

# **Memoria**

2) Si consideri una CPU dotata di memoria cache 2-associativa di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 8M parole.

- Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

- Si assuma che la cache appena descritta sia utilizzata per i dati, che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente). La CPU esegue un programma che accede in sequenza a tutti gli elementi di un array di 8320 parole (ogni elemento ha le dimensioni di una parola) che è memorizzato a partire dall'indirizzo 0. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 5 volte.

Si assume che il tempo di accesso alla cache sia di 1T e che il tempo di accesso alla memoria sia di 10T (entrambi i tempi si riferiscono alla lettura di una parola). Calcolare il rapporto (fattore di miglioramento) tra il sistema in presenza di cache e in assenza di cache per l'esecuzione di questo programma.

2) Si consideri una CPU dotata di memoria cache 4-associativa di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 4M parole.

- Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

- Si assuma che la cache appena descritta sia utilizzata per i dati, che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente). La CPU esegue un programma che accede in sequenza a tutti gli elementi di un array di 8320 parole (ogni elemento ha le dimensioni di una parola) che è memorizzato a partire dall'indirizzo 0. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 5 volte.

Si assume che il tempo di accesso alla cache sia di 1T e che il tempo di accesso alla memoria sia di 10T (entrambi i tempi si riferiscono alla lettura di una parola). Calcolare il rapporto (fattore di miglioramento) tra il sistema in presenza di cache e in assenza di cache per l'esecuzione di questo programma.

2) Si consideri una memoria cache 4-set associativa della dimensione di 32 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte.

- Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

- Si assuma che la cache appena descritta sia utilizzata per memorizzare i dati di un programma. Si assuma che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente).

Il programma di benchmark accede in sequenza a tutti gli elementi di un array di 2000 numeri interi memorizzato a partire dall'indirizzo 0. Si ricorda che ogni intero ha una dimensione di 4 byte. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 10 volte. Si assume che il tempo di accesso ad un byte della cache sia 1T ed in memoria sia 10T. Calcolare i tempi di accesso ai dati in presenza e assenza della cache.

3) Si consideri una memoria cache 4-set associativa della dimensione di 16 Kbyte con 512 byte per blocco. La cache è collegata ad una memoria di 2Mbyte indirizzabile per byte.

- Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

- Si assuma che la cache appena descritta sia utilizzata per memorizzare i dati di un programma. Si assume che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente).

Il programma di benchmark accede in sequenza a tutti gli elementi di un array di 2000 numeri interi memorizzato a partire dall'indirizzo 0. Si ricorda che ogni intero ha una dimensione di quattro byte. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 5 volte. Si assume che il tempo di accesso ad un byte della cache sia 1T ed in memoria sia 10T. Calcolare il tempo di accesso ai dati in presenza della cache.

4) Si consideri una memoria cache completamente associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 16Mbyte indirizzabile per byte.

- Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

- Si assuma che la cache appena descritta sia utilizzata per memorizzare i dati di un programma. Si assuma che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo MRU (sostituzione dell'ultimo elemento utilizzato).

Il programma di benchmark accede in sequenza a tutti gli elementi di un array di 70 record ognuno delle dimensione di 1KByte. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 10 volte. Si assume che il tempo di accesso ad un byte della cache sia 1T ed in memoria sia 10T. Calcolare il tempo di accesso ai dati in presenza e assenza della cache.

- 3) Si consideri il problema della codifica binaria delle informazioni.
- Quali sono i passi che permettono di .
- 
- Si assuma che l'architettura precedente sia dotata di 64Mbyte di memoria fisica e che le pagine abbiano dimensione di 4Kbyte.  
Se un programma di benchmark accede in sequenza a tutti gli elementi di un array di 2000 numeri interi, quale deve essere la dimensione del *working set* perché si abbiano dei *page fault* solamente nella fase di caricamento del programma? Quanti *page fault* si hanno durante l'esecuzione del programma ipotizzando che il *working set* abbia una pagina meno dell'ottimo trovato?

3) Si descriva il concetto di memoria virtuale e la funzionalità di una MMU.

- Si assuma che un computer con memoria virtuale, strutturata a pagine di 4Kbyte, sia dotato di 64Mbyte di memoria fisica.

Si consideri un programma con un codice di 7.2Kbyte che accede ciclicamente in sequenza a tutti gli elementi di un array di 1000 record in cui ogni campo è composto da 2 numeri interi. Quale deve essere la dimensione del *working set* perché si abbiano dei *page fault* solamente nella fase di caricamento del programma con esecuzione del primo ciclo di accesso agli elementi dell'array? Quanti *page fault* si avrebbero durante l'esecuzione del programma ipotizzando che il *working set* abbia una pagina meno di quanto definito nel punto precedente e che il ciclo di accesso venga ripetuto 10 volte?

- 3) Si consideri una memoria cache 4-set associativa della dimensione di 32 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 1Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

- Quali sono i vantaggi e gli svantaggi delle memoria completamente associative rispetto alle memoria non associative.
- Quali sono le motivazioni che fanno preferire la realizzazione di unità di controllo cablate rispetto a quelle micropogrammate.
- Definire lo schema di un controllore cablato indicando i segnali utilizzati e la funzione dei blocchi presenti.

- 3) Si consideri una memoria statica di 4Mbyte indirizzabili un byte alla volta.
- Definire le dimensioni dei segnali di dato e indirizzamento e disegnare lo schema del chip di memoria avendo a disposizione moduli da 1Mbit.
- 
- Come si potrebbe utilizzare il concetto di *memoria interlacciata* per migliorare le prestazioni del chip precedente?
- 
- Come si complicherebbe lo schema del chip nel caso di *memoria dinamica*?
- 
- Su quali concetti si basa lo schema della *gerarchia di memoria*.

2) Si consideri una CPU dotata di memoria cache di 128K byte con 64 byte per blocco. Questa CPU è collegata ad una memoria RAM da 16M byte indirizzabile per byte.

- Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO (o SET) ed ETICHETTA in cui questo indirizzo può essere suddiviso. Definire queste dimensioni per una memoria cache di tipo:

- ad accesso diretto

- 2-set associativa

- completamente associativa

- Descrivere il principio su cui si basa il funzionamento della memoria cache.

- Disegnare lo schema della memoria RAM precedente utilizzando chip 256Kx1

Esame architettura del 18-09-2008

- 1) Si progetti un dispositivo che faccia il prodotto
  1. La sequenza di dati termina con la coppia di bit 11
  2. Start[1] a 1 quando il circuito inizia il calcolo
  3. Dato[2] fornisce i valori
  4. Ok[1] a 1 per 1024 cicli di clock dopo l'esecuzione del prodotto
  5. Mull[8] è inizializzato a 0.....0, al termine del prodotto racchiude la soluzione e la mantiene fino a che Ok[1] è ad 1
- 2) Dare la definizione di somma di prodotti e spiegare come passare a prodotti di somma
- 3) Perchè l'architettura a pagine è migliore?
- 4) Disegnare una scheda di memoria da 32Mb con indirizzi a parole da 32bit utilizzando chip da 2Mb con indirizzamento a byte
- 5) Spiegare l'impatto della parziale specifica su Quine & McCluscky

- 3) Si consideri un sistema a memoria virtuale con spazio logico di 4G parole, una memoria fisica di 32M parole e dimensione delle pagine di 16K parole.

- Determinare il numero di bit che definiscono:

<b>Lunghezza indirizzo logico:</b>	
<b>Di cui per Num. Pag. Logica</b>	
<b>per lo spiazzamento</b>	
<b>Lunghezza indirizzo fisico</b>	
<b>di cui per Num. Pag. Fisica</b>	
<b>per lo spiazzamento</b>	

- Nella seguente tabella sono riportati alcuni valori del parametro R (numero di pagine residenti per processo); sapendo che il Sistema Operativo occupa permanentemente 448 pagine e che sono stati creati 22 processi, indicare per ogni valore di R il numero di processi in stato di "fuori memoria" ossia che non possono avere tutte le pagine in memoria.

R	40	80	160	320	800
Numero di processi fuori memoria					

- Specificare come è stato calcolato il valore per R = 160:

- Si supponga di avere un sistema basato su memoria virtuale configurato con R=10 e di sapere che il valore massimo di k (numero di accessi) per il quale il Working Set W(k)=10 è 1000, cioè  $W(k)>10$  per  $k>1000$ . In base a queste ipotesi è possibile valutare i valori massimo e minimo del numero di page fault, delle percentuali di page fault e delle percentuali di successo (Hit Rate). Definire tali valori con una breve spiegazione.

<b>Numero massimo di page fault</b>	
<b>Motivo</b>	
<b>Percentuale di page fault</b>	
<b>Hit Rate</b>	
<b>Numero minimo di page fault</b>	
<b>Motivo</b>	
<b>Percentuale di page fault</b>	
<b>Hit Rate</b>	

3) Si consideri un gestore della memoria che utilizza la tecnica della paginazione.

- Quali sono le motivazioni che portano ad utilizzare la memoria paginata?

- La suddivisione della memoria in segmenti è compatibile con l'utilizzo della memoria paginata? Motivare la risposta.

- Si consideri un programma che accede in sequenza a tutti gli elementi di un array di 10000 parole (ogni elemento ha le dimensioni di una parola). Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 10 volte.

Si assume che le dimensioni di una pagina siano di 1K parole. Il programma occupa due pagine ed il suo *working set* è composto da 10 pagine. Quanti *page fault* avvengono durante l'esecuzione del programma considerando che nessuna pagina del programma è residente in memoria al momento della sua esecuzione?

- Si consideri una memoria cache 2-set associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

- 4) Identifica le parti dell' indirizzo RAM di una memoria da 8GB, indirizzabile per byte, nel caso sia collegata ad una memoria cache completamente associativa da 8MB con pagine da 1KB.  
Quali sono i vantaggi e gli svantaggi di una memoria completamente associativa?

# **Microistruzioni**



# Università di Verona

## Dipartimento Scientifico e Tecnologico

Architettura degli Elaboratori: prova intermedia 11/06/99

**Cognome:.....Nome: ..... Matricola: .....**

**Note:** *le soluzioni devono essere opportunamente commentate e motivate,  
è vietato utilizzare appunti o libri.*

- 1) Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che (%EBX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

ADDL (%EBX), %EAX

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....

- L'esecuzione di un programma compilato per una CPU antiquata da parte di una CPU più moderna, dotata solamente di un numero maggiore di registri generali, può migliorare i tempi di esecuzione?.

- 4) Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che (%EBX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

ADDL (%EBX), %EAX

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....

- Descrivere la struttura della CPU a 3 BUS e dire quali vantaggi potrebbero esserci nell'esecuzione dell'istruzione precedente.



# Università di Verona

## Dipartimento Scientifico e Tecnologico

Architettura degli Elaboratori: esame 12/12/00

**Cognome:.....Nome: ..... Matricola: .....**

**Note:** *le soluzioni devono essere opportunamente commentate e motivate,  
è vietato utilizzare appunti o libri.*

- 1) Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia tre BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

SUBL (%EAX), %EBX

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....

- Schematizzare il micropogramma necessario ad una CPU microprogrammata per eseguire la fase di fetch di una istruzione (considerare come esempio l'istruzione di questo esercizio).



# Università di Verona

## Dipartimento di Informatica

Architettura degli Elaboratori: esame 06/07/01

**Cognome: ..... Nome: ..... Matricola: .....**

**Note:** *le soluzioni devono essere opportunamente commentate,  
è vietato utilizzare appunti o libri,*  
**seconda prova intermedia: esercizi 1 e 2**  
**esame completo: esercizi 1, 2, 3 e 4**

- 1) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **TRE BUS**, che l'istruzione sia composta da una sola parola, che %EAX rappresenti un metodo di indirizzamento diretto a registro e che il salto sia di tipo indiretto (usare solamente le righe necessarie):

CALL %EAX

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Descrivere lo schema di controllo di una CPU cablata. Esemplificare lo schema utilizzando la fase di *fetch* dell'istruzione precedente costruendo gli opportuni segnali.

- 4) Elencare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

CALL (%EAX)

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Si sarebbero ottenuti dei vantaggi nell'esecuzione dell'istruzione precedente nel caso in cui la CPU fosse dotata di tre BUS?

- 4) Elencare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che l'indirizzo di salto della procedura sia relativo al PC (usare solamente le righe necessarie):

CALL (%EAX)

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....

- Dare la definizione di microprocessore superscalare e spiegare come sia possibile realizzarlo.

- 4) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia TRE BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto condizionato sia di tipo indiretto (usare solamente le righe necessarie):

CALL (%EAX)

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplicare lo schema utilizzando la fase di *fetch* dell'istruzione precedente.

- 4) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia tre BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

CALL (%EAX)

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplicare lo schema utilizzando la fase di *decodifica* dell'istruzione precedente..

- 4) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

CALL (%EAX)+%SI

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplificare lo schema utilizzando la fase di *fetch* dell'istruzione precedente..

- 4) 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 solo BUS, che l'istruzione sia composta da una sola parola, che (%Exx) rappresenti un metodo di indirizzamento indiretto a registro e che l'indirizzo di salto della procedura sia assoluto (usare solamente le righe necessarie e commentare ogni istruzione):

CALL (%EAX + %EBX)

commento

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....
16. ....
17. ....
18. ....
19. ....
20. ....

- 2) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **tre** BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro (usare solamente le righe necessarie):

JMP (%EAX)

*significato*

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....
8. .....
9. .....
10. .....

- Si consideri una memoria cache 2-set associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della cache e dell'indirizzo della RAM.

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

JNZ (%EAX)	commento
1. ....	
2. ....	
3. ....	
4. ....	
5. ....	
6. ....	
7. ....	
8. ....	
9. ....	
10. ....	
11. ....	
12. ....	

- Quali sono le motivazioni che spingono i microprocessori moderni a essere dotati di una pipeline.



# Università di Verona

## Dipartimento Scientifico e Tecnologico

Architettura degli Elaboratori: prova 27/06/00

seconda prova intermedia: esercizi 1 2 3 4

esame completo: esercizi 1 2 5 6

Cognome: ..... Nome: ..... Matricola: .....

**Note:** *le soluzioni devono essere opportunamente commentate e motivate,  
è vietato utilizzare appunti o libri.*

- 1) Elencare le micro istruzioni relative al caricamento, decodifica ed esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto condizionato sia di tipo diretto (usare solamente le righe necessarie):

JZ (%EAX)

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....

- Elencare le ottimizzazioni che devono essere eseguite sull'architettura di un calcolatore per raggiungere l'obiettivo di avere, per la maggioranza delle istruzioni, CPI=1

- 5) 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 u metodo di indirizzamento **indiretto a registro con spiazzamento** e che l' indirizzo del **salto** sia **assoluto**(usare solamente le righe necessarie e commentare ogni istruzione):

JNE \$8(%EBX)

commento

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....
16. ....
17. ....
18. ....

- Descrivere il meccanismo della memoria virtuale, specificando quali componenti devono essere presenti nella CPU per realizzarla.

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

JNZ (%EAX) + %EBX

## commento

1. ....
  2. ....
  3. ....
  4. ....
  5. ....
  6. ....
  7. ....
  8. ....
  9. ....
  10. ....
  11. ....
  12. ....
  13. ....
  14. ....

