Esame di Architetture – Canale AL – Prof. Sterbini – 31/1/17

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

Esercizio 1 (12 punti). Una CPU a ciclo di clock singolo (vedi sul retro) potrebbe avere la Control Unit rotta, che produce il segnale di controllo **AluSrc** attivo **solo quando** è attivo **uno solo** tra i segnali di controllo **RegWrite o RegDst**. Si assume che RegDst sia asserito solo per le istruzioni di tipo R, che MemToReg sia asserito solo per l'istruzione lw, che AluSrc sia asserito solo per le istruzioni lw e sw e di tipo immediato.

a) Si indichino *qui sotto* quali delle istruzioni base (**lw, sw, di tipo R, beq, j, R immediate**) funzioneranno male e qual'è il comportamento anomalo in caso di CU guasta.

Soluzione
Il segnale AluSrc è quello che potrebbe errare e per i diversi tipi di istruzione dovrebbe avere il valore

Istruzione	**AluSrc**	**se CU errata**	**RegWrite**	**RegDst**	**Comportamento**
lw	1	1	1	0	ok
sw	1	0	0	0 (don't care)	calcola male l'indirizzo
R	0	0	1	1	ok
R immediate	1	1	1	0	ok
beq	0	0	0	0 (don't care)	ok
j	0 (don't care)	0	0	0 (don't care)	ok

Quindi l'istruzione che non funziona è sw che calcola l'indirizzo sommando il contenuto di $rs e $rt invece che $rs+ES(immediata)

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

Soluzione
Sfruttiamo il fatto che sw se la CU è rotta memorizza in un indirizzo diverso per sovrascrivere con il valore 0 una locazione di memoria inizializzata a 1, se non viene sovrascritta allora la CU è errata

```
.data
UNO:   .word   1
.text
main:  sw $zero, UNO($zero)      # se CU rotta l'indirizzo calcolato sarà $rs+$rt = $zero+$zero = 0
       lw $s0, UNO($zero)        # che è sicuramente diverso dall'indirizzo in memoria della etichetta UNO
```

Esercizio 2A. Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro).
Si vuole aggiungere l'istruzione di tipo I **`b_min_gez rs, rt, label`** che fa un salto **condizionato** alla destinazione indicata dalla label se il minimo tra i due valori contenuti nei due registri **rs** e **rt** è maggiore o uguale di zero.
Nella soluzione non modificate gli ingressi/uscite o le funzionalità della ALU.

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

2) indicate sul diagramma tutti i segnali di controllo che la CU genera per realizzare l'istruzione

