



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 29 giugno 2023

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 **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

traduzione da C ad assembler

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 **è non 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 `getlist`, secondo il modello RISC V, e l’allocazione dei parametri e delle variabili locali della funzione `getlist` 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** `getlist` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali */
#define N 5 /* costante da 32 bit */
typedef long long int LONG
LONG LIST [N]
LONG count
/* testata procedura ausiliaria - è una procedura foglia */
void getnum (LONG * ptr) /* legge un numero da input */
/* funzione getlist */
LONG getlist (LONG * vect, LONG size) {
    LONG idx, num
    idx = 0
    num = 0
    while (idx < size) {
        getnum (&num) /* l'argomento è l'indirizzo di num */
        vect [idx] = num
        idx++
    } /* while */
    return idx
} /* getlist */
/* programma principale */
void main ( ) {
    count = getlist (LIST, N)
} /* main */
```

punto 1 – segmento dati statici

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
		indirizzi alti
COUNT	<i>0x 0000 0000 1000 0028</i>	
LIST	<i>0x 0000 0000 1000 0000</i>	indirizzi bassi

punto 1 – codice della sezione dichiarativa globale (numero di righe non significativo)			
	.eqv	N, 5	// costante numerica
	.data	0x 0000 0000 1000 0000	// seg. dati statici standard
LIST:	.space	40	// varglob LIST (vettore non inizializ.)
COUNT:	.space	8	// varglob COUNT (64 bit non inizializ.)

punto 2 – area di attivazione della funzione GETLIST		
contenuto simbolico	spiazz. rispetto a stack pointer	
<i>ra salvato</i>	<i>+16</i>	indirizzi alti
<i>s0 salvato</i>	<i>+8</i>	
<i>NUM (una parola da 8 byte)</i>	<i>0</i>	← <i>sp (fine area)</i>
<i>reg a2 (param VECT) salvato</i>		<i>max estensione pila di GETLIST</i>
		indirizzi bassi

La variabile locale NUM è acceduta per indirizzo dalla funzione *getnum*, dunque non può essere allocata in registro e va invece allocata in pila. La funzione *getlist* riutilizza l'argomento VECT (reg a2) dopo avere chiamata la funzione ausiliaria *getnum*, la quale ne fa uso, dunque la funzione *getlist* salva l'argomento in pila e lo ripristina a ogni iterazione del ciclo.

punto 2 – allocazione dei parametri e delle variabili locali di GETLIST nei registri	
parametro o variabile locale	registro
<i>vect</i>	<i>a2</i>
<i>size</i>	<i>a3</i>
<i>idx</i>	<i>s0</i>

punto 3 – codice dello statement riquadrato in MAIN (num. righe non significativo)	
// count = getlist (LIST, N)	
MAIN: la a2, LIST	// prepara param VECT
li a3, N	// prepara param SIZE
jal GETLIST	// chiama funz GETLIST
la t0, COUNT	// carica ind varglob COUNT
sd a0, (t0)	// aggiorna varglob COUNT

punto 4 – codice della funzione GETLIST (numero di righe non significativo)	
GETLIST:	addi sp, sp, -24 // COMPLETARE - crea area attivazione
	// direttive EQV - DA COMPLETARE
	.eqv RA, 16 // spi di reg ra salvato