- 4) Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto condizionato sia di tipo diretto (usare solamente le righe necessarie):

JZ (%EAX)+%SI

1. ....
2. ....
3. ....
4. ....
5. ....
6. ....
7. ....
8. ....
9. ....
10. ....
11. ....
12. ....
13. ....
14. ....
15. ....

- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplificare lo schema utilizzando la fase di *fetch* dell'istruzione precedente..

# Pipeline

- 2) Quali sono le scelte, sull'architettura di una CPU, necessarie a permettere di raggiungere l'obiettivo di un CPI medio uguale a 1?

- Si consideri una CPU con una pipeline a 4 stadi (F, D, E, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi che la pipeline sia vuota al tempo 1 e che `jz` faccia riferimento all'istruzione `subl`.

Clock/istruzione	1	2	3	4	5				
<code>Addl %eax, %ebx</code>									
<code>Movl %eax, %ecx</code>									
<code>Subl %ebx, %ecx</code>									
<code>jz loop</code>									

- Per quale motivo può essere utile avere *cache* per i dati e per le istruzioni separate?.

- 4) Quali sono le ottimizzazioni, per una architettura basata sul modello di Von Neumann, necessarie a permettere a raggiungere l'obiettivo di un CPI medio uguale a 1?

- Si consideri una CPU con una pipeline a 4 stadi (F, D, E, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi che la pipeline sia vuota al tempo 1 e che **jz** faccia riferimento all'istruzione **subl**.

Clock/istruzione	1	2	3	4	5					
Addl %eax, %ebx										
Movl %ebx, %ecx										
Subl %eax, %ecx										
jz loop										

- Quali sono i vantaggi e gli svantaggi di una *cache* completamente associativa?

4) Rispondere alle seguenti domande riportando la motivazione della risposta.

- Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 4 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

- Si consideri una CPU con una pipeline a 4 stadi (F, D, O, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1.

clock/istruzione	1	2	3	4	5							
addl %eax, %ebx												
movl ind, %ecx												
subl %ebx, %ecx												
jz loop												

- Descrivere il meccanismo e l'utilità della *predizione dei salti*.

- 4) Quali sono le motivazioni che hanno fatto affermare la filosofia RISC.

- Si consideri una CPU con una pipeline a 4 stadi (F, D, O, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1 e che la condizione del salto sia falsa.

clock/istruzione	1	2	3	4	5				
<code>cmpl %eax, %ebx</code>									
<code>jz START</code>									
<code>subl %ebx, %ecx</code>									
<code>movl %edx, DATA</code>									

- Come è possibile riordinare le istruzioni in modo da diminuire il CPI totale? Qual è la percentuale di miglioramento del CPI dopo il riordino?

3. Rispondere alle seguenti domande riportando la motivazione della risposta.

- Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 6 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

- Si consideri una CPU con una pipeline a 4 stadi (F, D, E, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1.

clock/istruzione	1	2	3	4	5						
loop: addl %eax, %ebx											
movl ind, %ecx											
subl %ebx, %ecx											
jz loop											
movl %ecx, ind											

- L'esecuzione di un programma compilato per una CPU antiquata da parte di una CPU più moderna, dotata solamente di un numero maggiore di registri generali, può migliorare i tempi di esecuzione?

- 3) Si consideri una CPU con una pipeline a 4 stadi(F , D, E, W).Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1 e si facciano le opportune ipotesi sul salto condizionale.

<b>clock/istruzione</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>						
init:movl %ecx, ind											
addl \$4, %ebx											
cmpl \$0, %ebx											
jnz init											
addl %eax, %ecx											

- 4) Identifica le parti dell' indirizzo RAM di una memoria da 8GB, indirizzabile per byte, nel caso sia collegata ad una memoria cache completamente associativa da 8MB con pagine da 1KB.

Quali sono i vantaggi e gli svantaggi di una memoria completamente associativa?

- 4) Si descriva la struttura di una CPU con una pipeline a 4 livelli specificando il significato dei vari componenti.

- Si consideri una CPU con una pipeline a 4 stadi (F, D, O, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1.

clock/istruzione	1	2	3	4	5						
<code>mull %eax, %ebx</code>											
<code>movl %ecx, ind</code>											
<code>cmpl %ebx, 0h</code>											
<code>jnz inizio</code>											

- Come si applica il meccanismo della predizione dei salti all'esempio precedente?

# **Assembly**

- 3) Si consideri un programma che legge da tastiera dei caratteri (1byte) utilizzando la procedura `READ(0, IND, 1)`. Questa procedura legge un byte e lo pone nella cella di memoria all'indirizzo `IND` appena viene premuto un tasto sulla tastiera. Se il carattere ASCII letto è il 13, o se sono stati letti più di `MAX` caratteri, il programma trasforma in maiuscolo tutti i caratteri letti e li stampa utilizzando la procedura `WRITE(1, IND, 1)`. Se nessuno dei due casi si è verificato, il programma continua a leggere dei caratteri e a memorizzarli.

Si scriva un programma in assembler AT&T per Intel 80386 che realizza questo programma ricordando che il numero decimale 65 corrisponde al carattere 'A' ed il 97 al carattere 'a'.

```
.section .data
MAX:    .int 32
IND:    .space 32, 48

.section .text
.globl main
main:
/* inizializza il base pointer */
pushl %ebp
movl %esp,%ebp
```

- Spiegare le motivazioni che portano ad avere una struttura a segmenti della

- 4) Si scriva la funzione MASSIMO (in assembler AT&T per Intel 80386) che esamina un array di numeri interi, indirizzato con l'etichetta ARRAY, la cui lunghezza è indirizzata dall'etichetta LEN. Utilizzando la procedura MAX(IND1, IND2, IND3), la funzione identifica qual è il numero massimo presente nell'array. La procedura MAX richiede l'indirizzo di due numeri da confrontare, IND1 e IND2, e l'indirizzo di una cella di memoria, IND3, in cui copiare il numero massimo tra i due. Il numero massimo identificato dalla funzione viene messo sulla pila prima che la funzione termini.

```
.section .data
ARRAY: ... /* definizione dell'array */
LEN:   .int ... /* lunghezza dell'array */
/* ulteriori definizioni se necessarie */

.section .text
.globl massimo
massimo:
/* funzione per il calcolo del massimo */
```

- 2) Si consideri un programma che legge da tastiera dei caratteri (1byte) utilizzando la procedura `READ(0, IND, 1)`. Questa procedura legge un byte e lo pone nella cella di memoria all'indirizzo `IND` appena viene premuto un tasto sulla tastiera. Gli unici caratteri ammessi sono i numeri da 0 a 9, ogni altro carattere non viene considerato ne' memorizzato. Se il carattere ASCII letto è il 13, o se sono stati letti più di `MAX` caratteri, il programma trasforma la parola letta nel numero intero corrispondente e lo memorizza in `NUMERO`.

Si scriva un programma in assembler AT&T per Intel 80386 che realizza questo programma ricordando che il numero decimale 48 corrisponde al carattere ASCII '0'.

```
.section .data
MAX:    .int 6
IND:    .space 6, 48
NUMERO: .int 1

.section .text
.globl main
main:
/* inizializza il base pointer */
pushl %ebp
movl %esp,%ebp
```

- 3) Si consideri un programma che legge da tastiera dei caratteri (1byte) utilizzando la procedura READ(0, IND, 1). Questa procedura legge un byte e lo pone nella cella di memoria all'indirizzo IND appena viene premuto un tasto sulla tastiera. Se il carattere ASCII letto è il 13, o se sono stati letti più di MAX caratteri, il programma trasforma in maiuscolo tutti i caratteri letti e li stampa utilizzando la procedura WRITE(1, IND, 1). Se nessuno dei due casi si è verificato, il programma continua a leggere dei caratteri e a memorizzarli.

Si scriva un programma in assembler AT&T per Intel 80386 che realizza questo programma ricordando che il numero decimale 65 corrisponde al carattere 'A' ed il 97 al carattere 'a'.

```
.section .data
MAX:    .int 32
IND:    .space 32, 48

.section .text
.globl main
main:
    /* inizializza il base pointer */
    pushl %ebp
    movl %esp,%ebp
```

# **Teoria**

## Pipeline

- Quali sono le motivazioni che spingono i microprocessori moderni a essere dotati di una pipeline.
- 4) Si descriva la struttura di una CPU con una pipeline a 4 livelli specificando il significato dei vari componenti.
- 4) Rispondere alle seguenti domande riportando la motivazione della risposta.
- Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 4 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?
3. Rispondere alle seguenti domande riportando la motivazione della risposta.
- Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 6 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

## CPI unitario

- 4) Quali sono le ottimizzazioni, per una architettura basata sul modello di Von Neumann, necessarie a permettere a raggiungere l'obiettivo di un CPI medio uguale a 1?
- 2) Quali sono le scelte, sull'architettura di una CPU, necessarie a permettere di raggiungere l'obiettivo di un CPI medio uguale a 1?
- Elencare le ottimizzazioni che devono essere eseguite sull'architettura di un calcolatore per raggiungere l'obiettivo di avere, per la maggioranza delle istruzioni, CPI=1

## Cache

- Descrivere il principio su cui si basa il funzionamento della memoria cache.
- Quali sono i vantaggi e gli svantaggi di una *cache* completamente associativa?
- Per quale motivo può essere utile avere *cache* per i dati e per le istruzioni separate?.

## Paginazione

- 3) Si consideri un gestore della memoria che utilizza la tecnica della paginazione.
- Quali sono le motivazioni che portano ad utilizzare la memoria paginata?
  - La suddivisione della memoria in segmenti è compatibile con l'utilizzo della memoria paginata? Motivare la risposta.

## Memoria virtuale

- Descrivere il meccanismo della memoria virtuale, specificando quali componenti devono essere presenti nella CPU per realizzarla.

## Altro sulla memoria

- Quali sono i vantaggi e gli svantaggi delle memoria completamente associative rispetto alle memoria non associative.
  - Quali sono le motivazioni che fanno preferire la realizzazione di unità di controllo cablate rispetto a quelle microprogrammate.
  - Definire lo schema di un controllore cablato indicando i segnali utilizzati e la funzione dei blocchi presenti.
- Come si potrebbe utilizzare il concetto di *memoria interlacciata* per migliorare le prestazioni del chip precedente?
- Come si complicherebbe lo schema del chip nel caso di *memoria dinamica*?
- Su quali concetti si basa lo schema della *gerarchia di memoria*.

## BUS sincrono e asincrono

- 4) Descrivere le differenze tra un BUS sincrono e uno asincrono e i loro diversi campi di applicazione.
- Disegnare il diagramma temporale di una operazione di scrittura in memoria riportando l'andamento dei segnali necessari, in un BUS sincrono, a collegare la CPU alla memoria.

## CPU. A 3 bus

- Descrivere la struttura della CPU a 3 BUS e dire quali vantaggi potrebbero esserci nell'esecuzione dell'istruzione precedente.

## CPU antiquata, RISC, Predizione salti, Superscalare

- L'esecuzione di un programma compilato per una CPU antiquata da parte di una CPU più moderna, dotata solamente di un numero maggiore di registri generali, può migliorare i tempi di esecuzione?
- 4) Quali sono le motivazioni che hanno fatto affermare la filosofia RISC.
- Descrivere il meccanismo e l'utilità della *predizione dei salti*.
  - Dare la definizione di microprocessore superscalare e spiegare come sia possibile realizzarlo.

## Modello di Von Neuman

- 3) Si descriva il modello di Von Neuman.

## Interrupt, DMA e handshake

- 3) Si descriva il meccanismo di interrupt e i suoi campi di applicazione.
- Perché il semplice meccanismo di interrupt mal si adatta a dispositivi che trasferiscono grandi quantità di dati in memoria?
  - Disegnare il diagramma temporale della fase di scrittura in memoria, basata su handshake, in un BUS asincrono.

## Controllo decodifica e fetch

- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplificare lo schema utilizzando la fase di *fetch* dell'istruzione precedente..
- Schematizzare il microprogramma necessario ad una CPU microprogrammata per eseguire la fase di fetch di una istruzione (considerare come esempio l'istruzione di questo esercizio).
- Descrivere lo schema di controllo di una CPU cablata. Esemplificare lo schema utilizzando la fase di *fetch* dell'istruzione precedente costruendo gli opportuni segnali.
- Descrivere lo schema di controllo di una CPU microprogrammata. Esemplificare lo schema utilizzando la fase di *decodifica* dell'istruzione precedente..



### Esercizio 1

by Nicola Dall'Ora - Monday, 22 March 2021, 10:56 PM

Scrivere le microistruzioni per effettuare il fetch di un'istruzione, commentando ogni passo.



### Esercizio 2

by Nicola Dall'Ora - Monday, 22 March 2021, 10:59 PM

Descrivere le microistruzioni relative alla seguente istruzione:

INC %EAX



### Esercizio 3

by Nicola Dall'Ora - Monday, 22 March 2021, 11:00 PM

Descrivere le microistruzioni relative a questa istruzione:

INC variable

Si assume che la variabile 'variable' sia costituita da un indirizzo immediato, direttamente codificato nell'istruzione (IR\_imm\_field\_out).



### Esercizio 4

by Nicola Dall'Ora - Monday, 22 March 2021, 11:02 PM

Si descrivano le microistruzioni relative alla seguente istruzione:

CALL etichetta

Assunzioni:

La flag 'etichetta' costituita da un indirizzo immediato, direttamente codificato nell'istruzione: salto relativo, in quanto l'indirizzo di salto è specificato tramite etichetta simbolica.



### Esercizio 5

by Nicola Dall'Ora - Monday, 22 March 2021, 11:03 PM

Descrivere le microistruzioni della seguente istruzione:

CALL (%EAX, %EBX )



Assunzioni:

- L'indirizzo a cui saltare è %eax + %ebx (indirizzamento indiretto)

- Salto assoluto.

### Esercizio 6

by Nicola Dall'Ora - Monday, 22 March 2021, 11:03 PM

Descrivere le microistruzioni della seguente istruzione:

RET

(ritorno di una funzione).



### Esercizio 7

by Nicola Dall'Ora - Monday, 22 March 2021, 11:04 PM

Descrivere le microistruzioni della seguente istruzione:

JMP (%eax).



### Esercizio 8

by Nicola Dall'Ora - Monday, 22 March 2021, 11:04 PM

Descrivere le microistruzioni della seguente istruzione:

JZ (%eax).



### Esercizio 16

by Nicola Dall'Ora - Monday, 22 March 2021, 11:09 PM

Elencare e commentare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler:

CALL (%eax, %ebx)

dove:

- si assume che l'indirizzo a cui saltare sia memorizzato nella locazione di memoria puntata dal valore %eax+%ebx (indirizzamento indiretto);
- si assume salto assoluto (salvo diversamente specificato);
- si assume l'utilizzo di architettura Intel dove lo stack cresce verso indirizzi di memoria inferiori rispetto ed %ESP punta alla cima occupata dallo stack.



### Esercizio 17

by Nicola Dall'Ora - Monday, 22 March 2021, 11:09 PM

Elencare e commentare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler:

XCHG variabile, %eax

- si assume che la variabile sia costituita da un indirizzo immediato direttamente codificato dentro l'istruzione (con il segnale IR\_imm\_field disponibile).



