Esame di Architetture – Canale MZ – Prof. Sterbini – 8/6/15

Cognome e Nome: _____ Matricola: _____

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

Esercizio 1 (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 NON è attivo **MemRead**.
Si assume che **RegDst** sia asserito solo per le istruzioni di tipo R e che **MemRead** e **MemToReg** siano asseriti solo per l'istruzione lw.

---

a) Si indichino *qui sotto* quali delle istruzioni base (**lw, sw, add, sub, and, or, xor, slt, beq, j, li, addi**) funzioneranno male, perché, e quali sono i comportamenti diversi.

Soluzione

La CPU rotta produrrà solo le due combinazioni di segnali MemRead=0 e ALUSrc=1 oppure MemRead=1 e ALUSrc=0 e quindi le istruzioni che funzioneranno male sono quelle che devono avere la coppia di segnali ALUSrc e MemRead uguali:

beq	confronterà $rs con il valore della parte immediata invece che $rt
di tipo R	diventeranno istruzioni immediate
lw	leggerà il dato dall'indirizzo $rs+$rt invece che da $rs + parte immediata

---

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.

Soluzione

Possiamo sfruttare la lw che legge dal posto sbagliato ad esempio con il frammento di programma

```
li    $s0, 1
sw    $zero, 12($zero)     # un qualsiasi offset diverso da 1 e dalle due word vicine
sw    $s0, 1($zero)        # la costante deve essere uguale al valore di $s0
lw    $s0, 12($zero)       # se rotta legge da 1 altrimenti da 12
```

Esercizio 2 (16 punti). Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro). Vogliamo aggiungere l'istruzione di tipo I **`b_odd_sum rs, rt, label`** (branch if sum is odd) che esegue un salto condizionato all'indirizzo (relativo) corrispondente a **`label`** se la somma $rs+$rt è un valore dispari.

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

2) indicate sul diagramma 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 **150ns**, 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

Per determinare se la somma è dispari basta prendere il bit meno significativo (LSB) del risultato dell'ALU. La logica del salto relativo è già presente, per attivarla basta mettere in AND il segnale LSB col (nuovo) segnale b_add_odd da far generare alla CU e quindi metterlo in OR all'ingresso del MUX che abilita il salto relativo.

