SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Architettura degli Elaboratori

Lez. 10 – Esercizi su CPU MIPS a 1 ciclo di clock

Prof. Andrea Sterbini – sterbini@di.uniroma1.it

Argomenti

- **Argomenti della lezione**
- Esercizi sulla CPU MIPS a 1 colpo di clock
  - Segnali di controllo errati:
    - cosa succede
    - come individuarli
  - Aggiungere altre istruzioni
- Soluzione esercizio per casa

CU rotta

- Abbiamo visto che i segnali di controllo corretti delle istruzioni sono

	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

CU rotta

- Abbiamo visto che i segnali di controllo corretti delle istruzioni sono

	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 sballati?
- Dobbiamo:
- Capire **quale combinazione di segnali viene generata**
- Capire **quali istruzioni vengono influenzate** dalle nuove combinazioni e **cosa fanno**
- Una volta individuate le «nuove» funzionalità delle istruzioni possiamo cercare di scrivere un breve programma che evidenzia se la CPU è rotta o no

RegWrite = Branch

Supponiamo che il segnale **RegWrite** sia **sempre uguale** al segnale **Branch**

Ovvero che **se Branch = 0 allora RegWrite = 0** e **se Branch = 1 allora RegWrite = 1**

Assumiamo che MemToReg = 1 solo per la lw e che RegDest = 1 solo per le tipo R

1) Individuate quali istruzioni ne sono affette e perché
2) Scrivete un breve programma che distingue se la CPU è rotta (mettendo in $s0 il valore 0) oppure se è funzionante ($s0=1)

RegWrite = Branch

Supponiamo che il segnale **RegWrite** sia **sempre uguale** al segnale **Branch**

Ovvero che **se Branch = 0 allora RegWrite = 0** e **se Branch = 1 allora RegWrite = 1**

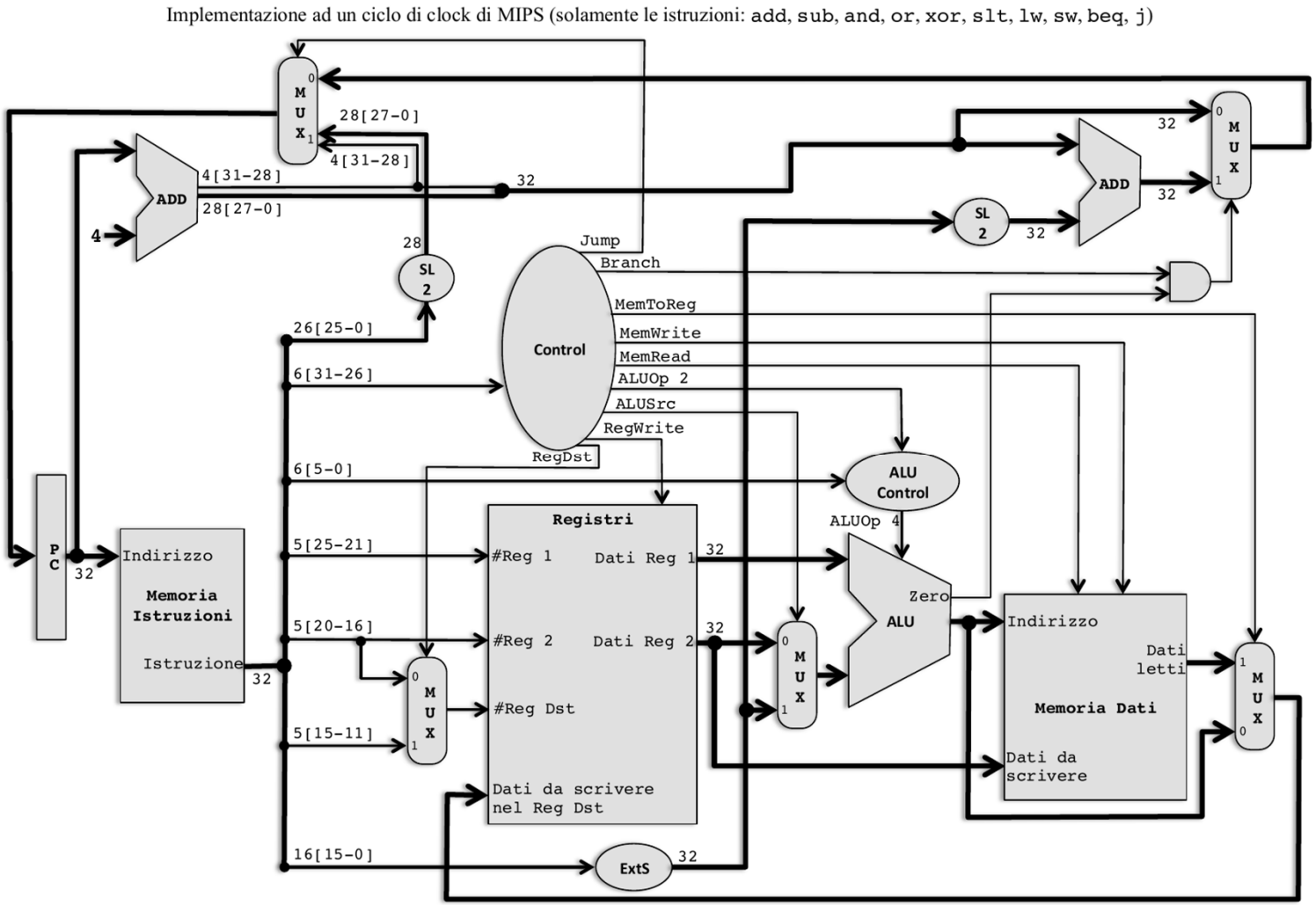
Assumiamo che MemToReg = 1 solo per la lw e che RegDest = 1 solo per le tipo R

