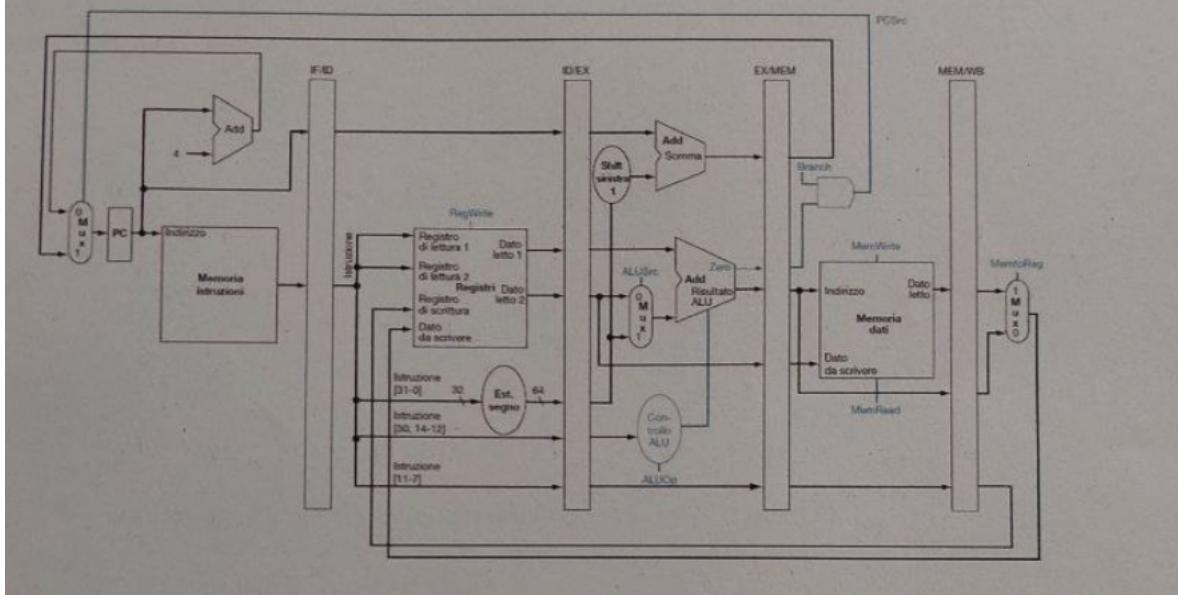


## Esercizio 1

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



Nella figura notiamo che viene utilizzata la tecnica della pipeline in quanto sono presenti i registri aggiuntivi per suddividere il circuito nei suoi diversi stadi.

La pipeline è una tecnica utilizzata per ottimizzare l'esecuzione dei programmi consentendo alla CPU di lavorare su più istruzioni contemporaneamente.

La pipeline cerca di sfruttare tutte le risorse disponibili, infatti quando una risorsa è libera viene assegnata alla prima istruzione che ne ha bisogno aumentando il throughput complessivo.

Ogni stadio lavora in contemporanea utilizzando risorse diverse.

L'incremento potenziale di velocità è determinato dal numero di stadi presente nella pipeline.

Tuttavia la velocità complessiva è influenzata dallo stadio più lento in quanto bisogna aspettare che tutte le istruzioni vengano eseguite.

Come ho detto prima sono presenti dei registri aggiuntivi, per preservare il valore delle istruzioni in esecuzione, il PC e altri valori vengono usati i seguenti registri:

- IF = instruction fetch, Prelevo la prossima istruzione da eseguire ed incremento il PC
- ID = instruction decode, decodifico l'istruzione e leggo gli eventuali operandi dal register file.
- EX = execute, eseguo i calcoli con la ALU
- MEM = Memory usata per istruzioni di tipo load/store, accedo alla memoria dati
- WB = WriteBack , scrivo "indietro" il risultato nel Register File.

Nella figura non è rappresentata la Hazard Detection Unit, questa unità mi permette di rilevare eventuali errori (criticità) e andare a risolverli. Le criticità che possono verificarsi sono di tre tipi:

- 1) Strutturali
- 2) Criticità sui dati
- 3) Criticità di controllo

Nella figura sono presenti i segnali di controllo provenienti dall'unità di controllo anche se essa non è disegnata, questi segnali sono molto importanti per l'esecuzione delle diverse istruzioni in quanto se sono attivi (bit = 1) svolgono determinate istruzioni. Per esempio, simuliamo l'esecuzione di un istruzione di tipo R:

Iniziatutto si parte dallo stadio IF/ID, in questo stadio si preleva l'istruzione da eseguire ed incremento il PC, si puo notare che è presente un MUX che è governato da una porta logica che esegue l'AND tra il segnale zero in uscita della ALU e un segnale di controllo branch che indica che il tipo di istruzione è un salto.

