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 – mercoledì 8 febbraio 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 (5 punti) _____

esercizio 2 (2 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (3 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 `compute`, secondo il modello RISC V, e l'allocazione dei parametri e delle variabili locali della funzione `compute` 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** `compute` (vedi tab. 4 strutturata).

```
/* costanti e variabili globali                                    */
#define N 6                                  /* costante da 32 bit */
typedef long long int LONG
LONG point = 0                             /* punto della primitiva */
LONG value                                /* valore della primitiva */
/* testate funzioni ausiliarie - sono procedure foglia             */
LONG getnum ( )    /* legge un numero da input - no argomenti */
LONG funct (LONG coord)          /* dà i valori della funzione */
/* calcola funzione integrale e ne restituisce un punto           */
LONG compute (LONG num, LONG * ptr) {
   LONG idx, acc                    /* variabili locali scalari */
   LONG TMP [N]                     /* variabile locale vettore */
   acc = 0
   for (idx = num; idx >= 0, idx--) {
      acc = acc + funct (idx)
      TMP [idx] = acc
   } /* for */
   return TMP [*ptr]
}  /* compute */
/* programma principale                                           */
void main ( ) {
   value = compute (getnum ( ), &point)
}  /* main */
```

punto 1 – segmento dati statici

contenuto simbolico	indirizzo assoluto iniziale (in hex)	
		indirizzi alti
VALUE	*0x 0000 0000 1000 0008*	
POINT	*0x 0000 0000 1000 0000*	indirizzi bassi

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

```
            .eqv      N, 6                    // costante numerica
            .data     0x 0000 0000 1000 0000  // seg. dati statici standard
POINT:      .dword    0              // varglob POINT (64 bit) inizializ. a 0
VALUE:      .space    8              // varglob VALUE (64 bit) non inizializ.
```

punto 2 – area di attivazione della funzione **COMPUTE**		
contenuto simbolico	**spiazz. rispetto a stack pointer**	
ra salvato	*+64*	indirizzi alti
s0 salvato	*+56*	
s1 salvato	*+48*	
TMP (6 parole da 8 byte)	*0*	← *sp (fine area)*
		indirizzi bassi

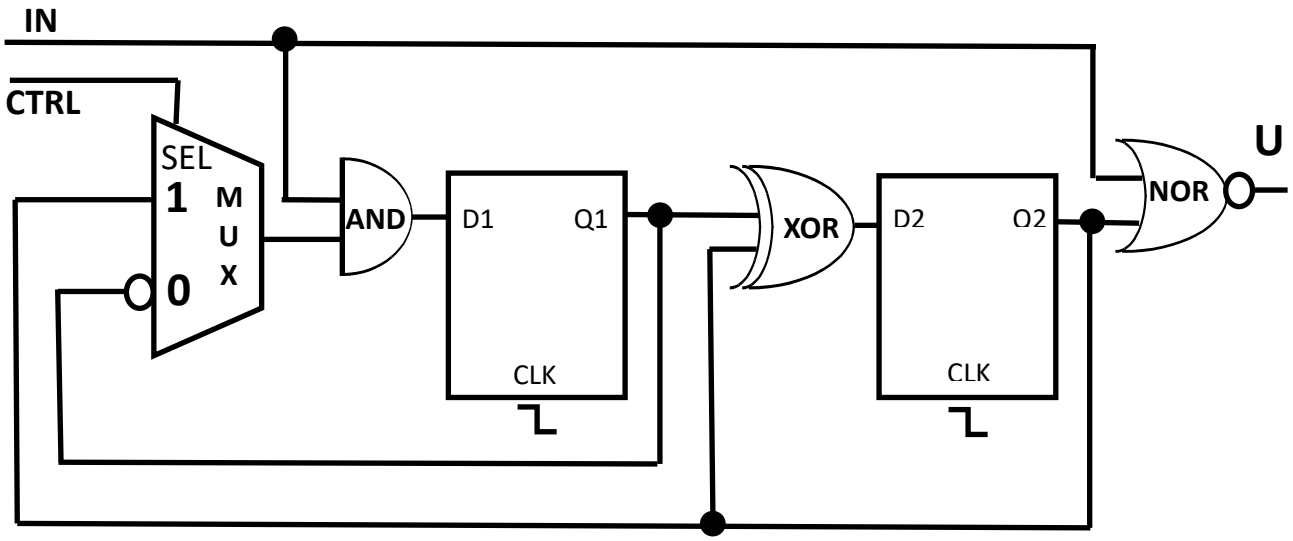
punto 2 – allocazione dei parametri e delle variabili locali di **COMPUTE** nei registri	
parametro o variabile locale	**registro**
num	*a2*
ptr	*a3*
idx	*s0*
acc	*s1*