1) Individuate quali istruzioni ne sono affette e perché
2) Scrivete un breve programma che distingue se la CPU è rotta (mettendo in $s0 il valore 0) oppure se è funzionante ($s0=1)

1) Le istruzioni affette da questo guasto sono:

- Tutte le istruzioni che modificano un registro (**tipo R e lw**) che NON memorizzeranno il risultato nel registro, lasciandolo invariato
- **Branch**: che oltre ad effettuare il salto modificherà uno dei registri perché RegWrite=1
  - Il registro che viene modificato è il registro **rt** perché **RegDest=0** (per assunzione)
  - Il valore inserito è il risultato della differenza usata per confrontare i due operandi (**MemtoReg=0** per assunzione)
- Le istruzioni **sw** e **jump**, invece, funzioneranno correttamente

RegWrite = Branch



Individuare il guasto

- 2) Per individuare se la CPU è rotta cerchiamo di inventare un programma che lascia il valore **0** nel registro **$s0** se è rotta e **1** se funziona correttamente, tenendo conto che:
  - Non possiamo caricare un valore immediato in un registro perché RegWrite = 0

Individuare il guasto

- 2) Per individuare se la CPU è rotta cerchiamo di inventare un programma che lascia il valore **0** nel registro **$s0** se è rotta e **1** se funziona correttamente, tenendo conto che:
  - Non possiamo caricare un valore immediato in un registro perché RegWrite = 0

Potremmo assumere che in memoria sia presente il valore 1 e provare a leggerlo

```
.data                          .text
uno:    .word  1               main  lw $s0, uno
```

Individuare il guasto

- 2) Per individuare se la CPU è rotta cerchiamo di inventare un programma che lascia il valore **0** nel registro **$s0** se è rotta e **1** se funziona correttamente, tenendo conto che:
  - Non possiamo caricare un valore immediato in un registro perché RegWrite = 0

Potremmo assumere che in memoria sia presente il valore 1 e provare a leggerlo

```
.data                          .text
uno:    .word  1               main  lw $s0, uno
```

Oppure basta una qualsiasi istruzione che generi un valore diverso da 0 (una **tipo R** o una **li**), come una delle seguenti

```
    nor  $s0, $zero, $zero  # nega 0 e produce 11..11 = -1
    li   $s0, 1             # = 1
    addi $s0, $zero, 1      # = 1
```

Individuare il guasto

- 2) Per individuare se la CPU è rotta cerchiamo di inventare un programma che lascia il valore **0** nel registro **$s0** se è rotta e **1** se funziona correttamente, tenendo conto che:
  - Non possiamo caricare un valore immediato in un registro perché RegWrite = 0

Potremmo assumere che in memoria sia presente il valore 1 e provare a leggerlo

```
.data                          .text
uno:   .word  1                main  lw $s0, uno
```

Oppure basta una qualsiasi istruzione che generi un valore diverso da 0 (una **tipo R** o una **li**), come una delle seguenti

```
    nor  $s0, $zero, $zero  # nega 0 e produce 11..11 = -1
    li   $s0, 1             # = 1
    addi $s0, $zero, 1      # = 1
```

Oppure potremmo costruire una **beq** che mette 0 in **$s0** se la CPU è rotta, calcolando la differenza tra due valori uguali ($s0 e se stesso). Però per distinguere il caso di CPU rotta da quello con CPU funzionante dobbiamo assumere che inizialmente $s0=1

```
.text                          # assumo che $s0=1
main:  beq $s0, $s0, uno
```

MemWrite = not RegWrite

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

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.
Si assume che RegDst sia asserito solo per le istruzioni di tipo R
e che MemtoReg sia asserito solo per l'istruzione lw.

MemWrite = not RegWrite

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

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.
Si assume che RegDst sia asserito solo per le istruzioni di tipo R
e che MemtoReg sia asserito solo per l'istruzione lw.

	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	0	1	0	**0**	X	**1**	0	0	0	0
beq	0	0	0	**0**	X	**0**	1	0	0	1
j	0	X	0	**0**	X	**0**	X	1	X	X

