Esame di Architetture – Canale AL – Prof. Sterbini – 8/11/16

Cognome e Nome: _____ Matricola: _____

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

Esercizio 1. Si ha il dubbio che in una partita di CPU a ciclo di clock singolo (vedi sul retro) la Control Unit sia rotta, producendo il segnale di controllo **Branch** attivo **se e solo se NON** è attivo il segnale di controllo **AluSrc**. Assumete che RegDst sia asserito solo per le istruzioni di tipo R, che MemToReg sia asserito solo per l'istruzione lw e che AluSrc sia asserito solo per le istruzioni lw e sw e di tipo immediato.

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

Soluzione

Istruzione	Branch	AluSrc	Risultato
lw	0	1	ok
sw	0	1	ok
R	0 (1)	0	le R saltano relativamente al PC ad una distanza che dipende dai 16 LSB ma solo se il risultato è 0
R immediate	0	1	ok
beq	1	0	ok
J	0 (1)	0	ok perché il secondo mux ignora l'indirizzo di salto relativo calcolato

b) Scrivete *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. Potete usare la sezione .data per definire staticamente il contenuto iniziale della memoria.

Soluzione

E' necessario sfruttare una istruzione R che dà risultato 0 in modo che esegua un salto
Ad esempio l'istruzione **add $s0, $zero, $zero** che è codificata dai campi
oc=0 rs=0 rt=0 rd=**XXXXX** shamt=0, func=100000 (con XXXXX indice del registro destinazione)
e quindi ha la codifica binaria 0000000000000000 **XXXXX**00000100000
e salta avanti o indietro di un numero di istruzioni **XXXXX**00000100000 (in binario)
Supponendo di scegliere un registro di destinazione che va indietro (ad esempio $s0 che ha indice 16=10000)

```
.text
        li $s0, 1               # se salta indietro dopo una serie di nop passerà qui e metterà 1 in $s0
fine:   li $v0, 10
        syscall
main:   add $s0, $zero, $zero   # se rotto salta indietro
        j fine                  # altrimenti mette 0 in $s0 e va a terminare il programma
```

Esercizio 2A. Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro).
Si vuole aggiungere l'istruzione di tipo I **`badd rs, rt, offset`** che esegue **sempre** un salto relativo rispetto al PC di un numero di istruzioni pari alla somma **($rs)+($rt)+offset**

1) modificate il diagramma aggiungendo 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 **133ns**, l'accesso ai registri **66ns**, le operazioni dell'ALU e dei sommatori **200ns**, 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.

