Architettura degli elaboratori lezione 9

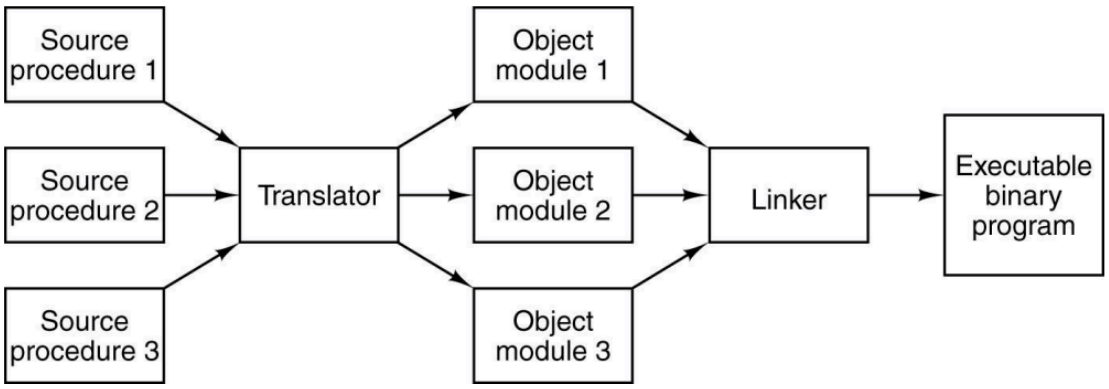
📚 Appunti di Davide Scarlata 2024/2025

👨‍🏫 **Prof:** Claudio Schifanella
✉️ **Mail:** claudio.schifanella@unito.it
📌 **Corso:** C
🔗 **Moodle Unito**
🗓️ **Data:** 24/03/2025

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 oggetto** in nodo da formare un unico **modulo eseguibile**



Linker

è un programma che esegue la funzione di collegamento dei moduli oggetto in nodo da formare un unico modulo eseguibile

