Esame di Architetture – Prof. Sterbini – 28/6/18

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 **MemWrite attivo solo quando** NON sono attivi **né AluSrc né Branch**.
Si assume 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.
NOTA: Assumete che **in caso di don't care i segnali della CU siano tutti 1**

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

	AluSrc	Branch	MemWrite	
lw	1	0	0	
sw	1	0	1→0	le sw NON scrivono in memoria ...
R	0	0	0→1	scrivono $rt in memoria all'indirizzo del risultato ALU
R-imm	1	0	0	
b	0	1	0	
j	x(1)	0	0	

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.

```
.data
valore: .word 1

.text
sw $zero, valore        # provo a sovrascrivere l'1 in memoria
lw $s0, valore          # se è azzerato la CU è OK
```

Esercizio 2. Considerate l'architettura MIPS a ciclo singolo in figura (diagramma sul retro).
Si vuole aggiungere l'istruzione di **tipo R** `sub_2_pow rd, rs, esponente` che sottrae al valore del registro **`rs`** la potenza di 2 il cui esponente è indicato nel campo **shamt** (da 0 a 31) e mette il risultato nel registro **`rd`**.

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 **33ns**, l'accesso ai registri **6ns**, 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.

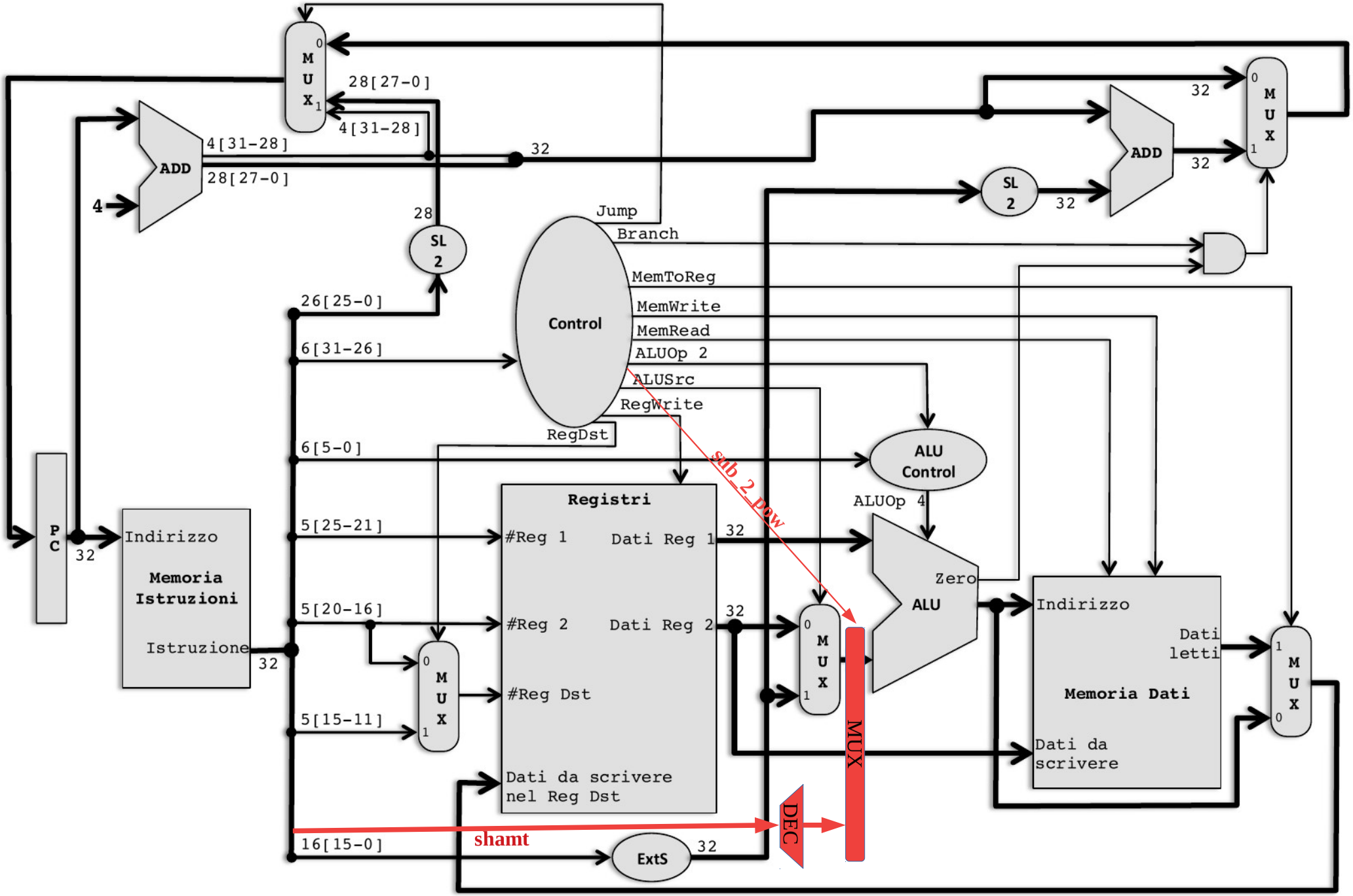
Soluzione

E' necessario aggiungere un decodificatore che trasformi i bit del campo shamt della istruzione nella corrispondente potenza di 2, da usare come secondo argomento per una sottrazione, quindi si inserisce un MUX per fornire questo valore al posto del secondo argomento ALU.

I segnali di controllo sono: Jump=0, Branch=0, RegWrite=1, RegDst=1, MemRead=X, MemWrite=0, AluSrc=X, AluOp=sub, MemToReg=0, sub_2_pow=1

Il tempo totale è: IF=33ns + ID=6ns + EXE=100ns + WB=6ns => 145ns

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



Esame di Architetture – Prof. Sterbini – 28/6/18

Cognome e Nome: _____ Matricola: _____

Parte 2 (per tutti)

Esercizio 3A. Si consideri l'architettura MIPS con pipeline mostrata in figura (sul retro) ed il programma qui a destra che calcola l'altezza massima di un albero binario, rappresentato in memoria come coppie di puntatori ad altri nodi (0 se il figlio sx/dx del nodo non esiste).

NOTA: jal salta in fase IF, jr salta in fase ID

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

2) tra quali istruzioni sono presenti control hazard,

3) quanti cicli di clock sono necessari a eseguire il programma con il forwarding
SUGGERIMENTO: contate prima quante volte viene eseguito il caso base con $a0=0 e quante volte viene eseguito il caso ricorsivo con $a0!=0

