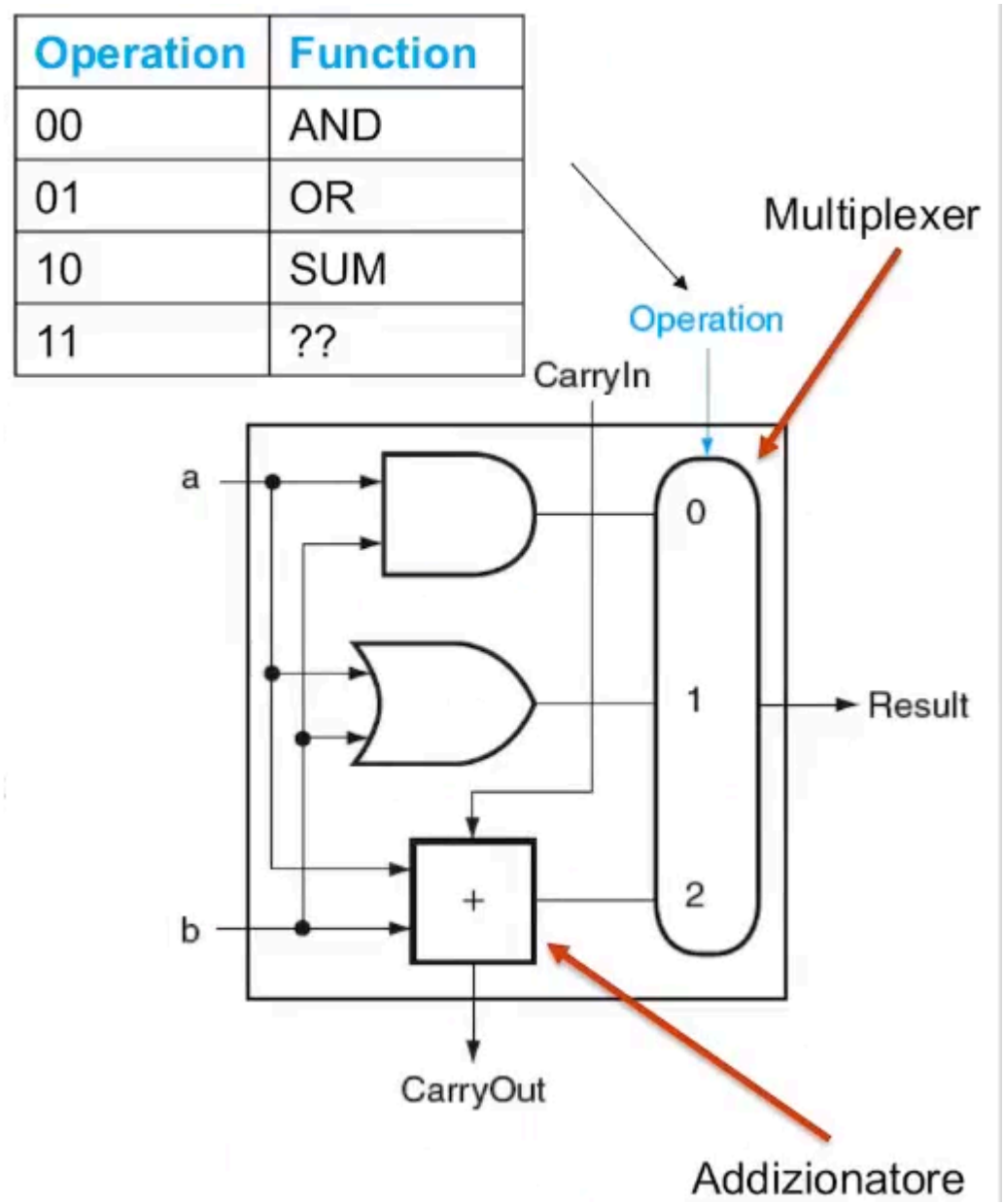




# ALU (1 bit)

ALU (o Arithmetic Logic Unit) è un unità fondamentale di un processore. Permette di eseguire un certo calcolo tra input dati. Le operazioni che possono essere eseguite sono and, or e somma. La ALU è formata da :

- Gli ingressi degli input A e B.
- L'ingresso di CarryIn preso da un'altra ALU o impostato a 0 se è la prima ALU.
- L'ingresso di Operation, il quale permette di selezionare quale operazione si vuole svolgere.
- L'uscita di CarryOut dato dal full adder.
- L'uscita Result che restituisce il risultato dell'operazione selezionata.