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

Modulo	Lunghezza	Indirizzo partenza
A	400	100
B	600	500
C	500	1100
D	300	1600

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

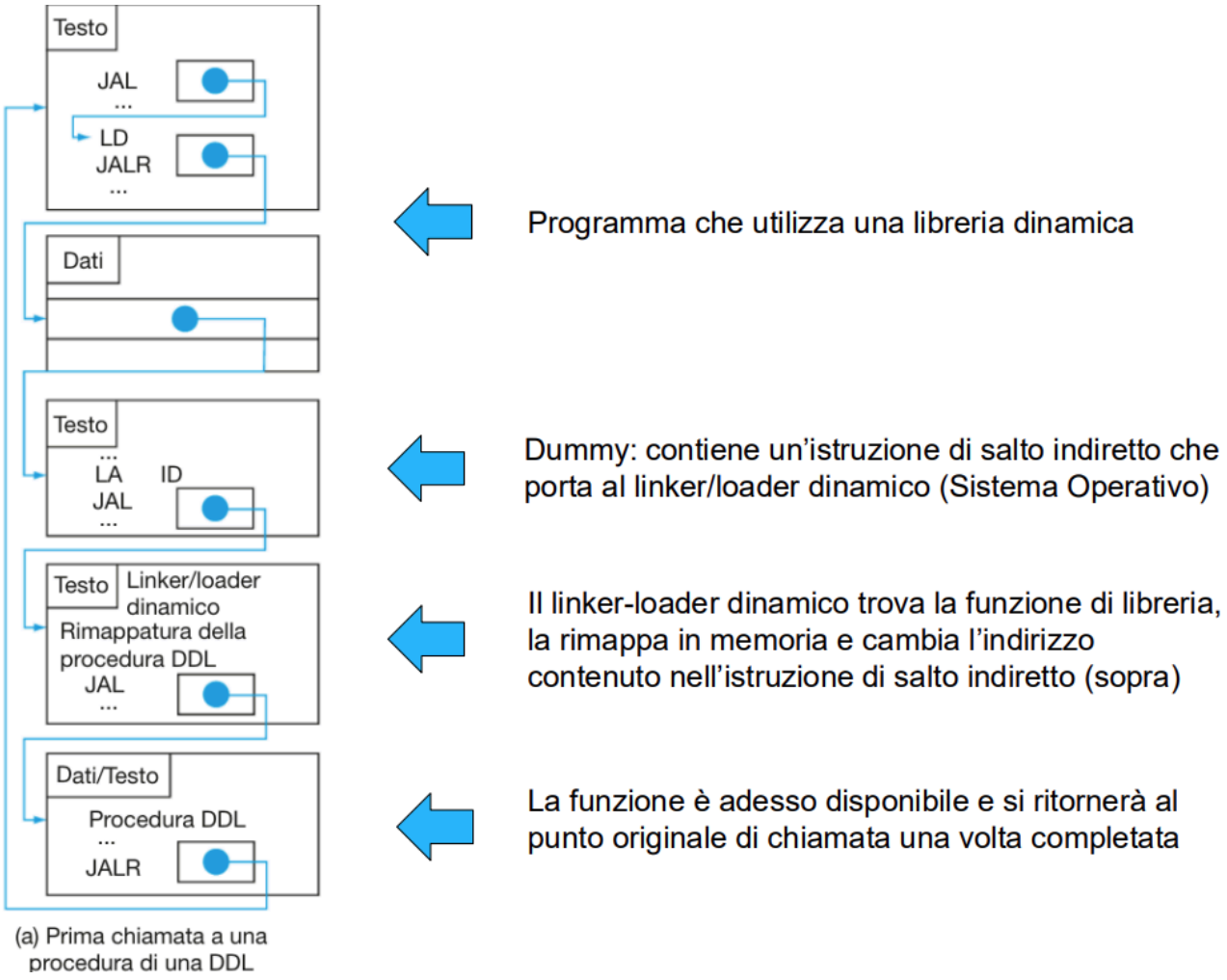
Binding e rilocazione dinamica

Collegamento statico:

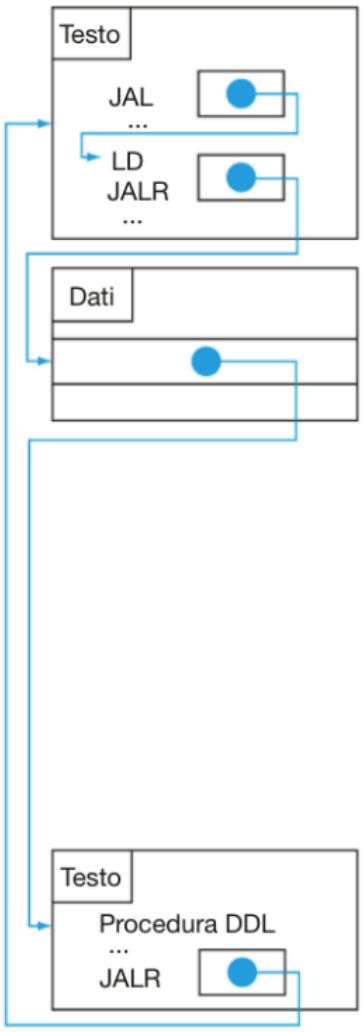
- Le funzioni di libreria diventano parte del codice eseguibile

- 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



(a) Prima chiamata a una procedura di una DDL

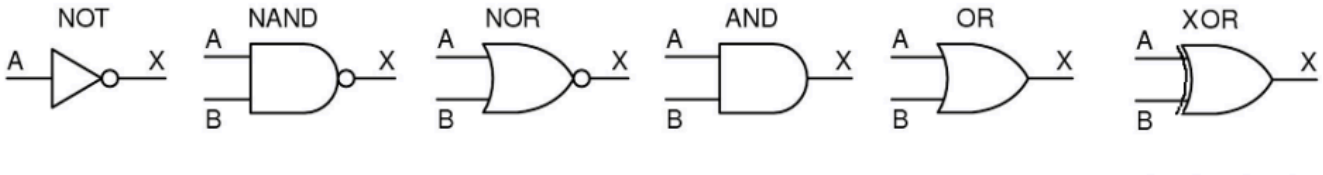


(b) Chiamate successive alla procedura della DDL

In seguito la chiamata alla funzione di libreria prevederà un salto indiretto unico alla sua prima istruzione, senza salti addizionali

Circuiti digitali

Porte logiche



A	X
0	1
1	0

(a)

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

(b)

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

(c)

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

(d)

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

(e)

