Esame di Architetture – Sterbini – Compito A – 20/6/18

Cognome e Nome: _____ Matricola: _____

Parte 1 (per chi non ha superato l'esonero – 1 ora)

Esercizio 1A (14 punti). In una partita di CPU a ciclo di clock singolo (vedi sul retro) la Control Unit potrebbe essere rotta, producendo il segnale di controllo **AluSrc** attivo se e solo se **è attivo il segnale MemRead ma non è attivo il segnale RegDst**.
NOTA: A proposito dei valori don't care, **si assuma che valgano sempre 1**.

a) Si indichino *qui sotto* quali delle istruzioni base (**lw, sw, di tipo R, di tipo R-imm., beq, j**) funzioneranno male, indicando quali sono i comportamenti diversi.

Soluzione

	MemRead	RegDst	AluSrc	Rotto		
lw	1	0	1	1	ok	
sw	x(1)	x(1)	1	0	ko	Address = ($rs) + ($rt)
R	x(1)	1	0	0	ok	
R-imm	x(1)	0	1	1	ok	
beq	x(1)	x(1)	0	0	ok	
j	x(1)	x(1)	x(1)	0	ok (era don't care)	

b) si scriva *qui sotto* un breve programma assembly MIPS (senza pseudoistruzioni) che termina valorizzando il registro $s0 con il valore 0 se il processore è guasto, altrimenti con 1.
Se necessario potete usare il blocco **.data** per inizializzare i valori della memoria.

Soluzione

```
.data
uno:        .word 1
valore: .word 0
.text
lw $t0, uno
sw $t0, valore      # memorizzo 1 in valore se la CU è corretta, altrimenti lo memorizzo altrove
lw $s0, valore      # leggo 1 se tutto OK, altrimenti leggo 0
```

Esercizio 2A (16 punti). Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro). Vogliamo aggiungere l'istruzione di tipo I **`pop rs, rt`** (pop del valore in cima allo stack indirizzato dal registro $rs, nel registro $rt) che: legge il valore indirizzato dal registro $rs e lo mette in $rt, e contemporaneamente incrementa il registro $rs di una word.

1) modificate il diagramma mostrando gli eventuali altri componenti necessari a realizzare l'istruzione e realizzando i datapath necessari e se necessario modificando i componenti presenti.

2) indicate sul diagramma i valori di tutti i segnali di controllo necessari a realizzare l'istruzione

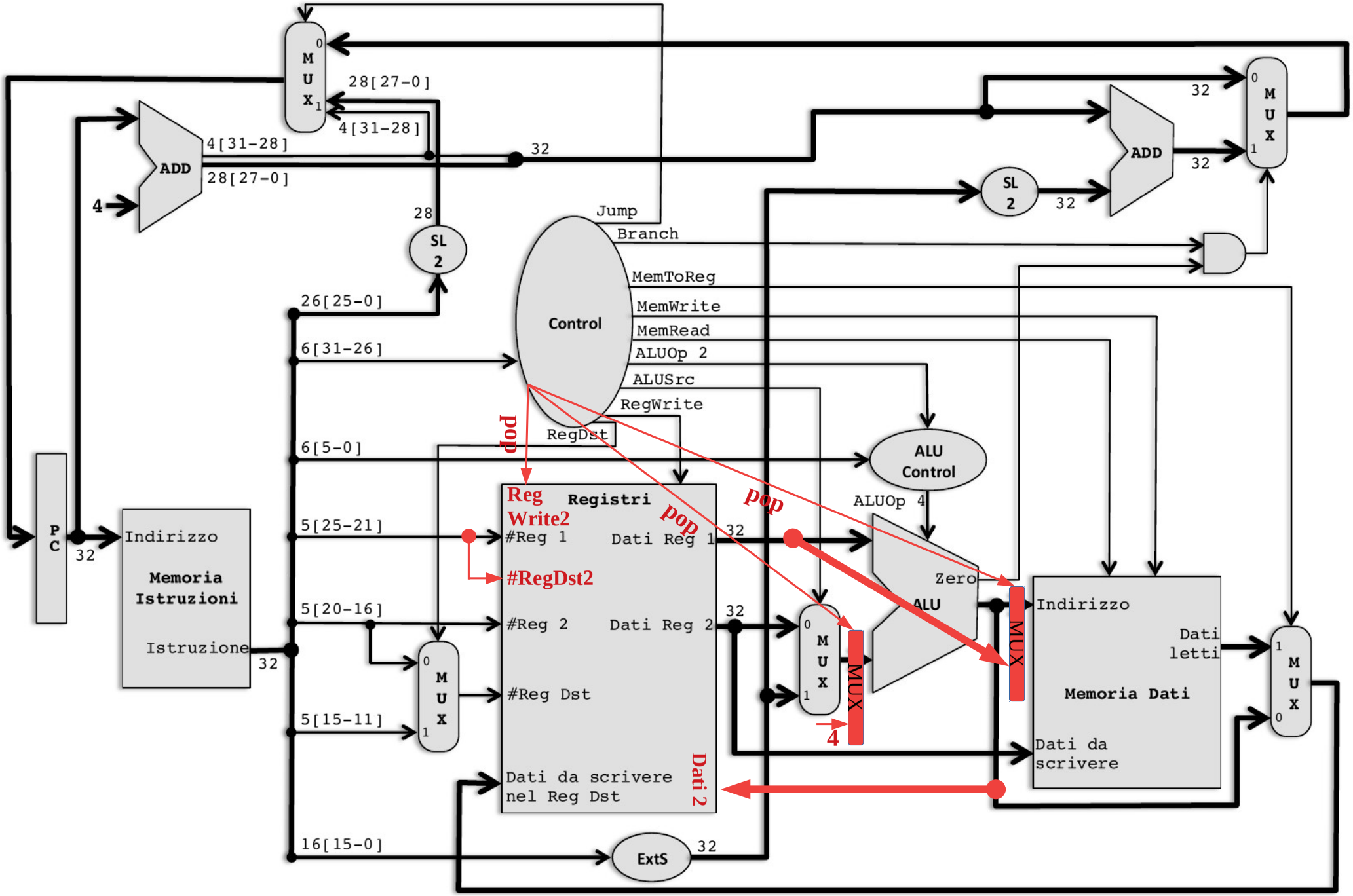
3) supponendo che l'accesso alle memorie impieghi **45ns**, l'accesso ai registri **5ns**, le operazioni dell'ALU e dei sommatori **100ns**, e ignorando gli altri ritardi di propagazione dei segnali, indicate sul diagramma la durata totale del ciclo di clock per permettere l'esecuzione della nuova istruzione e se la durata totale del ciclo di clock necessario è aumentata rispetto alla CPU senza la nuova istruzione.