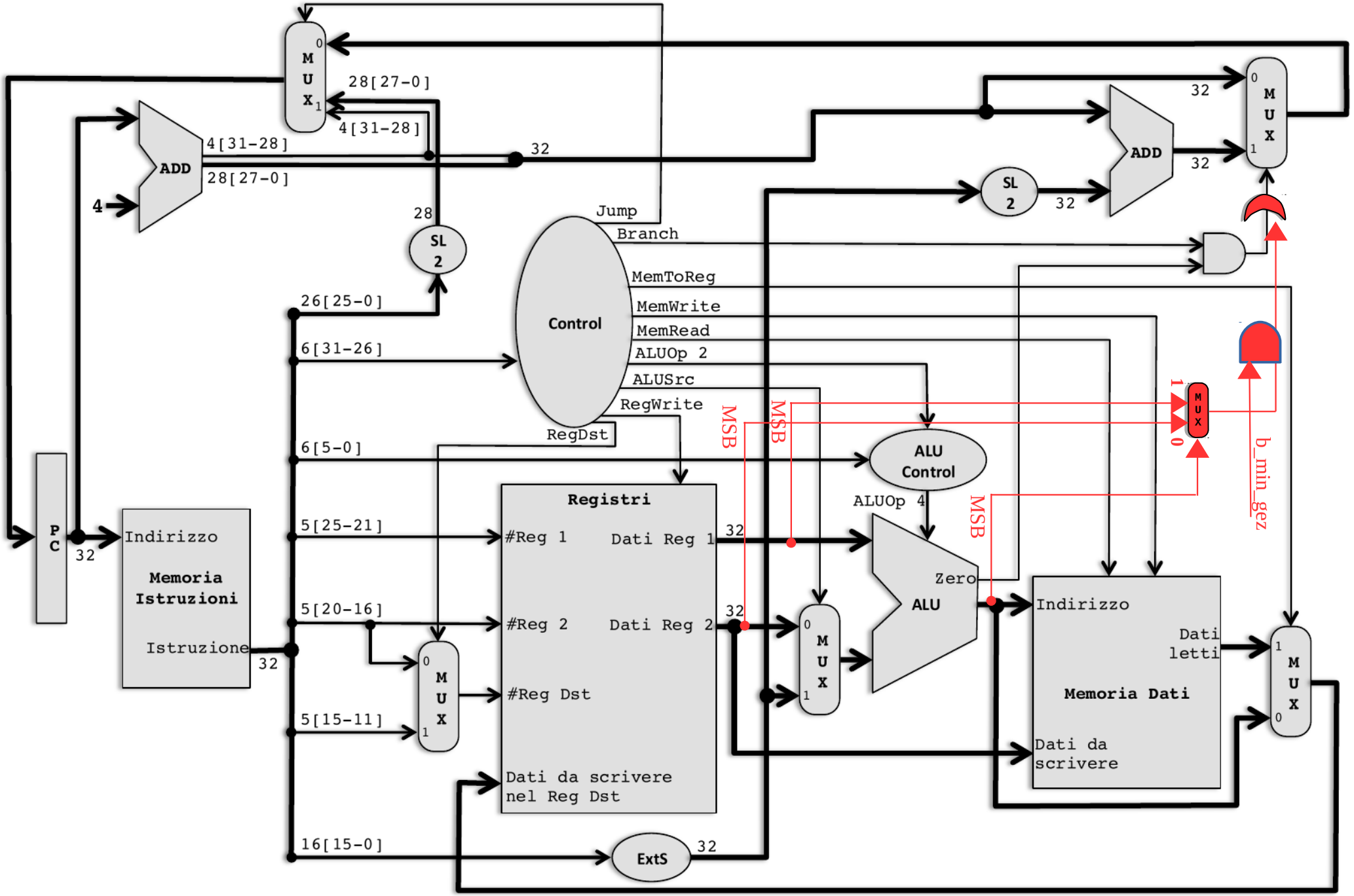
3) supponendo che l'accesso alle memorie impieghi **125ns**, l'accesso ai registri **50ns**, le operazioni dell'ALU e dei sommatori **100ns**, e ignorando gli altri ritardi di propagazione dei segnali, indicate sul diagramma la durata totale del ciclo di clock per permettere l'esecuzione anche della nuova istruzione.

Soluzione

Per calcolare qual'è il valore minimo tra $rs e $rt basta eseguire una differenza e vedere qual'è il segno X del risultato, per far questo possiamo usare direttamente l'ALU ed osservare il MSB all'uscita. Una volta individuato qual'è il valore minimo possiamo selezionare il segno del valore del registro corrispondente ed attivare il salto condizionato se necessario. Per selezionare il segno del minimo si può usare un MUX che seleziona il MSB di ($rs) o di ($rt) usando come segnale di selezione il bit X precedente. I tempi di esecuzione della istruzione sono uguali a quelli di una istruzione beq. I segnali che la CU deve generare sono: **Jump=0, Branch=0, MemToReg=X, MemWrite=0, MemRead=X, AluOp=diff, AluSrc=0, RegWrite=0, RegDst=X, b_min_gez=1**

