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

prova di venerdì 4 novembre 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 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 2 h : 00 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (6 punti) _____

esercizio 2 (2 punti) _____

esercizio 3 (5,5 punti) _____

esercizio 4 (2,5 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – linguaggio macchina – RISC V

prima parte – traduzione da C a linguaggio macchina

Si deve tradurre in linguaggio macchina simbolico (assemblatore) ***RISC-V*** il frammento di programma C riportato sotto. Il modello di memoria è quello **standard *RISC-V*** e le variabili intere sono da **64 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**
- l'allocazione delle variabili in memoria non è allineata (non c'è frammentazione di memoria)

Si chiede di svolgere i quattro punti seguenti (usando le varie tabelle predisposte nel seguito):

1. **Si descriva** il segmento dei dati statici indicando gli indirizzi assoluti iniziali delle variabili globali e **si traducano** in linguaggio macchina le dichiarazioni delle variabili globali.
2. **Si descriva** l'area di attivazione della funzione `uphill`, secondo il modello RISC V, e l'allocazione dei parametri e delle variabili locali della funzione `uphill` 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** `uphill` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali                                    */
#define N 8                                /* costante da 32 bit */
#define START ...                          /* costante da 32 bit */
typedef long long int LONG
LONG height [N]          /* vettore di rilevamenti altimetrici  */
LONG climb = 0           /* dislivello positivo totale          */
/* testata procedura ausiliaria - è una procedura foglia       */
void altitude (LONG * point) /* rileva quota da altimetro      */
/* registra quota e calcola dislivello positivo totale         */
LONG uphill (LONG prev, LONG * next, LONG idx) {
   LONG step, retval
   altitude (next)
   step = *next - prev
   if (step <= 0) { step = 0 } /* if 1 */
   idx++
   if (idx == N) { retval = 0 }
   else {
      retval = uphill (*next, &height[idx], idx) + step
   } /* if 2 */
   return retval
}  /* uphill */
/* programma principale                                            */
void main ( ) {
   climb = uphill (START, height, 0)
}  /* main */
```

punto 1 – segmento dati statici

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
		indirizzi alti
CLIMB	*0x 0000 0000 1000 0040*	
HEIGTH [7]	*0x 0000 0000 1000 0038*	
...	...	
HEIGTH [0]	*0x 0000 0000 1000 0000*	indirizzi bassi

punto 1 – codice della sezione dichiarativa globale (numero di righe non significativo)			
	.eqv	N, 8	// costante numerica
	.eqv	START, ...	// costante numerica
	.data	0x 0000 0000 1000 0000	// seg. dati statici standard
HEIGHT:	***.space***	*64*	*// varglob HEIGHT (8 LONG cioè 64 byte)*
CLIMB:	***.dword***	*0*	*// varglob CLIMB (8 byte) inizializzata*

punto 2 – area di attivazione della funzione **`UPHILL`**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
ra salvato	*+16*	indirizzi alti
s0 salvato	*+8*	
s1 salvato	*0*	*← sp (fine area)*
a2 (arg PREV salvato)	*–8*	*← max est. pila*
		indirizzi bassi

La funzione UPHILL non è di tipo foglia, dunque essa salva in pila il registro ra (callee-saved). I registri s0 e s1 (callee-saved) vanno salvati in pila, dato che vengono usati per allocare, rispettivamente, le variabili locali STEP e RETVAL. Complessivamente l'area di attivazione di UPHILL ingombra tre LONG (cioè 24 byte).

La procedura ALTITUDE altera il registro di argomento a2, pertanto come da specifiche nel testo occorre salvarlo perché la funzione UPHILL lo utilizza dopo la chiamata alla procedura ALTITUDE.

