UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Datapath, Controllo e Finite State Machine (FSM)

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

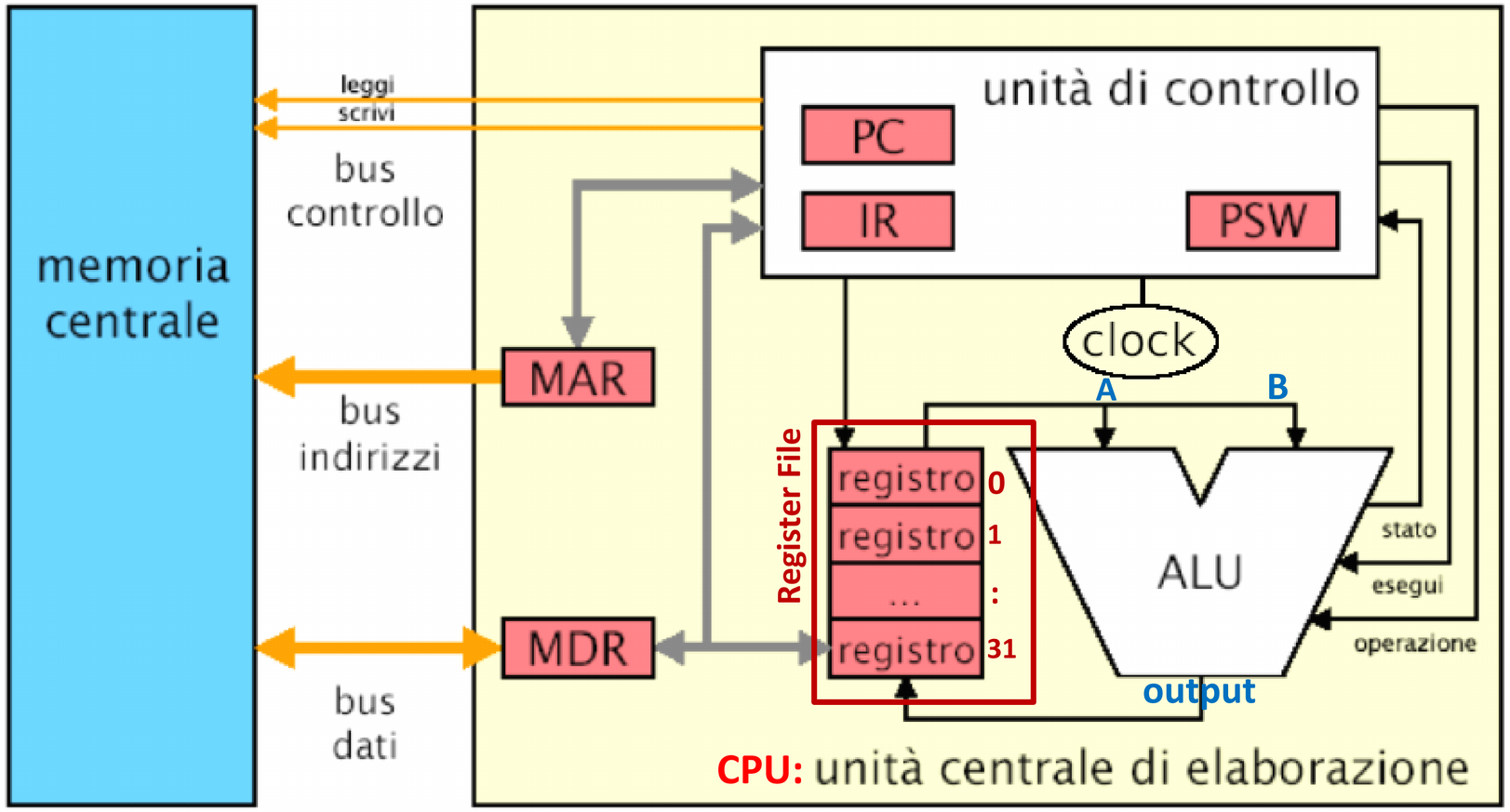
Trieste, 14 aprile 2026

AGENDA DI OGGI E GIOVEDÌ

- Datapath
- Programmable logical array (PLA)
- Controllo del datapth: requisiti
- Finite State Machine (FSM)
- Control Unit (CU)

Jigsaw!

RECAP



Con questa «architettura» è possibile realizzare tutte le istruzioni dell'ISA (MIPS32).

DATAPATH VS CONTROL UNIT

Costruiamo il datapath

Datapath (flusso dei dati)

- ALU per eseguire le operazioni
- Register File contiene dati (più accessibili alla CPU)
- *Memoria RAM contiene istruzioni e dati*
- Bus di sistema include le connessioni tra le varie parti dell'architettura (distinte per funzione: bus di controllo, bus di indirizzo, bus dei dati)

Control Unit (unità di controllo)

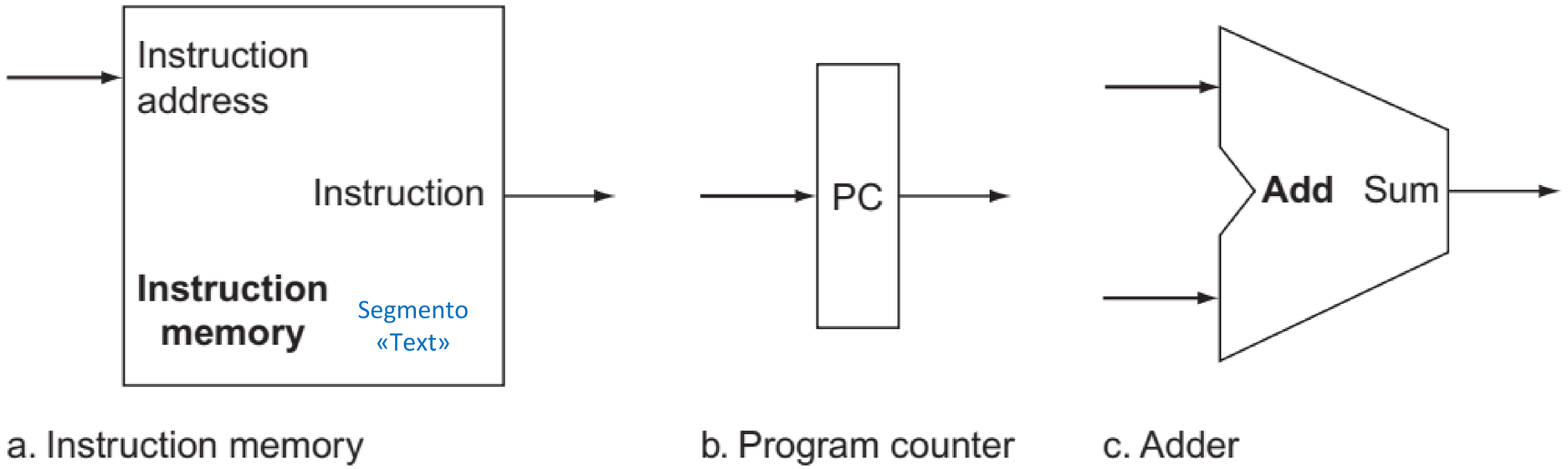
- attiva i segnali di controllo giusti (per ogni mux/decoder)
- usando il clock scandisce le fasi (fetch, decode, exe)
- permette di differenziare l'esecuzione (uso del datapath) in base all'istruzione

NOTA. Anche la ALU ha una sua piccola control unit.

