Architettura degli elaboratori - lezione 13

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

29/04/2025

Contenuti

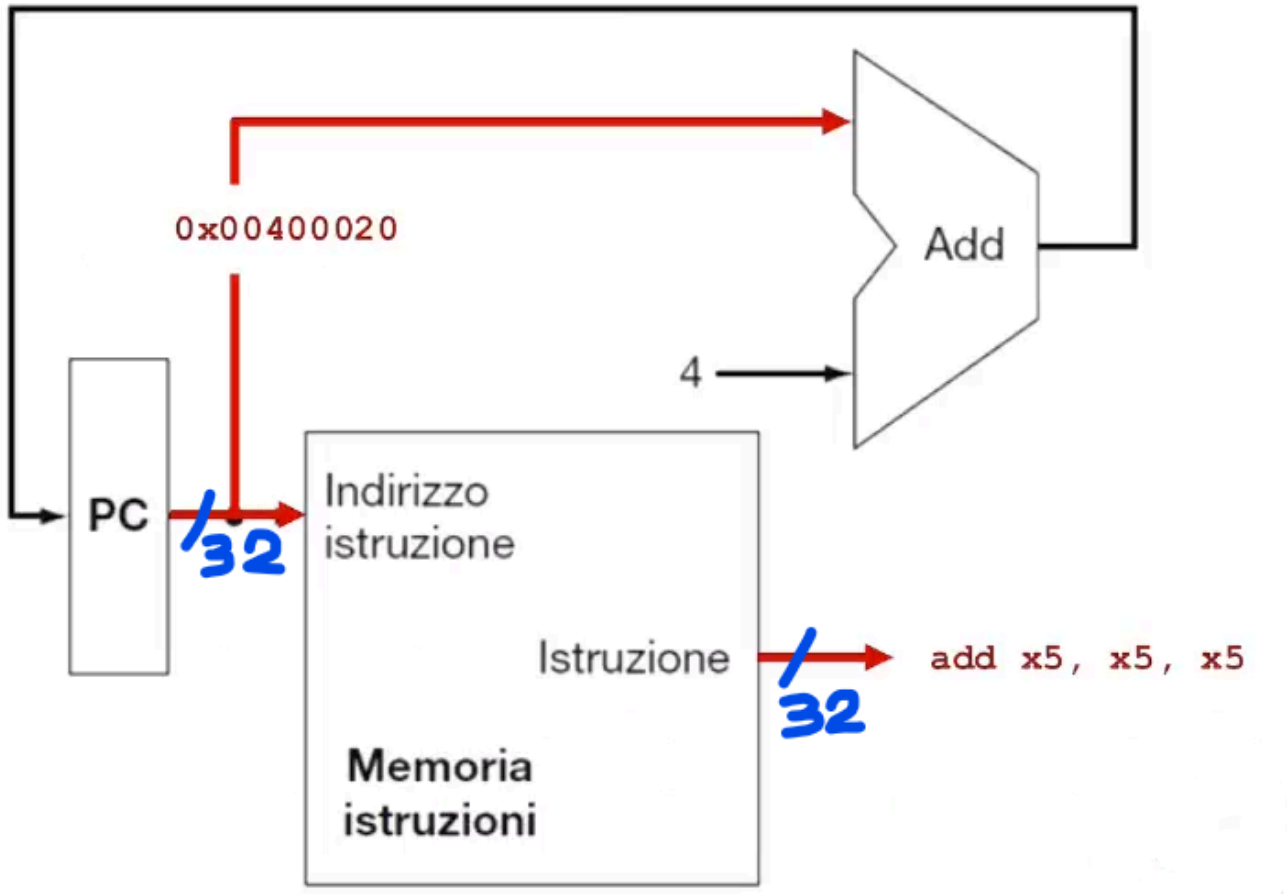
Metodologia di temporizzazione

Il segnale di clock determina quando gli elementi di stato scrivono la loro memoria interna. Gli ingressi a un elemento di stato devono raggiungere un valore stabile prima che il fronte attivo del clock provochi l'aggiornamento dello stato.
La metodologia sensibile ai fronti (edge-triggered) permette di leggere e scrivere un elemento di stato nello stesso ciclo di clock. Il ciclo di clock deve avere una durata sufficiente a garantire che gli ingressi siano stabili quando arriva il fronte attivo del clock.

