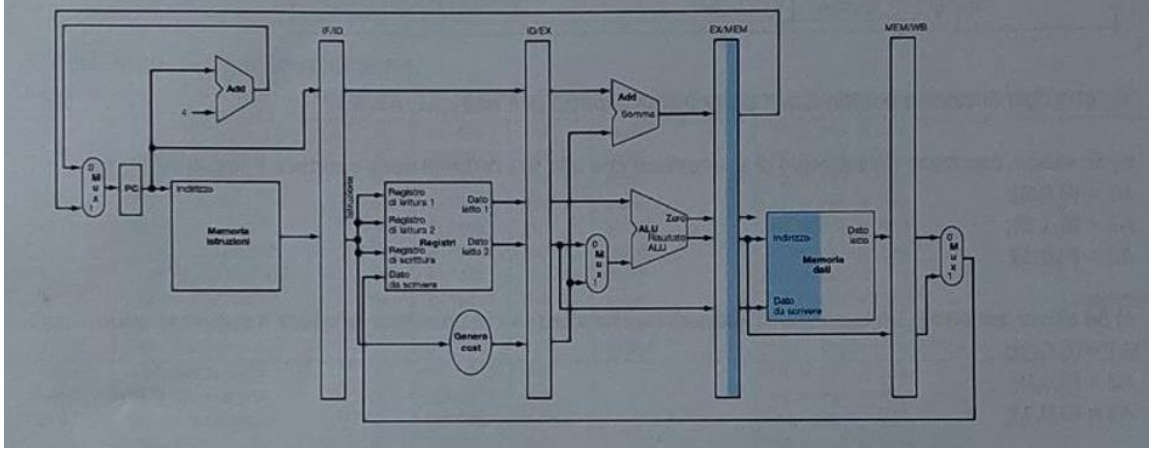
Architettura dei Calcolatori Test del 03/07/2024

1) La figura seguente rappresenta il quarto stadio della pipeline del processore Risc-V durante l'esecuzione di una istruzione specifica tra le 4 tipologie viste a lezione (tipo R, lw, sw, beq). Quale? Descrivi come viene eseguita in tutti e cinque gli stadi.

4 punti



In questa figura viene rappresentata l'istruzione sw (store word). Partendo da IF, l'instruction fetch, il PC punta all'istruzione dalla memoria delle istruzioni (e viene poi incrementato di 4 per l'istruzione successiva). In ID, instruction decode, si decodifica l'istruzione, quindi la base dell'indirizzo, il dato da salvare in memoria e l'offset, che viene estratto con il Generatore di Costante. Nell'EX, l'ALU calcola l'indirizzo dove deve essere salvato il dato, prendendo la base dell'indirizzo e sommandola con l'offset. MEM prende i dati e l'indirizzo ritornato dalla ALU dal registro EX/MEM (i registri sono tra ogni stadio e sono la memoria che fa in modo che un'istruzione possa avere i dati e risultati ricavati dallo stadio precedente) e scrive il dato nell'indirizzo. Il quinto stadio WB non fa nulla perché la funzione sw si ferma in memoria, facendo lo store del dato in memoria e non dovendo riportare nulla nei registri.

2) Traduci il seguente programma nell'assembler del Risc-V

```
int sum5(int a, int b, int c, int d, int e) {
   return a + b + c + d + e;
}
...
sum5(1,2,3,4,5);
```

4 punti

```
main:
        li a0, 1
        li a1, 2
        li a2, 3
        li a3, 4
        li a4, 5
        jal ra, sum5            # salta a sum5 e salva il ritorno in ra
```

sum5:

```
add t0, a0, a1   # a + b
add t1, a2, a3   # c + d
add t1, t1, a4   # c + d + e
add a0, t0, t1   # a0 = a+b + c+d+e
ret
```

3) Converti il numero 00000000 10000000 00000000 00000000 considerando la codifica unsigned, signed e la IEEE 754 . Indica ad esempio 2^63+2^38, e non il valore finale.