punto 2 – allocazione dei parametri e delle variabili locali di **`UPHILL`** nei registri	
parametro o variabile locale	**registro**
prev	*a2*
next	*a3*
idx	*a4*
step	*s0*
retval	*s1*

punto 3 – codice dello statement riquadrato in **`MAIN`** (num. righe non significativo)
`// climb = uphill (START, height, 0)`
`MAIN:  li  a2, START  // prepara param PREV`
`  la  a3, HEIGHT  // prepara param NEXT`
`  mv  a4, zero  // prepara param IDX`
`  jal  UPHILL  // chiama funz UPHILL`
`  la  t0, CLIMB  // carica ind di varglob CLIMB`
`  sd  a0, (t0)  // aggiorna varglob CLIMB`

punto 4 – codice della funzione **UPHILL** (numero di righe non significativo)			
UPHILL:	**addi**	sp, sp, -24	// COMPLETARE - *crea area attivazione*
	// direttive EQV e salvataggio registri - NON VANNO RIPORTATI		
	// altitude (next)		
	addi	*sp, sp, -8*	*// inizia push*
	sd	*a2, (sp)*	*// push param PREV*
	mv	*a2, a3*	*// prepara param POINT*
	jal	*ALTITUDE*	*// chiama proc ALTITUDE*
	ld	*a2, (sp)*	*// pop param PREV*
	addi	*sp, sp, 8*	*// finisci push*
	// step = *next - prev		
	ld	*t0, (a3)*	*// carica oggetto puntato da NEXT*
	sub	*s0, t0, a2*	*// aggiorna varloc STEP*
IF1:	// if (step <= 0)		
	bgt	*s0, zero, ENDIF1*	*// se STEP > 0 vai a ENDIF1*
THEN1:	// step = 0		
	mv	*s0, zero*	*// azzera varloc STEP*
ENDIF1:	// idx++		
	addi	*a4, a4, 1*	*// aggiorna param IDX*
IF2:	// if (idx == N)		
	li	*t0, N*	*// carica costante N*
	bne	*a4, t0, ELSE2*	*// se IDX != N vai a ELSE2*
THEN2:	// retval = 0		
	mv	*s1, zero*	*// azzera varloc RETVAL*
	j	*ENDIF2*	*// vai a ENDIF2*
ELSE2:	// retval = uphill (*next, &height[idx], idx) + step		
	ld	*a2, (a3)*	*// prepara param PREV*
	la	*t0, HEIGHT*	*// carica ind di varglob HEIGHT*
	slli	*t1, a4, 3*	*// allinea indice IDX*
	add	*a3, t0, t1*	*// prepara param NEXT*
	jal	*UPHILL*	*// chiama funz UPHILL*
	add	*s1, a0, s0*	*// aggiorna varloc RETVAL*
ENDIF2:	// codice rimanente - NON VA RIPORTATO		

seconda parte – assemblaggio e collegamento

Dati i due moduli assemblatore seguenti, **si compilino** le tabelle (**in parte già compilate**) relative a:

1. i due moduli oggetto MAIN e LIBGCC (**aggiungendo le istruzioni e gli argomenti mancanti – si indichino direttamente qui i valori degli spiazzamenti nelle istruzioni autorilocanti**)
2. le basi di rilocazione del codice e dei dati di entrambi i moduli
3. la tabella globale dei simboli
4. la tabella di impostazione del calcolo delle costanti e degli spiazzamenti di istruzione e di dato (solo i calcoli che si ritengono necessari)
