Tutorato Architettura degli Elaboratori Modulo 2
Lezione 2

Davide Cecchini

31 Marzo 2022

1 Caching

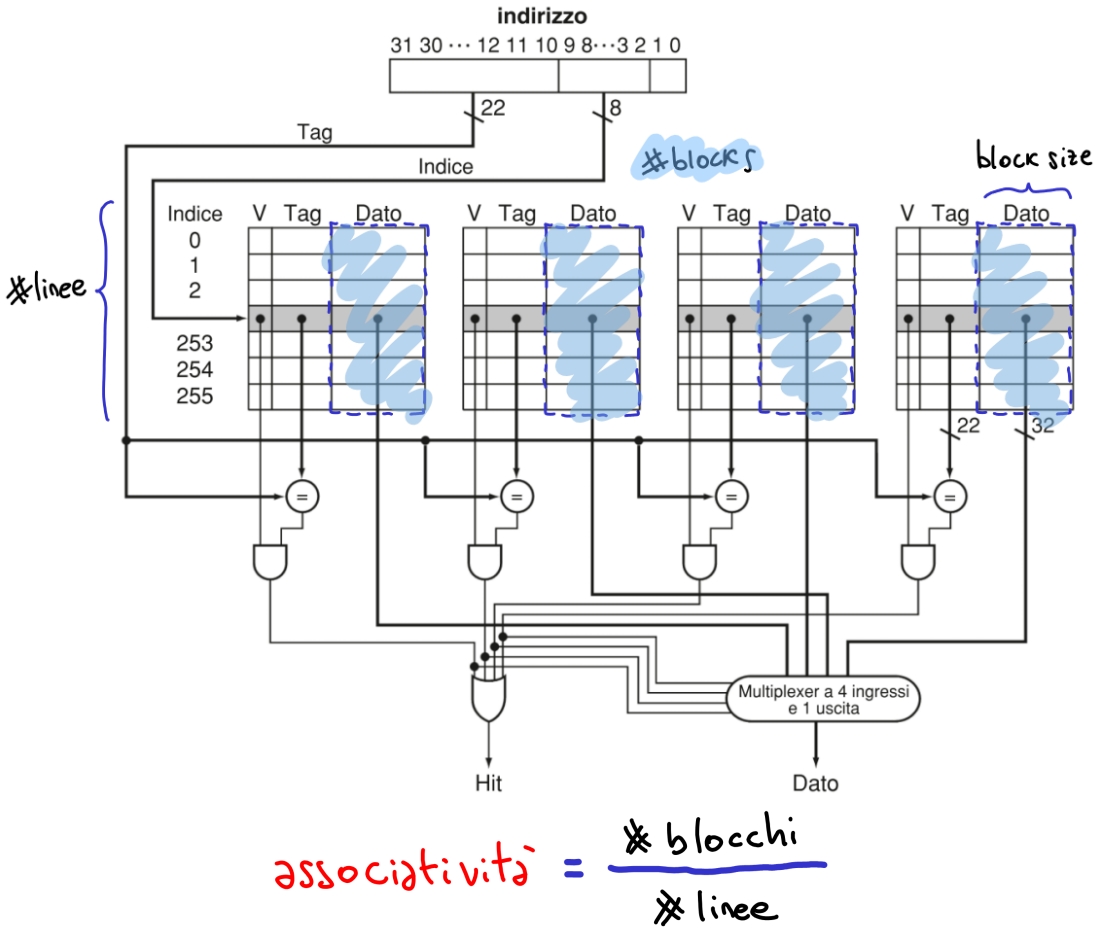

Figura 1: Esempio 4-way associative cache + reminder associatività

Esercizio 1

I seguenti gruppi di bit degli indirizzi su 32 bit, suddivisi nei diversi campi, vengono utilizzati per l'accesso a una cache a mappatura diretta.

Tag	Index	Offset
31-10	9-5	4-0

1. Quanti blocchi contiene la cache?
2. Qual è la dimensione di un blocco della cache (in numero di words)?
3. Qual è il rapporto tra il numero totale di bit contenuti in questa cache e il numero di bit utilizzati per memorizzare i dati?

Soluzione

1. La composizione degli indirizzi è la seguente:
   - `OFFSET`: 5 bit
   - `INDEX`: 5 bit
   - `TAG`: 22 bit

   Essendo la cache ad accesso diretto, abbiamo che il numero di blocchi della cache corrisponde al numero delle entrate (linee). Quindi $2^5 = 32$ blocchi della cache.
2. Siccome abbiamo 5 bit di `OFFSET` ogni blocco contiene $2^5 = 32$ Byte (ricordiamo che l'indirizzamento nel MIPS avviene al Byte). Abbiamo quindi $2^5 \cdot 2^3 = 2^8$ bit totali per ogni blocco della cache. Ossia, passando da bit a words, abbiamo $2^8/32 = 8$ words per blocco.
3. I dati contenuti nella cache sono: $2^5 \cdot 2^5$ Byte ossia $2^5 \cdot 2^5 \cdot 2^3 = 2^{13}$ bit che corrispondono a 8 kb. Per ricavare la quantità di dati necessari per la memorizzazione, dobbiamo moltiplicare i bit di `TAG` e di Validità per il numero di blocchi della cache. Ossia $2^5 \cdot (22 + 1) = 736$ bit. Abbiamo quindi che la dimensione totale è $(2^{13} + 736)/2^{13} = 1.0898$ volte la dimensione dei soli dati.