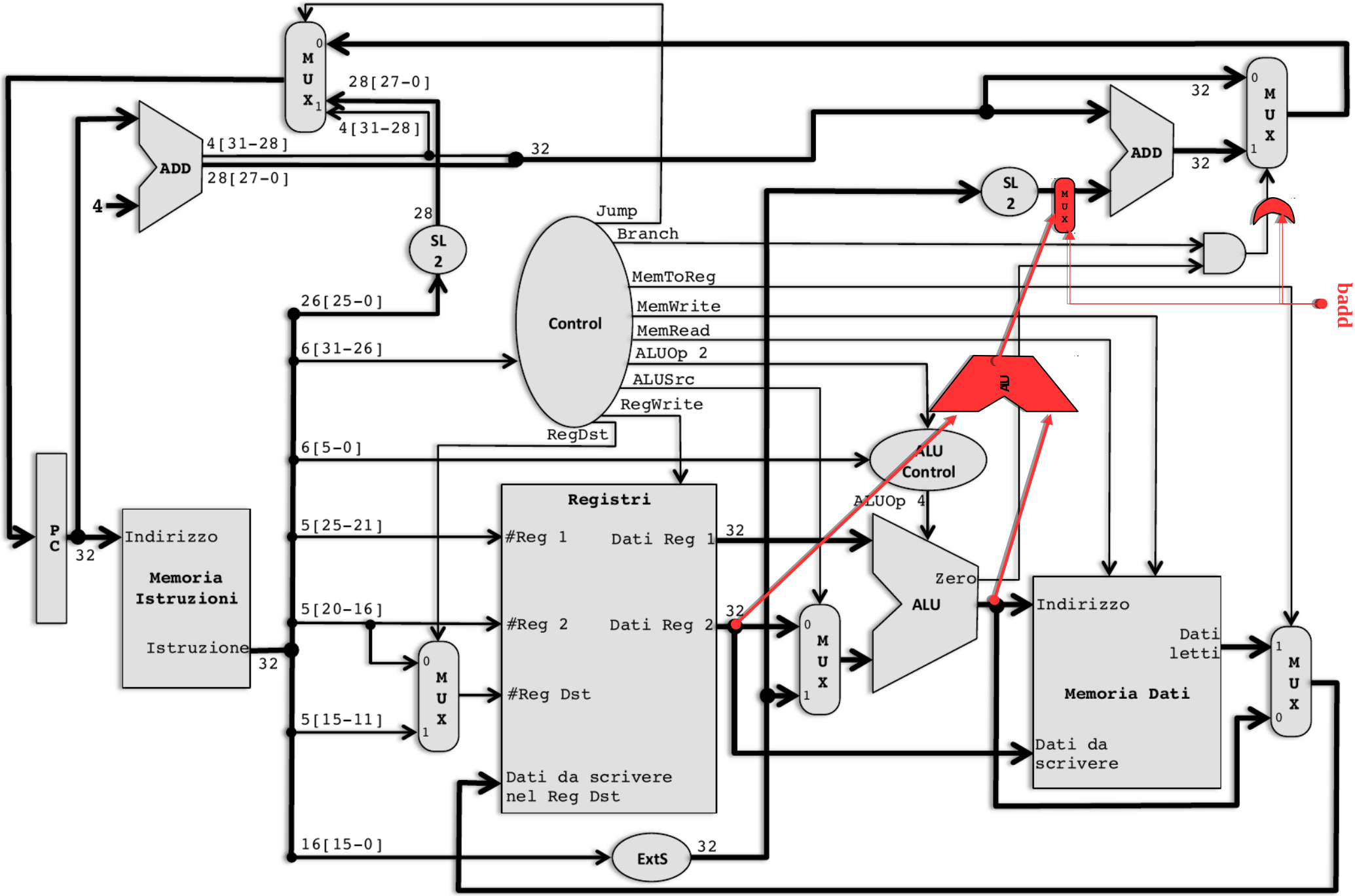
Soluzione: con l'ALU possiamo fare $rs+offset, va aggiunta una seconda alu per sommare $rt, il risultato può essere sommato a PC col sommatore del branch (con un MUX). Il segnale che comanda il mux del salto condizionato dev'essere 1 per questa istruzione, quindi serve una porta OR al suo ingresso con il nuovo segnale **badd** da aggiungere alla CU.

Segnali: Jump=0, Branch=X, MemToReg=X, MemWrite=0, MemRead=X, AluOp=sum, AluSrc=1, RegWrite=0, RegDst=X, badd=1

Tempi: 133ns fetch + 66ns decode + 200ns prima somma + 200ns seconda somma + 200 ns somma per salto relativo (nel frattempo c'è stato tutto il tempo di calcolare PC+4 per il salto relativo) = 799ns

Il periodo di clock dev'essere aumentato.

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 – 8/11/16

Cognome e Nome: _____ Matricola: _____

Parte 2 (per tutti)

Esercizio 3. Si consideri l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il frammento di programma qui a destra che calcola la somma della diagonale di una matrice 10x10 usando puntatori dalla fine all'inizio.

```
.data
M:      .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
        ...
DIM:    .word 10
.text
main:   add  $s0, $0, $0        # somma=0
        lw   $s1, DIM           # N
1 stallo
        addi $s2, $s1, 1        # N+1
        sll  $s2, $s2, 2        # 4(N+1)
        mul  $s1, $s1, $s1      # N^2
        subi $s1, $s1, 1        # N^2-1
        sll  $s1, $s1, 2        # 4(N^2-1)
        la   $s1, M($s1)        # ultimo el
loop:   lw   $t0, ($s1)         # M[i]
1 stallo
        add  $s0, $s0, $t0      # somma+M[i]
        sub  $s1, $s1, $s2      # prox el
        la   $s3, M             # inizio
1 stallo
        bge  $s1, $s3, loop     #
1 stallo su salto verso loop
        li   $v0, 1             # print
        move $a0, $s0           # la somma
        syscall                 #
```

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

Soluzione vedi colori

2) tra quali istruzioni sono presenti control hazard (CH),

Soluzione
solo il salto per loop

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

Soluzione
solo gli stalli indicati
4 fill pipeline +8 istr. +**1 stallo DH**+10*(5 ist.+**2 stalli DH**+1 stallo CH) -1 stallo CH non eseguito alla fine del loop + 3 istr. = 4+8+1+10*(5+2+1)-1+3 = 15 + 80 = 95

4) quanti ne sarebbero necessari se il forwarding non esistesse

Soluzione
Per ogni data hazard ci sono **2 stalli**
4+8+**12**+10*(5+**4**+1) -1 + 3 = 26 + 100 = 126