AGENDA DI OGGI E GIOVEDÌ

- **Datapath**
- *Programmable logical array (PLA)*
- *Controllo del datapth: requisiti*
- *Finite State Machine (FSM)*
- *Control Unit (CU)*

DATAPATH PER IL **FETCH** DI UN'ISTRUZIONE (1/2)



Andare a prendere l'istruzione in memoria

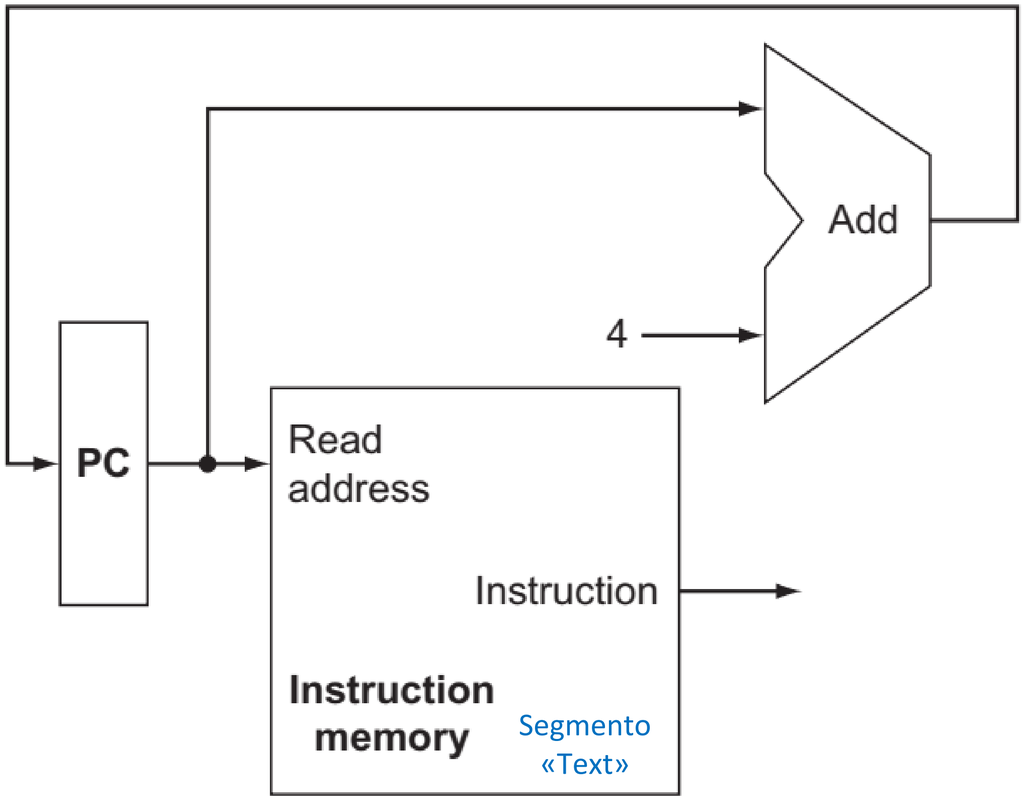
Tenere a portata di mano (per la CPU) l'indirizzo dove si trova l'istruzione in esecuzione

Aggiornare l'indirizzo per leggere la prossima istruzione

Patterson & Hennessy (Cap.4.3)

DATAPATH PER IL **FETCH** DI UN'ISTRUZIONE (2/2)

Tenere a portata di mano (per la CPU) l'indirizzo dove si trova l'istruzione in esecuzione

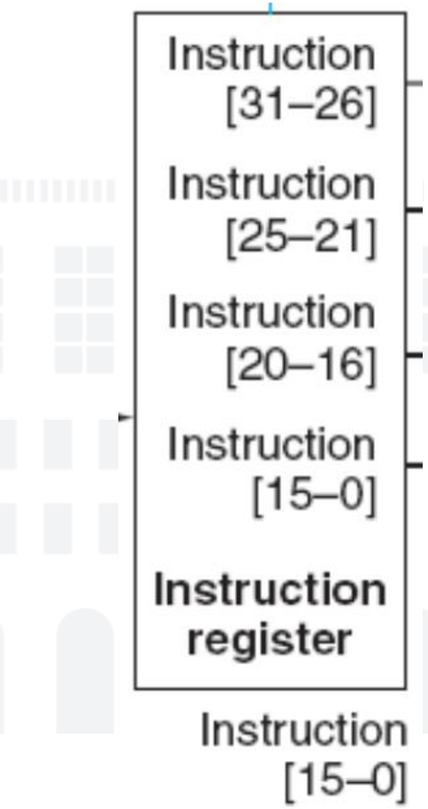


Aggiornare l'indirizzo per leggere la prossima istruzione

Andare a prendere l'istruzione in memoria

DECODE

Il processore MIPS legge i vari campi dell'istruzione e identifica il tipo di istruzione da eseguire (**OPCODE** e **FUNC CODE** se necessario).



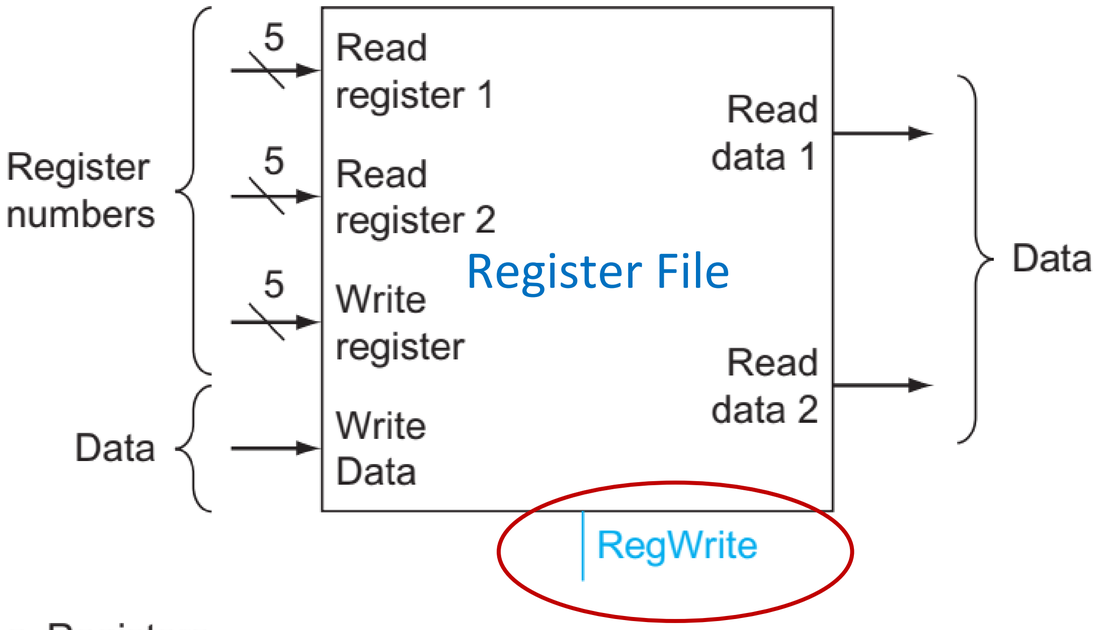
UNIVERSITÀ DEGLI STUDI DI TRIESTE

add $10, $8, $9

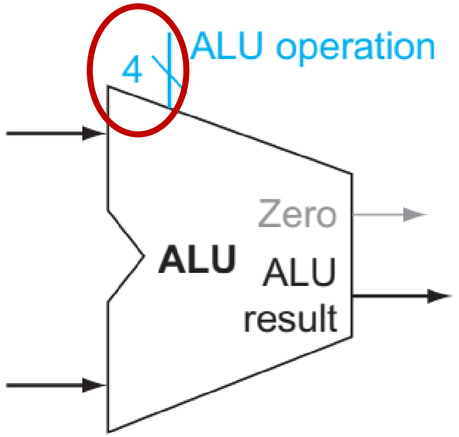
DATAPATH PER IL **EXECUTE** DI UN'ISTRUZIONE **R-TYPE**

opcode	rs	rt	rd	shamt	funct

Istruzioni come add, sub, AND, OR e slt LEGGONO da **due registri**, eseguono un'OPERAZIONE (tramite ALU) e SCRIVONO il risultato in un **terzo registro**.



a. Registers



b. ALU

addi $10, $8, 4

lw $10, 4($8)

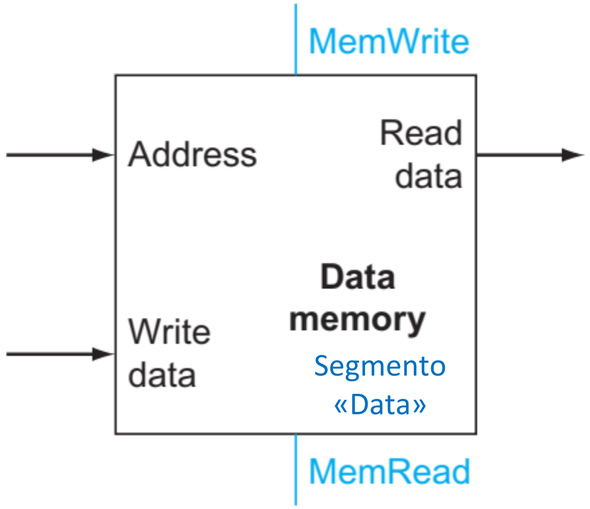
sw $10, 4($8)

bne $t1, $t2, offset

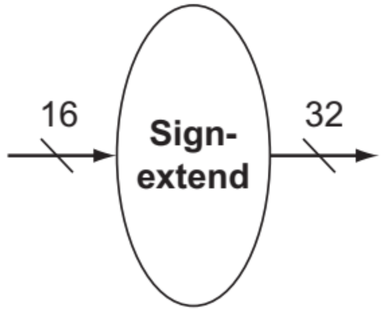
DATAPATH PER IL **EXECUTE** DI UN'ISTRUZIONE **I-TYPE**

opcode 6	rs 5	rt 5	Immediate (offset) 16

Calcolano un indirizzo di memoria sommando il **registro base** con il campo **offset con segno a 16 bit** contenuto nell'istruzione.



a. Data memory unit



b. Sign extension unit

Memoria RAM è indirizzata con 32 bit

La ALU lavora a 32 bit

Store, il valore da memorizzare deve essere letto dal Register File (nel registro indicato dall'istruzione).
Load, il valore letto dalla memoria deve essere scritto nel Register File nel registro specificato dall'istruzione.