Una seconda soluzione che non usa l'ALU può controllare che i due MSB dei due registri siano entrambi 0 (se il minimo è maggiore o uguale a 0 lo sarà anche l'altro valore)

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



Esame di Architetture – Canale AL – Prof. Sterbini – 31/1/17

Parte 2 (per tutti)

Esercizio 3A. Si consideri l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il frammento di programma qui a destra che somma i valori della diagonale pincipale di una matrice 10x10 di half word, scandendo la matrice per puntatori dalla fine all'inizio a passi di N+1 elementi.

```
.data
matrice:  .space 200
DIM:      .word 10  #lato matrice
.text
main:   add  $s4, $zero, $zero # somma=0
        lw   $s1, DIM          # N
1 stallo
        mul  $s1, $s1, $s1     # N^2
        sll  $s1, $s1, 1       # 2N^2
        subi $s1, $s1, 2       # last el.
loop:   lh   $s3, matrice($s1) # x=M[i]
1 stallo
        add  $s4, $s4, $s3     # somma+=x
        lw   $s2, DIM          # N
1 stallo
        addi $s2, $s2, 1       # N+1
        sll  $s2, $s2, 1       # 2(N+1)
        sub  $s1, $s1, $s2     # el. prima
1 stallo perché bgez è anticipato in ID
        bgez $s1, loop         # non fine?
1 stallo se si salta
end:    li   $v0, 10
        syscall
```

Si indichino qui sotto o sul codice (PASSAGGI INCLUSI):
1) tra quali istruzioni sono presenti data hazard,

Soluzione
vedi le coppie di colori

2) tra quali istruzioni sono presenti control hazard,

Soluzione
Solo tra bgez e etichetta loop (se viene fatto il salto)

3) quanti cicli di clock sono necessari a eseguire il programma con il forwarding

Soluzione
Molti DH vengono risolti dal forwarding, tranne quelli indicati in cui bisogna inserire 1 stallo. In totale vengono eseguiti 10 cicli, quindi il numero di cicli di clock totali è:
4 riempimento pipeline + 5 istruzioni prima del loop + 1 stalli + 10 * [7 istruzioni nel loop + 3 stalli DH + 1 stallo CH] - 1 stallo CH quando si esce dal loop + 2 istruzioni finali = 4 + 5 + 1 + 10*[7 + 3 + 1] - 1 + 2 = 11 + 10*11 = **121**

4) quanti ne sarebbero necessari se il forwarding non esistesse

Soluzione
Se il FW non esiste servono sempre 2 stalli per ogni DH per comunicare i dati dalla istruzione che li produce a quella che ne ha bisogno (sovrapponendo la fase WB della prima alla fase ID della seconda), quindi in totale servono:
4 riempimento pipeline + 5 istruzioni + 4*2 stalli DH + 10 * [7 istruzioni + 1 stallo CH + 5*2 stalli DH] - 1 stallo CH in uscita dal loop + 2 istruzioni = 4 + 5 + 8 + 10*[7 + 1 + 10] -1 +2 = 18 + 10*18 = **198**

5) quali sono le istruzioni contenute nei registri della pipeline durante il **11°** ciclo di clock (con FW)

Soluzione
WB: **lh** MEM: **stallo** EXE: **add** ID: **lw** IF: **addi (che diventerà stallo)**

6) come riordinare le istruzioni per ridurre il numero di stalli al massimo (su foglio a parte)

