



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 – martedì 12 settembre 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 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) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – linguaggio macchina

prima parte - 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, salvo sia specificato diversamente nel codice**. 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 *eval*, secondo il modello RISC V, e l’allocazione dei parametri e delle variabili locali della funzione *eval* 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** *eval* (vedi tab. 4 strutturata).

```
/* costanti e variabili globali */
#define N 9 /* costante da 32 bit */
typedef long long int LONG
int parity, diff = 0 /* interi da 32 bit */
LONG VECT [N]
/* funzione eval */
int eval (LONG * addr, int * res) {
    LONG idx, par, null
    par = 0
    null = N
    for (idx = 0; idx < N; idx++) {
        if (addr [idx] != 0) {
            par = par || 1 /* OR logico bit a bit */
        } else {
            par = par && 0 /* AND logico bit a bit */
            null--
        } /* if */
    } /* for */
    *res = (int) par /* cast da LONG a INT a 32 bit */
    return (N - null)
} /* eval */
/* programma principale */
int main ( ) {
    diff = eval (VECT, &parity)
} /* main */
```

punto 1 – segmento dati statici

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
VECT [N – 1]	<i>0x 0000 0000 1000 0048</i>	indirizzi alti
VECT [0]	<i>0x 0000 0000 1000 0008</i>	
DIFF	<i>0x 0000 0000 1000 0004</i>	
PARITY	<i>0x 0000 0000 1000 0000</i>	indirizzi bassi

punto 1 – codice della sezione dichiarativa globale (numero di righe non significativo)			
	.eqv	N, 9	// costante numerica
	.data	0x 0000 0000 1000 0000	// seg. dati statici standard
PARITY:	.space	4	// varglob PARITY (intero 32 bit non iniz.)
DIFF:	.word	0	// varglob DIFF (intero 32 bit iniz.)
VECT:	.space	72	// varglob VECT (9 x 8 byte = 72 byte)

punto 2 – area di attivazione della funzione EVAL		
contenuto simbolico	spiazz. rispetto a stack pointer	
<i>s0 salvato (idx)</i>	<i>+16</i>	indirizzi alti
<i>s1 salvato (par)</i>	<i>+8</i>	
<i>s2 salvato (null)</i>	<i>+0</i>	← <i>sp</i> (fine area)
		indirizzi bassi

La funzione `eval` è di tipo foglia e non salva in pila il registro di rientro `ra`. Ha tre variabili locali.

punto 2 – allocazione dei parametri e delle variabili locali di EVAL nei registri	
parametro o variabile locale	registro
<i>addr</i>	<i>a2</i>
<i>res</i>	<i>a3</i>
<i>idx</i>	<i>s0</i>
<i>par</i>	<i>s1</i>
<i>null</i>	<i>s2</i>

punto 3 – codice dello statement riquadrato in MAIN (num. righe non significativo)	
// diff = eval (VECT, &parity)	
MAIN: la a2, VECT	// prepara param ADDR
la a3, PARITY	// prepara param RES
jal EVAL	// chiama funz EVAL
la t0, DIFF	// carica ind varglob DIFF
sw a0, (t0)	// aggiorna varglob DIFF (a 32 bit)

punto 4 – codice della funzione EVAL (numero di righe non significativo)	
EVAL:	addi sp, sp, -24 // COMPLETARE - crea area attivazione
	// direttive EQV e salvataggio registri - NON VANNO RIPORTATI
	// par = 0
	mv s1, zero // aggiorna varloc PAR
	// null = N
	li s2, N // aggiorna varloc NULL
	// for (idx = 0; idx < N; idx++)
	mv s0, zero // inizializza varloc IDX
FOR:	li t0, N // carica cost N (32 bit)
	bge s0, t0, ENDOFR // se IDX >= N vai a ENDFOR
IF:	// if (addr [idx] != 0)
	slli t0, s0, 3 // allinea indice IDX
	add t1, a2, t0 // calcola ind di elem ADDR [IDX]
	ld t2, (t1) // carica elem ADDR [IDX]
	beq t2, zero, ELSE // se ADDR[IDX] == 0 vai a ELSE
THEN:	// par = par 1
	ori s1, s1, 1 // aggiorna varloc PAR
	j ENDIF // vai a ENDIF
ELSE:	// par = par && 0
	andi s1, s1, 0 // aggiorna varloc PAR
	// null--
	addi s2, s2, -1 // aggiorna varloc NULL
ENDIF:	addi s0, s0, 1 // aggiorna varloc IDX
	j FOR // vai a FOR
ENDFOR:	// *res = par
	sw s1, (a3) // aggiorna ogg. a 32 bit puntato da RES
	// return (N - null)
	li t0, N // carica cost N (a 32 bit)
	sub a0, t0, s2 // calcola espr. e prepara VALUSC
	// il ripristino dei registri non va riportato
	addi sp, sp, 24 // elimina area di attivazione
	jr ra // rientra a chiamante