	.eqv S0, 8 // spi di reg s0 salvato
	.eqv NUM, 0 // spi di varloc NUM allocata in pila
	// salvataggio registri - NON VA RIPORTATO
	// idx = 0
	mv s0, zero // inizializza varloc IDX
	// num = 0
	addi t0, sp, NUM // calcola ind di varloc NUM
	sd zero, (t0) // aggiorna varloc NUM
WHILE:	// while (idx < size)
	bge s0, a3, ENDWHILE // se idx >= size vai a ENDWHILE
	// getnum (&num)
	addi sp, sp, -8 // push reg a2
	sd a2, (sp) // fine push
	addi a2, sp, 8 // prepara param PTR (= &NUM)
	jal GETNUM // chiama proc GETNUM
	ld a2, (sp) // pop reg a2
	addi sp, sp, 8 // fine pop
	// vect [idx] = num
	addi t0, sp, NUM // calcola ind di varloc NUM
	ld t1, (t0) // carica varloc NUM
	slli t2, s0, 3 // allinea indice IDX
	add t2, a2, t2 // calcola ind di elem VECT [IDX]
	sd t1, (t2) // aggiorna elem VECT [IDX]
	// idx++
	addi s0, s0, 1 // aggiorna valoc IDX
	j WHILE // torna a WHILE
ENDWHILE:	// return idx
	mv a0, s0 // prepara valusc
	// ripristino registri e rientro - NON VA RIPORTATO

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 LIBMATH (**aggiungendo le istruzioni e gli argomenti mancanti – si indichino direttamente qui i valori degli spiazamenti nelle istruzioni autorilocanti**)
2. le basi di rilocalizzazione 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 spiazamenti di istruzione e di dato (solo i calcoli che si ritengono necessari)
5. la tabella del codice eseguibile

modulo MAIN	modulo LIBMATH
<pre>.eqv TOT, 4088 .data NUM: .dword 0 VECT: .space 160 .text .globl MAIN MAIN: .mv t0, a0 .ld a2, (t0) .li a3, TOT .beq a0, zero, MATH .jal MATH FOLLOW: .mv t1, a0 .beq t1, a3, FOLLOW END: .nop</pre>	<pre>.data SHORT: .word 255 .text .globl MATH MATH: .beq a3, zero, CONT .la t0, NUM CONT: .sd a2, (t0) .addi a1, a1, 17 .j MAIN</pre>

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 rilocalizzazione successiva, vanno posti a zero e avranno un valore definitivo nel codice eseguibile

(1) – moduli oggetto					
modulo MAIN		modulo LIBMATH			
dimensione testo: 24 hex (36 dec)		dimensione testo: 18 hex (24 dec)			
dimensione dati: A8 hex (168 dec)		dimensione dati: 04 hex (4 dec)			
testo		testo			
indirizzo di parola	istruzione (COMPLETARE)	indirizzo di parola	istruzione (COMPLETARE)		
0	addi t0, a0, 0	0	beq a3, zero, 0x006 (= +6)		
4	ld a2, (t0)	4	auipc t0, 0x 0000 0		
8	lui a3, 0x 0000 1	8	addi t0, t0, 0x 000		
C	addi a3, a3, 0x FF8	C	sd a2, (t0)		
10	beq a0, zero, 0x 000	10	addi a1, a1, 0x011		
14	jal ra, 0x 0 0000	14	jal zero, 0x 0 0000		
18	addi t1, a0, 0	18			
1C	beq t1, a3, 0x FFE (= -2)	1C			
20	addi zero, zero, 0	20			
24		24			
dati		dati			
indirizzo di parola	contenuto	indirizzo di parola	contenuto		
0	0x 0000 0000 0000 0000 (NUM)	0	0x 0000 00FF (SHORT)		
8	non specificato (VECT)	4			
A8					
tabella dei simboli		tabella dei simboli			
tipo può essere T(testo) oppure D(dato)		tipo può essere T(testo) oppure D(dato)			
simbolo	tipo	valore			
NUM	D	0x 0000 0000 0000 0000	SHORT	D	0x 0000 0000 0000 0000
VECT	D	0x 0000 0000 0000 0008	MATH	T	0x 0000 0000 0000 0000
MAIN	T	0x 0000 0000 0000 0000	CONT	T	0x 0000 0000 0000 000C