MemWrite = not RegWrite

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

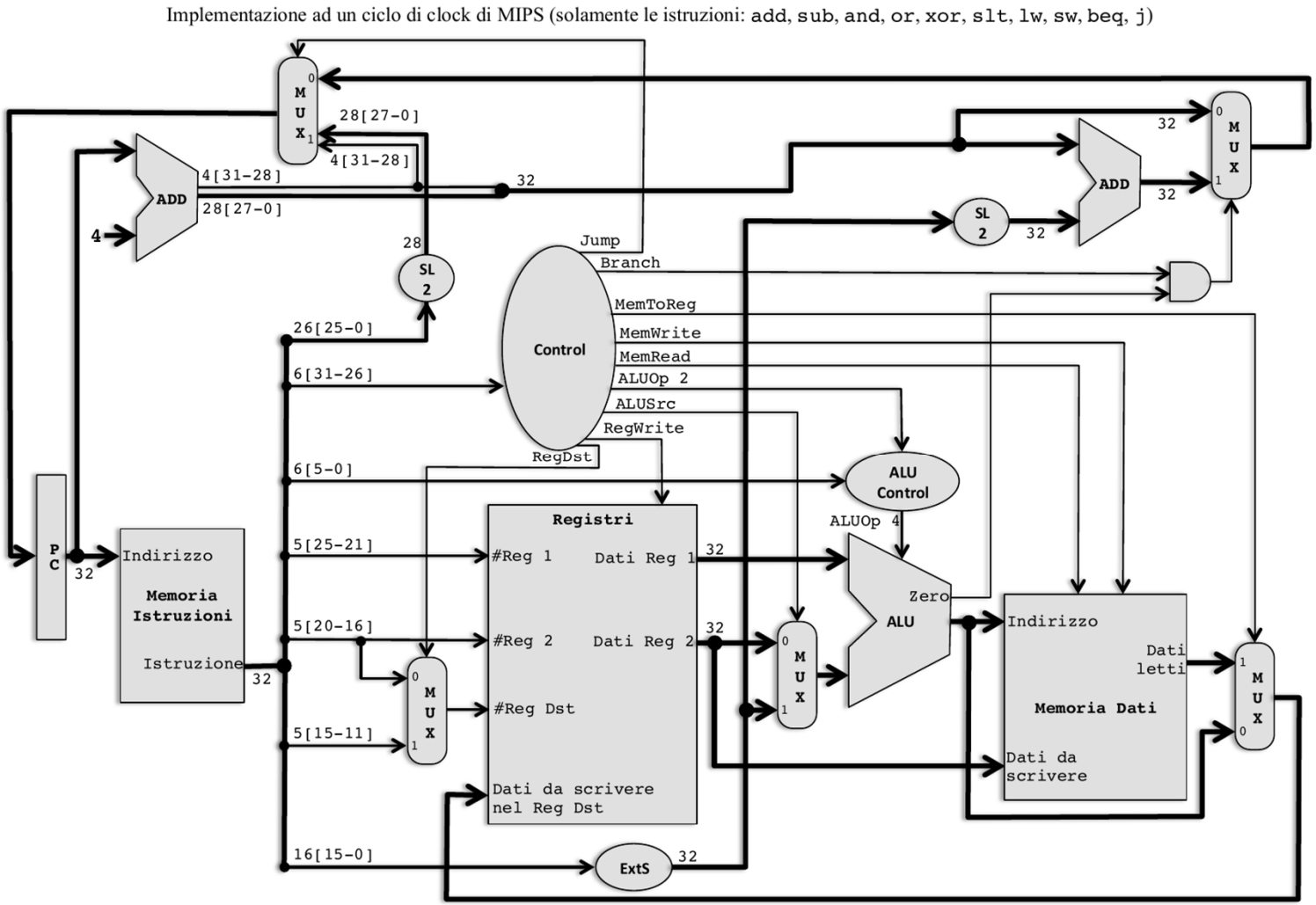
a) Si indichino quali delle istruzioni funzioneranno male e perché.

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.
Si assume che RegDst sia asserito solo per le istruzioni di tipo R
e che MemtoReg sia asserito solo per l'istruzione lw.

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

Quindi sono danneggiate le istruzioni che hanno i due segnali entrambi a 0 o entrambi a 1
- **lw**: funziona correttamente (1 0)
- **sw**: funziona correttamente (0 1)
- **beq**: salta correttamente MA INOLTRE SCRIVE IN MEMORIA (0 1 invece che 0 0)
- **j**: salta correttamente MA INOLTRE SCRIVE IN MEMORIA (0 1 invece che 0 0)
- **tipo R**:funzionano correttamente (1 0)

MemWrite = not RegWrite



Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

`beq` **`$rs`**`, `**`$rt`**`, `***`etichetta`*** che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

beq **$rs, $rt, *etichetta*** che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

j ***etichetta*** che memorizza:

- il valore del registro indicato dal campo rt della istruzione (che sono i 5 bit 20.16) e che si trovano in mezzo all'indirizzo di destinazione

- all'indirizzo che si ottiene in uscita dalla ALU, di cui però non sappiamo quale funzione viene svolta

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

`beq $rs, $rt, etichetta` che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

`j etichetta` che memorizza:

- il valore del registro indicato dal campo rt della istruzione (che sono i 5 bit 20.16) e che si trovano in mezzo all'indirizzo di destinazione

- all'indirizzo che si ottiene in uscita dalla ALU, di cui però non sappiamo quale funzione viene svolta

Conviene quindi usare la **beq** che possiamo controllare meglio:

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

beq **$rs, $rt, *etichetta*** che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

j ***etichetta*** che memorizza:

- il valore del registro indicato dal campo rt della istruzione (che sono i 5 bit 20.16) e che si trovano in mezzo all'indirizzo di destinazione

- all'indirizzo che si ottiene in uscita dalla ALU, di cui però non sappiamo quale funzione viene svolta

Conviene quindi usare la **beq** che possiamo controllare meglio:

```
move $s0, $zero          # $s0 = 0
sw   $s0, 0              # memorizzo 0 all'indirizzo 0



lw   $s0, 0              # carico il contenuto dell'indirizzo 0
```

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

beq **$rs, $rt, *etichetta*** che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

j ***etichetta*** che memorizza:

- il valore del registro indicato dal campo rt della istruzione (che sono i 5 bit 20.16) e che si trovano in mezzo all'indirizzo di destinazione

- all'indirizzo che si ottiene in uscita dalla ALU, di cui però non sappiamo quale funzione viene svolta

Conviene quindi usare la **beq** che possiamo controllare meglio:

```
move $s0, $zero          # $s0 = 0
sw   $s0, 0              # memorizzo 0 all'indirizzo 0

beq  $s1, $s1, 0         # salto di zero istruzioni = continua
                         # se rotta viene memorizzato 1 all'indirizzo 0
lw   $s0, 0              # carico il contenuto dell'indirizzo 0
```

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare una delle due istruzioni di salto:

beq **$rs, $rt, *etichetta*** che memorizza il valore del registro $rt all'indirizzo calcolato dalla ALU facendo la differenza dei due operandi $rs e $rt per realizzare il confronto

j ***etichetta*** che memorizza:

- il valore del registro indicato dal campo rt della istruzione (che sono i 5 bit 20.16) e che si trovano in mezzo all'indirizzo di destinazione

- all'indirizzo che si ottiene in uscita dalla ALU, di cui però non sappiamo quale funzione viene svolta

Conviene quindi usare la **beq** che possiamo controllare meglio:

```
move $s0, $zero          # $s0 = 0
sw   $s0, 0              # memorizzo 0 all'indirizzo 0
li   $s1, 1              # $s1 = 1
beq  $s1, $s1, 0         # salto di zero istruzioni = continua
                         # se rotta viene memorizzato 1 all'indirizzo 0
lw   $s0, 0              # carico il contenuto dell'indirizzo 0
```

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 RegDst sia asserito solo per le istruzioni di tipo R, che MemRead sia asserito solo per l'istruzione lw e che MemtoReg sia asserito solo per l'istruzione lw.

