Architettura degli Elaboratori

Esercizi d'esame sull'architettura CPU

SAPIENZA
UNIVERSITÀ DI ROMA

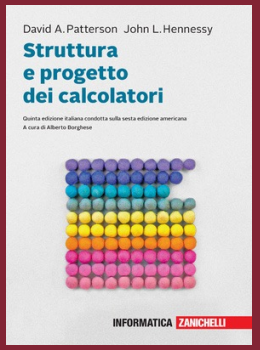
Alessandro Checco

checco@di.uniroma1.it

Special thanks and credits:

Andrea Sterbini, Iacopo Masi,
Claudio di Ciccio

[S&PdC]
4.1-4.4



Appello 2020-09-11 - Esercizio assembly

Si realizzi in assembly MIPS il seguente programma. Sia dato in memoria un array di lunghezza N, con N non minore di 2, contenente numeri in formato di 4 byte ciascuno.

Il programma assembly deve definire nella sezione dedicata al **data segment** l'array (da denominarsi `M`) e la lunghezza (da denominarsi `N`). Si assume per ipotesi che la lunghezza dell'array non sia minore di 2 come su menzionato: non occorre dunque effettuare alcun controllo per verificare la sua validità.

Nella sezione dedicata al **text segment**, il programma deve avere nel comparto **main** il caricamento dei dati da memoria, la chiamata ad una funzione `sommaContaUgualiPrec` definita di seguito e la stampa a terminale dei risultati di `sommaContaUgualiPrec`. La stampa a terminale *non deve avvenire all'interno della funzione* ma nel comparto main chiamante.

La funzione **`sommaContaUgualiPrec`** accetta come parametri di **input**:
– `$a0`: l'indirizzo base dell'array;
– `$a1`: la lunghezza dell'array;
e restituisce in **output**:
– `$v0`: la somma degli elementi di valore uguale all'elemento precedente dell'array;
– `$v1`: il conteggio degli elementi di valore uguale all'elemento precedente dell'array.

Esempi:
a) Input: `$a0` è l'indirizzo in memoria di `M` definito come di seguito, con lunghezza `$a1` che vale `N` = 5
1, 1, 4, 2, 2
Output: `$v0` vale 1 + 2 = 3; `$v1` vale 2.
b) Input: `$a0` è l'indirizzo in memoria di `M` definito come di seguito, con lunghezza `$a1` che vale `N` = 7
0, 6, 7, 8, 8, 8, 6
Output: `$v0` vale 8 + 8 = 16; `$v1` vale 2.

Note: Commentare *ogni riga di codice* avendo cura di spiegare a cosa servano i registri. Una soluzione ricorsiva sarà premiata con un bonus di 1 punto.

Esercizi

SAPIENZA
UNIVERSITÀ DI ROMA

Malfunzionamento della CU

Esercizio per casa (vj)

Esercizio d’esame progettazione istruzione

Cosa fare se la Control Unit è malfunzionante?

SAPIENZA
UNIVERSITÀ DI ROMA

Possiamo identificare guasti hardware via software?

Control Unit malfunzionante

Segnali di controllo

	Reg Dest	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
tipo R	1	0	0	1	X	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0	0
sw	X	1	X	0	X	1	0	0	0	0
beq	X	0	X	0	X	0	1	0	0	1
j	X	X	X	0	X	0	X	1	X	X

Cosa succede se la CU genera dei segnali errati?

Dobbiamo individuare:

- **quale combinazione di segnali venga generata**
- **quali istruzioni vengano influenzate** dalle nuove combinazioni e **cosa facciano**

Una volta individuate le «nuove» funzionalità (fallaci) delle istruzioni…

… possiamo cercare di scrivere un breve programma che evidenzia se la CPU sia malfunzionante o meno

Jump ← MemRead

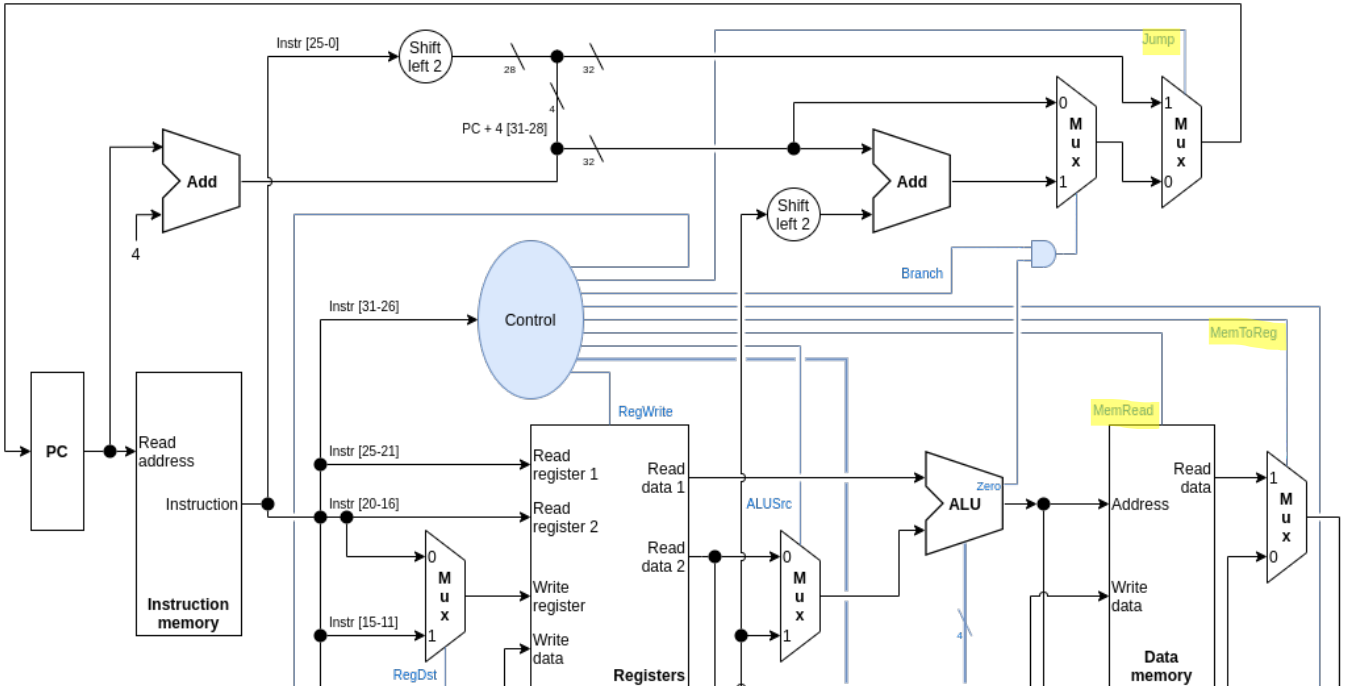
Si ha il dubbio che in alcune CPU MIPS la Control Unit sia rotta, producendo il segnale di controllo **Jump attivo se e solo se è attivo il segnale MemRead.**

