FETCH → DECODE → EXECUTE → MEMORY → WRITE-BACK
F D E M W

Esempi:

mov $8, %eax →
1. istruzione caricata in memoria
2. valore letto $8
3. ——
4. ——
5. scrive in %eax

add %ecx, %eax →
1. istruzione caricata in memoria
2. legge da registro %ecx, %eax
3. compie la somma in ALU
4. ——
5. scrive %eax

add $60 , (%ecx) →
1. istruzione caricata in memoria
2. legge dr, legge %ecx
3. ——
4. legge (%ecx) TROPPI TEMPI !
5. ——

RISC → perché istruzioni (F-W)
CISC → molte istr (F0-F5) → istruzioni complesse

L'esecuzione di ogni istr richiede cicli di controllo di processamento viene scandito dal clock di sistema.

ciclo di clock
tempo di salita

ESECUZIONE CON PIPELINING

Immaginiamo di eseguire tre istruzioni

CICLI DI CLOCK
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 t
istr 1: F D E M W
istr 2: F D E M W
istr 3: F D E M W
5 cicli
ESECUZIONE SEQUENZIALE
15 cicli

1 2 3 4 5 6 7 8 9 10 11 t
istr 1: F D E M W
istr 2: F D E M W
istr 3: F D E M W
5 cicli
7 cicli
ESECUZIONE CON PIPELINING

ATTENZIONE: non sempre è possibile mantenere la politica "piena" di istruzioni, a causa delle presenze di "hazard"

Un "hazard" è una incompatibilità presente tra due stadi di esecuzione di due istruzioni differenti.

HAZARD STRUTTURALI: due o più istruzioni richiedono l'uso contemporaneo di una stessa risorsa presente nella CPU

Es. mov (%ebx), %eax
Istr | F | D | E | M | W
Mem | Reg | | | Mem | Reg
Istr2 | | Mem | Reg | | Mem | Reg
Istr3 | | | Mem | Reg | | Mem | Reg
Istr4 | | | | Mem | Reg | | Mem | Reg
1 2 3 4 5 6 7 8
Tempo (cicli di clock)

Mitigazioni:
1. Duplico l'HW
2. Stallo anche soprattutto per istruzioni e dati
3. Utilizzare la tecnica dello "stallo"

HAZARD SUI DATI: si verificano quando esistono dipendenze tra i dati su cui operano istruzioni diverse

Es.:
1. mov1 %ebx, %eax
2. mov2 %eax, %ecx

F D E M W
1 2 3 4 5 6 7

Soluzioni:

TECNICA DELLO STALLO: Le CPU fanno la pipeline in "stallo" (non eseguo nuova istruzione) per 1 numero di cicli di clock necessario a risolvere l'hazard

1 2 3 4 5 6 7 8
F D E M W
Istr1
Istr2
Istr3
Istr4
E

L'introduzione di stalli prolungati rende SEMPRE qualcuno inoperoso.
ATTENZIONE: l'allocazione di stalli riduce il throughput

RIORDINAMENTO DEL CODICE: cambio l'ordine di esecuzione delle istruzioni per evitare l'hazard

Es.:
mov1 %ebx, %eax
mov2 %ecx, %edx
mov3 %ebx, %ecx
mov4 %edx, %eax
mov5 %eax, %ecx

HAZARD SUL CONTROLLO: l'esecuzione di una istruzione dipende dalla valutazione di una precedente condizione

Es.:
```
    TEST %eax, %eax
    jz L1
    mov1 %eax, %edx
L1: add1 ...
```

Mitigazione: la CPU usa tecniche di "branch prediction" per cercare di "indovinare" quale sarà la successiva istruzione da eseguire.

Oppure... potrebbe istruzione che non generano hazard sul controllo

```
    TEST %ecx , %eax
    cmovl %eax, %edx
```

HAZARD SUL CONTROLLO: l'esecuzione di una istruzione dipende dalla valutazione di una precedente condizione

Es.:
```
    TEST %eax , %eax
    jz L1
    mov1 %eax, %edx
L1: add1 ...
```

Mitigazione: la CPU usa tecniche di "branch prediction" per cercare di "indovinare" quale sarà la successiva istruzione da eseguire.

Oppure... potrebbe istruzione che non generano hazard sul controllo

```
    TEST %ecx , %eax
    cmovl %eax, %edx
```

INTERRUZIONI

Un "interrupt" è un meccanismo generato dalla CPU su cui un dispositivo opportuna istruzione sul Un meccanismo che richiede l'attenzione del sistema.

In caso di interruzione, l'esecuzione del processo corrente viene sospesa e il suo posto il SO eseguire una routine di gestione dell'evento chiamata INTERRUPT HANDLER

Eventi diversi sono rappresentati da interrupt diversi ciascuno con il suo handler.

La mappatura tra interrupt ed handler viene mediante la interrupt vector table (IVT)

indirizzo IVT
IVT
interrupt handler
spostamento da seguire per trovare l'indirizzo dell'interrupt handler
interrupt vector

INTERRUPT

interruzioni non bloccanti
recuperabili
non recuperabili

ASINCRONI

Interruzioni HW generate da un dispositivo esterno segnale elettrico. Sono fondamentali generati la periferiche.

SINCRONI

TRAP: interruzioni generate dal programmatore. Per 16-32 le trap possono essere generate tramite l'istruzione "int N" (con N < 256 dell'interrupt da generare. Dopo l'esecuzione dell'handler il processo riprende da dove era stato interrotto.

ABORT: interrupt non-mascherabile non recuperabile e non riparabile

FAULT: generati dalla CPU al verificarsi di alcuni errori. Spesso recuperabili. Dopo l'esecuzione dell'handler, il processo riprende dall'istruzione in cui è stato generato il fault

ALCUNE INFORMAZIONI APPROFONDITE SUGLI INTERRUPT

CONTEXT SWITCH CON PREEMPTION

context switch: interrompere l'esecuzione del processo in corso e preparare la CPU per eseguire il successivo processo scelto dallo scheduler.

Avviene tre passi:
1. La stato del processo interrotto viene salvato nel suo PCB
2. Il PCB del processo interrotto viene aggiornato alla fine del processo in stato READY
3. Lo stato del processo READY da cui eseguire viene ripristinato a partire dal suo PCB.

time-slice A
time-slice 2
TS 3
PA
PB
generato un interrupt
context switch
user mode
kernel mode

Il context switch rappresenta "pura overhead"

SYSTEM CALL

In 16-32 una specifica trap viene usata per invocare l'esecuzione di una system call. Usiamo l'istruzione ASM

INT $0x80 o INT 0x80

per richiedere l'esecuzione di una system call.
Come passiamo i parametri alla system call ?
I parametri vengono passati attraverso i registri:

eax → identificativo numerico della syscall da eseguire
ebx → 1° parametro
ecx → 2° "
edx → 3° "
esi → "
edi → "

Il valore restituito dalla syscall si in eax

Esempio: chiudiamo la syscall "exit(int)" da tre righe 2

```
global _my_exit

_my_exit:
    push1 %ebx
    mov1 8(%esp), %ebx    // 1° parametro
    mov1 $1, %eax         // 1 → exit
    INT 0x80              // valore di ritorno come primo parametro
    pop1 %ebx
    ret
```

PASSAGGIO DI UN PROCESSO DA WAITING A READY

scheduler
READY
RUNNING
Timer interrupt
System call Trap 0x80
ricezione di un segnale interrupt
WAITING