Architettura pt. 2

in sintesi ma
non troppo,
con colori carini.

Grazie a:
- appunti del Mega per la teoria
- @Koalas e @Emre per la pratica
- me per la bellezza di questi appunti :)

ECCEZIONI

GESTIONE DI:

- **ECCEZIONI** origine interna
- **INTERRUPT** origine esterna

} eventi distinti dai salti che alterano il flusso sequenziale di esecuzione di istruzioni

e.g. due tipi di eccezioni possono essere l'esecuzione di un'istruzione non valida e l'overflow aritmetico

OPERAZIONE FONDAMENTALE salvare l'indirizzo dell'istruzione che genera l'istruzione nell'EPC, exception program counter e trasferire il controllo ad un indirizzo preciso del sistema operativo, che a sua volta potrà intraprendere le giuste azioni:

- interrompere l'esecuzione oppure
- utilizzare l'EPC per determinare il punto da cui partire.

COMUNICARE L'ECCEZIONE ci sono due metodi per farlo:

- [MIPS] si prevede un registro dedicato **cause register** contenente un campo che indica la causa
- **interrupt vettorizzati** l'indirizzo a cui si deve trasferire il controllo viene determinato dalla causa dell'eccezione stessa

Nel dettaglio, in MIPS si ha:

- **istruzione indefinita** dopo lo stato 1 non si ha alcuna istruzione, il campo op non identifica alcun opcode e si avrà lo stato 10 (eccezione per istruzione indefinita)
- **overflow aritmetico** l'ALU riesce autonomamente ad identificarlo e si ha un segnale chiamato overflow come output che porta allo stato 11 (eccezione per overflow aritmetico)

e vengono aggiunti 2 nuovi segnali di controllo:

- EPCWrite
- CausaWrite
- PCsrc, che è un segnale aggiuntivo di 1 bit per fornire il valore corretto al bit meno significativo del cause register, detto CausaInt

ALTRE MODIFICHE ALLA DATAPATH

- Il multiplexor anteposto al PC diviene di 4 ingressi: un ingresso aggiuntivo connesso alla costante 0xC0000000, selezionabile con PCSource = 11
- in EPC deve essere scritto PC-4, quindi l'input dell'EPC è collegato all'output della ALU
- eventuali eccezioni annidate si gestiscono con politica FIFO

IMPLEMENTAZIONE MIPS

Introduce i seguenti registri:

- **BadVAddr** indirizzo di memoria di una reference di memoria sbagliata (bad address)
- **Status** interrupt mask
- **Cause** tipo di eccezione
- **EPC** indirizzo dell'istruzione che ha causato l'eccezione
- **Config** configurazione

↳ accessibili con l'istruzione **mfc0** e **mtc0**

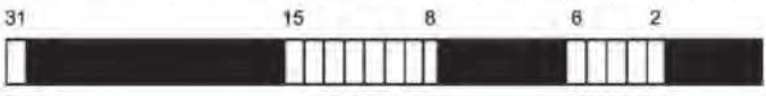
interrupt mask contiene 1 bit per ognuno dei 6 livelli hardware e 2 software di interrupt { 1 allows interrupt / 0 disables interrupt

Quando arriva un'interrupt, imposta il suo **interrupt pending bit** nel cause register. Quando è in pending, interromperà il processore quando la sua mask bit sarà attiva.

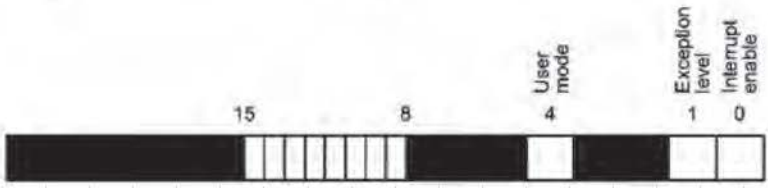
status register oltre alla interrupt mask, contiene:

- **user mode** 0 = kernel mode, spim è 1 perché non implementa la kernel mode
- **exception level** 1 dopo il verificarsi di una eccezione, se è 1 blocca la gestione di altre interrupt o exception
- **interrupt enable** 1 per consentire le interrupt

