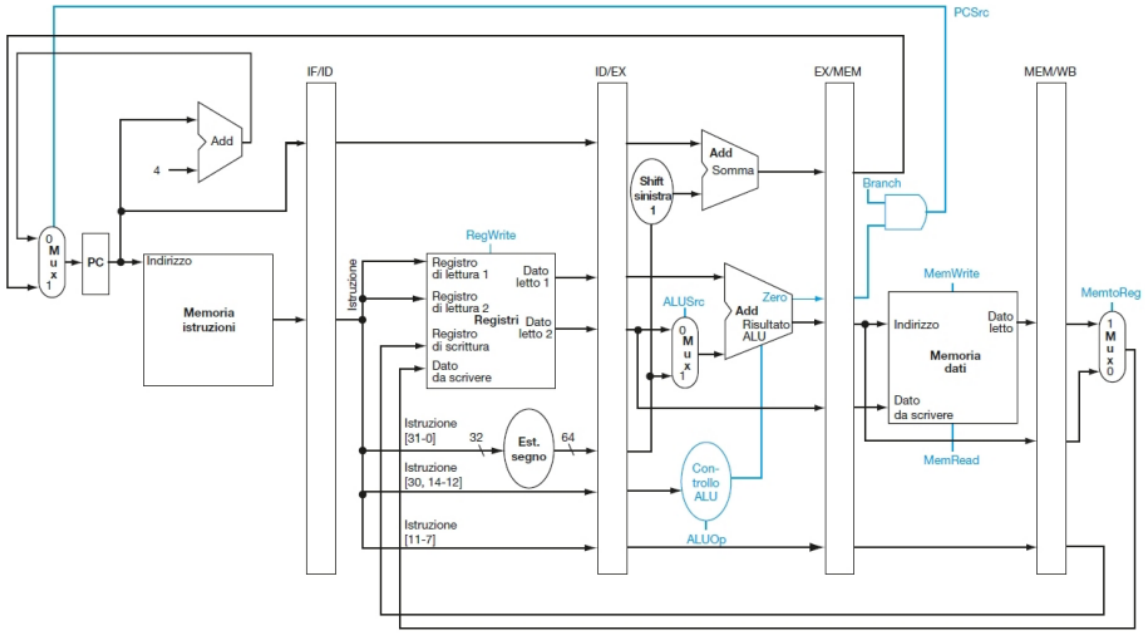
Esame scritto del 29/5/23 di Architettura dei Calcolatori

1) Descrivi le caratteristiche principali dell'architettura rappresentata nella figura seguente.

4 punti



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) Scrivi un programma nell'assembler del Risc-V che, dato un vettore A di interi ne calcoli la media e la memorizzi nella variabile B. Il vettoreA contiene in A[0] il numero degli elementi in cui è composto. Ad esempio A = [5, 20, 2, 3, 4, 7], la cui media sarà B = (20+2+3+4+7)/5

4 punti

```
.data
A:      .word 5, 20, 2, 3, 4, 7          # array A, A[0] = 5 numero di elementi
B:      . word 0
.text
.globl main
main:
        la t0, A        # load address t0 = indirizzo base dell'array A
        lw t1, 0(t0)    # load word t1 = A[0] = numero di elementi da sommare
        li t2, 0        # load immidiate t2 = somma = 0
        li t3, 1        # t3 = indice i = 1

loop:
        bgt t3, t1, fine        # se i > n, esco dal ciclo
        slli t4, t3, 2          # t4 = i * 4 (word i-esima)
        add t5, t0, t4          # t5 = indirizzo di A[ i ]
        lw t6, 0(t5)            # t6 = A[ i ]
        add t2, t2, t6          # somma = somma + A[ i ]
        addi t3, t3, 1          # i++
        j loop                  # jump loop

fine:
        div t7, t2, t1          # t7 = somma / n (media)
        la t8, B
        sw t7, 0(t8)            # salviamo la media in B
```

3) Converti in decimale il numero 10100000000000000000000000000000 . E' richiesto di indicare le operazioni, (e.g. 2^63+2^38) non il valore finale, di almeno due possibili interpretazioni binarie.

