Esercizio 1 (11 punti).

Sia data una CPU con processore a **16GHz** e **4 CPI** (Clock per Instruction) che adoperi indirizzi da 32 bit e memoria strutturata su due livelli di **cache** (L1, L2), il cui setup è come segue:
L1 è una cache **set-associativa** a **3 vie** con **2 set** e **blocchi da 4 word**; adopera una politica di rimpiazzo **LRU**. Ricordiamo che consideriamo il set come l'insieme del blocco in cache con tag e bit di validità, mentre la linea è il gruppo di set con il medesimo indice.
L2 è una cache **direct-mapped** con **4 linee** e **blocchi da 64 word**.

1) Supponendo che all'inizio nessuno dei dati sia in cache, indicare quali degli accessi in memoria indicati di seguito sono HIT o MISS in ciascuna delle due cache. Per ciascuna **MISS** indicare se sia di tipo Cold Start (**Cold**), Capacità (**Cap**) o Conflitto (**Conf**). Utilizzare la tabella sottostante per fornire i risultati ed indicare la metodologia di calcolo più in basso.

	Address	9040	9048	9056	9096	256	264	960	2048	9040	9056	9096	1024
L1	Block#	565	565	566	568	16	16	60	128	565	566	568	64
	Index	1	1	0	0	0	0	0	0	1	0	0	0
	Tag	282	282	283	284	8	8	30	64	282	283	284	32
	HIT/MISS	MISS	HIT	MISS	MISS	MISS	HIT	MISS	MISS	HIT	MISS	MISS	MISS
	Miss type	COLD		COLD	COLD	COLD		COLD	COLD		CONF	CONF	COLD
L2	Block#	35		35	35	1		3	8		35	35	4
	Index	3		3	3	1		3	0		3	3	0
	Tag	8		8	8	0		0	2		8	8	1
	HIT/MISS	MISS		HIT	HIT	MISS		MISS	MISS		MISS	HIT	MISS
	Miss type	COLD				COLD		COLD	COLD		CONF		COLD

2) Calcolare le dimensioni in bit (compresi i bit di controllo ed assumendo che ne basti uno per la LRU) delle due cache: (a) L1 e (b) L2.

L1: $3 \cdot 2 \cdot (128 + 1 + 1 + 27) = 942$ bit

L2: $4 \cdot (2048 + 1 + 22) = 8284$ bit

TOT = 9226 bit

3) Assumendo che gli accessi in **memoria** impieghino **300 ns**, che gli **hit** nella cache **L1** impieghino **3 ns** e gli **hit** nella cache **L2** impieghino **30 ns**, calcolare (a) il **tempo totale** per la sequenza di accessi, (b) il tempo **medio** per la sequenza di accessi, e (c) **quante istruzioni** vengono svolte nel tempo medio calcolato.

$6 \cdot 300 + 3 \cdot 3 + 30 \cdot 3 = 1899$ ns TEMPO TOTALE

$1899 / 12 = 158.25$ ns TEMPO MEDIO

$158.25 \cdot 16 = 2532$ C.C. COLPI MEDI

$2532 / 4 = 633$ ISTRUZIONI MEDIE

4) Calcolare il word offset del sesto indirizzo (264) per la cache L2 spiegando i calcoli effettuati.

$$\text{WORD OFFSET} = \left\lfloor \frac{\text{ADR \% BYTE X BLOCCO}}{4} \right\rfloor = \left\lfloor \frac{264 \% 256}{4} \right\rfloor = 2$$

5) Supponendo che gli indirizzi nella tabella siano virtuali e la memoria virtuale consti di **1024 pagine** di **2KiB** ciascuna, indicarne i numeri di pagina virtuale. Si assume che la cache sia a monte della memoria virtuale.