Incremento del program counter

Il program counter manda l'indirizzo dell'istruzione corrente in due posti :

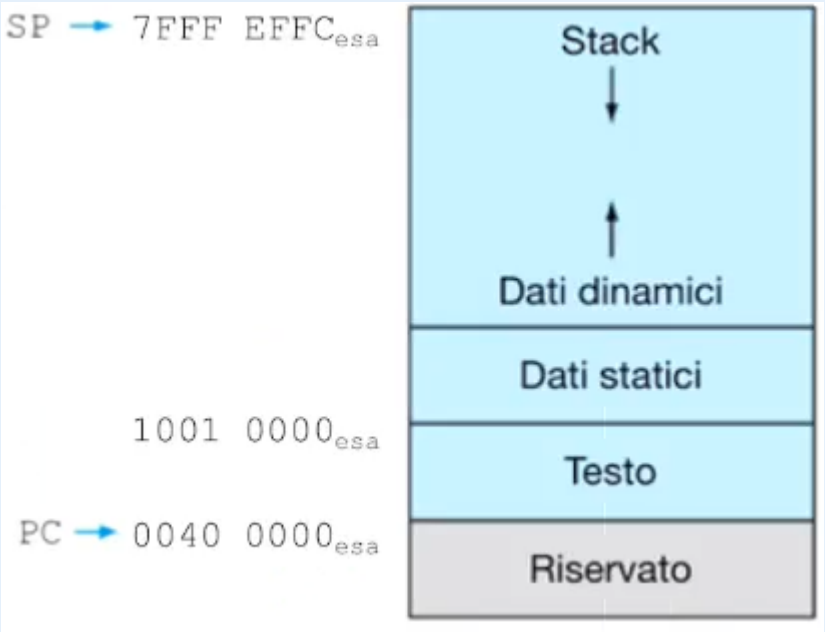
1. Memoria istruzione : si occupa di "capire l'istruzione".
2. Adder : aumenta, staticamente, l'indirizzo per avere l'indirizzo della prossima istruzione e il risultato lo rimanda al program counter (in un'architettura a 32 bit, l'indirizzo viene incrementato di 4 byte (32 bit)).
   L'incremento avviene in contemporanea con l'accesso alla memoria (per la figura sotto, si intende che quando l'indirizzo esce da PC, va contemporaneamente sia a "Memoria istruzioni" sia all'add (ALU) che incrementa staticamente di 4).



Note

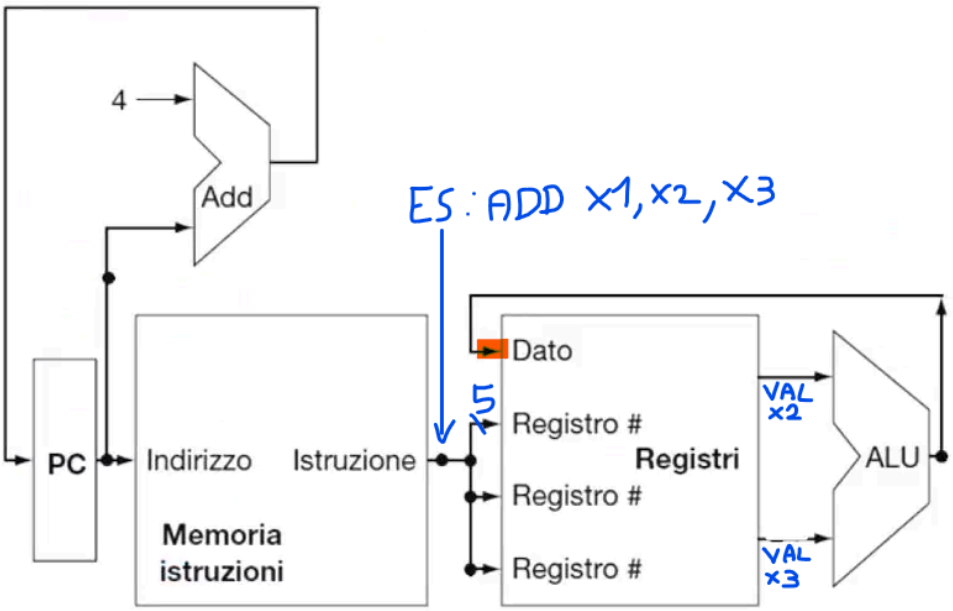
- Il PC non cambia il suo contenuto fino al primo fronte di salita.
- La "memoria istruzioni" non ha un segnale di controllo per scrivere, perché non si può scrivere nel segmento di testo. Guarda figura sotto per capire. (Dividiamo concettualmente "memoria istruzioni" e "memoria dati", ma fisicamente sono una cosa

sola).