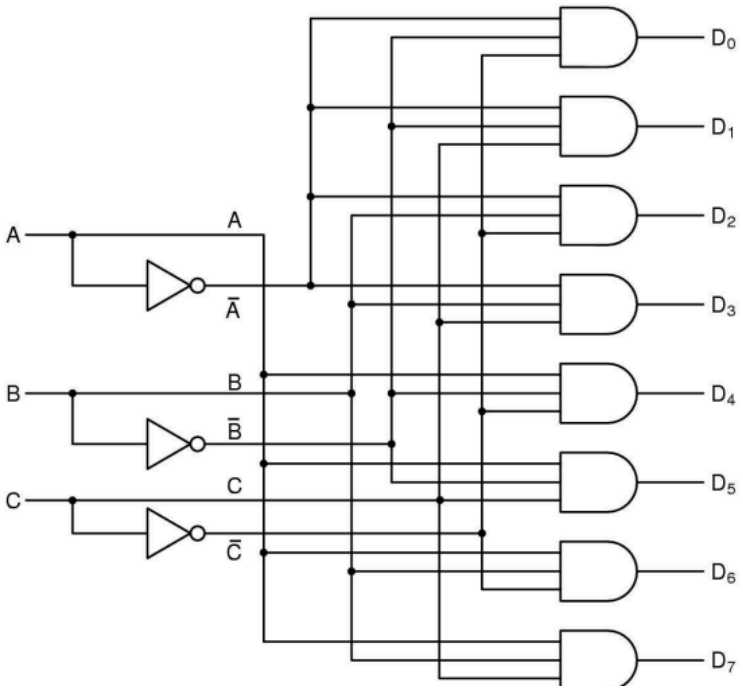
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

(f)

Circuiti combinatori

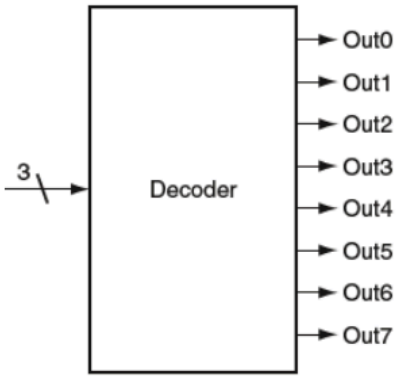
- Circuito combinatorio: l'output viene **determinato solo dagli input**

- Convenzione: l'incrocio tra due linee non implica alcuna connessione a meno che non sia presente il simbolo • nel punto di intersezione



il decoder

- Decoder: prende un numero di n bit come ingresso e lo usa per selezionare (mettere a 1, asserire) una delle 2 alla n linee di uscita
- Può essere utilizzato per attivare una certa componente (vedi ALU più avanti), oppure un banco di memoria, ecc.
- 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

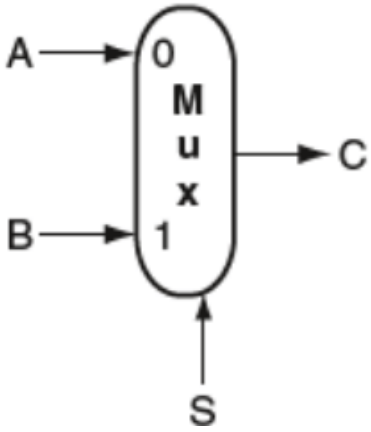
Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

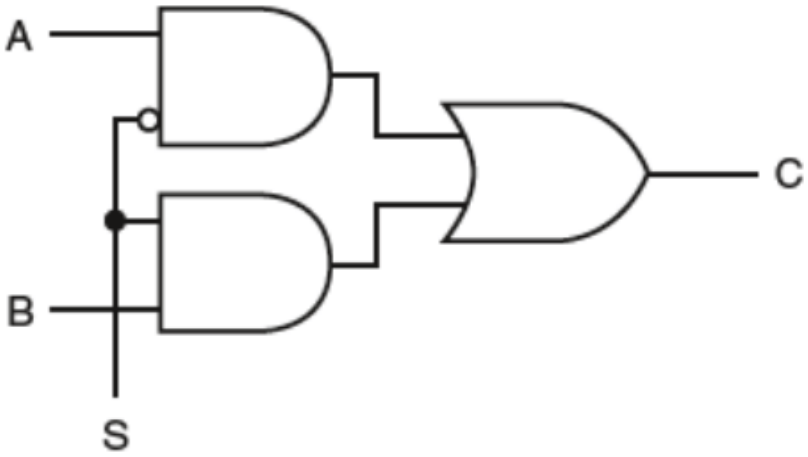
b. The truth table for a 3-bit decoder

multiplexer

Multiplexer (o selettore): 2 ingressi, 1 uscita e 1 ingresso di controllo

- La linea di controllo determina quale dei 2 ingressi deve essere selezionato per essere inviato all'uscita
- Esempio con 2 ingressi (A e B), un uscita (C) ed un ingresso di controllo (S)