Soluzione

Bisogna contemporaneamente eseguire **lw $rt, ($rs)** e **addi $rs, $rs, 4** che eseguono 2 scritture contemporanee su due registri. Quindi bisogna modificare il blocco dei registri aggiungendo: una porta "dati da scrivere 2" da 32 bit, una porta "#Reg dest 2" da 5 bit per indicare in quale registro scrivere il dato, ed un segnale di controllo "RegWrite2" per attivare la scrittura solo se necessario. Inoltre è necessario inserire il valore 4 come secondo argomento dell'ALU per calcolare $rs=$rs+4, e creare un bypass per fornire il valore iniziale di $rs come indirizzo alla memoria, quindi servono 2 MUX. I segnali di controllo necessari sono: RegDst=0, RegWrite=1, AluSrc=X, Jump=0, Branch=0, MemWrite=0, MemRead=1, AluOp=sum, MemToReg=1, pop=1

Tempi: IF=45ns+ID=5ns+EXE=100ns (MEM può avvenire in contemporanea a EXE)+WB=5 => 155ns

Implementazione ad un ciclo di clock di MIPS (solamente le istruzioni: `add`, `sub`, `and`, `or`, `xor`, `slt`, `lw`, `sw`, `beq`, `j`)



Esame di Architetture – Sterbini – Compito A – 20/6/18

Cognome e Nome: _____ Matricola: _____

Parte 2 (per tutti – 2 ore)

Esercizio 3A (16 punti). Considerate l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il frammento di programma qui sotto che somma gli elementi di una lista che usa puntatori in memoria.

NOTA: <u>jr salta alla fine della fase ID, jal alla fine della fase IF</u>
NOTA: assumete che tutte le istruzioni usate nel programma siano di base (nessuna pseudoistruzione).