DATAPATH PER IL **EXECUTE** DI UN'ISTRUZIONE **I-TYPE**

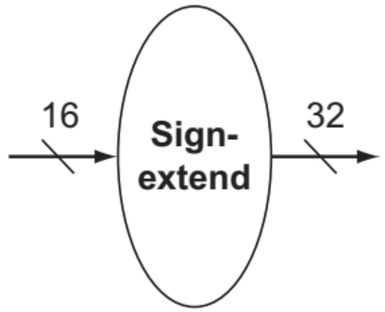
`lw $10, 4($8)`

`sw $10, 4($8)`

`addi $10, $8, 4`

L'offset è un numero di byte in **CA2 (con segno) a 16 bit. Va sommato al registro base (32 bit) per ottenere l'indirizzo in memoria.**

L'intero in CA2 (positivo o negativo) viene **sign-extended a 32 bit** prima della somma.

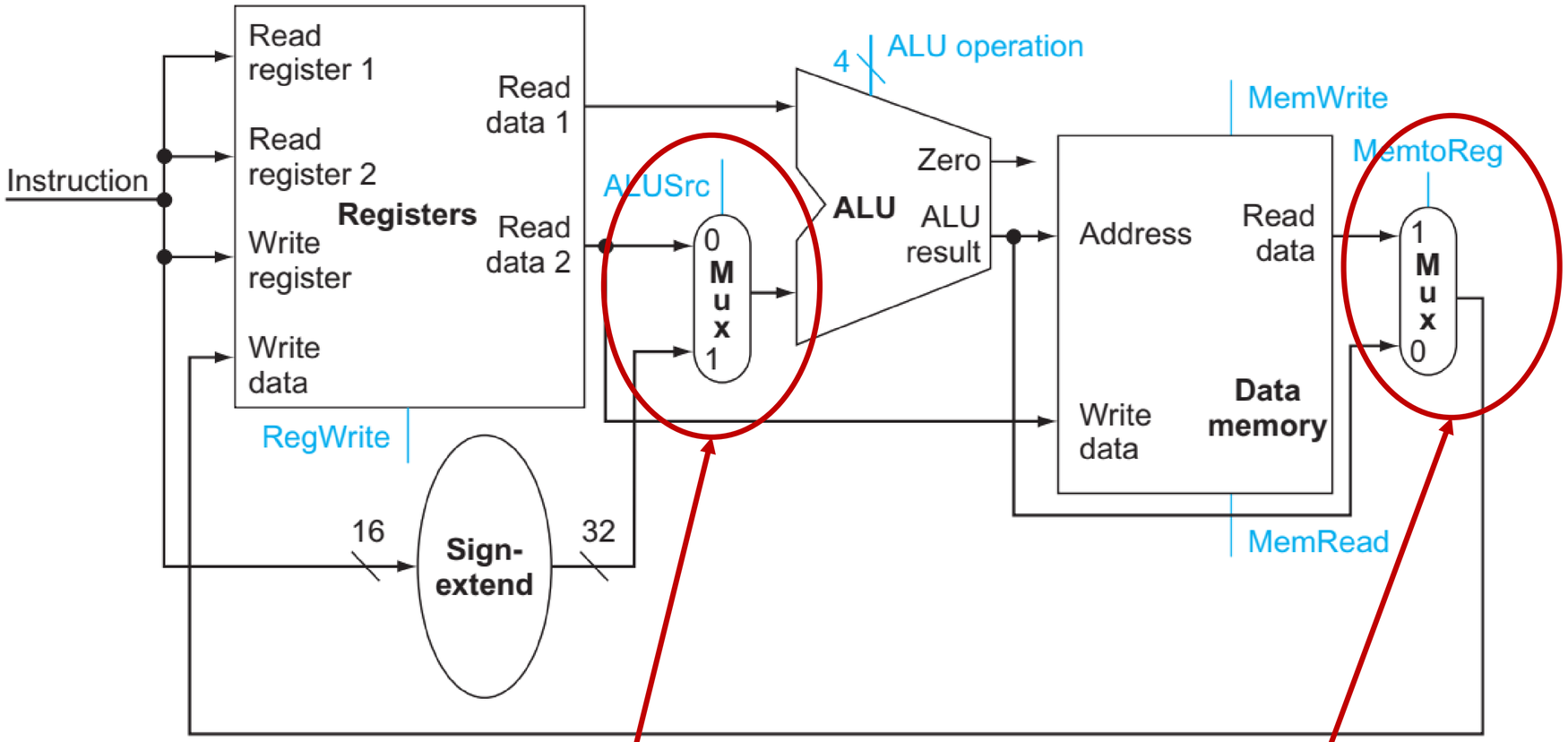


b. Sign extension unit

Memoria RAM è indirizzata con 32 bit

La ALU lavora a 32 bit

DATAPATH PER ESECUZIONE DI **R-TYPE E *MEMORY-TYPE***



MUX ALUSrc
Sceglie il secondo operando della ALU:
0: dato dal registro (Read data 2) per istruzioni **R-type**
1: immediato esteso (sign-extend) per istruzioni **lw/sw**

MUX MemtoReg
Sceglie cosa scrivere nel registro:
0: risultato ALU se istruzioni **R-type**
1: dato letto da memoria se istruzione **lw**

bne $t1, $t2, offset

DATAPATH PER IL EXECUTE DI UN'ISTRUZIONE **BRANCH**

1. **Calcolare** l'**indirizzo di destinazione** del branch sommando al PC il campo offset dell'istruzione (offset esteso con segno)

L'offset è il numero di word di cui far saltare il PC*. Viene esteso a 32 bit (per poter fare somma nell'ALU), poi shiftato a sinistra di 2 bit (garantire allineamento alla word).