## Esercizio 1

by Nicola Dall'Ora - Tuesday, 11 May 2021, 12:42 PM

Si consideri una memoria cache 2-set associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 4Mbyte indirizzabile per byte. Definire le dimensioni ed il significato delle parti dell'indirizzo della RAM.



## Esercizio 2

by Nicola Dall'Ora - Tuesday, 11 May 2021, 12:48 PM

Si consideri una CPU dotata di memoria cache 8-set associativa ( $2^3$  pagine nel set) di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 8M parole.

Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

Si assuma che la cache appena descritta sia utilizzata per i dati, che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente). La CPU esegue un programma che accede in sequenza a tutti gli elementi di un array di 8320 parole (ogni elemento ha le dimensioni di una parola) che è memorizzato a partire dall'indirizzo 0. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 5 volte.

Si assuma che il tempo di accesso alla cache sia di 1T e che il tempo di accesso alla memoria sia di 10T (entrambi i tempi si riferiscono alla lettura di una parola). Calcolare il rapporto (fattore di miglioramento) tra il sistema in presenza di cache e in assenza di cache per l'esecuzione di questo programma.



## Esercizio 3

by Nicola Dall'Ora - Tuesday, 11 May 2021, 12:56 PM

Si consideri una CPU dotata di memoria cache 2-associativa di 8K parole con 64 parole per blocco. Questa CPU è collegata ad una memoria RAM da 8M parole.

Definire le dimensioni dell'indirizzo necessario a indirizzare tutta la memoria RAM e definire le dimensioni dei campi PAROLA, BLOCCO ed ETICHETTA in cui questo indirizzo può essere suddiviso. Motivare la risposta con un opportuno schema.

Si assuma che la cache appena descritta sia utilizzata per i dati, che sia inizialmente vuota e che utilizzi un algoritmo di sostituzione dei blocchi di tipo LRU (sostituzione dell'elemento meno utilizzato di recente). La CPU esegue un programma che accede in sequenza a tutti gli elementi di un array di 8320 parole (ogni elemento ha le dimensioni di una parola) che è memorizzato a partire dall'indirizzo 0. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 5 volte.

Si assuma che il tempo di accesso alla cache sia di 1T e che il tempo di accesso alla memoria sia di 10T (entrambi i tempi si riferiscono alla lettura di una parola). Calcolare il rapporto (fattore di miglioramento) tra il sistema in presenza di cache e in assenza di cache per l'esecuzione di questo programma.



## Esercizio 4

by Nicola Dall'Ora - Tuesday, 11 May 2021, 12:57 PM

Quali sono i vantaggi e gli svantaggi delle memorie completamente associative rispetto alle memorie non associative?



## Esercizio 5

by Nicola Dall'Ora - Tuesday, 11 May 2021, 12:57 PM

Si assuma che un computer con memoria virtuale, strutturata a pagine di 4Kbyte, sia dotato di 64Mbyte di memoria fisica.

Si consideri un programma con un codice di 7.2Kbyte che accede ciclicamente in sequenza a tutti gli elementi di un array di 1000 record in cui ogni campo è composto da 2 numeri interi.

Quale deve essere la dimensione del working set perché si abbiano dei page fault solamente nella fase di caricamento del programma con esecuzione del primo ciclo di accesso agli elementi dell'array?

Quanti page fault si avrebbero durante l'esecuzione del programma ipotizzando che il working set abbia una pagina meno di quanto definito nel punto precedente e che il ciclo di accesso venga ripetuto 10 volte?



## Esercizio 6

by Nicola Dall'Ora - Tuesday, 11 May 2021, 1:02 PM

Quali sono i vantaggi di una cache a indirizzamento diretto?



## Esercizio 7

by Nicola Dall'Ora - Tuesday, 11 May 2021, 1:04 PM

Si consideri una memoria cache completamente associativa della dimensione di 64 Kbyte con 1024 byte per blocco. La cache è collegata ad una memoria di 16 Mbyte. Definire le dimensioni ed il significato delle parti dell'indirizzo della RAM.

### Esercizio 1

by Nicola Dall'Ora - Friday, 28 May 2021, 3:54 PM

Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 5 stadi.

Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

## Esercizio 2

by Nicola Dall'Ora - Friday, 28 May 2021, 3:56 PM

Per il seguente esercizio che coinvolge 4 istruzioni successive eseguite da una CPU con pipeline a 5 stadi, individuare lo stadio in cui si trova ogni istruzione ad ogni ciclo macchina (dove qui consideriamo un massimo di 11 cicli coinvolti).

Dopodichè ripetere gli esercizi introducendo le migliorie architettonali discusse a lezione al fine di ridurre gli stalli nella pipeline.

Per migliorie architettoniche parliamo di:

- 1) Inoltro di operandi (sia per operazioni logico-aritmetiche che di memoria)
  - 2) Branch prediction (o delay slot)

L'inoltro può avvenire su istruzioni a distanza maggiore di 1, naturalmente. E' evidente che la dipendenza di dati si può avere, in questa pipeline a 5 stadi, tra istruzioni fino a distanza 4.

Ognuno di questi casi va identificato (in software o hardware) e prevenuto/risolto onde evitare esecuzione scorretta.

### Esercizio 3

by Nicola Dall'Ora - Friday, 28 May 2021, 3:59 PM

Per il seguente esercizio che coinvolge 4 istruzioni successive eseguite da una CPU con pipeline a 5 stadi, individuare lo stadio in cui si trova ogni istruzione ad ogni ciclo macchina (dove qui consideriamo un massimo di 11 cicli coinvolti).

Dopodiché ripetere gli esercizi introducendo le migliorie architettoniche discusse a lezione al fine di ridurre gli stalli nella pipeline.

### Esercizio 4

Esercizio 4 by Nicola Dall'Ora - Friday, 28 May 2021, 4:01 PM

Per il seguente esercizio che coinvolge 5 istruzioni successive eseguite da una CPU con pipeline a 5 stadi, individuare lo stadio in cui si trova ogni istruzione ad ogni ciclo macchina.

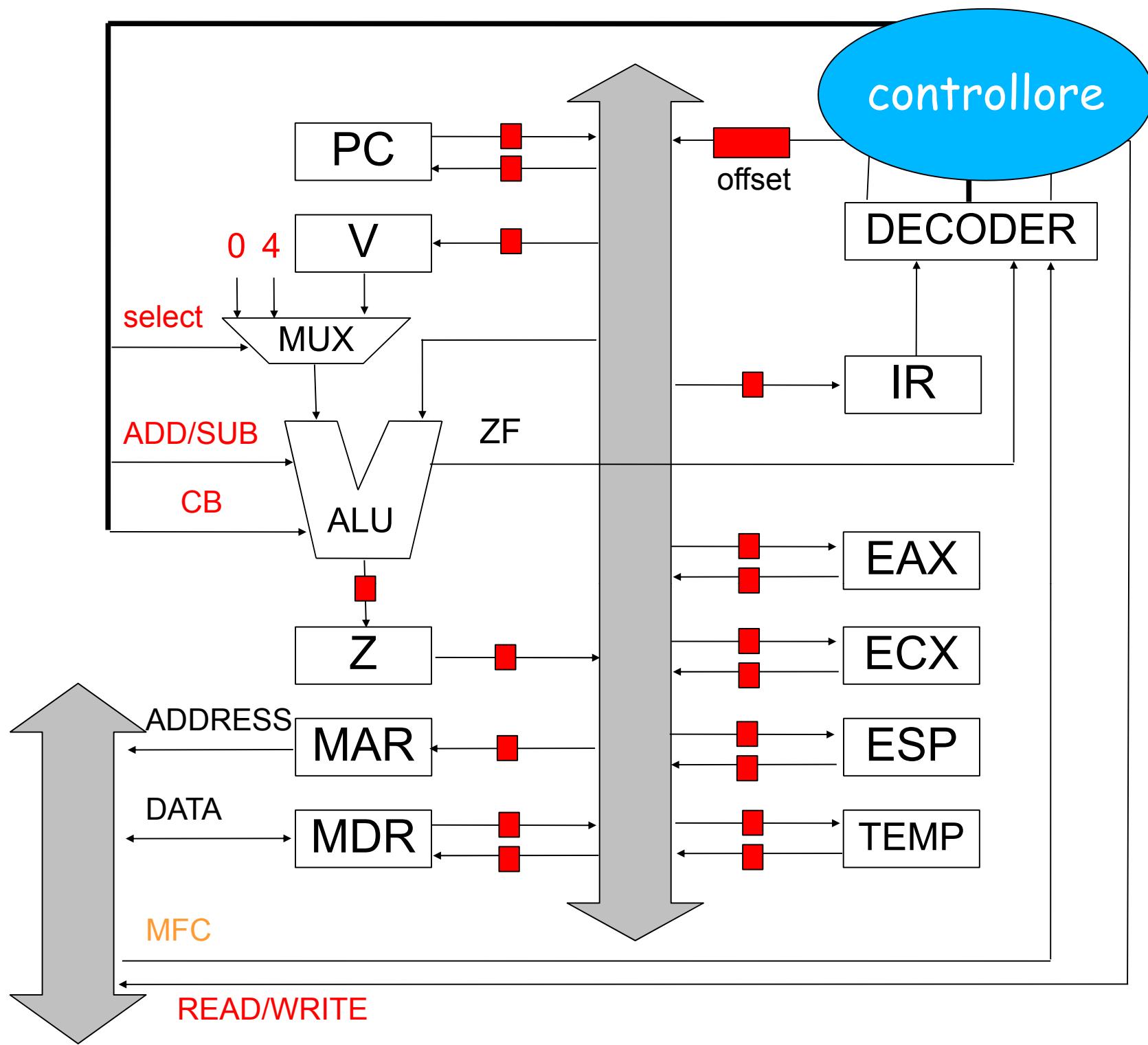
Dopodichè ripetere l'esercizio introducendo le migliorie a livello hardware o di compilatore discusse a lezione al fine di ridurre gli stalli nella pipeline.

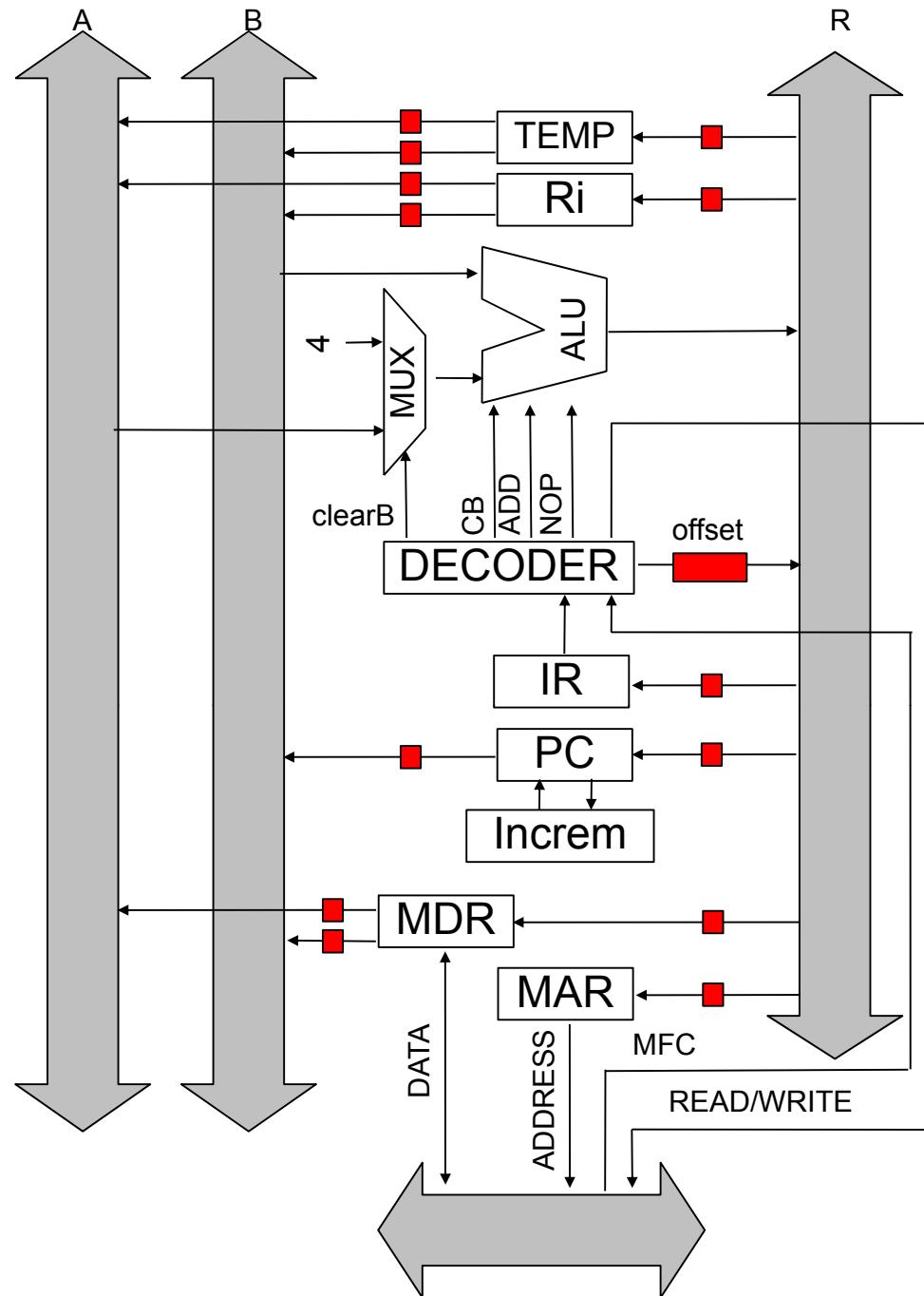
### Esercizio 5

Esercizio 3

Per il seguente esercizio che coinvolge 6 istruzioni eseguite da una CPU con pipeline a 5 stadi, individuare lo stadio in cui si trova ogni istruzione ad ogni ciclo macchina ipotizzando di predire che il salto non avvenga e che tale predizione sia confermata. Si assume che l'istruzione successiva al salto sia conosciuta al ciclo immediatamente successivo (perchè usiamo predizione statica oppure perchè l'istruzione di salto non ha effetti diretti su queste altre istruzioni).

Dopodiché ripetere l'esercizio introducendo le migliorie a livello hardware o di compilatore discusse a lezione al fine di ridurre gli stalli.







