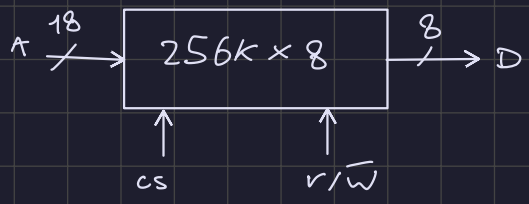
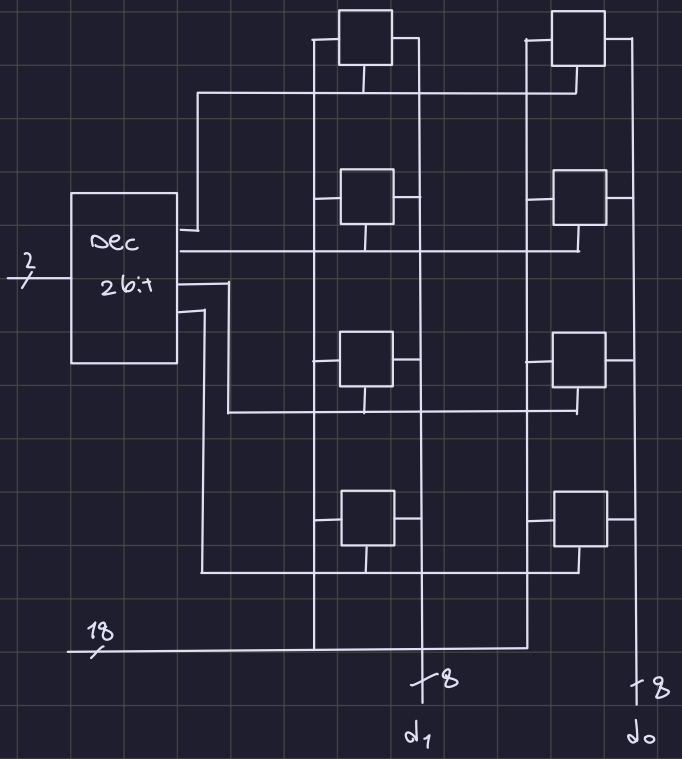
Esame 06/2021

1) Si deve realizzare un banco di memoria con indirizzi a 20 bit e parole di 16 bit. Si dispone di componenti base di memoria discreti ciascuno in grado di immagazzinare 256K righe di 8 bit. Disegnare la struttura di tale modulo, identificando chiaramente la corrispondenza tra i bit di indirizzo/dato esterni al banco ed i segnali interni.

indirizzo 20 bit
parola 16 bit

dim RAM = 2^{20} = 1M indirizzi





I bit di indirizzo corrispondono a quanti indirizzi il banco di memoria riesce a memorizzare, questo banco di memoria poi verrà indirizzato a 16 bit che sarebbero i bit della parola. Siccome il componente è da 256K righe di 8 bit si avrà un banco di memoria da 2 colonne con output 8 bit (16 bit totali di indirizzamento) e da 4 righe che sono il numero di componenti necessari per indirizzare 1M indirizzi (2^20).

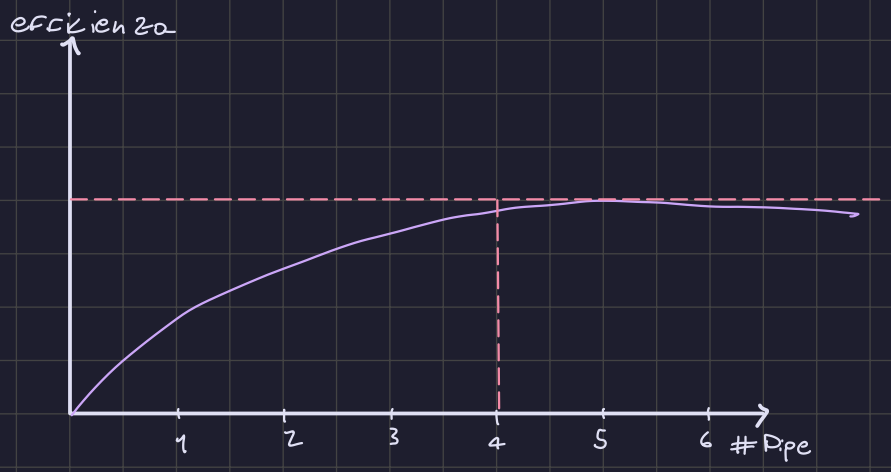
- Descrivere sommariamente le 3 condizioni necessarie affinchè la pipeline vista a lezione sia caratterizzata da CPI pari a 1 in un determinato (sotto)programma. Assumere che siano disponibili tutte le ottimizzazioni spiegate e che vi sia cache distinta per istruzioni e dati.

