



**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 – giovedì 23 giugno 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 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** *verify* (vedi tab. 4 strutturata).

```
/* costanti e variabili globali */
#define N 28
char WORD [N]
int maius

/* testata funzione ausiliaria - è una funzione foglia */
/* se c è carattere maiuscolo restituisce 1, altrimenti 0 */
int ucase (char c)

/* funz. verify - verifica se la stringa è maiuscola */
int verify (char * STR, int dim) {
    int * ptr
    int yes
    int cnt
    ptr = STR
    yes = 1
    for (cnt = dim - 1, cnt >= 0, cnt--) {
        yes = ucase (*ptr) && yes
        ptr++
    } /* for */
    return yes
} /* verify */

/* programma principale */
int main ( ) {
    maius = verify (WORD, N)
} /* main */
```

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

contenuto simbolico	indirizzo assoluto iniziale (in hex)	spiazzamento rispetto a <i>gp</i> = <b>0x 1000 8000</b>	
			indirizzi alti
...	...	...	
MAIUS	<i>0x 1000 001C</i>	<i>0x 801C</i>	
WORD [N – 1]	<i>0x 1000 001B</i>	<i>0x 801B</i>	
...	...	...	
WORD [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, 28</i>	// costante <i>N</i> = 28
<i>WORD:</i>	<b>.space</b>	<i>N</i>	// varglob <i>WORD</i> ( <i>N</i> = 28 byte)
<i>MAIUS:</i>	<b>.space</b>	4	// varglob <i>MAIUS</i> (un intero non iniz)

punto 2 – area di attivazione della funzione <b>VERIFY</b>		
contenuto simbolico	spiazz. rispetto a stack pointer	
<i>\$ra salvato</i>	<i>+12</i>	indirizzi alti
<i>\$s0 salvato</i>	<i>+8</i>	
<i>\$s1 salvato</i>	<i>+4</i>	
<i>\$s2 salvato</i>	<i>0</i>	← <i>sp</i> (fine area)
		indirizzi bassi

La funzione **VERIFY** non è foglia, dunque essa 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 *ptr*, *yes* e *cnt*. Complessivamente l'area di attivazione di **VERIFY** ingombra quattro interi (cioè 16 byte).

punto 2 – allocazione dei parametri e delle variabili locali di <b>VERIFY</b> nei registri	
parametro o variabile locale	registro
<i>STR</i>	<i>\$a0</i>
<i>dim</i>	<i>\$a1</i>
<i>ptr</i>	<i>\$s0</i>
<i>yes</i>	<i>\$s1</i>
<i>cnt</i>	<i>\$s2</i>

punto 3 – codice MIPS dello statement riquadrato in <b>MAIN</b> (num. righe non significativo)			
// maius = verify (WORD, N)			
MAIN:	<b>la</b>	<i>\$a0, WORD</i>	// prepara param <i>STR</i> di funz <b>VERIFY</b>
	<b>li</b>	<i>\$a1, N</i>	// prepara param <i>DIM</i> di funz <b>VERIFY</b>
	<b>jal</b>	<i>VERIFY</i>	// chiama funz <b>VERIFY</b>
	<b>sw</b>	<i>\$v0, MAIUS</i>	// aggiorna varglob <b>MAIUS</b>

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

```
VERIFY:  addiu  $sp, $sp, -16      // COMPLETARE - crea area attivazione
// direttive EQU e salvataggio registri - NON VANNO RIPORTATI
// ptr = STR
move    $s0, $a0                // inizializza varloc PTR

// yes = 1
li      $s1, 1                  // inizializza varloc YES

// for (cnt = dim - 1, cnt >= 0, cnt--)
move    $t0, $a1                // inizializza varloc CNT (nota)
subi    $s2, $t0, 1             // inizializza varloc CNT

FOR:     blt    $s2, $zero, ENDFOR // se CNT < 0 vai a ENDFOR

// yes = ucase (*ptr) && yes
lb      $a0, ($s0)              // prepara param *PTR
jal     UCASE                   // chiama funz UCASE
and     $s1, $v0, $s1           // calcola espr ... && ...

// ptr++
addi    $s0, $s0, 1             // aggiorna varloc PTR

// cnt--
subi    $s2, $s2, 1             // aggiorna varloc CNT
j       FOR                     // vai a FOR

ENDFOR:  // ripristino registri - NON VANNO RIPORTATI
// restituisci valore, elimina area e rientra
move    $v0, $s1                // prepara valusc
addiu   $sp, $sp, +16           // elimina area attivazione
jr      $ra                     // rientra a chiamante

nota: coppia di istruzioni ottimizzabile come subi $s2, $a1, 1
```

