Politecnico di Milano
Dip. di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri **prof.ssa Donatella Sciuto**
prof. Gerardo Pelosi **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

PRIMA PARTE – martedì 30 agosto 2022

Cognome_____ **Nome** _____

Matricola_____**Firma** _____

Istruzioni

Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.

Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta, se staccati, vanno consegnati intestandoli con nome e cognome.

È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.

Non è possibile lasciare l'aula conservando il tema della prova in corso.

Tempo a disposizione 1 h : 30 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (6 punti) _____

esercizio 2 (2 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – linguaggio macchina

prima parte – traduzione da C a linguaggio macchina

Si deve tradurre in linguaggio macchina simbolico (assemblatore) *MIPS* il frammento di programma C riportato sotto. Il modello di memoria è quello **standard *MIPS*** e le variabili intere sono da **32 bit**. Non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. Si facciano le ipotesi seguenti:

- il registro "frame pointer" *fp* **non è in uso**
- le variabili locali sono allocate nei registri, se possibile
- vanno **salvati** (a cura del chiamante o del chiamato, secondo il caso) **solo i registri necessari**

Si chiede di svolgere i quattro punti seguenti (usando le varie tabelle predisposte nel seguito):

1. **Si descriva** il segmento dei dati statici dando gli spiazzamenti delle variabili globali rispetto al registro global pointer *gp*, e **si traducano** in linguaggio macchina le dichiarazioni delle variabili globali.
2. **Si descriva** l'area di attivazione della funzione `chksum`, secondo il modello MIPS, e l'allocazione dei parametri e delle variabili locali della funzione `chksum` usando le tabelle predisposte.
3. **Si traduca** in linguaggio macchina **il codice degli statement riquadrati nella funzione** `main`.
4. **Si traduca** in linguaggio macchina il codice **dell'intera funzione** `chksum` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali                                    */
#define N 7
int code = 18
char LETTERS [N]

/* funzione chksum                                                 */
int chksum (char * byte, int weight) {
   int idx
   int * ptr
   int partial
   ptr = &partial
   *ptr = 0
   for (idx = N - 1; idx >= 0; idx--) {
      *ptr = *ptr + byte [idx] - weight
      weight++
   } /* for */
   return *ptr
}  /* chksum */

/* programma principale                                            */

int main ( ) {
   code = chksum (LETTERS, code)
}  /* main */
```

punto 1 – segmento dati statici (numero di righe non significativo)

contenuto simbolico	indirizzo assoluto iniziale (in hex)	spiazzamento rispetto a *gp* = **0x 1000 8000**	
…	*…*	*…*	indirizzi alti
LETTERS [N – 1]	*0x 1000 000A*	*0x 800A*	
…	*…*	*…*	
LETTERS [1]	*0x 1000 0005*	*0x 8005*	
LETTERS [0]	*0x 1000 0004*	*0x 8004*	
CODE	*0x 1000 0000*	*0x 8000*	indirizzi bassi

punto 1 – codice MIPS della sezione dichiarativa globale (numero di righe non significativo)