FOLLOW	T	0x 0000 0000 0000 0018			
END	T	0x 0000 0000 0000 0020			
tabella di rilocalizzazione		tabella di rilocalizzazione			
indirizzo di parola	cod. operativo	simbolo			
10	beq	MATH	4	auipc	NUM
14	jal	MATH	8	addi	NUM
			14	jal	MAIN

(2) – posizione in memoria dei moduli

modulo MAIN		modulo LIBMATH	
base del testo:	0x 0000 0000 0040 0000	base del testo:	0x 0000 0000 0040 0024
base dei dati:	0x 0000 0000 1000 0000	base dei dati:	0x 0000 0000 1000 00A8

(3) – tabella globale dei simboli

simbolo	valore finale	simbolo	valore finale
NUM	0x 0000 0000 1000 0000	SHORT	0x 0000 0000 1000 00A8
VECT	0x 0000 0000 1000 0008	MATH	0x 0000 0000 0040 0024
MAIN	0x 0000 0000 0040 0000	CONT	0x 0000 0000 0040 0030
FOLLOW	0x 0000 0000 0040 0018		
END	0x 0000 0000 0040 0020		

(4) tabella OPZIONALE per calcolare costanti e spiazziamenti 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 LIBMATH
lui %hi(TOT) = 0x 0000 0000 0000 0FF8 = 0x 0000 1 (20 bit sup + delta[11] che qui vale 1) - viene risolto già in fase di assemblaggio	beq %pcrel(CONT) / 2 = (0x 0000 0000 0000 000C - 0x 0000 0000 0000 0000) / 2 = 0x 0000 0000 0000 000C / 2 == 0x 006 (12 bit inf) - AUTORILOCANTE
addi %lo(TOT) = 0x 0000 0000 0000 0FF8 = 0x FF8 (12 bit inf) - viene risolto già in fase di assemblaggio	auipc %pcrel_hi(NUM) = 0x 0000 0000 1000 0000 - 0x 0000 0000 0000 0040 0028 = 0x 0000 0000 0FBF FFD8 = 0x 0FC0 0 (20 bit sup + delta[11] che qui vale 1)
beq %pcrel(MATH) / 2 = (0x 0000 0000 0000 0024 - 0x 0000 0000 0000 0010) / 2 = 0x 0000 0000 0000 0014 / 2 == 0x 00A (12 bit inf)	addi %pcrel_lo(NUM) = 0x 0000 0000 1000 0000 - 0x 0000 0000 0040 0028 = 0x 0000 0000 0FBF FFD8 = 0x FD8 (12 bit inf)
jal %pcrel(MATH) / 2 = (0x 0000 0000 0040 0024 - 0x 0000 0000 0040 0014) / 2 = 0x 0000 0000 0000 0010 / 2 = 0x 0 0008 (20 bit inf)	jal %pcrel(MAIN) / 2 = (0x 0000 0000 0040 0000 - x 0000 0000 0040 0038) / 2 = 0x FFFF FFFF FFFF FFC8 / 2 = 0x F FFE4 (20 bit inf)
beq %pcrel(FOLLOW) / 2 = (0x 0000 0000 0000 0018 - 0x 0000 0000 0000 001C) / 2 = 0x FFFF FFFF FFFF FFFC / 2 == 0x FFE (12 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	<i>lui</i> a3, 0x 0000 1 // MAIN: <i>li</i> TOT
C	<i>addi</i> a3, a3, 0x FF8 // MAIN: <i>li</i> TOT
10	<i>beq</i> a0, zero, 0x 00A // MAIN: <i>beq</i> MATH
14	<i>jal</i> ra, 0x 0 0008 // MAIN: <i>jal</i> MATH
...	...
1C	<i>beq</i> t1, a3, 0x FFE // MAIN: <i>beq</i> FOLLOW
...	...
24	<i>beq</i> a3, zero, 0x 006 // LIBMATH: <i>beq</i> CONT
28	<i>auipc</i> t0, 0x 0FC0 0 // LIBMATH: <i>la</i> NUM
2C	<i>addi</i> t0, t0, 0x FD8 // LIBMATH: <i>la</i> NUM
...	...
38	<i>jal</i> zero, 0x F FFE4 // LIBMATH: <i>j</i> MAIN
...	...

esercizio n. 2 – logica digitale

Sia dato il circuito sequenziale con 2 ingressi **I1** e **I2** descritto dalle equazioni logiche seguenti:

$$D1 = \text{not} (I1 \text{ xor } I2)$$

$$D2 = (\text{not } Q1) \text{ and } (Q2 \text{ or } I2)$$

$$U = Q1 \text{ and not } I1$$

Il circuito è composto dai **due** bistabili master / slave di tipo D (D1, Q1) e (D2, Q2), con Di ingresso del bistabile e Qi stato / uscita del bistabile.