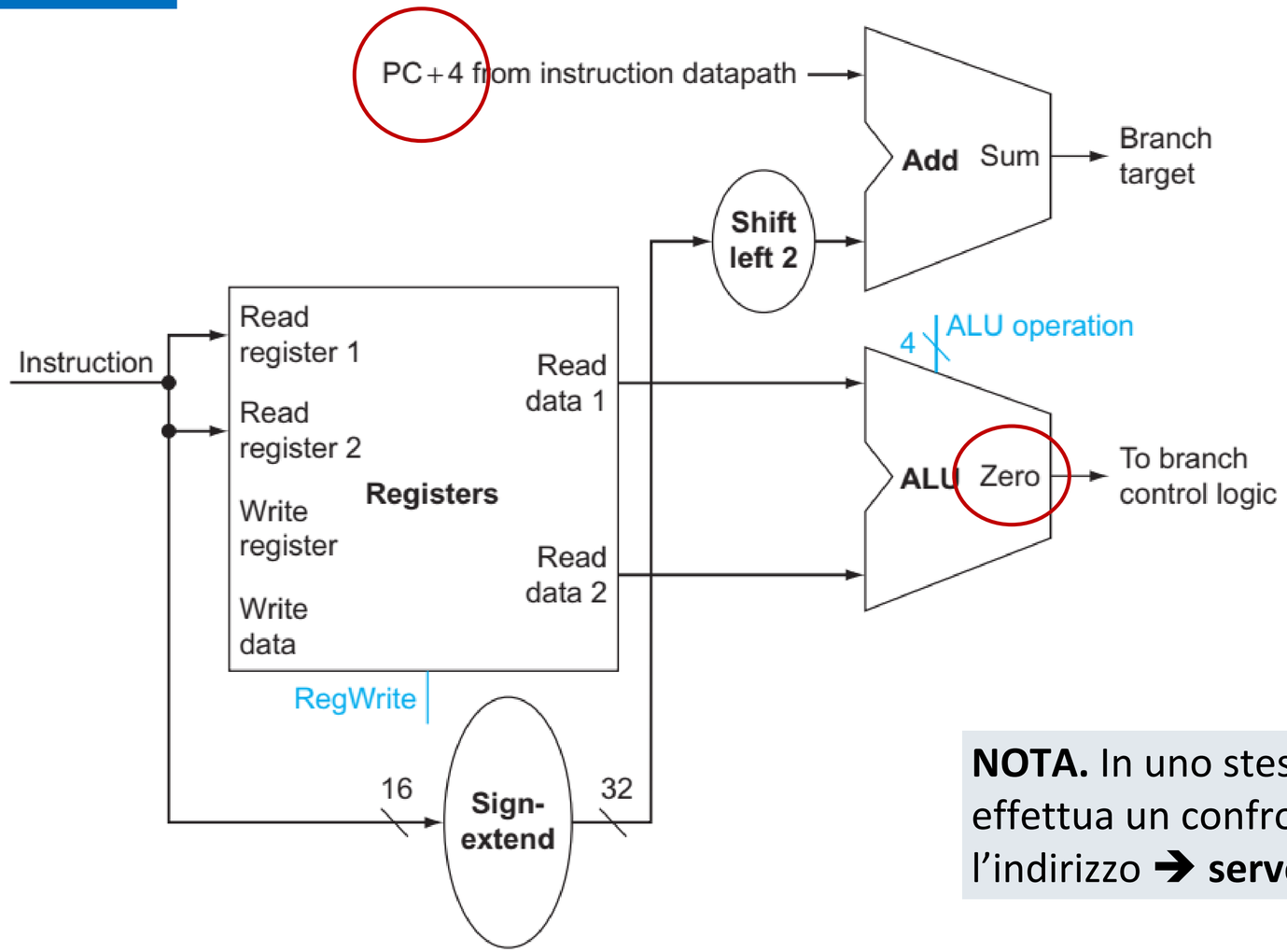
Nota. Per convenzione, per il calcolo dell'indirizzo del branch si parte dall'istruzione *successiva* al branch. Durante la fase di fetch si è già calcolato PC + 4 (indirizzo dell'istruzione successiva al branch), si usa questo **indirizzo come base** per calcolare l'indirizzo di destinazione del branch.

2. **Determinare se la prossima istruzione sia quella sequenziale oppure quella all'indirizzo di destinazione del branch**, ovvero verificare se la condizione è vera (cioè gli operandi sono uguali). Si leggono gli operandi dai due registri $t1 e $t2, si inviano alla ALU e si legge l'uscita «Zero» della ALU.

Nota. Sebbene l'uscita **Zero** segnali sempre se il risultato è 0, noi la useremo solo per implementare il test di uguaglianza dei branch

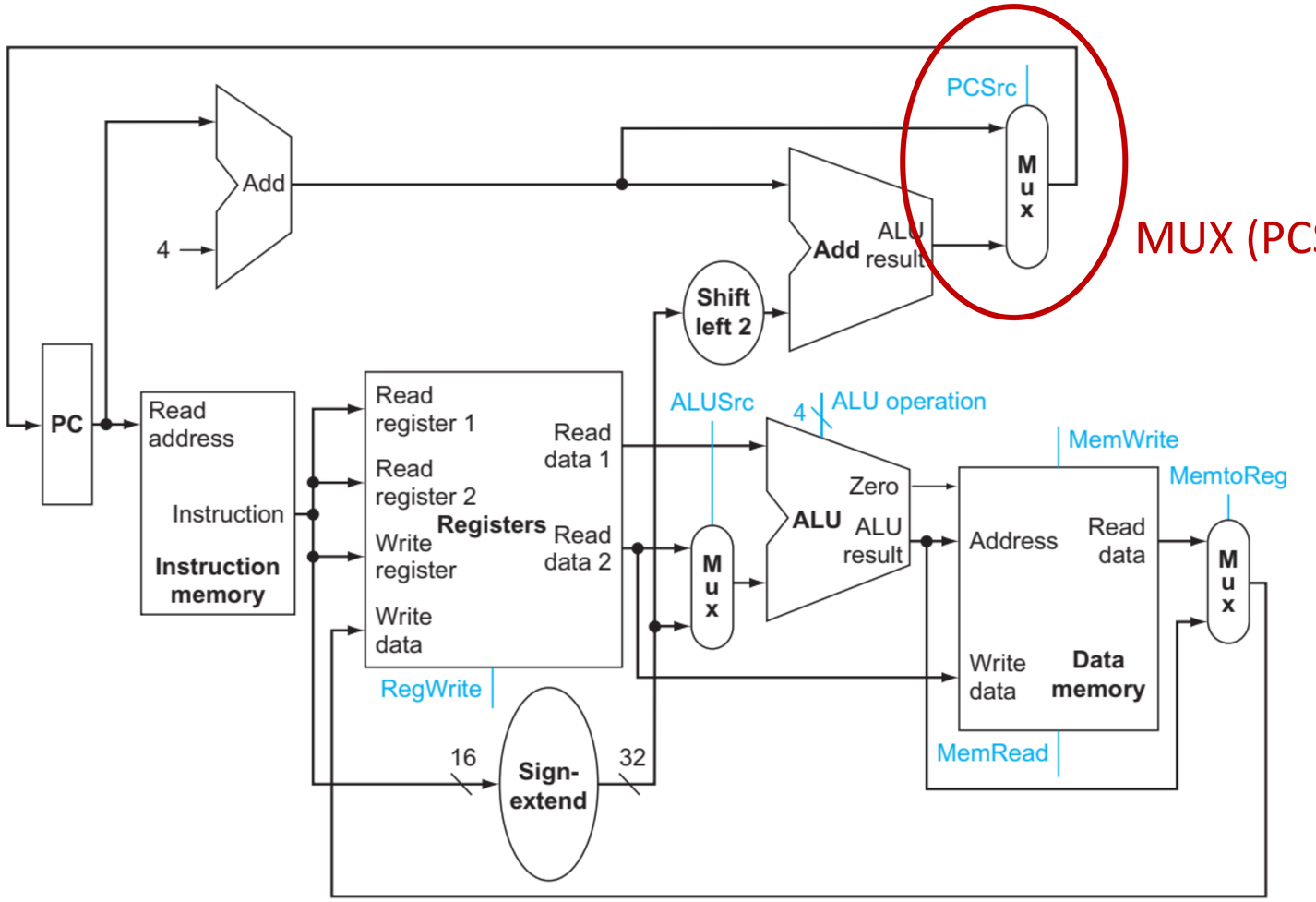
DATAPATH PER IL EXECUTE DI UN'ISTRUZIONE **BRANCH**

bne $t1, $t2, offset



NOTA. In uno stesso ciclo di clock o si effettua un confronto o si calcola l'indirizzo ➔ **serve una seconda ALU**.

DATAPATH PER ESECUZIONE DI R-TYPE, MEMORY-TYPE **E BRANCH**



MUX (PCSrc) sceglie il prossimo PC

FIGURE 4.11 The simple datapath for the core MIPS architecture combines the elements required by different instruction classes. The components come from Figures 4.6, 4.9, and 4.10. This datapath can execute the basic instructions (load-store word, ALU operations, and branches) in a single clock cycle. Just one additional multiplexor is needed to integrate branches. The support for jumps will be added later.

SONDAGGIO JIGSAW DI DOMANI

UNIVERSITÀ DEGLI STUDI DI TRIESTE

E PER ISTRUZIONI J-TYPE?