```
            .data    0x 1000 0000     // segmento dati statici standard
            .eqv     N,7              // costante N = 7
CODE:       .word    18               // varglob CODE (inizializzata a 18)
LETTERS:    .space   7                // varglob LETTERS (7 byte)
```

punto 2 – area di attivazione della funzione **CHKSUM**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
$s0 salvato	*+8*	indirizzi alti
$s1 salvato	*+4*	
varloc PARTIAL	*0*	*← sp (fine area)*
		indirizzi bassi

La funzione CHKSUM è foglia, dunque essa non salva in pila il registro ra. I registri $s0, $s1 (callee-saved) vanno salvati in pila, dato che vengono usati per allocare, rispettivamente, le variabili locali idx e ptr. La variabile locale partial va allocata in pila, poiché è puntata da ptr e deve avere un indirizzo di memoria. Complessivamente l'area di attivazione di CHKSUM ingombra tre interi (cioè 12 byte).

punto 2 – allocazione dei parametri e delle variabili locali di **CHKSUM** nei registri	
parametro o variabile locale	**registro**
byte	*$a0*
weight	*$a1*
idx	*$s0*
ptr	*$s1*

punto 3 – codice MIPS degli statement riquadrati in **MAIN** (num. righe non significativo)
`MAIN:  //  code = chksum (LETTERS, code)`
`la  $a0, LETTERS  // prepara param byte`
`lw  $a1, CODE  // prepara param weight`
`jal  CHKSUM  // chiama funz CHKSUM`
`sw  $v0, CODE  // aggiorna varglob CODE`

punto 4 – codice MIPS della funzione **CHKSUM** (numero di righe non significativo)			
CHKSUM:	**addiu**	$sp, $sp, -12	// **COMPLETARE** - crea area attivazione
	// direttive EQV e salvataggio registri – **DA COMPLETARE**		
	.eqv	*S0, 8*	*// spi di reg $s0*
	.eqv	*S1, 4*	*// spi di reg $s1*
	.eqv	*PARTIAL, 0*	*// spi di varloc PARTIAL*
	sw	*$s0, S0($sp)*	*// salva reg $s0*
	sw	*$s1, S1($sp)*	*// salva reg $s1*
	// ptr = &partial		
	mv	*$t0, $sp*	*// calcola ind di varloc PARTIAL*
	addi	*$t0, $t0, PARTIAL*	*// somma spi di varloc PARTIAL*
	mv	*$s1, $t0*	*// aggiorna varloc PTR*
*// NB: poiché qui PARTIAL è = 0, si può ottimizzare con **mv** $s1, $sp*			
	// *ptr = 0		
	sw	*$zero, $s1*	*// aggiorna varloc puntata da PTR*
	// for (idx = N - 1; idx >= 0; idx--)		
	li	*$s0, N - 1*	*// inizializza varloc IDX*
FOR:	***blt***	*$s0, $zero, ENDFOR*	*// se IDX < 0 vai a ENDFOR*
	// *ptr = *ptr + byte [idx] - weight		
	lw	*$t0, ($s1)*	*// carica varloc puntata da PTR*
	addu	*$t1, $a0, $s0*	*// calcola ind di elem BYTE[IDX]*
	lw	*$t2, ($t1)*	*// carica elem NUM[IDX] in TMP*
	add	*$t0, $t0, $t2*	*// calcola subexpr ... + ...*
	sub	*$t0, $t0, $a1*	*// calcola subexpr ... - ...*
	sw	*$t0, ($s1)*	*// aggiorna varloc puntata da PTR*
	// weight++		
	addi	*$a1, $a1, 1*	*// aggiorna param WEIGHT*
	// idx--		
	subi	*$s0, $s0, 1*	*// aggiorna varloc IDX*
	j	*FOR*	*// torna a FOR*
ENDFOR:			
	// return *ptr		
	lw	*$v0, ($s1)*	*// prepara valusc*
	// chiusura funzione – **NON RIPORTARE**		

seconda parte – assemblaggio e collegamento

Dati i due moduli assemblatore seguenti, **si compilino** le tabelle relative a:

1. i due moduli oggetto MAIN e TASK
2. le basi di rilocazione del codice e dei dati di entrambi i moduli
3. la tabella globale dei simboli e la tabella del codice eseguibile

modulo MAIN

```
          .eqv      CONST, 0x14A7B92D
          .data
WEIGHT:   .word     30
REF:      .word     40
          .text
          .globl    MAIN
MAIN:     addi  $a0, $s0, 0
          li    $a1, CONST
FUNCT:    jal   TASK
          beq   $v0, $zero, NEXT
          sw    $v0, WEIGHT
NEXT:     sw    $v0, LOCAL
          j     MAIN
```

modulo TASK