## seconda parte – assemblaggio e collegamento

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

1. i due moduli oggetto MAIN e SECONDARY
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 SECONDARY		
	<b>.data</b>			<b>.data</b>	
BLOCK:	<b>.space</b>	30	SUB:	<b>.word</b>	0
	<b>.text</b>			<b>.text</b>	
	<b>.globl</b>	MAIN		<b>.globl</b>	SEC
MAIN:	<b>move</b>	\$a0, \$zero	SEC:	<b>bne</b>	\$a0, \$t0, FUN
	<b>la</b>	\$t0, BLOCK		<b>move</b>	\$a0, \$zero
FUN:	<b>jal</b>	SEC	LOOP:	<b>addi</b>	\$a0, \$a0, 1
	<b>bne</b>	\$v0, \$zero, OVER		<b>sw</b>	\$a0, BLOCK
	<b>move</b>	\$t0, \$v0		<b>beq</b>	\$t0, \$a0, LOOP
OVER:	<b>addi</b>	\$t0, \$t0, 1		<b>jr</b>	\$ra
	<b>sw</b>	\$t0, SUB			
	<b>j</b>	FUN			

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 SECONDARY		
dimensione testo: 24 hex (36 dec)			dimensione testo: 18 hex (24 dec)		
dimensione dati: 1E hex (30 dec)			dimensione dati: 04 hex (4 dec)		
testo			testo		
indirizzo di parola	istruzione (COMPLETARE)		indirizzo di parola	istruzione (COMPLETARE)	
0	move	\$a0, \$zero	0	bne	\$a0, \$t0, 0000
4	lui	\$t0, 0000	4	move	\$a0, \$zero
8	ori	\$t0, \$t0, 0000	8	addi	\$a0, \$a0, 1
C	jal	000 0000	C	sw	\$a0, 0000 (\$gp)
10	bne	\$v0, \$zero, 0001 = +1	10	beq	\$t0, \$a0, FFFD = -3
14	move	\$t0, \$v0	14	jr	\$ra
18	addi	\$t0, \$t0, 1	18		
1C	sw	\$t0, 0000 (\$gp)	1C		
20	j	000 0000	20		
24			24		
28			28		
2C			2C		
dati			dati		
indirizzo di parola	contenuto		indirizzo di parola	contenuto	
0	non specificato		0	0x 0000 0000	
1E			4		
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
BLOCK	D	0x 0000 0000	SUB	D	0x 0000 0000
MAIN	T	0x 0000 0000	SEC	T	0x 0000 0000
FUN	T	0x 0000 000C	LOOP	T	0x 0000 0008
OVER	T	0x 0000 0018			
tabella di rilocalizzazione			tabella di rilocalizzazione		
indirizzo di parola	cod. operativo	simbolo	indirizzo di parola	cod. operativo	simbolo
4	lui	BLOCK	0	bne	FUN
8	ori	BLOCK	C	sw	BLOCK
C	jal	SEC			
1C	sw	SUB			
20	j	FUN			

(2) – posizione in memoria dei moduli			
modulo MAIN		modulo SECONDARY	
base del testo:	0x 0040 0000	base del testo:	0x 0040 0024
base dei dati:	0x 1000 0000	base dei dati:	0x 1000 001E

(3) – tabella globale dei simboli				
simbolo	valore finale		simbolo	valore finale
BLOCK	0x 1000 0000		SUB	0x 1000 001E
MAIN	0x 0040 0000		SEC	0x 0040 0024
FUN	0x 0040 000C		LOOP	0x 0040 002C
OVER	0x 0040 00018			

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

(3) – codice eseguibile	
testo	
indirizzo	codice (con codici operativi e registri in forma simbolica)
...	...
4	<i>lui \$t0, 0x 1000</i> // MAIN: <i>lui \$t0, BLOCK %hi</i>
8	<i>ori \$t0, 0x 0000</i> // MAIN: <i>ori \$t0, BLOCK %lo</i>
C	<i>jal 0x 010 0009</i> // MAIN: <i>jal SEC</i>
...	...
1C	<i>sw \$t0, 0x 801E(\$gp)</i> // MAIN: <i>sw \$t0, SUB</i>
20	<i>j 0x 010 0003</i> // MAIN: <i>j FUN</i>
...	...
24	<i>bne \$a0, \$t0, 0x FFF9 = -7</i> // SECONDARY: <i>bne \$a0, \$t0, FUN</i>
30	<i>sw \$a0, 0x 8000(\$gp)</i> // SECONDARY: <i>sw \$a0, BLOCK</i>
...	...