Soluzione
Si noti che le 3 istruzioni che calcolano 2*(N+1) possono essere spostate fuori dal loop perché calcolano sempre lo stesso valore, ed inoltre possono usufruire della lettura di DIM eseguita alla seconda istruzione.
Le prime due istruzioni possono essere scambiate per eliminare uno stallo.
Nel ciclo resterebbero due stalli ma se si scambiano le due istruzioni **sub** e **add** li possiamo eliminare entrambi.
Restano i DH indicati, che possono tutti essere risolti con il forwarding.
Lo stallo per CH non può essere eliminato.

```
.data
matrice:  .space 200
DIM:      .word 10  #lato matrice
.text
main:   lw   $s1, DIM          # N                  5
        add  $s4, $zero, $zero # somma=0            6
        addi $s2, $s1, 1       # N+1                7
        sll  $s2, $s2, 1       # 2(N+1)             8
        mul  $s1, $s1, $s1     # N^2                9
        sll  $s1, $s1, 1       # 2N^2               10
        subi $s1, $s1, 2       # last el.           11
loop:   lh   $s3, matrice($s1) # x=M[i]             12   17 ...    57
        sub  $s1, $s1, $s2     # el. prima          13   18        58
        add  $s4, $s4, $s3     # somma+=x           14   19        59
        bgez $s1, loop         # non fine?          15   20        60
1 stallo se si salta                                16   21 ...
end:    li   $v0, 10                                             61
        syscall                                                  62
```

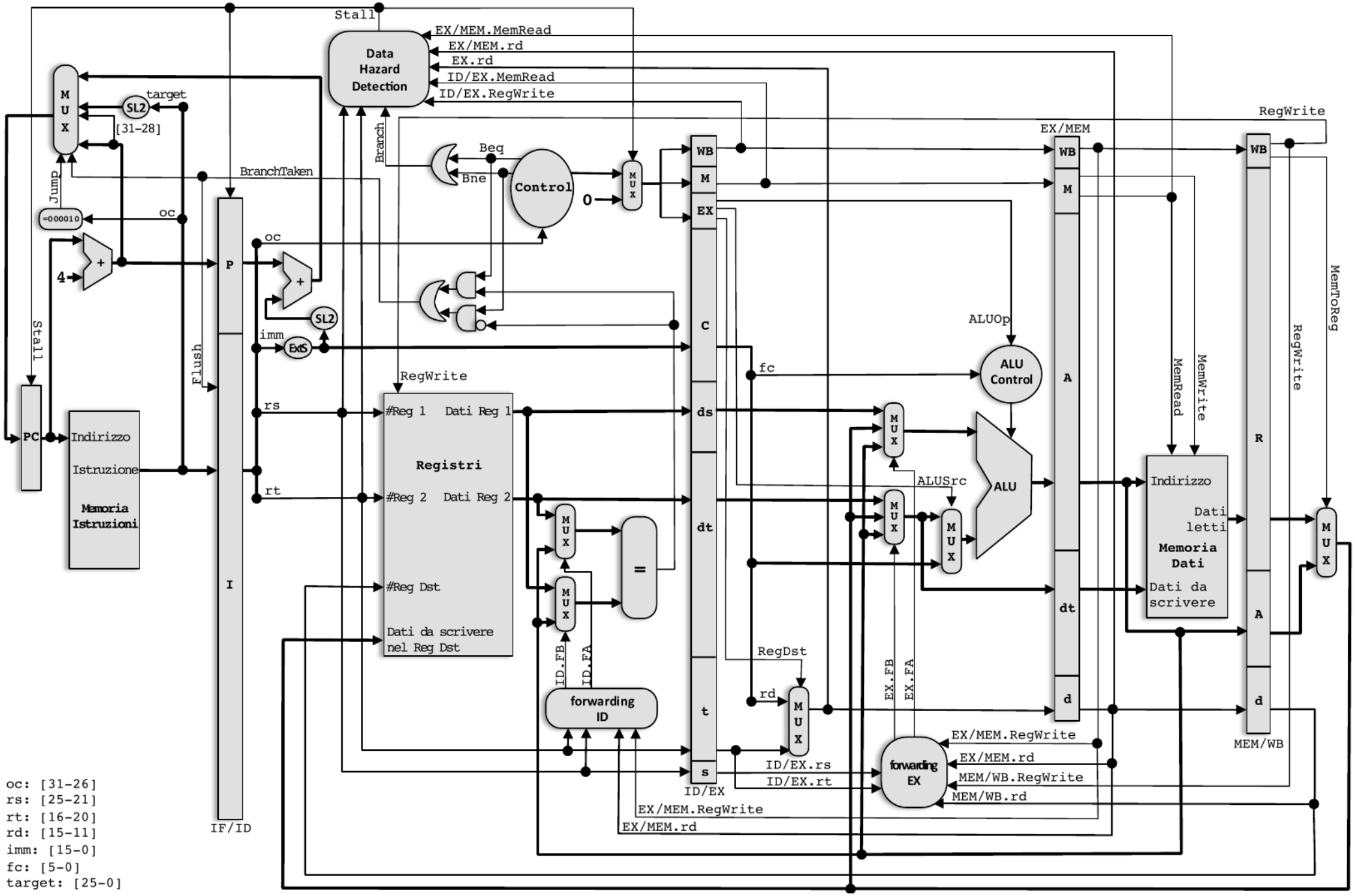
7) quanti colpi di clock sarebbero necessari nel codice ottimizzato