Si assume che:

- MemToReg = 1 solo per la lw ed altrimenti valga 0 (mai X)
- RegDest = 1 solo per le istruzioni di tipo R ed altrimenti valga 0 (mai X)
- MemRead = 1 solo per l'istruzione lw ed altrimenti valga 0 (mai X).

a) Si indichino quali delle istruzioni funzioneranno male e perché.

b) Si scriva un breve programma assembly MIPS che termina valorizzando il registro **$s0** con il valore **1 se il processore è guasto**, altrimenti con 0.



Jump ← MemRead

	Reg Dest	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
tipo R	1	0	0	1	**0**	0	0	**0**	1	0
lw	0	1	1	1	**1**	0	0	**1**	0	0
sw	**0**	1	**0**	0	**0**	1	0	**0**	0	0
beq	**0**	0	**0**	0	**0**	0	1	**0**	0	1
j	**0**	X	**0**	0	**0**	0	X	**0**	X	X

Assunzione | Difetto

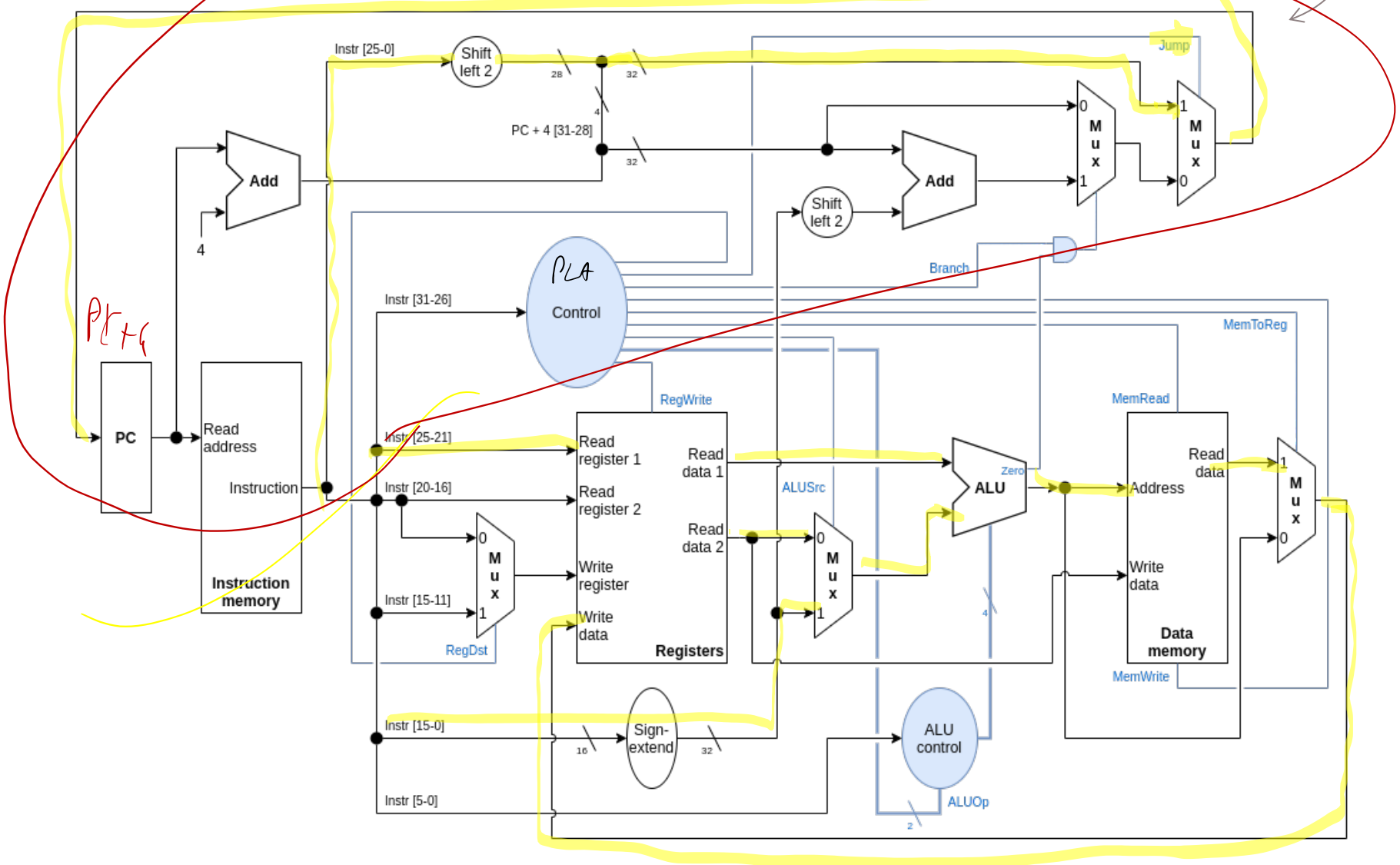
a) Si indichino quali delle istruzioni funzioneranno male e perché.