punto 3 – codice dello statement riquadrato in **MAIN** (num. righe non significativo)
`// value = compute (getnum ( ), &point)`
`MAIN:  jal  GETNUM  // chiama funz GETNUM (senza arg)`
`  mv  a2, a0  // prepara param NUM`
`  la  a3, POINT  // prepara param PTR`
`  jal  COMPUTE  // chiama funz COMPUTE`
`  la  t0, VALUE  // carica ind di varglob VALUE`
`  sd  a0, (t0)  // aggiorna varglob VALUE`

punto 4 – codice della funzione **COMPUTE** (numero di righe non significativo)		
COMPUTE:	**addi** sp, sp, -72	// COMPLETARE - crea area attivazione
	// direttive EQV	
	.eqv RA, 64	// spi di reg ra
	.eqv S0, 56	// spi di reg s0
	.eqv S1, 48	// spi di reg s1
	.eqv TMP, 0	// spi di (base) vettore TMP
	// salvataggio registri - NON VA RIPORTATO	
	// acc = 0	
	mv s1, zero	// inizializza varloc ACC
	// for (idx = num; idx >= 0, idx--)	
	mv s0, a2	// inizializza varloc IDX
FOR:	**blt** s0, zero, ENDFOR	// se IDX < 0 vai a ENDFOR
	// acc = acc + funct (idx)	
	mv a2, s0	// prepara param COORD
	jal FUNCT	// chiama funz FUNCT
	add s1, s1, a0	// calcola espr e aggiorna varloc ACC
	// TMP [idx] = acc	
	slli t0, s0, 3	// allinea indice IDX
	add t0, sp, t0	// calcola ind di elem TMP[IDX]
	sd s1, TMP(t0)	// aggiorna elem TMP[IDX]
	// idx--	
	addi s0, s0, -1	// aggiorna valoc IDX
	j FOR	// torna a FOR
ENDFOR:	// return TMP [*ptr]	
	ld t0, (a3)	// carica oggetto puntato da PTR
	slli t0, t0, 3	// allinea indice *PTR
	add t0, sp, t0	// calcola ind di elem TMP[*PTR]
	ld a0, TMP(t0)	// prepara valusc TMP[*PTR]
	// codice rimanente - NON VA RIPORTATO	

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), **due ingressi IN e CTRL, 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 i segnali di ingresso CTRL e 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
- si noti che nel primo intervallo i segnali Q1 e Q2 sono già dati (rappresentano lo stato iniziale)

CTRL	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
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*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
Q1	0	0	0	0	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
D2	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
Q2	0	0	0	0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
U	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*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