Data path per le istruzioni di tipo R

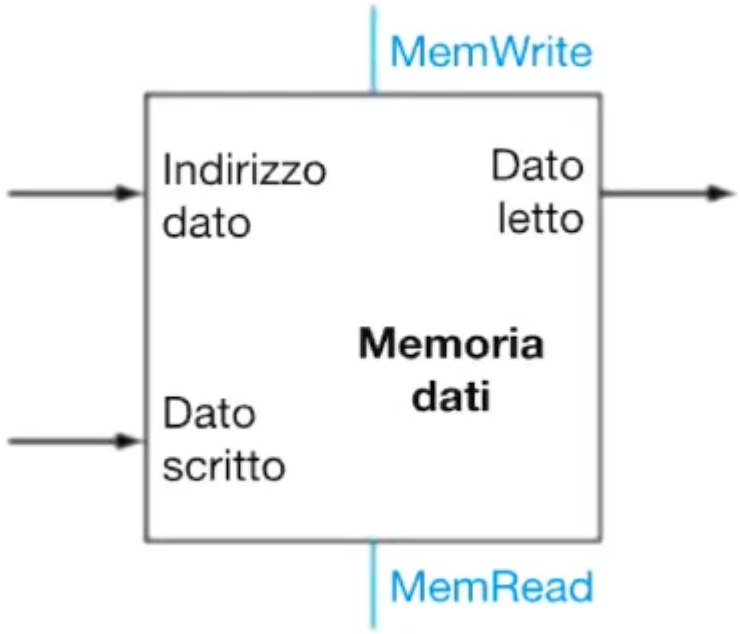
Nel data path per le istruzioni di tipo R troviamo sempre il circuito come sopra e in più troviamo il register file e l'ALU. Il register file ci serve per prendere prima i valori contenuti nei due registri salvati e poi per salvare il risultato dell'operazione calcolato dall'ALU nell'apposito registro indicato.



Gli ingressi con su scritto "Registro #" prendono 5 bit in ingresso sempre per il discorso che avendo 32 registri, ci servono 5 bit per selezionarli ($\log_2 32 = 5$).
L'ingresso "Dato" è evidenziato in rosso per far notare che li verrà salvato il risultato dell'operazione fatta dalla ALU, ma solo sul fronte di salita successivo.

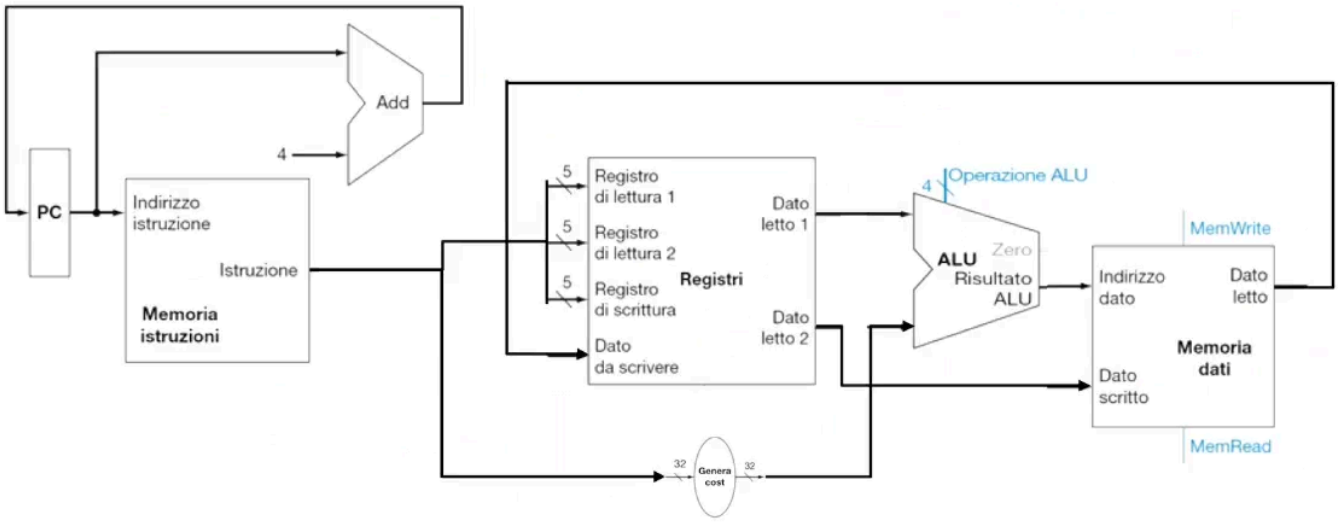
Memoria dati

Questa è la memoria RAM. L'ingresso "indirizzo dato" è grande 32 bit. "Dato letto" invece è variabile, se facciamo "lb" o "lh" o... otteniamo risultati diversi (rispettivamente 8 e 16 bit). Troviamo anche due segnali di controllo ("MemWrite" e "MemRead") che ci permetto di scrivere o leggere.