Cause register



Status register



EXCEPTION HANDLER IN MIPS

- Si trova alla locazione 0x80000180 del segmento kernel text
- Registri del coprocessore riservati al gestore:
  - $k0, $k1

  Deve preservare tutti i registri macchina, tranne i due precedenti
- MIPS rileva e tratta le eccezioni PRIMA del completamento dell'esecuzione della istruzione corrente:
  - se l'eccezione era interruzione, l'esecuzione deve riprendere dall'istruzione corrente (EPC)
  - se si tratta di un altro tipo di eccezione e si può proseguire, l'esecuzione deve riprendere dall'istruzione successiva (EPC+4).

GESTIONE I/O

GESTIONE DELLE PERIFERICHE

Si usa la parte di memoria dove ci saranno gli vari spazi dedicati alle varie periferiche e ogni registro della periferica appare anche in memoria in una locazione dedicata, permette le operazioni di accesso alla periferica esattamente come le letture e scritture in memoria

periferica componente con cui la CPU comunica mediante un'interfaccia costituita da registri periferica e specifica la periferica coinvolta in uno certa operazione mediante un indirizzo

INSIEME MINIMALE DI REGISTRI DI INTERFACCIA se nel bus si ha la giusta combinazione binaria si attiva la periferica corrispondente

I registri di interfaccia nella comunicazione tra CPU e periferica:

- **registro di stato** rappresenta lo stato della periferica
- **registro dati** registro di input/output per la CPU a seconda della periferica (di ingresso come la tastiera, o di uscita come la console)

PROBLEMA DI CONTROLLO

La CPU esegue ininterrottamente istruzioni ad una certa frequenza mentre le periferiche generano dati solo in certi momenti ad una certo frequenza.

↳ si introduce il **controllo di programma** la CPU controlla il valore del registro di stato di una periferica e ne copia il valore della locazione nello spazio di indirizzamento in cui è mappato ad un registro della CPU, per poterlo usare successivamente in una comparazione: { PERIFERICA NON PRONTA: riesegue il controllo fintantoché è pronta, senza fare altro
PERIFERICA PRONTA: inizia il trasferimento dei dati

Questo "stallo" è chiamato **busy wait**.

TRASFERIMENTO

Il trasferimento dal registro della periferica alla memoria dati si ottiene caricandone prima il valore nei registri della CPU, una lw/sw da registro periferica e poi una lw/sw con la memoria.

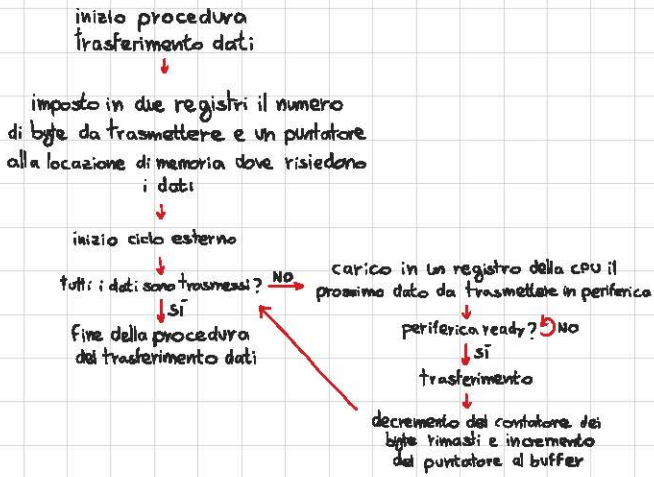
EFFICIENZA DELLA GESTIONE

Ci sono due metodi di valutazione:

- **banda passante** quantità di dati che si può trasferire per unità di tempo, misura di flusso
- **latenza** tempo che intercorre tra l'istanza Ready, che indica che la periferica è pronta, e l'istante in cui il dato viene trasferito, misura di tempo

TRASFERIMENTO MULTIPLO DI DATI

Stesso nucleo di comunicazione (ciclo interno) che dovrà essere ripetuto molte volte, in un ciclo esterno che comprenderà altre istruzioni ausiliarie:



URGENZA DI TRASFERIMENTO

Alcune periferiche hanno più urgenza di trasferire i dati, quindi esistono eventuali interruzioni durante l'input/output, chiamate **interrupt**, che libera la cpu dal busy/wait e fa comunicare CPU e periferiche senza l'intervento diretto della CPU.

