Progettazione a livello di microarchitettura

A cura di:
Luca Breveglieri Giuseppe Pozzi Donatella Sciuto

DEI, PoliMI, Milano

luca.breveglieri,giuseppe.pozzi,donatella.sciuto@polimi.it

- versione dell'11 aprile 2003-

Progettazione a livello di microarchitettura

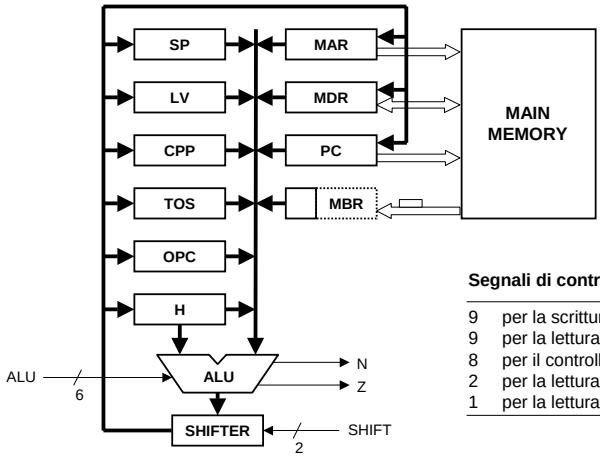
Ottimizzazione delle prestazioni

- **Bibliografia**:

  Tanenbaum A. S., Goodman J. R,
  "**Architettura dei computer** - Un approccio strutturato",
  Prentice Hall International, 2000 - paragrafo 4.4

- **Sommario**:
  - Problematiche di progettazione: costi vs prestazioni
  - Architettura a tre bus.
  - Pipelining.

Microarchitettura Mic-1



Segnali di controllo

9	per la scrittura dal BUS C
9	per la lettura sul BUS B (codificati a 4 bit)
8	per il controllo di ALU e SHIFTER
2	per la lettura/scrittura della memoria
1	per la lettura tramite PC/MBR

Costi vs prestazioni

- Definizione della microarchitettura dipende da un compromesso tra costi e prestazioni
- Incremento della prestazioni determinato da
  - Tecnologia
  - Architettura
- Costi difficili da definire (progetto, dimensioni, produzione, resa....)

Incremento delle prestazioni

- Ipotesi: definite tecnologia e ISA
- Approcci microarchitetturali per migliorare prestazioni:
  - Riduzione del numero di cicli di clock per l'esecuzione di ogni istruzione
  - Riduzione della durata del ciclo di clock semplificando l'organizzazione dell'architettura
  - Sovrapposizione dell'esecuzione delle istruzioni (pipelining)

Esecuzione di un'istruzione

- L'esecuzione di ogni istruzione richiede:
  - Lettura dell'istruzione dalla memoria
  - Decodifica dei campi dell'istruzione
- In termini di microistruzioni:
  - PC passa attraverso la ALU e viene incrementato
  - PC viene usato per leggere il byte seguente nel programma
  - PC ed ALU vengono usati per leggere gli eventuali operandi

Utilizzo delle risorse

- La ALU è utilizzata per diversi cicli di clock:
  - Per operazioni non legate alla esecuzione di una istruzione ma piuttosto al suo caricamento
  - Come collegamento per copiare un valore da un registro ad un altro, senza svolgere alcuna operazione
- Ciò comporta un notevole spreco di tempo

Riduzione del numero di cicli di clock

- Replicazione di moduli hardware per evitare conflitti di risorse
  - Soluzione 1: Inserire un sommatore aggiuntivo per incrementare il PC
    - Costoso in termini di area sul silicio
    - non si verifica un vantaggio sostanziale: incremento del PC avviene durante la fase di lettura dell'istruzione e quindi la ALU non è impegnata
  - Soluzione 2: Aggiungere percorsi addizionali
    - Ad esempio percorsi da TOS a MDR o vice versa
    - Aumento del numero dei segnali di controllo
    - Microistruzioni e microcodice più complessi
  - Soluzione 3: Aggiungere un bus addizionale