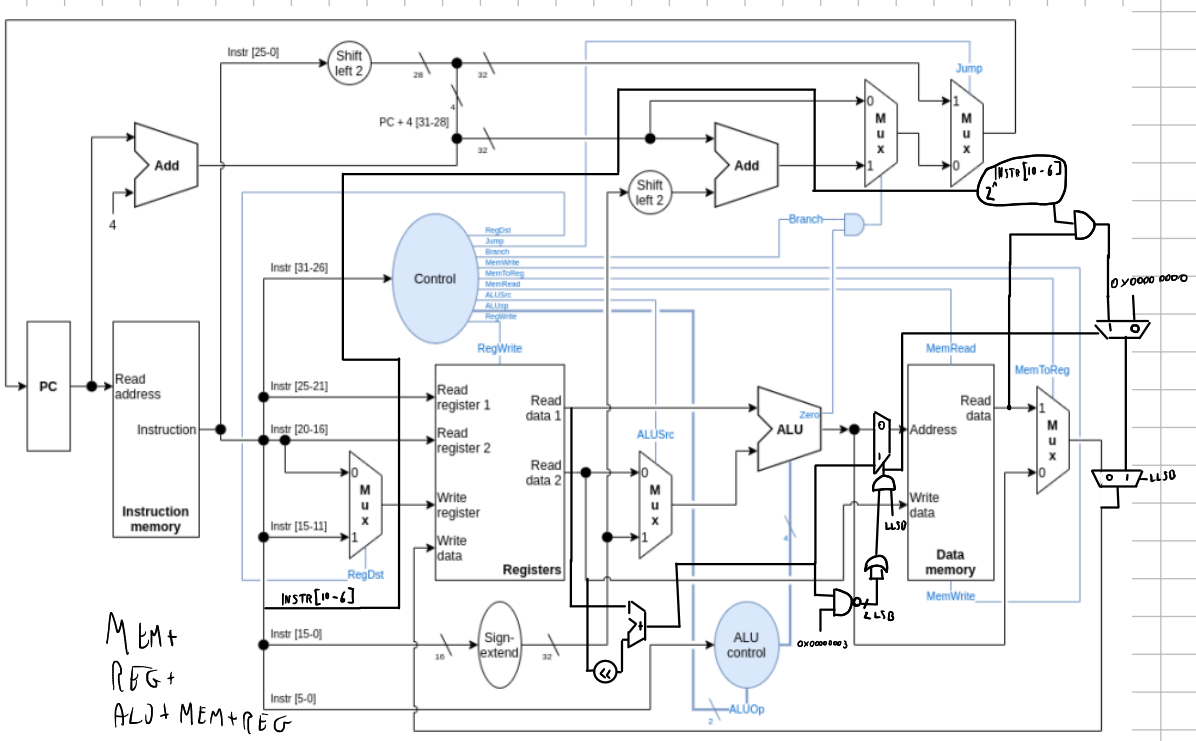
Address	9040	9048	9056	9096	256	264	960	2048	9040	9056	9096	1024
Page#	4	4	4	4	0	0	0	1	4	4	4	0

Esercizio 2 (11 punti).

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere alla CPU l'istruzione **load LSBs** (`llsb`), di **tipo R** e sintassi assembly
`llsb $base_reg, $offset_reg, $out_reg, lsb_select`
(dove `base_reg`, `offset_reg`, `out_reg` e `lsb_select` sono rispettivamente nei campi `rs`, `rt`, `rd` e `shamt` dell'istruzione) che:
1) calcola l'indirizzo per il caricamento di una word da 32 bit considerando il valore `$base_reg` come base e il valore `$offset_reg` come spiazzamento
2) carica la word da 32 bit come segue:
2.1) se l'indirizzo di cui al punto (1) è un multiplo di 4, allora carica la word da memoria a quell'indirizzo;
2.2) se invece l'indirizzo di cui al punto (1) non è un multiplo di 4, allora imposta quella word a `0x0000 0000`;
3) filtra i 32 bit della parola al punto (2) tenendo tanti least significant bit quanti indicati da `lsb_select` (e ponendo i most significant bit restanti a `0`);
4) carica in registro `$out_reg` la parola presa da (2).