2 punti

- Unsigned: 2^{23}
- Signed: 2^{23} perché il bit più significativo è 0
- IEE 754: $(-1)^0 * 2^{(2^0-127)} * (1) = 2^{-126}$

4) Confronta, usando anche diagrammi ed esempi, le architetture di Moore e Mealy usate nei circuiti sequenziali,

4 punti

(idk)

5) Considerate il seguente programma multi-threaded scritto in pseudo-codice:

Programma P:
array A1 = {-1,-1,-1};
array A2 = {-1,-1,-1};
array A3 = {-1,-1,-1};

Thread T1:	**Thread T2:**	**Thread T3:**	**Thread T4:**

i=0;
while (i<3){
A1[i]=0;
i=i+1;
}
	j=0;
	while (j<3){
	A2[j]=0;
	j=j+1;
	}
		k=0;
		while (k<3){
		A3[k]=0;
		k=k+1;
		}
			A1[0]=1;
			if A1[0]=/=0 then A2[1]=1;
			if A2[1]==0 then A3[2]=1;

a) Che tipo di calcolo parallelizza il programma rispetto agli array A1, A2, A3?

b) Se esiste, mostrare un'esempio di esecuzione che alla sua terminazione produca il seguente stato:
A1 = {0,0,0};
A2 = {0,1,0};
A3 = {0,0,1};

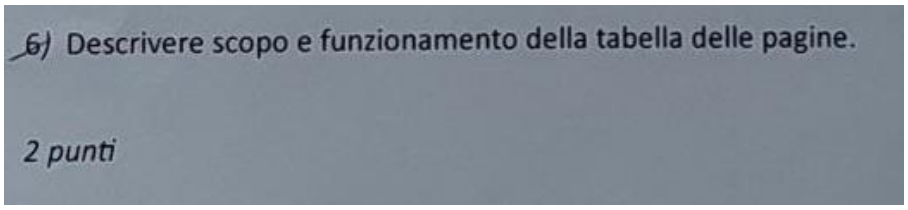
c) Se esiste, mostrare un'esempio di esecuzione che alla sua terminazione produca il seguente stato:
A1 = {0,0,0};
A2 = {0,0,0};
A3 = {0,0,1};

4 punti

a) Il programma parallelizza un calcolo elemento per elemento (?) su array distinti: ogni thread T1, T2, T3 azzerano il contenuto di A1, A2, A3 in parallelo. T4 invece esegue un controllo condizionale sui valori degli array modificati e può confliggere con con gli altri thread se eseguito in contemporanea.

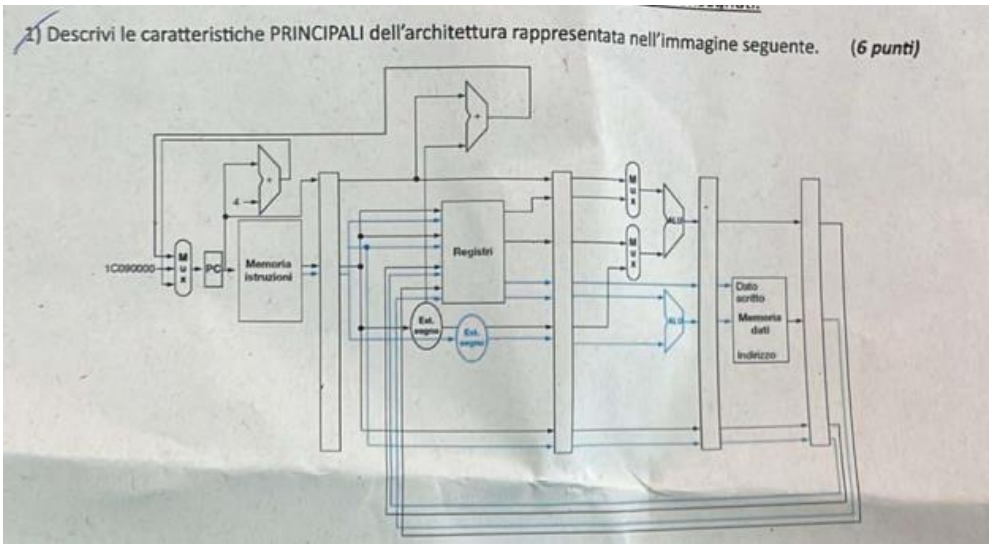
b) Non può esistere

c) Si esegue e T3 che azzera tutto A3, poi T4 che modifica cui A1 che diventa {1, -1, -1} e A2 che diventa {-1, 1, -1}, T2 azzera A2, e quindi T4 può continuare con la terza riga di codice. Infine, si esegue T1.