Genera cost (unità di estensione del segno)

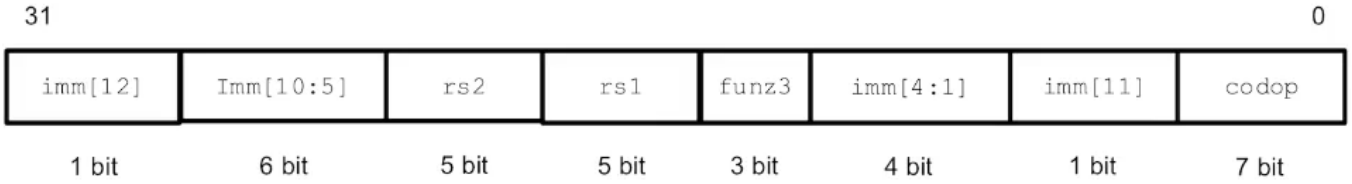
A genera cost passiamo l'istruzione intera. Questa si occuperà di estrarre "cost" composta da 12 bit in molti tipi di istruzioni (tipo I, S) e poi ne estende il segno (perchè la ALU prende in ingresso sempre 32 bit, non ne bastano 12. Inoltre, in input, prende 32 bit perché la posizione dei vari "cost" nei vari tipi di istruzione è diverso, quindi gli si deve passare l'intera istruzione e poi lui si occuperà di prendere i bit giusti).



Questa sopra è il modo in cui "Genera cost" e "memoria dati" vengono implementati nel processore.

Supporto dell'istruzione beq (tipo SB)

BEQ è un'istruzione che richiede il confronto fra due valori e che consente il trasferimento del controllo a un altro indirizzo del programma a seconda del risultato del confronto. Le istruzioni di salto condizionato utilizzano il formato di tipo SB.