Il datapath precedente non supporta j-type perché manca:

- concatenazione (PC[31:28] + target<<2)

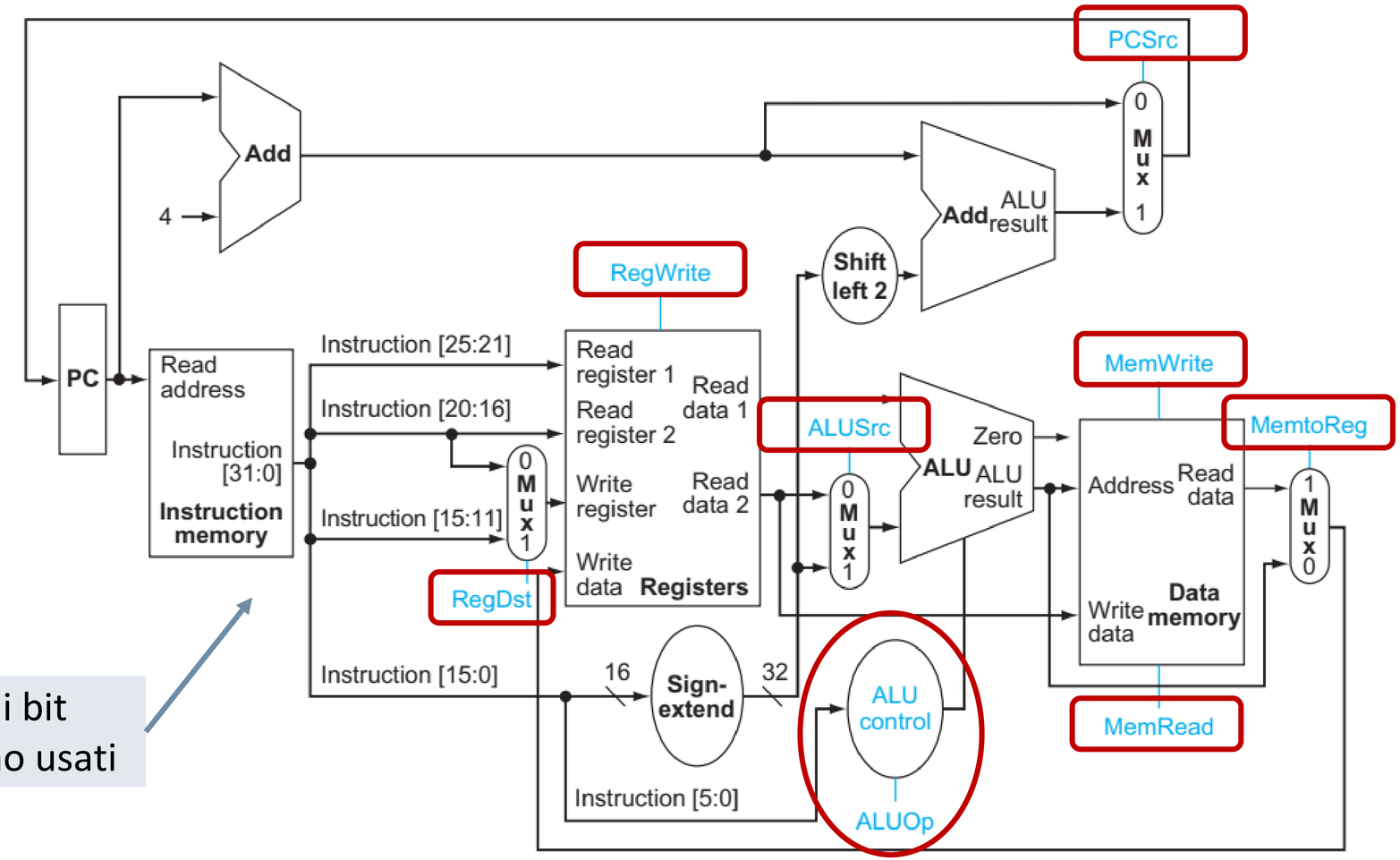
- ingresso aggiuntivo al MUX del PC

L'istruzione di *jump* opera:

- uno shift di 2 bit a sinistra i 26 bit meno significativi dell'istruzione (ovvero *target address*). Questo shift si ottiene semplicemente **concatenando 00 al *target address***.
- **sostituzione dei 28 bit meno significativi del PC** con i 26 bit del *target address*

Sarà aggiunto.. *(to be continued)*

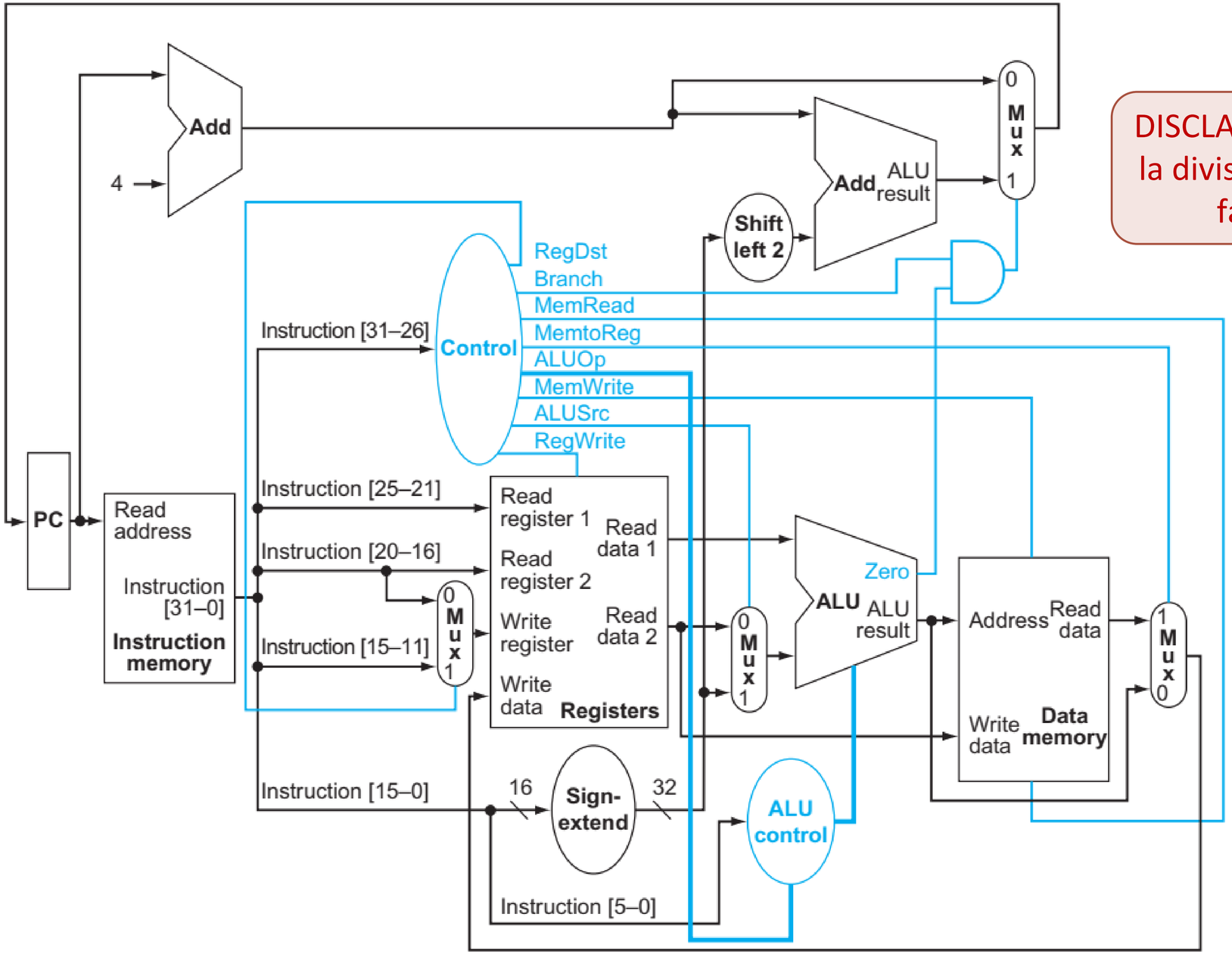
CONTROLLO DEL DATAPATH



Dettaglio di come i bit dell'istruzione sono usati

FIGURE 4.15 The datapath of Figure 4.11 with all necessary multiplexors and all control lines identified. The control lines are shown in color. The ALU control block has also been added. The PC does not require a write control, since it is written once at the end of every clock cycle; the branch control logic determines whether it is written with the incremented PC or the branch target address.