```
          .data
RESERVED: .space    24
LOCAL:    .word     0
          .text
          .globl    TASK
TASK:     bne   $a0, $zero, AFTER
          andi  $a0, $a1, 0x 1234
          sw    $a0, REF
AFTER:    beq   $s0, $a0, NEXT
          jr    $ra
```

Regola generale per la compilazione di **tutte** le tabelle contenenti codice:

- i codici operativi e i nomi dei registri vanno indicati in formato simbolico
- tutte le costanti numeriche all'interno del codice vanno indicate in esadecimale, con o senza prefisso 0x, e di lunghezza giusta per il codice che rappresentano

  esempio: un'istruzione come ***addi $t0, $t0, 15*** è rappresentata: ***addi $t0, $t0, 0x000F***
- nei moduli oggetto i valori numerici che non possono essere indicati poiché dipendono dalla rilocazione successiva, vanno posti a zero e avranno un valore definitivo nel codice eseguibile

(1) – moduli oggetto

modulo `MAIN`

dimensione testo: *20 hex (32 dec)*

dimensione dati: *08 hex (8 dec)*

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** $a0, $s0, 0
4	**lui** $a1, *0x 14A7*
8	**addi** $a1, $a1, *0x B92D*
C	**jal** ***0x 000 0000***
10	**beq** $v0, $zero, *0x0001 = +1*
14	**sw** $v0, ***0x 0000 ($gp)***
18	**sw** $v0, ***0x 0000 ($gp)***
1C	**j** ***0x 000 0000***
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto
0	*0x 0000 001E*
4	*0x 0000 0028*

tabella dei simboli
tipo può essere *T* (testo) oppure *D* (dato)

simbolo	tipo	valore (hex)
WEIGHT	*D*	*0x 0000 0000*
REF	*D*	*0x 0000 0004*
MAIN	*T*	*0x 0000 0000*
FUNCT	*T*	*0x 0000 000C*
NEXT	*T*	*0x 0000 0018*

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
C	***jal***	*TASK*
14	***sw***	*WEIGHT*
18	***sw***	*LOCAL*
1C	***j***	*MAIN*

modulo `TASK`

dimensione testo: *14 hex (20 dec)*

dimensione dati: *1C hex (28 dec)*

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**bne** $a0, $zero, *0x 0002 = +2*
4	**andi** $a0, $a1, 0x 1234
8	**sw** $a0, ***0x 0000 ($gp)***
C	**beq** $s0, $a0, ***0x 0000***
10	**jr** $ra
14	
18	
1C	
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto
0	*non specificato*
18	*0x 0000 0000*

tabella dei simboli
tipo può essere *T* (testo) oppure *D* (dato)

simbolo	tipo	valore (hex)
RESERVED	*D*	*0x 0000 0000*
LOCAL	*D*	*0x 0000 0018*
TASK	*T*	*0x 0000 0000*
AFTER	*T*	*0x 0000 000C*

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
8	***sw***	*REF*
C	***beq***	*NEXT*

(2) – posizione in memoria dei moduli	
modulo `MAIN`	**modulo `TASK`**
base del testo: **`0x 0040 0000`**	base del testo: *`0x 0040 0020`*
base dei dati: **`0x 1000 0000`**	base dei dati: *`0x 1000 0008`*

(3) – tabella globale dei simboli				
simbolo	**valore finale (hex)**		**simbolo**	**valore finale (hex)**
`WEIGHT`	*`0x 1000 0000`*		**`RESERVED`**	*`0x 1000 0008`*
`REF`	*`0x 1000 0004`*		**`LOCAL`**	*`0x 1000 0020`*
`MAIN`	*`0x 0040 0000`*		**`TASK`**	*`0x 0040 0020`*