| Operation | Function |
|---|---|
| 00 | AND |
| 01 | OR |
| 10 | SUM |
| 11 | ?? |



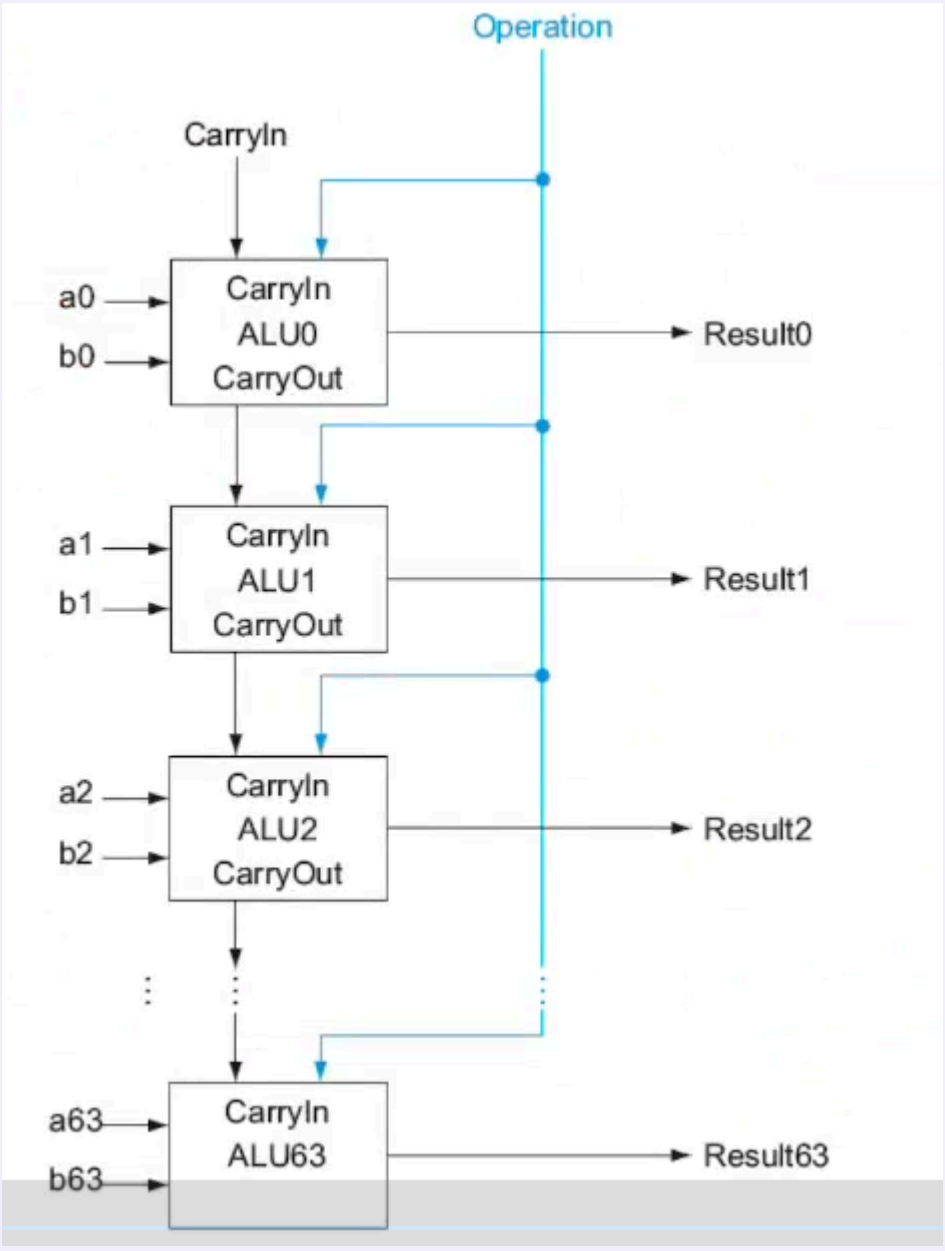

Come si vede da sopra, le operazioni vengono eseguite tutte quante e poi viene scelto il giusto output tramite il multiplexer.

## ALU (a $2^n$ bit)

Per svolgere un calcolo tra due registri da 64 bit non basta un ALU da 1 bit, ma avremo bisogno di 64 ALU da 1 bit collegate in serie. Il risultato di ogni ALU viene preso e messo nella giusta posizione di un altro registro a 64 bit.