Indicate:
1) tra quali istruzioni sono presenti data hazard (sottolineate in modo chiaro le coppie di registri implicate),

2) tra quali istruzioni sono presenti control hazard, (sottolineate in modo chiaro le coppie di istruzioni implicate),

3) tra quali istruzioni sono necessari stalli (data e control) con il forwarding (e quanti stalli)

4) calcolate quanti cicli di clock sono necessari a eseguire tutto il programma **con il forwarding**

```
.globl main
.data
Nodo4: .word 4, 0       # ultimo nodo
Nodo3: .word 3, Nodo4   # penultimo
Nodo2: .word 2, Nodo3   # ...
Nodo1: .word 1, Nodo2   # primo
Lista: .word Nodo1      # puntatore al primo

.text
main:  lw   $a0,  Lista       # addr. primo nodo
       jal  sommaLista        # sommo i nodi
fine:  move $a0, $v0          # print somma
       li   $v0, 1
       syscall
       li   $v0, 10           # fine
       syscall

sommaLista:
       bnez $a0, continua     # se != 0 avanti
       li   $v0, 0            # altrimenti 0
       jr   $ra
continua:
       subi $sp, $sp, 8       # alloco 2 word
       sw   $ra, 0($sp)       # salvo su stack
       sw   $a0, 4($sp)       # $ra e $a0
       lw   $a0, 4($a0)       # prossimo nodo
       jal  sommaLista        # sommo il resto
       lw   $ra, 0($sp)       # $ra e $a0
       lw   $a0, 4($sp)       # recupero da stack
       addi $sp, $sp, 8       # e disalloco
       lw   $t0, ($a0)        # valore del nodo
       add  $v0, $v0, $t0     # somma += valore
       jr   $ra               # return
```

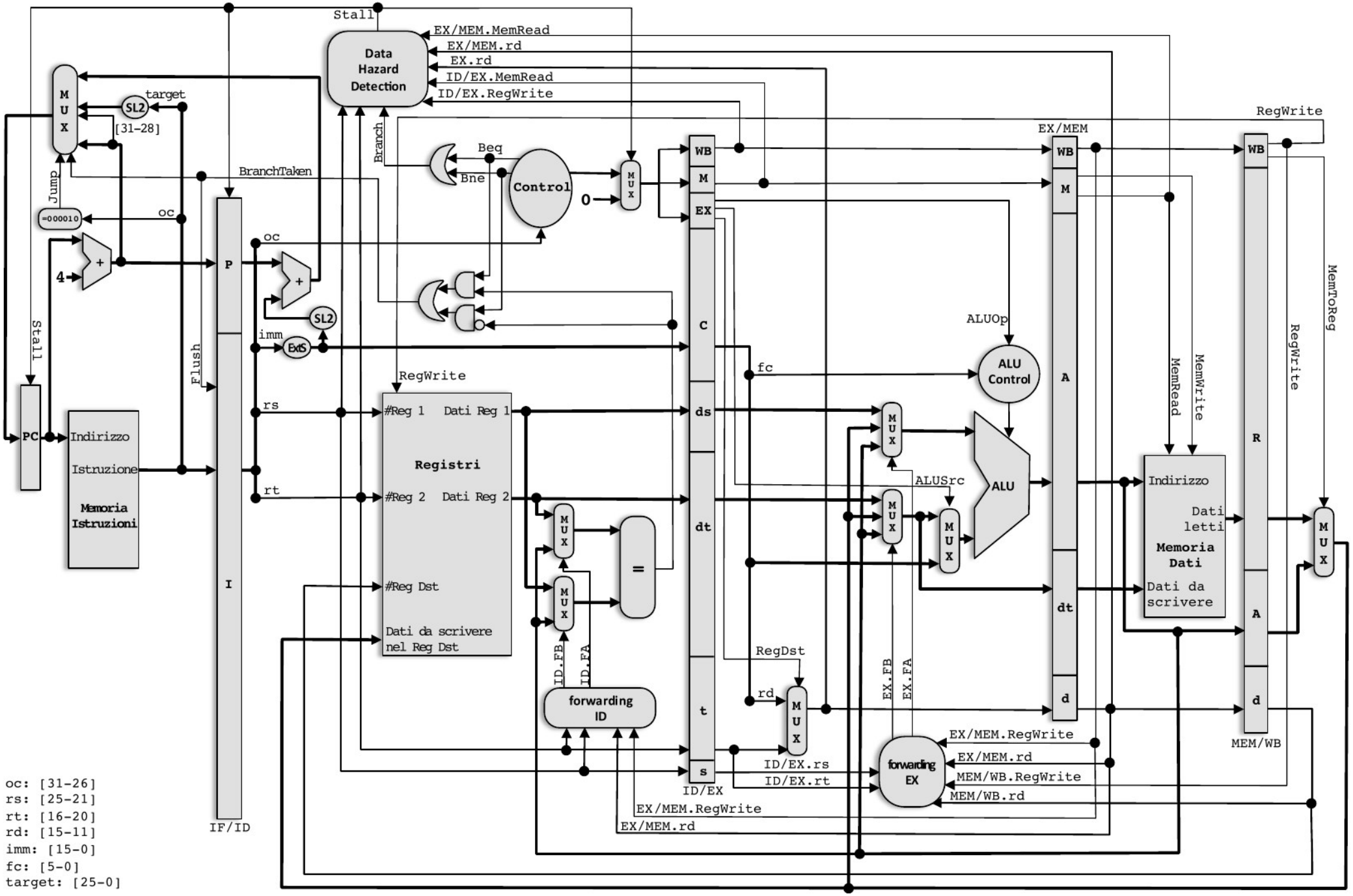
5) indicate il contenuto della pipeline (quali istruzioni si trovano in ciascuna delle 5 fasi) durante il **13°** colpo di clock (con il forwarding)