Esempio: Supponiamo che `$base_reg` sia `$s0`, contenente `0x1001 1010`, che `$offset_reg` sia `$s1`, contenente `0x0000 000C`, che `$out_reg` sia `$v0` e che `lsb_select` sia 9.
L'indirizzo risultante dal calcolo di cui al punto (1) è multiplo di quattro. Pertanto viene caricata la parola da memoria a quell'indirizzo (supponiamo: `0000 1111 0000 1111 0000 1111 0000 1010`$_{due}$). In `$v0` ne vengono salvati solo i 9 LSB (quelli appena evidenziati in grigio).
Il valore di `$v0` diviene pertanto: `0000 0000 0000 0000 0001 0000 1010`$_{due}$.
Si noti che se l'indirizzo di cui al punto (1) non fosse stato un multiplo di quattro (ad es., se `$base_reg` fosse stato pari a `0x1001 1011`), a `$v0` sarebbe stato assegnato il valore `0x0000 0000`.

1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.



REGDST = LLSB = REGWRITE = MEMREAD = 1

TUTTI GLI ALTRI D.C.

MEMWRITE = JUMP = BRANCH = 0

2) Indicare il contenuto in bit della word che esprime l'istruzione
`llsb $s0, $s1, $v0, 9`
compilando la tabella sottostante (assumiamo che lo *OpCode* di `llsb` sia `0x3E` mentre il campo *func* sia `0x37`)

1	1	1	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	1	0	1	1	1

3) Supponendo che l'accesso alle **memorie** impieghi **400 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU** e dei **sommatori 200 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati.*

si accede alla memoria ISTRUZIONI (400 ns), DOPO AI REGISTRI (50 ns), I DAT dai registri passano per un sommatore (200 ns), poi per la memoria (400 ns) ed infine nei registri per il WRITE BACK (50 ns)

400 + 50 + 200 + 400 + 50 = 1100 ns

IL CICLO DI CLOCK NON VA ALLUNGATO

4) Indicando con `Llsb` il segnale di controllo che viene asserito per eseguire la nuova istruzione, assumiamo che
a) tutti i segnali di tipo *don't care* siano pari a `0` e che
b) la Control Unit della CPU MIPS modificata per supportare `Llsb` sia difettosa e sovrascriva il segnale `RegDst` come segue:
`RegDst` ← `1 - Llsb` (il simbolo ← denota che la variabile a sinistra assume il valore dato della variabile a destra)

In tal caso, indicare quale valore sia assegnato a `$v0` al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `$v0`, `$s0` e `$` siano inizializzati a `0`. Motivare la propria risposta.

```
        addi $s1,  $0, 0xA
        add  $s0,  $s0, $s1
        beq  $s1,  $s0, Out
        j    Exit
Out:    subi $s0,  $s0, 0x9
Exit:   addi $v0,  $s0, 0x5
```

LLSB È SEMPRE 0, QUINDI REGDST È SEMPRE 1, QUINDI, IL REGISTRO DI SCRITTURA È SEMPRE INSTR[15-11]

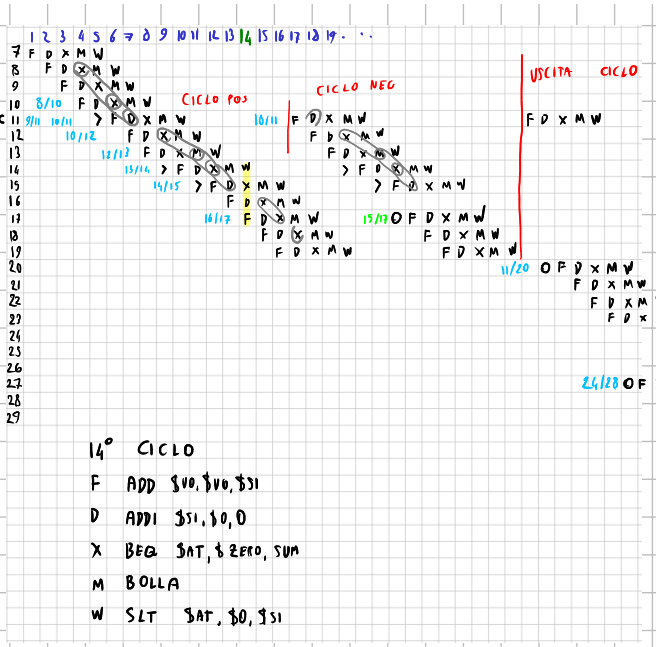
1° $S1 = 0$
2° SCRIVE IN $S0 \leftarrow S0 + S1$ OSSIA $0 + 0 = 0$
3° $0 = 0$ QUINDI VA AD OUT
4°
5° SCRIVE NEL REG. $0 \leftarrow S0 + S0$
6° SCRIVE NEL REG. $0 \leftarrow S0 + S0$