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

- 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 RegDst sia asserito solo per le istruzioni di tipo R, che MemRead sia asserito solo per l'istruzione lw e che MemtoReg sia asserito solo per l'istruzione lw.

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.

	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	**0**	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	**1**	X	X

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 RegDst sia asserito solo per le istruzioni di tipo R, che MemRead sia asserito solo per l'istruzione lw e che MemtoReg sia asserito solo per l'istruzione lw.

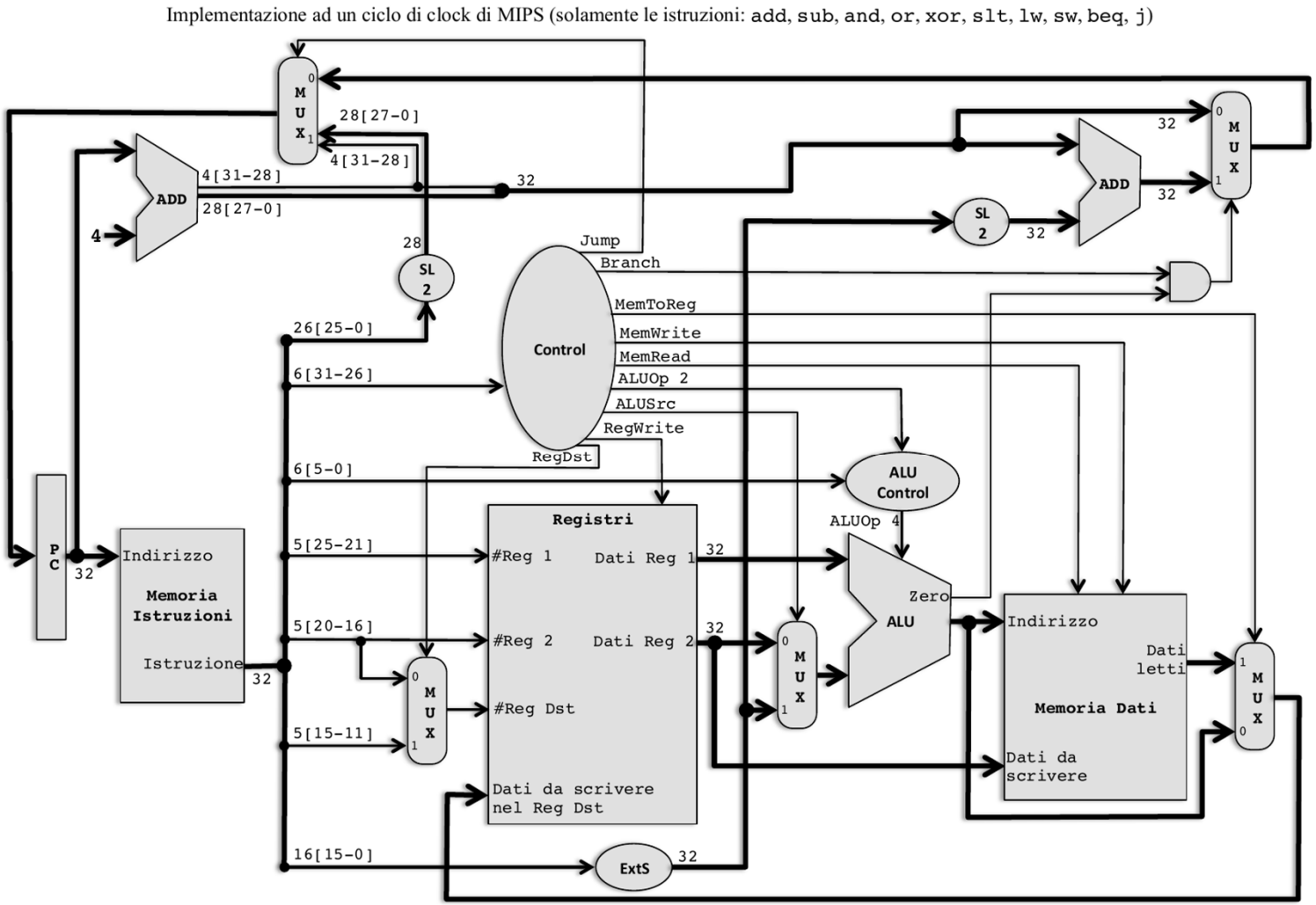
a) Si indichino quali delle istruzioni funzioneranno male e perché.

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.

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



Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare **j** oppure **lw**:

Soluzione

b) Per distinguere se la CPU è rotta dobbiamo quindi usare **j** oppure **lw**:

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

`j` `etichetta` che NON salta

Soluzione

b) Per distinguere se la CPU è rotta 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** che è più semplice:

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

Aggiungere l'istruzione jral

- Vogliamo aggiungere l'istruzione di tipo R
- `jral $rs, $rt` (**J**ump to **R**egister **a**nd **L**ink to rt)
- che salta incondizionatamente all'indirizzo contenuto nel registro **rs** e memorizza nel registro **rt** l'indirizzo della istruzione successiva.

a) Modificate lo schema per realizzare l'istruzione
b) Indicate tutti i segnali di controllo che la CU deve generare
c) Calcolate il tempo di esecuzione della istruzione assumendo che:
Accesso a memorie = 100ns, accesso ai registri = 50ns, ALU e sommatori = 150ns