Serve hardware aggiuntivo:

- **bus di controllo** che trasmette il segnale ready dalla perifevica alla cpu.
- quando una periferica genera un'interrupt, la CPU esegue una serie di istruzioni predefinite contenuta a partire dalla locazione di memoria prestabilita dentro quella dedicata al Kernel che sono:
  - salva lo stato della computazione al momento dell'interrupt
  - identificazione della periferica interrompente
  - gestione della periferica, col trasferimento del dato
  - ripristina lo stato della computazione a prima dell'interruzione
  - ritorno dell'interrupt (eret)

TIPI DI GESTIONE

- **unica linea di interruzione** per tutte le periferiche, si ha un unico gestore ma c'è, nella CPU, modo di analizzare lo stato di alcune linee di interruzione grazie al cause register (scelta di: MIPS)
- **vettorizzazione** l'interrupt genera un codice che la identifica su delle linee di bus dedicate. Quando arriva l'interrupt la CPU usa il codice come spiazzamento nel vettore interruzioni ed esegue il codice all'indirizzo "base del vettore + codice periferica".
- **direct memory access** si rende la periferica autonoma nell'accesso alla memoria, così da renderla capace di gestire da sola i trasferimenti, liberando la CPU dal busy/wait. Si aggiungono due registri che specificano • indirizzo di memoria con il quale scambiare dati
  • specifica la quantità dei dati da trasferire.
  ↳ performance elevate a livello di banda passante e latenza

IMPLEMENTAZIONE MIPS

Un device terminale è composto da due unità indipendenti:

- **receiver** riceve l'input dal terminale
- **transmitter** trasmette l'input alla console

Un programma li controlla usando 4 registri:

- **receiver control register** 0xffff0000, si usano 2 bit: • ready, 1 se è presente un input
  • interrupt enable, richiesta interrupt
- **receiver data register** 0xffff0004, 8 bit che conservano l'input (1 carattere)
- **transmitter control register** 0xffff0008, si usano 2 bit: • ready, 1 se è pronto ad accettare
  • interrupt enable, richiesta interrupt
- **transmitter data register** 0xffff000c, 8 bit che vengono inviati alla console

CACHE

GERARCHIA DI MEMORIA

Un programma non accede a tutte le sue istruzioni e a tutti i suoi dati con la stessa probabilità. Come soluzione si è trovata la gerarchia di memoria che consiste in un insieme di livelli di memoria, ciascuno caratterizzato da una diversa velocità e dimensione.

PRINCIPIO DI LOCALITÁ

Un programma in un certo momento accede soltanto ad una piccola porzione del suo spazio di indirizzamento.

Ci sono due tipi di località:

- **temporale** c'è la tendenza a riferirsi allo stesso elemento in poco tempo
- **spaziale** c'è la tendenza a riferirsi ad elementi con indirizzi vicini tra loro

Questo principio struttura la memoria in modo gerarchico:

- + vicina al processore => + veloce
- + grande => + lontana
- + costosa => + vicina al processore
- + economica => + lontana

DEFINIZIONI BASE

Un livello di memoria vicino alla CPU contiene dati memorizzati in ogni livello sottostante e tutti i dati sono nel livello più basso.

