



**Politecnico di Milano**

**Dip. di Elettronica, Informazione e Bioingegneria**

**prof. Luca Breveglieri**  
**prof. Gerardo Pelosi**

**prof.ssa Donatella Sciuto**  
**prof.ssa Cristina Silvano**

## **AXO – Architettura dei Calcolatori e Sistemi Operativi**

**Prova di lunedì 8 novembre 2021**

**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 2 h : 00 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 rosso)**

## 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 accoppiare 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 spiazziamenti 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 *vsign*, secondo il modello MIPS, e l'allocazione dei parametri e delle variabili locali della funzione *vsign* usando le tabelle predisposte
3. **Si traduca** in linguaggio macchina **il codice dello statement riquadrato nella funzione** *main*.
4. **Si traduca** in linguaggio macchina il codice **dell'intera funzione** *vsign* (vedi tab. 4 strutturata).

```
/* costanti e variabili globali */
#define N 5
int VECTOR [N]
int signature = 0

/* testate funzioni ausiliarie - ambo sono funzioni foglia */
int getstdin ( ) /* legge un intero da standard input */
int checksum (int *) /* calcola sommario (hash) di vettore */
/* funz. vsign - legge e firma vettore tramite una chiave */
int vsign (int key, int * base) {
    int count
    int * hash
    hash = base
    count = N
    do {
        count--
        VECTOR [count] = getstdin ( ) + count
    } while (count != 0) /* do */
    *hash = checksum (base)
    return (*hash - key)
} /* vsign */

/* programma principale */
int main ( ) {
    signature = vsign (getstdin ( ), VECTOR)
} /* main */
```

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

contenuto simbolico	indirizzo assoluto iniziale (in hex)	spiazzamento rispetto a <b><i>gp</i> = 0x 1000 8000</b>	
			indirizzi alti
...	...	...	
SIGNATURE	<i>0x 1000 0014</i>	<i>0x 8014</i>	
VECTOR [4]	<i>0x 1000 0010</i>	<i>0x 8010</i>	
...	...	...	
VECTOR [0]	<i>0x 1000 0000</i>	<i>0x 8000</i>	indirizzi bassi

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

<b>.data</b>	0x 1000 0000	// segmento dati statici standard
<b>.eqv</b>	<i>N, 5</i>	
<b>VECTOR:</b>	<b>.space</b> 20	// varglob VECTOR (5 interi cioè 20 byte)
<b>SIGNATURE:</b>	<b>.word</b> 0	// varglob SIGNATURE inizializzata

punto 2 – area di attivazione della funzione VSIGN		
contenuto simbolico	spiazz. rispetto a stack pointer	
<i>\$ra salvato</i>	<i>+8</i>	indirizzi alti
<i>\$s0 salvato</i>	<i>+4</i>	
<i>\$s1 salvato</i>	<i>0</i>	← <i>sp (fine area)</i>
<i>(\$a0 salvato)</i>		← <i>max estens. pila</i>
		indirizzi bassi

La funzione VSIGN non è foglia, dunque essa 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 locale count e hash. Dato che il registro \$a0 viene modificato da VSIGN prima di chiamare la funzione CHECKSUM e viene usato da VSIGN dopo la chiamata a CHECKSUM, esso va salvato in pila prima di modificarlo e poi va ripristinato subito dopo la chiamata a CHECKSUM; pertanto la pila viene dinamicamente estesa (cioè viene estesa in corso di esecuzione solo quando serve) di una parola. Complessivamente l'area di attivazione di VSIGN ingombra tre interi (cioè **12 byte**), e dinamicamente si estende di un ulteriore intero (solo quando si chiama CHECKSUM).

punto 2 – allocazione dei parametri e delle variabili locali di VSIGN nei registri	
parametro o variabile locale	registro
<i>key</i>	<i>\$a0</i>
<i>base</i>	<i>\$a1</i>
<i>count</i>	<i>\$s0</i>
<i>hash</i>	<i>\$s1</i>