5. la tabella del codice eseguibile

modulo MAIN

```
        .data
        .eqv    LENGTH, 4098
CHAR:   .byte   0
BUF:    .space  56
        .text
        .globl  MAIN
MAIN:   mv      a2, zero
        lb      a3, (t0)
        li      a4, LENGTH
        jal     PRINT
        beq     a0, zero, END
CONT:   mv      t1, a0
        add     t1, t1, t1
        sd      t1, (t0)
END:    j       MAIN
```

modulo LIBGCC

```
        .data
        .eqv    OFFS, -1
ASCII:  .byte   1
        .text
        .globl  PRINT
PRINT:  beq     a3, zero, CONT
        la      t0, BUF
LOOP:   sd      a3, (t0)
        addi    a4, a4, OFFS
        bne     a4, zero, LOOP
        ret
```

Regola generale per la compilazione di **tutte** le tabelle contenenti codice:

- espandere **tutte** le pseudo-istruzioni
- 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, 0x 00F***
- 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: 28 hex (40 dec)

dimensione dati: 39 hex (57 dec)

testo

indirizzo di parola	istruzione (COMPLETARE)
0	**addi** a2, zero, 0
4	**lb** a3, (t0)
8	**lui** a4, ***0x 0000 1***
C	**addi** a4, a4, ***0x 002***
10	**jal** zero, ***0x 0 0000***
14	**beq** a0, zero, ***0x 008***
18	**addi** t1, a0, 0
1C	**add** t1, t1, t1
20	**sd** t1, (t0)
24	**jal** zero, ***0x F FFEE***

dati

indirizzo di parola	contenuto
0	*0x 00*
1	*non specificato*
39	

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

simbolo	tipo	valore
CHAR	**D**	**0x 0000 0000 0000 0000**
BUF	*D*	*0x 0000 0000 0000 0001*
MAIN	**T**	**0x 0000 0000 0000 0000**
CONT	*T*	*0x 0000 0000 0000 0018*
END	*T*	*0x 0000 0000 0000 0024*

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
10	***jal***	*PRINT*

modulo LIBGCC

dimensione testo: *1C hex (28 dec)*

dimensione dati: *01 hex (1 dec)*

testo

indirizzo di parola	istruzione (COMPLETARE)
0	***beq*** *a3, zero,* ***0x 000***
4	***auipc*** *t0,* ***0x 0000 0***
8	***addi*** *t0, t0,* ***0x 000***
C	***sd*** *a3, (t0)*
10	***addi*** *a4, a4,* ***0x FFF***
14	***bne*** *a4, zero,* ***0x FFC***
18	***jalr*** *zero, (rd)*
1C	
20	
24	

dati

indirizzo di parola	contenuto
0	*0x 01*
1	

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

simbolo	tipo	valore
ASCII	**D**	**0x 0000 0000 0000 0000**
PRINT	**T**	**0x 0000 0000 0000 0000**
LOOP	*T*	*0x 0000 0000 0000 000C*

tabella di rilocazione

indirizzo di parola	cod. operativo	simbolo
0	***beq***	*CONT*
4	***auipc***	*BUF*
8	***addi***	*BUF*

(2) – posizione in memoria dei moduli

