



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 – lunedì 14 febbraio 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 (5 punti) _____

esercizio 2 (4 punti) _____

esercizio 3 (5 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

esercizio n. 1 – linguaggio macchina

traduzione da C ad assembler

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*, (che ha valore **NON STANDARD**) e **si traducano** in linguaggio macchina le dichiarazioni delle variabili globali.
2. **Si descriva** l'area di attivazione della funzione *verify*, secondo il modello MIPS, e l'allocazione dei parametri e delle variabili locali della funzione *verify* 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 7
int VECTOR [N]
int points = N
int result
/* testate funzioni e procedure ausiliarie */
int getint (void)
void reset (int * addr)
/* funzione verify */
int verify (int dim, int level) {
    int count
    int * ptr
    for (count = 0; count < dim; count++) {
        ptr = &VECTOR[count]
        if (*ptr >= level) {
            points--
        } else {
            reset (ptr)
        } /* if */
    } /* for */
    return (dim + level - count)
} /* verify */
/* programma principale */
int main ( ) {
    result = verify (N, getint ( )) + 10
} /* main */
```

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

contenuto simbolico	indirizzo assoluto iniziale (in hex)	spiazzamento rispetto a <i>gp</i> = 0x 1000 4000 NON STANDARD	
			indirizzi alti
...			
RESULT			
POINTS			
VECTOR [N-1]			
...			
VECTOR [0]	0x 1000 0000		indirizzi bassi

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

	<code>.data</code>	<code>0x 1000 0000</code>	<code>// segmento dati statici standard</code>
N:	<code>.equ</code>	<code>7</code>	<code>// costante simbolica</code>
VECTOR:			
POINTS:			
RESULT:			

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

indirizzi alti

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)
// result = verify (N, getint ()) + 10
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
           // for (count = 0; count < dim; count++)

FOR:
           // ptr = &VECTOR[count]

           // if (*ptr >= level)

THEN:      // points--

ELSE:      // reset (ptr)

ENDIF:

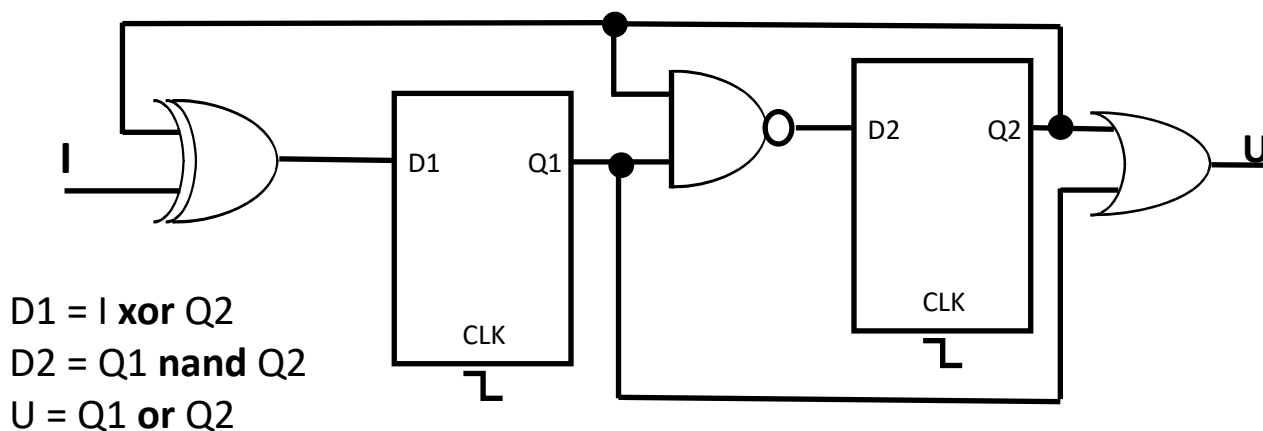
ENDFOR:    // return (dim + level - count)

           // ripristino registri - NON VANNO RIPORTATI - COMPLETARE SOTTO
           addiu    $sp, $sp,
           jr
```

esercizio n. 2 – logica digitale e memoria cache

prima parte – logica sequenziale

Sia dato il circuito sequenziale composto da **due bistabili master-slave di tipo D** (D_1, Q_1 e D_2, Q_2 , dove D è l'ingresso del bistabile e Q è lo stato / uscita del bistabile), **un ingresso I e un'uscita U** , descritto dal circuito seguente:



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 (come indicato anche in figura)

tabella dei segnali (diagramma temporale) da completare

- per i segnali D_1, Q_1, D_2, Q_2 e U , **ricavare**, 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 Q_1 e Q_2 sono già dati (rappresentano lo stato iniziale)

I	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1
D1																												
Q1	0	0	0	0																								
D2																												
Q2	0	0	0	0																								
U																												
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

seconda parte – memoria cache

Si consideri un sistema di memoria (memoria centrale + cache) con le caratteristiche seguenti:

- memoria di lavoro di **8 K byte**, indirizzata a livello di singolo byte
- cache associativa a gruppi (set-associative) a **2 vie**, da **1 K byte** in totale
- ogni blocco della cache contiene **256 byte**, dunque la cache contiene **4 blocchi**, denotati dalle lettere **A, B, C e D**
- i blocchi **A** e **B** appartengono all'**insieme 0**
- i blocchi **C** e **D** appartengono all'**insieme 1**
- la politica di sostituzione adottata nella cache è **LRU** (Least Recently Used)

Si chiede di completare la tabella seguente, considerando la sequenza di richieste alla memoria (lettura o scrittura non fa differenza) riportata nella colonna **indirizzo richiesto**.