5) quali sono le istruzioni contenute nei registri della pipeline durante il **26°** ciclo di clock (con FW)

WB: **la** MEM: **stallo** EXE: **bge** ID: **stallo** IF: **lw**

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

Soluzione
tutti gli stalli possono essere rimossi ed una istruzione può essere estratta dal ciclo

- scambiare le prime 2 istruzioni
- spostare l'istruzione `la $s3, M` fuori dal loop
- scambiare nel ciclo le due istruzioni `add $s0, $s0, $t0` e `sub $s1, $s1, $s2`

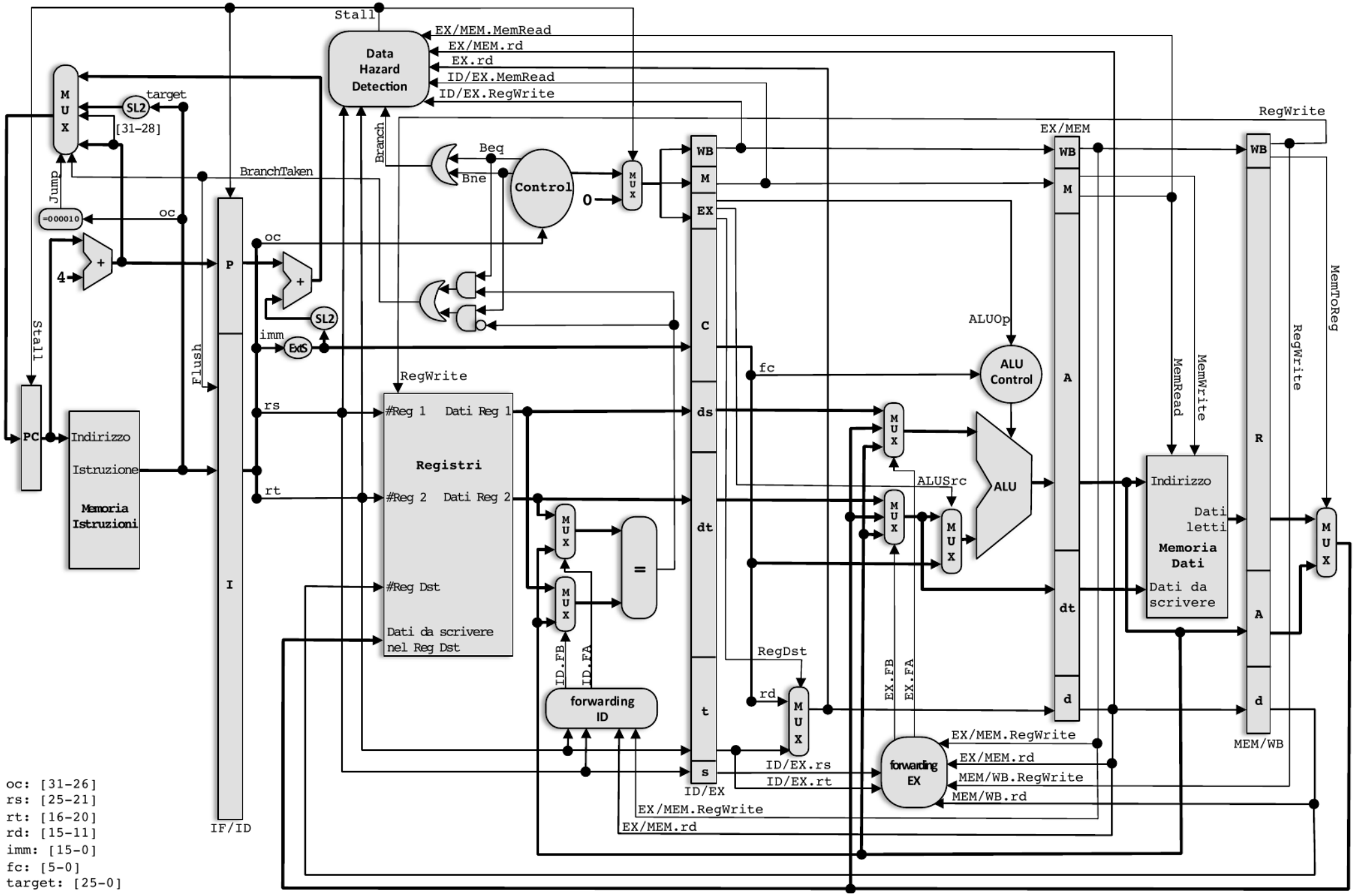
7) quanti ne sarebbero necessari riordinando le istruzioni per ridurre il numero di stalli al massimo