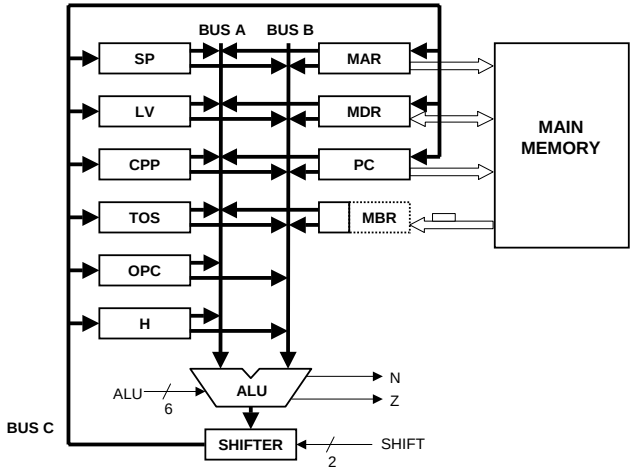
Architettura a tre bus

- Le operazioni aritmetiche richiedono:
  - Un primo ciclo di clock per caricare un operando nel registro H
  - Durante tale ciclo la ALU viene usata solo come connessione
  - Un secondo ciclo di clock per eseguire l'operazione
- Un possibile miglioramento consiste nel rendere possibile la connessione diretta di tutti i registri con entrambi gli ingressi dati della ALU
- Questa soluzione richiede l'aggiunta di un ulteriore bus

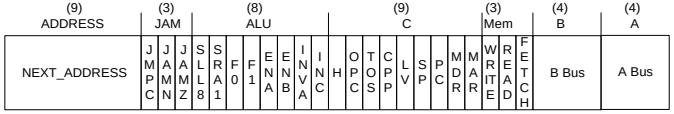
Architettura a tre bus

- L'architettura risultante è pertanto detta a tre bus:
  - BUS A: Lettura dei registri sull'operando A della ALU
  - BUS B: Lettura dei registri sull'operando B della ALU
  - BUS C: Scrittura dell'uscita della ALU (e Shifter) sui registri

Architettura a tre bus (Mic-1)



Formato microistruzioni (3 bus)



B bus, A bus registers: 0=MDR, 1=PC, 2=MBR, 3=MBRU, 4=SP, 5=LV, 6=CPP, 7=TOS, 8=OPC, 9-15 unused

Architettura a tre bus

- Notevole risparmio in termini di cicli di clock
- Esempio: **MDR=TOS+SP**
- Architettura a due bus:
  - T=1:**Read SP, Write H**
  - T=2:**Read TOS, ALU=ADD, Write MDR**
- Architettura a tre bus:
  - T=1:**ReadA SP, ReadB TOS, ALU=ADD, Write MDR**
- In molte istruzioni la ALU viene semplicemente usata per copiare il valore da un registro al registro temporaneo **H**

Riduzione del numero di cicli di clock

- Riduzione del numero di cicli necessari a leggere le istruzioni (fase di fetch)
  - Necessaria la sovrapposizione dell'esecuzione delle istruzioni
- Separare i circuiti di fetch dal data path principale utilizzando una unità funzionale specializzata
  - Porta della memoria di 8 bit e i registri MBR e PC per leggere in modo indipendente il codice operativo o l'operando seguente

Prefetching

- Introduzione di una unità dedicata per la fase di fetch
  - In grado di gestire il program counter autonomamente
  - In grado di leggere alcuni byte dalla memoria prima che siano effettivamente necessari
  - Non molto costosa in termini di area sul silicio
- L'unità aggiuntiva è detta *Instruction Fetch Unit* o IFU

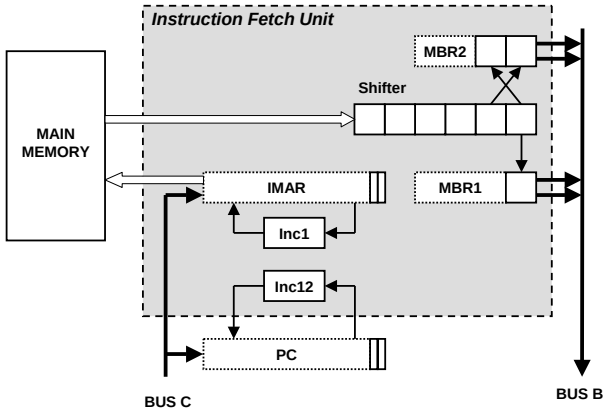
Instruction Fetch Unit

L'unità svolge le seguenti operazioni:

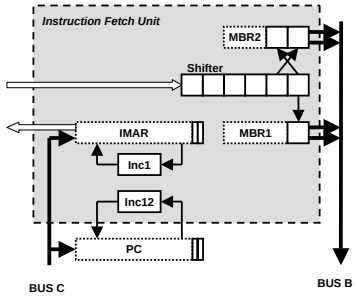
- Gestione del program counter
- Lettura di byte dalla memoria
- Assemblaggio di operandi a 8 o 16 bit

- Per fare ciò la IFU:
  - Legge i due byte successivi
  - Rende disponibile il byte successivo (8 bit)
  - Rende disponibili due byte successivi (16 bit)
- L'unità principale di esecuzione richiede i byte necessari

Instruction Fetch Unit



Prefetching



MBR1

- Mantiene l'ultimo byte letto

MBR2

- Mantiene gli ultimi 2 byte letti

Shifter

- Mantiene 6 byte letti dalla memoria

IMAR

- Registro dedicato per il prefetch
- Contiene l'indirizzo da cui leggere

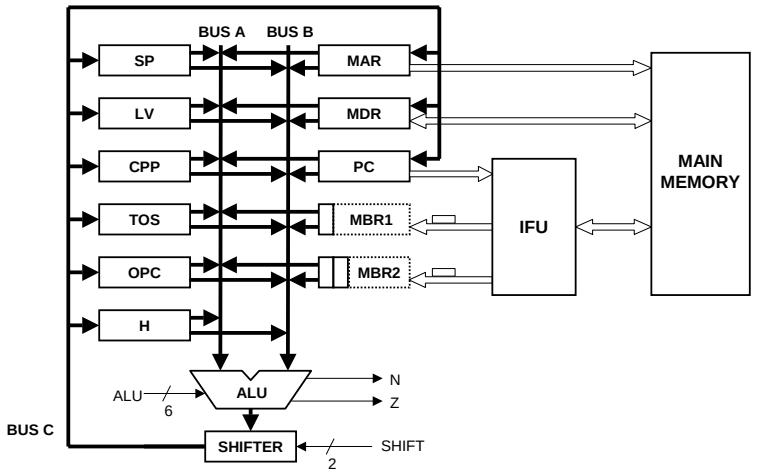
Inc12

- Incrementa di 1 o 2

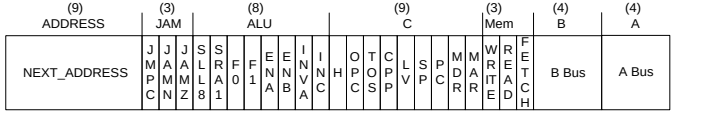
Inc1

- Incrementa di 1

Prefetching (Mic-2)



Formato microistruzioni Mic-2



B bus, A bus registers: 0=MDR, 1=PC, 2=MBR1, 3=MBR1U, 4=SP, 5=LV, 6=CPP, 7=TOS, 8=OPC, 9=MBR2, A=MBR2U

Lettura dalla memoria

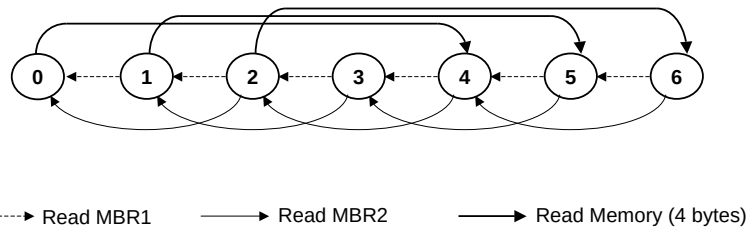
- IFU responsabile per la lettura del flusso di byte
  - Utilizza la porta convenzionale da 4 byte
  - Legge parole intere di 4 byte prima del necessario
  - Carica i byte consecutivi nel registro di shift