a) «**Jump attivo se e solo se è attivo il segnale MemRead»** vuol dire che
- se MemRead = 0 allora Jump = 0
- se MemRead = 1 allora Jump = 1

Quindi sono danneggiate le istruzioni che hanno i due segnali diversi
- **lw:** carica correttamente dalla memoria MA FA ANCHE UN SALTO (1, 0)
- **sw:** funziona correttamente (0, 0)
- **beq:** funziona correttamente (0, 0)
- **j:** NON SALTA (0, 0) – invece che (0, 1)
- **tipo R:** funzionano correttamente (0, 0)

Jump ← MemRead

Qui: scarabocchi



Jump ← MemRead

b) Per distinguere se la CPU sia malfunzionante dobbiamo quindi usare **j** oppure **lw**:

`lw $rt, etichetta($rs)` che salta all'indirizzo che corrisponde alla sua codifica

`j etichetta` che NON salta

Conviene usare la **j** perché è più semplice:

```
   move $s0, $zero     # $s0 = 0
   j    End            # salto senza eseguire la seguente istr.
   li   $s0, 1         # $s0 = 1 se non viene eseguito il salto
End:
```

Esercizio per casa: vj

Si vuole aggiungere alla CPU l'istruzione vectorised jump (**vj**), di tipo **I** e sintassi assembly

```
vj $indice, vettore
```

che salta all'indirizzo contenuto nell'elemento $indice-esimo del *vettore* *di word*.

Esempio: se in memoria si è definito staticamente il

```
vettore: .word 16, 24, 312, 44
```

e al registro **$t0** è assegnato il valore **3** allora

```
vj $t0, vettore
```

salterà all'indirizzo **44** (che è l'elemento con indice 3 del vettore)

a) si disegnino le modifiche necessarie a realizzare la funzione, aggiungendo tutti gli eventuali MUX, segnali di controllo, bus, ALU e sommatori (ecc) che ritenete necessari.

b) indicate i valori di tutti i segnali di controllo, in modo da eseguire l'istruzione **vj.**

c) tenendo conto che il tempo di accesso ai registri (sia in lettura che scrittura) è di **25ns**, l'accesso alla memoria impiega **75ns**, la ALU e i sommatori impiegano **100ns** e ignorando gli altri ritardi, calcolate il tempo di esecuzione minimo della istruzione **vj** e indicate se è necessario aumentare il periodo di clock della CPU per poter svolgere questa nuova istruzione.

Esercizio per casa: vj

Si vuole aggiungere alla CPU l'istruzione vectorised jump (**vj**), di tipo I e sintassi assembly

```
vj $indice, vettore
```

che salta all'indirizzo contenuto nell'elemento $indice-esimo del *vettore di word*.

Esempio: se in memoria si è definito staticamente il

```
vettore: .word 16, 24, 312, 44
```

e al registro **$t0** è assegnato il valore **3** allora

```
vj $t0, vettore
```

salterà all'indirizzo **44** (che è l'elemento con indice 3 del vettore)

a) si disegnino le modifiche necessarie a realizzare la funzione, aggiungendo tutti gli eventuali MUX, segnali di controllo, bus, ALU e sommatori (ecc) che ritenete necessari.