Soluzione
4 riempimento pipeline + 7 istruzioni + 0 stalli + 10 * [4 istruzioni + 0 stalli DH + 1 stallo CH] - 1 stallo CH alla fine del ciclo + 2 istruzioni = 12 + 10*5 = **62**

8) quali sono le istruzioni contenute nei registri della pipeline durante il **13°** ciclo di clock nel codice ottimizzato

Soluzione
WB: **sub** MEM: **add** EXE: **bgez** ID: **stallo** IF: **lh**

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




```
oc: [31-26]
rs: [25-21]
rt: [16-20]
rd: [15-11]
imm: [15-0]
fc: [5-0]
target: [25-0]
```

Esame di Architetture – Canale MZ – 31/1/17

Esercizio 4 (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 **4 word** e strategia di rimpiazzo **LRU**.
- L2 è una cache **1-way** set-associativa con **4 set** e blocchi grandi **16 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)**

	Address	1120	545	129	1099	1100	2056	319	445	2060	538	131	318	130	528
L1	Block#	70	34	8	68	68	128	19	27	128	33	8	19	8	33
	Index	6	2	0	4	4	0	3	3	0	1	0	3	0	1
	Tag	8	4	1	8	8	16	2	3	16	4	1	2	1	4
	Hit/Miss	M	M	M	M	H	M	M	M	H	M	H	H	H	H
	Tipo miss	L	L	L	L		L	L	L		L				
L2	Block#	17	8	2	17		32	4	6		8				
	Index	1	0	2	1		0	0	2		0				
	Tag	4	2	0	4		8	1	1		2				
	Hit/Miss	M	M	M	H		M	M	M		M				
	Tipo miss	L	L	L			L	L	L		Cap.				

La ultima miss in L2 è di Capacità perché anche se L2 fosse fully associative non sarebbe possibile ricordare più di 4 accessi e l'accesso precedente al blocco 8 è avvenuto 6 accessi (a blocchi diversi) prima.

3) calcolate le dimensioni in bit delle due cache L1, L2 compresi i bit di controllo
Scrivete le soluzioni CON I PASSAGGI qui sotto continuando se necessario sul retro del foglio

Dimensioni L1:
dim. blocco = 4 word * 4 byte= 16 byte * 8 bit = 128 bit
offset = log2(16) = 4 bit index = log2(8) = 3 bit tag = 32 – offset – index = 32 – 4 – 3 = 25 bit
dimensioni L1 = 2 ways * 8 set * [V + D + U + blocco + tag] = 2 * 8 * [3 + 128 + 25] = 16 * 156 = **2496 bit**