La tabella delle pagine è una struttura dati usata dal sistema operativo per gestire la memoria virtuale. Viene usata per tradurre indirizzi logici generati dai processi in indirizzi fisici effettivamente usati in RAM e per garantire l'isolamento tra processi. Ogni processo ha una sua tabella delle pagine e quando un processo accede ad un indirizzo virtuale, questo viene diviso in numero di pagina, che serve per cercare nella tabella quale frame fisico (blocco di RAM) contiene quella pagina virtuale, e offset, un puntatore all'interno della pagina per identificare la posizione esatta. Ogni voce nella tabella ha inoltre dei bit di controllo, usati dal sistema operativo, e sono i bit di validità (se la pagina è in memoria o no), il bit di protezione (quindi i permessi, solo lettura, scrittura, ecc), il bit di modifica (se è stato modificato di recente), il bit di accesso (se è stato usato di recente). Ha anche un caching chiamato TBL, che salva le pagine più utilizzate / utilizzate di recente per velocizzare il processo.

Architettura dei Calcolatori Test del 06/06/2024



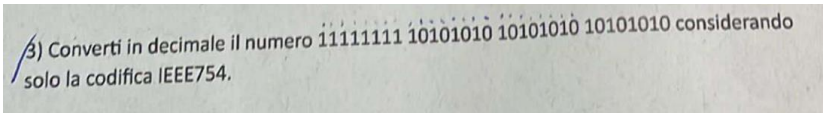
L'architettura rappresentata è un'architettura Pipeline a 5 stadi. Lo scopo della pipeline è quello di aumentare il numero di istruzioni eseguite in un certo tempo, eseguendole in parallelo. Gli stadi sono IF (instruction fetch), ID (instruction decode), EX (execute), MEM (memory access) e WB (write back). L'IF legge l'istruzione dalla memoria istruzioni usando il program counter (PC); l'ID decodifica l'istruzione; EX esegue le operazioni tramite ALU, e calcola l'indirizzo in casi di accesso di branch, facendo PC + costante (offset decodificato da ID); MEM accede alla memoria dati in caso di load o store; WB scrive il risultato nel registro di destinazione. Nella foto ci sono 4 registri IF/ID (64 bit), ID/EX (128 bit), EX/MEM (97 bit) e MEM/WB (64 bit). I registri servono per mantenere i dati tra uno stadio e l'altro, permettendo così di eseguire più istruzioni contemporaneamente, ognuna in uno stadio diverso della pipeline.
Le parti blu dello schema indicano i percorsi di forwarding (bypass), una tecnica usata per risolvere le dipendenze sui dati tra istruzioni consecutive, senza inserire stall. Questi percorsi prelevano i risultati non ancora scritti nei registri e li inoltrano direttamente alla ALU, rendendo la pipeline più efficiente.

2) Scrivi un programma nell'assembler del Risc-V che, dato in memoria un array A di SHORT calcoli la media dei valori pari e la memorizzi nella variabile acc. Il vettore A utilizza come terminatore il valore -1. Un esempio in C è il seguente **(6 punti)**

```
short A[13] = {1,2,3,4,5,6,7,8,9,10,11,12,-1};
float acc = 0;
int i,n = 0;
while(A[i] != -1) {
    if(A[i]%2 == 0) { acc+=A[i]; n++; }
    i++;
}
If(n!=0) acc /= n;                    //Il risultato sarebbe 7
```

```
.data
A:      .half 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, -1
acc:    .float 0.0
n:      .word 0
i:      .word 0
```

```
.text
.globl main
main:
        la s0, A            # s0 = indirizzo base di A
        li t0, 0            # t0 = i
        li t1, 0            # t1 = somma acc
        li t2, 0            # t2 = n
loop:
        add t3, s0, t0      # t3 = indirizzo A[i]
        lh t4, 0(t3)        # loadhalf, carica A[i] in t4

        li t5, -1
        beq t4, t5, fine    # se A[i] = -1 fine loop

        rem t6, t4, 2       # t6 = A[i] % 2
        bne t6, zero, skip  # se dispari skip

        add t1, t1, t4      # acc = acc + A[i]
        addi t2, t2, 1      # n = n + 1
skip:
        addi t0, t0, 2      # i = i + 1
        j loop
fine:
        beq t2, zero, end   # se n = 0, no divisione
        div t1, t1, t2      # acc = acc + n
end:
```