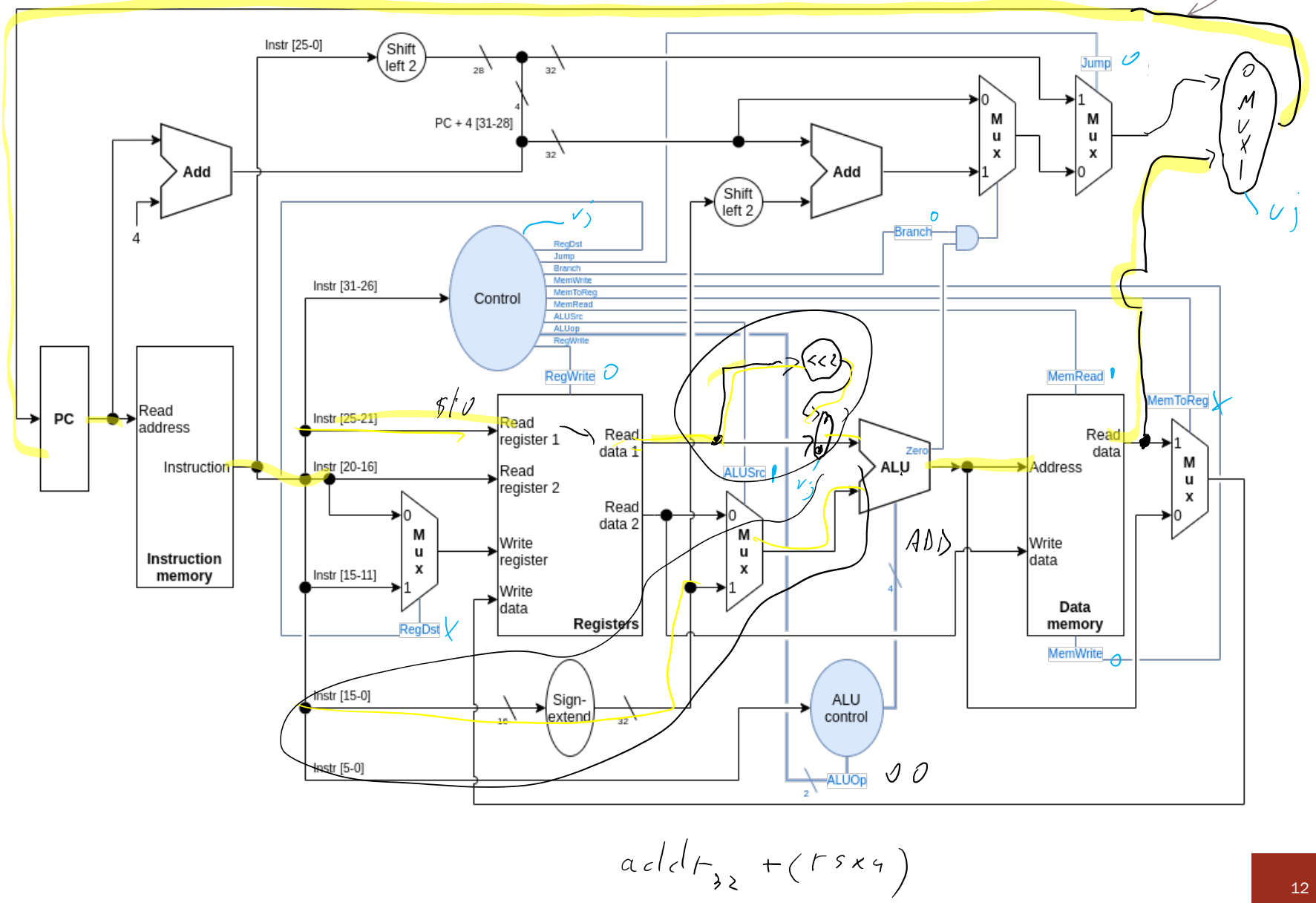
b) indicate i valori di tutti i segnali di controllo, in modo da eseguire l'istruzione **vj.**

c) tenendo conto che il tempo di accesso ai registri (sia in lettura che scrittura) è di **25ns**, l'accesso alla memoria impiega **75ns**, la ALU e i sommatori impiegano **100ns** e ignorando gli altri ritardi, calcolate il tempo di esecuzione minimo della istruzione **vj** e indicate se è necessario aumentare il periodo di clock della CPU per poter svolgere questa nuova istruzione.

flusso: fetch, reg x 4, ALU (sum with Immediate), MEM, PC

Esercizio per casa: vj (soluzione /a)

Qui: scarabocchi



vj rs, constant (soluzione /b, c)

(b) Segnali dalla Control Unit

VJ	Reg Dest	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
1	X	1	X	0	1	0	X	X	0	0

Accesso a memorie = 75ps, accesso ai registri = 25ps, ALU e sommatori = 100ps

(c) Tempo di esecuzione

Fetch 75ps	Reg[rs] 25ps	add = (rs<<2) + const. 100ps	jumpTo = Mem[add] 75ps	PC ← jumpTo 0ps
PC+4 (non utilizzato) 100ps				

Totale: 275ps

Non è necessario incrementare il periodo di clock

Istruzione	Instr. Fetch	Instr. Decode	Execution	MEM	Wr. Back	Totale
lw	75	25	100	75	25	300

Esercizio per casa: modificare vj in vrj (salvando il $pc corrente in $ra prima del salto)

vj rs, constant (soluzione /b, c)

(b) Segnali dalla Control Unit										
VJ	Reg Dest	ALU Src	Mem toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
1	X	1	X	0	1	0	X	X	0	0

Accesso a memorie = 75ps, accesso ai registri = 25ps, ALU e sommatori = 100ps

(c) Tempo di esecuzione				
Fetch 75ps	Reg[rs] 25ps	add = rs<<2 + const. 100ps	jumpTo = Mem[add] 75ps	PC ← jumpTo 0ps
PC+4 (non utilizzato) 100ps				

Totale: 275ps

Non è necessario incrementare il periodo di clock

Istruzione	Instr. Fetch	Instr. Decode	Execution	MEM	Wr. Back	Totale
lw	75	25	100	75	25	300

Si svolga nuovamente l'ex se il vettore in memoria dati è ora
vettore: .half 15, 24, 313, 42

Esercizio d'esame (8 punti)

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere l'istruzione di tipo I
`diff_and_store $rs, $rt, addr`
(calcola la differenza in valore assoluto dei registri rs ed rt e salva il risultato in memoria, all'indirizzo addr)
che equivale al seguente frammento di codice

```
          sub $rs, $rs, $rt          # (a)
          bge $rs, $0, save          # (b)
          sub $rs, $0, $rs           # (c)
    save: sw  $rs, addr              # (d)
```

