



**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**

**PRIMA PARTE – venerdì 15 luglio 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 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 *altsum*, secondo il modello MIPS, e l’allocazione dei parametri e delle variabili locali della funzione *altsum* 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** *altsum* (vedi tab. 4 strutturata).

```
/* costanti e variabili globali */
#define N 10
int value = 0
int NUMBERS [N]
int * ptr

int altsum (int * num, int sign) {          /* funzione altsum */
    int idx, tmp, res
    idx = 0
    while (idx < N) {
        tmp = num [idx]
        if (sign > 0) {
            res = res + tmp
        } else {
            res = res - tmp
        } /* if */
        idx++
        sign = -sign
    } /* while */
    return res
} /* altsum */

/* programma principale */
int main ( ) {
    ptr = &value
    *ptr = altsum (NUMBERS, 1)
} /* 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
PTR	0x 1000 002C	0x 802C	
NUMBERS [N – 1]	0x 1000 0028	0x 8028	
...	...	...	
NUMBERS [0]	0x 1000 0004	0x 8004	
VALUE	0x 1000 0000	0x 8000	indirizzi bassi

<b>punto 1</b> – codice MIPS della sezione dichiarativa globale (numero di righe non significativo)			
	<b>.data</b>	0x 1000 0000	// segmento dati statici standard
	<b>.eqv</b>	N, 10	// costante N = 10
VALUE:	<b>.word</b>	0	// varglob VALUE (inizializzata a 0)
NUMBERS:	<b>.space</b>	40	// varglob NUMBERS (4 * N = 40 byte)
PTR:	<b>.space</b>	4	// varglob PTR (non inizializzata)

punto 2 – area di attivazione della funzione <b>ALTSUM</b>		
contenuto simbolico	spiazz. rispetto a stack pointer	
<i>\$s0 salvato</i>	<i>+8</i>	indirizzi alti
<i>\$s1 salvato</i>	<i>+4</i>	
<i>\$s2 salvato</i>	<i>0</i>	← <i>sp</i> (fine area)
		indirizzi bassi

La funzione **ALTSUM** è foglia, dunque essa non salva in pila il registro *ra*. I registri *\$s0*, *\$s1*, *\$s2* (callee-saved) vanno salvati in pila, dato che vengono usati per allocare, rispettivamente, le variabili locali *idx*, *tmp* e *res*. Complessivamente l'area di attivazione di **ALTSUM** ingombra tre interi (cioè 12 byte).

punto 2 – allocazione dei parametri e delle variabili locali di <b>ALTSUM</b> nei registri	
parametro o variabile locale	registro
<i>num</i>	<i>\$a0</i>
<i>sign</i>	<i>\$a1</i>
<i>idx</i>	<i>\$s0</i>
<i>tmp</i>	<i>\$s1</i>
<i>res</i>	<i>\$s2</i>

<b>punto 3</b> – codice MIPS degli statement riquadrati in <b>MAIN</b> (num. righe non significativo)	
MAIN:	// ptr = &value
<b>la</b>	\$t0, VALUE // carica ind di varglob VALUE
<b>sw</b>	\$t0, PTR // aggiorna varglob PTR
	// *ptr = altsum (NUMBERS, 1)
<b>la</b>	\$a0, NUMBERS // prepara param NUM di funz ALTSUM
<b>li</b>	\$a1, 1 // prepara param SIGN di funz ALTSUM
<b>jal</b>	ALTSUM // chiama funz ALTSUM
<b>lw</b>	\$t0, PTR // carica varglob PTR
<b>sw</b>	\$v0, (\$t0) // aggiorna varglob VALUE

punto 4 – codice MIPS della funzione <b>ALTSUM</b> (numero di righe non significativo)		
ALTSUM:	<b>addiu</b>	\$sp, \$sp, -12 // <b>COMPLETARE</b> - crea area attivazione
		// direttive EQV e salvataggio registri - <b>DA COMPLETARE</b>
	<b>.eqv</b>	\$0, 8 // spi di reg \$s0
	<b>.eqv</b>	\$1, 4 // spi di reg \$s1
	<b>.eqv</b>	\$2, 0 // spi di reg \$s2
	<b>sw</b>	\$s0, 0(\$sp) // salva reg \$s0
	<b>sw</b>	\$s1, 4(\$sp) // salva reg \$s1
	<b>sw</b>	\$s2, 8(\$sp) // salva reg \$s2
		// idx = 0
	<b>mv</b>	\$s0, \$zero // inizializza varloc IDX
WHILE:		// while (idx < N)
	<b>li</b>	\$t0, N // carica cost N
	<b>bge</b>	\$s0, \$t0, ENDWHILE // se IDX >= N vai a ENDWHILE
		// tmp = num [idx]
	<b>sll</b>	\$t0, \$s0, 2 // allinea indice IDX
	<b>addu</b>	\$t0, \$a0, \$t0 // calcola ind di elem NUM[IDX]
	<b>lw</b>	\$s1, (\$t0) // carica elem NUM[IDX] in TMP