modulo MAIN		modulo LIBGCC	
base del testo:	0x 0000 0000 0040 0000	base del testo:	*0x 0000 0000 0040 0028*
base dei dati:	0x 0000 0000 1000 0000	base dei dati:	*0x 0000 0000 1000 0039*

(3) – tabella globale dei simboli

simbolo	valore finale		simbolo	valore finale
CHAR	0x 0000 0000 1000 0000		ASCII	*0x 0000 0000 0040 0039*
BUF	*0x 0000 0000 1000 0001*		PRINT	*0x 0000 0000 0040 0028*
MAIN	0x 0000 0000 0040 0000		LOOP	*0x 0000 0000 0040 0034*
CONT	*0x 0000 0000 0040 0018*			
END	*0x 0000 0000 0040 0024*			

(4) tabella OPZIONALE per calcolare costanti e spiazzamenti di istruzione e di dato

RIPORTARE SOLO I CALCOLI CHE SI RITENGONO NECESSARI PER COMPRENDERE LA SOLUZIONE
(numero di righe non significativo)

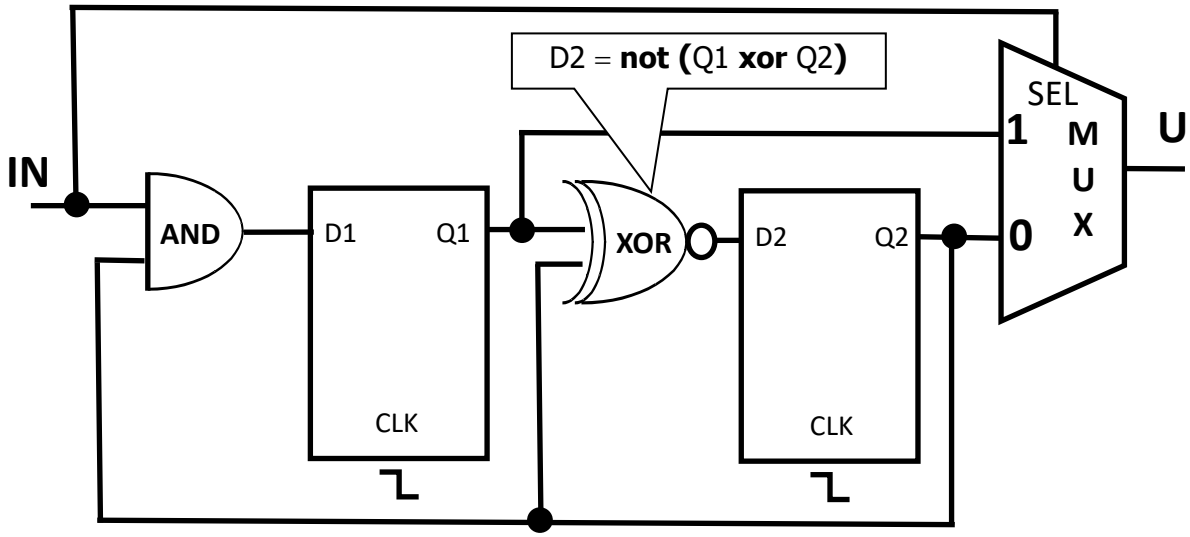
modulo MAIN	modulo LIBGCC
lui *%hi(LENGTH) = 0x 0000 0000 0000 1**0**02 = 0x 0000 **1** (20 bit sup + delta[11] che qui vale 0) – viene risolto già in fase di assemblaggio*	***beq*** *%pcrel(CONT) / 2 = (0x 0000 0000 0040 0018 – 0x 0000 0000 0040 0028) / 2 = 0x FFFF FFFF FFFF FFF0 / 2 == 0x FF8 (12 bit inf)*
addi *%lo(LENGTH) = 0x 0000 0000 0000 1002 = 0x 002 (12 bit inf) – viene risolto già in fase di assemblaggio*	***auipc*** *%pcrel_hi(BUF) = 0x 0000 0000 1000 0001 – 0x 0000 0000 0040 002C = 0x 0000 0000 0FBF F**F**D5 = 0x 0F**C0 0** (20 bit sup + delta[11] che qui vale 1)*
jal *%pcrel(PRINT) / 2 = (0x 0000 0000 0040 0028 – 0x 0000 0000 0040 0010) / 2 = 0x 0000 0000 0000 0018 / 2 = 0x 0 000C (20 bit inf)*	***addi*** *%pcrel_lo(BUF) = 0x 0000 0000 1000 0001 – 0x 0000 0000 0040 002C = 0x 0000 0000 0FBF F**F**D5 = 0x **F**D5 (12 bit inf)*
beq *%pcrel(END) / 2 = (0x 0000 0000 0000 0024 – 0x 0000 0000 0000 0014) / 2 = 0x 0000 0000 0000 0010 / 2 == 0x 008 (12 bit inf) – AUTORILOCANTE*	***bne*** *%pcrel(LOOP) / 2 = (0x 0000 0000 0000 000C – 0x 0000 0000 0000 0014) / 2 = 0x FFFF FFFF FFFF FFF8 / 2 == 0x FFC (12 bit inf) – AUTORILOCANTE*
jal *%pcrel(MAIN) / 2 = (0x 0000 0000 0000 0000 – 0x 0000 0000 0000 0024) / 2 = 0x FFFF FFFF FFFF FFDC / 2 = 0x F FFEE (20 bit inf) – AUTORILOCANTE*	

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

(5) – codice eseguibile	
testo	
indirizzo (hex)	**codice** (con codici operativi e registri in forma simbolica)
...	...
8	`lui  a4, 0x 0000 1  // MAIN: li LENGTH`
C	`addi  a4, a4, 0x 002  // MAIN: li LENGTH`
10	`jal  zero, 0x 0 000C  // MAIN: jal PRINT`
...	
28	`beq  a3, zero, 0x FF8  // PRINT: beq CONT`
2C	`auipc t0, 0x 0FC0 0  // PRINT: la BUF`
30	`addi  t0, t0, 0x FD5  // PRINT: la BUF`
...	...