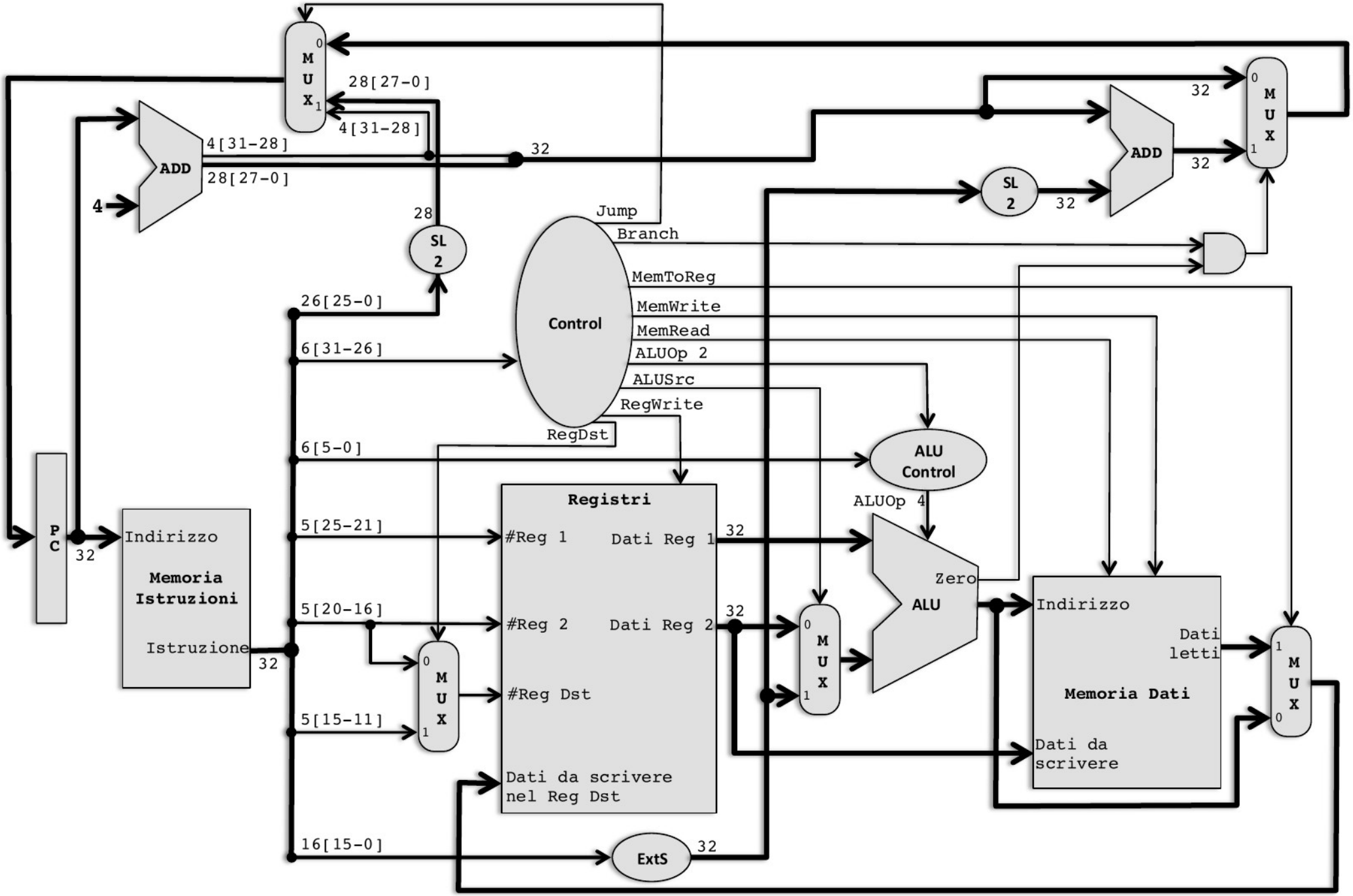
Segnali della CU:

b_add_odd=1, branch=0, jump=0, MemToReg=X, MemWrite=0, MemRead=X, AluOp=ADD, ALUSrc=0, RegWrite=0, RegDst=X

Tempi: (qua sotto due dataflow della istruzione e dell'aggiornamento del PC che avvengono in parallelo)

Fetch = 75ns, ID=25ns, EX=150ns	=> 250ns	Tempo della istruzione = 300ns < 350 della LW
PC+4 = 150ns, Salto relativo= 150ns	=> 300ns	Quindi NON va aumentato il periodo del clock

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 MZ – Prof. Sterbini – 8/6/15

Cognome e Nome: _____ **Matricola:** _____

Parte 2 (per tutti – 2 ore)

Esercizio 3 (16 punti). Considerate l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il frammento di programma qui sotto che calcola l'istogramma della frequenza dei caratteri di un testo .

NOTA: sono presenti solo le unità di forwarding presenti nella figura (sul retro).
NOTA: assumete che tutte le istruzioni usate nel programma siano di base (nessuna pseudoistruzione).

```
.data
TESTO:  .asciiz    "vediamo se oggi me la cavo"
HISTO:  .word      0:256     # istogramma

.text
main:   la    $a1, HISTO(0)  # indir. dell'istogr.
        la    $a0, TESTO(0)  # indir. del testo
  0 stalli>
next:   lb    $t1, ($a0)     # c = TESTO[i]
  2 stalli(la beqz è anticipata alla fase ID)>
        beqz $t1, fine       # se testo finito
  1 stallo solo sul salto>
        add   $t2, $a1, $t1  # addr. HISTO[c]
  0 stalli>
        lw    $t3, ($t2)     # X = HISTO[c]
  1 stalli>
        addi $t3, $t3, 1     # X += 1
  0 stalli>
        sw    $t3, ($t2)     # HISTO[c] = X
        addi $a0, $a0, 1     # addr. prox carattere
  0 stalli sul lb seguente >
        j     next

fine:   li          $v0, 10
        syscall
```