## 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.

$$D1 = !I \text{ xor } Q2$$

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

$$U = I \text{ or } Q1$$

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>	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1
<b>D1</b>	0	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1
<b>Q1</b>	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
<b>D2</b>	1	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	0	0
<b>Q2</b>	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
<b>U</b>	1	0	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	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
0x 0040 0800	lw \$t1, 0x008B(\$t0)
0x 0040 0804	addi \$t2, \$t3, 32
0x 0040 0808	sw \$t3, 0x00AB(\$t0)
0x 0040 080C	add \$t4, \$t1, \$t3
0x 0040 0810	beq \$t0, \$t2, 0x0080
0x 0040 0814	

registro	contenuto iniziale
\$t0	0x 1001 4021
\$t1	0x 0001 CCCC
\$t2	0x 0001 80AA
\$t3	0x 0010 800A
memoria	contenuto iniziale
0x 1001 4004	
0x 1001 4008	
0x 1001 40AC	0x 1001 1A1A (\$t1 finale)
0x 1001 40CC	0x 1001 FFCC

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 – sw			IF	ID	EX	MEM	WB				
	4 – add				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):

$$0x\ 1001\ 4021 + 0x\ 0000\ 008B = 0x\ 1001\ 40AC \underline{\hspace{2cm}}$$

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

$$0x\ 0010\ 800A + 0x\ 0000\ 0020 = 0x\ 0010\ 802A\ (\$t2\ finale) \underline{\hspace{2cm}}$$

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

$$0x\ 1001\ 4021 + 0x\ 0000\ 00AB = 0x\ 1001\ 40CC \underline{\hspace{2cm}}$$

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

$$0x\ 0040\ 0814 + 0x\ 0000\ 0200 = 0x\ 0040\ 0A14 \underline{\hspace{2cm}}$$

**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
<i>(beq)</i>	<i>(add)</i>	<i>(sw)</i>	<i>(addi)</i>
registro IF/ID	registro ID/EX	registro EX/MEM	registro MEM/WB
	.WB.MemtoReg <i>0</i>	.WB.MemtoReg <i>X</i>	.WB.MemtoReg <i>0</i>
	.WB.RegWrite <i>1</i>	.WB.RegWrite <i>0</i>	.WB.RegWrite <i>1</i>
	.M.MemWrite <i>0</i>	.M.MemWrite <i>1</i>	
	.M.MemRead <i>0</i>	.M.MemRead <i>0</i>	
	.M.Branch <i>0</i>	.M.Branch <i>0</i>	
.PC <i>0x 0040 0814</i>	.PC <i>0x 0040 0810</i>	.PC *****	
.istruzione <i>beq</i>	.(Rs) <i>(\$t1) finale</i> <i>1001 1A1A</i>		
	.(Rt) <i>(\$t3)</i> <i>0x 0010 800A</i>	.(Rt) <i>(\$t3)</i> <i>0x 0010 800A</i>	
	.Rt <i>\$t3 0B</i>	.R *****	.R <i>\$t2 0A</i>
	.Rd <i>\$t4 0C</i>		
	.imm/offset esteso *****	.ALU_out <i>ind mem sw</i> <i>0x 1001 40CC</i>	.ALU_out <i>\$t2 finale</i> <i>0x 0010 802A</i>
	.EX.ALUSrc <i>0</i>	.Zero *****	.DatoLetto *****
	.EX.RegDest <i>1</i>		

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 <i>\$t1 09 add</i>	RF.DatoLetto1 <i>0x 0001 CCCC (\$t1) iniz.</i>	RF.RegScrittura <i>\$t1 lw</i>
RF.RegLettura2 <i>\$t3 0B add</i>	RF.DatoLetto2 <i>0x 0010 800A (\$t3) iniz.</i>	RF.DatoScritto <i>0x 1001 1A1A (\$t1) fin.</i>
segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 6)		
RF.RegLettura1 <i>\$t0 08 beq</i>	RF.DatoLetto1 <i>0x 1001 4021 (\$t0) iniz.</i>	RF.RegScrittura <i>\$t2 addi</i>
RF.RegLettura2 <i>\$t2 0A beq</i>	RF.DatoLetto2 <i>0x 0001 80AA (\$t2) iniz.</i>	RF.DatoScritto <i>0x 0010 802A (\$t2) fin.</i>

## 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	sw	\$t1, 0x 00AA (\$t0)	IF	ID 0,1	EX	MEM	WB					
2	lw	\$t2, 0x 00BB (\$t0)		IF	ID 0	EX	MEM	WB 2				
3	add	\$t3, \$t1, \$t2			IF	ID 1, 2	EX	MEM	WB 3			
4	add	\$t4, \$t3, \$t3				IF	ID 3	EX	MEM	WB 4		
5	sw	\$t4, 0x 00CC (\$t0)					IF	ID 0,4	EX	MEM	WB	

### punto 1

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 A** 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** le dipendenze, i percorsi di propagazione attivati con gli stalli associati, e il ciclo di clock nel quale sono attivi i percorsi di propagazione.