$V0 = 0$

Esercizio 3 (11 punti)

Si consideri l'architettura MIPS con pipeline mostrata nella figura in basso (e in allegato). Il programma qui di seguito somma i numeri negativi nella sequenza fornita.

```
1  .data
2  A: .byte 2, 2, -7, -3, 3
3  L: .byte 5
4  S: .asciiz "\nThe result is bigger than 0!"
5  # Risultato atteso: -10
6  .text
7  main: la   $a0, A            # Indirizzo base array input
8        and  $v0, $zero, $0    # $v0: risultato
9        lb   $a2, L            # Lunghezza dell'array
10       sub  $a1, $v0, $v0     # $a1: Indice corrente
11 cycl: beq  $a1, $a2, exit    # Array scorso? Esci
12       add  $t1, $a0, $a1     # $t1: Indirizzo corrente
13       lb   $s1, ($t1)        # $s1: Elemento corrente
14       slt  $at, $0, $s1      # $at vale 1 se $s1 > 0
15       beq  $at, $zero, sum   # Se $s1 <= 0, somma
16       addi $s1, $0, 0        # Altrimenti, azzera
17 sum:  add  $v0, $v0, $s1     # Somma $s1
18       addi $a1, $a1, 1       # Passa a cella successiva
19       j    cycl              # Riprendi il ciclo
20 exit: add  $s0, $v0, $0      # Salva risultato in $s0
21       addi $v0, $0, 1        # Imposta stampa interi
22       add  $a0, $s0, $0      # Imposta stampa ris.
23       syscall                # Stampa
24       blt  $s0, $zero, term  # Se $s0 > 0, stampa
25       la   $a0, S            # Carica la stringa S
26       addi $v0, $zero, 4     # Imposta stampa
27       syscall                # Stampa
28 term: addi $v0, $0, 10       # Imposta uscita
29       syscall                # Esci
```



14° CICLO

F ADD $v0, $v0, $s1

D ADDI $s1, $0, 0

X BEQ $at, $zero, sum

M BOLLA

W SLT $at, $0, $s1

CON FORWARDING

	I	DH	CH	X	T
C.P.	4	0	0	1	4
P.C.	4	1	0	1	5
C.+	9	2	0	3	33
C.-	8	2	1	2	22
USC.	1	0	0	1	1
POST	7	0	2	1	9
					74

SENZA FOR.

	I	DH	CH	X	T	
C.C.	4	0	0	1	4	
PRE	4	3	0	1	7	
C+	9	9	0	3	53	← 54 MA NEL PRIMO CICLO HA 1 STALLO IN MENO
C-	8	7	1	2	32	
USC.	1	1	0	1	2	
POST	7	0	2	1	10	
					108	

DATA HAZARD

M	N	J
8	10	V0
9	11	A2
10	11	A1
10	12	A1
12	13	T1
13	14	S1
14	15	AT
16	17	S1
18	11	A1

CONTROL HAZARD

M	N
15	17
11	20
24	28

6) Si descriva nel maggior dettaglio possibile il funzionamento dei segnali di controllo ForwardA e ForwardB nella fase EXE del diagramma seguente.

I due segnali ForwardA e ForwardB si occupano di selezionare quali valori (eventualmente passati tramite forward da istruzioni precedenti) debbano essere argomento dell'ALU (gestiscono la fase di EXE) . ForwardA = primo argomento dell'ALU, ForwardB = secondo argomento dell'ALU. Essi sono entrambi composti da 2 bit, possono valere 00, 01 o 10, se valgono 00, entrambi manderanno come argomento all'ALU il valore della loro stessa istruzione ma dalla fase precedente, senza quindi utilizzare il forwarding. Se ForwardA o ForwardB valgono 01, come argomento dell'ALU, manderanno il valore preso dall'istruzione precedente in fase di MEM. Se valgono 10, come argomento dell'ALU, manderanno il valore preso dall'istruzione precedente in fase di EXE.