- **blocco/linea** la più piccola quantità di informazione che può essere presente/assente in una gerarchia di memoria
- **hit** l'informazione richiesta dal processore si trova in uno dei blocchi nel livello superiore di memoria
- **miss** l'informazione richiesta dal processore non si trova in uno dei blocchi nel livello superiore di memoria
- **hit rate** frequenza di hit, frazione degli accessi alla memoria nei quali l'informazione richiesta è stata trovata nel livello superiore di memoria.
- **miss rate** frequenza di miss, frazione degli accessi alla memoria nei quali l'informazione richiesta non è stata trovata nel livello superiore di memoria.
- **hit & miss rate** determina le prestazioni

- **tempo di hit** tempo di accesso al livello superiore della memoria, compreso il tempo necessario a stabilire se il tempo di accesso si risolva in un successo o in un fallimento
- **penalità di miss** il tempo necessario a sostituire un blocco del livello superiore con un nuovo blocco del livello inferiore della gerarchia, e trasferire i dati di questo blocco al processore

DIRECT MAPPED CACHE

La cache è il livello della memoria gerarchica che si trova tra il processore e la memoria principale.

La direct mapped associa una sola locazione della cache ad ogni word della memoria definendo una corrispondenza tra l'indirizzo in memoria e la locazione nella cache.

Per trovare il blocco che corrisponde ad un indirizzo della memoria principale:

indirizzo del blocco % numero di blocchi della cache

Si hanno poi i seguenti campi:

- **tag** contiene informazioni per verificare se una word della cache corrisponde ad una word cercata
- **indice** usato per selezionare il blocco della cache
- **blocco di validità** indica se il blocco di memoria associato contiene (True) o no (False) un campo valido

E le seguenti dimensioni:

- **dimensione della cache**
  - 2^n blocchi, n bit usati per l'indice
  - 2^m words, ossia m+2 byte, per cui m bit vengono usati per individuare una word all'interno di un blocco, mentre 2 bit per individuare un byte all'interno di una word
- **dimensione del campo tag** $32-(n+m+2)$
- **numero totale di bit** $2^n \cdot (\text{dim\_blocco} + \text{dim\_cache} + \text{bit\_validità}) = 2^n \cdot (2^m \cdot 32 + (32-(n+m+2)) + 1)$

Le **prestazioni** sono:

- blocchi più grandi => diminuisce la frequenza di miss
- se troppo grandi rispetto alla cache => cresce la frequenza di miss, infatti i blocchi vengono scaricati nella cache prima ancora di utilizzare i dati.

Inoltre => cresce il costo di una miss

ALTRI TIPI DI CACHE

- **fully associative** ogni blocco può essere collocato in qualsiasi locazione della cache, per trovarlo è necessario cercarlo in tutte le linee della cache, comportando lentezza e costo.
  Quindi, non viene impiegata alcuna indicizzazione.
- **set associative** soluzione intermedia: ciascun blocco della memoria ha a disposizione un numero fisso di locazioni in cache.
  I blocchi sono raggruppati in set ⇒ ogni indirizzo di memoria corrisponde ad un unico set e può essere ospitato in un blocco qualunque appartenente a quel set
  Stabilito il set, per determinare se un certo indirizzo è presente in un blocco del set è necessario confrontare in parallelo i tag di tutti i blocchi
  Ha degli svantaggi:
  - N comparatori invece di 1
  - ulteriore ritardo
  - il blocco è disponibile dopo l'hit/miss e la selezione del set, mentre in una ad accesso diretto il blocco è disponibile prima della decisione hit/miss

ALGORITMI PER LA SOSTITUZIONE DI BLOCCHI

Nell'accesso diretto, se il blocco di memoria è mappato in una linea di cache già occupata, si elimina il contenuto precedente della linea e si rimpiazza con il nuovo blocco
Nella fully, ogni blocco è un candidato.
Nelle set, ognuno degli n blocchi del set
In particolare, nelle ultime due si può seguire una politica:

- random
- Least recently Used
- First in first out

GESTIONE DELLA MISS

In generale, quando c'è una miss:

- inviare PC-4 alla memoria
- lettura della memoria
- scrittura nella cache
- riavvio dell'istruzione che ha causato la miss

ACCESSO IN SCRITTURA

Scrivere nella cache comporta il dover aggiornare i livelli inferiori della gerarchia di memorie

Ci sono 3 tecniche risolutive:

- **Write-through** i dati sono scritti nella cache e nel livello inferiore. È facile, ma diminuisce la velocità e aumenta il traffico sui bus
- **Write-back** i dati sono scritti solo nella cache. Il blocco viene scritto al livello inferiore solo quando deve essere sostituito.
  La scrittura avviene alla velocità della cache, ma ogni sostituzione può provocare un trasferimento in memoria
- **Write-buffer** interposto tra la cache e la memoria di livello inferiore, il processore scrive cache e buffer, poi il controller di memoria scrive il buffer in memoria
  È gestito LIFO: è efficace se la frequenza di scrittura è << 1 / write cycle, altrimenti il buffer può andare in saturazione.