Aggiungere l'istruzione jral

- Vogliamo aggiungere l'istruzione di tipo R
- `jral $rs, $rt` (**J**ump to **R**egister **a**nd **L**ink to rt)
- che salta incondizionatamente all'indirizzo contenuto nel registro **rs** e memorizza nel registro **rt** l'indirizzo della istruzione successiva.

a) Modificate lo schema per realizzare l'istruzione

b) Indicate tutti i segnali di controllo che la CU deve generare

c) Calcolate il tempo di esecuzione della istruzione assumendo che:
Accesso a memorie = 100ns, accesso ai registri = 50ns, ALU e sommatori = 150ns

a) Le operazioni da compiere sono:

PC ← Registri[rs] **(salto incondizionato)**

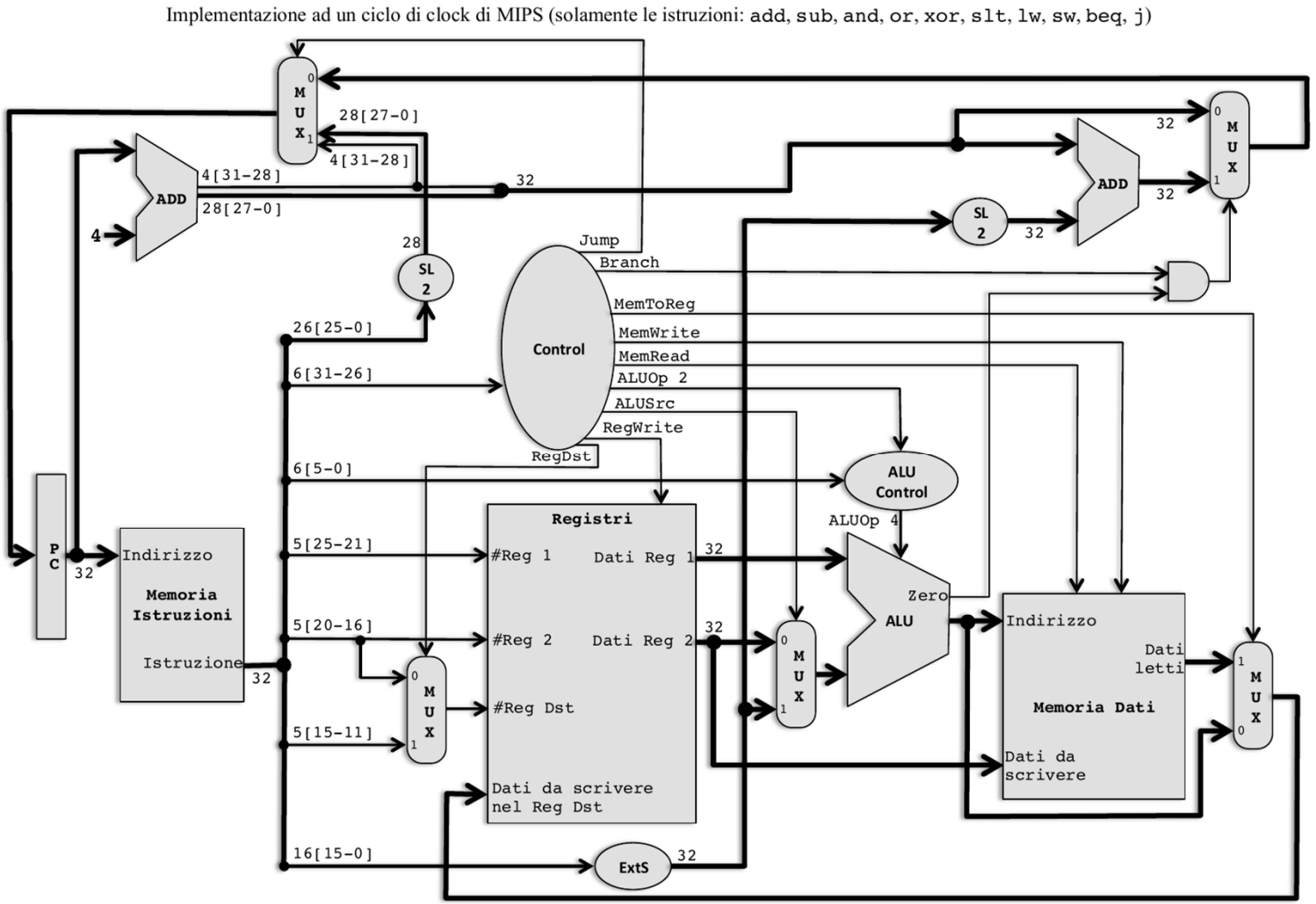
Registri[rt] ← PC + 4 **(link)**

Per cui non c'è bisogno di unità funzionali nuove, a parte i necessari MUX.