<b>punto 3</b> – codice MIPS dello statement riquadrato in <b>MAIN</b> (num. righe non significativo)		
// signature = vsign (getstdin ( ), VECTOR)		
MAIN:	<b>jal</b>	GETSTDIN // chiama funz GETSTDIN (senza arg)
	<b>move</b>	\$a0, \$v0 // prepara param KEY di funz VSIGN
	<b>la</b>	\$a1, VECTOR // prepara param BASE di funz VSIGN
	<b>jal</b>	VSIGN // chiama funz VSIGN
	<b>sw</b>	\$v0, SIGNATURE // aggiorna varglob SIGNATURE

**punto 4 – codice MIPS della funzione vsign (numero di righe non significativo)**

```
VSIGN:    addiu    $sp, $sp, -12        // COMPLETARE - crea area attivazione
// direttive EQU e salvataggio registri - NON VANNO RIPORTATI
// hash = base
move     $s1, $a1                    // inizializza varloc HASH

// count = N
li       $s0, N                      // inizializza varloc COUNT

DO:       // do
// count--
subi     $s0, $s0, 1                 // aggiorna varloc COUNT

// VECTOR [count] = getstdin ( ) + count
jal      GETSTDIN                    // chiama funz GETSTDIN
add      $t0, $v0, $s0               // calcola espr getstdin ( ) + count
la       $t1, VECTOR                 // carica ind elem VECTOR [0]
sll      $t2, $s0, 2                 // allinea indice COUNT
addu     $t1, $t1, $t2               // calcola ind elem VECTOR [COUNT]
sw       $t0, ($t1)                  // aggiorna elem VECTOR [COUNT]

// while (count != 0)
bne      $s0, $zero, DO              // se COUNT != 0 vai a DO

// *hash = checksum (base)
addiu    $sp, $sp, -4                // estendi pila
sw       $a0, ($sp)                  // push reg a0
move     $a0, $a1                    // prepara param BASE funz CHECKSUM
jal      CHECKSUM                     // chiama funz checksum
lw       $a0, ($sp)                  // pop reg a0
addiu    $sp, $sp, +4                // riduci pila
sw       $v0, ($s1)                  // aggiorna oggetto puntato da hash

// return (*hash - key)
lw       $t3, ($s1)                  // carica oggetto puntato da hash
sub      $v0, $t3, $a0               // prepara valusc funz VSIGN

// rientro
addiu    $sp, $sp, +12               // elimina area attivazione
jr       $ra                          // rientra a chiamante
```

## seconda parte – assemblaggio e collegamento

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

1. i due moduli oggetto MAIN e AUXILIARY
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		modulo AUXILIARY	
	<code>.data</code>		<code>.data</code>
BUF:	<code>.space 28</code>		<code>.eqv CONST, 5</code>
	<code>.text</code>	SUM:	<code>.word 10</code>
	<code>.globl MAIN</code>		<code>.text</code>
MAIN:	<code>move \$a0, \$zero</code>		<code>.globl AUX</code>
	<code>lw \$a1, SUM</code>	AUX:	<code>beq \$a0, \$a1, SKIP</code>
	<code>jal AUX</code>		<code>jr \$ra</code>
	<code>bne \$v0, \$zero, MAIN</code>	SKIP:	<code>addi \$a0, \$a0, CONST</code>
	<code>move \$t0, \$v0</code>		<code>sw \$a0, BUF</code>
	<code>addi \$t0, \$t0, 1</code>		<code>jr \$ra</code>
	<code>sw \$t0, BUF</code>		
	<code>j MAIN</code>		