esercizio n. 2 – logica digitale

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 IN 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 D1, Q1, D2, Q2 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 IN 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)

IN	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1
D1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
Q1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
D2	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
Q2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
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

esercizio n. 3 – microarchitettura – RISC V

prima parte – pipeline e segnali di controllo

Sono dati il seguente frammento di codice **macchina *RISC V*** (simbolico), che inizia l'esecuzione all'indirizzo indicato, e i valori iniziali per alcuni registri e parole di memoria – **notazione: 0^4 = 0000, e così via**.

indirizzo hex a 64 bit	codice RISC V
0^4 0^4 0040 0800	ld t2, 0x 018(t0)
0^4 0^4 0040 0804	ld t1, 0x 020(t0)
0^4 0^4 0040 0808	and t4, t3, t3
0^4 0^4 0040 080C	sd t2, 0x A10(t0)
0^4 0^4 0040 0810	beq t0, t1, 0x 040
0^4 0^4 0040 0814	

registro	contenuto iniziale hex a 64 bit
t0	0^4 0^4 1001 3FF8
t1	0^4 0^4 0001 2345
t2	0^4 0^4 0001 A00A
t3	0^4 0^4 0000 AAAA

memoria	contenuto iniziale hex a 64 bit
0^4 0^4 1001 4000	**********
0^4 0^4 1001 4008	**********
0^4 0^4 1001 4010	0^4 0^4 1001 1C1C *(t2 finale)*
0^4 0^4 1001 4018	0^4 0^4 1001 1A1A *(t1 finale)*

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

istruzione	1	2	3	4	5	6	7	8	9	10	11
1 – ld	IF	ID	EX	MEM	WB						
2 – ld		IF	ID	EX	MEM	WB					
3 – and			IF	ID	EX	MEM	WB				
4 - sd				IF	ID	EX	MEM	WB			
5 - beq					IF	ID	EX	MEM	WB		

Nelle seguenti domande 1, 2, 3 e 4, si calcolino solo i 32 bit meno significativi (8 cifre hex).

1) Calcolare il valore dell'indirizzo di memoria dati della prima istruzione *ld t2* (load):

1001 3FF8 + 0000 0018 = 1001 4010

2) Calcolare il valore dell'indirizzo di memoria dati della seconda istruzione *ld t1* (load):

1001 3FF8 + 0000 0020 = 1001 4018

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

1001 3FF8 + FFFF FA10 (est. segno da 12 a 32 bit) = 1001 3A08

4) Calcolare il valore dell'indirizzo di destinazione del salto (lo offset specificato in *beq* è a mezza parola):

0040 0810 + 0000 0040 x 2 = 0040 0810 + 0000 0080 = 0040 0890

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 – valori a 64 bit (16 cifre hex) (subito prima del fronte di SALITA del clock --- ciclo 5)			
IF	**ID**	**EX**	**MEM**
(beq)	***(sd)***	***(and)***	***(ld t1)***
registro IF/ID	***registro ID/EX***	***registro EX/MEM***	***registro MEM/WB***
	.WB.MemtoReg ***X***	.WB.MemtoReg ***0***	.WB.MemtoReg ***1***
	.WB.RegWrite ***0***	.WB.RegWrite ***1***	.WB.RegWrite ***1***
	.M.MemWrite ***1***	.M.MemWrite ***0***	
	.M.MemRead ***0***	.M.MemRead ***0***	
	.M.Branch ***0***	.M.Branch ***0***	
.PC ***0^4 0^4 0040 0810***	.PC ***0^4 0^4 0040 080C***	.PC **********	
.istruzione ***beq***	.(Rs1) ***(t0)*** ***0^4 0^4 1001 3FF8***		