Dimensioni L2:
dim. blocco = 16 word * 4 byte= 64 byte * 8 bit = 512 bit
offset = log2(64) = 6 bit index = log2(4) = 2 bit tag = 32 – offset – index = 32 – 6 – 2 = 24 bit
dimensioni L2 = 1 ways * 4 set * [V + D + blocco + tag] = 1 * 4 * [2 + 512 + 24] = 4 * 538 = **2152 bit**
NOTA: il bit USED non è presente nelle cache direct-mapped

4) assumendo che il processore vada a **5Ghz** con **4 CPI** (Clock Per Instruction), che gli accessi in memoria impieghino **25ns**, che gli hit nella cache L1 impieghino **1ns** 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.
Scrivete le soluzioni CON I PASSAGGI qui sotto continuando se necessario sul retro del foglio

Tempo totale:
6 hit L1 + 1 hit L2 + 7 miss L2 = 6 * 1ns + 1 * 5ns + 7 * 25ns = 6ns + 5 ns + 175ns = **186 ns**

Tempo medio per accesso:
186 ns / 14 = 13.28 ns

Numero di istruzioni eseguite nel tempo medio:
Se la frequenza di clock è 5 GHz il periodo è 1/5*10^9 = 0.2 ns, e una istruzione impiega 4*0.2ns = 0.8 ns
quindi in 13.28 ns vengono svolte **13.28/0.8 = circa 17 istruzioni**

Esame di Architetture – Canale AL – Prof. Sterbini – 31/1/17

Parte 3 (assembler)

Esercizio 5 (18 punti se iterativo, 30 se ricorsivo).

1) Scrivere una funzione **DIFFERENZA_MEDIA** in linguaggio assemblativo che, ricevendo gli indirizzi di due stringhe **testo1** e **testo2** di stessa lunghezza, conta la media della differenza tra i codici ascii dei caratteri che nelle due stringhe sono nella stessa posizione, ovvero:

- per ogni posizione N nella prima stringa
- dati i due caratteri **X=testo1[N]** e **Y=testo2[N]**
- calcola la media dei valori Z=X-Y usando numeri in virgola mobile

2) Scrivere il programma **main** che:

1. legge la prima stringa (di lunghezza massima 100 caratteri)
2. legge la seconda stringa (della stessa lunghezza)
3. chiama la funzione DIFFERENZA_MEDIA passando gli indirizzi delle due stringhe
4. stampa il risultato

NOTA: la funzione **DIFFERENZA_MEDIA** che scandisce le due stringhe può essere realizzata con un ciclo (**18 punti**) oppure il ciclo può essere simulato con la ricorsione (**30 punti**)

ESEMPI

testo1 = "**abcdef**"

testo2 = "**aaaaaa**"

media = (0 + 1 + 2 + 3 + 4 + 5)/6 = 2.5

testo1 = "**granturco**"

testo2 = "**abcdefghi**"