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			modulo AUXILIARY			
dimensione testo: 20 hex (32 dec)			dimensione testo: 14 hex (20 dec)			
dimensione dati: 1C hex (28 dec)			dimensione dati: 04 hex (4 dec)			
testo			testo			
indirizzo di parola	istruzione (COMPLETARE)		indirizzo di parola	istruzione (COMPLETARE)		
0	ori	\$a0, \$zero, 0x 0000	0	beq	\$a0, \$a1, 0x 0001	
4	lw	\$a1, 0000 (\$gp)	4	jr	\$ra	
8	jal	000 0000	8	addi	\$a0, \$a0, 0x 0005	
C	bne	\$v0, \$zero, 0x FFFC	C	sw	\$a0, 0000 (\$gp)	
10	ori	\$t0, \$v0, 0x 0000	10	jr	\$ra	
14	addi	\$t0, \$t0, 0x 0001	14			
18	sw	\$t0, 0000 (\$gp)	18			
1C	j	000 0000	1C			
20			20			
24			24			
28			28			
2C			2C			
dati			dati			
indirizzo di parola	contenuto		indirizzo di parola	contenuto		
0	non specificato		0	0x 0000 000A		
tabella dei simboli tipo può essere T(testo) oppure D(dato)			tabella dei simboli tipo può essere T(testo) oppure D(dato)			
simbolo	tipo	valore	simbolo	tipo	valore	
BUF	D	0x 0000 0000	SUM	D	0x 0000 0000	
MAIN	T	0x 0000 0000	AUX	T	0x 0000 0000	
			SKIP	T	0x 0000 0008	
tabella di rilocalizzazione			tabella di rilocalizzazione			
indirizzo di parola	cod. operativo	simbolo	indirizzo di parola	cod. operativo	simbolo	
4	lw	SUM	C	sw	BUF	
8	jal	AUX				
18	sw	BUF				
1C	j	MAIN				

*Il simbolo CONST dichiarato tramite .eqv è una pura costante e si può omettere dalla tabella simboli (la direttiva .eqv ha lo stesso significato di #define in C, e non ha rilevanza ai fini della rilocalizzazione).*



(2) – posizione in memoria dei moduli			
modulo MAIN		modulo AUXILIARY	
base del testo:	0x 0040 0000	base del testo:	0x 0040 0020
base dei dati:	0x 1000 0000	base dei dati:	0x 1000 001C

(3) – tabella globale dei simboli				
simbolo	valore finale		simbolo	valore finale
BUF	0x 1000 0000		SUM	0x 1000 001C
MAIN	0x 0040 0000		AUX	0x 0040 0020
			SKIP	0x 0040 0028

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

(3) – codice eseguibile	
testo	
indirizzo	codice (con codici operativi e registri in forma simbolica)
...	...
4	lw \$a1, 0x 801C(\$gp) // MAIN: lw \$a1, SUM
8	jal 010 0008 // MAIN: jal AUX
C	bne \$v0, \$zero, 0x FFFC // MAIN: bne \$v0, \$zero, MAIN
...	...
18	sw \$t0, 0x 8000(\$gp) // MAIN: sw \$t0, BUF
1C	j 010 0000 // MAIN: j MAIN
...	...
20	beq \$a0, \$a1, 0x 0001 // AUXILIARY: beq \$a0, \$a1, SKIP
28	addi \$a0, \$a0, 0x 0005 // AUXILIARY: addi \$a0, \$a0, CONST
2C	sw \$a0, 0x 8000(\$gp) // AUXILIARY: sw \$a0, BUF
...	...

## esercizio n. 2 – logica digitale

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), e dotato di un ingresso **I** e un'uscita **U**. Il circuito è descritto dalle equazioni nel riquadro.

$$D1 = \neg(Q2 \text{ or } I)$$

$$D2 = Q1 \text{ xor } \neg(Q2 \text{ and } I)$$

$$U = Q2 \text{ and } I$$

N.B. operatore XOR, uscita a 1 se e solo se solo uno degli ingressi è a 1

**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
- a solo scopo di chiarezza, per il segnale di ingresso I è riportata anche la forma d'onda per mettere in evidenza la corrispondenza tra questa e i valori 0 e 1 presenti nella tabella dei segnali complessiva
- si noti che nel primo intervallo i segnali Q1 e Q2 sono già dati (rappresentano lo stato iniziale)