### diagramma A

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1. sw \$t1	IF	ID 0,1	EX	M	WB										
2. lw \$t2		IF	ID 0	EX	M (2)	WB (2)									
3. add \$t3			IF	ID stall	ID 1,2	EX (3)	M (3)	WB (3)							
4. add \$t4				IF stall	IF	ID 3	EX (4)	M (4)	WB (4)						
5. sw \$t4						IF	ID 0,4	EX	M	WB					

**tabella 1**

N° istruzione	N° istruzione da cui dipende	registro coinvolto	stalli + percorso di propagazione	ciclo di clock in cui è attivo il percorso
<i>3</i>	<i>2</i>	<i>\$t2</i>	<i>1 stallo + MEM / EX</i>	<i>6</i>
<i>4</i>	<i>3</i>	<i>\$t3</i>	<i>EX / EX</i>	<i>7</i>
<i>5</i>	<i>4</i>	<i>\$t4</i>	<i>EX / EX</i>	<i>8</i>

## esercizio n. 4 – domande su argomenti vari

### memoria cache

Si consideri una gerarchia di memoria composta dalla memoria centrale da **1 Giga byte**, indirizzabile a byte con parole da **32 bit**, una memoria cache istruzioni da **1 Mega byte** e una memoria cache dati da **512 K byte**, entrambe a indirizzamento diretto con blocchi da **512 byte**.

Il tempo di accesso alle cache è pari a **1 ciclo di clock**. Il tempo di accesso alla memoria centrale è pari a **20 cicli di clock** per la prima parola del blocco e pari a **5 cicli di clock** per le parole a indirizzi successivi (memoria interallacciata). Il bus dati è da **32 bit**.

Rispondere alla quattro domande seguenti:

1. Indicare la **struttura degli indirizzi** di memoria per la cache **istruzioni** e la cache **dati**:

cache istruzioni		
etichetta	indice	spiazzamento
10 bit	11 bit	9 bit

cache dati		
etichetta	indice	spiazzamento
11 bit	10 bit	9 bit

2. Calcolare il **tempo medio** necessario per caricare in cache un blocco in caso di fallimento (miss).

$$N. \text{ parole per blocco} = 512 / 4 = 128 \text{ parole}$$

$$\text{Tempo medio per caricare un blocco} = 20 + 5 \times 127 = 655 \text{ cicli di clock}$$

3. Viene mandato in esecuzione un nuovo programma che:
  - a. accede sequenzialmente a un array di **1026 blocchi** e poi
  - b. esegue per **10 volte** un ciclo accedendo sequenzialmente ai blocchi **0, 1, 1024, 1025**

Calcolare:

- numero di **miss totali** alla cache **dati** =  $1026 + 4 \times 10 = 1066 \text{ miss}$ ; ci sono infatti 1026 miss per l'accesso ai 1026 blocchi del file, dunque per il ciclo del punto b) ci sono 4 miss per ogni iterazione
- numero di **accessi totali** alla cache **dati** = ogni blocco è di 128 parole, dunque si hanno  $1026 \times 128$  accessi per il punto a) e poi  $4 \times 10 \times 128$  per il punto b) =  $128 \times (1026 + 4 \times 10)$
- **miss rate** della cache **dati** =  $n^{\circ} \text{ miss totali} / n^{\circ} \text{ accessi totali} = (1026 + 4 \times 10) / 128 \times (1026 + 4 \times 10) = 1 / 128 \approx 0,78 \%$

4. Calcolare il **tempo medio** di accesso alla memoria di questo programma considerando che il **miss rate** della cache **istruzioni** è pari a **1%** e che la **percentuale di accessi ai dati** è del **25%**.

$$AMAT = 100 / 125 \times T \text{ medio istruzioni} + 25 / 125 \times T \text{ medio dati}$$

$$T \text{ medio istruzioni} = 0,99 \times 1 + 0,01 \times (655 + 1) = 7,55 \text{ cicli di clock}$$

$$T \text{ medio accesso ai dati} = (1 - 1 / 128) \times 1 + 1 / 128 \times (655 + 1) = 0,9922 \times 1 + 0,0078 \times 656 = 6,11 \text{ cicli di clock}$$

$$T \text{ medio accesso alla memoria} = 100 / 125 \times 7,55 + 25 / 125 \times 6,11 = 7,262 \text{ cicli di clock}$$

**spazio libero per continuazione o brutta copia**