UNIVERSITÀ  
di VERONA

Dipartimento  
di INFORMATICA

Prof. Franco Fummi  
Prof. Luca Geretti  
Prof. Davide Quaglia

# ARCHITETTURA DEGLI ELABORATORI

Eserciziario su Memoria Virtuale

Materiale originariamente prodotto da

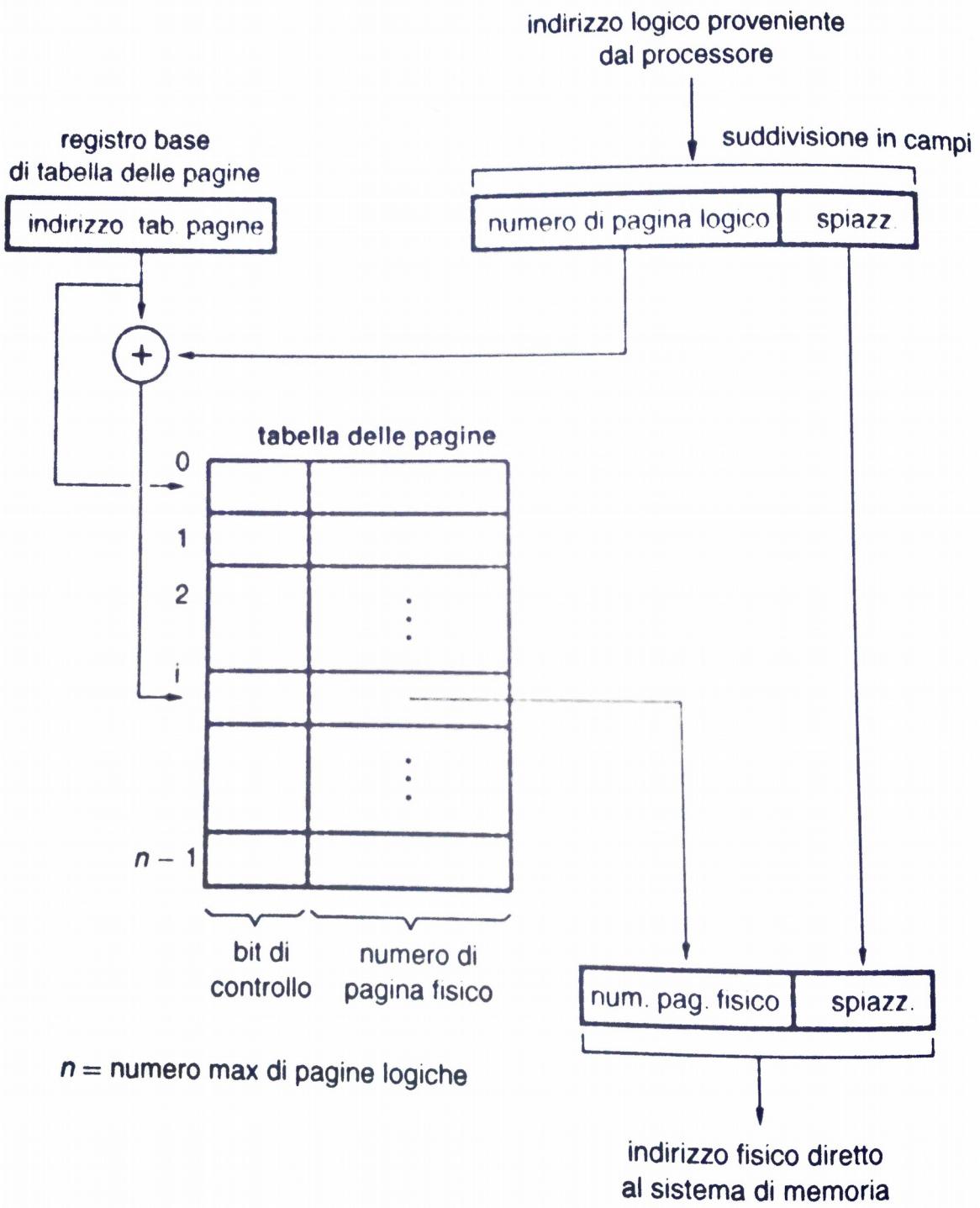
*Enrico Giordano*

*Serena Cavaletti*

*Katia Cracco*

*Roxana Belciug*

## SCHEMA DELLA MEMORIA VIRTUALE



## Esercizio 1

Si consideri un sistema a memoria virtuale con spazio logico di 4G parole, una memoria fisica di 32M parole e dimensione delle pagine di 16K parole.

1) Determinare il numero di bit che definiscono:

- Lunghezza indirizzo logico
- Numero di bit per definire la pagina logica
- Numero di bit per lo spiazzamento nella pagina logica
- Lunghezza indirizzo fisico
- Numero di bit per definire la pagina nella memoria fisica
- Numero di bit per lo spiazzamento nella pagina fisica

2) Nella seguente tabella sono riportati alcuni valori R del “resident set” o “working set” (=numero di pagine per processo residenti in memoria fisica); sapendo che il Sistema Operativo occupa permanentemente 448 pagine in memoria fisica e che sono stati creati 22 processi, indicare per ogni valore di R il numero di processi in stato di “fuori memoria” ossia che non possono avere tutte le pagine in memoria.

<b>R</b>	<b>40</b>	<b>80</b>	<b>160</b>	<b>320</b>	<b>800</b>
Numero di processi fuori memoria					

*Soluzione*

Lunghezza indirizzo logico = 32 bit

Numero di bit per definire la pagina logica =  $32 - 14 = 18$  bit

Numero di bit per lo spiazzamento (o offset) nella pagina logica = 14 bit

Lunghezza indirizzo fisico = 25 bit

Numero di bit per definire la pagina nella memoria fisica = 11 bit

Numero di bit per lo spiazzamento nella pagina fisica = 14 bit

Dim. Memoria fisica / dim. Pagina = Num. totale di pagine in memoria fisica

$2^{25} / 2^{14} = 2^{11} = 2048$  pagine in memoria fisica

$2048 - 448 = 1600$  pagine disponibili per i processi

#pagine disponibili / R = #processi ammissibili in memoria fisica

$1600 / 40 = 40 \Rightarrow 40 > 22 \Rightarrow$  Non ci sono processi fuori della memoria fisica

$1600 / 80 = 20 \Rightarrow 22 - 20 = 2$  processi fuori memoria fisica

$1600 / 160 = 10 \Rightarrow 22 - 10 = 12$  processi fuori memoria fisica

$1600 / 320 = 5 \Rightarrow 22 - 5 = 17$  processi fuori memoria fisica

$1600 / 800 = 2 \Rightarrow 22 - 2 = 20$  processi fuori memoria fisica

<b>R</b>	<b>40</b>	<b>80</b>	<b>160</b>	<b>320</b>	<b>800</b>
Numero di processi fuori memoria	0	2	12	17	20

## Esercizio 2

Si consideri un programma che accede in sequenza a tutti gli elementi di un array di 10000 elementi dove ogni elemento ha dimensione di una parola. Questa operazione di scansione è effettuata all'interno di un ciclo che viene eseguito 10 volte.

Si assume che la dimensione di una pagina di memoria virtuale sia di 1K parole. Il working set del processo è di 10 pagine di cui 2 occupate permanentemente dal codice del programma.

Quanti page fault avvengono durante l'esecuzione del programma considerando che nessuna pagina del programma è residente in memoria fisica all'avvia della sua esecuzione? Si assume che la politica di rimpiazzo sia LRU.

### Soluzione

Il codice del programma occupa 2 pagine e si assume che non venga mai tolto dal working set.

Occupazione dei dati:  $10000 \text{ parole} / 1024 \text{ parole} (\text{dimensione pagina}) = 9.76$   
Quindi servono 10 pagine per caricare tutto l'array.

Dopo aver caricato le prime 8 pagine dell'array il working set risulta il seguente.

Codice 1	Codice 2	Dati 1	Dati 2	Dati 3	Dati 4	Dati 5	Dati 6	Dati 7	Dati 8
----------	----------	--------	--------	--------	--------	--------	--------	--------	--------

1<sup>o</sup> ciclo: Considerando che la memoria non contiene nessuna pagina del programma e dei suoi dati, avvengono 12 page fault (2 per il caricamento del codice e 10 per il caricamento dell'array). La mappa del working set alla fine della prima scansione è:

Codice 1	Codice 2	Dati 9	Dati 10	Dati 3	Dati 4	Dati 5	Dati 6	Dati 7	Dati 8
----------	----------	--------	---------	--------	--------	--------	--------	--------	--------

2<sup>o</sup> - 10<sup>o</sup> ciclo: Utilizzando l'algoritmo LRU le pagine Dati 9 e Dati 10 hanno sostituito le pagine Dati 1 e Dati 2 nel ciclo precedente, pagine che sono necessarie nel secondo ciclo di scansione. Per caricare la memoria, utilizzando LRU, la pagina Dati 1 sostituisce la pagina Dati 3 e la pagina Dati 2 sostituisce la pagina Dati 4. In questo modo non si avrà mai una pagina Dati utile nella memoria. → 10 page fault per ciclo successivo al primo.

TOTALE:  $12 + 10 \times 9 = 102$  page fault.

### NOTE:

1. In mancanza di specifiche precise sulle pagine di codice, esse si devono ritenere fisse nella memoria fisica una volta che esse sono state caricate.
2. Tutte le volte che la CPU legge ciclicamente M pagine in sequenza da posizionare in uno spazio di N pagine con la politica di rimpiazzo LRU valgono i seguenti fatti:
  - al primo ciclo ci sono sempre M page fault
  - ai cicli successivi se  $N \geq M$  non ci sono page fault
  - ai cicli successivi se  $N < M$  ci sono sempre M page faultQueste considerazioni valgono anche per gli esercizi sulle cache completamente associative dove le pagine diventano blocchi e i page fault diventano cache miss.

### Esercizio 3

Si assuma che un computer abbia memoria virtuale strutturata a pagine di 4Kbyte.

Si consideri un programma con un codice di 7.2 Kbyte che accede ciclicamente in sequenza a tutti gli elementi di un array di 1000 record in cui ogni campo è composto da 2 numeri interi ciascuno di 4 byte.

Quale deve essere la dimensione del *working set* perché si abbiano dei *page fault* solamente nella fase di caricamento del programma con esecuzione del primo ciclo di accesso agli elementi dell'array?

Quanti *page fault* si avrebbero durante l'esecuzione del programma ipotizzando che il *working set* abbia una pagina meno di quanto definito nel punto precedente e che il ciclo di accesso venga ripetuto 10 volte?

*Soluzione*

Dim pagina

4 Kbyte

Dim codice

7.2 Kbyte

Dim array

1000 record di cui ogni campo contiene 2 numeri interi ciascuno da 4 byte

Il codice è 7.2 Kbyte e quindi occupa 2 pagine.

Parte di dati:  $2 \text{ (numeri)} * 4 \text{ (byte)} * 1000 \text{ (record)} = 8000 \text{ byte}$   
 $8000 \text{ byte} / 4 \text{ Kbyte} = 1,95$

La parte di dati occupa quindi 2 pagine.

Codice 1	Codice 2	Dati 1	Dati 2
----------	----------	--------	--------

La dimensione del *working set* perché si abbiano dei *page fault* solamente nella fase di caricamento del programma con esecuzione del primo ciclo di accesso agli elementi dell'array è di 4 pagine.

Ipotizzando che il *working set* abbia una pagina in meno di quanto definito nel punto precedente si hanno 3 pagine:

Codice 1	Codice 2	Dati 1 oppure 2
----------	----------	-----------------

1<sup>o</sup> ciclo: 4 page fault

Nei cicli successivi al primo tutte le pagine di dato generano sempre *page fault*

2<sup>o</sup> - 10<sup>o</sup> ciclo: 2 page fault \* 9 cicli = 18 page fault

TOTALE: 4 + 18 = 22 page fault



UNIVERSITÀ  
di **VERONA**

Dipartimento  
di **INFORMATICA**

**Prof. Franco Fummi**  
**Prof. Luca Geretti**  
**Prof. Davide Quaglia**

# ARCHITETTURA DEGLI ELABORATORI

**Eserciziario di Microistruzioni**

Da materiale di

***Enrico Giordano***

***Serena Cavalletti***

***Katia Cracco***

***Roxana Belciug***

## Regole ed accorgimenti

Per svolgere questi esercizi, è necessario innanzitutto sapere come funzionano le istruzioni Assembly Intel (in sintassi AT&T). Ciò è importante perché non bisogna imparare a memoria tutte le microistruzioni, ma bisogna ragionare per “istruire” il nostro microprocessore. Un buon esperimento consiste nel creare dei piccoli programmi in Assembly e vedere, tramite il Debugger, quali registri vengono modificati e come viene impostato in particolare il PC, il registro puntatore di Stack ESP e EBP. Capire bene le microistruzioni significa evitare i Segmentation Fault nella creazione di programmi in Assembly; spesso però, dalle Segmentation Fault si imparano le microistruzioni.

Bisogna tener presente che **un solo valore per volta può passare nel bus**; per sicurezza, ricordarsi sempre di controllare che non si stiano manipolando due valori differenti in un'unica microistruzione. Ad esempio, se devo utilizzare il valore del registro EAX e lo devo sommare al valore di EBX, sicuramente uno dei due deve essere precedentemente salvato in un registro ausiliario della ALU (che nella nostra architettura di riferimento si chiama V).

Particolare attenzione verso queste scritture in **notazione AT&T per Intel 80386**:

- \$5    Valore *immediato* contenuto direttamente nell'istruzione codificata;
- %REG                                        *Indirizzamento diretto a registro*: il valore viene prelevato dal registro;
- (%REG)                                      *Indirizzamento indiretto a registro*: il valore si trova nell'indirizzo di memoria puntato da EAX, quindi occorre decodificarlo inviando il valore del registro EAX nel registro MAR (Memory Address Register), aspettare la lettura in memoria (WMFC), prelevare il valore ottenuto tramite MDR (Memory Data Register).
- 4(REG1, REG2)                             *Indirizzamento indiretto con spiazzamento*: come per il precedente, con l'aggiunta che il valore di %REG1 deve essere sommato a %REG2 e allo spiazzamento 4 prima di essere scritto in MAR. Il valore davanti alla parentesi e uno dei parametri dentro le parentesi sono opzionali.

Esempi di istruzioni Assembly in notazione AT&T per INTEL 386:

**ADDL \$5, %eax**     $5+EAX \rightarrow EAX$  \$5 è un parametro IMMEDIATO L=32 bit

**ADDL %ebx, %eax**  $EAX+EBX \rightarrow EAX$  parametri sono registri

**ADDL (%ebx), %eax**  $EAX + RAM[EBX] \rightarrow EAX$  modalità indiretta a registro

**ADDL 4(%ebx), %eax**  $EAX + RAM[EBX+4] \rightarrow EAX$  modalità indiretta a registro con spiazzamento

Bisogna ricordare infine di distinguere la fase di Fetch, che in tutte le microistruzioni è uguale, e la fase di Execute. Di seguito vengono riportate alcune istruzioni Assembly Intel 80X86 e la relativa sequenza di passi.

#### ***MOV src, dst***

Il valore di “src” viene spostato in “dest”, bisogna quindi prelevare il valore del primo registro “src” e caricarlo nel registro “dst”. Questa istruzione consente l’inizializzazione di un registro o di un’area di memoria. Accetta inoltre i modificatori *l*, *w* e *b* per indicare la dimensione dell’operatore “src” (32, 16 e 8 bit rispettivamente).