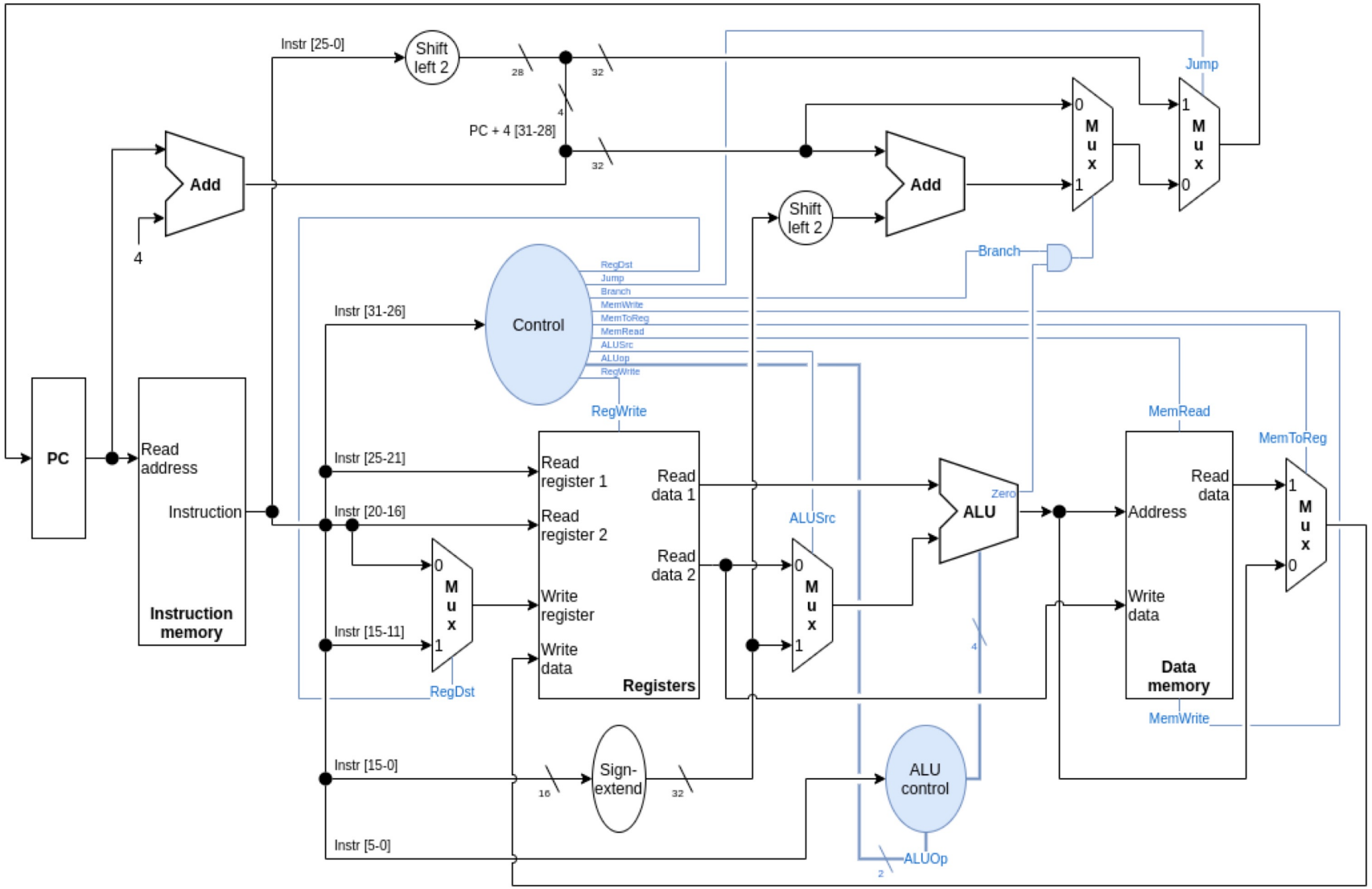
ossia (a) esegue la sottrazione tra `rs` ed rt, (b,c) se il risultato è negativo ne cambia il segno, infine (d) salva il valore risultante in memoria all'indirizzo indicato da `addr`. In particolare, `addr` è la parte immediata dell'istruzione e viene usato come indirizzo *assoluto* di destinazione.

1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.

2) Indicare il contenuto in bit della word che esprime l'istruzione
`diff_and_store $10, $11, 64`
compilando la tabella sottostante (assumiamo che lo *OpCode* di diff_and_store sia 0x38).

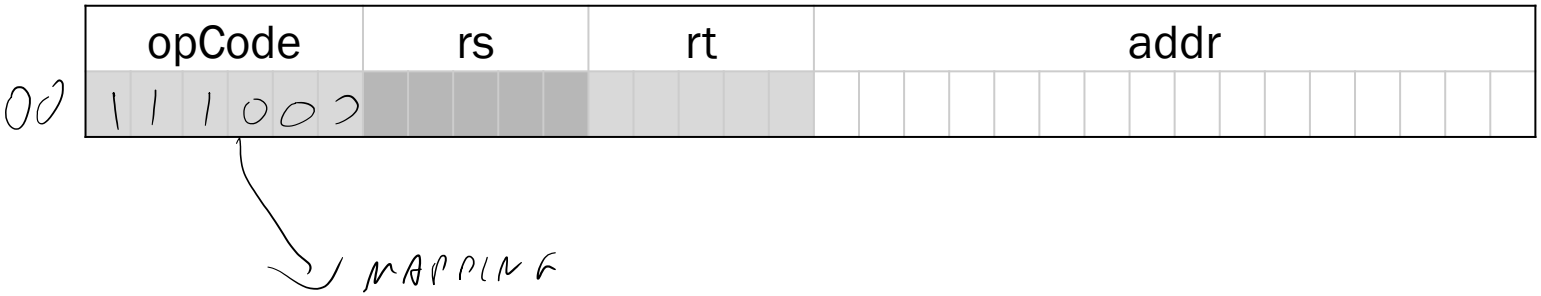
3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 100 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati*.

Esercizio d'esame (8 punti) **matita e gomma!**



Soluzione 2)

2) Indicare il contenuto in bit della word che esprime l'istruzione
`diff_and_store $10, $11, 64`
compilando la tabella sottostante (assumiamo che lo *OpCode* di diff_and_store sia 0x38).