IF: **ID:** **EXE:** **MEM:** **WB:**

6) calcolate quanti cicli di clock sarebbero necessari **se il forwarding non esistesse**

7) riordinate le istruzioni per ridurre al massimo gli stalli, **con il forwarding** e mantenendo invariata la sua semantica, (su foglio separato)
8) calcolate quanti cicli di clock sono necessari a eseguire il programma così ottimizzato

Implementazione pipeline di MIPS (solamente le istruzioni: `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `xor`, `xori`, `nor`, `slt`, `slti`, `lw`, `sw`, `beq`, `bne`, `j`).



Esame di Architetture – Sterbini – Compito A – 20/6/18

Cognome e Nome: _____ Matricola: _____

Esercizio 4A (14 punti).
Considerate un sistema con due livelli di cache: **CPU <=> L1 <=> L2 <=> RAM**
- L1 è una cache **1-way** set-associativa con **8 set** e blocchi grandi **2 word** e strategia di rimpiazzo **LRU**.
- L2 è una cache **2-way** set-associativa con **4 set** e blocchi grandi **4 word** e strategia di rimpiazzo **LRU**.

1) Supponendo che gli indirizzi siano da 32 bit (indirizzamento al byte) e che all'inizio nessuno dei dati sia in cache, indicate quali degli accessi in memoria più sotto sono hit o miss in ciascuna delle due cache
2) per ciascuna MISS indicate se è di tipo **Caricamento (L), Capacità (Cap) o Conflitto (Conf)**
3) calcolate le dimensioni in bit delle due cache L1, L2 compresi i bit di controllo
4) assumendo che il processore vada a **4Ghz** con **3 CPI** (Clock Per Instruction), che gli accessi in memoria impieghino **25ns**, che gli hit nella cache L1 impieghino **2ns** e gli hit nella cache L2 impieghino **5ns**, calcolate il **tempo medio** per questa sequenza di accessi e **quante istruzioni** vengono svolte nel tempo calcolato.

	Address	**1120**	**531**	**95**	**1095**	**1088**	**2056**	**536**	**310**	**2060**	**318**	**94**	**313**	**82**	**1101**
L1	**Block#**														
	Index														
	Tag														
	Hit/Miss														
	Tipo miss														
L2	**Block#**														
	Index														
	Tag														
	Hit/Miss														
	Tipo miss														

Scrivete le soluzioni CON I PASSAGGI qui sotto continuando se necessario sul retro del foglio

Dimensioni L1:

Dimensioni L2:

Tempo totale:

Tempo medio:

Numero di istruzioni nel tempo medio:

Esame di Architetture – Sterbini – Compito B – 20/6/18

Cognome e Nome: _____ Matricola: _____

Parte 1 (per chi non ha superato l'esonero – 1 ora)

Esercizio 1B (14 punti). In una partita di CPU a ciclo di clock singolo (vedi sul retro) la Control Unit potrebbe essere rotta, producendo il segnale di controllo **RegDst** attivo se e solo se è attivo il segnale **MemRead OPPURE il segnale Branch**.
NOTA: A proposito dei valori don't care (x), si assuma che valgano sempre 0.

a) Si indichino *qui sotto* quali delle istruzioni base (**lw, sw, di tipo R, di tipo R-imm., beq, j**) funzioneranno male, indicando quali sono i comportamenti diversi.

Soluzione

	MemRead	Branch	RegDst	seRotto	
lw	1	0	0	1	salva il valore letto in $rd (preso dalla parte imm)
sw	x(0)	0	x(0)	0	ok (don't care)
R	x(0)	0	1	0	salva il valore in $rt invece che in $rd
R-imm	x(0)	0	0	0	ok
beq	x(0)	1	x(0)	1	ok (don't care)
j	x(0)	x(0)	x(0)	0	ok (don't care)

b) si scriva *qui sotto* un breve programma assembly MIPS (senza pseudoistruzioni) che termina valorizzando il registro $s0 con il valore 1 se il processore è guasto, altrimenti con 0.
Se necessario potete usare il blocco **.data** per inizializzare i valori della memoria.