Funzione dello shifter

- Il registro a scorrimento della IFU chiamato Shifter:
  - Mantiene una coda di 6 byte letti da memoria ad indirizzi consecutivi
  - Il byte più vecchio alimenta MBR1
  - I due byte più vecchi, scambiati di posizione, alimentano MBR2
  - Ogni volta che MBR1 viene letto, Shift scorre di una posizione
  - Ogni volta che MBR2 viene letto, Shift scorre di due posizioni
  - Quando in Shift si hanno 4 byte vuoti, una nuova parola viene letta

Controllo della IFU

- Questo comportamento è regolato da una macchina a stati finiti a 7 stati
  - Ogni stato rappresenta il # di byte che si trovano in quel momento nel lo shifter



Program Counter

- Unità principale di esecuzione scrive sul PC solo quando si deve cambiare la sequenza di esecuzione, salto o chiamata a procedura (INVOKEVIRTUAL O IRETURN)
- IFU deve mantenere aggiornato il PC
- PC contiene sempre l'indirizzo del primo byte non consumato
- All'inizio di ogni istruzione MBR contiene l'indirizzo del codice operativo di quella istruzione

Prefetching e microprogrammazione

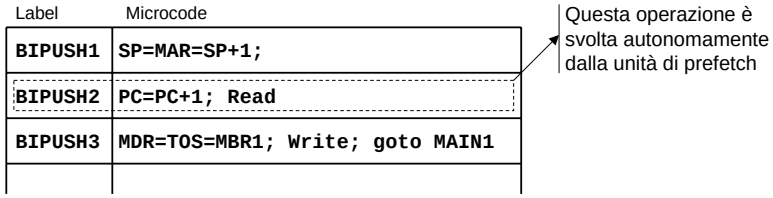
- Dal punto di vista della microprogrammazione la IFU:
  - Legge l'istruzione successiva in **MBR1**
  - Prepara gli eventuali operandi in **MBR2**
- Quindi:
  - Sparisce il microcodice per il fetch delle istruzoni
  - Sparisce il salto a **MAIN1** alla fine di ogni microprocedura
  - Il fetch e decode dell'istruzione successiva viene eseguito semplicemente dalla microistruzione **goto (MBR1)** alla fine di ogni microprocedura

Prefetching

- Una tale architettura comporta alcuni controlli aggiuntivi:
  - Il registro **MBR2** deve contenere due byte quando viene letto
  - Lo shifter deve attendere la fine della lettura dalla memoria
- Una IFU reale è più complessa di quella mostrata

Prefetching: Esempio

- Istruzione: **BIPUSH**
- Significato:
  - Legge in **MBR** il byte successivo all'opcode **BIPUSH**
  - Estende il segno
  - Mette il byte in **MBR** sulla cima dello stack
- Microcodice:



Conclusioni

- L'introduzione del bus A e della unità di prefetching ha migliorato le prestazioni dell'architettura per due motivi:
  - Non è necessario caricare un operando alla volta per usare la ALU
  - Non è necessario usare la ALU per caricare istruzioni e operandi
- Questo comporta una riduzione del numero medio di microistruzioni per ogni istruzione a livello ISA
- Si ha quindi un aumento del parallelismo

Incremento delle prestazioni

- Per migliorare ulteriormente le prestazioni si può:
  - Ridurre il periodo di clock
  - Aumentare ulteriormente il parallelismo oltre a fetch ed esecuzione

⇨ **Introduzione di un *pipelining più profondo***

Riduzione del periodo di clock

- L'esecuzione di una microistruzione comporta:
  - Lettura dei registri sui bus
  - Elaborazione dei dati da parte della ALU e dello SHIFTER
  - Scrittura dei risultati nei registri
- Le tre operazioni devono svolgersi in un ciclo di clock
  - ➢ Definisce la durata del periodo di clock

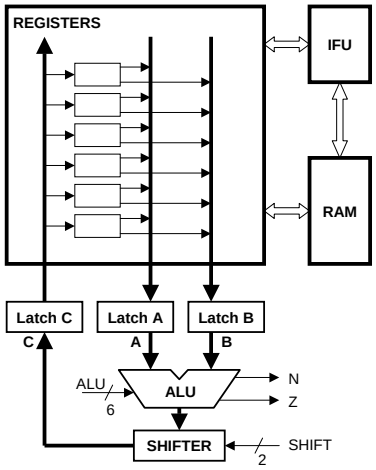
Riduzione del periodo di clock

- Per ridurre il periodo di clock è possibile:
  - Utilizzare una tecnologia (transistor) più veloci
  - Eseguire operazioni più semplici in ogni ciclo di clock:
    - Ciclo *i*: Lettura dei registri sui bus
    - Ciclo *i+1*: Elaborazione dati da parte della ALU e dello SHIFTER
    - Ciclo *i+2*: Scrittura dei risultati nei registri
- La prima soluzione, anche se comunemente utilizzata, non riguarda il progetto logico o architetturale