WRITE MISS

Possono essere indotte dalle scritture.

Soluzioni possibili:

- **write allocate** il blocco viene caricato in cache e si effettua la scrittura
- **no-write allocate** il blocco viene scritto direttamente nella memoria di livello inferiore, senza essere trasferito in cache

DATAPATH

REALIZZAZIONE DI UNA DATAPATH

- Si stabilisce il set di istruzioni da implementare
- Si identificano i componenti del datapath
- Si stabilisce la metodologia di clocking
- Si assembla il datapath e si identificano i segnali di controllo
- Si determina il setting dei segnali di controllo, per ogni istruzione
- Si assembla la logica di controllo

Consideriamo un processore MIPS che esegue operazioni:

- **aritmetico-logiche** add, sub, and, or, slt
- **interazione con la memoria** lw, sw
- **salto** beq, j

PASSI PER L'ESECUZIONE DI UN'ISTRUZIONE

- **fetch** • legge l'istruzione dalla memoria e la salva nell'IR
  - l'indirizzo di memoria che indica l'istruzione da leggere si trova nel PC
  - dopo la lettura, con l'ALU si incrementa di 4 il PC
- **decode** decodifica i vari campi dell'istruzione per decidere quali sono i passi di esecuzione
- **execute** esegue i passi necessari per eseguire l'istruzione

IMPLEMENTAZIONE

- Fetch implementata con instruction memory, program counter, Adder
- Decode identifica OPCODE e, se necessario, FUNC CODE
- Execute: • R-type con registers e ALU
  - load and store con data memory unit e sign-extension unit
  - Beq con ALU, Adder, registri, Sign-extended

METODOLOGIA DI CLOCKING

- **singolo ciclo** ciclo singolo di lunghezza fissa uguale al tempo necessario per eseguire l'istruzione più lunga: ogni istruzione viene eseguita in un solo ciclo di clock.

  Comporta delle sfighe:
  - le istruzioni sono rallentate
  - unità funzionali replicate
- **multiciclo** cido di lunghezza fissa più corto: ogni istruzione viene eseguita in più cidi di clock, comporta che:
  - le unità funzionali sono usate più volte
  - si usano registri aggiuntivi per memorizzare i risultati parziali nell'esecuzione delle istruzioni.

IMPLEMENTAZIONE MULTICICLO

- registri aggiuntivi che memorizzano valori intermedi:
  - instruction register
  - memory data register
  - registri A e B
  - ALUOut
- riutilizzo delle unità funzionali:
  - ALU usata anche per calcolare i salti e incrementare il PC
  - memoria usata sia per leggere le istruzioni che per leggere/scrivere i dati

PASSI PER L'ESECUZIONE MULTICICLO

L'istruzione più lunga si esegue in 5 passi, quella più corta in 3 passi.

Operativamente, per ogni passo, abbiamo le seguenti operazioni e i seguenti segnali di controllo:

Fetch per tutte le istruzioni Passo 1

OPERAZIONI
- IR = M[PC]
- PC = PC + 4

S. DI CONTROLLO
- MemRead, per leggere dalla memoria
- IRWrite, per scrivere IR
- IorD, per indicare l'indirizzo da dove leggere dalla memoria
- ALUSrcA, ALUSrcB, ALUop, per incrementare il PC
- PCWrite, per salvare il nuovo valore del PC

decode per tutte le istruzioni — Passo 2

OPERAZIONI

- A = Reg[IR[25:21]]
- B = Reg[IR[20:16]]
- ALUOut = PC + sign_extended(IR[15:0]) << 2