**Example**

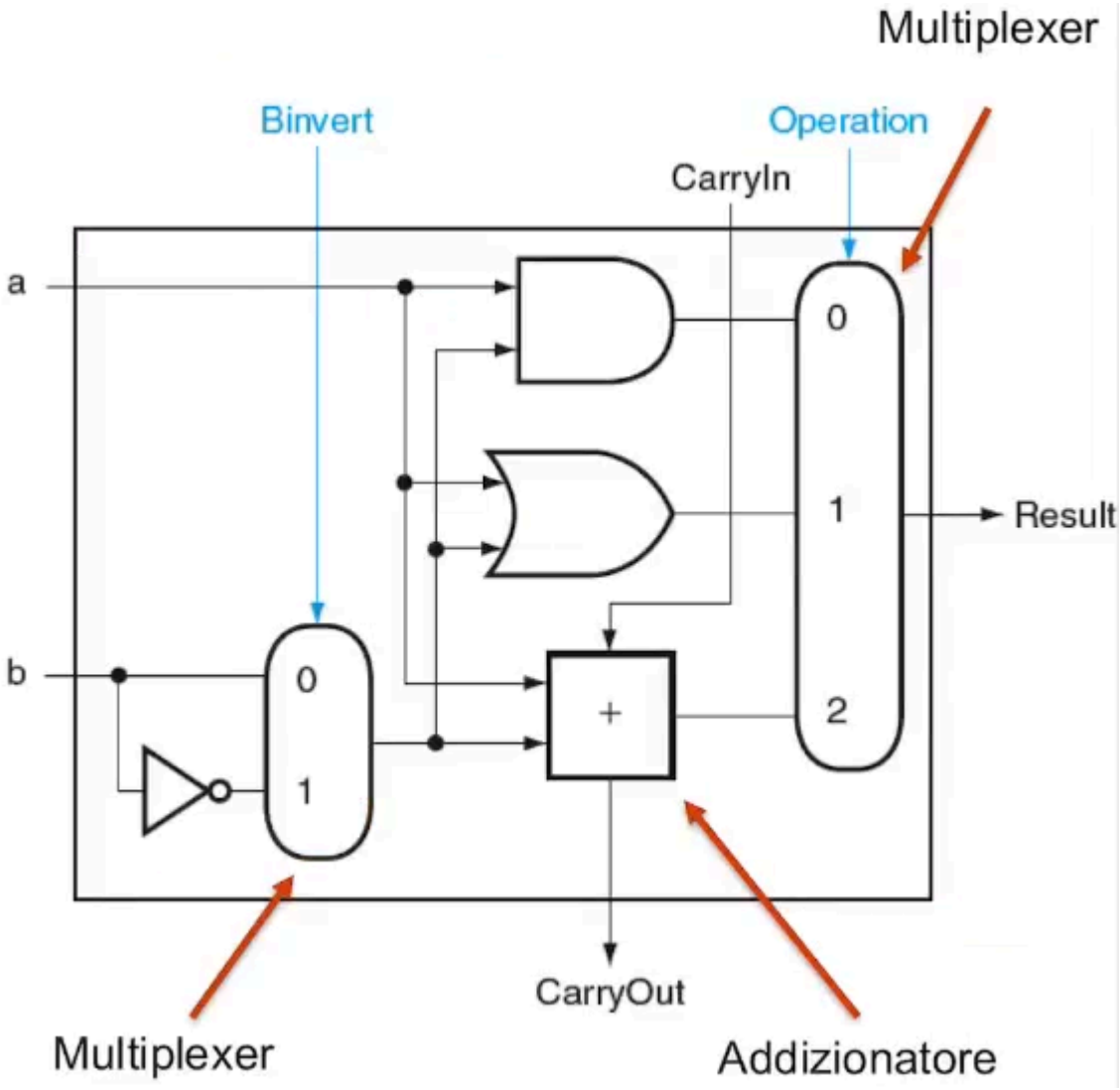
Sotto si può vedere un esempio. Si pensi ad A e B come due registri di cui si vuole fare la AND. Avremo $a_n$ con $a \in$ A e con $n$ l'ennesimo bit di A, stessa cosa per $b_n$. L'operation invece è la stessa per tutti ovviamente.