2 punti

- Come intero unsigned: $2^{31} + 2^{29}$
- Come intero signed in complemento a 2:
  il numero binario diventa 01100000000000000000000000000000, quindi $-2^{31} + 2^{29}$

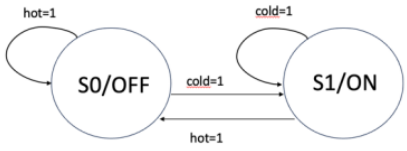
4) Spiega scopo e funzionamento di **multiplexer** e **demultiplexer**.

4 punti

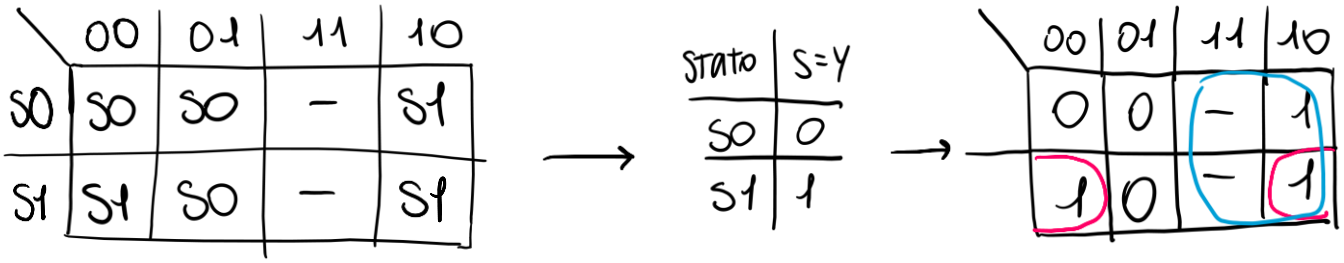
Un multiplexer è un circuito combinatorio che serve a selezionare uno tra diversi ingressi e a inoltrarlo in uscita. Seleziona dati presi da vari input e li trasmette in un'unica uscita. È un data selector perché seleziona un dato tramite select line/value e lo rende output, che è sempre 1. Ha N ingressi, un'uscita e $log_2(N)$ bit di selezione (select lines), che determinano quale ingresso viene collegato all'uscita. È come se fosse un interruttore controllato digitalmente.
Facendo un esempio, un MUX da 4:1 ha 4 ingressi, 2 bit di selezione e 1 uscita. I 2 bit di selezione creano il numero binario che corrisponde all'ingresso: con 00 verrà selezionato l'ingresso I0, con 01 l'ingresso I1, con 10 l'ingresso I2, e con 11 l'ingresso I3.

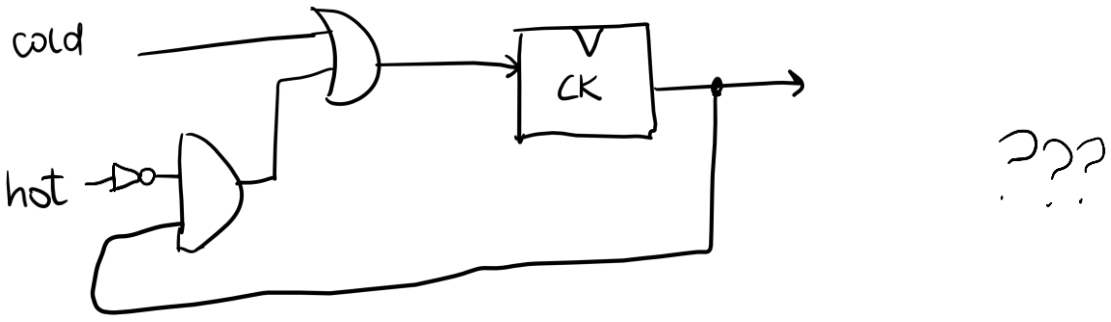
Un demultiplexer fa la funzione inversa: prende un unico ingresso e lo inoltra su una tra più uscite, in base ai bit di selezione. Si usa quando si vuole distribuire un segnale verso più destinazioni. Ha 1 ingresso, N uscite e $log_2(N)$ bit di selezione, che determinano quale delle uscite sarà attiva.
Facendo un esempio, un DEMUX da 1:4 ha 1 ingresso, 2 bit di selezione e 4 uscite. I 2 bit di selezione creano il numero binario che corrisponde all'output: con 00 verrà selezionato l'uscita I0, con 01 l'uscita I1, con 10 l'uscita I2, e con 11 l'uscita I3.