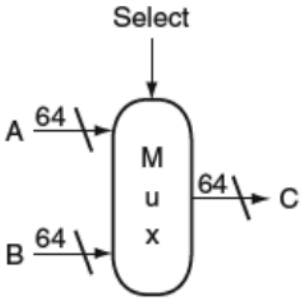


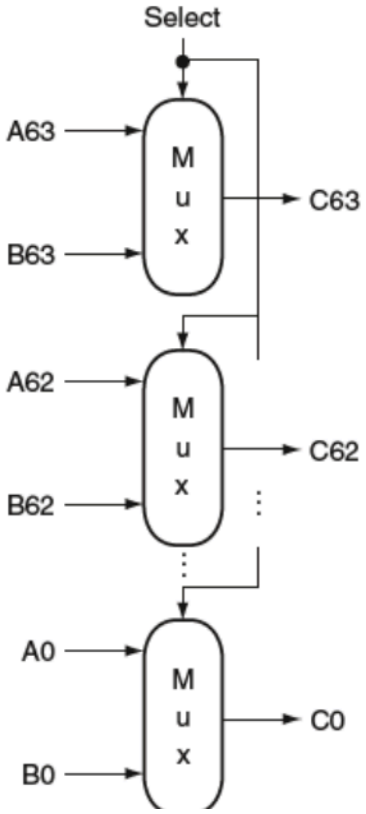


Multiplexer (o <u>selettore</u>): 2 ingressi da 64 line ciascuno, 1 uscita da 64 linee e *1* ingresso di controllo

La linea di controllo determina quale dei 2 ingressi da 64 linee deve essere selezionato per essere inviato all'uscita da 64 linee

Esempio con 2 ingressi (A e B), un uscita (C) ed un ingresso di controllo (Select)



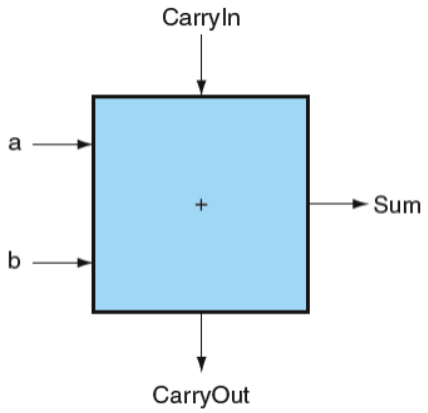


addizionatori

- riceve in ingresso due bit (cifre in base 2) da sommare, a e b nello schema
- riceve in ingresso un bit (cifra in base 2) di riporto, CarryIn nello schema
- restituisce un bit (cifra in base 2) in uscita che rappresenta il risultato, Sum nello schema

- restituisce un bit (cifra in base 2) in uscita che rappresenta il riporto, CarryOut nello schema

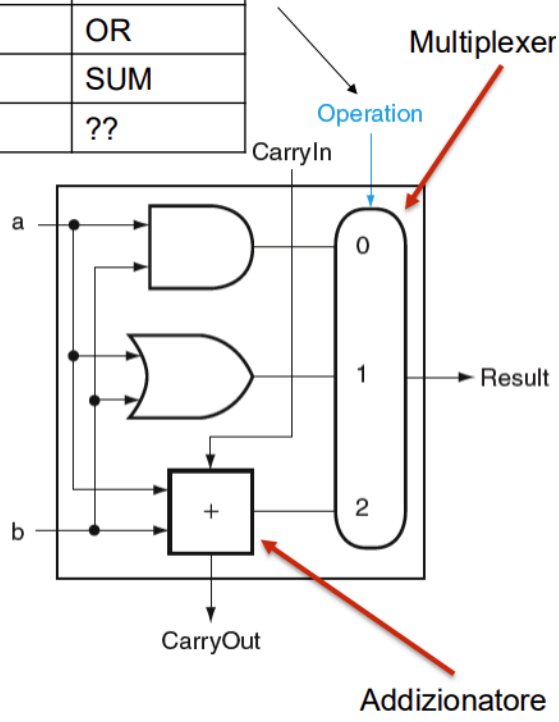
Inputs			Outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



RISC-V: ALU ad 1 bit

- Dati a e b è in grado di calcolare:
  - Result = a AND b
  - Result = a OR b
  - Result = a + b (somma)
- Un ingresso di controllo (Operation) seleziona l'operazione desiderata. Dovendo selezionare 3 possibili ingressi in realtà abbiamo 2 linee Operation di controllo.
- Un terzo ingresso fornisce in CarryIn per la somma
- Una seconda uscita rappresenta il CarryOut

Operation	Function
00	AND
01	OR
10	SUM
11	??



RISC-V: ALU ad 1 bit (sub)

Dati a e b è in grado di calcolare le funzioni precedenti con b o $\bar{b}$

Un ingresso di controllo aggiuntivo (Binvert) seleziona l'operazione desiderata

Serve, ad esempio, per le sottrazioni (a - b)

- Ricordare il complemento a 2
- Con l'uso di $\bar{b}$
- CarryIn = 1 per il bit meno significativo