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

- si devono trascurare completamente i ritardi di propagazione delle porte logiche AND e OR, e i ritardi di commutazione dei bistabili
- i bistabili sono di tipo master-slave con uscita che commuta sul fronte di discesa del clock

I1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0
I2	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	0	1	1	1	0	0	1	1	
D1	0	0	1	1	1	1	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	
Q1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0		
D2	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
Q2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
U	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	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 del processore pipeline

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: 0⁴ = 0000, e così via.**

indirizzo hex a 64 bit	codice RISC V
0 ⁴ 0 ⁴ 0040 0800	ld t1, 0x 08B(\$t0)
0 ⁴ 0 ⁴ 0040 0804	addi t2, t3, 32
0 ⁴ 0 ⁴ 0040 0808	sd t3, 0x 0AB(\$t0)
0 ⁴ 0 ⁴ 0040 080C	add t4, t1, t3
0 ⁴ 0 ⁴ 0040 0810	beq t0, t2, 0x 020
0 ⁴ 0 ⁴ 0040 0814	

registro	contenuto iniziale hex a 64 bit
t0	0 ⁴ 0 ⁴ 1001 4021
t1	0 ⁴ 0 ⁴ 0001 CCCC
t2	0 ⁴ 0 ⁴ 0001 80AA
t3	0 ⁴ 0 ⁴ 0010 800A
memoria	contenuto iniziale hex a 64 bit
0 ⁴ 0 ⁴ 1001 4004	0 ⁴ 0 ⁴ 1234 AA00
0 ⁴ 0 ⁴ 1001 4008	0 ⁴ 0 ⁴ 1001 1B1B
0 ⁴ 0 ⁴ 1001 40AC	0 ⁴ 0 ⁴ 1001 1A1A (t1 finale)
0 ⁴ 0 ⁴ 1001 40CC	0 ⁴ 0 ⁴ 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 – ld	IF	ID	EX	MEM	WB						
	2 – addi		IF	ID	EX	MEM	WB					
	3 – sd			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 *ld* (load):

$$1001\ 4021 + 0000\ 008B = 1001\ 40AC$$

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

$$0010\ 800A + 0000\ 0020 = 0010\ 802A\ (t2\ finale)$$

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

$$1001\ 4021 + 0000\ 00AB = 1001\ 40CC$$

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

$$0040\ 0810 + 0000\ 0020 \times 2 = 0040\ 0850$$

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
(beq)	(add)	(sd)	(addi)
registro IF/ID	registro ID/EX	registro EX/MEM	registro MEM/WB
	.WB.MemtoReg 0	.WB.MemtoReg X	.WB.MemtoReg 0
	.WB.RegWrite 1	.WB.RegWrite 0	.WB.RegWrite 1
	.M.MemWrite 0	.M.MemWrite 1	
	.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) (t1) finale 0 ⁴ 0 ⁴ 1001 1A1A		
	.(Rs2) (t3) 0 ⁴ 0 ⁴ 0010 800A	.(Rs2) (t3) 0 ⁴ 0 ⁴ 0010 800A	
	.Rd t4 1D	.Rd *****	.Rd t2 07
	.imm/offset est. 64-bit *****	.ALU_out ind mem sd 0 ⁴ 0 ⁴ 1001 40CC	.ALU_out (t2) finale 0 ⁴ 0 ⁴ 0010 802A
	.EX.ALUSrc 0	.Zero *****	.DatoLetto *****

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 t1 add	RF.DatoLetto1 0 ⁴ 0 ⁴ 0001 CCCC (t1) iniz.	RF.RegScrittura t1 ld
RF.RegLettura2 t3 add	RF.DatoLetto2 0 ⁴ 0 ⁴ 0010 800A (t3) iniz.	RF.DatoScritto 0 ⁴ 0 ⁴ 1001 1A1A (t1) fin.
segnali relativi al 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 4021 (t0) iniz.	RF.RegScrittura t2 addi
RF.RegLettura2 t2 beq	RF.DatoLetto2 0 ⁴ 0 ⁴ 0001 80AA (t2) iniz.	RF.DatoScritto 0 ⁴ 0 ⁴ 0010 802A (t2) fin.

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	ld	t1, 0x 140 (t0)	IF	ID 0	EX	MEM	WB 1					
2	ld	t2, 0x A1A (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	sd	t4, 0x 0CC (t0)					IF	ID 0, 4	EX	MEM	WB	

punto 1

- 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.