media = (6 + 16 + -2 + 10 + 15 + 15 + 11 + -5 + 6)/9 = 8.0

Soluzione

Una possibile soluzione iterativa (e anche ricorsiva) che stampa la somma delle differenze ed il numero di caratteri letti potrebbe essere la seguente

```
#################################################################
.data
testo1: .space 101          # allocazione statica di 100 caratteri più 0 terminatore
testo2: .space 101          # allocazione statica di 100 caratteri più 0 terminatore

.text
main:
        li $v0, 8       # read_string
        la $a0, testo1
        li $a1, 100     # num max di caratteri
        syscall

        li $v0, 8       # read_string
        la $a0, testo2
        li $a1, 100     # num max di caratteri
        syscall

        la $a0, testo1
        la $a1, testo2
        jal DIFFERENZA_MEDIA_ric
        # $v0 il numero di caratteri letti
        # $v1 la somma delle differenze tra caratteri
        move $a0, $v0
        li $v0, 1       # print integer
        syscall
        li $v0, 11      # print character
        li $a0, '/'
        syscall
        move $a0, $v1
        li $v0, 1       # print integer
        syscall
        li $v0, 10      # fine programma
        syscall
```

```
#################################################################
# VERSIONE ITERATIVA
# $a0 indirizzo di testo1
# $a1 indirizzo di testo2
# $v0 = somma delle differenze
# $v1 = numero di caratteri letti
DIFFERENZA_MEDIA:
        li $v0, 0               # inizializzazione
        li $v1, 0               # inizializzazione

ciclo:  lb $t0, ($a0)           # leggo il primo carattere
        lb $t1, ($a1)           # leggo il primo carattere

        beqz $t0, finedif               # la stringa è finita se il carattere è '\n' oppure 0
        beqz $t1, finedif
        beq $t0, '\n', finedif
        beq $t1, '\n', finedif

        # se siamo qui le due stringhe non sono finite
        sub $t2, $t0, $t1               # differenza tra i caratteri
        add $v0, $v0, $t2               # aggiorno la somma
        addi $v1, $v1, 1                # aggiorno il conteggio

        addi $a0, $a0, 1                # passo al carattere successivo
        addi $a1, $a1, 1                # in entrambe le stringhe
        j ciclo                         # continuo il ciclo

finedif:                        # finito il ciclo
        jr $ra                          # torno a chi ha chiamato la funzione
```

```
#################################################################
# VERSIONE RICORSIVA
# caso base:      se le due stringhe sono vuote, tornare 0 e 0
# caso ricorsivo: altrimenti,
#                 sommare ai due risultati della chiamata ricorsiva
#                 fatta sul resto delle due stringhe tolto il primo carattere
#                 $v0 += differenza dei primi due caratteri
#                 $v1 ++

# $a0 indirizzo di testo1
# $a1 indirizzo di testo2
# $v0 = somma delle differenze
# $v1 = numero di caratteri letti
DIFFERENZA_MEDIA_ric:
        # preservo su stack i dati che vengono modificati e che sono necessari dopo la chiamata jal
        # $ra, $t0, $t1
        subi $sp, $sp, 12     # alloco 3 word su stack
        sw   $ra, 0($sp)      # salvo $ra perché usando jal lo sovrascriverò
        sw   $t0, 4($sp)      # salvo i due caratteri letti in modo da poterli usare dopo la chiamata jal
        sw   $t1, 8($sp)

        lb $t0, ($a0)               # leggo il primo carattere
        lb $t1, ($a1)               # in entrambe le stringhe
        beqz $t0, casobase          # se sono 0 oppure '\n' siamo nel caso base (fine della stringa)
        beqz $t1, casobase
        beq $t0, '\n', casobase
        beq $t1, '\n', casobase

        # se siamo qui le stringhe non sono finite quindi siamo nel caso ricorsivo
        addi $a0, $a0, 1            # avanzo di un carattere
        addi $a1, $a1, 1            # in entrambe le stringhe
        jal DIFFERENZA_MEDIA_ric        # calcolo la somma delle differenze e il conteggio sul resto
        # $v0 e $v1 del resto delle due stringhe

        sub $t2, $t0, $t1           # diff. caratteri
        add $v0, $v0, $t2           # aggiorno $v0 sommando la differenza dei primi caratteri
        addi $v1, $v1, 1            # aggiorno $v1 sommando 1 carattere
```

```
        # ripristinare i registri salvati
        lw   $ra, 0($sp)
        lw   $t0, 4($sp)
        lw   $t1, 8($sp)
        addi $sp, $sp, 12     # disalloco 3 word su stack

        jr $ra                # torno al chiamante

casobase:                     # quando le stringhe sono vuote
        li $v0, 0             # $v0 = 0
        li $v1, 0             # $v1 = 0

        # ripristinare i registri salvati
        lw   $ra, 0($sp)
        lw   $t0, 4($sp)
        lw   $t1, 8($sp)
        addi $sp, $sp, 12     # disalloco 3 word su stack

        jr $ra                # torno al chiamante

#################################################################################
```