#### ***ADD src, dest***

Viene sommato il valore di “src” a “dest” e il valore ottenuto viene caricato in “dest”; bisogna perciò prelevare il valore del primo registro “src” e caricarlo nel registro ausiliario della ALU, poi prelevare il valore di “dest” e direttamente sommarlo al valore, precedentemente caricato nel registro ausiliario, tramite il comando ADD (segnale di controllo della ALU), per poi salvarlo nel registro di uscita della ALU (chiamato Z). Infine il valore in Z deve essere inviato in “dest”.

#### ***SUB src, dest***

Questa non è molto differente dalla ADD; semplicemente bisogna fare il complemento a 2 di “src” ed eseguire le medesime microistruzioni di ADD.

#### ***CMP src, dest***

Questa istruzione esegue un confronto tra “src” e “dst” e aggiorna il valore dei vari FLAG (Zero flag - ZF, Greater flag - GF, Equal flag - EF, Lower flag - LF) .

#### ***J... value***

Questa indica che deve essere eseguito un salto verso “value” (può essere un’etichetta, un valore di registro, ecc...) in base alla situazione descritta dopo la lettera “J”:

JZ: Jump if Zero

JE: Jump if Equal

JNE: Jump if Not Equal

JL : Jump if Lower

JLE: Jump if Lower or Equal

JG: Jump if Greater

JGE: Jump if Greater or Equal

JMP: Jump without condition (unconditional jump)

Prima di tutto, dopo la fase di Fetch, l'istruzione di *salto condizionato* controlla se il valore dei vari FLAG (Zero flag - ZF, Greater flag - GF, Equal flag - EF, Lower flag - LF) *generati dalla ALU da una precedente istruzione CMP (Compare)* soddisfa la condizione; se non la soddisfa, l'istruzione finisce subito, altrimenti bisogna eseguire il salto. L'istruzione CMP deve quindi necessariamente precedere un'istruzione di salto condizionato. Si fa un'eccezione per JMP, ovvero salto incondizionato, in cui si salta direttamente senza controlli.

Deve poi essere decodificato il valore “value” che può provenire da diversi luoghi, in base alla situazione (se è un'etichetta, proviene dall'OFFSET, se è un registro proviene da esso, se è un indirizzamento vedere sopra).

Una volta ottenuto il valore deve essere sommato al valore di PC e poi inserito nel PC (**salto relativo**). Si fa un'eccezione per i **salti assoluti**, in quanto il valore “value” deve essere inserito direttamente in PC senza somma.

### **PUSH value**

L'operazione di Push consiste nell'inserire un valore “value” nella cima dello Stack; occorre perciò decrementare il registro ESP di 4 per puntare ad una cella vuota dello Stack (superiore a quella attuale), caricare il nuovo valore di ESP sia in ESP sia in MAR, prendere il valore “value” e caricarlo nel registro MDR e dare il comando WRITE, in modo da scrivere il valore nella cella prescelta. Dopo WMFC l'istruzione termina. Può essere seguito solo dalle lettere “w” (16 bit) o “l” (32 bit). Il registro ESP viene decrementato di tanti byte quanti ne vengono messi sullo stack ( 4 con “l” e 2 con “w”)

### **POP dest**

L'operazione di POP consiste del prelevare il valore della cima dello Stack e caricarlo nel luogo “dest”; è l'esatto opposto della PUSH. Deve caricare ESP in MAR dando il comando di lettura READ e, dopo WMFC, si preleva il valore ottenuto da MDR e si carica nella destinazione “dest”. Successivamente deve essere incrementato di 4 il valore di ESP in modo da puntare ad una cella sottostante dello Stack e salvare il nuovo valore di ESP in ESP. Può essere seguito solo dalle lettere “w” (16 bit) o “l” (32 bit). Il registro ESP viene incrementato di tanti byte quanti ne vengono messi sullo stack ( 4 con “l” e 2 con “w”)

**N.B.** Il recupero di informazioni dello stack mediante più chiamate dell'istruzione POP deve avvenire nell'ordine inverso del loro inserimento nello stack mediante le istruzioni PUSH.

#### ***CALL value e RET***

Call indica una chiamata a funzione; la funzione è identificata da “value”, che può essere un indirizzo, un'etichetta, un indirizzamento diretto. Si può dire che prima viene eseguita un'operazione di Push del valore di PC; infatti viene caricato il valore decrementato di 4 di ESP in MAR, poi viene caricato il valore di PC in MDR e si dà il comando di scrittura. Questo serve perché si salva in cima allo Stack il valore di ritorno della funzione, cioè il valore del PC (per così dire serve al microprocessore per ricordarsi cosa stava facendo prima di iniziare ad eseguire la funzione chiamata), per poi riottenerla con l'istruzione finale della funzione che si chiama RET.

La RET estra dalla cima dello stack un valore che corrisponde all'indirizzo dell'istruzione successiva all'istruzione CALL, lo assegna al PC facendo saltare il processore a tale istruzione. In poche parole esegue l'esatto opposto di CALL, cioè preleva dallo Stack il valore di PC.

**N.B.** Al momento dell'esecuzione della RET in cima allo stack deve essere presente il valore che era stato messo dalla CALL. Quindi nella funzione il numero di istruzioni PUSHW (e PUSHL) deve essere uguale al numero di istruzione POPW (e POPL) affinchè al momento dell'esecuzione della RET lo stack sia nelle stesse condizioni in cui si trovava all'inizio della funzione.

Sia per la CALL sia per le varie istruzioni di salto bisogna fare attenzione al tipo di salto:

*Salto Assoluto:* il valore “value” deve essere inserito direttamente in PC;

**PC=[Value]**

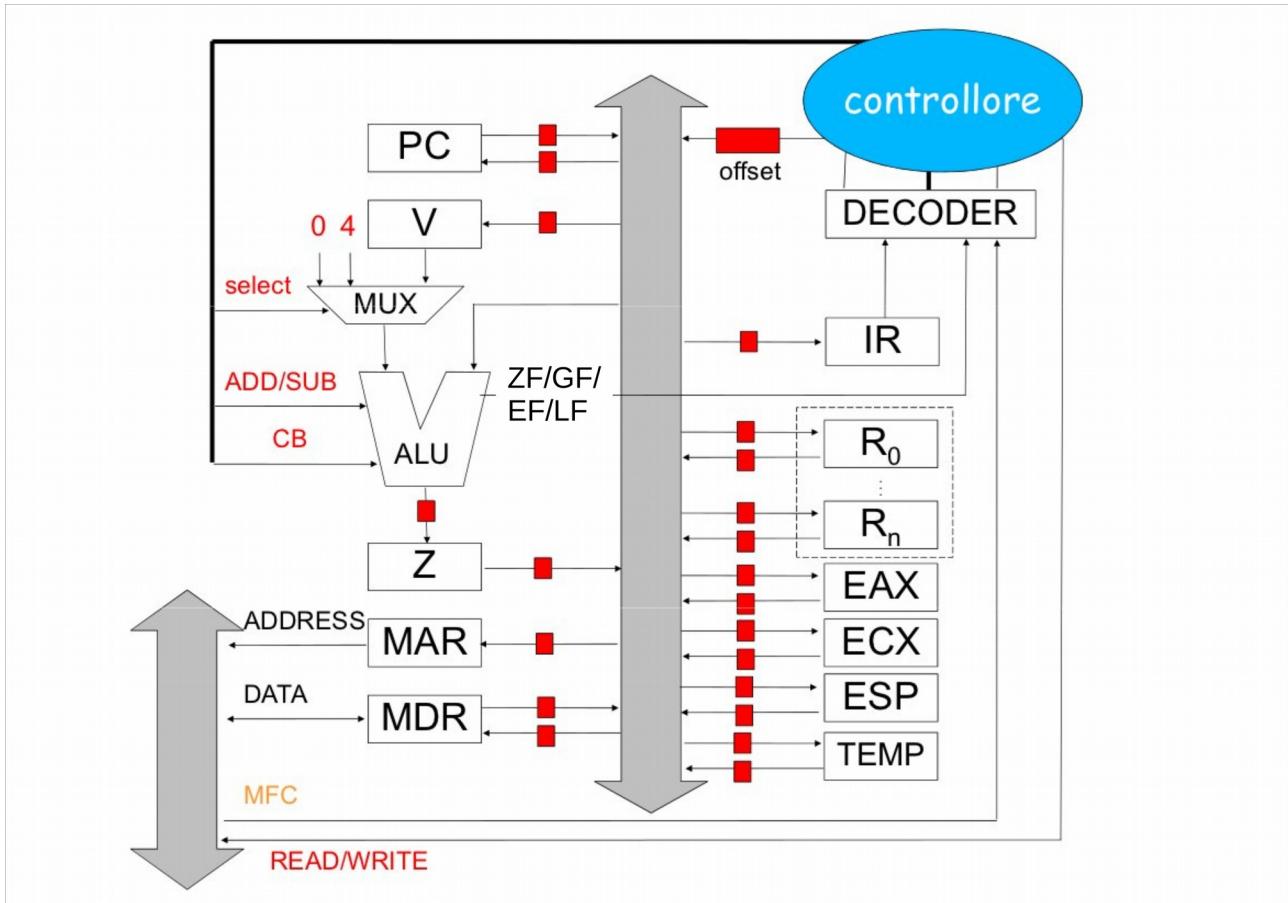
*Salto Relativo:* il valore “value” deve essere sommato a PC e il risultato deve essere inserito nel registro PC;

$$\text{PC}=[\text{PC}+\text{Value}]$$

Ovviamente tutti gli esercizi possono avere delle varianti di tema; può succedere che si presenti una ADD non semplicemente tra 2 valori come ad esempio ADD 4(%eax, %ebx), %ecx. Bisogna essere in grado di unire tutte le varianti e scindere l'istruzione in altrettante istruzioni conosciute, in modo da non sbagliare. Sicuramente un buon esercizio può aiutare a riconoscere le parti.

Di seguito si trova l'architettura di riferimento per svolgere gli esercizi. È buona cosa ricordarla a memoria per svolgere gli esercizi durante l'esame.

## Architettura ad 1 bus



## Esercizio 1

Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che \$20 rappresenti un operando immediato:

**MOVL \$20, %EBX**

Soluzione:

1.  $PC_{OUT}$ ,  $MAR_{IN}$ , **READ**, **SELECT[4]**, ADD,  $Z_{IN}$
  2. **WMFC**,  $Z_{OUT}$ ,  $PC_{IN}$
  3.  $MDR_{OUT}$ ,  $IR_{IN}$
  4. **OFFSET(IR)<sub>OUT</sub>**,  $EBX_{IN}$ , **END**
- $\left. \begin{array}{l} \text{1. } PC_{OUT}, MAR_{IN}, \text{READ, SELECT[4], ADD, } Z_{IN} \\ \text{2. WMFC, } Z_{OUT}, PC_{IN} \\ \text{3. } MDR_{OUT}, IR_{IN} \\ \text{4. OFFSET(IR)}_{OUT}, EBX_{IN}, END \end{array} \right\} \text{Fetch}$

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

### Execute:

4. Esce sul bus il valore immediato 20 tramite la parte OFFSET dell'IR e viene abilitato l'ingresso del registro EBX. Termine istruzione

NOTA: Il fatto che il PC, già nella fase di fetch, sia incrementato di 4 per puntare già all'istruzione successiva è una convenzione delle CPU della famiglia Intel.

## Esercizio 2

Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola e che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro:

**MOVL (%EAX), %EBX**

Soluzione:

1.  $PC_{OUT}$ ,  $MAR_{IN}$ , **READ**,  $SELECT[4]$ , ADD,  $Z_{IN}$
  2. **WMFC**,  $Z_{OUT}$ ,  $PC_{IN}$
  3.  $MDR_{OUT}$ ,  $IR_{IN}$
- } Fetch
4.  $EAX_{OUT}$ ,  $MAR_{IN}$ , READ
  5. WMFC
  6.  $MDR_{OUT}$ ,  $EBX_{IN}$ , **END**

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
  2. Viene messo in PC il valore  $PC+4$  e si aspetta il completamento della lettura da memoria
  3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)
- 
4. Il valore di EAX viene passato al registro degli indirizzi MAR per la lettura della memoria
  5. Aspetto il completamento della lettura da memoria
  6. Il dato in memoria a quel indirizzo è ora pronto in MDR e viene passato a EBX. Termine istruzione

### Esercizio 3

Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **un solo BUS**, che l'istruzione sia composta da una sola parola e che (%EBX) rappresenti un metodo di indirizzamento indiretto a registro:

**ADDL (%EBX), %EAX**

Soluzione:

1.  $\text{PC}_{\text{OUT}}$ ,  $\text{MAR}_{\text{IN}}$ , READ, SELECT[4], ADD,  $Z_{\text{IN}}$
  2. WMFC,  $Z_{\text{OUT}}$ ,  $\text{PC}_{\text{IN}}$
  3.  $\text{MDR}_{\text{OUT}}$ , IR<sub>IN</sub>
- }
- Fetch
- 
4. EBX<sub>OUT</sub>, MAR<sub>IN</sub>, READ
  5. WMFC, EAX<sub>OUT</sub>, V<sub>IN</sub>
  6. MDR<sub>OUT</sub>, SELECT[V], ADD, Z<sub>IN</sub>
  7. Z<sub>OUT</sub>, EAX<sub>IN</sub>, END

Commenti:

#### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
  2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
  3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)
- 
4. Il valore di EBX viene passato al registro degli indirizzi MAR per la lettura della memoria
  5. Aspetto il completamento della lettura da memoria attraverso il comando WMFC; nel frattempo, siccome il BUS non è occupato, il valore contenuto nel registro EAX viene memorizzato temporaneamente nel registro ausiliario della ALU chiamato V
  6. Il dato in memoria a quel indirizzo è ora pronto in MDR e viene sommato a V mettendo il risultato in Z
  7. Il risultato in Z viene memorizzato in EAX. Termine dell'istruzione.

#### Esercizio 4

Elencare le micro istruzioni relative al caricamento, decodifica ed esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che ETICHETTA sia un valore immediato e che il salto condizionato sia relativo al PC:

#### JZ ETICHETTA

Soluzione:

1.  $PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}$
  2.  $WMFC, Z_{OUT}, PC_{IN}, V_{IN}$
  3.  $MDR_{OUT}, IR_{IN}$
- } Fetch
4. if (!ZERO) END,  $OFFSET(IR)_{OUT}, SELECT[V], ADD, Z_{IN}$
  5.  $Z_{OUT}, PC_{IN}, END$

Commenti:

##### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
  2. Viene messo in PC **e in V** il valore  $PC+4$  e si aspetta il completamento della lettura da memoria
  3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)
- 
4. Se il risultato dell'istruzione precedente è diverso da zero allora termina, altrimenti il valore ETICHETTA contenuto del campo OFFSET di IR viene sommato a **quello del PC attuale già precedentemente parcheggiato in V (risparmiando un ciclo di clock)**
  5. L'indirizzo risultante della prossima istruzione da eseguire viene messo nel PC. Termine dell'istruzione.

NOTA: Il fatto che il PC, già nella fase di fetch, sia incrementato di 4 non deve preoccupare nei salti (o call) relativi al PC perché il valore specificato come argomento del salto/call sarà stato generato dal compilatore/assemblatore già tenendo conto di questa convenzione delle CPU Intel.

NOTA 2: si noti nel test alla riga 4 l'uso del segnale dello Zero flag (ZF) generato dall'ALU nell'istruzione precedente.