Architettura del data path pipeline Mic-3

Nuova architettura a tre bus:

- È simile alla precedente
- Utilizza tre latch per isolare le diverse sezioni del data path
- I tre latch sono scritti ad ogni ciclo di clock



Introduzione dei latch

- Necessari tre cicli di clock per utilizzare il data path
  - Caricamento dei latch A e B dai registri
  - Funzionamento dell'ALU e shifter per scrivere il latch C
  - Memorizzazione del contenuto del latch C nei registri

Ma .....

Introduzione dei latch

- Ciclo di clock determinato dal ritardo massimo tra lettura e scrittura nei registri
  - Ritardo massimo ora è più breve ⇨ frequenza del clock più elevata
    - Esempio: se dividendo il data path in tre sezioni il ciclo di clock diventa 1/3 del precedente la frequenza di clock triplica! (APPROSSIMAZIONE)
- Si possono utilizzare tutti i componenti del data path durante un ciclo di clock perché sono indipendenti tra loro
  - Esempio: ALU può essere utilizzata in tutti i cicli di clock, triplicando il lavoro

Esecuzione di un'istruzione

Codice di SWAP per architettura a tre bus

Label	Operations	Comments
swap1	MAR = SP – 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS

Esecuzione di swap con architettura pipeline

	Swap1	Swap2	Swap3	Swap4	Swap5	Swap6
Cy	MAR=SP–1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP–1;wr	TOS=H;goto (MBR1)
1	B=SP					
2	C=B–1	B=SP				
3	MAR=C; rd	C=B				
4	MDR=mem	MAR=C				
5			B=MDR			
6			C=B	B=TOS		
7			H=C; wr	C=B	B=SP	
8			Mem=MDR	MDR=C	C=B–1	B=H
9					MAR=C; wr	C=B
10					Mem=MDR	TOS=C
11						goto (MBR1)

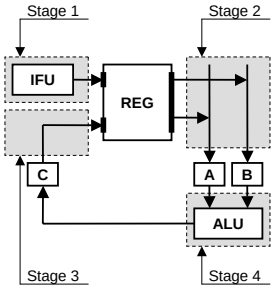
Pipelining

- Il pipelining consente di:
  - Ridurre il periodo di clock
    - Le operazioni svolte in ogni ciclo sono più semplici
  - Massimizzare il parallelismo
    - Le sezioni del data path che svolgono le operazioni semplici possono essere opportunamente isolate

Pipelining

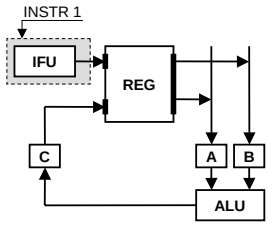
Gli elementi di interesse sono:

- **IFU:** Instruction Fetch Unit
- **REG:** Register File
- **A:** Latch on bus A
- **B:** Latch on bus B
- **C:** Latch on bus C
- **ALU:** ALU and SHIFTER

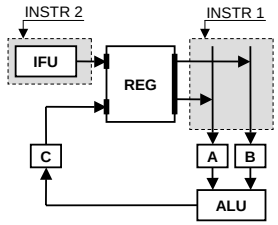


Pipelining

- Tempo T=0:
  - *Prefetch* dell'istruzione 1

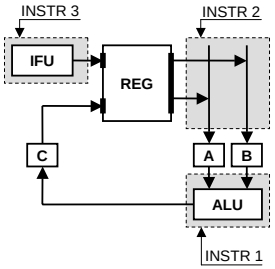


- Tempo T=1:
  - *Prefetch* dell'istruzione 2
  - *ReadReg* dell'istruzione 1

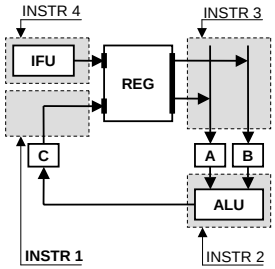


Pipelining

- Tempo T=2:
  - *Prefetch* dell'istruzione 3
  - *ReadReg* dell'istruzione 2
  - *Execute* dell'istruzione 1