`FUNCT`	*`0x 0040 000C`*		**`AFTER`**	*`0x 0040 002C`*
`NEXT`	*`0x 0040 0018`*			

NELLA TABELLA DEL CODICE ESEGUIBILE SI CHIEDONO SOLO LE ISTRUZIONI DEI MODULI MAIN E TASK CHE ANDRANNO COLLOCATE AGLI INDIRIZZI SPECIFICATI

(3) – codice eseguibile	
testo	
indirizzo (hex)	**codice** (con codici operativi e registri in forma simbolica)
`...`	`...`
`C`	***`jal  0x 010 0008`*** *`// MAIN: jal  TASK`*
`...`	
`14`	***`sw`*** *`$v0,`* ***`0x 8000 ($gp)`*** *`// MAIN: sw  $v0, WEIGHT`*
`18`	***`sw`*** *`$v0,`* ***`0x 8020 ($gp)`*** *`// MAIN: sw  $v0, LOCAL`*
`1C`	***`j  0x 010 0000`*** *`// MAIN: j  MAIN`*
`...`	
`28`	***`sw`*** *`$a0,`* ***`0x 8004 ($gp)`*** *`// TASK: sw  $a0, REF`*
`2C`	***`beq`*** *`$s0, $a0,`* ***`0x FFFA = -6`*** *`// TASK: beq  $s0, $a0, NEXT`*
`...`	`...`

esercizio n. 2 – logica digitale

logica sequenziale

Sia dato il circuito sequenziale composto da due bistabili master-slave di *tipo D* (D1, Q1 e D2, Q2, dove D è l'ingresso del bistabile e Q è lo stato / uscita del bistabile), un ingresso **I** e un'uscita **U**, e descritto dalle equazioni nel riquadro.

D1 = I xor Q1

D2 = I nand Q2

U = D1 xor (not Q2)

Si chiede di completare il diagramma temporale riportato qui sotto. Si noti che:

- si devono trascurare completamente i ritardi di propagazione delle porte logiche e i ritardi di commutazione dei bistabili
- i bistabili sono il tipo master-slave la cui struttura è stata trattata a lezione, con uscita che commuta sul fronte di discesa del clock

---

tabella dei segnali (diagramma temporale) da completare

- per i segnali D1, Q1, D2, Q2 e U, **si ricavi**, per ogni ciclo di clock, l'andamento della forma d'onda corrispondente riportando i relativi valori 0 o 1
- notare che nel primo intervallo i segnali Q1 e Q2 sono già dati (rappresentano lo stato iniziale)

I	0	1	1	1	1	0	0	0	0	1	0	1	1	0	0	0	0	1	1	1	1	1	0	1	0	0	1	1
D1	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*1*
Q1	0	0	0	0	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
D2	*1*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*
Q2	0	0	0	0	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
U	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*	*1*	*1*
CLK	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1

esercizio n. 3 – microarchitettura del processore pipeline

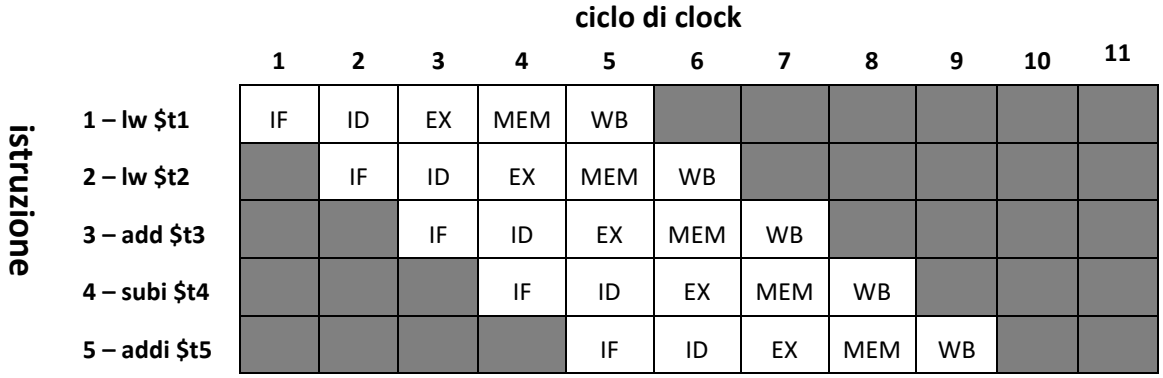
prima parte – pipeline e segnali di controllo

Sono dati il seguente frammento di codice **macchina** MIPS (simbolico), che inizia l'esecuzione all'indirizzo indicato, e i valori iniziali per alcuni registri e parole di memoria.

indirizzo	codice MIPS
0x 0040 0800	lw $t1, 0x 1B29($t0)
0x 0040 0804	lw $t2, 0x 193A($t3)
0x 0040 0808	add $t3, $t3, $t3
0x 0040 080C	subi $t4, $t1, 32
0x 0040 0810	addi $t5, $t2, 4
0x 0040 0814	

registro	contenuto iniziale
$t0	0x 1001 2E9B
$t1	0x 1001 ABAB
$t2	0x 1001 ABCD
