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 – 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) _____

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 `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 ***gp*** = **0x 1000 8000**	
			indirizzi alti
...			
MAIUS			
WORD [N – 1]			
...			
WORD [0]			indirizzi bassi

punto 1 – codice MIPS della sezione dichiarativa globale (numero di righe non significativo)
.data 0x 1000 0000 // segmento dati statici standard

punto 2 – area di attivazione della funzione **VERIFY**	
contenuto simbolico	**spiazz. rispetto a stack pointer**

indirizzi alti

indirizzi bassi

punto 2 – allocazione dei parametri e delle variabili locali di **VERIFY** nei registri	
parametro o variabile locale	**registro**

punto 3 – codice MIPS dello statement riquadrato in **MAIN** (num. righe non significativo)
`//  maius = verify (WORD, N)`
`MAIN:`

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

```
VERIFY:   addiu  $sp, $sp,      // COMPLETARE - crea area attivazione
          // direttive EQU e salvataggio registri - NON VANNO RIPORTATI
          // ptr = STR
          // yes = 1
          // for (cnt = dim - 1, cnt >= 0, cnt--)
FOR:
          // yes = ucase (*ptr) && yes
          // ptr++
          // cnt--
ENDFOR:   // ripristino registri - NON VANNO RIPORTATI
          // restituisci valore, elimina area e rientra
```

seconda parte – assemblaggio e collegamento

Dati i due moduli assemblatore 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

```
            .data
BLOCK:      .space    30
            .text
            .globl    MAIN
MAIN:       move   $a0, $zero
            la     $t0, BLOCK
FUN:        jal    SEC
            bne    $v0, $zero, OVER
            move   $t0, $v0
OVER:       addi   $t0, $t0, 1
            sw     $t0, SUB
            j      FUN
```

modulo SECONDARY

```
            .data
SUB:        .word     0
            .text
            .globl    SEC
SEC:        bne    $a0, $t0, FUN
            move   $a0, $zero
LOOP:       addi   $a0, $a0, 1
            sw     $a0, BLOCK
            beq    $t0, $a0, LOOP
            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:

dimensione dati:

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**move**
4	**lui**
8	**ori**
C	**jal**
10	**bne**
14	**move**
18	**addi**
1C	**sw**
20	**j**
24	
28	
2C	

dati

indirizzo di parola	contenuto

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

simbolo	tipo	valore
BLOCK		
MAIN		
FUN		
OVER		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

modulo `SECONDARY`

dimensione testo:

dimensione dati:

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**bne**
4	**move**
8	**addi**
C	**sw**
10	**beq**
14	**jr**
18	
1C	
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto

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

simbolo	tipo	valore
SUB		
SEC		
LOOP		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

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

(3) – tabella globale dei simboli				
simbolo	**valore finale**		**simbolo**	**valore finale**
`BLOCK`			`SUB`	
`MAIN`			`SEC`	
`FUN`			`LOOP`	
`OVER`				

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	
8	
C	
. . .	
1C	
20	
. . .	
24	
30	
. . .	

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 Q2

D2 = (I and !Q2) or (Q1 and !Q2)

U  = I or Q1
```

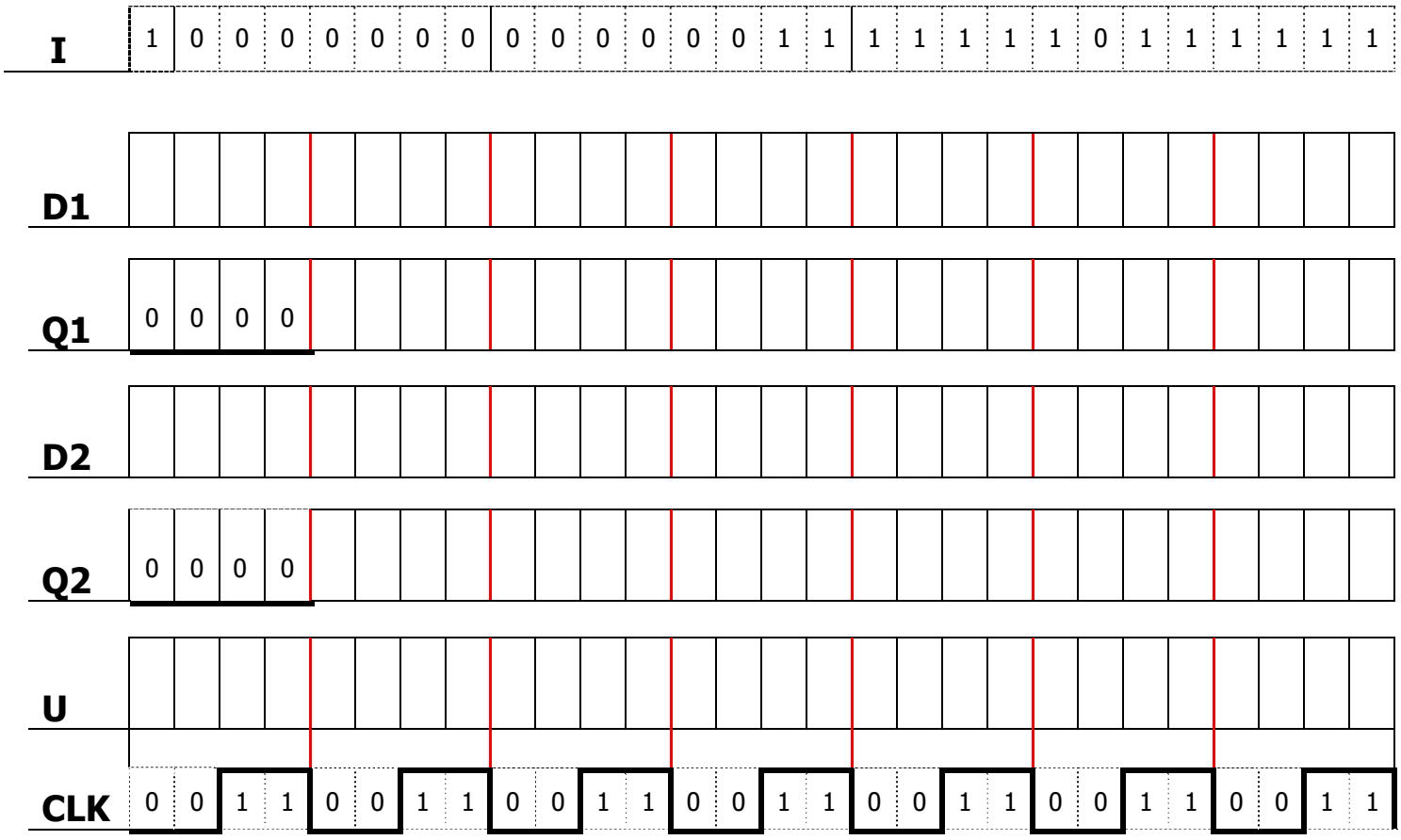
N.B. operatore XOR, uscita a 1 se e solo se solo uno degli ingressi è a 1