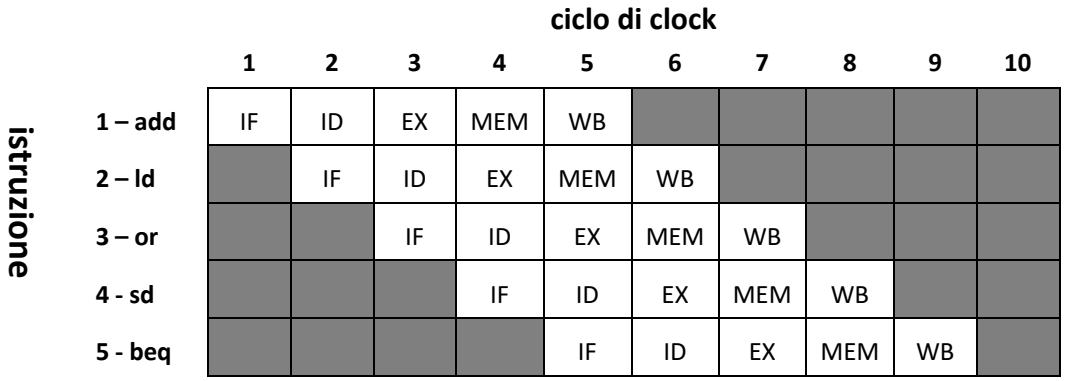
Sono dati il seguente frammento di codice **macchina *RISC V*** (simbolico), che inizia l'esecuzione all'indirizzo indicato, e i valori iniziali per alcuni registri e parole di memoria – **notazione: 0^4 = 0000, e così via**.

indirizzo hex a 64 bit	codice RISC V
0^4 0^4 0040 0800	**add** t2, t1, t2
0^4 0^4 0040 0804	**ld** t1, 0x 024(t0)
0^4 0^4 0040 0808	**or** t4, t3, t0
0^4 0^4 0040 080C	**sd** t2, 0x 02C(t0)
0^4 0^4 0040 0810	**beq** t0, t1, 0x 020
0^4 0^4 0040 0814	

registro	contenuto iniziale hex a 64 bit
t0	0^4 0^4 1001 3FEC
t1	0^4 0^4 0000 1100
t2	0^4 0^4 10AA 0248
t3	0^4 0^4 EFFE C013
t4	0^4 0^4 A0A0 B4B4

memoria	contenuto iniziale hex a 64 bit
0^4 0^4 1001 4000	0^4 0^4 1234 AA00
0^4 0^4 1001 4008	0^4 0^4 1001 1B1B
0^4 0^4 1001 4010	0^4 0^4 1001 AABB *(t1 finale)*
0^4 0^4 1001 4018	0^4 0^4 1234 89CC

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:



Nelle seguenti domande 1, 2, 3, 4 e 5, si calcolino solo i 32 bit meno significativi (8 cifre hex):

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

0000 1100 + 10AA 0248 = 10AA 1348 (t2 finale) _____

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

*1001 3FEC + 0000 0024 = 1001 4010*_____

3) Calcolare il valore del risultato (*t3* OR *t0*) dell'istruzione *or* (somma logica):

EFFE C013 + 1001 3FEC = FFFF FFFF (t4 finale) _____

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

*1001 3FEC + 0000 002C = 1001 4018*_____

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

*0040 0810 + 0000 0020 x 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 – valori a 64 bit (16 cifre hex) (subito prima del fronte di SALITA del clock – ciclo 5)			
IF	**ID**	**EX**	**MEM**
(beq)	***(sd)***	***(or)***	***(ld)***
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 3FEC***		
	.(Rs2) ***(t2) finale*** ***0^4 0^4 10AA 1348***	.(Rs2) **********	
	.Rd **********	.Rd ***t4***	.Rd ***t1***
	.imm/offset est. 64 bit ***0^4 0^4 0000 002C***	.ALU_out ***(t4) finale*** ***0^4 0^4 FFFF FFFF***	.ALU_out ***(ind di ld t1)*** ***0^4 0^4 1001 4010***
	.EX.ALUSrc ***1***	.Zero ***0***	.DatoLetto ***(t1 finale)*** ***0^4 0^4 1001 AABB***

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 3FEC (t0)***	RF.RegScrittura ***t2 add***
RF.RegLettura2 ***t2 sd***	RF.DatoLetto2 ***0^4 0^4 10AA 0248 (t2 iniziale)***	RF.DatoScritto ***0^4 0^4 10AA 1348 (t2 finale)***

