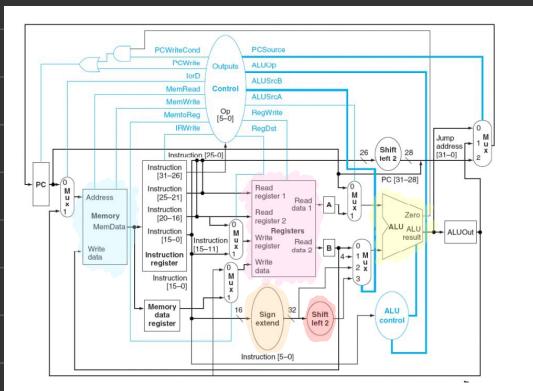




ARCHI PT. 2

DATAPATH

DEFINIZIONE: insieme di componenti elettroniche necessarie per l'implementazione e l'esecuzione di istruzioni



UNITÀ FUNZIONALI

- **MEMORIA**: Contiene istruzioni e dati
- **REGISTER FILE**
- **SIGN-EXTEND**: estende 16 bit a 32 bit
- **SHIFT-LEFT**: opera uno shift di 2 a sinistra
- **ALU**: esegue operazioni in base al tipo di istruzione richiesta

Un datapath può essere realizzato in SIMBOLICO CICLO oppure MULTICICLO

- DIFERENZE !**
- ~ **Simbolo Ciclo**: un ciclo è la lunghezza fissa uguale al tempo d'esecuzione dell'istruzione più lunga; istruzione eseguita in un unico ma lungo (anche innecessariamente) ciclo di clock, spreco di tempo; unità funzionali ripetute
 - ~ **Multiciclo**: un ciclo ha lunghezza fissa più corta; istruzioni eseguite in più cicli di clock. Corti: (variabile la quantità); meno ripetizioni di unità funzionali
- ↳ inoltre, il multiciclo è realizzato tramite registri di appoggio per ogni unità funzionale.

IMPLEMENTAZIONI delle ISTRUZIONI

Due passi in comune: FETCH + DECODE

1° **FETCH**: $IR \leftarrow PC$ trasferisco l'istruzione da eseguire nell'INSTRUCTION REGISTER

$PC \leftarrow PC + 4$ incremento il PC di 4 per puntare alla prossima istruzione da eseguire

2° **DECODE**: $A \leftarrow Reg[IR[25-21]]$ } copio nei registri auxiliari i register dell'operazione
 $B \leftarrow Reg[IR[20-16]]$

$ALUout \leftarrow \text{sign-extended}(IR[15-0]) \ll 2$ BRANCH PREDICTION

3° **EXECUTE**:

JUMP TOT: 3 passi

$PC \leftarrow PC[31-28] + \text{sign-extend}(IR[25-0]) \ll 2$

BEQ TOT: 3 passi
tramite una sottrazione
if $A == B$ then $PC \leftarrow ALUout$

(add, sub, and, or, st)

R-type

[REGISTRO]

[REGISTRO]

WRITE-BACK IN RD

store
+ passi offset

$ALUout \leftarrow A + \text{sign-extend}(IR[15-0])$

PASSO 4

Carico indirizzo in ALUout e poi Scrivo

$H[ALUout] \leftarrow B$

load TOT: 5 passi Carico il contenuto in memoria in MDR e poi Scrivo il register file

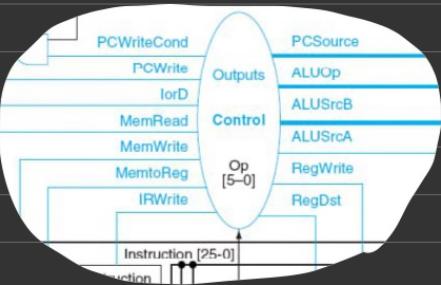
$ALUout \leftarrow A + \text{sign-extend}(IR[15-0]) \ll 2$ | $MDR \leftarrow M[ALUout]$ | $Reg[IR[20-16]] \leftarrow MDR$