5) Sintetizza un circuito sequenziale alla Moore (con D-latch) per un controllore (con uscita ON/OFF) di un impianto di riscaldamento attivato da un termostato attraverso i segnali hot e cold e macchina a stati definita come segue:



4 punti





6) Spiega 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[k]=1;
i=i+1;
}
	A[i]=2;
	k=k+1;
		A[k]=3;

2 punti

Le variabili, A, i e k sono condivise tra i 3 thread, ma non c'è nessuna sincronizzazione, rendendo possibile l'accesso simultaneo e quindi race condition. L'ordine di esecuzione è non deterministico, può cambiare ogni volta che il programma viene eseguito, portando quindi a risultati diversi.

Un esempio: eseguo i thread in ordine, quindi T1, T2 e T3.
Entro in T1, i = 0 quindi posso entrare nel while. A[k] = 1, quindi A[0] = 1; incrementiamo poi i, che diventa 1. Rientriamo nel while, A[k] = 1, quindi A[0] = 1; incrementiamo i che diventa 2. Non possiamo quindi più entrare nel while.
Usciamo dal T1 e andiamo in T2. A[i] = 2, quindi A[2] = 2, poi incremento k di 1, quindi k diventa 1.
Passiamo al T3, dove A[k] = 3, quindi A[2] = 3.
I valori risultanti sarebbero quindi i = 2, k = 1, A[0] = 1, A[1] = 3, A[2] = 2.

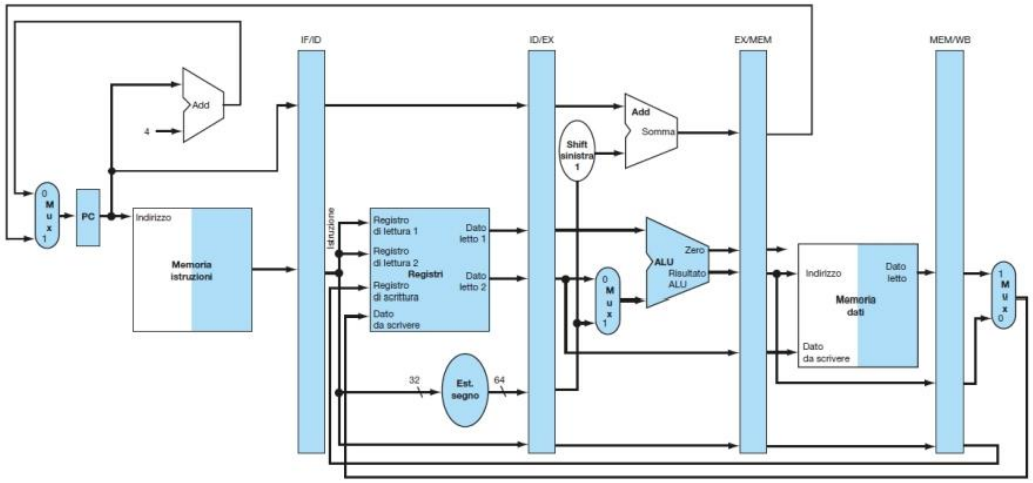
Se li eseguissi in un altro ordine, il risultato sarebbe diverso. Se faccio un altro esempio con T2, T1, T3.
Entro in T2, e A[i] = 2, quindi A[0] = 2; poi incremento k di 1.
Entro in T1, entro nel while e A[k] = 1, quindi A[1] = 1.
Finito il ciclo, entro in T3, e cambio A[k] = 3.
I valori risultanti sarebbero quindi i = 2, k = 1, A[0] = 2, A[1] = 3, A[2] = 0.

Quindi in base all'ordine dei thread, il risultato cambia.

Architettura dei Calcolatori Test del 8/06/2023

1) Quale operazione è raffigurata nella figura seguente? Descrivila.