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

modulo MAIN	modulo WIDGET
<pre>.eqv TOT, 1023 .data NUM1: .word 7 NUM2: .word 1000 VECT: .space 100 .text .globl MAIN MAIN: .addi t0, a0, TOT .li a3, BOX BEFORE: .beq a0, zero, AFTER .jal WIN .addi t1, a0, 0 AFTER: .bne t1, a3, PREV .addi zero, zero, 0</pre>	<pre>.data BOX: .dword 129 BASKET: .dword 0 .text .globl WIN WIN: .beq t0, a2, PREV .ld t1, (t0) .la t3, NUM2 PREV: .sd a2, (t0) .addi a1, a1, 33 .jal BEFORE</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 rilocazione successiva, vanno posti a zero e avranno un valore definitivo nel codice eseguibile

(1) – moduli oggetto							
modulo MAIN					modulo WIDGET		
dimensione testo: 20 hex (32 dec)					dimensione testo: 1C hex (28 dec)		
dimensione dati: 6C hex (108 dec)					dimensione dati: 10 hex (16 dec)		
testo					testo		
indirizzo di parola	istruzione (COMPLETARE)				indirizzo di parola	istruzione (COMPLETARE)	
0	addi	t0, a0, 0x 3FF (= +1023)			0	beq	t0, zero, 0x008 (= +8)
4	lui	a3, 0x 0000 0			4	ld	t1, (t0)
8	addi	a3, a3, 0x 000			8	auipc	t3, 0x 0000 0
C	beq	a0, zero, 0x 006 (= +6)			C	addi	t3, t3, 0x 000
10	jal	ra, 0x 0 0000			10	sd	a2, (t0)
14	addi	t1, a0, 0			14	addi	a1, a1, 0x021
18	bne	t1, a3, 0x 000			18	jal	ra, 0x 0 0000
1C	addi	zero, zero, 0			1C		
20					20		
dati					dati		
indirizzo di parola	contenuto				indirizzo di parola	contenuto	
0	0x 0000 0007 (NUM1)				0	0x 0000 0000 0000 0081 (BOX)	
4	0x 0000 03E8 (NUM2)				8	0x 0000 0000 0000 0000 (BASKET)	
8	non specificato (VECT)						
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			simbolo	tipo	valore
NUM1	D	0x 0000 0000 0000 0000			BOX	D	0x 0000 0000 0000 0000
NUM2	D	0x 0000 0000 0000 0004			BASKET	D	0x 0000 0000 0000 0008
VECT	D	0x 0000 0000 0000 0008			WIN	T	0x 0000 0000 0000 0000
BEFORE	T	0x 0000 0000 0000 000C			PREV	T	0x 0000 0000 0000 0010
AFTER	T	0x 0000 0000 0000 0018					
tabella di rilocazione					tabella di rilocazione		
indirizzo di parola	cod. operativo	simbolo			indirizzo di parola	cod. operativo	simbolo
4	lui	BOX			8	auipc	NUM2
8	addi	BOX			C	addi	NUM2
10	jal	WIN			18	jal	BEFORE
18	bne	PREV					

(2) – posizione in memoria dei moduli

modulo MAIN	modulo WIDGET
base del testo: 0x 0000 0000 0040 0000	base del testo: 0x 0000 0000 0040 0020
base dei dati: 0x 0000 0000 1000 0000	base dei dati: 0x 0000 0000 1000 006C