PASSO 3

PASSO 4

PASSO 5

AUTOMA di controllo del datapath



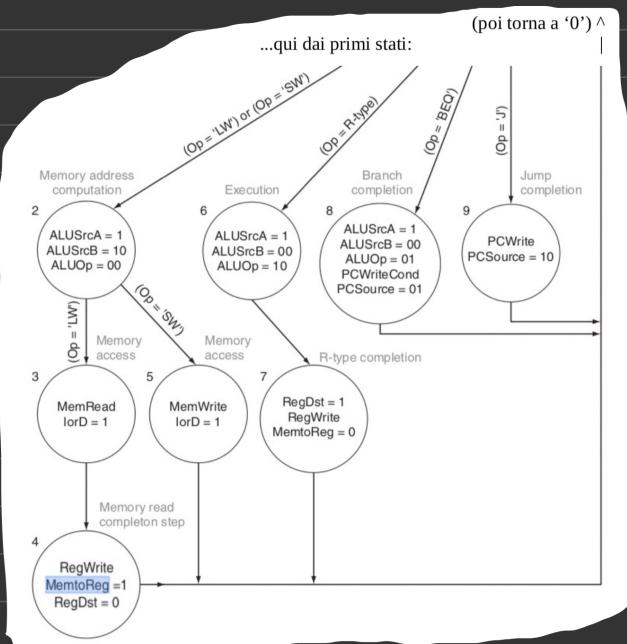
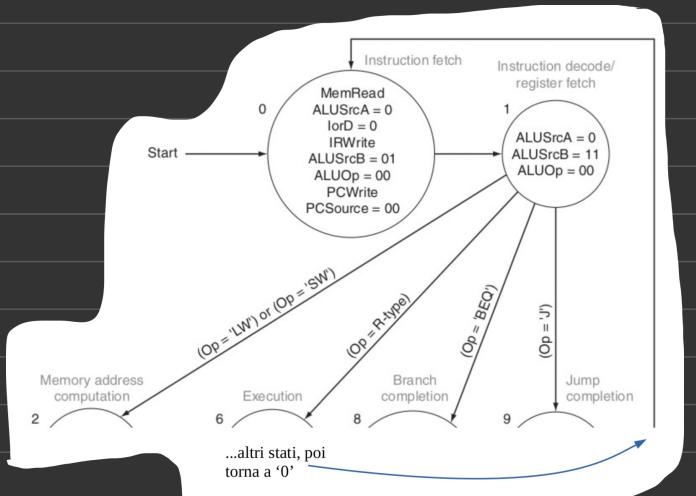
Che cosa fa? Emette i segnali utili all'esecuzione:

* controllo dei multiplexer

* controllo delle unità funzionali → scrittura, lettura...

E l'INPUT? Come input ha i 6 bit di Opcode dell'ISTRUZIONE corrente (IR [31:26])

L'AUTOMA può essere anche visto più da vicino tramite una next-state machine.

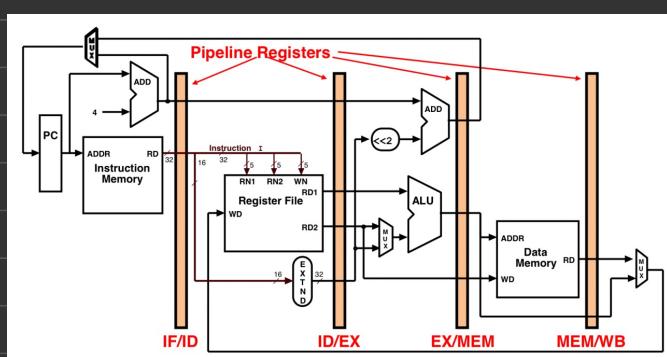


DATAPATH CON PIPELINE

Per risolvere le criticità del singolo ciclo si possono implementare le PIPELINE

CONSENTONO DI ELABORARE PIÙ PASSI DI DIVERSE ISTRUZIONI IN UN CICLO DI CLOCK CONTEMPORANEAMENTE

FUNZIONAMENTO: nei registri di pipeline vengono salvati tutti i risultati ottenuti dopo un ciclo di clock al ogni step dell'esecuzione (fetch, decode, exec, mem, WB)



L'AUTOMA DEI SEGNALI FLUISCE PASSO PASSO CON

L'ESECUZIONE, dall'IF (Instruction fetch)

Questo sistema di esecuzione parallela di più istruzioni in un ciclo può portare tuttavia a delle criticità, gli HAZARD

HAZARD (e come risolverli)

Esistono 3 tipi di hazard.