4 punti



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.
L'istruzione rappresentata è una lw (load word).
Parte dal PC (program counter), che prende l'istruzione dalla memoria istruzioni. ID (decodifica) prende l'istruzione dal registro IF/ID e la smezza; inoltre, prende l'offset costante e lo porta nel multiplexer dello stadio successivo. Nello stadio successivo EX, l'ALU calcola un indirizzo sommando un valore del registro (rs1) con quello dell'offset (il MUX gli manda la costante offset al posto del secondo registro di solito usato per le operazioni o il beq), per produrre l'indirizzo usato dalla memoria dati per leggere la parola. Nello stadio MEM, prende il valore salvato nel registro EX/MEM e legge dalla memoria dati usando l'indirizzo calcolato da EX. Il valore viene salvato nel registro MEM/WB, Write Back lo prende e lo riporta nel registro di destinazione rd. Questo registro di destinazione viene trasportato fino a WB lungo tutta la pipeline nei registri interstadio (ID/EX, EX/MEM, MEM/WB), che avranno quindi 5 bit in più. Se così non fosse, WB non saprebbe il registro dove scrivere la parola.

2) Scrivi un programma nell'assembler del Risc-V che, dato un vettore A di interi ne calcoli la media e la memorizzi nella variabile B. Il vettore A utilizza come terminatore il valore -1.
Ad esempio se A = [20, 2, 3, 4, 7,-1], il risultato sarà B = (20+2+3+4+7)/5

4 punti

```
.data
A:      .word 20, 2, 3, 4, 7, -1        # array terminato da -1
B:      .word 0                         # variabile dove memorizzare la media

.text
.globl main
```

```
main:
        la t0, A        # t0 = indirizzo base dell'array A
        li t1, 0        # t1 = somma
        li t2, 0        # t2 = count

loop:
        lw t3, 0(t0)            # t3 = A[i]
        li t4, -1
        beq t3, t4, fine        # se A[1] == -1 esci dal ciclo

        add t1, t1, t3          # somma = somma + A[i]
        addi t2, t2, 1          # i = i + 1

        addi t0, t0, 4          # prossimo elemento
        j loop

fine:
        div t5, t1, t2          # media = somma / contatore
        la t6, B
        sw t5, 0(t6)            # memorizza media in B
```

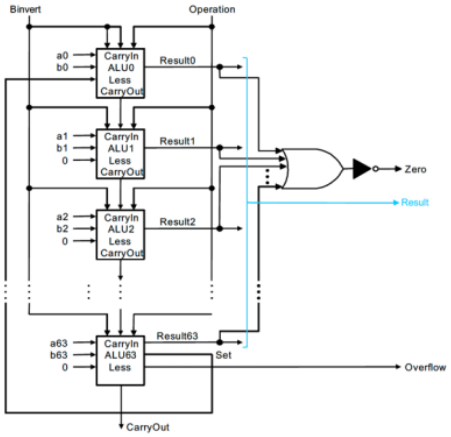
3) Che numero è il numero numero 11000000010000000000000000000000 ? Qualora fosse un numero binario e' richiesto di indicare le operazioni per la conversione in decimale considerando la codifica unsigned e la IEEE 754 . Quindi ad esempio 2^63+2^38 e non il valore finale.

2 punti

Non è un numero binario perché non ha 32 bit ma 33. È quindi un numero decimale.
Fosse a 32 bit (togliendo l'ultimo 0)

- Codifica unsigned: $2^{31} + 2^{30} + 2^{22}$
- Codifica IEEE 754: $(-1)^1 * 2^{(2^7-127)} * (1 + 2^{-1})$

4) Descrivere il funzionamento e il possibile uso del seguente circuito:



4 punti

L'immagine è una ALU da 64 bit, ed è adatta ad elaborare grandi quantità di dati, come numeri con virgola mobile, indirizzi di memoria più grandi ecc. Il compito generale di un'ALU è quello di eseguire operazioni, aritmetiche come somma e sottrazione, o logiche come and e or, o anche comparazioni per i salti (branch).
Una ALU da 1 bit è formata da due operandi (a, b), collegati in un AND, OR e in un ADDER. Hanno un multiplexer collegato alle 3 operazioni e in base a quale operazione viene scelta (tramite la linea Operation), il risultato sarà quello dell'operazione scelta. L'adder ha un CarryIn e un CarryOut (ossia i riporti dell'addizione), e gli operandi hanno anche una versione invert per fare le sottrazioni, i NOR e i NAND.
Una ALU da 64 bit è formata da 64 ALU da 1 bit, con il carryOut collegato come carryIn per l'ALU successiva; per il primo ALU, il carryIn è uguale alla linea Binvert.
Nel caso dell'immagine però, è un'uguaglianza. Gli ALU da un bit eseguiranno tutti una sottrazione tra gli operandi. Tutti i risultati saranno poi portati in un controllo che controlla se tutti i risultati della sottrazione sono 0. Se è così vuol dire che lo 'Zero' varrà 1, e vorrà dire che tutti e due gli operandi di ogni ALU sono uguali.
I risultati di tutti i 64 bit vengono raccolti insieme nel vettore Result[63:0] che rappresenta il risultato finale a 64 bit dell'operazione eseguita: questo vuol dire che per avere il risultato finale, si deve aspettare che tutti gli ALU da 0 a 63 abbiamo ricevuto il loro carry per eseguire la loro operazione.