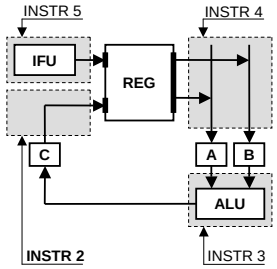


- Tempo T=3:
  - *Prefetch* dell'istruzione 4
  - *ReadReg* dell'istruzione 3
  - Execute dell'istruzione 2
  - ***Write-Back*** **dell'istruzione 1**

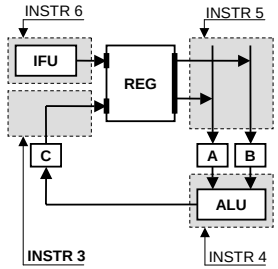


Pipelining

- Tempo T=4:
  - *Prefetch* dell'istruzione 5
  - *ReadReg* dell'istruzione 4
  - *Execute* dell'istruzione 3
  - ***Write-Back*** **dell'istruzione 2**



- Tempo T=5:
  - *Prefetch* dell'istruzione 6
  - *ReadReg* dell'istruzione 5
  - Execute dell'istruzione 4
  - ***Write-Back*** **dell'istruzione 3**



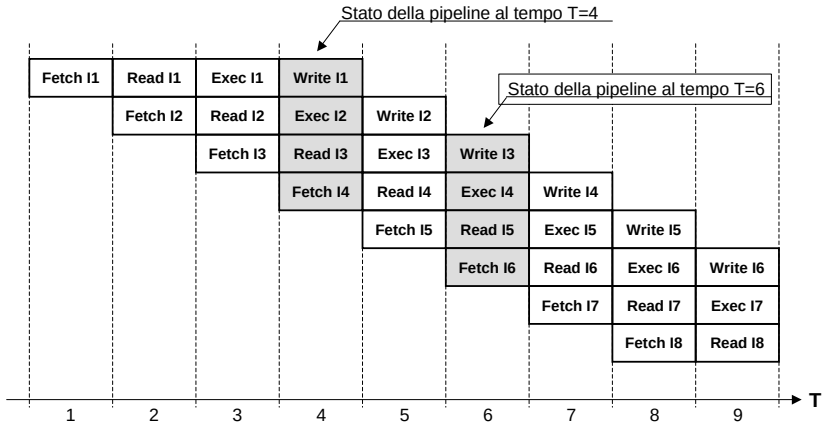
Aspetti importanti

- La prima microistruzione richiede 4 cicli per essere completata
  - Il tempo di completamento della prima microistruzione o, equivalentemente, il tempo che una istruzione impiega ad attraversare tutti gli stadi della pipeline viene detto *latenza*
- Nei cicli successivi viene completata una microistruzione ogni ciclo
  - Il numero di microistruzioni completate per unità di tempo è detto *throughput.* Nel caso ideale il trhoughput è pari alla frequenza del clock

Caratteristiche del pipeline

- Dopo una fase iniziale, tutti gli stadi sono simultaneamente utilizzati
  - Il parallelismo è massimizzato
- Ad ogni ciclo sono in esecuzione 4 istruzioni
  - Ogni istruzione si trova in uno stadio di elaborazione differente

Esecuzione in pipelining



Pipelining

- Sia **I1** una istruzione che inizia al tempo *t:*
  - **I1** legge i registri nello stadio 1, cioè al tempo *t+1*
  - **I1** scrive i registri nello stadio 3, cioè al tempo *t+3*
- Sia **I2** una istruzione che inizia al tempo *t+1:*
  - **I2** legge i registri nello stadio 1, cioè al tempo *t+2*
  - **I2** scrive i registri nello stadio 3, cioè al tempo *t+4*
- Se **I1** scrive un registro che **I2** deve leggere:
  - **I1** scriverà il risultato nel registro al tempo *t+3*
  - **I2** leggerà il registro al tempo *t+2*
- **I2 legge il registro prima che I1 lo abbia scritto**
- Questa situazione è detta *conflitto* o *hazard*