4) quanti ne sarebbero necessari se il forwarding non esistesse

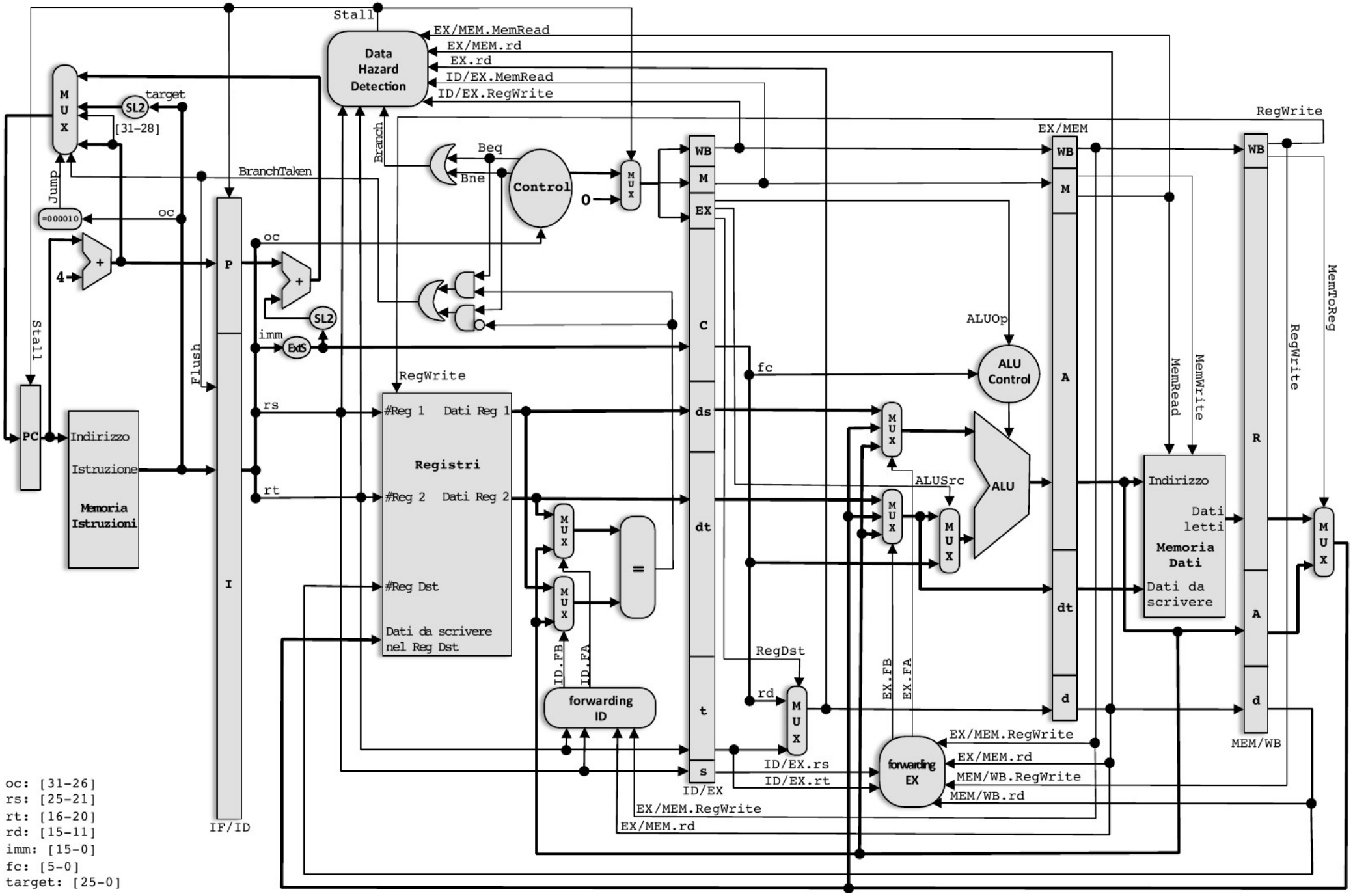
```
# L'albero seguente è di altezza massima 4
#             2
#            /
#       1   3
#        \ /
#         4   5
#          \ /
#           6
.data
Nodo1:    .word 0, 0              # foglia
Nodo2:    .word 0, 0              # foglia
Nodo3:    .word 0, Nodo2          # nodo
Nodo4:    .word Nodo1, Nodo3      # nodo
Nodo5:    .word 0, 0              # foglia
Nodo6:    .word Nodo4, Nodo5      # radice
ALBERO:   .word Nodo6             # address radice
.text
main:     lw   $a0, ALBERO        # lettura radice
          jal  altezzaMassima     # calcolo h max
          move $a0, $v0           # stampa
          li   $v0, 1             # del
          syscall                 # risultato
          li   $v0, 10            # fine
          syscall                 # del programma
---------------------------------------------------
altezzaMassima:
          bnez  $a0, non_nullo    # se il nodo esiste
                                  # else caso base
          move  $v0, $zero        # torno 0
          jr    $ra               # torno al chiamante
non_nullo:                        # se il nodo esiste
          # salvataggio su stack
          subi  $sp, $sp, 12      # alloco 3 word
          sw    $t0, 0($sp)       # salvo su stack
          sw    $a0, 4($sp)       # i 3 registri
          sw    $ra, 8($sp)       # usati
          lw    $a0, ($a0)        # conto il figlio SX
          jal   altezzaMassima
          move  $t0, $v0          # metto da parte hSX
          lw    $a0, 4($sp)       # recupero $a0
          lw    $a0, 4($a0)       # conto il figlio DX
          jal   altezzaMassima
          bgt   $v0, $t0, gia_max# se maggiore salto