- IEEE 754: il segno è 1, quindi negativo, l'esponente è 11111111, ossia 255, e questo è un caso speciale perché, se mantissa ≠ 0, rappresenta NaN (not a number), mentre se mantissa = 0, rappresenta ±∞. In questo caso, la mantissa non è 0 e quindi non rappresenta un numero.

✅ **Tabella dei casi speciali IEEE 754 (32 bit)**

Segno (S)	Esponente (8 bit)	Mantissa (23 bit)	Valore rappresentato
0	`00000000`	`000...000` (tutti zeri)	**+0** (zero positivo)
1	`00000000`	`000...000`	**−0** (zero negativo)
*	`00000000`	≠ 0	**Numero denormalizzato**
*	`01111111` (127)	000...000	**+1.0** (normale)
*	altro (1–254)	qualsiasi	**Numero normale** (±1.xxx · 2^n)
0	`11111111`	`000...000`	**+∞** (positivo infinito)
1	`11111111`	`000...000`	**−∞** (negativo infinito)
*	`11111111`	≠ 0	**NaN (Not a Number)**

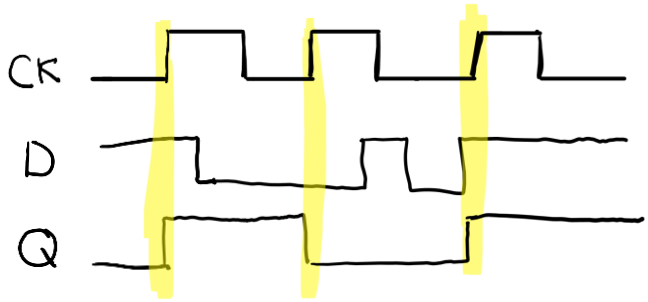
> * = qualsiasi valore (0 o 1)

4) Un processore che supporta AVX2 è un processore x86? E' teoricamente più performante di uno che supporta SSE4.2? Motiva in modo sintetico ma esaustivo. **(3 punti)**

Si, un processore che supporta AVX2 è un processore x86. È più performante di un SSE4.2 perché AVX2 introduce i registri a 256 bit, il doppio rispetto ai 128 bit di SSE4.2, facendo in modo di eseguire più operazioni SIMD in parallelo.

5) Disegna e descrivi il comportamento di un Flip Flop D edge triggered. **(5 punti)**

Un flip-flop D edge triggered è un circuito sequenziale che cattura il valore presente in ingresso D al fronte di salita e mantiene il valore catturato stabile in uscita Q fino al prossimo fronte di clock; quindi, ignorando le altre variazioni di D se non eseguite al fronte di clock.



6) Considerate il seguente programma multi-threaded scritto in pseudo-codice: **(7 punti)**

Programma P:
array A = {1,2,3}
array B = {0,0,0}
int i=0,k=0;

Thread T1:	**Thread T2:**	**Thread T3:**

while (i<2){
B[i]=A[i];
i=i+1;
}
	A[i]=5;
	k=k+1;
		B[k]=10;