1) STRUCTURAL HAZARD: tentativo di accesso da un'unità funzionale in utilizzo da un'altra istruzione nello stesso ciclo

SOLUZIONI

→ Se l'hazard è a livello di memoria: * Delay del secondo accesso in un altro ciclo di clock

* Aggiungere un layer di memoria per avere memorie separate per istruzioni & dati

→ Se l'hazard è provocato dal register-file: * subdivisione del clock: in salita (↑) scrivo, in discesa leggo (↓)

2) CONTROL HAZARD: Passare all'esecuzione della prossima istruzione prima di aver valutato la condizione (ALTERA IL FLUSSO)

Soluzioni: * aspettare

* controllare la decisione prima

* fare una predizione

* rimandare il controllo

3) DATO HAZARD: tentativo di stabilire di un dato prima che sia pronto

SOLUZIONI: * aspettare (delay)

* FORWARDING → aggiungere componenti al datapath per collegare direttamente ALUOut al registro interessato risparmiando così un ciclo di aggiornamento

ECCEZIONI

→ eventi inattesi che si possono verificare durante l'esecuzione delle istruzioni

Eccezioni: evento sincrono, generato nel processore e provocato da problemi nell'esecuzione di un'istruzione

Interruzione: evento asincrono, generato fuori dal processore, di solito da un I/O che comunica alla CPU il verificarsi di certi eventi

All'verificarsi di un'eccezione, la gestione viene affidata ad un handler che valuta se il problema è riadreibile o meno (in caso negativo termina l'esecuzione)

All'verificarsi di un'interruzione, la normale esecuzione viene interrotta per gestire l'interruzione e poi si riprende l'esecuzione del programma

GESTIONE DELL'EVENTO INATTESO

→ Si interrompe l'esecuzione corrente del programma

→ salvataggio parziale dello stato di esecuzione corrente → esclusi \$K1 e \$K0 che sono salvezze kernel

→ Salto ad una routine del SO per gestire l'evento (handler)

→ esecuzione della routine del S.O.

→ in caso positivo (risolto), ripristino dello stato di esecuzione del programma e continuazione

IDENTIFICARE L'EVENTO INATTESO

Due modi:

- Indirizzo fisso: si usa un registro dedicato "Cause" in cui il controllo della CPU salva un identificatore numerico del tipo di eccezione verificatosi; l'hardware accederà a questo registro per verificare la causa dell'eccezione
- Interrupt vettorizzati: hardware diversi per diverse eccezioni/interrupt, ci sarà quindi un vettore di indirizzi che indirizzerà tramite il codice numerico dell'eccezione/interrupt a cui il controllo accederà e sorterà

IMPLEMENTAZIONE DI MIPS

- Si usa l'indirizzo fisso, detto Cause, per memorizzare il motivo dell'eccezione

→ Viene usato inoltre anche un altro indirizzo, l'EPC, che serve a indicare l'indirizzo dell'istruzione che ha causato l'eccezione

PASSI

in fetch per l'interrupt

- Individuare l'evento inatteso e Salvare ID in Cause
- Interrrompere l'esecuzione corrente
- Salvare l'indirizzo dell'istruzione corrente nel EPC ($EPC \leftarrow PC - 4$, con addro quella corrente e non quella dopo ($PC + 4$))
- Passare il controllo al gestore software, interrompendo l'esecuzione corrente
- Una volta trattata l'eccezione si torna all'esecuzione [eret]

MIPS salva solo il PC, è la routine che salva altri stati d'esecuzione

→ L'ECCEZIONE VIENE RILEVATA E TRATTATA PRIMA del completamento dell'esecuzione dell'istruzione corrente:

- Se l'eccezione era un interrupt, dopo la gestione si riprende dall'istruzione corrente
- negli altri casi è possibile proseguire (se non ci sono errori irreparabili), l'esecuzione riprende dalla successiva (quella dopo la "Scatenante", $EPC + 4$)

LOCAZIONE DI MEMORIA A CUI SALTARE PER L'HANDLER: $0x80000180$ (si trova nello spazio Kernel text)

REGISTRO CAUSE: Exception Code [6:2 bit] (tabella in appendice)

INPUT/OUTPUT

Insieme di architetture per il trasferimento di informazioni da e verso l'elaboratore

Esiste un BUS di sistema che consente la comunicazione tra I/O e CPU

Il BUS è organizzato in 3 linee:

- Bus di dati: trasfondere dati da e verso dispositivi
- Bus di controllo: trasporta informazioni per la definizione delle operazioni da compiere e controllo di dispositivi
- Bus degli indirizzi: trasmette indirizzi in memoria che identificano i dati da leggere/scrivere da memoria o periferiche

nel COMPUTER

I dispositivi I/O possiedono delle interfacce, costituite da una parte hardware (controllore) e una software (driver).