I	0	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	1	1	
D1	1	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Q1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
D2	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0	0	1	1	1	1	
Q2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	
U	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	
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	registro	contenuto iniziale
0x 0040 0800	lw \$t1, 0x 001A(\$t0)	\$t0	0x 1001 3FF2
0x 0040 0804	addi \$t2, \$t3, 32	\$t1	0x 0001 0C0C
0x 0040 0808	nop	\$t2	0x 0001 8008
0x 0040 080C	sw \$t1, 0x 1A1A(\$t0)	\$t3	0x 0010 80FA
0x 0040 0810	beq \$t0, \$t2, 0x 0080	<b>memoria</b> <b>contenuto iniziale</b>	
0x 0040 0814		0x 1001 4004	
		0x 1001 4008	
		0x 1001 400C	0x 1001 1212 (\$t1 finale)
		0x 1001 5A0C	0x 1001 1A1A

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:

		ciclo di clock										
		1	2	3	4	5	6	7	8	9	10	11
istruzione	1 – lw	IF	ID	EX	MEM	WB						
	2 – addi		IF	ID	EX	MEM	WB					
	3 – nop			IF	ID	EX	MEM	WB				
	4 – sw				IF	ID	EX	MEM	WB			
	5 – beq					IF	ID	EX	MEM	WB		

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

$$0x\ 1001\ 3FF2 + 0x\ 0000\ 001A = 0x\ 1001\ 400C$$

2) **Calcolare** il valore del risultato ( $\$t3 + 32$ ) dell'istruzione *addi* (addizione con immediato):

$$0x\ 0010\ 80FA + 0x\ 0000\ 0020 = 0x\ 0010\ 811A\ (\$t2\ finale)$$

3) **Calcolare** il valore dell'indirizzo di memoria dati nell'istruzione *sw* (store):

$$0x\ 1001\ 3FF2 + 0x\ 0000\ 1A1A = 0x\ 1001\ 5A0C$$

4) **Calcolare** il valore dell'indirizzo di destinazione del salto (si ricorda che l'offset specificato nella *beq* è a parola):

$$0x\ 0040\ 0814 + 0x\ 0000\ 0200 = 0x\ 0040\ 0A14$$

**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 <i>(beq)</i>	ID <i>(sw)</i>	EX <i>(nop)</i>	MEM <i>(addi)</i>
registro IF/ID	registro ID/EX	registro EX/MEM	registro MEM/WB
	.WB.MemtoReg <b>X</b>	.WB.MemtoReg <b>X</b>	.WB.MemtoReg <b>0</b>
	.WB.RegWrite <b>0</b>	.WB.RegWrite <b>0</b>	.WB.RegWrite <b>1</b>
	.M.MemWrite <b>1</b>	.M.MemWrite <b>0</b>	
	.M.MemRead <b>0</b>	.M.MemRead <b>X</b>	
	.M.Branch <b>0</b>	.M.Branch <b>0</b>	
.PC <b>0x 0040 0814</b>	.PC <b>0x 0040 0810</b>	.PC *****	
.istruzione <b>beq</b>	.(Rs) <i>(\$t0)</i> <b>1001 3FF2</b>		
	.(Rt) <i>(\$t1) finale</i> <b>0x 1001 1212</b>	.(Rt) *****	
	.Rt <b>\$t1</b>	.R *****	.R <b>\$t2</b>
	.Rd *****		
	.imm/offset esteso <b>0x 0000 1A1A</b>	.ALU_out *****	.ALU_out <b>0x 0010 811A</b>
	.EX.ALUSrc <b>1</b>	.Zero *****	.DatoLetto *****
	.EX.RegDest <b>X</b>		