S. DI CONTROLLO

- ALUSrcA
- ALUSrcB
- ALUOp

} per il calcolo di un eventuale branch

Passo 3 :

execute per le istruzioni lw e sw

OPERAZIONI

- ALUOut = A + sign_extended (IR[15:0])

S. DI CONTROLLO

- ALUSrcA
- ALUSrcB
- ALUOp

} per il calcolo dell'indirizzo di memoria per lw o sw

execute per le istruzioni R-type

OPERAZIONI

- ALUOut = A op B

S. DI CONTROLLO

- ALUSrcA
- ALUSrcB
- ALUOp

} per il calcolo aritmetico-logico

execute per le istruzioni beq

OPERAZIONI

- Se A==B allora PC = ALUOut

S. DI CONTROLLO

- ALUSrcA
- ALUSrcB
- ALUOp

} per la comparazione tra A e B

- PCWriteCond
- PCSource

} per scrivere PC

execute per le istruzioni jump

OPERAZIONI

- PC = (PC[31:28], IR[25:0] << 2)

S. DI CONTROLLO

- PCWrite
- PCSource

} per scrivere PC

Passo 4:

execute per le istruzioni lw e sw

OPERAZIONI

- MDR = M[ALUOut] oppure
- M[ALUOut] = B

S. DI CONTROLLO

- IorD, per indicare l'indirizzo di memoria
- MemRead, per leggere dalla memoria
- MemWrite, per scrivere nella memoria

Execute per le istruzioni R-type

OPERAZIONI

- Reg[IR[15:11]] = ALUOut

S. DI CONTROLLO

- RegWrite, per scrivere nel register file
- RegDest, per indicare il registro da scrivere
- MemToReg, per scrivere il valore nella ALUOut

Passo 5:

execute per l'istruzione lw

OPERAZIONI

- Reg[IR[20:16]] = MDR

S. DI CONTROLLO

- RegWrite, per poter scrivere nel register file
- RegDest, per indicare il registro da scrivere
- MemToReg, per scrivere il valore dalla memoria

In sintesi:

nome	R-type	memory-reference	branches	jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]),2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

ESERCIZI PROBABILI

1. Tutti gli esercizi con il datapath seguono questo schema, in base alla richiesta:

- Scrittura nel register file => add, and, lw, or, slt, sub
- registro B sorgente } add, and, beq, or, slt, sub, sw **ma** se specifica "usato nella ALU", si toglie sw
- contenuto del registro B }
- registro A sorgente => tutte meno j
- due o più somme => tutte le istruzione
- due somme e una sottrazione => sub, beq, slt
- due somme => and, beq, j, or, slt, sub (NO ADD!)
- sottrazione => beq, slt, sub
- tre somme => add, lw, sw

2. Se il registro cause assume il valore 0x00000100, quale eccezione si è verificata?

0x00000100 => 0000 0010 0000 [mantieni i bit da 2 a 6]

↓

0 => interrupt [per conoscere l'eccezione, consultare tabella pg. 34, Appendice A]

3. Se assume valore 0x00000018, quale eccezione si è verificata?

↓

0001 1000

↓

6 => IBE error

tip 0x00000000 => nessuna interruzione / eccezione

4 Considerando di avere una cache di 16 blocchi di 4 words, a che numero di blocco viene mappato l'indirizzo 64 (indirizzo al byte):

Metodo @Koalas

log(blocchi) = bit indice log(16) = 4

log(words) = "offset" log(4) = 2 offset reale = offset + 2 = 2 + 2 = 4

Convertire in binario 64 = 1000000 → dal bit meno significativo, cancello l'offset

↓

0100 → ora si considerano i bit dell'indice, se sono meno si aggiungono 0 a sinistra

↓

4

n°. di blocco

Metodo @Emre

- troviamo l'indice come il metodo precedente
- Ampiezza = words · 4 byte 4 · 4 = 16
- Indirizzo al byte = 64 (da consegna)
- Indirizzo di blocco = $\frac{\text{indirizzo byte}}{\text{byte blocco}} = \frac{64}{16} = 4 \rightarrow 0100$ % blocchi = ris 4 % 16 = 4