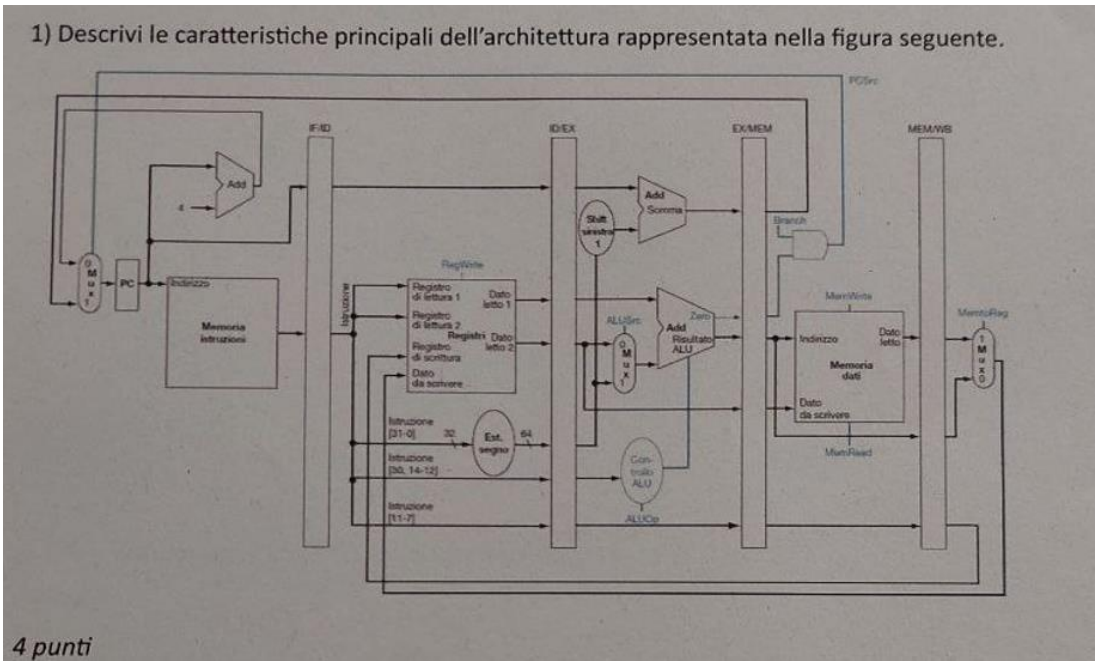
- L'esecuzione del programma termina sempre?
- L'esecuzione del programma può generare errori real time?
- Si possono verificare race condition su dati condivisi? Quali?
- Nei casi di terminazione, descrivere almeno quattro (4) casi relativi allo stato finale dell'array B utili a capire i problemi dovuti alla semantica per interleaving dell'esecuzione dei 3 thread.

- Si, l'esecuzione termina sempre ma con risultati diversi
- Si, perché i thread non sono sincronizzati e le variabili sono condivise tra più thread
- Si, su tutti i dati, i, k e gli array A e B
- Se li eseguo in ordine T1, T2, T3, array B sarà {1, 10, 5}
  Se eseguo T3, T2, T1, array B sarà {1, 2, 0}
  Se eseguo T1, T3, T2, array B sarà {10, 2, 5}
  Se eseguo T2, T1, T3, array B sarà {1, 10, 0}

*Confrontare le strategie **write through** e **write back** usate nella gestione della cache.* **(4 punti)**

La strategia write through, quando scrivo qualcosa di nuovo sulla cache, aggiorna subito anche la memoria principale. La write back, invece, scrive in cache, ma rimanda l'aggiornamento della memoria al momento in cui il blocco della cache deve essere sovrascritto.