Soluzione

```
.text
li $s0, 1
li $t0, 1
sub $s0, $t0, $t0          # se rotto scrive 0 in $t0 e lascia 1 in $s0, altrimenti scrive 0 in $s0
```

Esercizio 2B (16 punti). Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro). Vogliamo aggiungere l'istruzione di tipo R **`push rd, rs, rt`** (inserimento del valore contenuto nel registro $rt sullo stack indirizzato dal registo $rs) che:

- decrementa il registro $rs di 4 e lo mette in $rd,
- e inoltre memorizza il valore contenuto nel registro $rt all'indirizzo ottenuto.

1) modificate il diagramma mostrando gli eventuali altri componenti necessari a realizzare l'istruzione e realizzando i datapath necessari

2) indicate sul diagramma i valori di tutti i segnali di controllo necessari a realizzare l'istruzione

3) supponendo che l'accesso alle memorie impieghi **75ns**, l'accesso ai registri **25ns**, le operazioni dell'ALU e dei sommatori **100ns**, e ignorando gli altri ritardi di propagazione dei segnali, indicate sul diagramma la durata totale del ciclo di clock per permettere l'esecuzione della nuova istruzione e se la durata totale del ciclo di clock necessario è aumentata rispetto alla CPU senza la nuova istruzione.

Soluzione

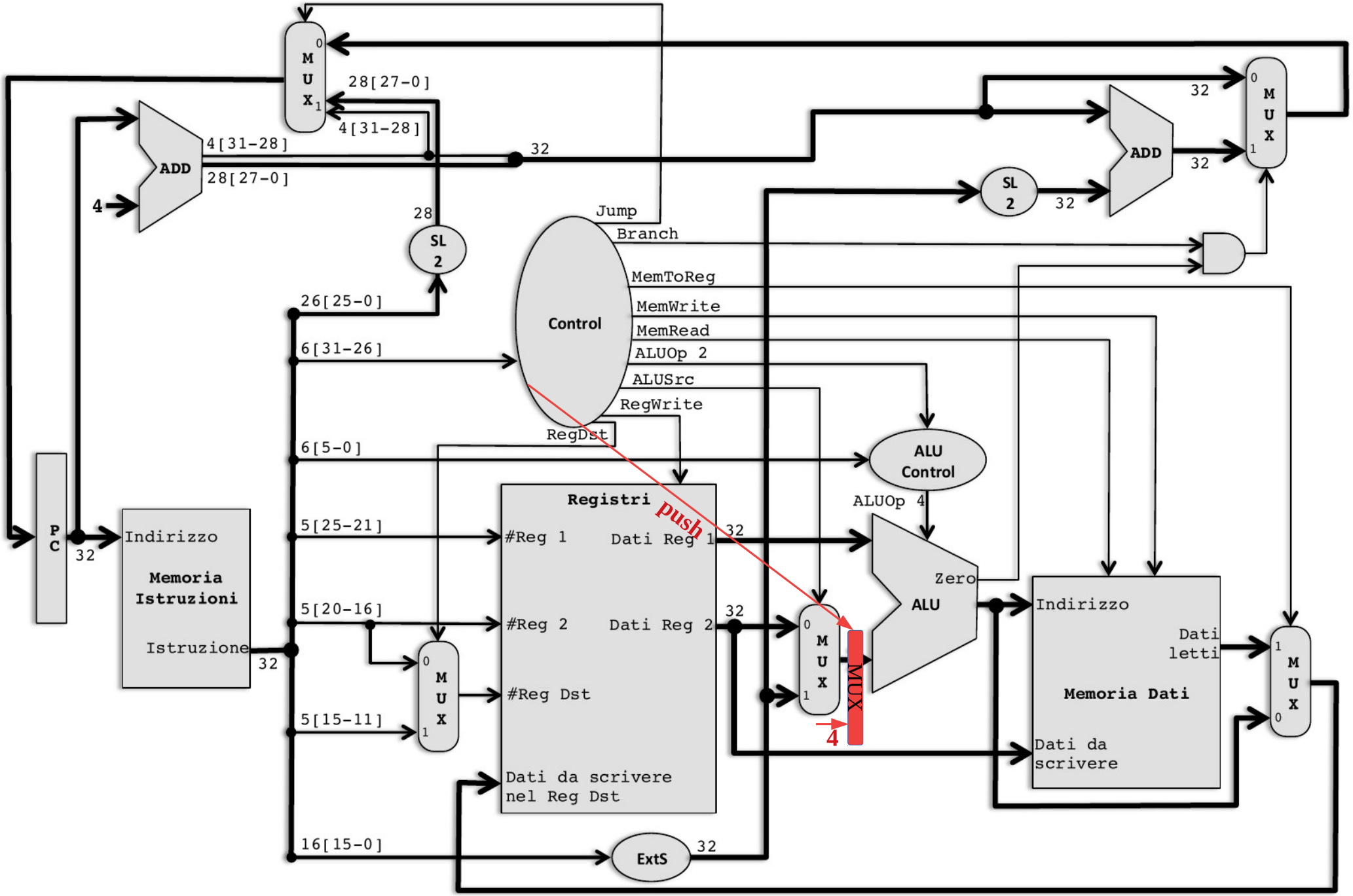
Bisogna svolgere sw $rt, -4($rs) e salvare l'indirizzo ottenuto in $rd. Le componenti necessarie ci sono, ma bisogna: sostituire il secondo argomento dell'ALU con 4 (ci vuole un MUX), chiedere all'ALU una sottrazione, portare in $rd il risultato dell'ALU, salvare $rt in memoria all'indirizzo calcolato.