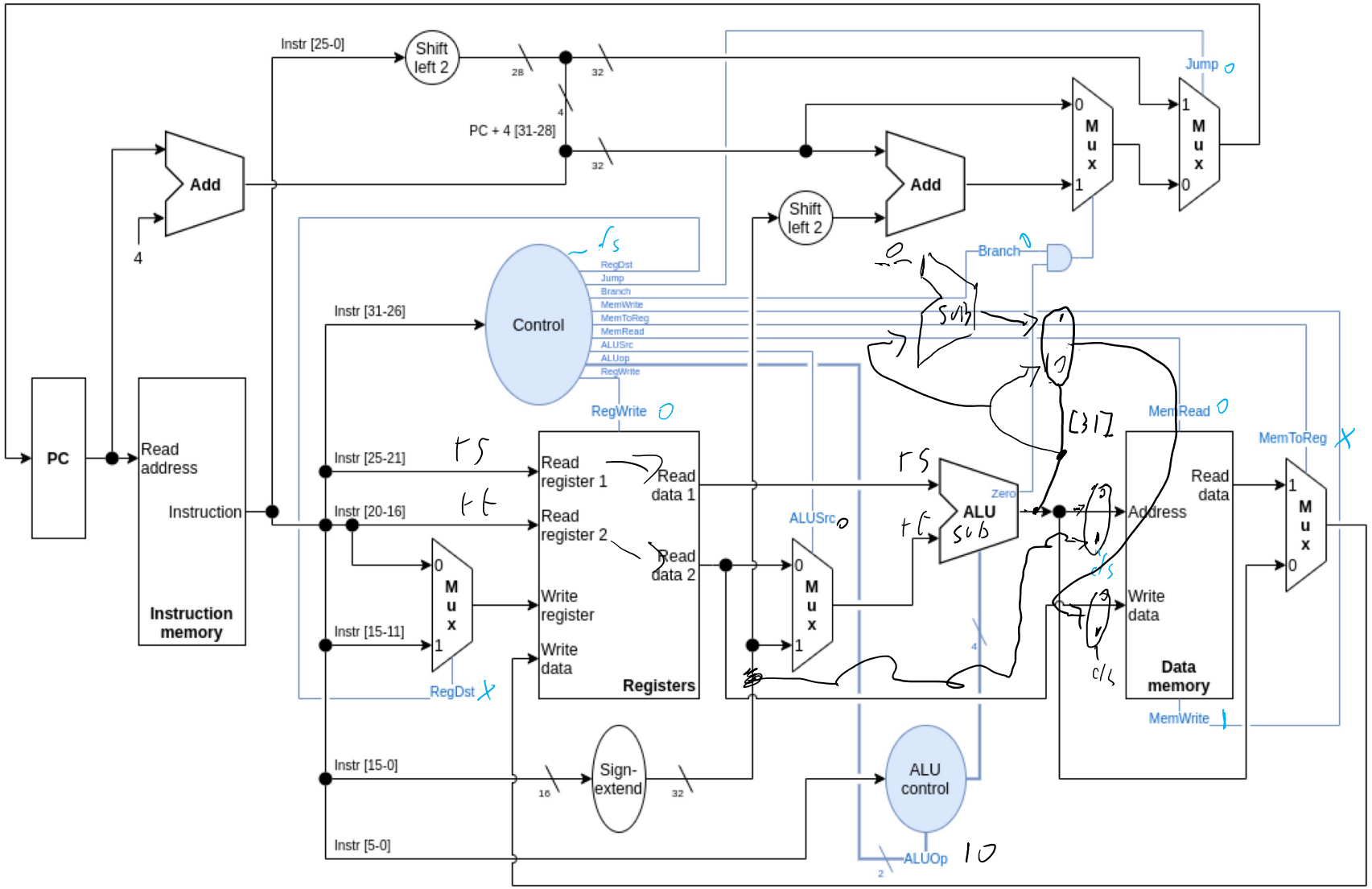
Soluzione 2)

2) Indicare il contenuto in bit della word che esprime l'istruzione
`diff_and_store $10, $11, 64`
compilando la tabella sottostante (assumiamo che lo *OpCode* di diff_and_store sia 0x38).