$t3	0x 1001 2E8A
memoria	**contenuto iniziale**
0x 1001 4004	0x 1001 AB40
0x 1001 47C4	0x 1001 1A1A *($t2 finale)*
0x 1001 49C4	0x 1001 A0A0 *($t1 finale)*

La pipeline è ottimizzata per la gestione dei conflitti di controllo, e si consideri il **ciclo di clock 5** in cui l'esecuzione delle istruzioni nei vari stadi è la seguente:



1) Calcolare il valore dell'indirizzo di memoria dati nell'istruzione *lw $t1* (prima load):

0x 1001 2E9B + 0x 0000 1B29 = 0x 1001 49C4 _____

2) Calcolare il valore dell'indirizzo di memoria dati nell'istruzione *lw $t2* (seconda load):

0x 1001 2E8A + 0x 0000 193A = 0x 1001 47C4 _____

3) Calcolare il valore del risultato ($t3 + $t3) dell'istruzione *add* (addizione):

0x 1001 2E8A + 0x 1001 2E8A = 0x 2002 5D14 ($t3 finale) _____

4) Calcolare il valore del risultato ($t1 - 32) dell'istruzione *subi* (sottrazione con immediato):

0x 1001 A0A0 ($t1 finale) - 0x 0000 0020 = 0x 1001 A080 ($t4 finale) ____

5) Calcolare il valore del risultato ($t2 + 4) dell'istruzione *addi* (addizione con immediato):

0x 1001 1A1A ($t2 finale) + 0x 0000 0004 = 0x 1001 1A1E ($t5 finale) ____

Completare le tabelle.

I campi di tipo *Istruzione* e *NumeroRegistro* possono essere indicati in forma simbolica, tutti gli altri in esadecimale (prefisso 0x implicito). Utilizzare **n.d.** se il valore non può essere determinato. N.B.: **tutti** i campi vanno completati con valori simbolici o numerici, tranne quelli precompilati con *******.

segnali all'ingresso dei registri di interstadio (subito prima del fronte di SALITA del clock --- ciclo 5)			
IF	**ID**	**EX**	**MEM**
(addi)	*(subi)*	*(add)*	*(lw $t2)*
registro IF/ID	*registro ID/EX*	*registro EX/MEM*	*registro MEM/WB*
	.WB.MemtoReg *0*	.WB.MemtoReg *0*	.WB.MemtoReg *1*
	.WB.RegWrite *1*	.WB.RegWrite *1*	.WB.RegWrite *1*
	.M.MemWrite *0*	.M.MemWrite *0*	
	.M.MemRead *0*	.M.MemRead *0*	
	.M.Branch *0*	.M.Branch *0*	
.PC *0x 0040 0814*	.PC *0x 0040 0810*	.PC **********	
.istruzione *addi*	.(Rs) *0x 1001 A0A0 ($t1 finale)*		
	.(Rt) **********	.(Rt) **********	
	.Rt *$t4*	.R *$t3*	.R *$t2*
	.Rd **********		
	.imm/offset esteso *0000 0020*	.ALU_out *0x 2002 5D14 ($t3 finale)*	.ALU_out **********
	.EX.ALUSrc *1*	.Zero *0*	.DatoLetto *0x 1001 1A1A ($t2 finale)*
	.EX.RegDest *0*		

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 *$t1 addi*	RF.DatoLetto1 *0x 1001 ABAB ($t1 iniz.)*	RF.RegScrittura *$t1 lw*
RF.RegLettura2 **********	RF.DatoLetto2 **********	RF.DatoScritto *0x 1001 A0A0 ($t1 finale)*

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 6)		
RF.RegLettura1 *$t2 addi*	RF.DatoLetto1 *0x 1001 ABCD ($t2 iniz.)*	RF.RegScrittura *$t2 lw*
RF.RegLettura2 **********	RF.DatoLetto2 **********	RF.DatoScritto *0x 1001 1A1A ($t2 finale)*

seconda parte – gestione di conflitti e stalli

Si consideri la sequenza di istruzioni sotto riportata eseguita in modalità pipeline:

		ciclo di clock									
	istruzione	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	`lw $s1, 0x AA($s0)`	IF	ID *0*	EX	MEM	WB *1*					
2	`lw $s2, 0x BB($s0)`		IF	ID *0*	EX	MEM	WB *2*				
3	`add $s3, $s1, $s2`			IF	ID *1, 2*	EX	MEM	WB *3*			
4	`subi $s4, $s3, 8`				IF	ID *3*	EX	MEM	WB *4*		
5	`addi $s5, $s4, 4`					IF	ID *4*	EX	MEM	WB *5*	

La pipeline è ottimizzata per la gestione dei conflitti di controllo**.**

punto 1

a. Definire **tutte** le dipendenze di dato completando la **tabella 1** della pagina successiva (colonne ***punto 1A***), indicando quali generano un conflitto, e per ognuna di queste quanti stalli sarebbero necessari per risolvere tale conflitto (stalli teorici), **considerando la pipeline senza percorsi di propagazione.**

b. Disegnare in **diagramma A** il diagramma temporale della pipeline senza propagazione di dato, con gli stalli **effettivamente** risultanti, e riportare il loro numero in **tabella 1** (colonne ***punto 1B***).

diagramma A

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1. lw	*IF*	*ID 0*	*EX*	*M*	*WB (1)*											
2. lw		*IF*	*ID 0*	*EX*	*M*	*WB (2)*										
3. add			*IF*	*ID stall*	*ID stall*	*ID 1, 2*	*EX*	*M*	*WB (3)*							
4. subi				*IF stall*	*IF stall*	*IF*	*ID stall*	*ID stall*	*ID 3*	*EX*	*M*	*WB (4)*				
5. addi							*IF stall*	*IF stall*	*IF*	*ID stall*	*ID stall*	*ID 4*	*EX*	*M*	*WB (5)*	

punto 2

Si faccia l'ipotesi che la pipeline sia **ottimizzata** e dotata dei percorsi di propagazione: **EX / EX**, **MEM / EX** e **MEM / MEM**:

a. Disegnare in **diagramma B** il diagramma temporale della pipeline, indicando **i percorsi di propagazione** che devono essere attivati per risolvere i conflitti e gli eventuali **stalli** da inserire affinché la propagazione sia efficace.

b. Indicare in **tabella 1** (colonne ***punto 2B***) i percorsi di propagazione attivati con gli stalli associati, e il ciclo di clock nel quale sono attivi i percorsi di propagazione.

diagramma B

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. lw	*IF*	*ID 0*	*EX*	*M (1)*	*WB (1)*										
2. lw		*IF*	*ID 0*	*EX*	*M (2)*	*WB (2)*									
3. add			*IF*	*ID stall*	*ID 1,2*	*EX (3)*	*M (3)*	*WB (3)*							
4. subi				*IF stall*	*IF*	*ID 3*	*EX (4)*	*M (4)*	*WB (4)*						
5. addi						*IF*	*ID 4*	*EX (5)*	*M (5)*	*WB (5)*					

tabella 1

punto 1A					punto 1B	punto 2B	
n° istruzione	**n° istruzione da cui dipende**	**registro coinvolto**	**conflitto (si / no)**	**n° stalli teorici**	**n° stalli effettivi**	**stalli + percorso di propagazione**	**ciclo di clock in cui è attivo il percorso**
3	*1*	*$s1*	*sì*	*1*	*assorbito*	*assorbito*	*–*
3	*2*	*$s2*	*sì*	*2*	*2*	*1 stallo + MEM / EX*	*6*
4	*3*	*$s3*	*sì*	*2*	*2*	*EX / EX*	*7*
5	*4*	*$s4*	*sì*	*2*	*2*	*EX / EX*	*8*

esercizio n. 4 – domande su argomenti vari

memoria cache