## Esercizio 5

Elencare le micro istruzioni relative al caricamento, decodifica ed esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto condizionato sia relativo al PC:

### JZ (%EAX)

Soluzione:

- 
1.  $PC_{OUT}$ ,  $MAR_{IN}$ , READ, SELECT[4], ADD,  $Z_{IN}$   
2. WMFC,  $Z_{OUT}$ ,  $PC_{IN}$   
3.  $MDR_{OUT}$ , IR<sub>IN</sub>
- } Fetch
4. if (!ZERO) END,  $EAX_{OUT}$ ,  $MAR_{IN}$ , READ  
5. WMFC,  $PC_{OUT}$ , V<sub>IN</sub>  
6.  $MDR_{OUT}$ , SELECT[V], ADD,  $Z_{IN}$   
7.  $Z_{OUT}$ ,  $PC_{IN}$ , END

Commenti:

#### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

#### Salto:

4. Se il risultato dell'istruzione precedente è diverso da zero allora termina, altrimenti il contenuto di EAX viene passato al MAR come un indirizzo e viene comandata una lettura
5. Attesa della lettura; nel frattempo, siccome il BUS non è occupato, il valore corrente del PC viene messo nel registro ausiliario della ALU chiamato V
6. L'indirizzo prodotto da MDR viene sommato a V in quanto il salto è relativo
7. L'indirizzo risultante (che punta alla prossima istruzione da eseguire) viene messo nel PC. Termine dell'istruzione.

## Esercizio 6

Elencare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che l'indirizzo di salto della procedura sia relativo al PC:

### CALL (%EAX)

Soluzione:

- 
- The diagram illustrates the micro-operations for a `CALL (%EAX)` instruction. It is divided into two main phases:
- Fetch:** This phase consists of three micro-operations:
    - 1.  $\text{PC}_{\text{OUT}}, \text{MAR}_{\text{IN}}, \text{READ}, \text{SELECT}[4], \text{ADD}, Z_{\text{IN}}$
    - 2.  $\text{WMFC}, Z_{\text{OUT}}, \text{PC}_{\text{IN}}$
    - 3.  $\text{MDR}_{\text{OUT}}, \text{IR}_{\text{IN}}$A brace on the right groups these three operations under the label "Fetch".
  - Salvataggio nello stack (PUSH):** This phase consists of five micro-operations:
    - 4.  $\text{ESP}_{\text{OUT}}, \text{SELECT}[4], \text{SUB}, Z_{\text{IN}}$
    - 5.  $Z_{\text{OUT}}, \text{ESP}_{\text{IN}}, \text{MAR}_{\text{IN}}$
    - 6.  $\text{PC}_{\text{OUT}}, \text{MDR}_{\text{IN}}, \text{WRITE}$
    - 7.  $\text{WMFC}$
    - 8.  $\text{EAX}_{\text{OUT}}, \text{MAR}_{\text{IN}}, \text{READ}$A brace on the right groups these five operations under the label "Salvataggio nello stack (PUSH)".
- Following the stack operation, there are three more micro-operations:
- 9.  $\text{WMFC}, \text{PC}_{\text{OUT}}, V_{\text{IN}}$
  - 10.  $\text{MDR}_{\text{OUT}}, \text{SELECT}[V], \text{ADD}, Z_{\text{IN}}$
  - 11.  $Z_{\text{OUT}}, \text{PC}_{\text{IN}}, \text{END}$

Commenti:

#### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore  $\text{PC}+4$  e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

#### Salvataggio dello stack:

4. Viene decrementato il valore dell'ESP (stack pointer) per puntare alla nuova cima dello Stack.
5. Il nuovo valore viene caricato in ESP e nel MAR per la fase di scrittura.
6. Il PC passa nel registro dei dati MDR e viene eseguita l'operazione di scrittura, salvando in memoria (nella cima dello Stack) l'indirizzo attuale di PC.
7. Attesa scrittura.

#### Chiamata a funzione:

8. Il valore di EAX viene passato al MAR come indirizzo di memoria. Viene comandata una lettura.
9. Attesa lettura; nel frattempo PC viene caricato nel registro V, siccome il BUS non è occupato.
10. Dal MDR esce il valore letto che viene sommato a V in quanto il salto è relativo.
11. L'indirizzo risultante dell'istruzione da eseguire viene passato al PC. Termine dell'istruzione.

## Esercizio 7

Elencare le micro istruzioni relative al caricamento, decodifica ed esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS e che l'istruzione sia composta da una sola parola:

### RET

1.  $\text{PC}_{\text{OUT}}$ ,  $\text{MAR}_{\text{IN}}$ , READ, SELECT[4], ADD,  $Z_{\text{IN}}$
  2. WMFC,  $Z_{\text{OUT}}$ ,  $\text{PC}_{\text{IN}}$
  3.  $\text{MDR}_{\text{OUT}}$ ,  $\text{IR}_{\text{IN}}$
- } Fetch
- 
4.  $\text{ESP}_{\text{OUT}}$ ,  $\text{MAR}_{\text{IN}}$ , READ, SELECT[4], ADD,  $Z_{\text{IN}}$
  5. WMFC,  $Z_{\text{OUT}}$ ,  $\text{ESP}_{\text{IN}}$
  6.  $\text{MDR}_{\text{OUT}}$ ,  $\text{PC}_{\text{IN}}$ , END
- } Recupero dallo Stack (POP)

Commenti:

#### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

#### Recupero dallo stack:

4. Viene messo il valore di ESP in MAR e comandata la lettura della cima dello Stack; il contenuto di ESP viene anche incrementato per puntare all'elemento sottostante dello Stack, nuova cima dello Stack.
5. Il nuovo valore viene caricato in ESP e si attende la fine della lettura.

#### Ritorno alla funzione chiamante:

6. Dall'MDR esce il valore letto che viene passato a PC in quanto il salto all'indietro in una RET è sempre assoluto. Termine dell'istruzione.

## Esercizio 8

Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX, %ESI) rappresenti un metodo di indirizzamento indiretto a registro con spiazzamento e che il salto condizionato sia relativo al PC:

**JZ (%EAX,%ESI)**

Soluzione:

- 1.PC<sub>OUT</sub>, MAR<sub>IN</sub>, READ, SELECT[4], ADD, Z<sub>IN</sub>  
2.WMFC, Z<sub>OUT</sub>, PC<sub>IN</sub>  
3.MDR<sub>OUT</sub>, IR<sub>IN</sub>
- 4.if (!ZERO) END, EAX<sub>OUT</sub>, V<sub>IN</sub>  
5.ESI<sub>OUT</sub>, SELECT[V], ADD, Z<sub>IN</sub>  
6.Z<sub>OUT</sub>, MAR<sub>IN</sub>, READ  
7.WMFC, PC<sub>OUT</sub>, V<sub>IN</sub>  
8.MDR<sub>OUT</sub>, SELECT[V], ADD, Z<sub>IN</sub>  
9.Z<sub>OUT</sub>, PC<sub>IN</sub>, END
- } Fetch

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)
4. Se il risultato dell'istruzione precedente è diverso da zero allora termina, altrimenti il contenuto di EAX viene parcheggiato in V
5. ESI viene sommato a V
6. L'indirizzo risultante viene messo in MAR e viene comandata una lettura da memoria
7. Attesa del completamento della lettura. Intanto il valore di PC viene trasferito in V
8. Il valore letto dalla memoria e messo in MDR viene sommato a V, il tutto viene salvato in Z
9. L'indirizzo risultante corrispondente alla prossima istruzione da eseguire viene messo nel PC.  
Termine dell'istruzione.

### Esercizio 9

Elencare le microistruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto sia relativo al PC

#### JNZ (%EAX)

Soluzione:

- 
1.  $PC_{OUT}, MAR_{IN}, READ, SELECT[4], ADD, Z_{IN}$   
2.  $WMFC, Z_{OUT}, PC_{IN}$   
3.  $MDR_{OUT}, IR_{IN}$
- Fetch
4. if (ZERO) END,  $EAX_{OUT}, MAR_{IN}, READ$   
5.  $WMFC, PC_{OUT}, V_{IN}$   
6.  $MDR_{OUT}, SELECT[V], ADD, Z_{IN}$   
7.  $Z_{OUT}, PC_{IN}, END$

Commenti:

#### Fetch:

- Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
- Viene messo in PC il valore  $PC+4$  e si aspetta il completamento della lettura da memoria
- L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

#### Salto:

- Se il risultato dell'istruzione precedente è zero allora termina, altrimenti il contenuto di EAX viene passato al MAR come un indirizzo e viene comandata una lettura
- Attesa del completamento della lettura; nel frattempo, siccome il BUS non è occupato, il valore corrente del PC viene messo nel registro ausiliario della ALU chiamato V
- Il valore messo in MDR viene sommato a V in quanto il salto è relativo
- L'indirizzo risultante (che punta alla prossima istruzione da eseguire) viene messo nel PC.  
Termine dell'istruzione.

## Esercizio 10

Elencare le micro istruzioni relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%Exx) rappresenti un metodo di indirizzamento indiretto a registro con spiazzamento e che il salto sia assoluto:

**JNZ (%EAX, %EBX)**

Soluzione:

- |   |   |       |
|---|---|-------|
| 1.PC <sub>OUT</sub> , MAR <sub>IN</sub> , READ, SELECT[4], ADD, Z <sub>IN</sub> | } | Fetch |
| 2.WMFC, Z <sub>OUT</sub> , PC <sub>IN</sub>                                     |   |       |
| 3.MDR <sub>OUT</sub> , IR <sub>IN</sub>   |   |       |
- 
- |  |
|--|
| 4.if (ZERO) END, EAX <sub>OUT</sub> , V <sub>IN</sub>  |
| 5.EBX <sub>OUT</sub> , SELECT[V], ADD, Z <sub>IN</sub> |
| 6.Z <sub>OUT</sub> , MAR <sub>IN</sub> , READ          |
| 7.WMFC   |
| 8.MDR <sub>OUT</sub> ,PC <sub>IN</sub> , END           |

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)
  
4. Se il risultato dell'istruzione precedente è zero allora termina, altrimenti il contenuto di EAX viene parcheggiato in V
5. EBX viene sommato a V
6. L'indirizzo risultante viene messo in MAR e viene comandata una lettura da memoria
7. Attesa del completamento della lettura
8. Il valore letto dalla memoria e messo in MDR contenente l'indirizzo della prossima istruzione da eseguire viene messo nel PC. Termine dell'istruzione.

## Esercizio 11

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 solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX, %EBX) rappresenti un metodo di indirizzamento indiretto a registro con spiazzamento e che il salto della procedura sia assoluto:

**CALL (%EAX, %EBX)**

Soluzione:

- |   |   |                               |
|---|---|-------------------------------|
| 1.PC <sub>OUT</sub> , MAR <sub>IN</sub> , READ, SELECT[4], ADD, Z <sub>IN</sub> | } | Fetch                         |
| 2.WMFC, Z <sub>OUT</sub> , PC <sub>IN</sub>                                     |   |                               |
| 3.MDR <sub>OUT</sub> , IR <sub>IN</sub>   | } | Salvataggio di PC nello Stack |
| 4.ESP <sub>OUT</sub> , SELECT[4], SUB, Z <sub>IN</sub>                          |   |                               |
| 5.Z <sub>OUT</sub> , ESP <sub>IN</sub> , MAR <sub>IN</sub>                      |   |                               |
| 6.PC <sub>OUT</sub> , MDR <sub>IN</sub> , WRITE                                 |   |                               |
| 7.WMFC, EAX <sub>OUT</sub> , V <sub>IN</sub>                                    |   |                               |
| 8.EBX <sub>OUT</sub> , SELECT[V], ADD, Z <sub>IN</sub>                          |   |                               |
| 9. Z <sub>OUT</sub> , MAR <sub>IN</sub> , READ                                  |   |                               |
| 10.WMFC   |   |                               |
| 11.MDR <sub>OUT</sub> , PC <sub>IN</sub> , END.                                 |   |                               |

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

### Salvataggio dello stack:

4. Viene decrementato il valore dell'ESP (stack pointer) per puntare ad una cella vuota dello Stack
5. Il nuovo valore viene scritto in ESP e in MAR per la fase di scrittura.
6. Il PC passa nel registro dei dati MDR e viene eseguita l'operazione di scrittura, salvando in memoria (nella nuova cima dello Stack) l'indirizzo attuale di PC.
7. Attesa completamento scrittura. Nel frattempo il valore di EAX viene messo nel registro ausiliario V della ALU (questa seconda operazione non fa parte del salvataggio dello stack, ma è stata scritta sulla stessa riga per "risparmiare" un ciclo, in quanto il BUS in quel momento non era occupato).

### Chiamata a funzione:

8. Il valore di EBX viene sommato a V
9. Il risultato viene passato al registro degli indirizzi MAR per la lettura
10. Attesa del completamento della lettura.
11. Il valore messo in MDR, che rappresenta l'indirizzo reale dell'istruzione da eseguire, viene messo nel PC in quanto il salto è assoluto. Termine istruzione

## Esercizio 12

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 solo BUS, che l'istruzione sia composta da una sola parola, che 4(%EAX) rappresenti un metodo di indirizzamento indiretto a registro con spiazzamento, che 4 sia un valore immediato e che il salto della procedura sia assoluto:

**CALL 4(%EAX)**

Soluzione:

- |   |   |                            |
|---|---|----------------------------|
| 1.PC <sub>OUT</sub> , MAR <sub>IN</sub> , READ, SELECT[4], ADD, Z <sub>IN</sub> | } | Fecth                      |
| 2.WMFC, Z <sub>OUT</sub> , PC <sub>IN</sub>                                     |   |                            |
| 3.MDR <sub>OUT</sub> , IR <sub>IN</sub>   |   |                            |
| 4.ESP <sub>OUT</sub> , SELECT[4], SUB, Z <sub>IN</sub>                          | } | Salvataggio PC nello Stack |
| 5.Z <sub>OUT</sub> , ESP <sub>IN</sub> , MAR <sub>IN</sub>                      |   |                            |
| 6.PC <sub>OUT</sub> , MDR <sub>IN</sub> , WRITE                                 |   |                            |
| 7.WMFC, EAX <sub>OUT</sub> , V <sub>IN</sub>                                    |   |                            |
| 8.OFFSET(IR) <sub>OUT</sub> , SELECT[V], ADD, Z <sub>IN</sub>                   | } |                            |
| 9.Z <sub>OUT</sub> , MAR <sub>IN</sub> , READ                                   |   |                            |
| 10.WMFC   |   |                            |
| 11.MDR <sub>OUT</sub> , PC <sub>IN</sub> , END                                  |   |                            |

Commenti:

### Fetch:

- Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
- Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
- L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

### Salvataggio dello stack:

- Viene decrementato il valore dell'ESP (stack pointer) per puntare ad una cella vuota dello Stack.
- Il nuovo valore viene messo in ESP e in MAR per la fase di scrittura.
- Il PC viene messo nel registro dei dati MDR e viene eseguita l'operazione di scrittura, salvando in memoria (nella nuova cima dello Stack) l'indirizzo attuale di PC.
- Attesa completamento della scrittura. Intanto il valore di EAX viene caricato nel registro ausiliario V (questa seconda operazione non fa parte del salvataggio dello stack, ma è stata scritta sulla stessa riga per "risparmiare" un ciclo, in quanto il BUS in quel momento non era occupato).