I segnali di controllo saranno: Jump=0, Branch=0, AluSrc=X, AluOp=sub, MemWrite=1, MemRead=X, MemToReg=0, RegDst=1, RegWrite=1, push=1

I tempi sono: IF=75ns + ID=25ns + EXE=100ns + MEM=75ns (in contemporanea si può fare il WB) = 275ns

Non è necessario aumentare i tempi (impiega meno di una lw)

Implementazione ad un ciclo di clock di MIPS (solamente le istruzioni: `add`, `sub`, `and`, `or`, `xor`, `slt`, `lw`, `sw`, `beq`, `j`)



Esame di Architetture – Sterbini – Compito B – 20/6/18

Cognome e Nome: _____ Matricola: _____

Parte 2 (per tutti – 2 ore)

Esercizio 3B (16 punti). Considerate l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il frammento di programma qui sotto che conta il numero di nodi di una lista rappresentata in memoria come vettore di elementi interi in cui:
- il primo elemento della lista è sempre l'elemento ad indice 0
- ciascun elemento può contenere:
  - l'indice del prossimo elemento
  - oppure il valore speciale -1 che indica che **questo nodo è l'ultimo**.

NOTA: <u>jr salta alla fine della fase ID, jal alla fine della fase IF</u>
NOTA: assumete che tutte le istruzioni usate nel programma siano di base (nessuna pseudoistruzione).

```
.globl main
.data
Lista: .word 5, 2, 3, 6, -1, 1, -1, -1 # lista
.text
main:  xor  $a0, $a0, $a0     # primo elemento
       jal  conta_nodi        # conto i nodi
       move $a0, $v0          # stampo
       li         $v0, 1
       syscall
       li         $v0, 10     # fine
       syscall

conta_nodi:
       bgez $a0, continua     # se non è l'ultimo
       move $v0, $zero        # somma = 0
       jr   $ra
continua:
       subi $sp, $sp, 4       # alloco 1 word
       sw   $ra, ($sp)        # salvo $ra
       sll  $a0, $a0, 2       # leggo il prox
       lw   $a0, Lista($a0)
       jal  conta_nodi        # conto il resto
       addi $v0, $v0, 1       # somma += 1
       lw   $ra, ($sp)        # ripristino $ra
       addi $sp, $sp, 4       # disalloco 1 word
       jr   $ra               # return
```

Indicate:
1) tra quali istruzioni sono presenti data hazard (sottolineate chiaramente le coppie di registri implicati),

2) tra quali istruzioni sono presenti control hazard (sottolineate chiaramente le coppie di istruzioni implicate),

3) tra quali istruzioni sono necessari stalli (data e control) con il forwarding e quanti stalli,

4) calcolate quanti cicli di clock sono necessari a eseguire tutto il programma **con il forwarding**