Si consideri un sistema di memoria costituito da una memoria centrale di **16 M parole** e da una cache dati set-associativa (associativa a gruppi o a insiemi) a **4 vie** di **8 K parole** con blocchi da **32 parole**, che utilizza un algoritmo di sostituzione del blocco di tipo **LRU**.

La cache dati è inizialmente vuota. La CPU esegue un ciclo che preleva sequenzialmente dalla memoria un array di **8256** elementi di una parola, partendo dall'indirizzo **0**. Il ciclo viene eseguito per **5** volte.

Si risponda alle domande seguenti.

1) **Si mostri** la struttura dell'indirizzo di memoria.

etichetta	indice	spiazzamento
13 bit	*6 bit*	*5 bit*

La memoria ha 16 M parole: 24 bit per l'indirizzo.

I blocchi sono da 32 parole: 5 bit per identificare la parola nel blocco.

Nella cache (8 KB) ci sono: $2^{13} / 2^5 = 2^8 = 256$ blocchi. La memoria cache è set-associativa a quattro vie, dunque avrà $2^8 / 2^2 = 2^6 = 64$ gruppi (o insiemi) di quattro blocchi ciascuno. Sono pertanto necessari 6 bit per individuare il gruppo. L'etichetta è dunque di 24 bit – 5 bit – 6 bit = 13 bit.

2) **Si indichi** il numero di MISS che si verificano durante la prima esecuzione del ciclo, spiegandone il motivo.

n. di miss nella prima iterazione del ciclo: 258

Nel primo ciclo si verificano 258 miss, tanti quanti sono i blocchi del vettore, poiché inizialmente la memoria cache è vuota.

La cache si riempie completamente quando si arriva a caricare il blocco 255, che apparterrà al gruppo 63 (111111).

Quando bisogna caricare il blocco numero 256, si osserva che esso ha i 6 bit di gruppo pari a 000000, dunque il suo "posto" è nel gruppo 000000: ma tutto il gruppo è occupato, dunque si verifica un miss e con l'algoritmo LRU verrà rimosso il blocco 0.

In modo analogo il blocco 257 (…0001 000001) finirà nel gruppo 000001, dove verrà rimosso il blocco 1.

3) **Si calcoli** il numero di MISS che si verificano durante **ciascuna iterazione del ciclo dopo la prima,** mostrandone chiaramente il calcolo.

n. di miss per ciascuna iterazione del ciclo dopo la prima: 10

Nella seconda iterazione, quando serve il blocco 0 si verifica un miss e nella cache viene rimosso il blocco 64; quando serve il blocco 1, si verifica un miss e viene rimosso il blocco 65.

Invece il blocco 2 e i successivi sono già in memoria. Avremo hit fino al blocco 63, quando si verificano 2 miss di seguito e andranno a sostituire i blocchi 128, 129, e così via, hit fino quando si arriva al blocco 127 e poi 2 miss, idem fino al blocco 191 e poi 2 miss, e idem fino al blocco 255 e poi 2 miss, che vanno infine a sostituire i blocchi 0, 1, con 256, 257. In totale si verificano dunque 10 miss nella seconda iterazione del ciclo. La situazione si riproduce nella terza iterazione, nella quarta e nella quinta.

spazio libero per continuazione o brutta copia

spazio libero per continuazione o brutta copia