Soluzione
4+9+0+10*(4+0+1) -1 + 3 = 15 + 50 = 65

8) quali sono le istruzioni contenute nei registri della pipeline durante il **24°** ciclo di clock nel codice così ottimizzato

WB: **lw** MEM: **sub** EXE: **add** ID: **bge** IF: **li che diventerà stallo**

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]
```

1°	2°	3°

Correzioni ricevute

Esame di Architetture – Canale MZ – 8/11/16

Inserite qui solo la vostra matricola: _____

Esercizio 4 (14 punti).
Considerate un sistema con un livello di cache e VM con TLB: **CPU <=> L1 <=> MMU <=> RAM**
v
TLB <=> Page Table

- la cache è **2-way** set-associativa con **4 set** e blocchi grandi **4 word** e strategia di rimpiazzo **LRU**.
- il TLB è **direct mapped** con **2 linee** e blocco grande 1 linea di page table.
- la memoria fisica disponibile è di **4 pagine** da **512 byte** ciascuna, con 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, e che la **cache agisca sugli indirizzi VIRTUALI**, indicate quali degli accessi in memoria più sotto sono hit o miss nella cache e nel TLB e quali page fault vengono generati.
2) per ciascuna MISS indicate se è di tipo **Caricamento (L), Capacità (Cap) o Conflitto (Conf)**

Address	540	1050	1090	1800	580	1055	1200	2800	4000	600	5000	1150	5100	6000
#block	33	65	68	112	36	65	75	175	250	37	312	71	318	375
tag	8	16	17	28	9	16	18	43	62	9	78	17	79	93
index	1	1	0	0	0	1	3	3	2	1	0	3	2	3
H/M	M	M	M	M	M	H	M	M	M	M	M	M	M	M
tipo	L	L	L	L	L		L	L	L	L	L	L	L	L
#pag.	1	2	2	3	1		2	5	7	1	9	2	9	11

Mapping di #pagina virtuale in #pagina fisica ottenuto dalla MMU per ogni accesso (riempite tutte le caselle)

#p. virt.	1	2	2	3	1		2	5	7	1	9	2	9	11
#p. fis.	0	1	1	2	0		1	3	2	0	1	3	1	2
P.F.?	Y	Y	N	Y	N		N	Y	Y	N	Y	Y	N	Y

Accessi al TLB **NOTA:** attenti alla interazione tra page fault e TLB

TLB tag	0	1	1	1	0		1	2	3	0	4	1	4	5
TLB index	1	0	0	1	1		0	1	1	1	1	0	1	1
H/M TLB	M	M	H	M	M		H	M	M	M	M	M	H	M
tipo	L	L		L	Cap			L	L	Cap	L	L		L

3) assumendo che il processore vada a **2Ghz** con **3 CPI** (Clock Per Instruction), che gli accessi in memoria impieghino **45ns**, che gli hit nella cache e in TLB impieghino **5ns**, calcolate il **tempo medio** per questa sequenza di accessi e **quante istruzioni** vengono svolte nel tempo calcolato (IGNORANDO IL TEMPO PER PAGE FAULT).
Scrivete qui sotto le soluzioni CON I PASSAGGI continuando se necessario sul retro del foglio
Tempo totale:
Soluzione
per ogni accesso dobbiamo considerare separatamente il tempo di traduzione da indirizzo virtuale a fisico, che dipende dagli hit/miss in TLB ed il tempo di accesso al dato, che dipende dagli hit/miss in cache L1
traduzione = 10 miss TLB + 3 hit TLB = 10*45ns + 3*5ns = 450ns+15ns = 465ns
accesso = 13 miss L1 + 1 hit L1 = 13*45ns + 1*5ns = 585ns + 5ns = 590ns
totale = 465ns + 590ns = 1055ns

Tempo medio: 1055ns / 14 accessi = 75.35ns

Numero di istruzioni: se clock 2Ghz = periodo di 0.5ns, una istruzione viene completata in 3*0.5ns = 1.5ns
Numero di istruzioni nel tempo medio = 75.35ns / 1.5ns = 50 istruzioni circa

Esame di Architetture – Canale AL – Prof. Sterbini – 8/11/16

Cognome e Nome: _____ Matricola: _____

Parte 3 (assembler)

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

1) Scrivere un programma assembly Mips che:
- legge due stringhe di lunghezza massima 100 caratteri ciascuna con l'appropriata syscall
- le confronta lessicograficamente chiamando una funzione separata (vedi punto 2)
- stampa la stringa "prima" oppure "seconda" o "uguali" a seconda se la maggiore è la prima, la seconda o sono uguali

2) Il confronto lessicografico va realizzato come funzione separata che:
- riceve come argomenti i due indirizzi delle due stringhe da confrontare
- torna -1, 1 o 0 a seconda dei casi (prima minore della seconda, uguali, prima maggiore della seconda)

NOTA: per confrontare i caratteri è sufficiente confrontare i corrispondenti byte
NOTA: ricordate che dopo l'ultimo carattere della stringa letta ci può essere '\n' oppure '\0'
NOTA: Il confronto lessicografico si fa un carattere per volta a partire dal primo carattere X ed Y di entrambe le stringhe
- se X = 0 e Y != 0 la prima stringa è minore (0 oppure '\n')
- se X != 0 e Y = 0 la prima stringa è maggiore (0 oppure '\n')
- se X = 0 e Y = 0 le due stringhe sono uguali (0 oppure '\n')
- se X < Y la prima stringa è minore
- se X > Y la prima stringa è maggiore
- se X = Y si passa al prossimo carattere e si continua il confronto

NOTA: Da questa definizione è facile costruire:
- sia una versione iterativa che usa un ciclo per scandire le due stringhe
- che una ricorsiva, che esamina solo il primo carattere delle due stringhe e chiama se stessa solo se X!=0 e Y!=0 e X=Y

Esempi: (in cui sottolineo la coppia di lettere che hanno deciso chi è maggiore)

""	= ""	uguali
"**A**"	> ""	prima
"A**A**"	> "A"	prima
"AAAA**A**"	< "AAAA**B**"	seconda
"AAAA**A**A"	< "AAAA**B**"	seconda
"AA**B**AAA"	> "AA**A**ABA"	prima
"Topolino"	= "Topolino"	uguali
"**t**opolino"	> "**T**opolino"	prima

Soluzione