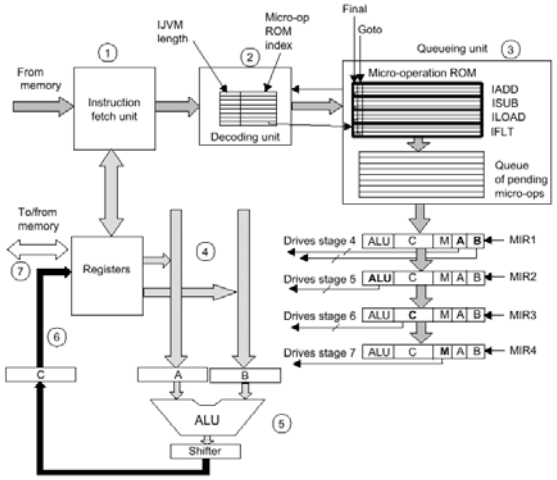
Conflitti

- Esitono diversi tipi di conflitti dovuti a cause diverse
- Conflitto Read After Write: *RAW hazard*
  - Una istruzione tenta di leggere un registro prima che sia stato scritto
  - Si risolve ritardando l'esecuzione della seconda istruzione
  - Un ritardo inserito dopo la fase di fetch prende il nome di *stallo*

Conflitti

- Conflitto di controllo: *Control hazard*
  - Una istruzione di salto provoca sempre un conflitto di controllo
  - L'unità di fetch ha già letto l'istruzione successiva, senza tener conto del fatto che il salto potrebbe prevenirne l'esecuzione
  - Si risolve inserendo una istruzione fittizia (NOP) tra un salto e l'istruzione seguente
- Pipeline più complesse (7 o 13 stadi) sono soggette anche ad altri tipi di conflitti

Pipeline a 7 stadi (Mic-4)



Pipeline a 7 stadi

Stadio 1: Instruction Fetch Unit
Stadio 2: Decoder
Stadio 3: Coda
Stadio 4: Operandi
Stadio 5: Esecuzione
Stadio 6: Riscrittura
Stadio 7: Memoria

Riduzione del numero di stalli

- Si può procedere in due modi, non mutuamente esclusivi:
  - Studio di opportune tecniche di compilazione mirate alla riduzione del numero di situazioni potenzialmente critiche tramite:
    - Riordino delle istruzioni nel rispetto delle dipendenze tra i dati
    - Introduzione di istruzioni fittizie
  - Realizzazione di una unità hardware in grado di prevedere situazioni critiche con un sufficiente margine di anticipo
    - Riordino dinamico delle istruzioni
    - Predizione dei salti

Branch Prediction: Problemi

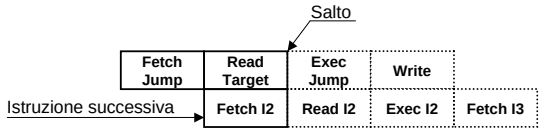
- Una architettura pipeline massimizza le prestazioni se il codice non contiene istruzioni di salto
- Il codice reale, tuttavia, ha una alta percentuale di salti, circa uno ogni 4-5 istruzioni

Branch Prediction

- Salti condizionati:
  - Non è noto a priori quale sarà l'istruzione successiva
  - L'unità di fetch deve attendere il completamento della fase di esecuzione dell'istruzione di salto (2 cicli, nell'esempio visto)
- Salti incondizionati
  - La destinazione è nota senza ambiguità
  - L'unità di fetch deve attendere il completamento della fase di lettura degli operandi dell'istruzione di salto (1 ciclo, nell'esempio visto)

Branch Prediction: Soluzioni

- Salti incondizionati
  - Una unità hardware dedicata ha un costo molto elevato
  - L'indirizzo di destinazione è noto al secondo stadio della pipeline quindi una sola istruzione dopo il salto è già stata iniziata



Branch Prediction: Soluzioni

- Salti condizionati
  - Una unità hardware dedicata ha un costo molto elevato
  - L'esecuzione dei salti condizionati richiede il completamento di un numero maggiore di stadi della pipeline
  - Più di una istruzione successiva al salto inizia ad essere eseguita