- Branch prediction
- Parallelismo
- Pre fetch?

Le 3 condizioni necessarie affinchè la pipeline abbia CPI pari a 1 sono:

- Branch Prediction, cioè l'esecuzione del ciclo fetch-decode-execute in entrambi i rami di un salto condizionato in modo da avere già pronta la CPU sia nel caso in cui il salto viene effettuato e sia nel caso in cui non viene effettuato. Nel ramo in cui non si entra le operazioni di fetch-decode-execute vengono cancellate. Un altro modo per fare branch prediction è quello di utilizzare l'ultimo risultato come indicatore per l'entrata in un branch, cioè se in un ciclo una condizione è vera si presuppone che sia vera anche nel successivo.

- Parallelismo, si possono avere più pipeline che lavorano parallelamente, diminuendo così il CPI, però tutto questo ha un costo, cioè più pipeline si hanno più è probabile che la dipendenza tra le istruzioni sia in pipeline diverse e questo è complicato da gestire. Infatti non si può aggiungere un gran numero di pipeline perchè dopo averne messe 3 o 4 non si riscontra più alcun miglioramento di prestazioni. Questo si può notare dal grafico dell'efficienza/numero di pipeline che segue un andamento logaritmico:



Si può notare che dopo un certo numero di pipeline non si ottiene più alcun miglioramento di prestazione.

- Pre fetch, consiste nel "esaminare" tutte le istruzioni prima di eseguire il ciclo fetch-decode-exec in modo da riordinarle dove possibile per evitare le dipendenze tra istruzioni. Questo componente permette di ridurre l'effetto bolla. Il pre fetch permette anche di migliorare il parallelismo assegnando le istruzioni non dipendenti a pipline diverse.

Queste 3 caratteristiche hanno permesso di portare il CPI sotto l'1 tenendo in considerazione tutte le altre ottimizzazioni dal punto di vista della memoria, della cache e dell'ISA.

2) Identificare le parti logiche dell'indirizzo RAM di una memoria da 256 MB, indirizzabile per byte, nel caso sia collegata ad una memoria cache associativa a gruppi, con 4 posizioni per gruppo, da 32KB in cui ogni posizione immagazzina un blocco di memoria di 16 byte.

- Descrivere i passaggi coinvolti nella conversione di un indirizzo logico in uno fisico, assumendo la presenza di TLB. Spiegare le conseguenze di un "hit" o "miss" in ogni consultazione di tabella.

Non fattibile

3) Elencare e **commentare** le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **un** BUS, che l'istruzione sia composta da **una sola parola**, che $xx(%Exx) rappresenti un metodo di indirizzamento **indiretto a registro con spiazzamento** e che l'indirizzo del **salto** sia **relativo** (usare solamente le righe necessarie e commentare ogni istruzione):

CALL $4(%EAX) commento

1. .....
2. .....
3. ...

- Disegnare il diagramma temporale dell'evoluzione dei segnali del BUS necessari a realizzare l'arbitraggio tra una CPU e due dispositivi DMA chiamati DMA1 e DMA2. Assumere che la CPU abbia priorità maggiore di DMA1, il quale ha priorità maggiore di DMA2. Come stato iniziale assumere che il BUS sia occupato da DMA1 e che sia CPU che DMA2 chiedano l'utilizzo.

F 1. PC_{out}, MAR_{in}, Select4, Add, Read, Z_{in}

Viene letto l'indirizzo nel program counter con una READ e viene sommata la costante 4 al PC per andare all'istruzione successiva, il risultato viene messo nel registro Z.

2. Z_{out}, PC_{in}, WMFC

L'istruzione succesiva viene messa nel program counter dal registro Z e successivamente si aspetta che venga completata la READ.

3. MDR_{out}, IR_{in}

L'istruzione letta all'indirizzo del program counter viene messa nell'istruction register.

DE 4. ESP_{out}, Select4, Sub, Z_{in}

Sottraggo 4 dallo stack pointer creando spazio per inserire il program counter.

5. Z_{out}, MAR_{in}, ESP_{in}

Il risultato della sottrazione lo metto nel mar e in esp

6. PC_{out}, MDR_{in}, WRITE

Inserisco il PC nello stack.

7. EAX_{out}, Select4, Add, Z_{in}, WMFC

Sommo 4 a eax (spiazzamento)

8. Z_{out}, MAR_{in}, READ

Il risultato dello spiazzamento viene messo nel mar per la lettura del dato.

9. PC_{out}, V_{in}, WMFC

Salvo il PC nel registro V per poterlo sommare successivamente e aspetto che la read finisca

10. MDR_{out}, $Select_V$, ADD, Z_{in}

Aggiungo al PC (salto relativo) il contenuto di MDR.

11. Z_{out}, PC_{in}, END

Il risultato della somma viene messo nel PC.