DATAPATH SINGOLO-CICLO *(NO FSM)*



DISCLAIMER: non c'è ancora la divisione temporale delle fasi (la vedremo)

AGENDA DI OGGI

- *Datapath*
- **Programmable logical array (PLA)**
- *Controllo del datapth: requisiti*
- *Finite State Machine (FSM)*
- *Control Unit (CU)*

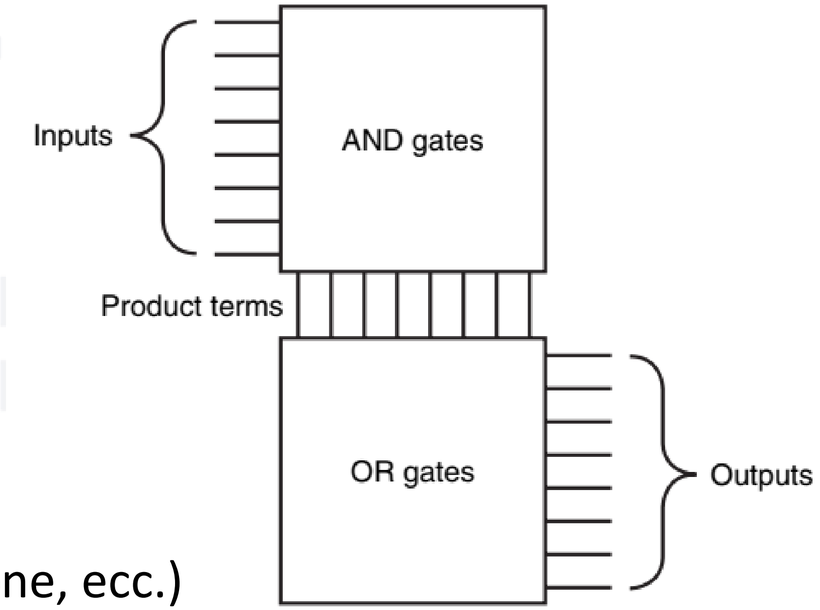
LOGICHE A DUE LIVELLI E PLA

Qualunque funzione logica può essere costruita usando porte AND, OR e NOT.

Logiche a due livelli:

- **Somma di prodotti**: somma logica (OR) di prodotti (AND) → T. mintermini
- **Prodotto di somme**: prodotto (AND) di somme (OR) → T. maxtermini

Qualunque set di funzioni logiche può essere costruita usando logica a due livelli.



Esempi di uso di circuiti in PLA:

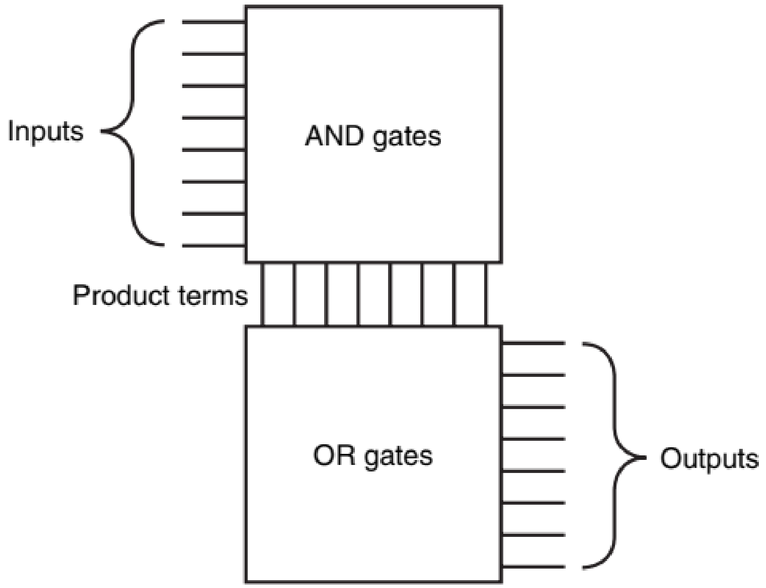
- Circuiti di controllo
- Decoder personalizzati
- Più segnali di uscita da input comuni (es. segnali di stato, di selezione, ecc.)

PROGRAMMABLE LOGICAL ARRAY (PLA)

Un **PLA** è un circuito logico programmabile che implementa **una o più funzioni booleane** nella forma somma di prodotti:

$$F = P_1 + P_2 + \cdots + P_n$$

dove ciascun P_i è un prodotto logico (AND) di input o dei loro complementi (variabile negata).



- Un insieme di **input**
- I corrispondenti input complementati (mediante inverter) per poter gestire più uscite

- Una **logica** a due stage:
  Primo stage: un array di porte logiche AND (prodotto)
  Secondo stage: un array di porte logiche OR (somma)

Output: una stessa base di prodotti logici (AND) può essere combinata in modi diversi per produrre diverse funzioni logiche. Ogni uscita può combinare (OR) i prodotti in modo diverso.

PLA: ESEMPIO CON 3 INGRESSI

Consideriamo la seguente tabella di verità.

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

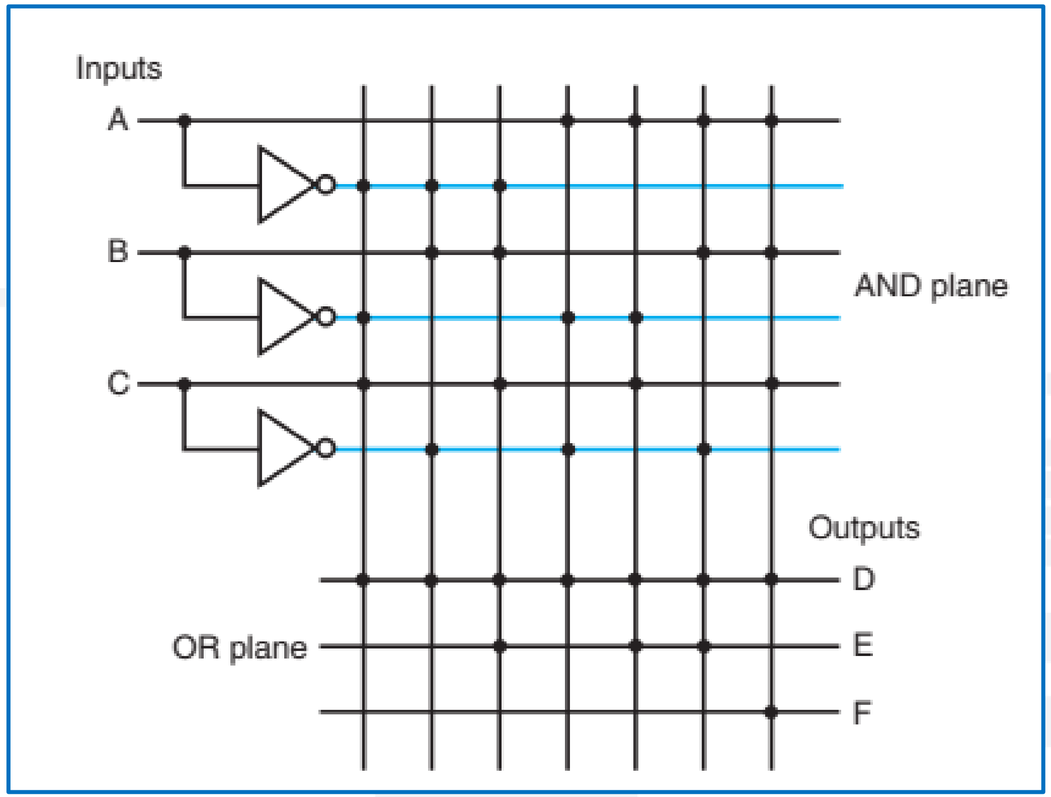
Costruire il PLA corrispondente come somme di prodotto

Come si fa?

Si costruisce la SOP per ogni colonna. Si tratta ogni colonna come una funzione logica indipendente.

PLA: ESEMPIO CON 3 INGRESSI

Inputs			
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Ogni elemento nella matrice di porte AND è un termine prodotto, costituito da **un qualsiasi numero di ingressi o ingressi negati**. Ogni elemento nella matrice di porte OR è un termine somma, costituito da un qualsiasi numero di questi termini prodotto.