(3) – tabella globale dei simboli

simbolo	valore finale	simbolo	valore finale
NUM1	0x 0000 0000 1000 0000	BOX	0x 0000 0000 1000 006C
NUM2	0x 0000 0000 1000 0004	BASKET	0x 0000 0000 1000 0074
VECT	0x 0000 0000 1000 0008	WIN	0x 0000 0000 0040 0020
BEFORE	0x 0000 0000 0040 000C	PREV	0x 0000 0000 0040 0030
AFTER	0x 0000 0000 0040 0018		

(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(BOX) = 0x 0000 0000 1000 006C = 0x 1000 0 (20 bit sup + delta[11] che qui vale 1)	beq %pcrel(PREV) / 2 = (0x 0000 0000 0000 0030 - 0x 0000 0000 0000 0020) / 2 = 0x 0000 0000 0000 0010 / 2 == 0x 008 (12 bit inf) - AUTORILOCANTE
addi %lo(BOX) = 0x 0000 0000 1000 006C = 0x 06C (12 bit inf)	auipc %pcrel_hi(NUM2) = 0x 0000 0000 1000 0004 - 0x 0000 0000 0040 0028 = 0x 0000 0000 0FBF FDC = 0x 0FC0 0 (20 bit sup + delta[11] che qui vale 1)
beq %pcrel(AFTER) / 2 = (0x 0000 0000 0000 0018 - 0x 0000 0000 0000 000C) / 2 = 0x 0000 0000 0000 000C / 2 == 0x 006 (12 bit inf) - AUTORILOCANTE	addi %pcrel_lo(NUM2) = 0x 0000 0000 1000 0004 - 0x 0000 0000 0040 0028 = 0x 0000 0000 0FBF FFDC = 0x FDC (12 bit inf)
jal %pcrel(WIN) / 2 = (0x 0000 0000 0040 0020 - 0x 0000 0000 0040 0010) / 2 = 0x 0000 0000 0000 0010 / 2 = 0x 00008 (20 bit inf)	jal %pcrel(BEFORE) / 2 = (0x 0000 0000 0040 000C - 0x 0000 0000 0040 0038) / 2 = 0x FFFF FFFF FFFF FFD4 / 2 = 0x F FFEA (20 bit inf)
bne %pcrel(PREV) / 2 = (0x 0000 0000 0040 0030 - 0x 0000 0000 0040 0018) / 2 = 0x 0000 0000 0000 0018 / 2 == 0x 00C (12 bit inf)	

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)
...	...
4	<i>lui</i> a3, 0x 1000 0 // MAIN: <i>li</i> BOX
8	<i>addi</i> a3, a3, 0x 06C // MAIN: <i>li</i> BOX
C	<i>beq</i> a0, zero, 0x 006 // MAIN: <i>beq</i> AFTER
10	<i>jal</i> ra, zero, 0x 0 0008 // MAIN: <i>jal</i> WIN
18	<i>bne</i> t1, a3, 0x 0 00C // MAIN: <i>bne</i> PREV
...	...
20	<i>beq</i> t0, zero, 0x 008 // WIDGET: <i>beq</i> PREV
...	...
28	<i>auipc</i> t3, 0x 0FC0 0 // WIDGET: <i>la</i> NUM2
2C	<i>addi</i> t3, t3, 0x FDC // WIDGET: <i>la</i> NUM2
...	...
38	<i>jal</i> ra, 0x F FFEA // WIDGET: <i>jal</i> BEFORE
...	...

esercizio n. 2 – logica digitale

logica sequenziale

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

$$D1 = I1 \text{ or } Q2$$

$$D2 = (\text{not } Q1) \text{ xor } I2$$

$$U = \text{not } (D2 \text{ xor } I1)$$

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	0	0	1	1	0	0
I2	0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1	1	0	1	1	0	0	0	1	1
D1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Q1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
D2	1	1	0	0	1	1	1	0	1	0	1	1	1	1	0	1	0	1	1	1	1	0	1	1	0	0	1	1
Q2	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
U	1	1	0	0	1	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	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	registro	contenuto iniz - hex 64 bit
0 ⁴ 0 ⁴ 0040 0800	ld t1, 0x 0BB(t4)	t0	0 ⁴ 0 ⁴ 0110 A010
0 ⁴ 0 ⁴ 0040 0804	sd t3, 0x 0A7(t2)	t1	0 ⁴ 0 ⁴ 0000 1111
0 ⁴ 0 ⁴ 0040 0808	nop	t2	0 ⁴ 0 ⁴ 1060 3455
0 ⁴ 0 ⁴ 0040 080C	add t5, t1, t2	t3	0 ⁴ 0 ⁴ 0050 0000
0 ⁴ 0 ⁴ 0040 0810	addi t6, t1, 0x 001	t4	0 ⁴ 0 ⁴ 1060 0B55
0 ⁴ 0 ⁴ 0040 0814		memoria	contenuto iniz - hex 64 bit
		0 ⁴ 0 ⁴ 1060 0C10	0 ⁴ 0 ⁴ 0044 0FFF (t1 finale)
		0 ⁴ 0 ⁴ 1060 0C14	0 ⁴ 0 ⁴ 11FF 0040
		0 ⁴ 0 ⁴ 1060 34FC	0 ⁴ 0 ⁴ 48F0 6610