          move  $v0, $t0          # o aggiorno il max
gia_max:
          addi  $v0, $v0, 1       # aggiungo 1 al max
          lw    $t0, 0($sp)       # ripristino
          lw    $a0, 4($sp)       # i 3 registri
          lw    $ra, 8($sp)       # da stack
          addi  $sp, $sp, 12      # e disalloco 3 word
          jr    $ra               # torno al chiamante
```

5) quali sono le istruzioni contenute nei registri della pipeline durante il **14°** ciclo di clock (con FW)

WB: MEM: EXE: ID: IF:

6) come riordinare le istruzioni per ridurre il numero di stalli al massimo (con forwarding) (su foglio a parte)
7) quanti colpi di clock sarebbero necessari per 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 – Prof. Sterbini – 28/6/18

Cognome e Nome: _____ 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 L1 è **1-way** set-associativa con **4 set** e blocchi grandi **16 word** e strategia di rimpiazzo **LRU**.
- il TLB è **2-way** set-associativo con **4 set** e dimensioni del blocco **4 linee di page table (ATTENZIONE).**
- la memoria fisica disponibile è di solo **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, indicando per ciascun accesso che lo richiede, la corrispondenza tra pagina virtuale e fisica.
2) per ciascuna MISS indicate se è di tipo **Caricamento (L), Capacità (Cap) o Conflitto (Conf)**

	Address	**5339**	**8000**	**15665**	**10295**	**12899**	**18090**	**15670**	**7990**	**10302**	**7997**	**5330**	**15679**	**18082**	**12910**
L1	**#block**														
	tag														
	index														
	H/M														
	tipo														
M.V.	**#pag. virt.**														
TLB	**#block**														
	tag														
	index														
	H/M														
	tipo														
M.V.	**#pag. fis.**														
	P. fault?														