Indicate:
1) tra quali istruzioni sono presenti data hazard,
2) tra quali istruzioni sono presenti control hazard,
3) tra quali istruzioni sono necessari stalli (data e control) (con forwarding)
4) indicate il contenuto della pipeline (quali istruzioni si trovano in quali fasi) alla fine dello **8°** colpo di clock (con forwarding)

stallo in WB
beqz in MEM
add in EXE
lw in ID
stallo in IF

5) quanti cicli di clock sono necessari a eseguire tutto il programma (con forwarding)

4+3+0+26x(8+3)+3+2+1=13+286=299
Le istruzioni eseguite in totale sono
3+26x(8)+3=6+208=214
quindi in media abbiamo 299/214=1.39 CPI

6) quanti ne sarebbero necessari se il forwarding non esistesse

Se il FW non esiste allora ogni data-hazard inserisce sempre 2 stalli tranne quello tra **addi $a0, $a0, 1** e la **lb** che contiene la **j** e quindi inserisce un solo stallo
4+3+2+26x(8+9)+3+2+1=15+ 442 =457 (in media 457/214=2.13 CPI)

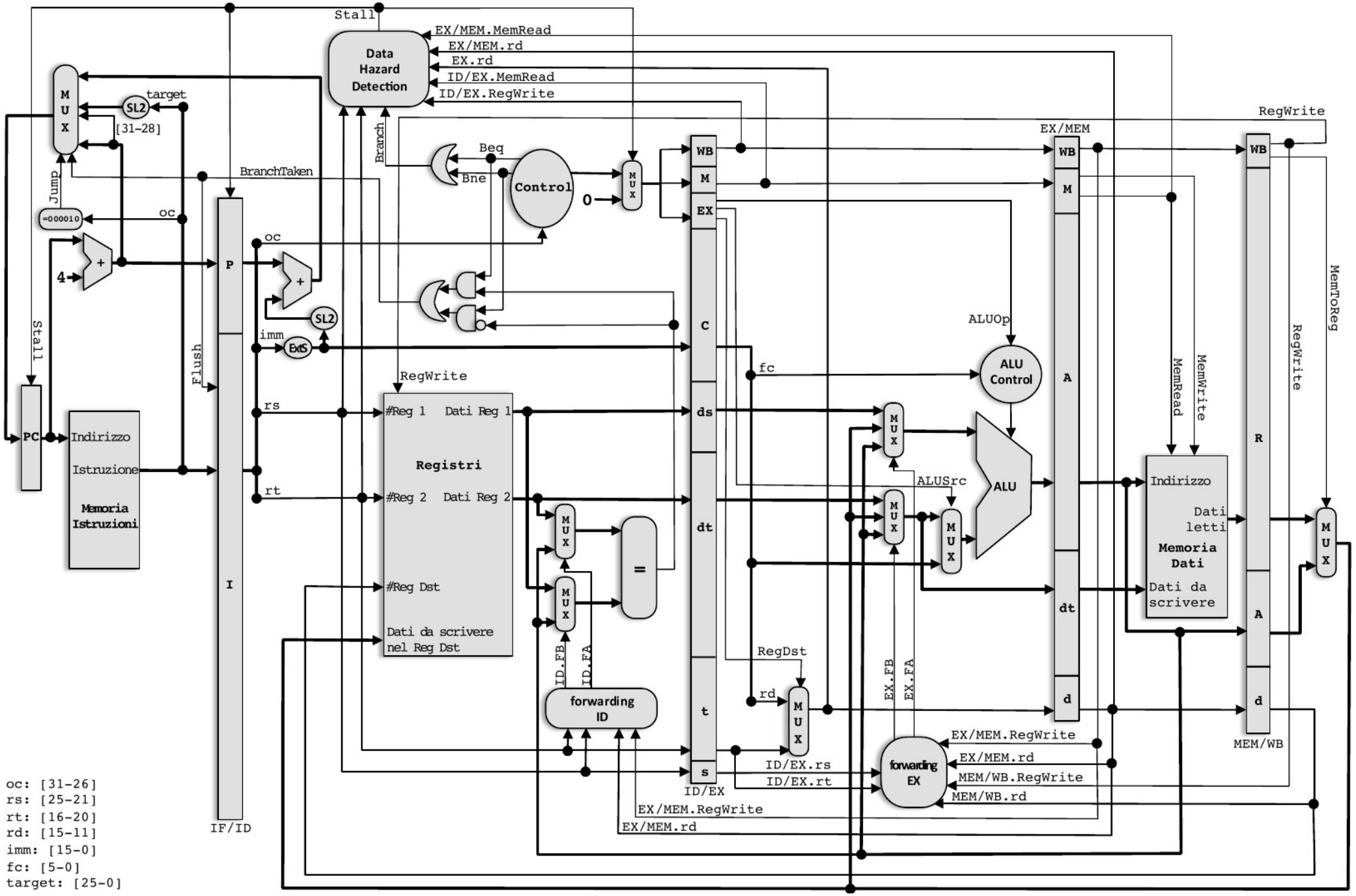
7) riordinate le istruzioni per ridurre al massimo gli stalli (mantenendo invariata la sua semantica)