- 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
1.ld t1	IF	ID 0	EX	M	WB (1)										
2.ld t2		IF	ID 0	EX	M	WB (2)									
3.add t3			IF	ID stall	ID stall	ID 1, 2	EX	M	WB (3)						
4.add t4				IF stall	IF stall	IF	ID stall	ID stall	ID 3	EX	M	WB (4)			
5.sd t4							IF stall	IF stall	IF	ID stall	ID stall	ID 0, 4	EX	M	WB

punto 2

Si faccia l'ipotesi che la pipeline sia **ottimizzata** e dotata dei seguenti 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.ld t1	IF	ID 0	EX	M (1)	WB 1										
2.ld 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.sd t4						IF	ID 0, 4	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
3	1	t1	si	1	1	assorbito	–
3	2	t2	si	2	2	1 stallo + MEM / EX	6
4	3	t3	si	2	2	EX / EX	7
5	4	t4	si	2	2	EX / EX	8

esercizio n. 4 – memoria cache

prima parte – dimensionamento

Si consideri un sistema di memoria così costituito: mem. centrale + cache istruzioni + cache dati. Il sistema è caratterizzato dalle dimensioni seguenti:

memoria centrale da **4 K parole**

memoria cache **istruzioni** da **512 parole** a **indirizzamento diretto** (direct mapped)
ogni blocco di cache istruzioni contiene **256 parole**

memoria cache **dati** da **1 K parole completamente associativa** (fully associative)
ogni blocco di cache dati contiene **512 parole**

Si indichi la struttura degli indirizzi per la memoria cache istruzioni e per quella dati.

soluzione

*memoria centrale: indirizzo di **12** bit*

*memoria cache istruzioni: indirizzo di **9** bit*

*memoria cache dati: indirizzo di **10** bit*

cache istruzioni a indirizzamento diretto

8 bit per la parola nel blocco

1 bit per l'indice di blocco nella cache

3 bit di etichetta

cache dati completamente associativa

9 bit per la parola nel blocco

3 bit di etichetta

seconda parte – simulazione

Si riprenda il sistema di memoria descritto e dimensionato come prima. **Si chiede** di completare la tabella di simulazione data sotto.

Note per compilare la tabella di simulazione:

- in ciascuna cache (istruzioni e dati) ci sono due posizioni (o blocchi), indicate con **A** e **B**, che rappresentano i due alloggiamenti di cui dispone ciascuna cache per contenere blocchi di istruzioni o di dati, rispettivamente
- in indirizzo è indicato l'indirizzo a cui si riferisce l'operazione di memoria (prelievo o accesso a dato)
- in cache è indicata la memoria cache interessata dall'operazione: per il prelievo di istruzione è la cache istruzioni, indicata con **I**; per l'accesso a dato è la cache dati, indicata con **D**
- in esito va scritto **H** se è hit (successo), oppure **M** se è miss (fallimento) e bisogna caricare il blocco
- in valido va scritto il bit di validità della posizione di cache: **1** se valida oppure **0** se invalida
- in etichetta va scritta l'etichetta del blocco, denotata in binario con il numero di bit stabilito dal dimensionamento
- in blocco va scritto il numero del blocco di memoria centrale a cui si accede in cache o che viene caricato in essa; tale numero va denotato in decimale
- in azione va scritto se è accesso o caricamento, con il blocco di memoria e la posizione di cache coinvolti

				cache istruzioni						cache dati				
passo	indirizzo	cache	esito	posizione A			posizione B			posizione A		posizione B		azione
				valido	etichetta	blocco	valido	etichetta	blocco	valido	etichetta	valido	etichetta	
				0	–	–	1	101	11	1	010	1	111	situazione iniziale
1	1011 1111 1000	I	H				1	101	11					accedi a blocco 11 in posizione B
2	1100 0110 0000	I	M	1	110	12								carica blocco 12 in pos. A – poi accedi
3	0101 0010 1100	D	H							1	010			accedi a blocco 2 in posizione A
4	1100 0010 0001	I	H	1	110	12								accedi a blocco 12 in posizione A
5	0101 0011 0101	D	H							1	010			accedi a blocco 2 in posizione A
6	0110 1010 0101	D	M									1	011	LRU: carica blocco 3 in pos. B – poi accedi