3) assumendo che il processore vada a **2.5 Ghz** con **4 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 medio calcolato (IGNORANDO IL TEMPO PER PAGE FAULT). Scrivete qui sotto le soluzioni CON I PASSAGGI

Tempo totale:

Tempo medio per ciascun accesso:

Numero di istruzioni che potrebbero essere eseguite nel tempo medio:

Esame di Architetture – Prof. Sterbini – 28/6/18 – Compito A

Cognome e Nome: _____ Matricola: _____

Parte 3 (assembler)

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

Si implementi sia la funzione 1) che il main 2)

1) si realizzi la funzione **RICORSIVA BinarySearch** che riceve l'indirizzo di un vettore ordinato **V** e vi cerca un valore **X** usando l'algoritmo di ricerca binaria, ed eseguendo:

1. ad ogni chiamata della funzione deve stampare, separati da spazio e terminati da accapo:
    1. l'indice **START** del minimo elemento della parte in cui cercare (all'inizio 0)
    2. l'indice **END** del massimo elemento della parte in cui cercare (all'inizio l'ultimo indice del vettore)
    3. l'indice **MID=(START+END)/2** dell'elemento **Y** che confronta col valore **X** cercato
    4. il valore letto **Y**
    5. la stringa "**LEFT**", "**RIGHT**" oppure "**FOUND**" oppure "**MISSING**" a seconda del caso
2. alla fine torna come risultato l'indice dell'elemento trovato (oppure **-1** se non l'ha trovato)

2) si realizzi il programma main che:

1. legge un vettore ordinato in ordine **CRESCENTE** terminato dal valore -1, di max 100 interi
2. legge il valore intero da cercare **X**
3. chiama la funzione **BinarySearch** passandole **X** e l'indirizzo del vettore e gli estremi (ed altri argomenti a piacer vostro)
4. Stampa il valore tornato dalla funzione

Esempio di esecuzione ricorsiva

Se il vettore è formato dai 10 elementi

1 3 5 7 9 11 13 15 17 19

Se X=4 il programma stampa

0 9 4 9 LEFT

0 3 1 3 RIGHT

2 3 2 5 LEFT

2 1 MISSING **(in questo caso non si stampano MID e Y)**

-1

Se invece X=19 il programma stampa

0 9 4 9 RIGHT

5 9 7 15 RIGHT

8 9 8 17 RIGHT

9 9 9 19 FOUND

9

Esame di Architetture – Prof. Sterbini – 28/6/18 – Compito B

Cognome e Nome: _____ Matricola: _____

Parte 3 (assembler)

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

Si implementi sia la funzione 1) che il main 2)

1) si realizzi la funzione **RICORSIVA Merge** che riceve l'indirizzo di due vettori ordinati crescenti **X** e **Y** e le loro dimensioni **M** ed **N** ed un terzo vettore di appoggio **Z** di dimensioni **N+M** (e tutti gli altri argomenti che ritenete necessari) che esegue il passo Merge dell'algoritmo MergeSort, ovvero:

1. se i due vettori sono finiti termina e torna 0
2. se X è finito Z[0] = Y[0] continua ricorsivamente sul resto di Y e di Z e torna 1 più del risultato
3. se Y è finito Z[0] = X[0] continua ricorsivamente sul resto di X e di Z e torna 1 più del risultato
4. se X[0] < Y[0] Z[0] = X[0] continua ricorsivamente sul resto di X e di Z e torna 1 più del risultato
5. altrimenti Z[0] = Y[0] continua ricorsivamente sul resto di Y e di Z e torna 1 più del risultato
6. **NOTA** ad ogni chiamata la funzione deve stampare il valore inserito in Z seguito da spazio
7. e alla fine deve tornare quanti elementi sono stati inseriti in Z

2) si realizzi il programma main che:

1. legge il vettore X ordinato in ordine **CRESCENTE** terminato dal valore -1, di max 100 interi
2. legge il vettore Y ordinato in ordine **CRESCENTE** terminato dal valore -1, di max 100 interi
3. chiama la funzione **Merge** passandole gli indirizzi dei vettori X, Y, le loro dimensioni M ed N e l'indirizzo del vettore Z (grande almeno 200 interi) (ed altri argomenti a piacer vostro)
4. Stampa su una nuova riga il valore tornato dalla funzione (numero di elementi in Z)

Esempio di esecuzione

Se i due vettori letti da tastiera sono

X = 1 3 5 7 9 11 13 15 17 19 -1 **M=10**

Y = 2 4 6 8 -1 **N=4**

Il programma stampa

1 2 3 4 5 6 7 8 9 11 13 15 17 19

14

Se i due vettori letti da tastiera sono

X = -1 **M=0**

Y = -1 **N=0**

Il programma stampa

0