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 <b>\$t0 sw</b>	RF.DatoLetto1 <b>0x 1001 3FF2 (\$t0) iniz.</b>	RF.RegScrittura <b>\$t1 lw</b>
RF.RegLettura2 <b>\$t1 sw</b>	RF.DatoLetto2 <b>0x 0001 0C0C (\$t1) iniz.</b>	RF.DatoScritto <b>0x 1001 1212 (\$t1) fin.</b>
segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 6)		
RF.RegLettura1 <b>\$t0 beq</b>	RF.DatoLetto1 <b>0x 1001 3FF2 (\$t0) iniz.</b>	RF.RegScrittura <b>\$t2 addi</b>
RF.RegLettura2 <b>\$t2 beq</b>	RF.DatoLetto2 <b>0x 0001 8008 (\$t2) iniz.</b>	RF.DatoScritto <b>0x 0010 811A (\$t2) fin.</b>

## 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	add \$t1, \$t0, \$t1	IF	ID 0, 1	EX	MEM	WB 1					
2	add \$t2, \$t1, \$t2		IF	ID 1, 2	EX	MEM	WB 2				
3	lw \$t0, 0x 00AA(\$t2)			IF	ID 2	EX	MEM	WB 0			
4	sw \$t0, 0x 00BB(\$t2)				IF	ID 0, 2	EX	MEM	WB		
5	beq \$t0, \$t2, 0x 0089					IF	ID 0, 2	EX	MEM	WB	

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

### punto 1

- 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.
- 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. add	IF	ID 0, 1	EX	M	WB (1)											
2. add		IF	ID stall	ID stall	ID 1, 2	EX	M	WB (2)								
3. lw			IF stall	IF stall	IF	ID stall	ID stall	ID 2	EX	M	WB (0)					
4. sw						IF stall	IF stall	IF	ID stall	ID stall	ID 0, 2	EX	M	WB		
5. beq									IF stall	IF stall	IF	ID 0, 2	EX	M	WB	

## punto 2

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

- Disegnare in **diagramma B** il diagramma temporale della pipeline, indicando i percorsi di propagazione che possono essere attivati per risolvere i conflitti e gli eventuali stalli da inserire affinché la propagazione sia efficace.
- Indicare in **tabella 1** (colonne "punto 2b") i percorsi di propagazione attivati e gli stalli associati, e il ciclo di clock in cui sono attivi i percorsi di propagazione.

**diagramma B**

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

**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
2	1	\$t1	sì	2	2	EX / EX	4
3	2	\$t2	sì	2	2	EX / EX	5
4	2	\$t2	sì	1	assorbito	MEM / EX	6
4	3	\$t0	sì	2	2	MEM / MEM	7
5	2	\$t2	no	0	0	0	--
5	3	\$t0	sì	1	assorbito	1 stallo	7

## esercizio n. 4 – domande su argomenti vari

### memoria cache

Si consideri una gerarchia di memoria composta dalla memoria centrale di **1 Giga byte** indirizzabile a **byte** con **parole da 32 bit**, una memoria **cache istruzioni** a **indirizzamento diretto** e una memoria **cache dati set-associativa** a **4 vie** da **1 Mega byte** ciascuna, con blocchi da **512 byte**. Il bus dati è a **32 bit**. Il tempo di accesso alle cache è pari a **1 ciclo di clock**. Il tempo di accesso alla memoria centrale è pari a **10 cicli di clock**. Rispondere alle domande seguenti:

1. Indicare la struttura degli indirizzi di memoria per le due memorie cache.
2. Calcolare il tempo necessario per caricare un blocco in caso di fallimento (miss).
3. Calcolare il tempo medio di accesso alla memoria di un programma in cui in media il **25 %** delle istruzioni eseguite richiede un accesso in lettura o scrittura a un dato. Il **miss rate** (frequenza di fallimento) della cache **istruzioni** è pari allo **1 %**, mentre per la cache **dati** è pari al **5 %**.