E' possibile spostare l'istruzione **addi $a0, $a0, 1** indietro ed eliminare 1 stallo tra **lw** e **addi $t3, $t3, 1**
(non conviene spostare **add $t2, $a1, $t1** prima del **beqz** perché la semantica cambia (poco ma cambia))

8) calcolate quanti cicli di clock sono necessari a eseguire il programma così ottimizzato (con forwarding)

4+3+0+26x(8+2)+3+2+1=13+260=273 (in media 273/214=1.27 CPI)

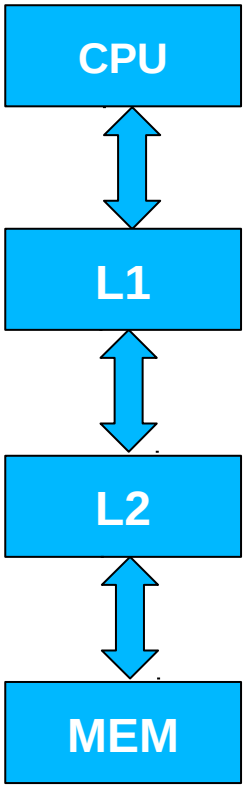
Implementazione pipeline di MIPS (solamente le istruzioni: `add`, `addi`, `sub`, `and`, `andi`, `or`, `ori`, `xor`, `xori`, `nor`, `slt`, `slti`, `lw`, `sw`, `beq`, `bne`, `j`).



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 – Prof. Sterbini – 8/6/15

Esercizio 4 (14 punti). c**ache a due livelli**
Sia data una gerarchia di memoria con due livelli di cachecome in figura,
- La **cache** L1 è
  - **set-associativa** a **2 vie**
  - con **blocchi da 2 word**
  - **4 set** per ogni via
  - politica di sostituzione **LRU**
- La **cache** L2 è
  - **set-associativa** a 4 **vie**
  - con **blocchi da 16 word**
  - **2 set** per ogni via
  - politica di sostituzione **LRU**



1) Per la seguente sequenza di accessi si determinino quali sono gli HIT/MISS sui due livelli di 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 compresi i bit di controllo
4) calcolate il tempo medio di accesso (su questa sequenza) se:
- tempo di HIT su L1 = **1.5 ns**
- tempo di HIT su L2 = **15 ns**
- tempo di accesso a RAM = **75 ns**
5) calcolate il numero di istruzioni corrispondente al tempo medio di accesso, se il processore ha una frequenza di clock di **2 GHz** e completa una istruzione ogni **3** colpi di clock.