### Chiamata a funzione:

- L'offset di IR che contiene il valore 4 (poiché è una costante) viene sommato a V
- Il risultato della somma passa al registro degli indirizzi MAR, in quanto il metodo di indirizzamento è indiretto. Viene comandata una lettura.
- Attesa completamento della lettura
- Il valore letto e messo in MDR passa nel PC in quanto il salto è assoluto. Termine istruzione

### Esercizio 13

Elencare le micro istruzioni relative al caricamento, decodifica ed esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia un solo BUS, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto condizionato sia relativo al PC:

#### JLE (%EAX)

Soluzione:

- 1.PC<sub>OUT</sub>, MAR<sub>IN</sub>, READ, SELECT[4], ADD, Z<sub>IN</sub>  
2.WMFC, Z<sub>OUT</sub>, PC<sub>IN</sub>  
3.MDR<sub>OUT</sub>, IR<sub>IN</sub>
- 4.if (**!LOWER&&!EQUAL**) END, EAX<sub>OUT</sub>, MAR<sub>IN</sub>, READ  
5.WMFC, PC<sub>OUT</sub>, V<sub>IN</sub>  
6.MDR<sub>OUT</sub>, SELECT[V], ADD, Z<sub>IN</sub>  
7.Z<sub>OUT</sub>, PC<sub>IN</sub>, END
- Fetch

Commenti:

#### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR il valore del PC e viene calcolato l'indirizzo dell'istruzione successiva (nell'architettura Intel a 32 bit, per fare ciò bisogna incrementare di 4 il PC)
2. Viene messo in PC il valore PC+4 e si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e viene messa nel registro delle istruzioni (IR)

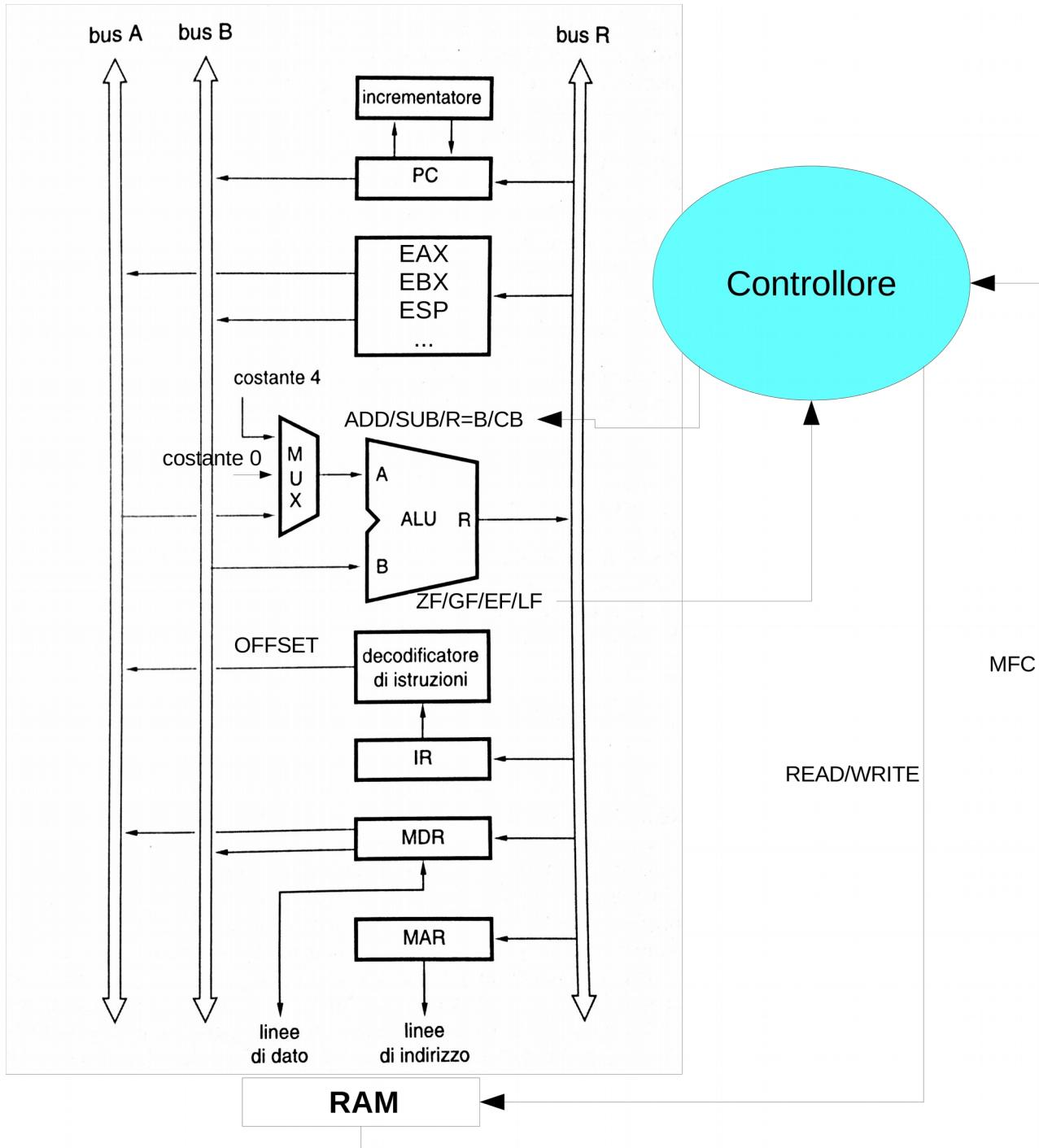
#### Salto:

4. Se il risultato dell'istruzione CMP precedente non corrisponde al valore dell'espressione logica dell'istruzione di salto allora termina, altrimenti il contenuto di EAX viene passato al MAR come un indirizzo e viene comandata una lettura
5. Attesa della lettura; nel frattempo, siccome il BUS non è occupato, il valore corrente del PC viene messo nel registro ausiliario della ALU chiamato V
6. L'indirizzo prodotto da MDR viene sommato a V in quanto il salto è relativo
7. L'indirizzo risultante (che punta alla prossima istruzione da eseguire) viene messo nel PC.  
Termine dell'istruzione.

NOTA: si noti nel test alla riga 4:

- l'uso combinato di vari segnali di FLAG generati dall'ALU nell'istruzione CMP precedente (cioè LF e EF)
- l'uso del teorema di De Morgan nella costruzione dell'espressione logica.

## Architettura a 3 bus



## Esercizio 14

Elencare le micro istruzioni relative alla completa esecuzione della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **TRE BUS**, che l'istruzione sia composta da una sola parola e che (%EBX) rappresenti un metodo di indirizzamento indiretto a registro:

**ADDL (%EBX), %EAX**

Soluzione:

1.  $\text{PC}_{\text{OUT B}}$ , **R=B**,  $\text{MAR}_{\text{IN R}}$ , READ, **INC PC**
  2. WMFC
  3.  $\text{MDR}_{\text{OUT B}}$ ,  $\text{R=B}$ ,  $\text{IR}_{\text{IN R}}$
- }
- Fetch
- 
4.  $\text{EBX}_{\text{OUT B}}$ ,  $\text{R=B}$ ,  $\text{MAR}_{\text{IN R}}$ , READ
  5. WMFC
  6. **SELECT[BUS\_A]**,  $\text{MDR}_{\text{OUT A}}$ , **EAX\_{OUT B}**,  $\text{EAX}_{\text{IN R}}$ , ADD, END

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR (attraverso il bus R) il valore del PC (messo sul bus B) e collegando bus B ed R abilitando la corrispondente funzionalità dell'ALU; viene anche calcolato l'indirizzo dell'istruzione successiva utilizzando il circuito incrementatore apposito
2. Si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e attraverso il bus B, collegato al bus R, viene messa nel registro delle istruzioni (IR)

### Execute:

4. Il valore di EBX viene messo in MAR attraverso il bus B collegato al bus R abilitando la corrispondente funzionalità dell'ALU; viene comandata una lettura
5. Attesa del completamento della lettura
6. Il risultato della lettura in MDR viene messo sul bus A che viene collegato ad un ingresso della ALU; il valore di EAX viene messo sul bus B sempre collegato al secondo ingresso della ALU; viene comandata una somma mettendo il risultato in EAX attraverso il bus R (l'aggiornamento di EAX avviene alla fine del ciclo di clock). Termine istruzione.

NOTA: Si confronti questa soluzione con quella dell'Esercizio 3. A parità di istruzione Assembly l'architettura a tre bus non riduce la fase di fetch, la cui lunghezza è dominata dall'accesso in memoria, ma solo la fase di execute quando si è in presenza di operazioni che richiedono il passaggio di due valori sul bus.

## Esercizio 15

Elencare le micro istruzioni (insieme dei segnali di controllo) relative alla completa esecuzione (caricamento, decodifica, esecuzione) della seguente istruzione assembler (Intel 80386 AT&T), assumendo che la CPU abbia **TRE BUS**, che l'istruzione sia composta da una sola parola, che (%EAX) rappresenti un metodo di indirizzamento indiretto a registro e che il salto sia di tipo relativo al PC:

**CALL (%EAX)**

Soluzione:

- |  |   |                            |
|--|---|----------------------------|
| 1.PC <sub>OUT_B</sub> , R=B, MAR <sub>IN_R</sub> , READ, INC_PC                                | } | Fetch                      |
| 2.WMFC   |   |                            |
| 3.MDR <sub>OUT_B</sub> , R=B, IR <sub>IN_R</sub>   |   |                            |
| 4.ESP <sub>OUT_A</sub> , SELECT[4], SUB, ESP <sub>IN_R</sub> , MAR <sub>IN_R</sub>             | } | Salvataggio PC nello stack |
| 5.PC <sub>OUT_B</sub> , R=B, MDR <sub>IN_R</sub> , WRITE                                       |   |                            |
| 6.WMFC   |   |                            |
| 7.EAX <sub>OUT_B</sub> , R=B, MAR <sub>IN_R</sub> , READ                                       |   |                            |
| 8.WMFC   |   |                            |
| 9. <b>SELECT[BUS_A], MDR<sub>OUT_A</sub>, PC<sub>OUT_B</sub>, ADD, PC<sub>IN_R</sub>, END.</b> |   |                            |

Commenti:

### Fetch:

1. Viene richiesta la lettura dalla memoria della prossima istruzione mettendo in MAR (attraverso il bus R) il valore del PC (messo sul bus B) e collegando bus B ed R abilitando la corrispondente funzionalità dell'ALU; viene anche calcolato l'indirizzo dell'istruzione successiva utilizzando il circuito incrementatore apposito
2. Si aspetta il completamento della lettura da memoria
3. L'istruzione è pronta in MDR e attraverso il bus B, collegato al bus R, viene messa nel registro delle istruzioni (IR)

### Salvataggio dello stack:

4. Viene decrementato il valore dell'ESP (stack pointer) per puntare alla nuova cima dello Stack e il nuovo valore viene caricato in ESP e nel MAR per la fase di scrittura.
5. Il PC passa nel registro dei dati MDR collegando bus B ed R attraverso la corrispondente funzionalità dell'ALU; viene comandata l'operazione di scrittura, salvando in memoria (nella cima dello Stack) l'indirizzo attuale di PC.
6. Attesa completamento scrittura.

### Chiamata a funzione:

7. Il valore di EAX viene passato al MAR collegando bus B ed R attraverso la corrispondente funzionalità dell'ALU. Viene comandata una lettura.
8. Attesa completamento lettura.
9. Da MDR esce sul bus A il valore letto; il bus A viene collegato in ingresso alla ALU; tale valore è sommato a PC (il bus B è sempre collegato alla ALU) e il risultato è messo in PC alla fine del ciclo di clock. Termine dell'istruzione.



UNIVERSITÀ  
di VERONA

Dipartimento  
di INFORMATICA

Prof. Franco Fummi  
Prof. Luca Geretti  
Prof. Davide Quaglia

# ARCHITETTURA DEGLI ELABORATORI

Eserciziario su Pipeline

Creato a partire da materiale di:

*Enrico Giordano*

*Serena Cavaletti*

*Katia Cracco*

*Roxana Belciug*

## Regole ed accorgimenti

Questo tipo di esercizi consiste nell'elencare le varie fasi di Fetch, Decode, Execute, Memory e Store di una istruzione Assembly in base alle situazioni proposte e alle dipendenze con altre istruzioni precedenti. Nella migliore delle ipotesi, l'istruzione successiva parte con la fase di Fetch durante la fase di Decode dell'istruzione precedente. In questi esercizi, se non scritto diversamente nel testo, si assume che:

- la fase E e S avvengono ciascuna in un solo ciclo di clock;
- istruzioni e dati sono presenti in cache in modo che l'accesso avvenga in un solo ciclo di clock (in tal caso istruzioni e dati sono anche in cache separate per evitare conflitti di risorse); in tal caso la fase M avviene in un solo ciclo di clock.

Graficamente si ottiene la seguente griglia dove la lista delle istruzioni è un dato dell'esercizio mentre vi si chiede di riempire la parte arancione:

CLOCK	1	2	3	4	5	6
Istruzione 1	F	D	E	M	S	
Istruzione 2		F	D	E	M	S

Qualora invece **gli input di un'istruzione dipendono dagli output di una di quelle precedenti**, la sua fase di Fetch inizia all'istante della fase di Decode della funzione precedente ma la sua fase di Decode perdura per tutto il tempo degli altri stadi dell'istruzione precedente (aggiunta delle caselle con "D" in grassetto) compresa una fase aggiuntiva dopo la fase di Store dell'istruzione precedente (caselle arancioni). La fase di Execute inizia dopo le fasi aggiuntive di Decode e quindi **almeno un periodo di clock dopo la terminazione della fase S dell'istruzione da cui dipende**. Infine l'istruzione successiva a quella con le "D" ripetute, sebbene indipendente da questa, deve necessariamente prolungare la fase di Fetch (aggiunta delle caselle con "F" in grassetto) finché l'unità di Decode non torna libera (caselle verdi).

CLOCK	1	2	3	4	5	6	7	8	9
Istruzione 1	F	D	E	M	D	S			
Istruzione 2		F	D	D	D	D	E	M	S
Istruzione 3			F	F	F	F	D	E	M

**NOTA:** non esiste dipendenza tra due istruzioni che scrivono nello stesso registro di destinazione perché non c'è modo per la prima scrittura di terminare dopo la seconda.

Qualora invece ci siano 3 istruzioni e l'ultima dipendesse dalla prima ma non dalla seconda, gli stati “D” ripetuti descritti precedentemente devono riferirsi alla prima istruzione (cioè quella da cui dipende l'istruzione considerata).