Nota. I pallini neri indicano quali termini prodotto e quali somme uso

Esempio: per la F mi serve solo 1 prodotto (1 pallino nel piano OR) di tutte le variabili dirette

Patterson & Hennessy (B-16)

PLA: ESEMPIO CON 3 INGRESSI

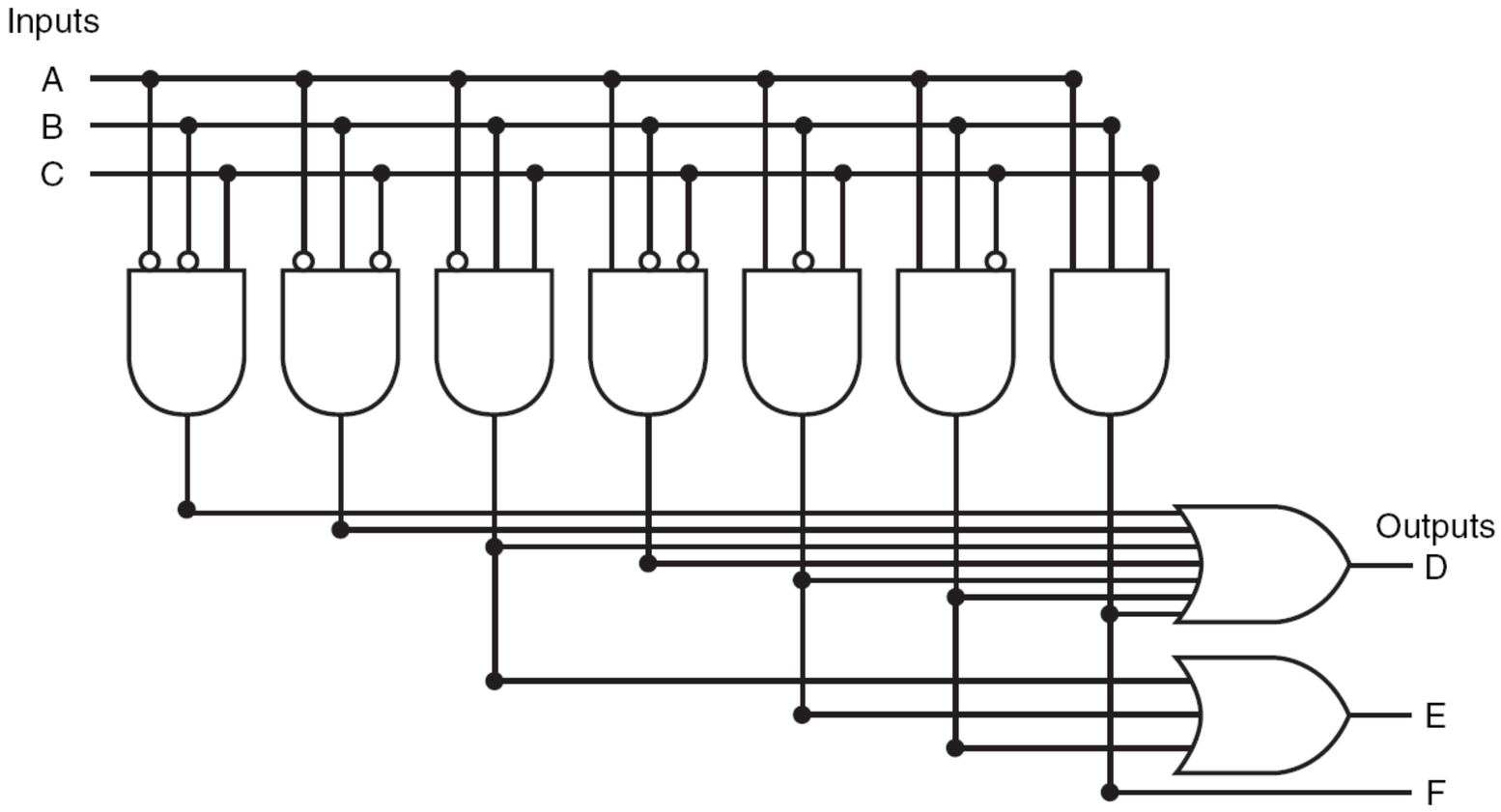


FIGURE C.3.4 The PLA for implementing the logic function described in the example.

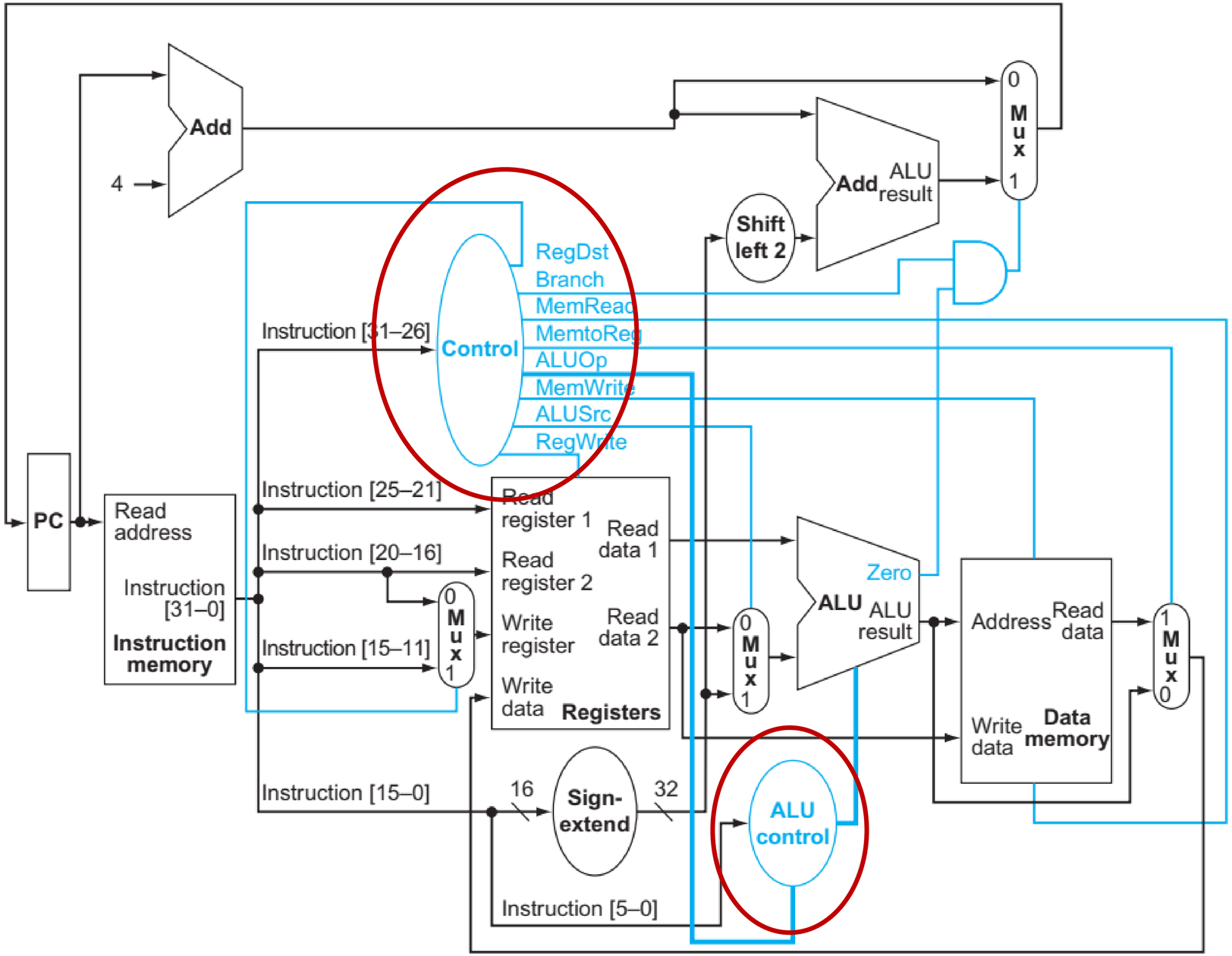
AGENDA DI OGGI E GIOVEDÌ

- *Datapath*
- *Programmable logical array (PLA)*
- **Controllo del datapth: requisiti**
- *Finite State Machine (FSM)*
- *Control Unit (CU)*

Cosa c'entra questo con la nostra architettura MIPS32?

PLA permette la realizzazione della Control Unit
e del circuito che controlla la ALU

DATAPATH SINGOLO-CICLO *(NO FSM)*



UNIVERSITÀ DEGLI STUDI DI TRIESTE

SEGNALI DI CONTROLLO (1/2)

La Control Unit riceve in input l'opcode e lo *stato del sistema* e come output restituisce i segnali in tabella:

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Patterson & Hennessy (p.264)

SEGNALI DI CONTROLLO (2/2)

ALU control determina i segnali di controllo per la ALU *che effettua l'operazione corrispondente* (add, sub, AND, slt, …).