.

Indirizzo	**27**	**330**	**123**	**333**	**155**	**91**	**126**	**190**	**107**	**95**	**331**	**812**	**163**
#blocco L1	3	41	15	41	19	11	15	23	13	11	41	101	20
Tag	0	10	3	10	4	2	3	5	3	2	10	25	5
Index	3	1	3	1	3	3	3	3	1	3	1	1	0
HIT/MISS	M	M	M	H	M	M	M	M	M	M	H	M	M
Tipo MISS	L	L	L		L	L	Conf	L	L	Conf		L	L
#blocco L2	0	5	1		2	1	1	2	1	1		12	2
Tag	0	2	0		1	0	0	1	0	0		6	1
Index	0	1	1		0	1	1	0	1	1		0	0
HIT/MISS	M	M	M		M	H	H	H	H	H		M	H
Tipo MISS	L	L	L		L							L	

Tempo: 2 x Hit1 + 6 x Hit2 + 5 x RAM = 2 x 1.5 + 6 x 15 + 5 x 75 = 468 ns
Tempo medio di accesso: 468/13 = 36 ns
Periodo di clock: 1/2*10^9 s = 0.5 ns
Tempo di una istruzione: 3*0.5 = 1.5 ns
Tempo di accesso medio misurato in numero di istruzioni: 36 / 1.5 = 24 istruzioni

Esame di Architetture – Canale MZ – Prof. Sterbini – 8/6/15

Parte 3 (assembler)

Esercizio 5 (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 **ricercaBinaria** che riceve come argomenti:

- **dati:** indirizzo di un vettore contenente elementi interi ORDINATO
- **inizio:** indice del primo elemento del vettore a partire dal quale bisogna cercare
- **N**: il numero di elementi tra cui cercare
- **X:** un numero intero da cercare

e cerca con la ricerca dicotomica la posizione del valore **X** nel vettore **dati** nel segmento [inizio .. inizio+N-1]

e torna

- l'indice della posizione di X in V se X è presente in V
- il valore -1 se X NON è presente

INOLTRE DURANTE LA RICERCA STAMPA LE POSIZIONI DEGLI ELEMNTI ESAMINATI

Esempio

Input:	**N:**	**9**	
	inizio:	**0**	
	dati:	**1, 45, 97, 97, 100, 122, 300, 900, 2000**	
	X:	**97**	
	risultato:	2	(E STAMPA LE POSIZIONI 4 1 2)

Esempio di algoritmo ricorsivo da realizzare:

- se **N=0** non si è trovato il numero, tornare **-1**
- se **N>0** si esamina l'elemento in posizione **Y=inizio+N/2** (divisione intera) E SI STAMPA **Y** SEGUITO DA SPAZIO:
  - se è uguale a **X** si torna la sua posizione **Y**
  - se è minore di **X** si esegue la ricerca sulla metà del vettore che segue la posizione **Y** (inizia dalla posizione **Y+1** e contiene **N-Z-1** elementi, con Z=N/2)
  - se è maggiore di **X** si esegue la ricerca sulla metà del vettore che precede la posizione **Y** (inizia dalla posizione **inizio** e contiene **Z=N/2** elementi)

2) Si realizzi un programma main che usa la funzione **ricercaBinaria** e che:

- legge da stdin:
  - il valore **0<M<=100** di elementi da inserire in un vettore
  - la successione di **M** valori interi ORDINATI e la memorizza nel vettore **dati**
  - il valore **X** da cercare
- chiama la funzione **ricercaBinaria** passando gli argomenti **M,** indirizzo di **dati**, **X** e **inizio=0**
- stampa il risultato

Soluzione

```
.globl main
.data
        DATI: .word 0:100
.text
        # $a0: indirizzo del vettore
        # $a1: indice dell'inizio della parte da cercare
        # $a2: lunghezza L della parte da cercare
        # $a3: elemento da cercare X
ricercaBinaria:
        bgtz $a2, casoRicorsivo
        # altrimenti la lunghezza è 0 e non lo si è trovato
        li   $v0, -1                    # torno -1
        jr   $ra                        # torno dalla funzione a chi l'ha chiamata
casoRicorsivo:
        subi $sp, $sp, 8                # alloco due word su stack ($a0 mi serve per le stampe)
        sw   $ra, 0($sp)                # preservo $ra
        sw   $a0, 4($sp)                # preservo $a0
        srl  $t0, $a2, 1                # Z=L/2
        add  $t1, $t0, $a1              # indice dell'elemento da esaminare Y=inizio+Z
        move $a0, $t1                   # stampo Y
        li   $v0, 1                     # print integer
        syscall
        li   $a0, ' '                   # stampo uno spazio
        li   $v0, 11                    # print char
        syscall
        lw   $a0, 4($sp)                # ripristino $a0 dopo le stampe
        sll  $t2, $t1, 2                # calcolo l'offset corrispondente nel vettore
        add  $t2, $t2, $a0              # indirizzo in memoria dell'elemento
        lw   $t2, ($t2)                 # leggo l'elemento
        beq  $t2, $a3, trovato          # se l'ho trovato ne torno la posizione $t1
        bgt  $t2, $a3, sinistra         # se è maggiore di X cerco a sinistra
        blt  $t2, $a3, destra           # se è minore di X cerco a destra

trovato:
        move $v0, $t1                   # posizione Y dell'elemento trovato nel vettore
ritorno:                                # tutti i casi ricorsivi tornano da qui
        lw   $ra, 0($sp)                # ripristino $ra
        lw   $a0, 4($sp)                # ripristino $a0
        addi $sp, $sp, 8                # disalloco le 2 word da stack
        jr   $ra                        # torno al chiamante
```

```
sinistra:
        move $a2, $t0           # se cerco a sinistra l'inizio resta uguale, la lunghezza è Z=N/2
        jal  ricercaBinaria
        j ritorno               # vado a disallocare stack e ritorno dalla funzione
destra:
        addi $a1, $t1, 1        # se cerco a destra l'inizio è Y+1
        sub  $a2, $a2, $t0      # e il numero di elementi è L-Z-1
        subi $a2, $a2, 1
        jal  ricercaBinaria
        j ritorno               # vado a disallocare stack e ritorno dalla funzione


main:
        li $v0, 5               # read integer
        syscall
        move $t0, $v0           # numero di elementi da leggere
        li   $t1, 0             # indice
ciclo:  li   $v0, 5             # read integer
        syscall
        sll  $t2, $t1, 2        # offset nel vettore
        sw   $v0, DATI($t2)     # memorizzo
        addi $t1, $t1, 1        # prossimo
        blt  $t1, $t0, ciclo    # se ancora non si è finito il ciclo
        la   $a0, DATI          # indirizzo del vettore  DATI
        li   $a1, 0             # si inizia dall'elemento 0
        move $a2, $t0           # L
        li   $v0, 5             # read integer
        syscall
        move $a3, $v0           # X
        jal  ricercaBinaria
        move $t0, $v0           # mi salvo un attimo il risultato
        li   $a0, '\n'          # stampo accapo
        li   $v0, 11            # print char
        syscall
        move $a0, $t0           # recupero il risultato
        li   $v0, 1             # print integer
        syscall
        li   $v0, 10            # end program
        syscall
```