## ALU con sottrazione

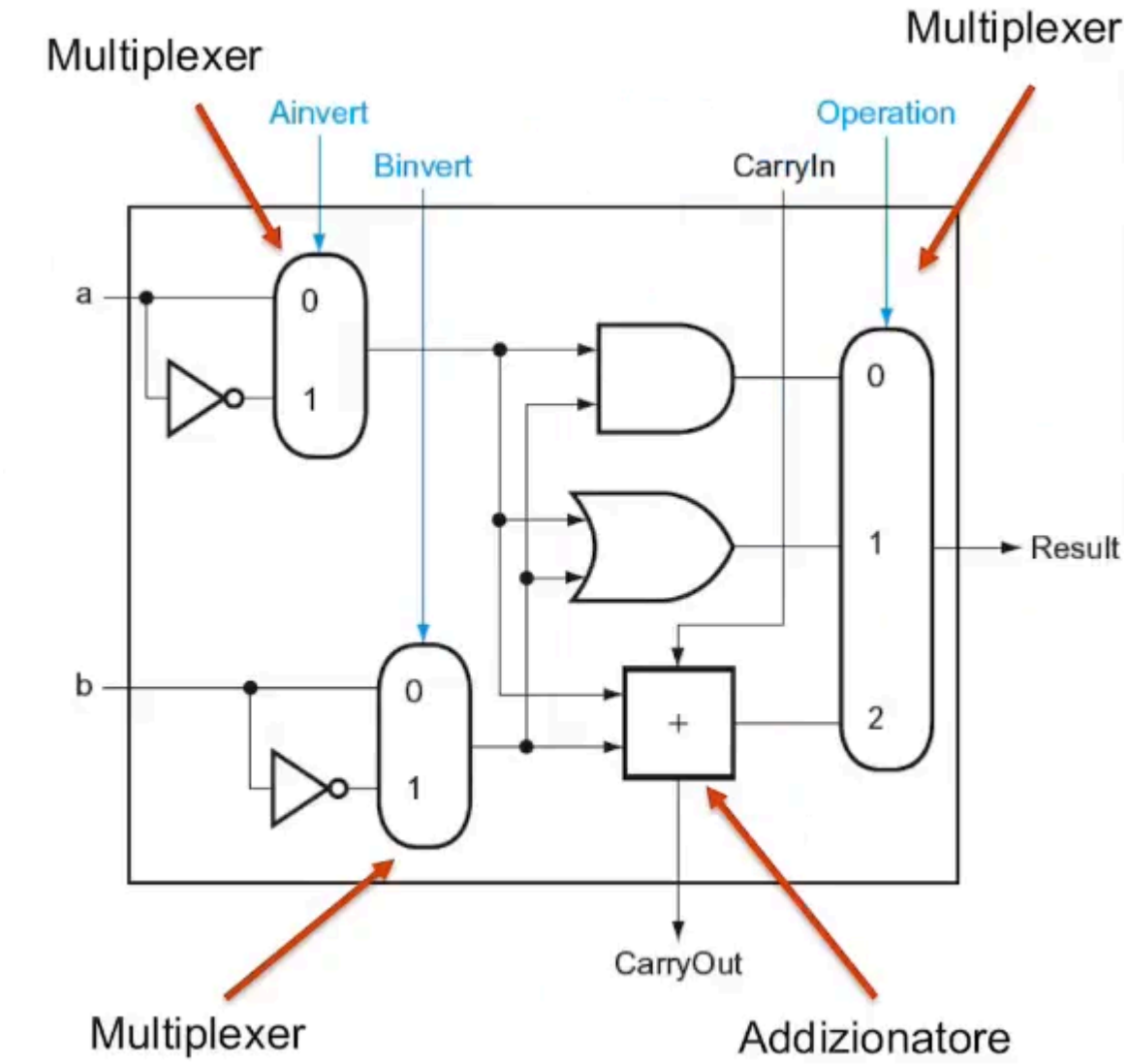
### Binvert

Come l'abbiamo vista fin ora, la ALU non può fare la sottrazione, ma in realtà è una cosa che si può rimediare molto facilmente. La sottrazione non è altro che $A + (-B)$, ovvero $A$ + l'opposto di B. L'opposto si ottiene facilmente facendo il complemento a 2 di B, ovvero facendo il NOT di ogni bit di B e aggiungendo un 1 alla fine (aggiungere l'uno alla fine è molto facile, basta mettere il CarryIn del bit meno significativo ad 1). Allora, mettendo un nuovo input detto "Binvert" (B-Invert) e un multiplexer, possiamo scegliere di sottrarre due numeri.




## Ainvert

Facendo lo stesso ragionamento, potremmo volere invertire anche A, quindi, facciamo la stessa cosa fatta per B e otteniamo la possibilità di fare $\overline{(a\ OR\ b)}$ con l'uso di $(\overline{a}\ AND\ \overline{b})$.