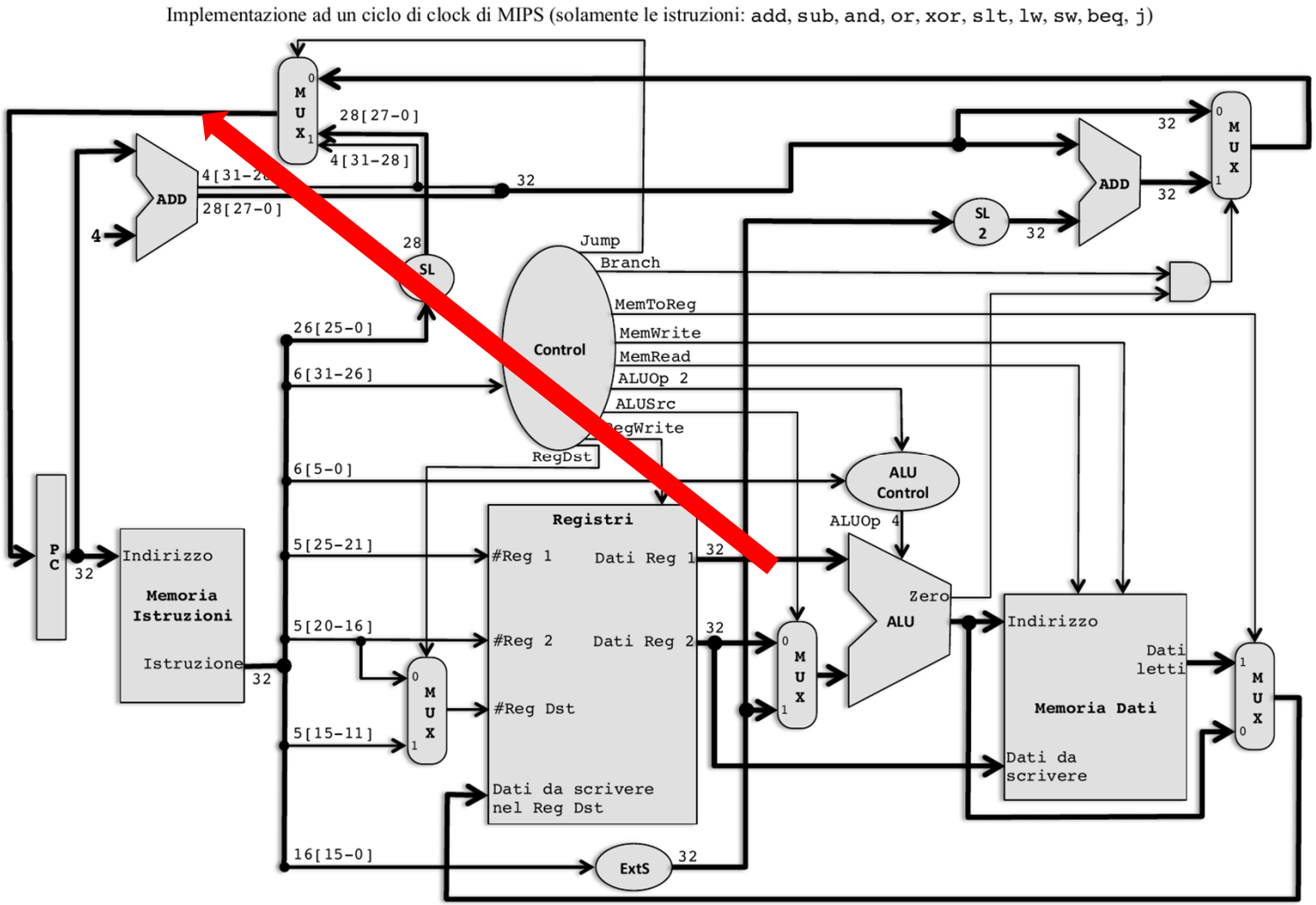
Modifiche ai Datapath:

- Bisogna mandare il PC+4 all'ingresso di scrittura del blocco registri (con un MUX)
- Bisogna mandare il campo rt della istruzione come registro destinazione (come in una lw)
- Bisogna mandare l'uscita del primo argomento del blocco registri a PC (con un MUX)