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 3FEC (t0)***	RF.RegScrittura ***t1 ld***
RF.RegLettura2 ***t1 beq***	RF.DatoLetto2 ***0^4 0^4 0000 1100 (t1 iniziale)***	RF.DatoScritto ***0^4 0^4 1001 AABB (t1 finale)***

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	`add t1, t1, t2`	IF	ID *1, 2*	EX	MEM	WB *1*					
2	`add t2, t1, t3`		IF	ID *1, 3*	EX	MEM	WB *2*				
3	`ld t0, 0x 140(t2)`			IF	ID *2*	EX	MEM	WB *0*			
4	`sd t1, 0x A1A(t0)`				IF	ID *0, 1*	EX	MEM	WB		
5	`beq t0, t1, 0x 040`					IF	ID *0, 1*	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. add	*IF*	*ID 1, 2*	*EX*	*M*	*WB (1)*											
2. add		*IF*	*ID stall*	*ID stall*	*ID 1, 3*	*EX*	*M*	*WB (2)*								
3. ld			*IF stall*	*IF stall*	*IF*	*ID stall*	*ID stall*	*ID 2*	*EX*	*M*	*WB (0)*					
4. sd						*IF stall*	*IF stall*	*IF*	*ID stall*	*ID stall*	*ID 0, 1*	*EX*	*M*	*WB*		
5. beq									*IF stall*	*IF stall*	*IF*	*ID 0, 1*	*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. add	*IF*	*ID 1, 2*	*EX* ***(1)***	*M (1)*	*WB (1))*										
2. add		*IF*	*ID 1, 3*	***EX*** ***(2)***	*M (2)*	*WB (2)*									
3. ld			*IF*	*ID 2*	***EX***	*M* ***(0)***	*WB (0)*								
4. sd				*IF*	*ID stall*	*ID 0, 1*	***EX***	*M*	*WB*						
5. beq					*IF stall*	*IF*	*ID 0, 1*	*EX*	*M*	*WB*					

Tabella 1

punto 1a				
N° istruzione	**N° istruzione da cui dipende**	**registro coinvolto**	**conflitto (si/no)**	**N° stalli teorici**
2	*1*	*t1*	*si*	*2*
3	*2*	*t2*	*si*	*2*
4	*1*	*t1*	*no*	*0*
4	*3*	*t0*	*si*	*2*
5	*1*	*t1*	*no*	*0*
5	*3*	*t0*	*si*	*1*

punto 1b
N° stalli effettivi
2
2
0
2
0
0

punto 2b	
stalli + percorso di propagazione	**ciclo di clock in cui è attivo il percorso**
0 stalli + EX / EX	*4*
0 stalli + EX / EX	*5*
---	*na*
1 stallo + MEM / EX	*7*
---	*na*
---	*na*

esercizio n. 4 – domande su argomenti vari

logica combinatoria

Si consideri una rete combinatoria a tre ingressi **A**, **B** e **C**, e un'uscita **U**, la cui tabella della verità è riportata sotto:

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

Rispondere alle seguenti domande:

1. Si sintetizzi l'uscita **U** in **prima forma canonica** (somma di prodotti – SoP) e si riporti l'espressione booleana corrispondente:

   U_SoP = *!A !B !C + !A !B C + A !B C*

2. Si sintetizzi l'uscita **U** in **seconda forma canonica** (prodotto di somme – PoS) e si riporti l'espressione booleana corrispondente:

   U_PoS = *(A + !B + C) (A + !B + !C) (!A + B + C) (!A + !B + C) (!A + !B + !C)*

3. Si calcolino costi (C) e ritardi (R) delle reti realizzate tramite le due forme canoniche. Per il costo si consideri il **numero di letterali** (ossia quante variabili figurano nella forma, contando anche le ripetizioni), mentre per il ritardo si supponga di avere a disposizione solo porte a **tre ingressi**, con identico ritardo per porte AND e OR (si supponga che il ritardo sia unitario), e di trascurare i ritardi delle porte NOT.