Questo MUX controlla solo quello che viene scritto nel PC (PC + 4 o indirizzo di destinazione della Branch).

Nello stadio ID abbiamo il register file che contiene tutti i registri e fornisce in uscita (in ogni istante) il contenuto dei registri in corrispondenza agli ingressi "registro di lettura".

Viene governato dal RegWrite che in questo caso vale 1 essendo di tipo R e quindi posso eseguire la scrittura.

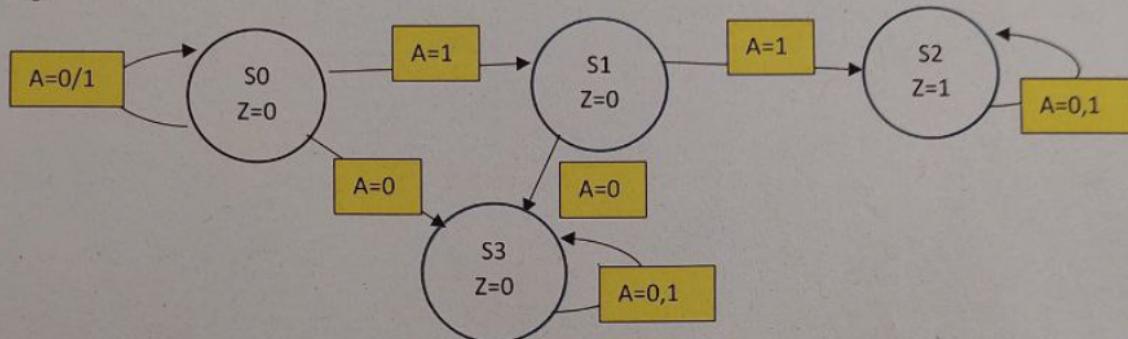
Alusrc vale 0 in quanto il secondo operando deve provenire dal register file e aulOP1 e aulOP2 valgono 1 0 per andare indicare che i segnali di controllo della ALU devono essere generati a partire dai cambi FUNZ.

MemRead vale 0 e MemWrite vale 1 in quanto devo andare a scrivere in memoria dati e infine memtoReG vale 1 per andare a scrivere nel register file.

## Esercizio 2

4)

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



4 punti

Soluzione:

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

STATO	$y_1 y_0$	A		$y_1 y_0$	$\delta$	$\tau$
$s_0$	00	$s_0$	00	10	01	
$s_1$	01	$s_1$	01	10	11	
$s_2$	11	$s_2$	11	11	m	
$s_3$	10	$s_3$	10	10	10	