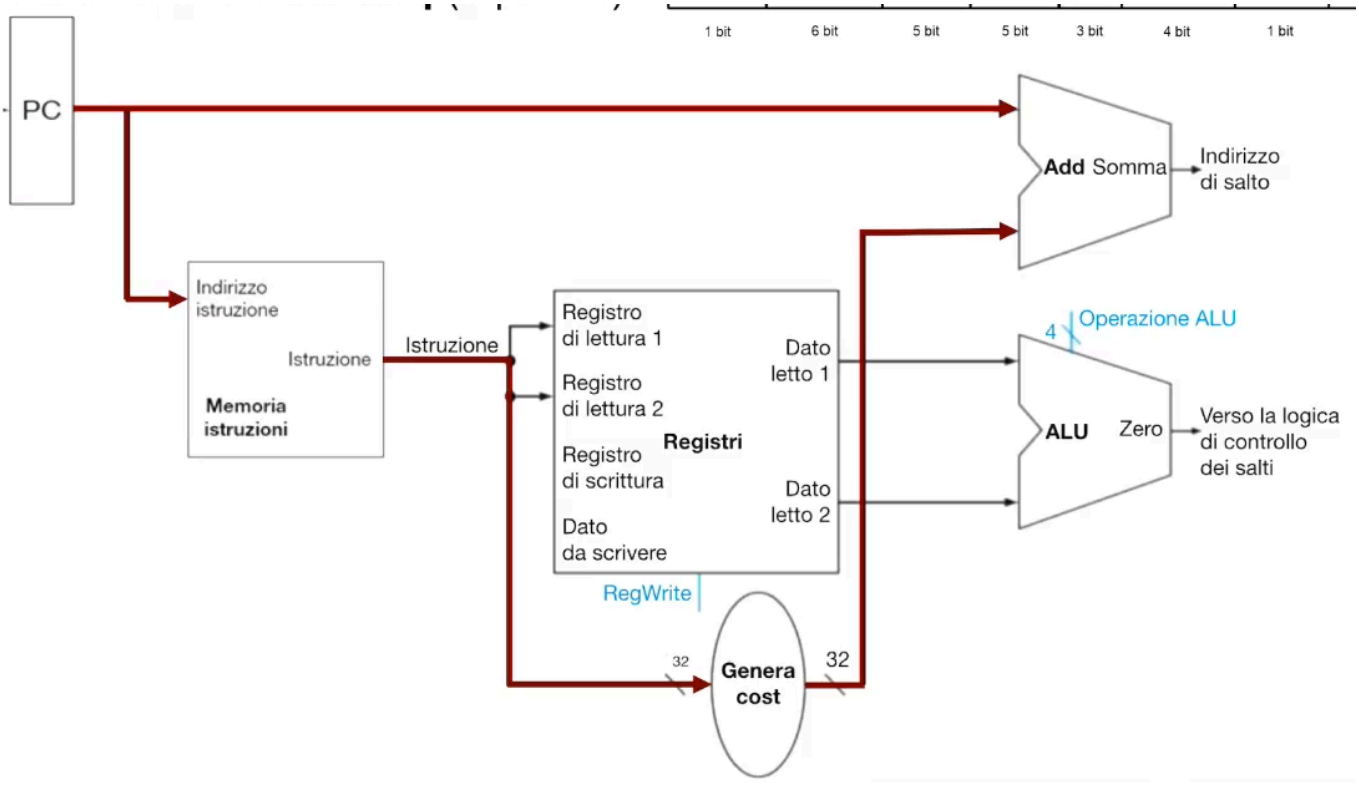
La struttura del formato SB

Come funziona?

Se due valori sono uguali, viene calcolata la distanza (può essere sia negativa che positiva, -4096/+4094) tra beq (l'attuale PC) e l'etichetta passata. Fatto il calcolo, si modifica il valore del PC.

Implementazione

Per implementare in modo hardware questa possibilità di salto abbiamo bisogno di alcuni componenti che abbiamo già visto sopra. Ecco come possiamo farlo :



Come possiamo vedere, dobbiamo prendere da "Genera cost" (che si occupa di fare

l'estensione del segno e uno shift a sinistra di 1 bit) e sommarlo al program counter. Avremo poi bisogno di un qualche controllo per dire che se "zero" della ALU = 1 (ovvero i due numeri sono uguali), allora dobbiamo prendere il risultato dell'indirizzo di salto, altrimenti il normale incremento di 4 sul PC.

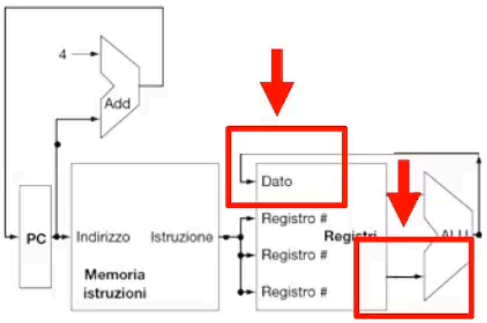
> **Note**
>
> A cosa serve shiftare a sinistra di 1 bit il risultato? Beh, serve perché non ci servono mai salti da 1 bit, quindi si shifta di uno per avere più spazio. Infatti si può notare che ci sono solo 12 bit (quindi $2^{12} = 4096$), ma il range è da -4096 a +4094. Shiftando abbiamo appunto questo bit "in più" che ci permette di raddoppiare l'estensione del salto.

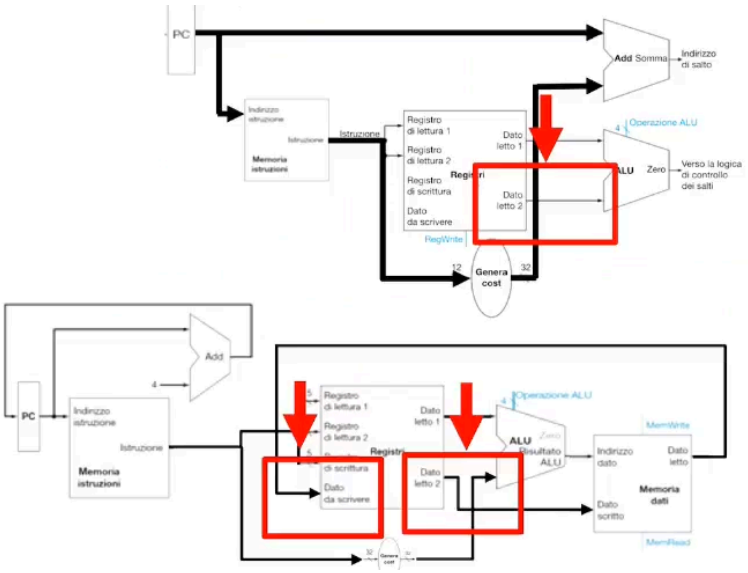
Unità di elaborazione unificata

Finora abbiamo visto come fare le istruzioni di tipo R, I, S, i salti e come usare genera cost. Abbiamo sempre usato circuiti diversi, ma molto simili tra di loro. Implementare ogni volta questi circuiti sarebbe uno spreco (oltre che poco fattibile a volte, non possiamo sdoppiare il Register file ad esempio), quindi dobbiamo trovare un modo di unificare questi circuiti.