STRUTTURA INTERFACCIA (registri della periferica):

- registri dati: "dati di Input + Output"
 - registri di stato: "ready", letto dalla CPU
- DENTRO LA PERIFERICA

I dispositivi I/O possono essere individuati in due modi:

1° MAPPATURA IN MEMORIA (RAM)

- ogni periferica ha uno spazio dedicato in memoria con un indirizzo e identificazione unico
- i registri della periferica (interfaccia) sono mappati in memoria
- non accessibili ai programmi utente, li gestisce il SO

2° BUS SEPARATI (dedicati ad ogni tipo di periferica)

TECNICHE DI GESTIONE I/O

INDICATORI TECNICI DI UNA BUONA COMUNICAZIONE

- BANDA PASSANTE: quantità di informazioni e dati trasmesse per un certo intervallo di tempo (indicatore del flusso)
- LATENZA: tempo che intercorre tra il ready di una periferica per il trasferimento e l'istante in cui viene trasferito

1° PROGRAMMED I/O (software)

La CPU si occupa del controllo e del trasferimento dati, predispose lei tutte le operazioni.

- Resta poi in attesa (senza fare nient'altro) del ready della periferica mettendo in ascolto [busy waiting]

VANTAGGI: RISPOSTA FULMINA AL READY

SVANTAGGI: BUSY WAITING

BANDA PASSANTE: ALTA, CPU gestisce l'I/O senza fare altro

LATENZA: MINIMA, LA CPU ricepisce subito il ready

2° INTERRUPT I/O

Più efficiente della prima, viene usato hardware (segnali di interrupt)

→ La periferica invia un segnale di interrupt

→ il processore lo capta in fetch e le risponde con un "interrupt acknowledge"

→ salva lo stato di esecuzione e inizia la procedura del gestore

→ una volta terminato si ripete la normale esecuzione

VANTAGGI: risolto il busy waiting

SVANTAGGI: la CPU deve ancora occuparsi del trasferimento

BANDA: MINOR BANDA PERCHÉ AUMENTA LA LATENZA

LATENZA: AUMENTA IN QUANTO LA RISPOSTA AD UN INTERRUPT È PIÙ COMPLESSA RISPIETTO AD "ASCOLTARE IL READY"

3° DMA (Direct Memory Access)

La periferica è incaricata di gestire il trasferimento dati senza l'intervento della CPU,

vengono coordinati gli accessi tra le due

~CPU impone il trasferimento, la periferica se ne occupa → consente l'autonomia

→ c'è bisogno di introdurre due nuovi registri nell'interfaccia (quindi hw in più)

* registro indicatore di dove trasferire i dati

* registro indicatore della quantità di dati da trasferire

→ INVIA infine un INTERRUPT per segnalare la fine del trasferimento

VANTAGGI: la CPU è libera dal trasferimento

SVANTAGGI: costo in hardware e bisogna prevedere la condivisione del BUS

BANDA: massima, CPU non coinvolta e accesso diretto alla RAM

LATENZA: minima, l'hw trasferisce appena il pronto e senza CPU

I/O IN ASSEMBLY

```

15) LI $13, $2      push counter
   LA $14, 0x1000100 mappatura
   LA $10, 0x80000440 periferica

L2: BEQ $13, $0, L3
    3 verifica se i dati siano finiti

L1:
   LW $11, 0($10)
   LI $12, 1
   AND $11, $11, $12
   BEQ $11, $0, L1
   LW $11, $11, L1
   SW $11, ($14)

ADDI $10, $12, -1
ADDI $14, $14, 4
J L2      ↗ solo incondizionato
          ↗ nessuna precedenti

L3: Specifica cosa fa "L3"
Opzioni risposta multipla:
  1) Verifica se la periferica è in stato "ready"
  2) Verifica se siano finiti i dati da trasferire
  3) Si salta incondizionatamente al trasferimento del prossimo dato
  4) Il dato proveniente dalla periferica viene scritto in memoria

```

CACHE

GERARCHIE di MEMORIA: organizzazione di livelli di memoria, con ciascuno propria velocità e dimensione.