CLOCK	1	2	3	4	5	6	7	8	9
<b>Istruzione 1</b>	F	D	E	M	S				
Istruzione 2		F	D	E	M	S			
<b>Istruzione 3</b>			F	D	D	D	E	M	S

Le istruzioni di salto condizionato dipendono sempre dall'istruzione precedente che aggiorna i FLAG nel registro Program Status Word nella fase S. Inoltre nel caso di istruzioni di salto condizionato, nel testo dell'esercizio deve essere specificato se il salto avviene oppure no oppure a fianco dell'istruzione di salto saranno presenti i commenti #yes e #no per indicare questa informazione.

**Attenzione:** nello schema grafico che si richiede come soluzione occorre prestare attenzione che, leggendo per colonna, non ci sia mai la stessa lettera ripetuta perché significherebbe che lo stesso stadio della pipeline è occupato da più di una istruzione, cosa chiaramente impossibile.

## Esercizio 1

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1.

*Soluzione*

ISTRUZIONE/CLOCK	1	2	3	4	5	6	7	8	9	10	11
addl %eax, %ebx	F	D	E	M	S						
movl \$4, %ecx			F	D	M	S					
subl %ebx, %ecx			F	D	D	D	E	M	S		
movl \$4, %edx				F	F	F	D	E	M	S	

Le prime due istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La terza istruzione dipende dalla seconda perché il registro ECX è output nella seconda e input nella terza; dunque è necessario che l'istruzione subl prolunghi la fase di decode fino all'ultimo ciclo di movl più uno. Infine, la quarta istruzione non dipende dalle altre e quindi il Fetch può partire subito anche se però deve essere poi prolungato a causa del protrarsi della fase di Decode dell'istruzione precedente dando vita al cosiddetto effetto “bolla” (caselle arancioni). Si noti che la terza istruzione dipende anche dalla prima per via del registro EBX ma siccome abbiamo risolto già la dipendenza dalla seconda, tutte le dipendenze “più lontane” sono automaticamente risolte.

## Esercizio 2

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1.

*Soluzione*

ISTRUZIONE/CLOCK	1	2	3	4	5	6	7	8	9	10	11
ciclo: addl %eax, %ebx	F	D	E	M	S		<b>F</b>	<b>D</b>	<b>D</b>	E	M
movl \$4, %ecx		F	D	E	M	S		<b>F</b>	<b>F</b>	D	E
subl %eax, %edx			F	D	E	M	S			<b>F</b>	D
movl \$6, %ebx				F	D	E	M	<b>S</b>			<b>F</b>
jmp ciclo					F	<b>D</b>	E	M	S		

Le prime quattro istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La quinta istruzione è un salto incondizionato e quindi non dipende dalle altre e può essere eseguita normalmente. Il Fetch dell'istruzione di destinazione del salto parte al settimo ciclo di clock, dopo la fase di Decode dell'istruzione di salto a causa di un'ottimizzazione della pipeline che riguarda le istruzioni di salto (caselle arancioni). Le istruzioni riprendono dall'etichetta **ciclo** (le fasi sono riportate in grassetto). Si noti che la prima istruzione ha una dipendenza **all'indietro** dalla quarta istruzione (cioè due istruzioni precedenti al ricominciare del ciclo) per via del registro EBX e quindi deve prolungare la fase D di un ciclo per attendere la scrittura del risultato della quarta istruzione (caselle verdi).

### Esercizio 3

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1. Si ipotizzi che il salto avvenga. Si ignorino le tecniche del Delay Slot e della Branch Prediction.

#### Soluzione

ISTRUZIONE/CLOCK	1	2	3	4	5	6	7	8	9	10	11	12	13
inizio: inc %ebx	F	D	E	M	S				<b>F</b>	D	E	M	S
movl %ecx, %edx		F	D	E	M	S				<b>F</b>	D	E	M
cmpl %eax, 0x86FF			F	D	E	M	S				<b>F</b>	D	E
jne inizio #yes				F	D	D	D	<b>D</b>	E	M	S	<b>F</b>	D
movl %ecx, %edx					<b>F</b>	<b>F</b>	<b>F</b>	<b>F</b>					<b>F</b>

Le prime tre istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La quarta istruzione è un salto condizionato e quindi dipende dalla terza perché il salto deve attendere che si sappia se la condizione è vera o falsa, affinché si possa capire se il salto deve avvenire oppure no. Di conseguenza l'istruzione `jne` deve prolungare la fase di decode fino all'ultimo ciclo di `cmpl` più uno. Si noti che l'istruzione successiva a quella di salto inizia subito la fase di Fetch che però viene annullata (caselle arancioni) non appena la fase D rende palese che il salto viene eseguito e che quindi occorre fare il Fetch dell'istruzione di destinazione del salto (caselle verdi). Successivamente si rieseguono le istruzioni del ciclo (le fasi sono riportate in grassetto).

#### Esercizio 4

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1 e che il salto non avvenga.

#### Soluzione

CLOCK/ISTRUZIONE	1	2	3	4	5	6	7	8	9	10	11
START: subl %eax, %ebx	F	D	E	M	S						
jz START #no		F	D	D	D	D	E	M	S		
subl %ebx, %ecx			F	F	F	F	D	E	M	S	
movl %edx, %eax							F	D	E	M	S

La prima istruzione viene eseguita normalmente. La seconda ha una dipendenza con la prima perché il salto condizionato deve conoscere il valore dei FLAG scritti dalla sottrazione in un registro apposito nella fase S. Poiché il salto non avviene, posso eseguire tranquillamente la terza istruzione ma occorre comunque aspettare che l'unità di Decode sia libera. Si noti che la terza istruzione dipende dalla prima per via del registro EBX, ma non è necessaria alcuna attesa ulteriore poiché la prima istruzione finisce prima che la terza cominci la fase di Decode. Infine, la quarta istruzione non ha dipendenze.

**NOTA:** immediatamente prima di un'istruzione di salto condizionato c'è sempre un'istruzione cmp, oppure sub, oppure add.

### Esercizio 5

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1 e che il salto non avvenga.

#### Soluzione

CLOCK/ISTRUZIONE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ciclo: addl %eax, %ebx	F	D	E	M	S										
movl %edx, %ecx		F	D	E	M	S									
subl %ebx, %ecx			F	D	D	D	D	E	M	S					
jz ciclo #no				F	F	F	F	D	D	D	D	E	M	S	
movl %ecx, %edx								F	F	F	F	D	E	M	S

Le prime due istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La terza dipende sia dalla prima (per via di EBX) che dalla seconda (per via di ECX); in questi casi prevale la dipendenza più vicina e quindi è necessario prolungare la fase di decode di subl fino all'ultimo ciclo di movl più uno. La quarta istruzione è un salto condizionato e quindi deve aspettare che l'istruzione precedente termini e quindi prolunga la fase di decode del salto fino all'ultimo ciclo di subl più uno. Infine, dato che il salto non avviene, la quinta viene eseguita normalmente ma occorre comunque aspettare che l'unità di Decode sia libera. Si noti che la quinta istruzione dipende dalla terza ma non occorre attendere ulteriormente perché la terza finisce prima che la quinta cominci la fase di Decode. Si notino anche i due effetti "bolla" successivi (caselle arancioni e verdi).

### Esercizio 6

Si consideri una CPU con una pipeline a 4 stadi (F, D, E, W). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi che la pipeline sia vuota al tempo 1 e che jz faccia riferimento all'istruzione sub1.

#### Soluzione

CLOCK/ISTRUZIONE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
ciclo: addl %eax, %ebx	F	D	E	W													
movl %ebx, %ecx		F	D	D	D	E	W										
subl %eax, %ecx			F	F	F	D	D	D	E	W							
jz ciclo #no						F	F	F	D	D	D	E	W				

Si notino i leggeri cambiamenti dovuti al diverso tipo di pipeline: M e S sono sostituiti dall'unica fase W, la sequenza di D (per la dipendenza) e la sequenza di F (per l'effetto bolla) sono più corte di un periodo. La prima istruzione viene eseguita normalmente. La seconda ha una dipendenza con la prima, dunque è necessario che l'istruzione movl prolunghi la fase di decode fino all'ultimo ciclo di addl più uno. La terza istruzione ha una dipendenza con la seconda, dunque è necessario che anche l'istruzione subl prolunghi la fase di decode fino all'ultimo ciclo di movl più uno. Infine, il salto fa riferimento all'istruzione subl, quindi la quarta istruzione dipende dalla terza e si deve prolungare con uguale criterio la fase di decode.

### Esercizio 7

Si consideri una CPU con una pipeline a 5 stadi (F, D, E, M, S). Si riporti nel seguente diagramma, per ogni istruzione, lo stadio della pipeline coinvolto in ogni istante di clock. Si ipotizzi la pipeline vuota al tempo 1 e si facciano le opportune ipotesi sul salto condizionale.

#### Soluzione

Quando si richiedono entrambi i casi è meglio partire da quello in cui il salto non avviene in quanto la sua soluzione è contenuta anche nell'altro caso.

1) Suppongo che il salto non avvenga.

CLOCK/ISTRUZIONE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
init: movl %ecx, %edx	F	D	E	M	S										
addl \$4, %ebx			F	D	M	S									
cmpl 0x319FA, %ebx			F	D	D	D	D	E	M	S					
jnz init #no				F	F	F	F	D	D	D	D	E	M	S	
addl %eax, %ecx								F	F	F	F	D	E	M	S

Le prime due istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La terza dipende dalla seconda per via di EBX, quindi è necessario prolungare la fase di decode di cmpl fino all'ultimo ciclo di addl più uno. La quarta istruzione dipende dalla terza e quindi anche qui si deve prolungare allo stesso modo la fase di decode. Infine, poiché ipotizzo che il salto non avvenga, posso passare alla quinta istruzione, la quale non ha dipendenze e viene eseguita normalmente appena l'unità di Decode si libera.

2) Suppongo che il salto avvenga.

CLOCK/ISTRUZIONE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
init: movl %ecx, %edx	F	D	E	M	S							F	D	E	M	S		
addl \$4, %ebx			F	D	E	M	S						F	D	E	M	S	
cmpl 0x319FA, %ebx			F	D	D	D	D	E	M	S			F	D	D	D	D	
jnz init #yes				F	F	F	F	D	D	D	D	D	E	M	S	F	F	
addl %eax, %ecx								F	F	F	F							

Le prime due istruzioni vengono eseguite normalmente, poiché non ci sono dipendenze. La terza dipende dalla seconda per via di EBX, quindi è necessario prolungare la fase di Decode di cmpl fino all'ultimo ciclo di addl più uno. La quarta istruzione dipende dalla terza e quindi anche qui si deve prolungare allo stesso modo la fase di decode. Infine, poiché ipotizzo che il salto avvenga, la quinta istruzione non viene eseguita, e ricominciano le istruzioni dall'etichetta init.

## Domande di teoria sulle pipeline

### Domanda 1

Una CPU con una pipeline a 2 stadi viene sostituita con una CPU con una pipeline a 4 stadi. Se il tempo totale di esecuzione di una singola istruzione è rimasto invariato, qual è il minimo ed il massimo incremento delle prestazioni che si può attendere?

Soluzione:

Una pipeline ad  $n$  stadi è in grado di aumentare le prestazioni idealmente di  $n$  volte e di raggiungere lo scopo di riuscire ad eseguire una istruzione a ciclo di clock. Quindi se da una pipeline a due stadi si passa ad una a quattro stadi, essa può aumentare idealmente le prestazioni di due volte. Il minimo incremento delle prestazioni che si può ottenere è di una volta.

L'aumento degli stadi nella pipeline aumenta anche la probabilità di stallo della CPU. In condizione di istruzione di salto, nel caso in cui l'algoritmo di predizione dei salti fallisca, dato che più istruzioni vengono eseguite in parallelo, si annulla il vantaggio della pipeline con la perdita delle istruzioni fino a quel momento erroneamente elaborate, è quindi possibile che non si ottenga alcun incremento in quanto aumentando gli stadi della pipeline aumentano le condizioni di criticità, quindi si potrebbero formare delle "bolle" che manderebbero in stallo la CPU.

### Domanda 2

Descrivere il meccanismo e l'utilità della *predizione dei salti*.

Soluzione:

I salti interrompono il flusso della pipeline e per minimizzare il numero di interruzioni è necessario uno schema di *branch prediction*.

I salti sono molto frequenti, in media uno ogni 6 istruzioni; nei processori superscalari che eseguono anche 4 istruzioni per ciclo di clock, la *predizione dei salti* è molto importante.

Molti schemi di *branch prediction* hanno un algoritmo che tiene traccia del comportamento di un certo salto l'ultima volta che è stato eseguito.

Infatti se è stato eseguito un salto nella precedente istruzione, si farà l'ipotesi che il salto avvenga nuovamente.

La pipeline ora contiene un'istruzione di salto condizionato e altre istruzioni successive, ma non si sa ancora se tali istruzioni serviranno o meno. Se il salto verrà eseguito, l'esecuzione del programma potrà continuare tranquillamente, se invece non verrà eseguito, tutte le istruzioni caricate nella pipeline dovranno essere eliminate.

La *predizione dei salti* si divide in due tipi: statica e dinamica.

Predizione STATICÀ (*compile-time*): è realizzata dal compilatore.

La parola di codice operativo dell'istruzione indica all'unità di prelievo delle istruzioni se deve prevedere che il salto sia effettuato o meno. Il risultato della predizione è lo stesso ogni volta che si incontra una data istruzione di salto.

**Predizione DINAMICA(*run-time*):** L'hardware del processore determina la probabilità che un salto venga eseguito ogni volta che si incontra una certa istruzione. Ciò può essere ottenuto mantenendo traccia del risultato della decisione di salto l'ultima volta che tale istruzione è stata eseguita e ipotizzando che la decisione nella presente istanza possa essere la stessa.

### **Domanda 3**

Elencare le ottimizzazioni che devono essere eseguite sull'architettura di un calcolatore per raggiungere l'obiettivo di avere, per la maggioranza delle istruzioni, CPI = 1

Soluzione:

Ottimizzazioni per ottenere CPI<sub>medio</sub> = 1

- Architettura a tre bus
- Incremento PC separato
- Cache
- No operazioni su memoria
- Cache dati separata dalla cache del programma
- ALU in cui tutte le operazioni vengono eseguite in un ciclo di clock
- Pipeline

### **Domanda 4**

Quali sono le motivazioni che spingono i microprocessori moderni ad essere dotati di una pipeline.

Soluzione:

Lo scopo di adottare le pipeline nei processori moderni è quello di ridurre il più possibile il CPI<sub>medio</sub> e di ottenere CPI<sub>medio</sub> = 1.

Le pipeline sono delle strutture interne alla CPU; esse sono dotate di più stadi (come ad esempio Fetch, Decode, Execute e WriteBack) e permettono di ridurre il CPI aumentando l'IPS.

Per avere dei netti miglioramenti in quantità di tempo è indispensabile cercare di fare entrare la CPU in stallo il meno possibile, di tenerla quindi sempre a regime.

Per fare questo, un meccanismo indispensabile è la *predizione dei salti (branch prediction)*.

Teoricamente se una pipeline ha  $n$  stadi, aumenta l'efficienza di  $n$  volte.

In realtà non è così poiché aumenta anche la probabilità di dipendenza fra i dati, motivo per cui i processori moderni cercano di avere diversi stadi ma in numero non eccessivo ed un sistema di *branch prediction* il più affidabile possibile.