### Soluzione

#### Domanda 1:

numero di bit per indirizzare un byte in memoria centrale: **30**

numero di bit per indirizzare un byte nel blocco: **9**

cache istruzioni:

numero di blocchi in cache:  $2^{20} / 2^9 = 2^{11} = 2048$

struttura indirizzo per cache istruzioni:

numero di bit per indirizzare un byte nel blocco: **9**

numero di bit per indirizzare un blocco in cache: **11**

numero di bit di etichetta:  $30 - 9 - 11 = 10$

cache dati:

numero di insiemi in cache:  $n. \text{ blocchi} / \text{dim. insieme in blocchi} = 2^{11} / 2^2 = 2^9 = 512$

struttura indirizzo per cache dati:

numero di bit per indirizzare un byte nel blocco: **9**

numero di bit per indirizzare un insieme in cache: **9**

numero di bit di etichetta:  $30 - 9 - 9 = 12$

#### Domanda 2:

numero di parole in un blocco:  $512 \text{ byte} / 4 \text{ byte per parola} = 128$

penalità di fallimento (tempo di caricamento di un blocco in caso di miss):  $10 \text{ cicli per parola} \times 128 \text{ parole} = 1280 \text{ cicli}$

#### Domanda 3:

T medio di accesso alla cache istruzioni:  $1 \text{ ciclo} + 0,01 \text{ mr} \times 1280 \text{ pf} = 13,8 \text{ cicli}$

T medio di accesso alla cache dati:  $1 \text{ ciclo} + 0,05 \text{ mr} \times 1280 \text{ pf} = 65 \text{ cicli}$

T medio di accesso alla memoria:  $100/125 \times 13,8 + 25/125 \times 65 = 11,04 + 13 = 24,04 \text{ cicli}$

## logica combinatoria

Si vuole progettare una rete combinatoria con tre ingressi **A**, **B** e **C**, rispettivamente, e un'uscita **U**.

L'uscita **U** vale **1** se i tre ingressi sono identici oppure se l'ingresso **A** vale **0**; altrimenti l'uscita **U** vale **0**. Progettare la rete come **somma di prodotti** (SOP) rispondendo alle seguenti domande.

**Completare la tabella sottostante con i valori dell'uscita U.** Per facilitare lo svolgimento, le colonne ABC che rappresentano gli ingressi della rete sono già state completate.

ABC	U
000	1
001	1
010	1
011	1
100	0
101	0
110	0
111	1

**Scrivere l'espressione della prima forma canonica di U** in funzione di A, B e C, senza ricorrere a semplificazioni o minimizzazioni:

$$U(A, B, C) = \text{_____} \quad !A !B !C + !A !B C + !A B !C + !A B C + A B C$$

**Trasformare l'espressione** in prima forma canonica ricavata al punto precedente – **tramite le proprietà dell'algebra di commutazione** – in modo da ottenere una nuova espressione costituita da soli termini prodotto, con un numero ridotto di termini e con termini prodotto costituiti da al più due variabili, così da diminuire il costo di realizzazione della rete. Si indichino le singole operazioni svolte, e il nome oppure la forma della proprietà utilizzata (numero di righe non significativo).

U (A, B, C)	proprietà
$!A !B !C + !A !B C + !A B !C + !A B C + A B C$	
$!A !B !C + !A !B C + !A B !C + !A B C + A B C + !A B C$	proprietà di idempotenza di <b>!ABC</b>
$!A !B (!C + C) + !A B (!C + C) + (A + !A) B C$	proprietà distributiva AND rispetto a OR
$!A !B 1 + !A B 1 + 1 B C$	proprietà inversa
$!A !B + !A B + B C$	proprietà identità
$!A (!B + B) + B C$	proprietà distributiva AND rispetto a OR
$!A 1 + B C$	proprietà inversa
$!A + B C$	proprietà identità



**spazio libero per continuazione o brutta copia**