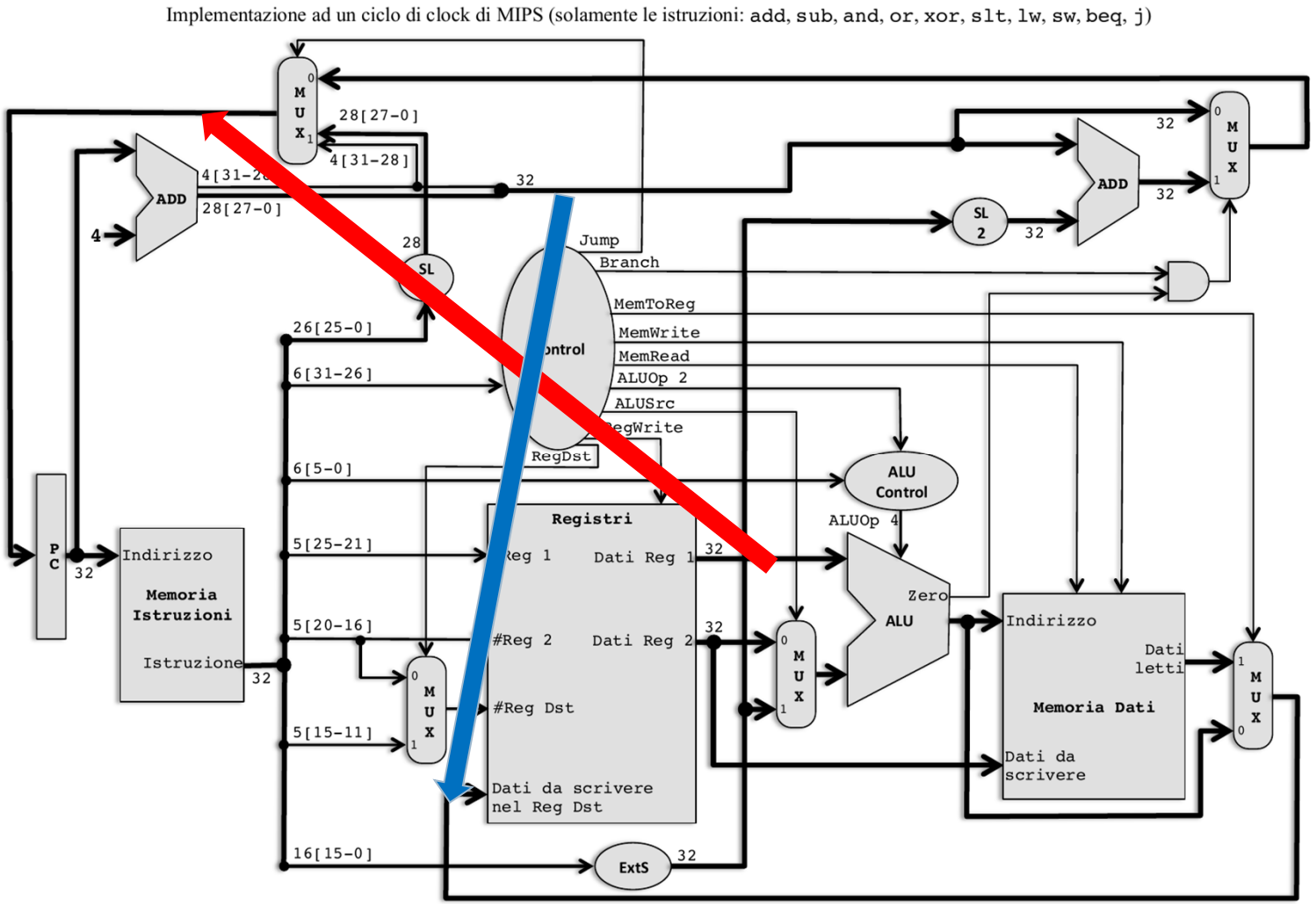
Modifiche per JRAL



Modifiche per JRAL



Modifiche per JRAL



Soluzione jral

b) I segnali da inviare sono quindi:

Soluzione jral

b) I segnali da inviare sono quindi:

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

Soluzione jral

b) I segnali da inviare sono quindi:

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

c) Il tempo di esecuzione è dato dalle fasi di **Fetch**, **ID** e **WB** che avvengono parzialmente in parallelo:

Fetch 100ns	Reg[rs] 50ns	PC←Reg[rs] 0ns	
PC+4 150ns		Reg[rt]←PC+4 50ns	

per cui sono necessari 200ns, quindi il tempo di esecuzione delle istruzioni non si allunga

Soluzione esercizio

- Aggiungere alla CPU l'istruzione **`jrr rs`** (Jump Relative to Register)
- di tipo R, che salta all'indirizzo (relativo al PC) contenuto nel registro **rs**
- Ovvero che esegue come prossima istruzione quella all'indirizzo **PC+4+Registri[rs]**

a) Modificate lo schema per realizzare l'istruzione
b) Indicate tutti i segnali di controllo che la CU deve generare
c) Calcolate il tempo di esecuzione della istruzione assumendo che:
Accesso a memorie = 66ns, accesso ai registri = 33ns, ALU e sommatori = 100ns

a) Le operazioni da compiere sono:

PC ← PC+4+Registri[rs]

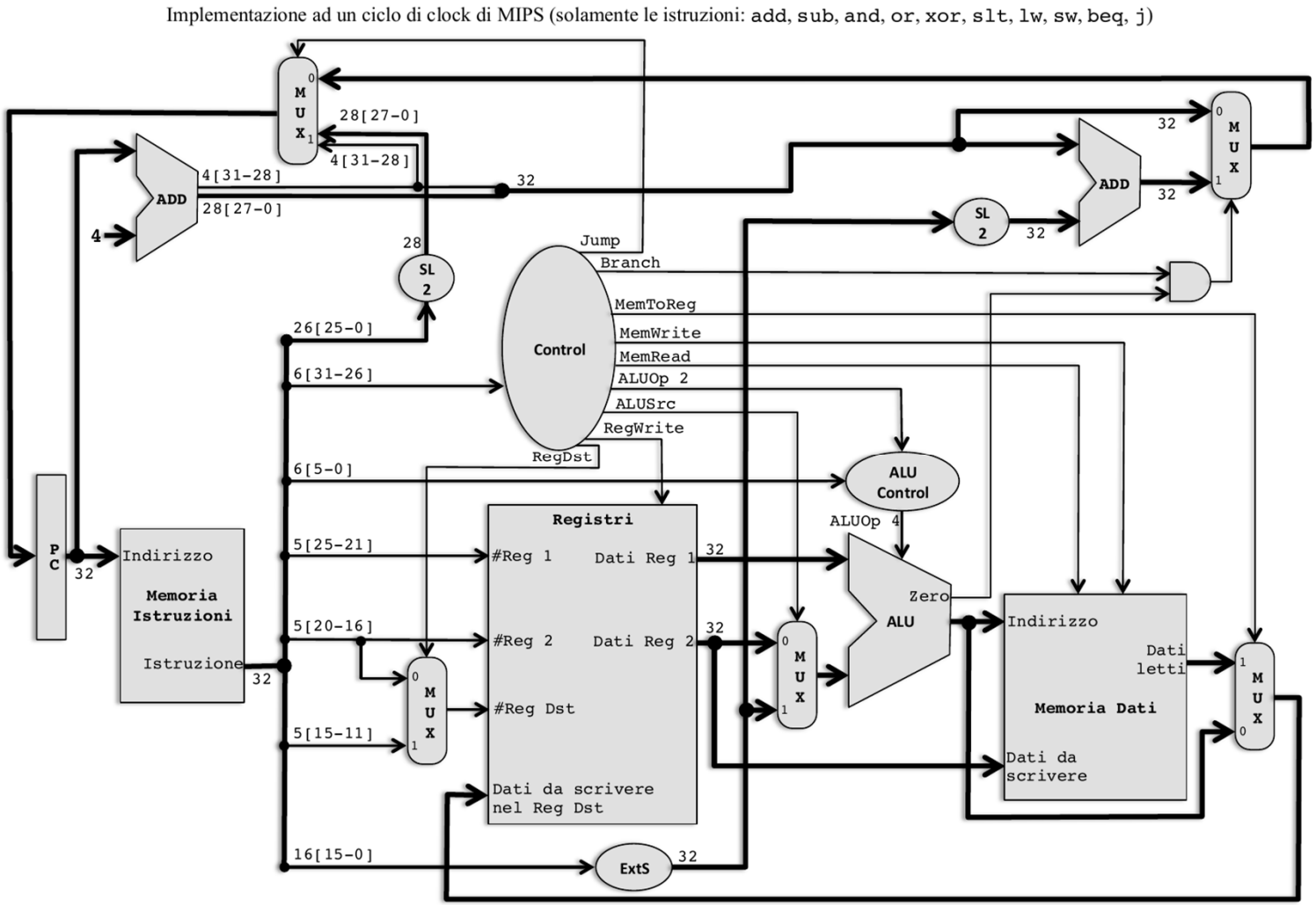
Quindi dobbiamo:

- Leggere il registro rs e usarlo come salto relativo al PC+4

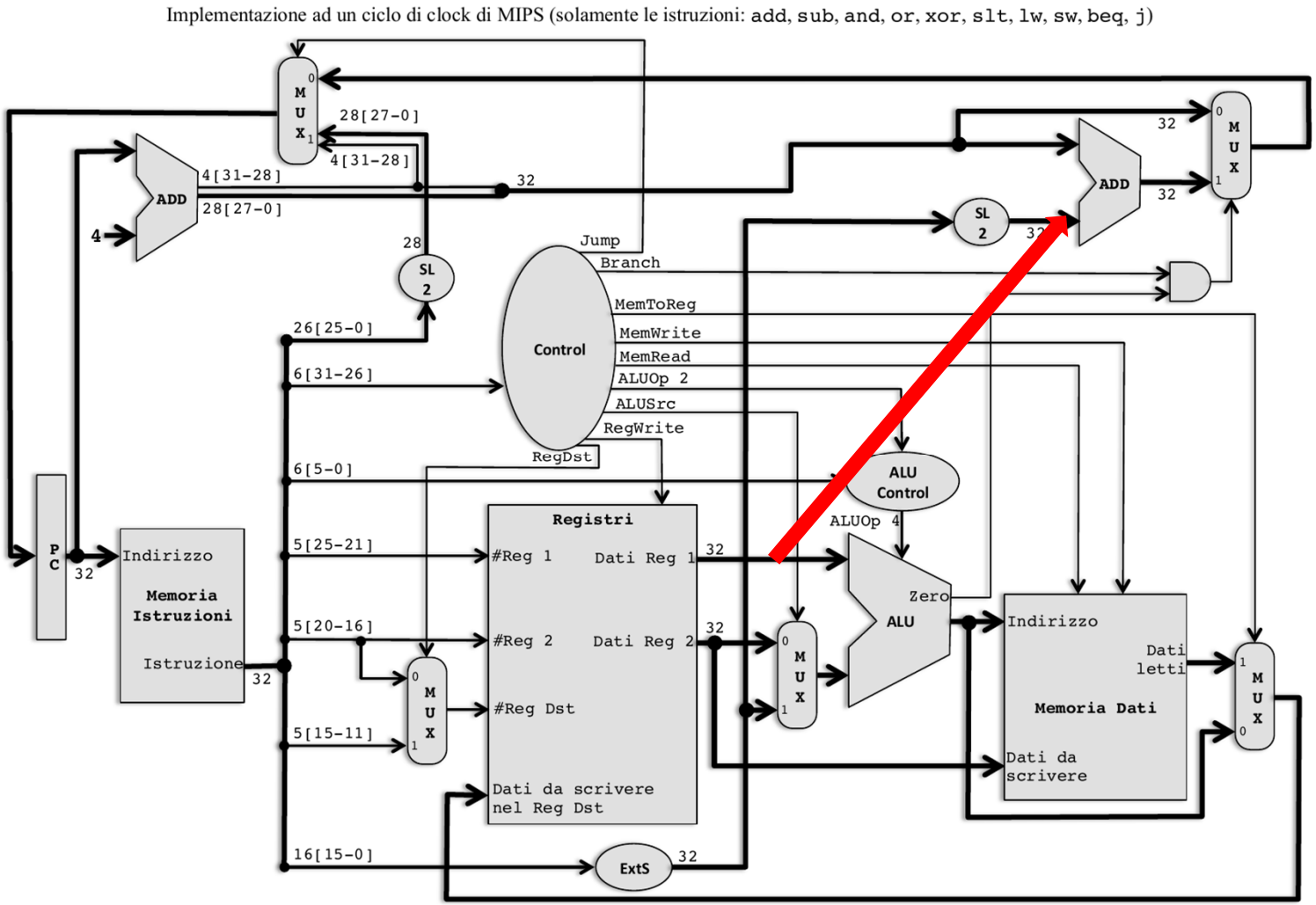
Dobbiamo quindi fare un salto relativo (come in un **branch**) ma senza l'estensione del segno e il prodotto per 4 perché il contenuto del registro già contiene l'offset in byte (il testo dell'esercizio non parla di offset come numero di istruzioni)

Per quanto riguarda il datapath dobbiamo collegare l'uscita del primo argomento del blocco registri (Reg[rs]) all'ingresso del sommatore dei salti relativi (con un MUX)

Modifiche per jrr $rs



Modifiche per jrr $rs



Soluzione jrr $rs

b) I segnali da inviare sono quindi:

Soluzione jrr $rs

b) I segnali da inviare sono quindi:

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

Soluzione jrr $rs

b) I segnali da inviare sono quindi:

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

c) Il tempo di esecuzione è dato dalle fasi di **Fetch**, **ID** e **calcolo dell'indirizzo relativo** che avvengono parzialmente in parallelo:

Fetch 66ns	Reg[rs] 33ns			
PC+4 100ns		salto = PC+4 + Reg[rs] 100ns	PC ← salto 0ns	

per cui sono necessari 200ns, quindi il tempo di esecuzione delle istruzioni non si allunga

Esercizio

Si vuole aggiungere alla CPU l'istruzione vectorized 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 15, 24, 313, 42
```

e nel registro **$t0** c'è il valore **3** allora `vj $t0, vettore` salterà all'indirizzo **42** (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 **25ps**, l'accesso alla memoria impiega **75ps**, la ALU e i sommatori impiegano **100ps** 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.