→ (un programma usa solo una parte del suo ragionamento)

In generale, esistono due **PRINCIPI DI LOCALITÀ** che vengono impiegati nella gerarchia

- **TEMPORALE:** tendenza di riferirsi allo stesso elemento in poco tempo
- **SPAZIALE:** tendenza di riferirsi ad elementi con indirizzi vicini

QUINDI

$$\Delta \uparrow \begin{matrix} +\text{veloce} \\ +\text{costoso} \end{matrix} \quad \Delta \downarrow \begin{matrix} +\text{grande} \\ +\text{economico} \end{matrix}$$

Definizioni base

- blocco/linea: la più piccola quantità di info in memoria
- hit: l'info richiesta è stata trovata nel livello superiore
- miss: l'info non è presente nel livello superiore, bisogna andare a quelli sotto
- hit rate: $\frac{\text{hit}}{\text{miss} + \text{hit}}$
- Miss Rate: $1 - \text{hit rate}$
- Tempo di hit: tempo di accesso al livello sup, compresa decisione di hit/miss
- Tempo di miss: tempo necessario a sostituire un blocco sup e trasferire i dati

CACHE: il livello di memoria più vicino al processore

ORGANIZZATA IN 3 MODI:

- 1) **Direct Mapped Cache:** associa una posizione precisa in memoria ad ogni blocco

L'INDIRIZZO DI MEMORIA È COMPOSTO DA:

- **tag**: contiene informazioni necessarie a capire se una word in cache corrisponde a quella cercata
- **indice**: used per selezionare il blocco della cache
- **offset**: bit per allestire il byte richiesto della word (una word sono 4 byte, 32 bit)

→ gli ultimi $\log_2 n$ bit dell'indirizzo devono corrispondere al blocco

i bit di indice = $\log_2 n$ dove n è il numero di blocchi in cache

i bit di offset = $\log_2 \times$ dove \times è il numero di word in un blocco

le **linee** invece costituiscono i blocchi contengono:

→ **Bit di validità**: verifica la validità del dato in quella linea (all'avvio è falso, perciò **COMPULSORY MISS**)

→ **tag**: identificano il blocco a cui appartiene la linea

→ **blocco di dati**: una o più Word

LA MAPPATURA INDIRIZZI SI EFFETTUÀ CO SI:

indirizzo del dato (in byte) = indirizzo del blocco → indirizzo del blocco / numero blocchi in cache
byte per blocco

Prestazioni della Direct Mapped

- con blocchi più grandi in memoria diminuisce la frequenza di miss, ma la miss penalty aumenta drasticamente
- se i blocchi sono troppo grandi rispetto alla cache cresce la frequenza di miss, non bastano i blocchi in cache

2) FULLY ASSOCIATIVE CACHE → blocchi posizionati in una qualsiasi locazione in cache, bisogna cercarli in tutta la memoria
↓
bomba e caccia

3) SET ASSOCIATIVE: blocchi raggruppati in set

Svantaggi: → COMPLESSA RISPETTO ALLA DIRECT

- M blocchi per set, posizione qualsiasi
- si confrontano in parallelo i tag per verificare la presenza in quel set o meno del blocco ricercato
- IL BLOCCO È DISPONIBILE SOLO DOPO LA DECISIONE DI HIT O MISS

ALGORITMI PER LA SOSTITUZIONE DI BLOCCHI (IN UN MISS)

CANDIDATI:

- direct mapped: il contenuto di quella locazione mappata viene sostituito
- fully: ogni blocco è un candidato
- sets: ogni blocco del set è un candidato

POLITICHE fully e sets:

* random;

* LRU, il meno utilizzato viene aggiornato (realizzato tramite un contatore che decrementa)

* FIFO, il blocco più vecchio viene sostituito

Se si verifica un miss:

- PC + 4
- leggo in memoria e scrivo nella cache
- riparto dall'istruzione successiva