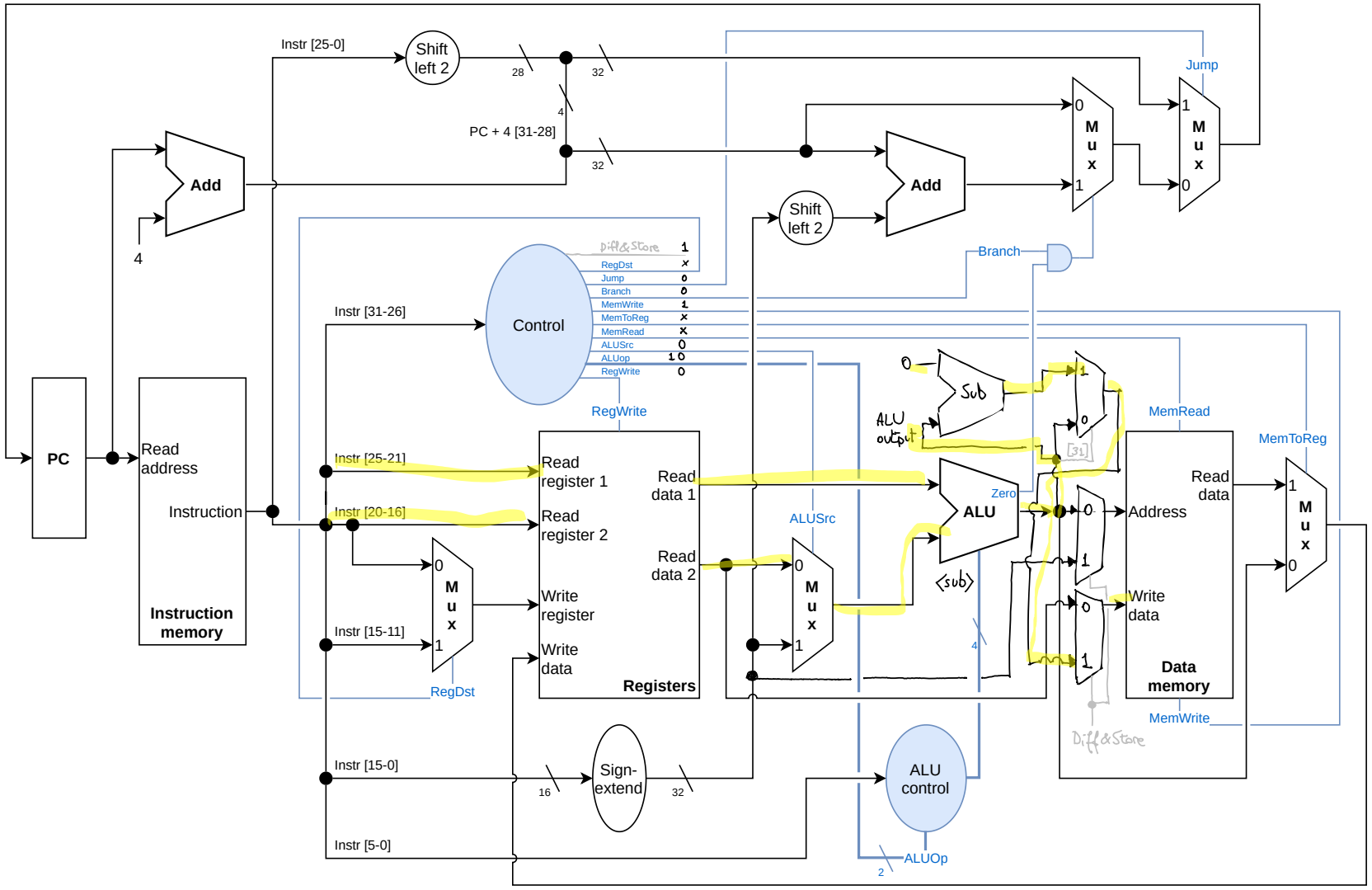
opCode						rs					rt				addr															
1	1	1	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Soluzione 1-v1) 0 - ALU if 31-th bit