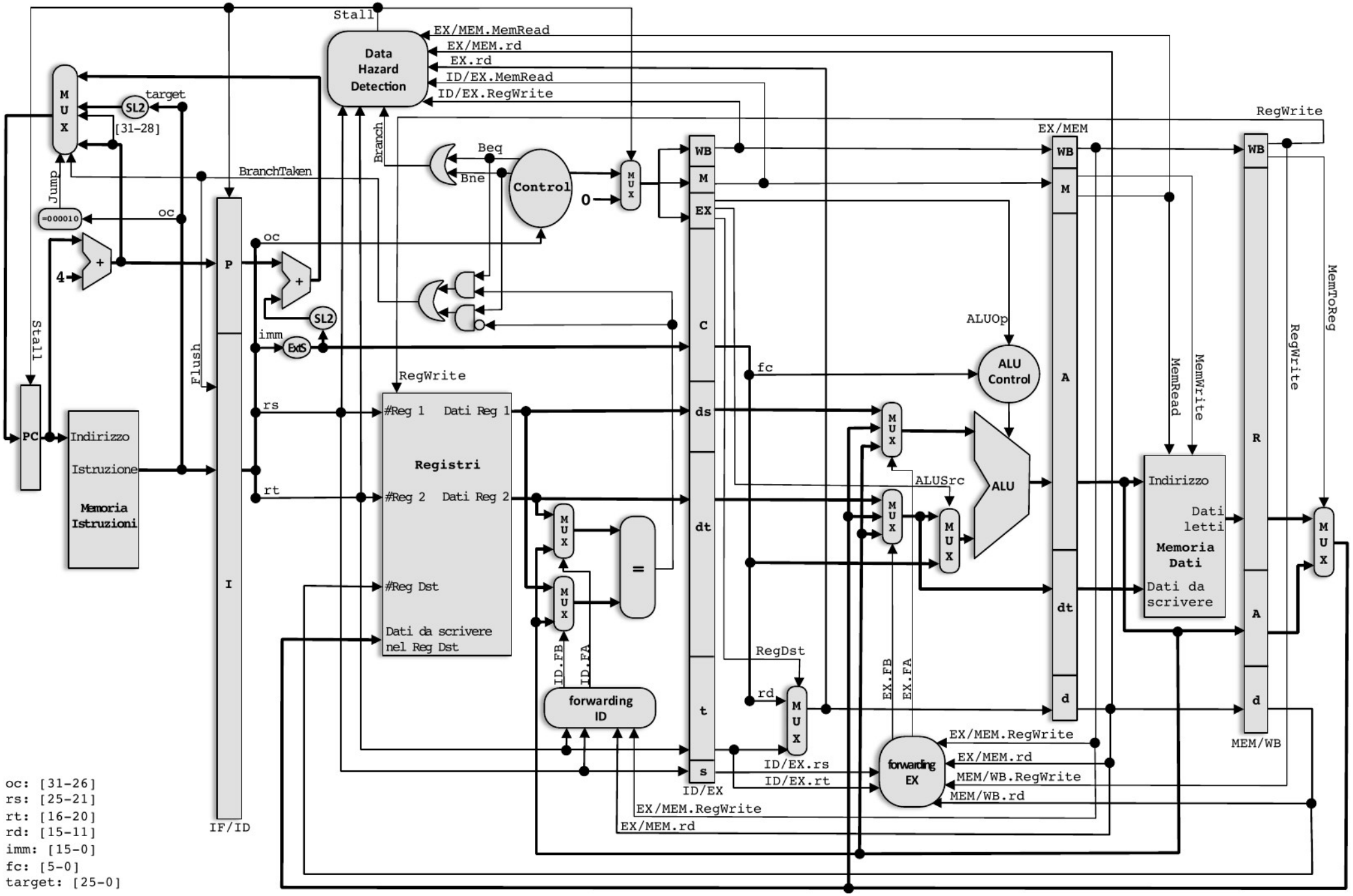
5) indicate il contenuto della pipeline (quali istruzioni si trovano in ciascuna delle 5 fasi) durante il **12°** colpo di clock (con il forwarding)

IF: **ID:** **EXE:** **MEM:** **WB:**

6) calcolate quanti cicli di clock sarebbero necessari **se il forwarding non esistesse**

7) riordinate le istruzioni per ridurre al massimo gli stalli, **con il forwarding** e mantenendo invariata la sua semantica, (su foglio separato)
8) calcolate quanti cicli di clock sono necessari a eseguire il programma così ottimizzato

Implementazione pipeline di MIPS (solamente le istruzioni: `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `xor`, `xori`, `nor`, `slt`, `slti`, `lw`, `sw`, `beq`, `bne`, `j`).



Esame di Architetture – Sterbini – Compito B – 20/6/18

Cognome e Nome: _____ Matricola: _____

Esercizio 4B (14 punti).
Considerate un sistema con due livelli di cache: **CPU <=> L1 <=> L2 <=> RAM**
- L1 è una cache **2-way** set-associativa con **8 set** e blocchi grandi **1 word** e strategia di rimpiazzo **LRU**.
- L2 è una cache **1-way** set-associativa con **4 set** e blocchi grandi **4 word** e strategia di rimpiazzo **LRU**.