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 – sd		IF	ID	EX	MEM	WB					
	3 – nop			IF	ID	EX	MEM	WB				
	4 - add				IF	ID	EX	MEM	WB			
	5 - addi					IF	ID	EX	MEM	WB		

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

$$0^4 \ 0^4 \ 1060 \ 0B55 + 0^4 \ 0^4 \ 0000 \ 00BB = 0^4 \ 0^4 \ 1060 \ 0C10 \ \underline{\hspace{2cm}}$$

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

$$0^4 \ 0^4 \ 1060 \ 3455 + 0^4 \ 0^4 \ 0000 \ 00A7 = 0^4 \ 0^4 \ 1060 \ 34FC \ \underline{\hspace{2cm}}$$

3) **Calcolare** il valore del risultato (t1 + t2) dell'istruzione *add* (addizione):

$$0^4 \ 0^4 \ 0044 \ 0FFF + 0^4 \ 0^4 \ 1060 \ 3455 = 10A4 \ 4454 \ (t5 \ finale) \ \underline{\hspace{2cm}}$$

4) **Calcolare** il valore del risultato (t1 + 0x001) dell'istruzione *addi* (addizione con immediato):

$$0^4 \ 0^4 \ 0044 \ 0FFF + 0^4 \ 0^4 \ 0000 \ 0001 = 0044 \ 1000 \ (t6 \ finale) \ \underline{\hspace{2cm}}$$

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
<i>(addi)</i>	<i>(add)</i>	<i>(nop)</i>	<i>(sd)</i>
registro IF/ID	registro ID/EX	registro EX/MEM	registro MEM/WB
	.WB.MemtoReg <i>0</i>	.WB.MemtoReg <i>X</i>	.WB.MemtoReg <i>X</i>
	.WB.RegWrite <i>1</i>	.WB.RegWrite <i>0</i>	.WB.RegWrite <i>0</i>
	.M.MemWrite <i>0</i>	.M.MemWrite <i>0</i>	
	.M.MemRead <i>0</i>	.M.MemRead <i>0</i>	
	.M.Branch <i>0</i>	.M.Branch <i>0</i>	
.PC <i>0⁴ 0⁴ 0040 0810</i>	.PC <i>0⁴ 0⁴ 0040 080C</i>	.PC *****	
.istruzione <i>addi</i>	.(Rs1) <i>(t1) finale</i> <i>0⁴ 0⁴ 0044 0FFF</i>		
	.(Rs2) <i>(t2)</i> <i>0⁴ 0⁴ 1060 3455</i>	.(Rs2) *****	
	.Rd <i>t5 1E</i>	.Rd *****	.Rd *****
	.imm/offset est. 64-bit *****	.ALU_out *****	.ALU_out <i>ind mem sd</i> <i>0⁴ 0⁴ 1060 34FC</i>
	.EX.ALUSrc <i>0</i>	.Zero *****	.DatoLetto *****

segnali relativi al RF (subito prima del fronte di DISCESA interno al ciclo di clock – ciclo 5)		
RF.RegLettura1 <i>t1 06 add</i>	RF.DatoLetto1 <i>0⁴ 0⁴ 0000 1111 (t1) iniziale</i>	RF.RegScrittura <i>t1 06 ld</i>
RF.RegLettura2 <i>t2 07 add</i>	RF.DatoLetto2 <i>0⁴ 0⁴ 1060 3455 (t2)</i>	RF.DatoScritto <i>0⁴ 0⁴ 0044 0FFF (t1) finale</i>