Dove sono le differenze?

Ne vediamo alcune qua sotto :

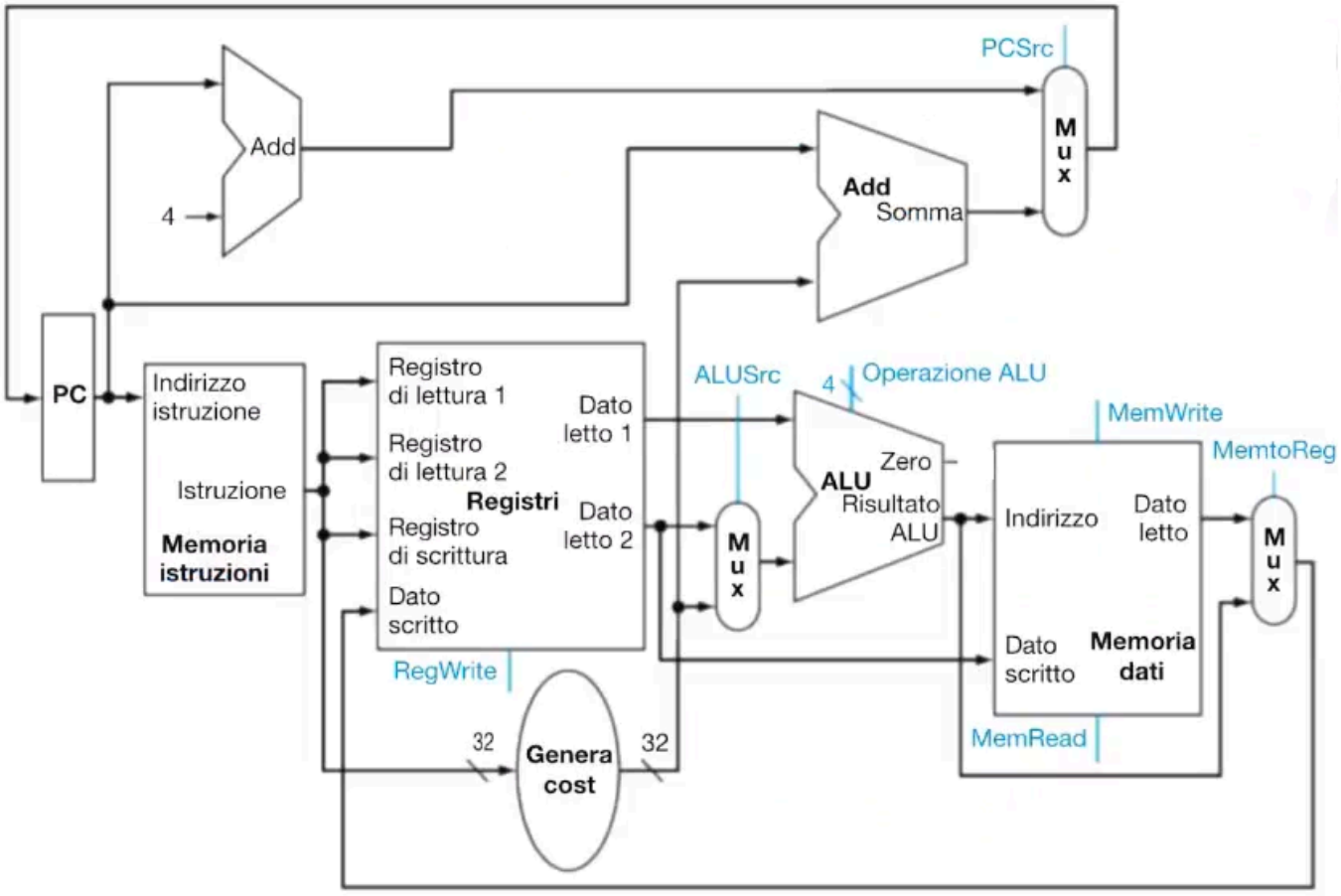




Si nota che se unissimo tutto assieme, avremmo dei problemi. Prendiamo in esempio l'ingresso 2 della ALU. Nelle istruzioni R è direttamente collegato al register file, mentre nelle istruzioni di tipo S/I è collegato a genera cost, come possiamo fare? Beh, la soluzione è semplice, basta usare un multiplexer per decidere quale valore far passare.

Risultato

Collegando quindi tutti gli output e input che vanno in contrasto in modo corretto, otteniamo qualcosa del genere.



Notiamo però, che abbiamo molte linee di controllo che non abbiamo visto (ALUSrc, MemtoReg, PCSrc), chi le imposta queste linee? Queste linee vengono impostate dall'unità di controllo.