1) Supponendo che gli indirizzi siano da 32 bit (indirizzamento al byte) e che all'inizio nessuno dei dati sia in cache, indicate quali degli accessi in memoria più sotto sono hit o miss in ciascuna delle due cache
2) per ciascuna MISS indicate se è di tipo **Caricamento (L), Capacità (Cap) o Conflitto (Conf)**
3) calcolate le dimensioni in bit delle due cache L1, L2 compresi i bit di controllo
4) assumendo che il processore vada a **2.5 Ghz** con **2 CPI** (Clock Per Instruction), che gli accessi in memoria impieghino **50ns**, che gli hit nella cache L1 impieghino **1ns** e gli hit nella cache L2 impieghino **10ns**, calcolate il **tempo medio** per questa sequenza di accessi e **quante istruzioni** vengono svolte nel tempo calcolato.

	Address	**1119**	**532**	**95**	**1095**	**1088**	**2056**	**536**	**320**	**2060**	**318**	**94**	**313**	**82**	**1101**
L1	**Block#**														
	Index														
	Tag														
	Hit/Miss														
	Tipo miss														
L2	**Block#**														
	Index														
	Tag														
	Hit/Miss														
	Tipo miss														

Scrivete le soluzioni CON I PASSAGGI qui sotto continuando se necessario sul retro del foglio

Dimensioni L1:

Dimensioni L2:

Tempo totale:

Tempo medio:

Numero di istruzioni nel tempo medio:

Esame di Architetture – Sterbini – Compito A – 20/6/18

Parte 3 (assembler)

Esercizio 5A (30 punti se corretto e ricorsivo, 18 se corretto e iterativo, 0 se non funziona).

Vanno svolti sia la funzione 1) che il main 2)

1) Si realizzi la funzione **MCM(x, y)** che riceve due interi e calcola il Minimo Comune Multiplo dei due numeri sapendo che MCM(x, y) = x*y/MCD(x, y).

Per calcolarlo usate la funzione **RICORSIVA** MCD(x, y), Massimo Comun Divisore, usando l'algoritmo di Euclide:

MCD(x, y) = x se y == 0

MCD(x, y) = MCD(y, x%y) se x > y

MCD(x, y) = MCD(y, x) se x < y

Esempio:

MCM(10, 75) = 10*75/MCD(10, 75) = 10*75/5 = 150 infatti

MCD(10, 75) = MCD(75, 10)= MCD(10, 5) = MCD(5, 0) = 5

2) Si realizzi un programma **main** che legge da stdin una sequenza di almeno 2 interi positivi terminata dal valore 0 e che per ogni coppia di valori consecutivi a partire dal secondo (1° e 2°, 2° e 3°, 3° e 4° …) stampa (separati da tab) il loro MCM

Esempio: se l'input fosse

10 75 27 21 49 35 100 0

Il main stamperebbe la sequenza

150 675 189 147 245 700

Esame di Architetture – Sterbini – Compito B – 20/6/18

Parte 3 (assembler)

Esercizio 5B (30 punti se corretto e ricorsivo, 18 se corretto e iterativo, 0 se non funziona).

Vanno svolti sia la funzione 1) che il main 2)

1) Si realizzi la funzione RICORSIVA **Maiuscolizza** che riceve una stringa (vettore di caratteri null-terminated) e che la trasforma **modificandola** in modo che:

- ogni parola inizi e finisca con lettera maiuscola ed abbia tutti gli altri caratteri minuscoli

- gli altri caratteri non alfabetici restino invariati

NOTA: una parola è una sequenza ininterrotta di lettere comprese tra 'a' e 'z' o tra 'A' e 'Z' (si considerino tutti gli altri caratteri comprese le cifre e le lettere accentate come separatori di parole)

Suggerimento: scandite ricorsivamente la stringa tenendo d'occhio il carattere seguente e sostituendo le prime ed ultime lettere delle parole con maiuscole e le altre con minuscole.

NOTA: per convertire un numero da minuscolo a maiuscolo e viceversa ci sono molti modi (somma/differenza di 32, usare una tabella di caratteri, calcolare le differenze rispetto ad 'a' ed 'A' eccetera …. vedete voi quale preferite).

2) Si realizzi un programma **main** che legge da stdin una stringa di massimo 100 caratteri, poi usa la funzione **Maiuscolizza** e infine stampa la stringa risultante dopo la trasformazione

Esempi (in cui le parole sono sottolineate)

Input: "paOliNo PApEriNO 113"

Output: "PaolinO PaperinO 113"

Input: "grisù il dr4gh3tto"

Output: "GriSù IL DR4GH3TtO" (anche le cifre sono separatori)

Input: "0123456789"

Output: "1234567890"

Input: "Vingardium leviosa"

Output: "VingardiuM LeviosA"