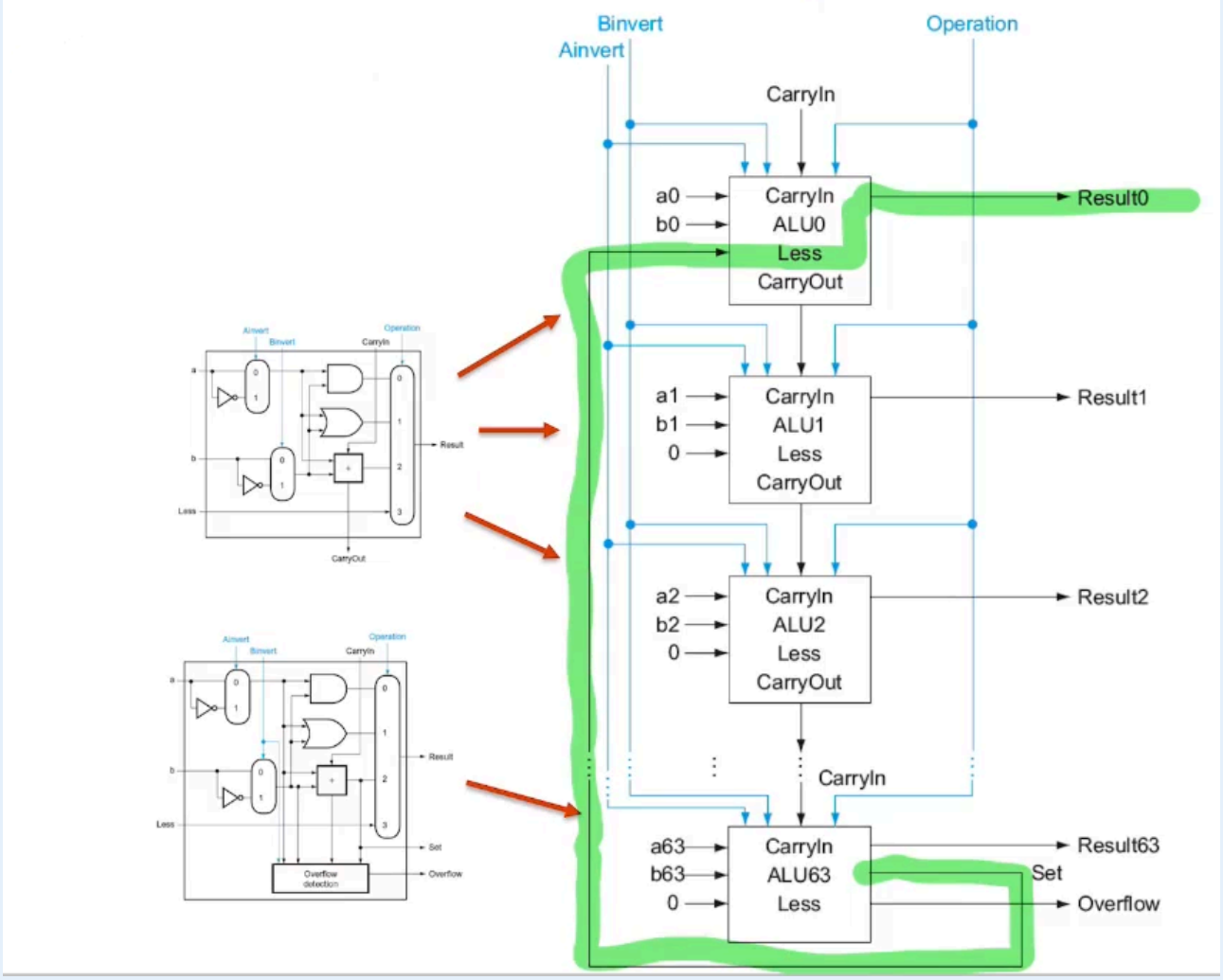



## SLT

L'istruzione slt (set less than) restituisce 1 se $rs1 < rs2$ e 0 altrimenti. In output avremo che il valore di tutti i bit (tranne quello meno significativo) valgono 0, mentre quello meno significativo prende il valore dal confronto. Per poter implementare ciò, basta vedere se $a - b < 0$, allora ciò implica che $a < b$. Come possiamo farlo? Basta eseguire la sottrazione e sapendo che i risultati sono salvati con i segni, basta controllare il bit più significativo. Se il valore di questo è 1, allora $a < b$. In modo fisico, per fare ciò, basta prendere il risultato dell'ultima ALU che fa il calcolo. Adesso, facciamo uscire il risultato della sottrazione di questo ultimo bit e lo riportiamo alla prima ALU come campo "LESS" (nelle altre ALU, mettiamo 0).

**Note**

Struttura finale della ALU




## Note

Ultima ALU con uscita SET