IF:		// if (sign > 0)
	<b>ble</b>	\$a1, \$zero, ELSE // se SIGN <= 0 vai a ELSE
THEN:		// res = res + tmp
	<b>add</b>	\$s2, \$s2, \$s1 // calcola RES + TMP e aggiorna RES
	<b>j</b>	ENDIF // vai a ENDIF
ELSE:		// res = res - tmp
	<b>sub</b>	\$s2, \$s2, \$s1 // calcola RES - TMP e aggiorna RES
ENDIF:		// idx++
	<b>addi</b>	\$s0, \$s0, 1 // aggiorna varloc IDX
		// sign = -sign
	<b>neg</b>	\$a1, \$a1 // aggiorna param SIGN
	<b>j</b>	WHILE // torna a WHILE
ENDWHILE:		// return e chiusura funzione - <b>NON RIPORTARE</b>

## seconda parte – istruzione macchina

Si supponga che i due registri temporanei  $t1$  e  $t2$  del processore MIPS contengano **due valori interi in complemento a due**. Si vogliono realizzare le istruzioni MIPS *min* e *max*, così:

*min*  $\$t0, \$t1, \$t2$       // se  $t1 < t2$  allora  $t0 = t1$  altrimenti  $t0 = t2$

*max*  $\$t0, \$t1, \$t2$       // se  $t1 \geq t2$  allora  $t0 = t1$  altrimenti  $t0 = t2$

Si scrivano due programmi assembler che effettuano le operazioni descritte. Il programma **non deve contenere pseudo-istruzioni**, e i registri  $t1$  e  $t2$  **non vanno modificati**. Se occorre, il programma può usare e modificare il registro *at* (assembler temporary).

Si usino le tabelle seguenti (numero di righe non significativo):

### PER MIN:

#	eventuale etichetta	istruzione macchina	commento sintetico
1		<i>add</i> $\$t0, \$t1, \$zero$	<i>copia t1 in t0</i>
2		<i>slt</i> $\$at, \$t1, \$t2$	<i>se t1 &lt; t2 at = 1</i> <i>se t1 ≥ t2 at = 0</i>
3		<i>bne</i> $\$at, \$zero, END$	<i>se at != 0 vai a END</i>
4		<i>add</i> $\$t0, \$t2, \$zero$	<i>copia t2 in t0</i>
5	<i>END</i>		
6			

### PER MAX:

#	eventuale etichetta	istruzione macchina	commento sintetico
1		<i>add</i> $\$t0, \$t1, \$zero$	<i>copia t1 in t0</i>
2		<i>slt</i> $\$at, \$t1, \$t2$	<i>se t1 &lt; t2 at = 1</i> <i>se t1 ≥ t2 at = 0</i>
3		<i>beq</i> $\$at, \$zero, END$	<i>se at == 0 vai a END</i>
4		<i>add</i> $\$t0, \$t2, \$zero$	<i>copia t2 in t0</i>
5	<i>END</i>		
6			

*Nota: vengono usate solo istruzioni native, non pseudo-istruzioni; sono possibili altre soluzioni.*

## 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.

$$D1 = I \text{ or } (!Q1 \text{ or } Q2)$$

$$D2 = !I \text{ and } (!Q1 \text{ or } !Q2)$$

$$U = Q1 \text{ xor } Q2$$

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 evidenziare la corrispondenza tra questa e i valori 0 e 1 presenti nella tabella dei segnali complessiva
- notare che nel primo intervallo i segnali Q1 e Q2 sono già dati (rappresentano lo stato iniziale)

<b>I</b>	0	0	0	0	1	1	1	1	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0
<b>D1</b>	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
<b>Q1</b>	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
<b>D2</b>	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1
<b>Q2</b>	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1
<b>U</b>	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1
<b>CLK</b>	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 0BBB (\$t4)	\$t0	0x 0110 A010
0x 0040 0804	sw \$t3, 0x 0AA7 (\$t2)	\$t1	0x 0000 1111
0x 0040 0808	nop	\$t2	0x 1060 3455
0x 0040 080C	add \$t5, \$t1, \$t2	\$t3	0x 0050 0000
0x 0040 0810	addi \$t6, \$t1, 0x A001	\$t4	0x 1060 0055
0x 0040 0814		<b>memoria</b>	<b>contenuto iniziale</b>
		0x 1060 0C10	0x 0044 0FFF (\$t1 finale)
		0x 1060 0C14	0x 11FF 0040
		0x 1060 3EFC	0x 48F0 6610

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 – sw		IF	ID	EX	MEM	WB					
	3 – nop			IF	ID	EX	MEM	WB				
	4 - add				IF	ID	EX	MEM	WB			
	5 - addi					IF	ID	EX	MEM	WB		

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

$$0x\ 1060\ 0055 + 0x\ 0000\ 0BBB = 0x\ 1060\ 0C10$$

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