2) Mappe di Karnaugh per A1 e A2

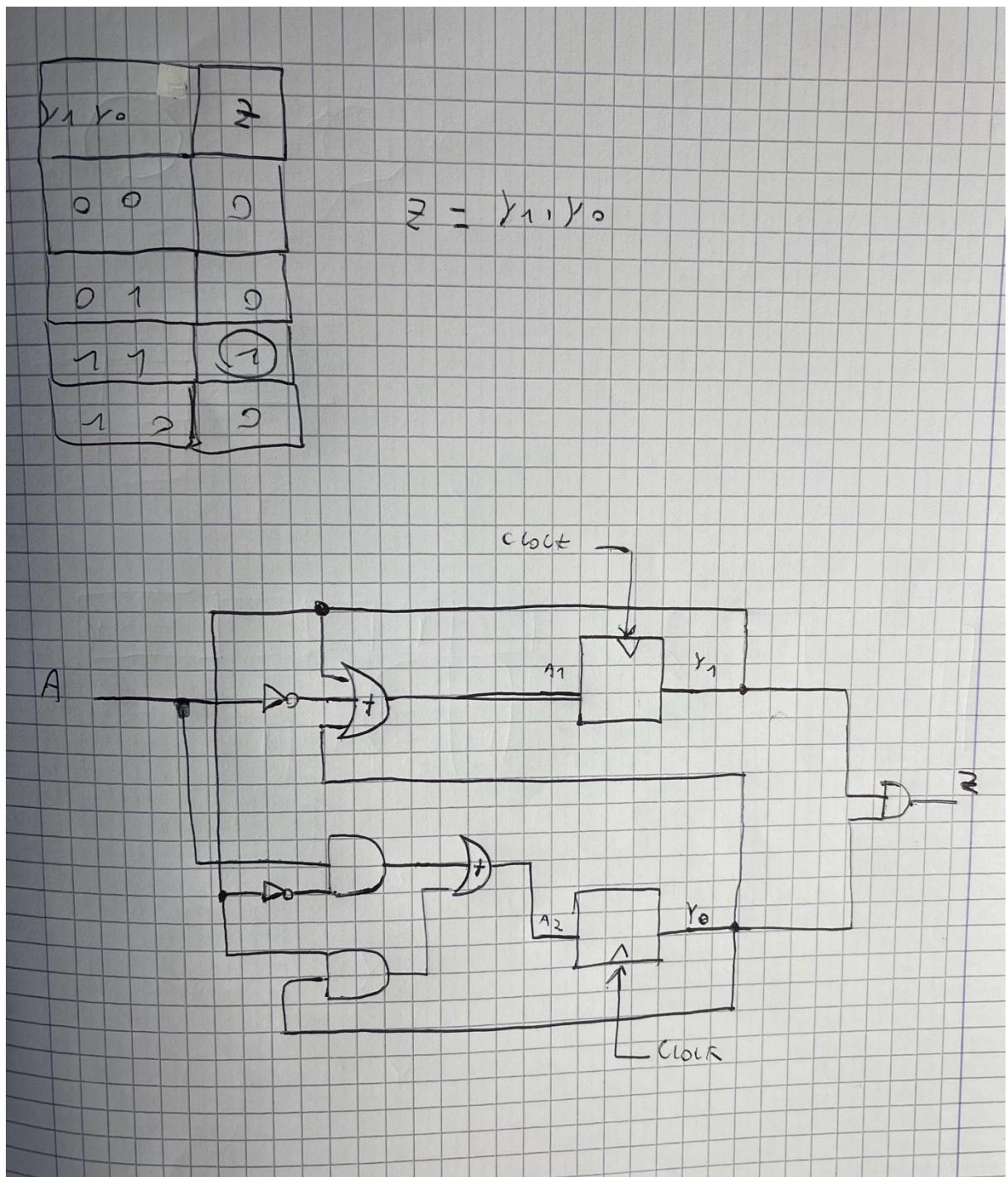
$y_1 y_0$	A	0	1
0 0	0	1	0
0 1	1	1	1
1 1	1	1	1
1 0	1	0	1

$$A_1 = \bar{A} + Y_0 + Y_1$$

$y_1 y_0$	A	0	1
0 0	0	0	1
0 1	0	1	1
1 1	1	1	1
1 0	0	0	0

$$A_2 = (A \cdot \bar{Y}_1) + (Y_1 \cdot Y_0)$$

3) Karnaugh uscita e circuito finale



Esercizio 5

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;

Non possiamo sapere per certo il tipo di output in quanto dipende dall'ordine con cui vengono eseguiti i thread.

Possiamo però simulare l'esecuzione (per esempio) dei Thread in sequenza per vedere come si comportano.

Se eseguiamo T1 otterremo che  $A[0] = 1$  durante il primo ciclo, poi  $A[1] = 1$  e usciamo da T1.

In T2  $k = 1$  e  $A[2] = 5$ .

Infine in T3 otteniamo che  $A[1]$  vale 7 e  $A[2]$  vale 10.

Alla fine di tutto otteniamo i seguenti valori:

array A = { 1, 7, 10}

I = 2

K = 1

Quindi osserviamo che si verifica il fenomeno del race condition in quanto diversi thread usano la stessa variabile e può comportare risultati diversi visto che l'output dipende dall'ordine con cui vengono eseguite le istruzioni.

## Esercizio 6

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

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.

I passi per la paginazione:

## 1) Suddivisione degli indirizzi

Gli indirizzi virtuali e fisici vengono divisi in due parti: indirizzo di pagina e uno di offset

## 2) Tabella delle pagine

Il sistema operativo mantiene una tabella delle pagine per ogni processo. La tabella mappa gli indirizzi di pagina virtuale agli indirizzi di pagina fisica. Ogni tabella possiede un bit di validità per verificare se la pagina si trova in memoria fisica.

## 3) Mappatura degli indirizzi

Il sistema operativo accede alla tabella delle pagine per associare l'indirizzo fisico corrispondente solo se il bit di validità è appunto valido , in caso valesse 0 allora la pagina viene caricata in memoria secondaria .

## 4) Uso della TLB

La TLB è una cache dedicata per la tabella delle pagine che mi permette di ridurre il tempo di traduzione degli indirizzi.

La RAM puo contenere diversi programmi in esecuzione e lo spazio di memoria è suddiviso tra essi, lo spazio di indirizzamento è virtuale ed è più grande di quello allocato dalla RAM.

Quindi la RAM funge da cache per i programmi in esecuzione (processi): lo spazio di indirizzamento dei processi è suddiviso in pagine opportunamente caricate quando necessario