TECNICHE di Aggiornamento

Per mantenere una coerenza tra dati in memoria e dati in cache:

- ~ WRITE-THROUGH: scrittura contemporanea tra cache e memoria
 - ✓ soluzione semplice
 - ✓ coerenza garantita
 - ✗ velocità limitata della memoria inferiore, diminuite prestazioni
 - ✗ aumenta il traffico sul BUS

WRITE MISS

SOLUZIONE:

- * allocare: blocco caricato in cache e scritto
- * no-allocare: scrittura diretta in memoria

~ WRITE-BACK: quando si deve sostituire un blocco in cache, il blocco viene aggiornato in memoria inferiore

✓ SCRITTURA A VELOCITÀ CACHÉ (avviengono solo lì fino a sostituzione)

✗ ogni sostituzione provoca la scrittura, anche quando magari il dato è intatto

~ WRITE-THROUGH CON BUFFER: introducendo un buffer tra cache e memoria inferiore

* processore scrive in cache e buffer

* il controllore della memoria scrive dal buffer alla memoria



✗ bisogna evitare la saturazione del buffer, altrimenti ci saranno stalli di writing

ESERCIZI: A CHE NUMERO DI BLOCCO?

1) Ho una cache di 16 blocchi da 4 word, a che numero di blocco

Viene mappato l'indirizzo 64? (indirizzamento al byte)

$$\log_2 16 = 4 \text{ bit indice}$$

$$\log_2 4 = 2 \text{ bit offset per word} \rightarrow \text{al byte invece } 2+2=4$$

quando $64_{10} = 1000000_2$

scarto offset
↓
 0100_2
→ blocco 4

1.2) Cache = 16 blocchi indirizzo 128_{10}

word = 8 al byte?

$$\log_2 16 = 4 \text{ indice}$$

$$\log_2 8 = 3 \text{ offset} \rightarrow \text{reale } 3+2=5 \text{ bit}$$

$128_{10} = 10000000_2$

0100_2
↓
m° 4

1.3) Cache = 4 blocchi indirizzo 128_{10}

word = 8 al byte?

$$\log_2 4 = 2 \text{ indice}$$

$$\log_2 8 = \text{offset} \rightarrow \text{reale } 3+2=5 \text{ bit}$$

$128_{10} = 10000000_2$

↓
 00_2

1.4) Cache = 8 blocchi $256_{10}?$

word = 16 al byte?

$$\log_2 8 = 3 \text{ indice}$$

$$\log_2 16 = 4 \text{ offset} \rightarrow \text{reale } 4+2=6$$

$256_{10} = 10000000_2$

↓
 $100_2 = 4$

1.5) 8 blocchi 96_{10}
8 word al byte?

$$\log_2 8 = 3$$

$$\log_2 8 = 3 \rightarrow 3+2=5$$

$$96_{10} = 64+32 = 110000_2$$

↓
 $011_2 = 3$

1.6) $64_{10}?$ $64_{10} = 1000000_2$
↓
 $010_2 = 2$

ESERCIZIO: NUMERO MEDIO di CPI

2) 10 CPI per accesso in cache hit probability di 0,8
15 cicli di miss penalty

NUMERO MEDIO di CPI? $\text{HIT} * \text{HIT PROB} + \text{MISS} * (\text{1-HIT PROB})$
quindi

$$0,8 * 10 + 15 * 0,2 = 8 + 3 = 11$$

2.1) 10 CPI
hit prob 0,9
15 miss penalty

2.2) 5 CPI per cache
10 miss penalty
hit di 0,3

medio?

$$\text{medio} = 5 * 0,3 + 10 * 0,7 = 1,5 + 7 = 8,5$$

$$10 * 0,9 + 15 * 0,1 = 9 + 1,5 = 10,5$$

ESERCIZIO: VELOCITÀ MEDIA IN ISTRUZIONI PER SECONDO

3) 800 MHz frequenza di clock
10 CPI accesso cache
30 cicli di penalty
0,7 hit probability

1° converto in MHz (per aver milioni)

2° divido per numero medio

Velocità media in istruzioni/secondo?

$$\frac{800 \text{ MHz}}{10 * 0,7 + 30 * 0,3} = \frac{800 \text{ MILLIONI}}{16} = 50 \text{ milioni}$$

3.1) 2.2 GHz frequenza di clock
10 CPI per accesso in cache
15 cicli di penalty
hit 0,8

$$\frac{2.2 * 10^3}{10 * 0,8 + 15 * 0,2} = \frac{2200 \text{ Mhz}}{11} = 200 \text{ Mhz}$$

ESERCIZIO: QUANTI MISS CI SARANNO?