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)



esercizio n. 3 – microarchitettura del processore pipeline

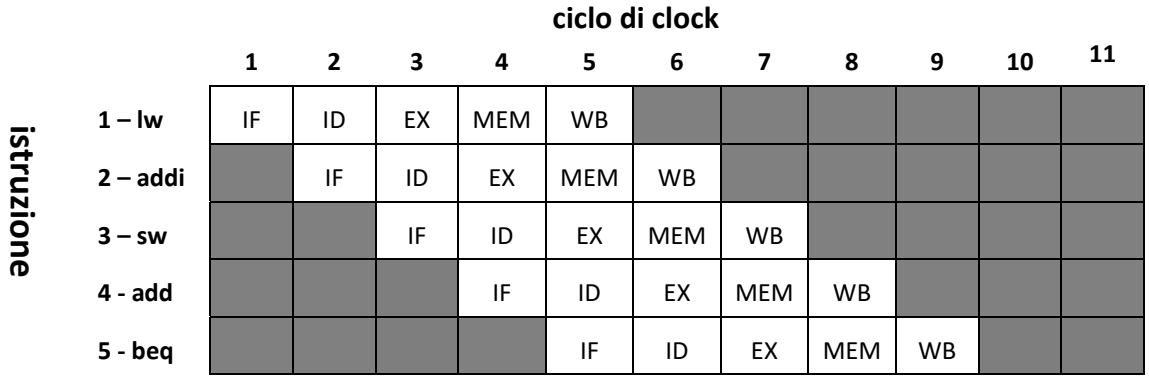
prima parte – pipeline e segnali di controllo

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)
	addi $t2, $t3, 32
	sw $t3, 0x00AB($t0)
	add $t4, $t1, $t3
	beq $t0, $t2, 0x0080

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
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:



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

_____

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

_____

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

_____

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

_____

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**
registro IF/ID	***registro ID/EX***	***registro EX/MEM***	***registro MEM/WB***
	.WB.MemtoReg	.WB.MemtoReg	.WB.MemtoReg
	.WB.RegWrite	.WB.RegWrite	.WB.RegWrite
	.M.MemWrite	.M.MemWrite	
	.M.MemRead	.M.MemRead	
	.M.Branch	.M.Branch	
.PC	.PC	.PC **********	
.istruzione	.(Rs)		
	.(Rt)	.(Rt)	
	.Rt	**********	.R
	.Rd		
	.imm/offset esteso **********	.ALU_out	.ALU_out
	.EX.ALUSrc	.Zero **********	.DatoLetto **********
	.EX.RegDest		

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1	RF.DatoLetto1	RF.RegScrittura
RF.RegLettura2	RF.DatoLetto2	RF.DatoScritto
segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 6)		
RF.RegLettura1	RF.DatoLetto1	RF.RegScrittura
RF.RegLettura2	RF.DatoLetto2	RF.DatoScritto

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	EX	MEM	WB					
2	`lw $t2, 0x 00BB($t0)`		IF	ID	EX	MEM	WB				
3	`add $t3, $t1, $t2`			IF	ID	EX	MEM	WB			
4	`add $t4, $t3, $t3`				IF	ID	EX	MEM	WB		
5	`sw $t4, 0x 00CC($t0)`					IF	ID	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**:

a. 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.

b. 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`															
`2.lw $t2`															
`3.add $t3`															
`4.add $t4`															
`5.sw $t4`															

tabella 1

N° istruzione	N° istruzione da cui dipende	registro coinvolto	stalli + percorso di propagazione	ciclo di clock in cui è attivo il percorso

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 acceso 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

cache dati		
etichetta	indice	spiazzamento

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

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** =
   - numero di **accessi totali** alla cache **dati** =
   - **miss rate** della cache **dati** =

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 =

   T medio istruzioni =

   T medio accesso ai dati =

   T medio accesso alla memoria =

spazio libero per continuazione o brutta copia