   realizzazione U_SoP C_SoP = *9* R_SoP = *2*

   realizzazione U_PoS C_PoS = *15* R_PoS = *3*

bus del calcolatore

Si consideri un bus di calcolatore con queste ipotesi:

- Il bus è di tipo **sincrono** e svolge un'operazione (lettura o scrittura) entro **uno o più cicli di clock**, secondo i casi 1 e 2 sotto.
- Al bus è collegato un banco di memoria, che può essere letto o scritto tramite una porta di accesso.
- Il bus di controllo è dotato dei segnali **CS** (chip select) e **WE** (write enable), entrambi **attivi alt**i. Se il segnale **CS** è attivo, il banco di memoria risponde a comandi e indirizzi, e in tale caso il segnale **WE** definisce il verso dell'operazione (alto scrittura e basso lettura).
- Tutti i segnali del bus (linee di indirizzo, di dato e di controllo) hanno **ritardo di propagazione nullo**.

Il processore, al tempo **10 ns dopo l'inizio del ciclo di clock** (vedi diagramma sotto), inizia un'operazione di lettura di una parola di memoria. Si svolgano i punti seguenti:

1) Il banco di memoria rileva le variazioni di valore dei segnali di bus con un ritardo di **10 ns**, e fornisce il dato **10 ns** dopo avere rilevato comandi e indirizzo. Il processore richiede **10 ns** per acquisire il dato.

   Si completi il diagramma temporale sotto (si scriva 0 o 1 per i segnali singoli, I e D per indicare il valore dell'indirizzo di lettura e il dato letto, e X per indicare un valore non significativo), fino a quando il processore ha acquisito il dato e il bus è tornato a riposo (numero di colonne non significativo):

operazione (transazione) di lettura di un dato da memoria										
indirizzo	X	*I*	*I*	*I*	*X*					
CS	0	*1*	*1*	*1*	*0*					
WE	X	*0*	*0*	*0*	*X*					
dato	X	*X*	*X*	*D*	*X*					
tempo (ns)	0	10	20	30	40	50	60	70	80	90

   Quanto vale la **frequenza di clock massima** per completare l'operazione in **un solo ciclo di clock**?

   *Durata operazione = 40 ns, frequenza di clock massima = 1 / 40 ns = 25 MHz*_____

2) Ora si supponga che il banco di memoria non abbia un tempo costante per fornire il dato, ma generi e mandi al processore un segnale di **WAIT** (attivo alto), che tiene attivo per tutto l'intervallo di tempo necessario a fornire il dato. Per l'operazione descritta al punto 1, la memoria attiva il segnale di **WAIT** per una durata di **40 ns**. Si rifaccia il diagramma temporale, con **WAIT**:

operazione (transazione) di lettura di un dato da memoria										
indirizzo	X	*I*	*I*	*I*	*I*	*I*	*I*	*X*		
CS	0	*1*	*1*	*1*	*1*	*1*	*1*	*0*		
WE	X	*0*	*0*	*0*	*0*	*0*	*0*	*X*		
WAIT	0	*0*	*1*	*1*	*1*	*1*	*0*	*0*		
dato	X	*X*	*X*	*X*	*X*	*X*	*D*	*X*		
tempo (ns)	0	10	20	30	40	50	60	70	80	90

   Se la frequenza di clock è quella calcolata al punto 1, **quanti cicli di clock** occorrono ora per completare l'operazione ? E **quante parole** sono trasferibili in **un secondo** ?

   Durata operazione = 70 ns, periodo di clock 40 ns (da punto 1), numero di cicli di clock = 2 cicli __

   *Numero di parole = 1 s / (2 cicli × 40 ns) = 1 s / 80 ns ≈ 12,5 × 10^6 parole = 12,5 mega parole*__