Architettura dei Calcolatori Test del 06/06/2024



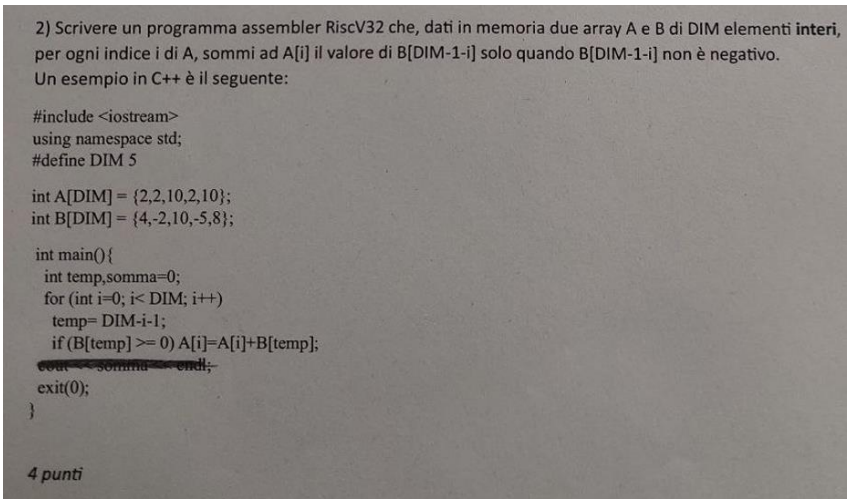
L'architettura rappresentata è un'architettura Pipeline a 5 stadi. Lo scopo della pipeline è quello di aumentare il troughput del processore, ossia il numero di istruzioni eseguite in un certo tempo, eseguendole in parallelo.

Gli stadi sono IF (instruction fetch), ID (instruction decode), EX (execute), MEM (memory access) e WB (write back). L'IF legge l'istruzione dalla memoria istruzioni usando il program counter (PC); l'ID decodifica l'istruzione; EX esegue le operazioni tramite ALU, e calcola l'indirizzo in casi di accesso di branch, facendo PC + costante (offset decodificato da ID); MEM accede alla memoria dati in caso di load o store; WB scrive il risultato nel registro di destinazione.

Nella foto ci sono 4 registri IF/ID (64 bit), ID/EX (128 bit), EX/MEM (97 bit) e MEM/WB (64 bit). I registri servono per mantenere i dati tra uno stadio e l'altro, permettendo così di eseguire più istruzioni contemporaneamente, ognuna in uno stadio diverso della pipeline.

Ci sono poi dei controlli (che di solito sono generati dalla control unit, che però non è nell'immagine), che nell'immagine sono ALUOp, ALUSrc, MemRead, MemWrite, Branch, … Controllo ALU, sotto all'ALU dell'EX, è il modulo che manda un segnale ad ALU per fargli capire quale operazione deve eseguire, basandosi sull'istruzione decodificata da ID.

Molto importante è il branch che si trova nel'MEM. In caso di branch, l'ALU usa il segnale Zero per indicare che gli indirizzi sono uguali, e il nuovo indirizzo che è stato calcolato in EX viene portato allo stadio iniziale (IF). Lì si trova un MUX (multiplexer) che aggiorna il PC normalmente a PC + 4 (quindi al registro successivo) se branch non viene eseguito, mentre se Zero = 1, il PC si aggiorna al nuovo indirizzo calcolato prima. L'informazione del registro di destinazione deve essere trasportata fino a WB, se no Write Back non saprà dove riscrivere il risultato. Rd viene quindi trasportata lungo tutta la pipeline nei registri interstadio (ID/EX, EX/MEM, MEM/WB), che avranno quindi 5 bit in più.



2) Scrivere un programma assembler RiscV32 che, dati in memoria due array A e B di DIM elementi **interi**, per ogni indice i di A, sommi ad A[i] il valore di B[DIM-1-i] solo quando B[DIM-1-i] non è negativo.
Un esempio in C++ è il seguente:

```
#include <iostream>
using namespace std;
#define DIM 5

int A[DIM] = {2,2,10,2,10};
int B[DIM] = {4,-2,10,-5,8};

 int main(){
   int temp,somma=0;
   for (int i=0; i< DIM; i++)
     temp= DIM-i-1;
     if (B[temp] >= 0) A[i]=A[i]+B[temp];
  cout << somma << endl;
  exit(0);
}
```