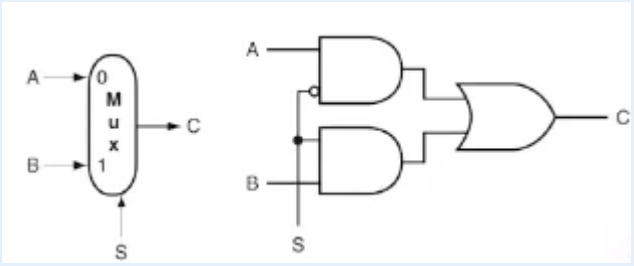
Note

Nell'immagine sopra, i circuiti combinatori sono :

- Le varie ALU (ALU e 2 add, le quali sono ALU semplificate)
- I multiplexer
- Genera cost
  Invece, sono sequenziali tutte le memorie (PC, memoria istruzioni/dati e register file), perché contengono i dati, quindi ovviamente l'output varia in base ad uno stato interno.

Note

Ricordo che un multiplexer (con 2 ingressi e 1 di controllo) è formato in questo modo :



Unità di controllo

L'unità di controllo è un circuito combinatorio che si occupa di prendere in ingresso un'istruzione da eseguire e da esse derivarne :

- Un segnale di scrittura per ciascun elemento di stato.
- Un segnale di selezione per ciascun multiplexer.
- I segnali di controllo per l'ALU (il controllo dell'ALU è particolare, conviene quindi progettarlo prima delle altre parti dell'unità di controllo).
  L'unità di controllo è un componente essenziale e costruirne la tabella di verità sarebbe molto complesso, pe questo non verrà trattato, ma cercheremo comunque di capire come funziona.

Ragionamento per la costruzione

Come possiamo quindi ottenere tutte le informazioni che vogliamo dall'istruzione? Beh, per prima cosa dobbiamo guardare il codop. In questo modo sappiamo di che istruzione si tratta. Inoltre dobbiamo vedere anche i possibili campi "funz3" e "funz7".

Istruzione (R)	funz7	rs2	rs1	funz3	rd	codop	Esempio
`add`	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
`sub` (sottrazione)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Come possiamo notare da questa immagine, add e sub (e anche altre istruzioni) hanno lo

stesso codop, quindi per differenziali, abbiamo bisogno di funz3/7 (e da questi poi possiamo controllare la ALU ad esempio per mettere Binvert ad 1, etc...).

Nell'immagine sotto, possiamo vedere varie istruzioni e i loro valori in vari campi. Possiamo notare che lw e sw ad esempio, non hanno nulla nel campo "funz3/7", perché non servono. Invece, l'ingresso nella ALU sia per lw che sw sono uguali, perché? Beh, perché devono fare sostanzialmente la stessa cosa, ovvero la somma dell'offset.
Notiamo invece, che per istruzioni diverse, gli ingressi della ALU cambiano parecchio.

Codice operativo istruzione	ALUOp	Operazione eseguita dall'istruzione	Campo funz7	Campo funz3	Operazione dell'ALU	Ingresso di controllo alla ALU
lw	00	load di 1 parola	XXXXXXX	XXX	somma	0010
sw	00	store di 1 parola	XXXXXXX	XXX	somma	0010
beq	01	salto condizionato all'uguaglianza	XXXXXXX	XXX	sottrazione	0110
Tipo R	10	add	0000000	000	somma	0010
Tipo R	10	sub	0100000	000	sottrazione	0110
Tipo R	10	and	0000000	111	AND	0000
Tipo R	10	or	0000000	110	OR	0001

I 4 bit di controllo della ALU possono essere generati utilizzando una piccola unità di controllo che riceve in ingresso i campi funz7 e funz3 dell'istruzione e un campo di controllo su 2 bit, chiamato ALUOp :

- ALUOp = 00, somma per le istruzioni di load e store
- ALUOp = 01, sottrazione per le beq
- ALUOp = 10, l'operazione viene determinata dal contenuto dei campi funz7 e funz3
  Per fare una tabella di verità derivata da questo ragionamento, abbiamo bisogno di 12 bit in input (2 bit per ALUOp, 7 per funz7 e 3 per funz3) e 4 in output (i 4 bit di "ingresso di controllo alla ALU"), ovvero una tabella da 4096 (2^{12}) righe.