	.(Rs2) ***(t2) finale*** ***0^4 0^4 1001 1C1C***	.(Rs2) **********	
	.Rd **********	.Rd ***t4***	.Rd ***t1***
	.imm/offset est. 64 bit ***F^4 F^4 FFFF FA10***	.ALU_out ***0^4 0^4 0000 AAAA***	.ALU_out *(ind di ld t1)* ***0^4 0^4 1001 4018***
	.EX.ALUSrc ***1***	.Zero ***0***	.DatoLetto *(t1 fin)* ***0^4 0^4 1001 1A1A***

segnali relativi a RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 ***t0 sd***	RF.DatoLetto1 ***0^4 0^4 1001 3FF8 (t0)***	RF.RegScrittura ***t2 ld***
RF.RegLettura2 ***t2 sd***	RF.DatoLetto2 ***0^4 0^4 0001 A00A (t2) iniz***	RF.DatoScritto ***0^4 0^4 1001 1C1C (t2) fin***

segnali relativi a RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 6)		
RF.RegLettura1 ***t0 beq***	RF.DatoLetto1 ***0^4 0^4 1001 3FF8 (t0)***	RF.RegScrittura ***t1 ld***
RF.RegLettura2 ***t1 beq***	RF.DatoLetto2 ***0^4 0^4 0001 2345 (t1) iniz***	RF.DatoScritto ***0^4 0^4 1001 1A1A (t1) fin***

seconda parte – gestione di conflitti e stalli

Supponendo che **la pipeline sia ottimizzata per la gestione dei conflitti di controllo,** 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	`ld t0, 0x 0AA(t3)`	IF	ID *3*	EX	MEM	WB *0*					
2	`add t2, t1, t0`		IF	ID *0, 1*	EX	MEM	WB *2*				
3	`sd t2, 0x 0BB(t3)`			IF	ID *2, 3*	EX	MEM	WB			
4	`add t4, t0, t0`				IF	ID *0, 0*	EX	MEM	WB *4*		
5	`beq t5, t4, 0x 080`					IF	ID *4, 5*	EX	MEM	WB	

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. ld	*IF*	*ID 3*	*EX*	*M*	*WB (0)*											
2. add		*IF*	*ID stall*	*ID stall*	*ID 0, 1*	*EX*	*M*	*WB (2)*								
3. sd			*IF stall*	*IF stall*	*IF*	*ID stall*	*ID stall*	*ID 2, 3*	*EX*	*M*	*WB*					
4. add						*IF stall*	*IF stall*	*IF*	*ID 0, 0*	*EX*	*M*	*WB (4)*				
5. beq									*IF*	*ID stall*	*ID stall*	*ID 4,5*	*EX*	*M*	*WB*	

punto 2

Si faccia l'ipotesi che la pipeline sia dotata dei percorsi di propagazione **EX/EX**, **EX/ID, MEM/EX** e **MEM/MEM**:

a. Disegnare in **diagramma B** il diagramma temporale della pipeline, indicando i percorsi di propagazione che possono 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 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. ld	*IF*	*ID 3*	*EX*	*M* ***(0)***	*WB (0)*										
2. add		*IF*	*ID stall*	*ID 0, 1*	***EX*** ***(2)***	*M (2)*	*WB (2)*								
3. sd			*IF stall*	*IF*	*ID 2, 3*	***EX***	*M*	*WB*							
4. add					*IF*	*ID 0, 0*	*EX* ***(4)***	*M (4)*	*WB (4)*						
5. beq						*IF*	*ID stall*	***ID*** *4, 5*	*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*	*t0*	*si*	*2*	*2*	*1 stallo + MEM/EX*	*5*
3	*2*	*t2*	*si*	*2*	*2*	*EX-EX(§)*	*6*
4	*1*	*t0*	*no*	*0*	*0*	*0*	*na*
5	*4*	*t4*	*si*	*2(*)*	*2(*)*	*1 stallo + EX/ID*	*8*

(*) Anche la soluzione che introduce 3 stalli è stata considerata corretta.

(§): Il percorso MEM/MEM potrebbe essere attivato al ciclo 7 tra le istruzioni 2 (add) e 3 (sd), propagando il registro t2 calcolato da add e poi scritto in memoria da sd; tuttavia il percorso EX/EX si attiva prima, al ciclo 6, pertanto il percorso MEM/MEM non si attiverà. La soluzione MEM/MEM è stata considerata cmq corretta.