4) 4 blocchi
2 word
2 loop
5,2,3,4,4 load/store
QUANTI MISS?

1° divido per word (intero)	$5/2 = 2 \text{ } \% 4 = 2 \text{ H H}$
2° % di blocchi	$2/2 = 1 \text{ } \% 4 = 1 \text{ H H}$
	$3/2 = 1 \text{ } \% 4 = 1 \text{ H H}$
	$4/2 = 2 \text{ } \% 4 = 2 \text{ H H}$
	$7/2 = 2 \text{ } \% 4 = 2 \text{ H H}$

Missi 2 volte

4.1) 8 blocchi da 8 word

$$\begin{aligned} 58/8 &= 7 \text{ r. } = \text{H H} \\ 26/8 &= 3 \text{ r. } = \text{H H} \\ 31/8 &= 3 \text{ r. } = \text{H H} \\ 17/8 &= 2 \text{ r. } = \text{H H} \\ 42/8 &= 5 \text{ r. } = \text{H H} \end{aligned}$$

4.2) 8 blocchi, 1 word faccio solo il modulo numero blocchi.

$$22 \% 8 = 6 \text{ H}$$

$$\begin{aligned} 26 \% 8 &= 10 \text{ H} \\ 22 \% 8 &= 6 \text{ H} \\ 26 \% 8 &= 10 \text{ H} \\ 16 \% 8 &= 0 \text{ H} \\ 3 \% 8 &= 3 \text{ H} \\ 16 \% 8 &= 0 \text{ H} \\ 18 \% 8 &= 2 \text{ H} \\ 16 \% 8 &= 0 \text{ H} \end{aligned}$$

ESERCIZIO: è POSSIBILE GESTIRE IL TRASFERIMENTO?

formula: somma_istruzioni * cicli di clock * frequenza periferica

DATI	applico la formula
80+120 = istr. totali	$200 * 5 * 0,1 \text{ MHz} = 100 \text{ MHz}$
5 cicli di clock	\downarrow
100 kHz frequenza periferica	100 > 10
10 MHz clock processore	PERDITA DI DATI

confronto il risultato con clock processore (10 MHz)
 > perdita dati (in linea teoria)
 = bilanci, ma non ottimizza
 < bilanci a fare tutto

CONVERSIONI

$$\begin{array}{ccccccccc} T & 10^{-12} & 10^{-9} & 10^{-6} & 10^3 & \dots & \text{mili} & \text{micro} & \text{nano} & \text{pico} \\ & & & & & & 10^{-3} & 10^{-6} & 10^{-9} & 10^{-12} \end{array}$$

ESERCIZIO: TEMPO ESECUZIONE?

10) 100 ms esecuzione
50 ns accesso RAM

1 ns accesso cache (10^{-9})
0,05 ms ritardo

$$\frac{100 \text{ ms}}{50 \text{ ns}} = \frac{100 * 10^6 \text{ ns}}{50 \text{ ns}} = 2 \text{ miln} \quad (\text{numero di accessi in memoria})$$

51 ms miss penalty (50+1)

$0,05 * 1 + 51 * 0,05 = 3,5 \text{ ms}$ numero medio di CPI

$200 \text{ miln} * 3,5 = 700 \text{ miln} * 10^{-6} = 7 \text{ ms}$ tempo impiegato con la cache

ESERCIZIO: WRITE-BACK VETTORE

A $0x00100000$ \rightarrow B $0x001000FF$

altezza = 16 byte
ampiezza = 4 * 4 byte = 16 byte

CACHE: 16 blocchi da 4 word

quindi $B - A = 2047 + 1 = 2048 = \text{dimensione}$

2°) $\frac{\text{dimensione}}{\text{altezza}} = \frac{2048 \text{ byte}}{16 \text{ byte}} = 128$ 3°) $128 - n^{\circ} \text{ blocchi (16)} = 112 \text{ write-back}$ (altezza)