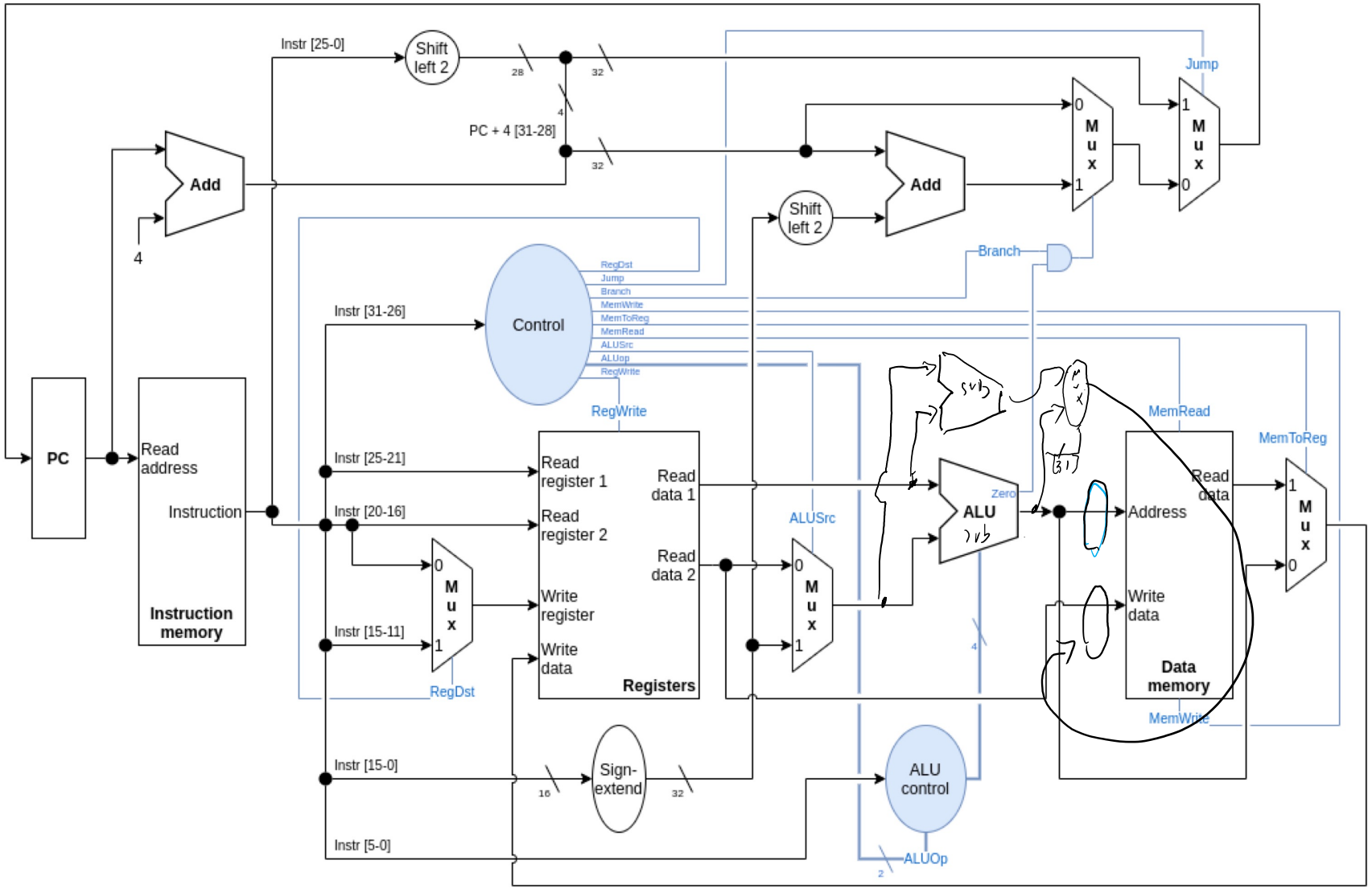
ds $rs, $rt, addr



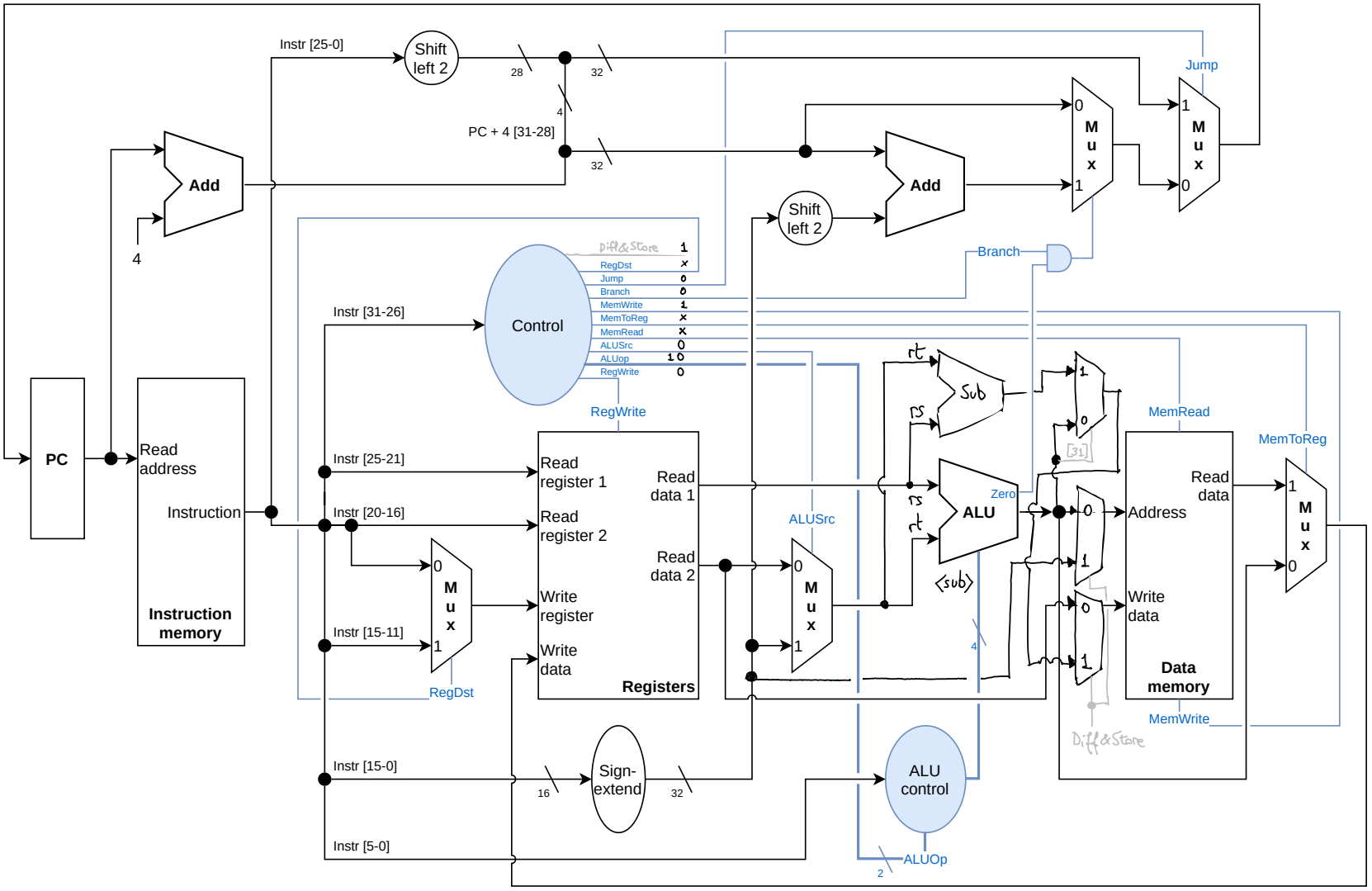
Soluzione 1-v1) 0 - ALU if 31-th bit



Soluzione 1-v2) switch registers if 31-th bit



Soluzione 1-v2) switch registers if 31-th bit



Soluzione 3)

Accesso a memorie = 200ps, accesso ai registri = 50ps, ALU e sommatori = 100ps

v1				
Fetch 200ps	Reg[rs] 50ps	ALU: rs - rt 100ps	SUB: 0 – (rs-rt) 100ps	Mem 200ps
PC+4 100ps	PC ← jumpTo 0ps			
Totale: 650ps				

v2			
Fetch 200ps	Reg[rs] 50ps	ALU: rs - rt 100ps SUB: rt - rs 100ps	Mem 200ps
PC+4 100ps	PC ← jumpTo 0ps		
Totale: 550ps			

Istruzione	Instr. Fetch	Instr. Decode	Execution	MEM	Wr. Back	Totale
lw	200	50	100	200	50	600