5 Data una macchina con 10 CPI per accessi solo in cache, 15 cicli di miss penality e hit probability 0.8, qual è il numero medio di CPI?

hit = 0.8

miss = 1 - hit = 1 - 0.8 = 0.2

numero medio = CPI · hit + CHP · miss =

= 10 · 0.8 + 15 · 0.2 = 8 + 3 = 11

6 Considerando i dati dell'esercizio precedente, con 2.2 GHz di frequenza di clock, qual è la velocità media in istruzioni/secondi?

$$\frac{\text{Frequenza}}{\text{n°. medio}} = \frac{2.2\ \text{GHz}}{11} = \frac{2.2 \cdot 10^9\ \text{Hz}}{11} = \frac{22}{11} \cdot 10^8 = 2 \cdot 10^8 = 200 \text{ milioni}$$

tip scala di multipli e sottomultipli

10^{12}	10^9	10^6	10^3	...	10^{-3}	10^{-6}	10^{-9}	10^{-12}
TERA	GIGA	MEGA	KILO		MILLI	MICRO	NANO	PICO
T	G	M	k		m	μ	n	p

7 Di seguito esercizi simili sulle cache:

7.1 Si supponga di avere una cache a mappaggio diretto con 4 blocchi da 2 words. Se la cache parte vuota e si esegue 2 volte un loop che utilizza le locazioni di memoria 5,2,3,4,4, quante miss ci saranno

4 blocchi
2 words
5,2,3,4,4

	5	2	3	4	4
contenuto →	2	1	1	2	2
blocco ←	2	1	1	2	2

⌊/words⌋ ⌊/2⌋ → % blocchi % 4

BLOCCHI	CONTENUTO				
0					
1		1	1	1	1
2	2	2	2	2	2
3					
	m	m	h	h	h

=> { 2 miss, 3 hit }

7.2 4 blocchi
4 word
6, 9, 11, 14, 1

	6	9	11	14	1
⌊/4⌋	1	2	2	3	0
% 4	1	2	2	3	0
	m	m	h	m	m

=> { 4 miss, 1 hit }

7.3 8 blocchi
2 word
7, 9, 12, 2, 6

	7	9	12	2	6
⌊/2⌋	3	4	6	1	3
% 8	3	0	6	1	3
	m	m	m	m	h

=> { 4 miss, 1 hit }

8 Si supponga di scrivere dati in maniera sequenziale in un vettore in memoria da 0x00100000 a 0x001007FF (estremi inclusi, indirizzamento al byte). Vi sia una cache a mappaggio diretto con 16 blocchi da 4 word; se la cache, inizialmente vuota, alloca sulla scrittura e adotta una politica di write-back, quanti write-back ci saranno dopo aver scritto tutto il vettore?

Altezza = 16 blocchi

Ampiezza = 4 words => 4·4 byte => 16 byte

0x00100000 => 0000 0000 0000 => 0

0x001007FF => 0111 1111 1111 => 2047 + 1 => 2048 (grandezza di tutto il vettore)

$$\frac{\text{dimensione}}{\text{Ampiezza}} = \frac{2048 \text{ byte}}{16 \text{ byte}} = 128 \text{ n°blocchi} = 112$$ [numero di write-back]

9 Si consideri un processore con clock a 10 MHz, in grado di eseguire tutte le istruzioni in 5 cicli di clock. Questo processore ha una periferica che genera dati da 4 byte alla frequenza di 100 kHz. Supponendo che per arrivare all'istruzione che sposta il dato alla periferica ci vogliono 120 istruzioni dell'handler e che dopo il trasferimento ci vogliono altre 80 istruzioni dell'handler, si dica se è possibile gestire il trasferimento o se il processore perde dati:

somma_istruzioni · cicli_clock · freq. periferica
- > freq. cpu => perdita di dati
- = freq. cpu => riesce, ma non avanza il programma
- < freq. cpu => si riesce a fare tutto

(80+120) · 5 · 0.1 MHz = 100 MHz > 10 MHz => perdita di dati

Empiricamente:

- CPU > periferica => si riuscirà anche...
- CPU < periferica => Non è possibile...
- CPU = periferica => In linea teorica...