Input alla ALU Control:

1. **ALUOp** viene dalla control unit (a seconda del tipo di istruzione, opcode)
2. **funct field** usato come input solo per istruzioni R-type

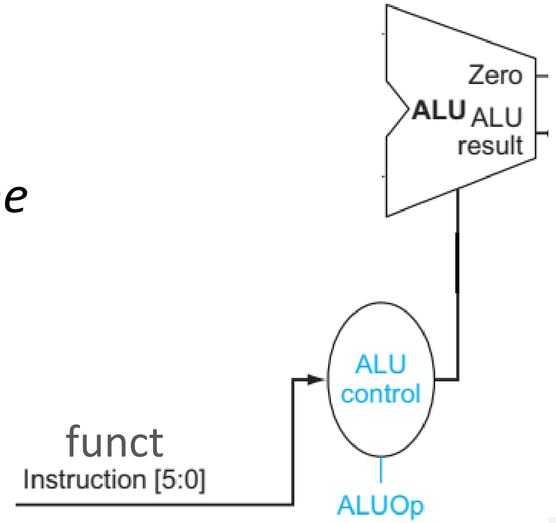


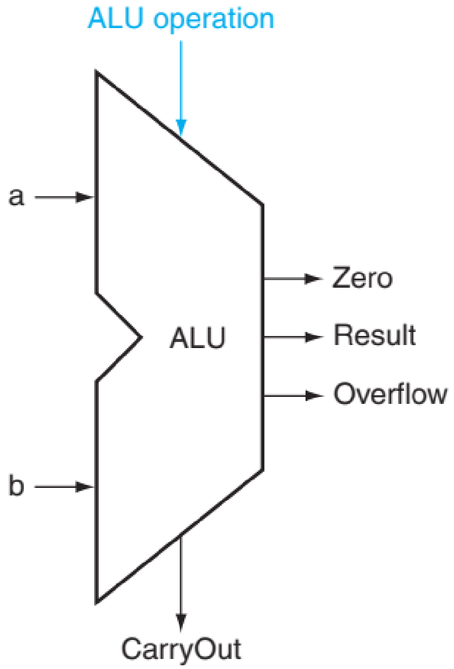
Tabella di verità dell'unità di controllo della ALU.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Output!

SEGNALI DI CONTROLLO (2/2)

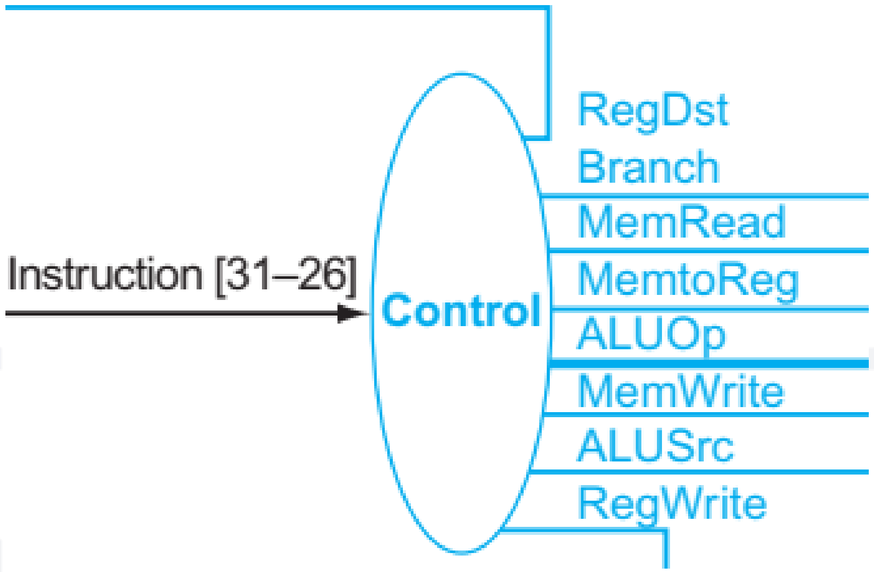
L'operazione effettivamente eseguita dalla ALU viene determinata dalla **ALU Control** che riceve input ***ALUOp*** (dalla Control Unit principale) e i ***6 bit di funct*** (dall'IR).



Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Patterson & Hennessy (B-36)

COME SI REALIZZA L'UNITÀ DI CONTROLLO PRINCIPALE?



***Spoiler**: non è completamente combinatoria. Abbiamo bisogno anche di logica sequenziale e del concetto di Finite State Machine (FSM).*

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Datapath (JIGSAW)

Prof.ssa Giulia Cisotto

giulia.cisotto@units.it

Jigsaw Method

La strategia didattica **jigsaw** è un approccio di **apprendimento collaborativo**, sviluppato per la prima volta da **Elliot Aronson** e dai suoi studenti presso l'**Università del Texas** e l'**Università della California** nei primi anni '70.
Nel jigsaw, ogni studente all'interno di un gruppo si assume la responsabilità di una parte del contenuto e poi la insegna agli altri membri del gruppo. Come i pezzi di un puzzle, gli studenti uniscono le loro singole parti per formare un insieme completo di conoscenza.

La strategia jigsaw si basa sulla **"cooperazione progettata"**, in cui nessuno studente può avere pieno successo a meno che tutti lavorino bene insieme come una squadra. Questa consapevolezza porta gli studenti a valorizzarsi reciprocamente come contributori a un compito condiviso.

La strategia jigsaw consente agli studenti di assumere ***un ruolo attivo nel proprio apprendimento*** *e favorisce la memorizzazione, il tutoring tra pari, le competenze comunicative e il recupero dei concetti* ***(Sabbah, 2016).*** *Negli studi che confrontano il metodo Jigsaw con l'insegnamento tradizionale diretto, gli studenti istruiti con il metodo Jigsaw hanno mostrato un aumento del senso di autonomia, competenza e motivazione intrinseca* ***(Hänze & Berger, 2007).***

Obbiettivo a livello «GRUPPO»: usare il datapath per eseguire l'istruzione assegnata. Poi insegnare agli altri due gruppi.

Obbiettivo a livello «CLASSE»: saper usare il datapath per eseguire tutte e tre le istruzioni.

Jigsaw Method

1. Sarete divisi in X gruppi: **ad ogni gruppo è assegnata un'istruzione – 5'**
2. **Ogni persona in ogni gruppo sceglie il proprio ruolo** nel gruppo: *Controllore*, *Macchinista*, *Operatore*. Ogni gruppo deve avere almeno una persona in ogni ruolo. **– 5'**
3. **Studio individuale**: **– 10'**
   - *Operatore*: cosa fa l'istruzione assegnata? Quali unità funzionali sono coinvolte?
   - *Controllore*: rivedere i segnali di controllo e il loro uso. Quali segnali servono in questa istruzione?
   - *Macchinista*: che tipo di istruzione è? Quali e quanti stati servono per questo tipo di istruzione?
4. **Lavoro di gruppo**: ciascun gruppo elenca le operazioni svolte dal datapath ad ogni fase, mettendo insieme le informazioni recuperate durante lo studio individuale. **- 15'**
5. **Peer-teaching**: i membri del primo gruppo vanno ad insegnare quello che hanno appreso negli altri gruppi **- 15'**
6. **Peer-teaching ROTAZIONE CON GLI ALTRI GRUPPI - 15'**
7. **Presentazione finale di ogni gruppo/commenti docente – 15'**

Total time: TBD (<2h)

Sondaggio per la presenza (i tempi verranno aggiustati in base al numero di partecipanti/gruppi)

Annunci

- Modifica calendario lezioni (effetto dalla settimana prossima)
- SEMINARI:
  - ✓ Prof. Stefano Ghidoni, Università di Padova – *Topic: TBD* – 19/05/2026 h.14-16
  - ✓ *Prof. Alessandro Raganato, Università di Milano-Bicocca (TBC)*
- **Domani** JIGSAW su datapath (**la lezione inizia alle h.16**)
- Risultati del parziale entro fine settimana

Materiale per la lezione

- Patterson & Hennessy, capitolo 4
- Patterson & Hennessy, Appendice B

*Prossima lezione: 15 aprile, **h.16**, **aula 3C***