Il significato delle diverse colonne della tabella è il seguente:

Nella colonna **esito** riportare **H** (hit) se il blocco richiesto si trova nella cache, oppure **M** (miss) se invece il blocco va caricato dalla memoria.

Nelle colonne **dati** va riportato il **numero del blocco della memoria** che si trova nel corrispondente blocco di cache. Si noti che questi valori vanno espressi come numeri **decimali**, mentre le etichette sono scritte in **binario**.

Nella colonna **azione** va indicato il **blocco a cui si accede** (in caso di successo **H**) o il blocco in cui vengono caricati i dati della memoria (in caso di fallimento **M**).

			insieme 0						insieme 1						
passo	indirizzo richiesto	esito	blocco A			blocco B			blocco C			blocco D			azione
			valido	etichetta	dati	valido	etichetta	dati	valido	etichetta	dati	valido	etichetta	dati	
0	–		1	1011	22	0	–	–	0	0001	3	1	0011	7	situazione iniziale
1	00011 1101 0110														
2	11100 0110 1010														
3	11001 1001 0011														
4	10110 1000 1001														
5	00001 1101 1101														
6	11000 0100 0110														

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 1A18(\$t0)	\$t0	0x 1001 2FAC
	lw \$t2, 0x 1818(\$t0)	\$t1	0x 0001 ABCD
	add \$t3, \$t3, \$t3	\$t2	0x 1004 1234
	addi \$t4, \$t1, 32	\$t3	0x 0048 0840
	ori \$t5, \$t2, 64	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:

		ciclo di clock										
		1	2	3	4	5	6	7	8	9	10	11
istruzione	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 – addi \$t4				IF	ID	EX	MEM	WB			
	5 – ori \$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 *addi* (addizione con immediato):

5) **Calcolare** il valore del risultato ($\$t2 \text{ OR } 64$) dell'istruzione *ori* (or logico 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 \$t1, 0x 1A18(\$t0)	IF	ID	EX	MEM	WB					
2	lw \$t2, 0x 1818(\$t0)		IF	ID	EX	MEM	WB				
3	add \$t3, \$t1, \$t2			IF	ID	EX	MEM	WB			
4	addi \$t4, \$t3, 32				IF	ID	EX	MEM	WB		
5	ori \$t5, \$t4, 64					IF	ID	EX	MEM	WB	

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

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. lw \$t1															
2. lw \$t2															
3. add															
4. addi															
5. ori															

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

rete combinatoria

Si vuole progettare la rete combinatoria **in prima forma canonica** (forma SOP, cioè somma di prodotti) della funzione booleana a tre ingressi e un'uscita descritta dalla seguente tabella di verità:

A	B	C	U
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) **Scrivere l'espressione booleana della prima forma canonica di U** in funzione di **A, B e C** (equivalente a una rete combinatoria a due livelli), senza ricorrere a semplificazioni o minimizzazioni.

(b) Si supponga di avere a disposizione porte logiche **OR** con **due** ingressi, porte **AND** con **due** ingressi e porte **NOT**, con ritardi di propagazione di **4 ns, 2 ns e 1 ns**, rispettivamente.

Determinare i ritardi delle possibili reti combinatorie con più di due livelli corrispondenti all'espressione di cui al punto (a), senza ricorrere a semplificazioni o minimizzazioni, ma sostituendo le porte AND e OR a più ingressi con circuiti composti da sole porte a due ingressi.

Forma canonica: _____

Ritardo: _____

bus del calcolatore

Si consideri un bus con queste ipotesi:

- Il bus è **sincrono** e svolge un'operazione (arbitraggio, lettura, scrittura) entro **un ciclo di clock**.
- Al bus sono collegate tre periferiche PER_1, PER_2 e PER_3, gestite tramite il meccanismo di Interrupt. Il collegamento è in daisy chain (festone), per il quale sono previsti i segnali di controllo **INT_REQ** attivo basso e **INT_ACK** attivo alto. Gli altri segnali di controllo sono **attivi alti**. Tutti i segnali hanno **ritardo di propagazione nullo**.

La periferica **PER_2**, al tempo = **5** nel diagramma sotto, lancia un Interrupt. Il processore è interrompibile e quindi può passare a eseguire la routine di interrupt corrispondente. Si svolgano i punti seguenti:

- 1) Il circuito del processore rileva le variazioni di valore del segnale INT_REQ con un ritardo di **5 ns**. La periferica PER_2 rileva lo acknowledge con un ritardo di **10 ns**. Il trasferimento del vettore di interrupt ha la durata di **15 ns**.

Si completi il diagramma temporale sotto (scrivendo 0 o 1 per i segnali singoli, e V per indicare il valore del vettore di interrupt), fino a quando il processore ha acquisito il vettore di interrupt e il bus è tornato nella situazione iniziale (numero di colonne non significativo):

operazione (transazione) di interrupt e trasferimento vettore									
INT_REQ	1	0							
INT_ACK	0	0							
lettura	0	0							
dato	X	X							
tempo (ns)	0	5	10	15	20	25	30	35	40

- 2) Quanto **tempo** occorre affinché il processore abbia tutte le informazioni necessarie per eseguire la routine di Interrupt ?

Tempo: _____

spazio libero per continuazione o brutta copia