10 Si supponga che un programma impieghi 100 ms per essere eseguito su una macchina sprovvista di cache e con un tempo di accesso alla memoria RAM di 50ns. Si supponga di eseguire lo stesso programma su una macchina con lo stesso tempo di memoria RAM e con una cache con tempo di accesso di 1ns (10^{-9}). Se la hit rate è 0.95, qual è il tempo di esecuzione del programma?

Troviamo N, n°. di accessi in memoria $\frac{100\,ms}{50\,ns} = \frac{100\,ms}{5 \cdot 10^{-5}\,ms} = 2000000$

Quindi abbiamo, con la cache: $\%\,hit \cdot accesso\_cache + \%\,miss \cdot accesso\_ram =$

$= 2000000 \cdot 0.95 \cdot 1\,ns + 2000000 \cdot 0.05 \cdot 50\,ns =$

$= 1900000 + 5000000 = 6900000\,ns \simeq 7\,ms$

11 Una CPU MIPS esegue una istruzione ogni ciclo di clock, il clock della CPU è 500 MHz, il device emette al massimo 1Kb/s, impostare i registri del DMA occupa 10 cicli di clock; Si vuole trasmettere un testo di $2 \cdot 10^6$ byte. Qual è il rapporto dei tempi di impiego della CPU nel caso di DMA e di I/O a controllo di programma?

CDP: $t_c = \frac{2 \cdot 10^6 \text{ byte}}{1\,Kb/s} = \frac{2000\,Kb}{1\,Kb/s} = 2000\,s$

DMA: $t_1 = 10 \cdot \frac{1}{500\,MHz} = 2 \cdot 10^{-8}$ $\quad t_d = \frac{DIM}{t_1} = \frac{2 \cdot 10^6}{2 \cdot 10^{-8}} = 10^{14}$

Rapporto: $\frac{10^{14}}{2 \cdot 10^3} = 10^{11}$

12 Calcolare il tempo di trasferimento con controllo di programma, avendo 2000 Kb da trasmettere e una banda passante di 1Kb/s:

$T = \frac{\text{byte da trasmettere}}{\text{banda passante}} = \frac{2000\,Kb}{1\,Kb/s} = 2000\,s$

13 Calcolare il tempo di trasferimento con DMA, sapendo che impostare i registri del DMA occupa 10 cicli di clock, il clock ha una frequenza di 500 MHz:

$T_{ogni\ ciclo} = \frac{1}{f}$

$T_{trasf} = N°\,cicli \cdot T_{ogni\ ciclo}$

$10 \cdot \frac{1}{500\,MHz} = \frac{1}{50\,MHz} = \frac{1}{5 \cdot 10^7} = 0.2 \cdot 10^{-7} = 2 \cdot 10^{-8}$

14 Quanti bit sono necessari per una cache a mappaggio diretto con 16 KByte di dati e blocchi da 4 word, assumendo un indirizzamento a 32 bit?

4 word = 16 byte per blocco

$$n^\circ \text{ di blocchi} = \frac{\text{dati}}{\text{byte per blocco}} = \frac{16 \text{ Kbyte}}{16 \text{ byte}} = 1000$$

$$\log(1000) = 10 \Rightarrow 2^{10}$$

n°. di bit per il tag = 32 - 10 - 4 = 18

n°. di bit totali = n°. blocchi totali (n°word · 32 + bit tag + bit validità) =
$= 2^{10}(4 \cdot 32 + 18 + 1) = 147$ Kbit

15 Data una cache a mappaggio diretto con 8 blocchi di 2 word, sequenza indirizzi dal blocco 4, partendo da 0

blocco 4 => n°. 3

8 byte ogni blocco:

BLOCCO	INIZIALE
0	0
1	8
2	16
3	24

=> da 24 in poi

alternativa:

$\frac{x}{8} = 3 \quad x = 24$

x: ind. al byte; 8: grandezza; 3: blocco