5) Scrivere lo schema (non serve che "compili") di un programma multithreaded in C/C++ per calcolare il ***complemento ad 1*** *di un numero binario memorizzato in un array di 100 posizioni parallelizzando il calcolo attraverso 5 thread.*

4 punti

T1	T2	T3	T4	T5

6) Confrontare le strategie ***write through*** *e* ***write back*** *usate nella gestione della cache.*

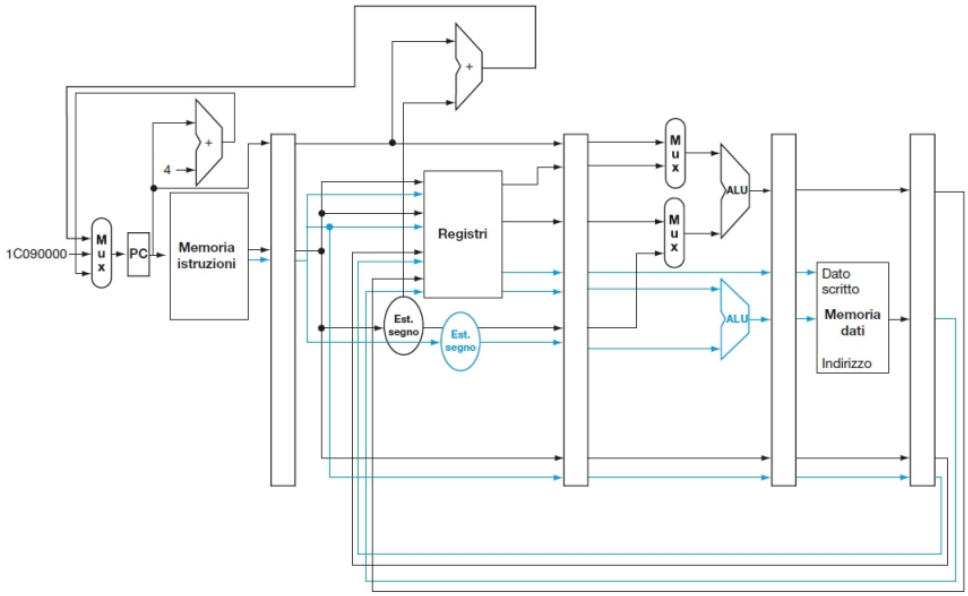
2 punti

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 17/07/2023

1) Quale tipo di parallelismo a livello di istruzioni è raffigurata nella figura seguente? Descrivilo brevemente.

4 punti



Il tipo di parallelismo rappresentato è il parallelismo a livello di istruzioni (Instruction Level Parallelism, ILP), che è una forma di esecuzione simultanea di più di un'istruzione per ciclo di clock. La pipeline è la prima forma di ILP, dove ogni stadio lavora in un'istruzione diversa, e in cui maggiore il parallelismo, minore è il tempo di esecuzione.
Le decisioni sono prese durante la compilazione, direttamente nell'hardware, dove il compilatore si occupa di fornire i pacchetti di istruzioni e prevenire o ridurre gli hazard, anche con la predizione. La predizione prevede la speculazione sulle istruzioni successive, anziché attendere il completamento delle istruzioni precedenti. Se la previsione è corretta, si ottiene un miglioramento delle prestazioni, ma se è errata, si devono scartare le istruzioni eseguite sotto predizione e rieseguirle, svuotando la pipeline e tornando indietro. Nella speculazione dinamica quindi i dati vengono memorizzati nel buffer e copiati nel register file solo se la speculazione è corretta.
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 vettore A di SHORT INT con segno calcoli la media dei valori maggiori o uguali a 10 e la memorizzi nella variabile B sotto forma di INT. Il vettore A utilizza come terminatore il valore -1. Ad esempio se short A = [2, 4, 9, 10, 10, 11, 12, -1], il risultato da memorizzarsi in B come int sarà (10+10+11+12)/4.