esercizio n. 4 – domande su argomenti vari

memoria cache

Si consideri una gerarchia di memoria composta dalla memoria centrale da **4 Giga byte**, indirizzabile **a byte** con parole da **64 bit**, una memoria cache istruzioni da **512 K byte** e una memoria cache dati da **2 Mega byte**, entrambe **a indirizzamento diretto** con blocchi da **512 byte**.

Il tempo di acceso alle memorie 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 **64 bit**.

Rispondere alla **cinque** domande seguenti:

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

cache dati		
etichetta	indice di blocco	spiazzamento
11 bit (32 – 12 – 9)	*12 bit (4 K blocchi cache = 2^{12})*	*9 bit (512 byte = 2^9)*

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

   N. di parole per blocco = *512 byte / 8 byte = 64 parole*

   Tempo per caricare dalla memoria centrale in cache un blocco = *20 cicli + 5 cicli × 63 parole = 335 cicli di clock*

3. Viene mandato in esecuzione un **nuovo programma** che:
   a. accede **sequenzialmente** per **una volta** a un array di **4100 blocchi**, e poi
   b. esegue per **cinque volte** un ciclo accedendo **sequenzialmente** ai cinque blocchi **0, 1, 3, 4097, 4099**

   Calcolare:

   - **numero di miss totali alla cache dati** =

     4100 miss + 5 miss + (4 miss × 4 iterazioni) = 4121 miss (di blocco);

     infatti prima ci sono 4100 miss per l'accesso iniziale ai 4100 blocchi dell'array nel punto (a), poi ci sono 5 miss (relativi ai blocchi 0, 1, 3, 4097, 4099) per la prima iterazione del ciclo al punto (b), e infine ci sono 4 miss (relativi ai blocchi 1, 3, 4097, 4099) per ciascuna delle 4 iterazioni rimanenti del ciclo al punto (b)

   - **numero di accessi totali alla cache dati** =
     = (4100 + 5 × 5) × 64 = 4125 x 64 = 264 000 accessi (a parole)
     essendo ogni blocco di 64 parole, si hanno 4100 blocchi × 64 accessi a parola per il punto (a) e poi 5 blocchi × 5 iterazioni × 64 accessi a parola per il punto (b)
     oppure il numero di accessi ai blocchi = num. hit + num. di miss = 4 (hit al blocco 0) + 4121 miss = 4125 accessi (di blocco) => 4125 x 64 = 264 000 accessi (a parole).

   - **miss rate della cache dati** =
     n° miss totali / n° accessi totali = 4121 / 264 000 ≈ 0,015 = 1,56 %

4. **Calcolare** il **tempo medio** di accesso alla memoria di questo programma, considerando che il **miss rate** della cache **istruzioni** è pari al **2 %** e che la **percentuale di accessi ai dati** è pari al **20 %**.

   T medio accesso istruzioni = *hit time + miss rate istruzioni × miss penalty = 1 ciclo + 0,02 × 335 cicli = 7,7 cicli di clock*

   T medio accesso dati = *hit time + miss rate dati × miss penalty = 1 ciclo + 0,0156 × 335 cicli = 6,226 cicli di clock*

   T medio accesso alla memoria = *100 / 120 × T medio accesso istruzioni + 20 / 120 × T medio accesso dati = 100 / 120 × 7,700 cicli + 20 / 120 × 6,226 cicli = 6,42 + 1,04 = 7,46 cicli di clock*

5. Si supponga ora che la **cache dati** sia di tipo **completamente associativo** con **politica di sostituzione LRU**.

   Calcolare il numero di miss totali alla cache dati = *4100 miss + 3 miss = 4103 miss; infatti prima ci sono 4100 miss per l'accesso iniziale ai 4100 blocchi dell'array nel punto (a), e infine ci sono 3 miss alla prima iterazione (ricarica i blocchi 0, 1 e 3 sovrascrivendoli ai blocchi 4, 5 e 6) del ciclo al punto (b) (è la prima delle 5 iterazioni che leggono solo 5 blocchi), mentre le 4 iterazioni rimanenti del ciclo al punto (b) hanno solo hit*