```
.data
stringa1:  .space 101
stringa2:  .space 101
prima:     .asciiz "prima"
seconda:   .asciiz "seconda"
uguali:    .asciiz "uguali"

.text
main:
      # lettura prima stringa da confrontare
      li    $v0, 8
      la    $a0, stringa1
      li    $a1, 100
      syscall

      # lettura seconda stringa da confrontare
      li    $v0, 8
      la    $a0, stringa2
      li    $a1, 100
      syscall

      # chiamata alla funzione
      la    $a0, stringa1
      la    $a1, stringa2
      jal   confronta

      # controlli per la stampa
      beqz  $v0, print_uguali     # se 0
      bltz  $v0, print_seconda    # se -1
print_prima:                      # se +1
      la    $a0, prima
      j     print
print_seconda:
      la    $a0, seconda
      j     print
print_uguali:
      la    $a0, uguali
print:
      li    $v0, 4
      syscall
      li    $v0, 10
      syscall


# segue
```

```
# confronto ricorsivo
# $a0, indirizzo prima stringa
# $a1, indirizzo seconda stringa
confronta:
      lb    $s0, ($a0) # $s0 = X primo carattere della prima stringa
      lb    $s1, ($a1) # $s1 = Y primo carattere della seconda stringa

      # sostituisco '\n' con 0 per semplificare i controlli dopo
      bne   $s0, '\n', avanti1
      li    $s0, 0
avanti1:
      bne   $s1, '\n', avanti2
      li    $s1, 0
avanti2:

      # la differenza X-Y tra i due caratteri X ed Y può essere:
      # negativa -> la prima lettera è minore
      # positiva -> la prima è maggiore
      # nulla -> le due lettere sono uguali:
      #     - se X=Y=0 le stringhe sono terminate
      #     - altrimenti si passa al prossimo carattere
      sub   $s2, $s0, $s1
      beqz  $s2, uguali_o_finito
      bltz  $s2, fine_minore
fine_maggiore:              # X>Y
      li    $v0, +1
      jr    $ra
fine_minore:                # X<Y
      li    $v0, -1
      jr    $ra
uguali_o_finito:            # X=Y
      bnez  $s0, prossimo_carattere    # se X=Y!=0 si passa al prossimo carattere
      # X=Y=0 le stringhe sono uguali
      li    $v0, 0
      jr    $ra

prossimo_carattere:         # chiamata ricorsiva
      # allocazione stack
      subi  $sp, $sp, 4
      sw    $ra, ($sp)

      # si avanza al prossimo carattere
      addi  $a0, $a0, 1
      addi  $a1, $a1, 1

      # chiamata ricorsiva
      jal   confronta

      # disallocazione stack
      lw    $ra, ($sp)
      addi  $sp, $sp, 4

      # ritorno
      jr    $ra

# fine
```