4 punti

```
.data
A:      .half 2, 4, 9, 10, 11, 12, -1       # vettore short
B:      .word 0

.text
.globl main
```

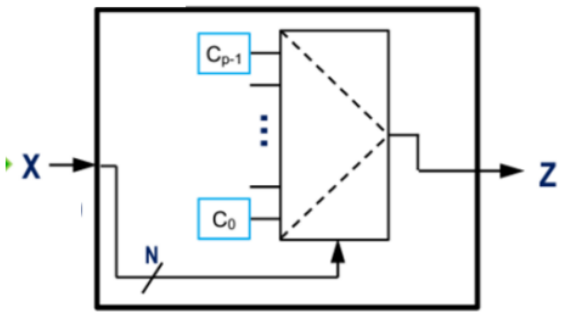
```
main:
        la t0, A
        li t1, 0          # somma
        li t2, 0          # count

loop:
        lh t3, 0(t0)      # carica A[i] come half signed
        li t4, -1
        beq t3, t4, fine  # se A[i] == -1, esci

        li t5, 10
        blt t3, t5, skip  # se A[i] < 10, skip

        add t1, t1, t3    # somma = somma + A[i]
        addi t2, t2, 1    # count = count + 1

skip:
        addi t0, t0, 2    # vai al prossimo elemento (half = 2 bytes)
        j loop

fine:
        div t6, t1, t2    # media = somma / contatore
        la t7, B
        sw t6, 0(t7)      # salva media in B
```

3) Che numero è il numero 00000000 11000000 00000000 00000000 ? Qualora fosse un numero binario e' richiesto di indicare le operazioni per la conversione in decimale considerando la codifica complemento a due e la IEEE 754 . Quindi ad esempio 2^63+2^38 e non il valore finale.

2 punti

- Complemento a 2: numero binario diventa 11111111 01000000 00000000 00000000
  E il risultato in decimale è $-2^{23} + 2^{22}$
- In IEEE 754 diventa $(-1)^0 * 2^{0-127} * (1 + 2^{-1})$

4) Spiegar quali componenti include e a cosa potrebbe servire il seguente circuito:



4 punti

Il circuito rappresenta una Look Up Table (LUT), che utilizza un multiplexer per implementare funzioni logiche predefinite, imitando una tabella di verità. Memorizza i valori di uscita di una funzione logica per ogni possibile combinazione di ingressi. Gli ingressi del MUX contengono i risultati predefiniti della funzione logica, e i bit del MUX rappresentano le variabili logiche. Quindi, in base al valore binario delle variabili logiche, viene selezionato uno degli ingressi, che rappresenta l'output corrispondente nella tabella di verità.

5) Scrivere di un programma con 5 procedure A, B, C, D, E tali che

- *A e B vengono eseguite se possibile in parallelo prima di C*
- *D ed E vengono eseguite se possibile in parallelo dopo C.*

4 punti

array Arr = {0, 0}
int i = 0;
int k = 0;

A	B	C	D	E

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

(non so se è giusto)

6) Spiegare brevemente funzionamento, posizionamento all'interno di un calcolatore e scopo della TLB.

2 punti

La TBL (Translation Lookaside Buffer) è una cache specializzata che memorizza le traduzioni tra indirizzi virtuali e indirizzi fisici. Quando la CPU accede ad un indirizzo virtuale, deve tradurlo in indirizzo fisico usando la Page Table (tabella delle pagine). La TBL evita di accedere ogni volta alla tabella delle pagine: se la TBL contiene già la traduzione, allora TBL hit, e la conversione viene data alla CPU; se non è presente, TBL miss, il sistema operativo o l'hardware carica l'informazione da Page Table. La TLB si trova tra la CPU e la MMU, ma è molte volte integrata al processore. Il suo scopo è appunto quello di velocizzare la traduzione degli indirizzi virtuali in fisici, migliorando le prestazioni dell'accesso alla memoria in presenza di paginazione.