$$0x\ 1060\ 3455 + 0x\ 0000\ 0AA7 = 0x\ 1060\ 3EFC$$

3) **Calcolare** il valore del risultato ( $t1 + t2$ ) dell'istruzione *add* (addizione):

$$0x\ 0044\ 0FFF + 0x\ 1060\ 3455 = 0x\ 10A4\ 4454\ (\$t5\ finale)$$

4) **Calcolare** il valore del risultato ( $t1 + 0xA001$ ) dell'istruzione *addi* (addizione con immediato):

$$0x\ 0044\ 0FFF + 0x\ FFFF\ A001 = 0x\ 0043\ B000\ (\$t6\ finale)$$

**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)	(add)	(nop)	(sw)
registro IF/ID	registro ID/EX	registro EX/MEM	registro MEM/WB
	.WB.MemtoReg <b>0</b>	.WB.MemtoReg <b>X</b>	.WB.MemtoReg <b>X</b>
	.WB.RegWrite <b>1</b>	.WB.RegWrite <b>0</b>	.WB.RegWrite <b>0</b>
	.M.MemWrite <b>0</b>	.M.MemWrite <b>0</b>	
	.M.MemRead <b>0</b>	.M.MemRead <b>0</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>addi</b>	.(Rs) <b>(\$t1) finale</b> <b>0044 0FFF</b>		
	.(Rt) <b>(\$t2)</b> <b>0x 1060 3455</b>	.(Rt) *****	
	.Rt <b>\$t2 0A</b>	.R *****	.R *****
	.Rd <b>\$t1 09</b>		
	.imm/offset esteso *****	.ALU_out *****	.ALU_out <i>ind mem sw</i> <b>0x 1060 3EFC</b>
	.EX.ALUSrc <b>0</b>	.Zero *****	.DatoLetto *****
	.EX.RegDest <b>1</b>		

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 <b>\$t1 09 add</b>	RF.DatoLetto1 <b>0x 0000 1111 (\$t1) iniz.</b>	RF.RegScrittura <b>\$t1 lw</b>
RF.RegLettura2 <b>\$t2 0A add</b>	RF.DatoLetto2 <b>0x 1060 3455 (\$t2) iniz.</b>	RF.DatoScritto <b>0x 0044 0FFF (\$t1) 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, \$t3, \$t0	IF	ID 0, 3	EX	MEM	WB 1					
2	add \$t0, \$t3, \$t1		IF	ID 1, 3	EX	MEM	WB 0				
3	lw \$t3, 0x 00AA(\$t0)			IF	ID 0	EX	MEM	WB 3			
4	sw \$t3, 0x 00CC(\$t1)				IF	ID 1, 3	EX	MEM	WB		

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

### punto 2

Si faccia l'ipotesi che la pipeline sia **ottimizzata** e 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 devono 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 \$t1	IF	ID 0, 3	EX (1)	M	WB (1)										
2.add \$t0		IF	ID 1, 3	EX (0)	M	WB (0)									
3.lw \$t3			IF	ID 0, 3	EX	M (3)	WB (3)								
4.sw \$t3				IF	ID 1, 3	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	\$t0	sì	2	2	EX / EX	5
4	3	\$t3	sì	2	2	MEM / MEM	7

## esercizio n. 4 – domande su argomenti vari

### logica combinatoria

Si consideri la funzione booleana di quattro variabili  $G(a, b, c, d)$  rappresentata dall'espressione seguente:

$$G(a, b, c, d) = \neg a \neg(b + c) + c \neg(a + b) + \neg a \neg c \neg(b + d) + c \neg b \neg(a + d)$$

Si trasformi – tramite le proprietà dell'algebra di commutazione – l'espressione di  $G$  in modo da ridurre il costo della sua realizzazione, indicando per nome la singola trasformazione svolta oppure la forma della proprietà utilizzata. Allo scopo si usi la tabella seguente (il numero di righe non è significativo).

### soluzione

espressione trasformata	proprietà
$\neg a \neg(b + c) + c \neg(a + b) + \neg a \neg c \neg(b + d) + c \neg b \neg(a + d)$	<i>De Morgan</i>
$\neg a \neg b \neg c + c \neg a \neg b + \neg a \neg c \neg b \neg d + c \neg b \neg a \neg d$	<i>commutativa</i>
$\neg a \neg b \neg c + \neg a \neg b c + \neg a \neg b \neg c \neg d + \neg a \neg b c \neg d$	<i>commutativa</i>
$\neg a \neg b \neg c + \neg a \neg b \neg c \neg d + \neg a \neg b c + \neg a \neg b c \neg d$	<i>distributiva</i>
$\neg a \neg b \neg c (1 + \neg d) + \neg a \neg b c (1 + \neg d)$	<i>elemento nullo</i>
$\neg a \neg b \neg c 1 + \neg a \neg b c 1$	<i>elemento unitario</i>
$\neg a \neg b \neg c + \neg a \neg b c$	<i>distributiva</i>
$\neg a \neg b (\neg c + c)$	<i>elemento inverso</i>
$\neg a \neg b 1$	<i>elemento unitario</i>
$\neg a \neg b$	<i>forma minima</i>

**spazio libero per continuazione o brutta copia**

**spazio libero per continuazione o brutta copia**