4 punti

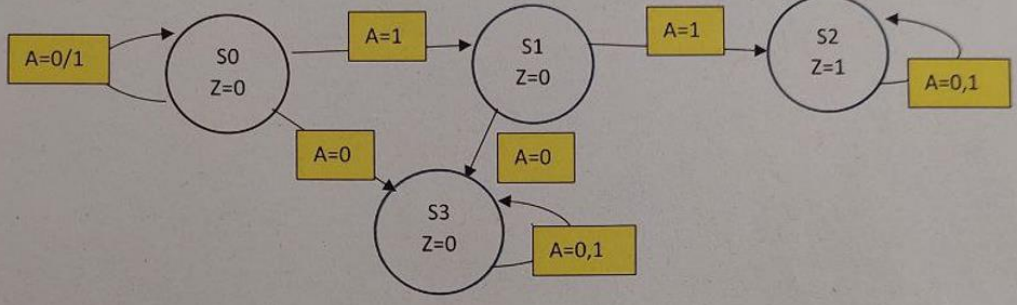
```
.data
DIM:    .word 5
A:      .word 2, 2, 10, 2, 10
B:      . word 4, -2, 10, -5, 8
.text
.globl main
main:
        li t0, 0            # i = 0
        la t1, A
        la t2, B
        lw t3, DIM      # t3 = DIM
loop:
        bge t0, t3, end         # se i >= DIM, si esce dal loop
        addi t4, t3, -1
        sub t4, t4, t0          # t4 = DIM – 1 – i

        slli t5, t0, 2          # t5 = i * 4
        add t6, t1, t5          # t6 = &A[i]

        slli t7, t4, 2          # t7 = i * 4
        add t8, t2, t7          # t8 = &B[temp]

        lw t9, 0(t8)            # t9 = B[temp]
        blt t9, zero, skip      # se B[temp] < 0, skip

        lw s0, 0(t6)            # s0 = A[i]
        add s0, s0, t9          # s0 = A[i] + B[temp]
        sw s0, 0(t6)            # A[i] = s0

skip:
        addi t0, t0, 1          # i++
        j loop
end:
```

3) Converti in decimale il numero 10100000 00000000 00000010 11010100 interpretato nei tre seguenti modi:
a) codifica binaria senza segno,
b) codifica binaria in complemento a due,
c) codifica binaria IEEE 754.

2 punti

- Codifica unsigned: $2^{31} + 2^{29} + 2^9 + 2^7 + 2^6 + 2^4 + 2^2$
- Codifica in complemento a due: il numero diventa 01011111 11111111 11111101 00101100
- Codifica IEEE 754: $(-1)^1 * 2^{2^6-127} * (1 + (2^{-14} + 2^{-16} + 2^{-17} + 2^{-19} + 2^{-21}))$

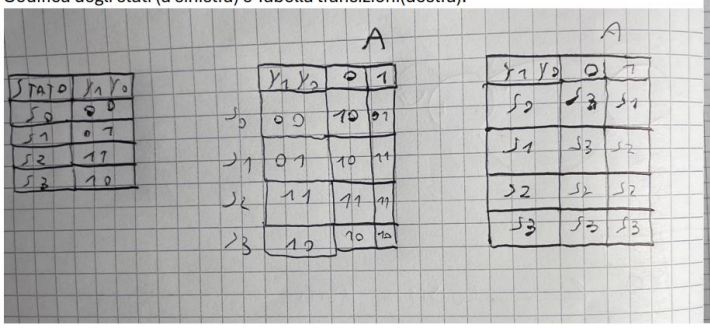
4)
Sintetizzare un circuito sequenziale alla Moore (con D-latch) con input A e uscita Z, specificato dal seguente diagramma a stati:

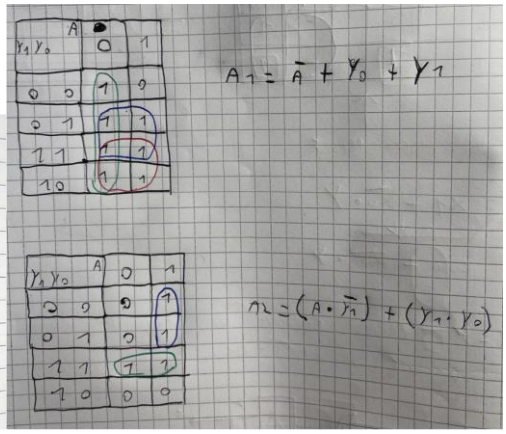


4 punti

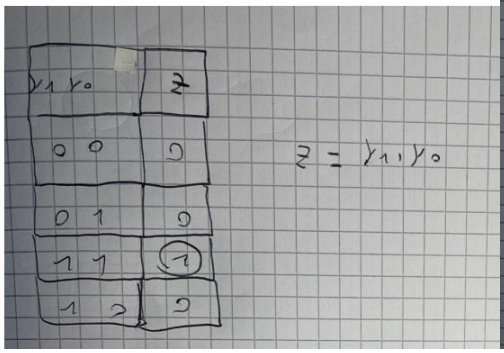
roba rubata al mio ragazzo:)

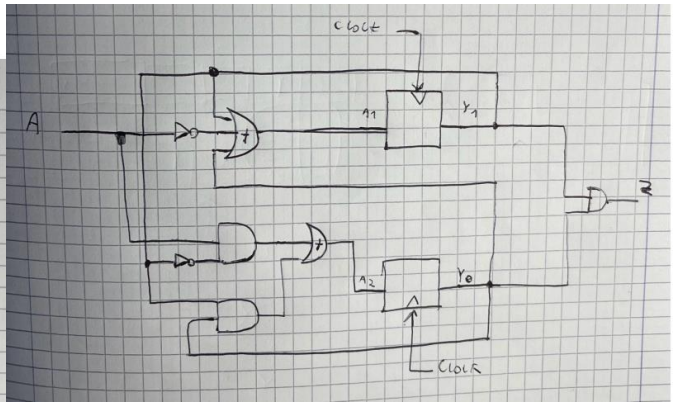
1) Codifica degli stati (a sinistra) e Tabella transizioni(destra):





3) Karnaugh uscita e circuito finale





5) Spiegare il comportamento del seguente programma multithreaded.

```
array A = {0,0,0}
int i=0
int k=0
```

Thread T1:	Thread T2:	Thread T3:

while (i<2){
A[i]=1;
i=i+1;
}
	k=k+1;
	A[i]=5;
		A[k]=7;
		A[i]=10;

4 punti

Non possiamo sapere per certo il tipo di output in quanto dipende dall'ordine con cui vengono eseguiti i thread. Si verifica il fenomeno del race condition in quanto diversi thread usano la stessa variabile e può comportare a risultati diversi visto che l'output dipende dall'ordine con cui vengono eseguite le istruzioni.

Ad esempio, se li eseguo nell'ordine T1, T2, T3, entro nel ciclo, A[0] = 1, poi incremento i e rientro nel ciclo, e anche A[1] diventa = 1, trovandoci quindi con A = {1, 1, 0}. Quando eseguiamo T2, incrementiamo k di 1, quindi = 1, e A[i] = A[2] = 5, quindi A = {1, 1, 5}. Arriviamo a T3 e lo eseguiamo. Il risultato finale sarà quindi A = {1, 7, 10}.

Se lo eseguo in un altro ordine, quindi ad esempio T2, T3, T1, inizio entrando in T2, incremento k e A[0] = 5, poi entro in T3, e quindi A = {7, 10, 0}. Entro poi in T1, nel ciclo, e il risultato finale è A={1, 1, 0}.

6) Spiegare il funzionamento della paginazione chiarendo bene il ruolo dell'hardware in una sua possibile realizzazione in un elaboratore.

2 punti

La paginazione è una tecnica di gestione della memoria che suddivide lo spazio degli indirizzi della memoria virtuale in blocchi di dimensione fissa chiamati pagine, mentre i blocchi della memoria fisica sono chiamati frame. Lo spazio di indirizzamento dei processi è suddiviso in pagine opportunatamente caricate quando necessario.
Ogni processo utilizza indirizzi virtuali, che devono essere tradotti in indirizzi fisici. La tabella delle pagine associa ogni pagina virtuale al corrispondente frame fisico.
L'hardware che esegue questi compiti è la MMU, Memory Management Unit. Esegue la traduzione automatica degli indirizzi virtuali in fisici, consultando la page table. Può anche usare una cache, la TLB (Translation Lookaside Buffer) per velocizzare l'accesso.