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) _____

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]			
…			
LETTERS [1]			
LETTERS [0]			
CODE			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 **`CHKSUM`**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
		indirizzi alti
		indirizzi bassi

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

punto 3 – codice MIPS degli statement riquadrati in **`MAIN`** (num. righe non significativo)
`MAIN:  //  code = chksum (LETTERS, code)`

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

```
CHKSUM:   addiu  $sp, $sp,      // COMPLETARE - crea area attivazione
          // direttive EQV e salvataggio registri – DA COMPLETARE
          .eqv
          .eqv
          .eqv




          // ptr = &partial




          // *ptr = 0


          // for (idx = N - 1; idx >= 0; idx--)


FOR:

          // *ptr = *ptr + byte [idx] - weight




          // weight++


          // idx--


ENDFOR:

          // return *ptr


          // 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:

dimensione dati:

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** $a0, $s0, 0
4	**lui** $a1,
8	**addi** $a1, $a1,
C	**jal**
10	**beq** $v0, $zero,
14	**sw** $v0,
18	**sw** $v0,
1C	**j**
20	
24	
28	
2C	

dati

indirizzo di parola	contenuto

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

simbolo	tipo	valore (hex)
WEIGHT		
REF		
MAIN		
FUNCT		
NEXT		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

modulo `TASK`

dimensione testo:

dimensione dati:

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**bne** $a0, $zero,
4	**andi** $a0, $a1, 0x 1234
8	**sw** $a0,
C	**beq** $s0, $a0,
10	**jr** $ra
14	
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 (hex)
RESERVED		
LOCAL		
TASK		
AFTER		

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo

(2) – posizione in memoria dei moduli	
modulo `MAIN`	**modulo** `TASK`
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 (hex)**		**simbolo**	**valore finale (hex)**
`WEIGHT`			`RESERVED`	
`REF`			`LOCAL`	
`MAIN`			`TASK`	
`FUNCT`			`AFTER`	
`NEXT`				

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	
. . .	
14	
18	
1C	
. . .	
28	
2C	
. . .	

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)

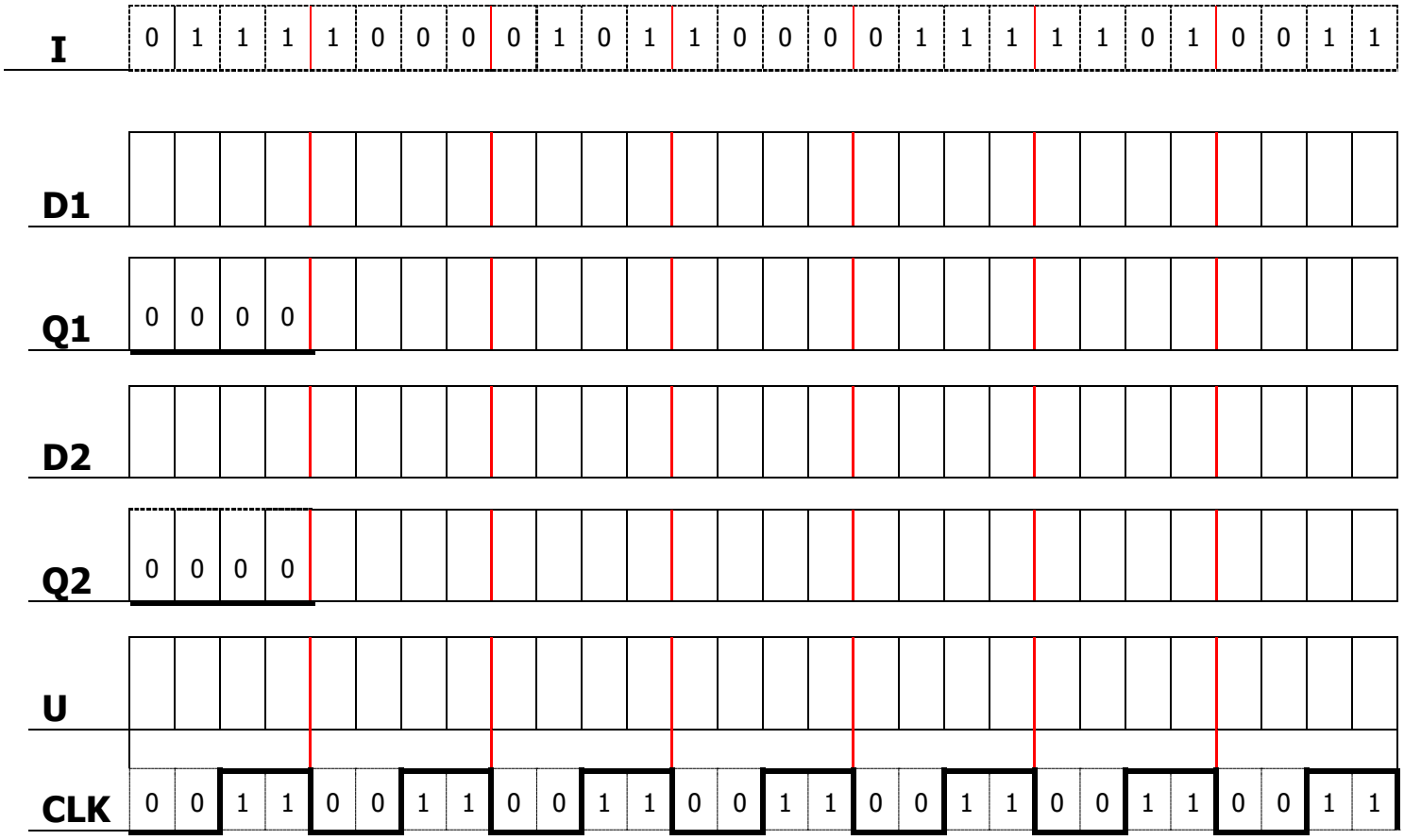
Si chiede di completare il diagramma temporale riportato qui sotto. Si noti che:

- 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)



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)`
	`lw $t2, 0x 193A($t3)`
	`add $t3, $t3, $t3`
	`subi $t4, $t1, 32`
	`addi $t5, $t2, 4`

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`
`0x 1001 49C4`	`0x 1001 A0A0`

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:

istruzione \ ciclo di clock	1	2	3	4	5	6	7	8	9	10	11
1 – lw $t1	IF	ID	EX	MEM	WB						
2 – lw $t2		IF	ID	EX	MEM	WB					
3 – add $t3			IF	ID	EX	MEM	WB				
4 – subi $t4				IF	ID	EX	MEM	WB			
5 – addi $t5					IF	ID	EX	MEM	WB		

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

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

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

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

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

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	.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	`lw $s1, 0x AA($s0)`	IF	ID	EX	MEM	WB					
2	`lw $s2, 0x BB($s0)`		IF	ID	EX	MEM	WB				
3	`add $s3, $s1, $s2`			IF	ID	EX	MEM	WB			
4	`subi $s4, $s3, 8`				IF	ID	EX	MEM	WB		
5	`addi $s5, $s4, 4`					IF	ID	EX	MEM	WB	

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																
2. lw																
3. add																
4. subi																
5. addi																

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															
2. lw															
3. add															
4. subi															
5. addi															

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**

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

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:

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:

spazio libero per continuazione o brutta copia

spazio libero per continuazione o brutta copia