Controllo operazioni della ALU

Ci sono molteplici livelli di decodifica :

1. L'unità di controllo principale imposta i bit ALUOp, poi utilizzati come ingresso dal secondo livello
2. Unità di controllo dall'ALU, usa i bit di ALUOp precedentemente impostati e genera i segnali effettivi dell'ALU.
   Più livello di controllo possono ridurre le dimensioni dell'unità di controllo principale, riducendo la latenza dell'unità di controllo (spesso l'unità di controllo è un elemento critico

per la definizione della durata del ciclo di clock, quindi livelli diminuire la latenza fa diminuire anche la durata del ciclo di clock).

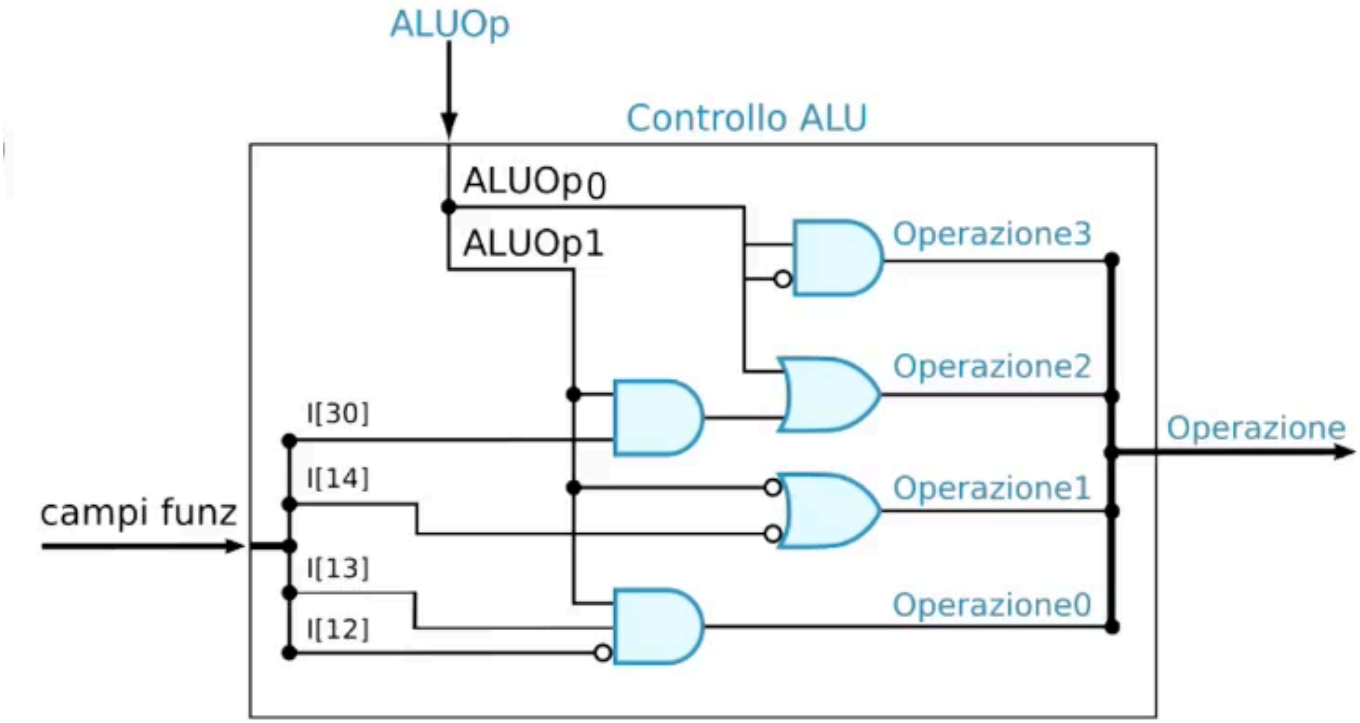
Tabella di verità per ottenere il controllo dell'ALU

Termini indifferenti (don't care)

	ALUOp		Campo funz7							Campo funz3			Operazione
	ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
lw / sw	0	0	X	X	X	X	X	X	X	X	X	X	0010
beq	X	1	X	X	X	X	X	X	X	X	X	X	0110
add	1	X	0	0	0	0	0	0	0	0	0	0	0010
sub	1	X	0	1	0	0	0	0	0	0	0	0	0110
and	1	X	0	0	0	0	0	0	0	1	1	1	0000
or	1	X	0	0	0	0	0	0	0	1	1	0	0001

Tabella di verità: Corrispondenza tra i 2 bit del campo ALUOp e i bit dei campi funz con i 4 bit di controllo della ALU che selezionano l'operazione

Non è importante saperla. Non viene approfondita più di così, però è utile avere un'idea generale di come potrebbe funzionare.



L'unità di elaborazione con segnali di controllo

Detto tutto questo, possiamo vedere il circuito finito, il quale si comporrà nel seguente modo :