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	sd	t1, 0x 00A (t0)	IF	ID 0, 1	EX	MEM	WB					
2	ld	t2, 0x 00B (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 00C (t0)					IF	ID 0, 4	EX	MEM	WB	

Si risponda alle domande seguenti:

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	16
1. sd	IF	ID 0, 1	EX	M	WB											
2. ld		IF	ID 0	EX	M	WB 2										
3. add			IF	ID stall	ID stall	ID 1, 2	EX	M	WB 3							
4. add				IF stall	IF stall	IF	ID stall	ID stall	ID 3	EX	M	WB 4				
5. sd							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**, **MEM / MEM**:

- 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.
- 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 B

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.sd t1	IF	ID 0, 1	EX	M	WB										
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	2	t2	si	2	2	1 stallo + MEM / EX	6
4	3	t3	sì	2	2	EX / EX	7
5	4	t4	si	2	2	EX / EX	8

esercizio n. 4 – domande su argomenti vari

memoria cache

Si consideri un sistema di memoria costituito da una memoria centrale di **32 blocchi** e da una memoria cache **dati** di **8 blocchi** di tipo **set-associativa** (associativa a **gruppi** o a **insiemi**) a **2 vie**. Tale memoria cache utilizza un algoritmo di sostituzione del blocco di tipo **LRU** e una strategia di gestione della scrittura di tipo **write-back** (scrittura differita), **ossia il blocco di cache modificato viene scritto in memoria centrale solo quando occorre sostituirlo in cache**.

Gli indirizzi dei blocchi di memoria siano espressi in notazione decimale come segue:

indirizzi dei blocchi di memoria: [0, 1, 2, 3, 4, 5, 6, 7,, 31]₁₀

Gli indirizzi dei blocchi di cache siano espressi in lettere come segue:

indirizzi dei blocchi di cache: [a, b, c, d, e, f, g, h]

Essendo la cache set-associativa a 2-vie, è organizzata in **4 set (insiemi)** ciascuno contenente **2 blocchi**:

set_0: [a, b]

set_1: [c, d]

set_2: [e, f]

set_3: [g, h]

- 1) Inizialmente la cache dati è **vuota**. La CPU esegue la serie di dieci accessi (in lettura o scrittura) ai blocchi riportata nella seguente tabella. **Si chiede** di completare la tabella di simulazione seguente:

numero accesso	lettura o scrittura	indirizzo blocco di memoria	esito HIT o MISS	etichetta (tag)	blocco di cache	set	write-back in memoria
1	scrittura	[12] ₁₀	MISS	[011] ₂	a	set_0	no
2	lettura	[12] ₁₀	HIT	[011] ₂	a	set_0	no
3	lettura	[10] ₁₀	MISS	[010] ₂	e	set_2	no (blocco di cache era vuoto)
4	scrittura	[10] ₁₀	HIT	[010] ₂	e	set_2	no (è uno hit, non un miss)
5	scrittura	[26] ₁₀	MISS	[110] ₂	f	set_2	no (blocco di cache era vuoto)
6	lettura	[14] ₁₀	MISS	[011] ₂	e	set_2	sì – scrittura in Mem [10] ₁₀
7	scrittura	[30] ₁₀	MISS	[111] ₂	f	set_2	sì – scrittura in Mem [26] ₁₀
8	scrittura	[16] ₁₀	MISS	[100] ₂	b	set_0	no (blocco di cache era vuoto)
9	lettura	[18] ₁₀	MISS	[100] ₂	e	set_2	no (blocco 14 non modificato)
10	lettura	[24] ₁₀	MISS	[110] ₂	a	set_0	sì – scrittura in Mem [12] ₁₀

- 2) Calcolare la frequenza di miss (o Miss Rate):

$$\text{Miss Rate} = \text{numero di miss} / \text{numero di accessi a memoria} = 8 / 10 = 0,8 = 80 \%$$

soluzione – la tabella è stata compilata utilizzando la seguente tabella di mappatura tra indirizzi

etichetta	set_0 [a, b]	set_1 [c, d]	set_2 [e, f]	set_3 [g, h]
000	0	1	2	3
001	4	5	6	7
010	8	9	10	11
011	12	13	14	15
100	16	17	18	19
101	20	21	22	23
110	24	25	26	27
111	28	29	30	31