Esercizio 2

Considerare due cache la cui parte dati è di dimensione 128 kB, con blocchi di 64 bit. Dato un indirizzo fisico di 32 bit, determinare le organizzazioni (suddivisione dei bit dell'indirizzo e livello di associatività) delle cache per dimensioni della `TAG` di 16 bit e di 18 bit.

Soluzione

Tag 16 bit Abbiamo che ogni blocco contiene 64 bit (2^6 bit) ossia 8 Byte ($2^3 \cdot 2^3$ bit). Ricordando che l'indirizzamento nel blocco nel MIPS si effettua al Byte, avremo che il campo `OFFSET` dell'indirizzo sarà composto da 3 bit. Il campo `INDEX` sarà dunque $32 - 3 - 16 = 13$ bit. Significa che vi saranno presenti 2^{13} linee della cache.

Per quanto riguarda il livello di associatività, dobbiamo prima ricavare il numero di blocchi della cache. Ricaviamo il numero totale di blocchi dividendo il totale dei dati (128 kB) per la dimensione di un blocco (64 bit). Trasformiamo la scala dei dati da kB a bit nel seguente modo: 128 kB $= (128 \cdot 1024 \cdot 8)$ bit. Ora possiamo calcolare il numero di blocchi:

$$\#blocchi = \frac{128 \cdot 1024 \cdot 8}{64} = \frac{2^7 \cdot 2^{10} \cdot 2^3}{2^6} = 2^{14}$$

Il livello di associatività sarà dato da $2^{14}/2^{13} = 2$. La cache quindi è una 2-way associative cache.

Tag 18 bit Il campo `OFFSET` non cambia e richiede sempre 3 bit. Cambia invece il campo `INDEX` che sarà formato da $32 - 3 - 18 = 11$ bit. Significa che in questo caso $\#linee = 2^{11}$.

Il livello di associatività sarà $2^{14}/2^{11} = 2^3 = 8$. La cache, dunque, è una 8-way associative cache.

Esercizio 3

Si supponga di avere una cache con una parte dati di 64 kB, avente blocchi da 16 Byte. L'organizzazione della cache può essere diretta (a) o 2-way associative (b).

1. Calcolare la dimensione dell'`INDEX`, `TAG` e `OFFSET` per il caso (a) e (b) nel caso di indirizzi fisici a 32 bit.

2. Rispetto alla cache diretta, si supponga che questa sia all'inizio vuota. Si supponga che un programma faccia riferimento alla memoria con un puntatore p, inizialmente $p = 0x17100004$ (indirizzo fisico di 32 bit espresso in esadecimale), via via p viene incrementato del valore 0x00000080 (ossia 128 in decimale). Valutare per quale valore dell'indirizzo (e per quale corrispondente valore di `INDEX`) si verifica il primo conflitto.

Soluzione

1. (a) **Cache diretta.** Ricaviamo la dimensione della parte dati in bit: $64\,\text{kB} = 2^6 \cdot 2^{10} \cdot 2^8 = 2^{19}$. Ricaviamo la dimensione del blocco in bit: $16\,\text{Byte} = 2^4 \cdot 2^3 = 2^7$.
   Ora possiamo ricavare il numero di blocchi di cui è composta la cache, nello specifico sarà: $\#blocks = 2^{19}/2^7 = 2^{12} = 4096$ blocchi. Considerando che la cache è diretta abbiamo un blocco per linea quindi la dimensione dell'`INDEX` sarà di 4096 entrate ossia 2^{12} che corrispondono a 12 bit nell'indirizzo. Sappiamo inoltre che la dimensione del singolo blocco è di 16 Byte, sono necessari, dunque, 4 bit di `OFFSET` per indirizzare il byte corretto nel blocco. Possiamo ora ricavare la dimensione dei bit di `TAG`, che sarà $32 - 12 - 4 = 16$.

   (b) **2-way associative cache.** In questo caso abbiamo che per ogni linea della cache abbiamo due blocchi, ossia: $2^7 \cdot 2^1 = 2^8$. Possiamo dunque ricavare il numero delle linee della cache come segue $\#linee = 2^{19}/2^8 = 2^{11}$. Per indirizzare 2^{11} entries abbiamo bisogno di 11 bit per l'`INDEX`. La dimensione dei bit dell'`OFFSET` non cambia mentre la dimensione del `TAG` diventa 17 bit.

2. Considerando la cache ad accesso diretto abbiamo che ogni indirizzo è suddiviso nel seguente modo:

$$0x\,\underbrace{1710}_{TAG}\,\overbrace{000}^{INDEX}\,\underbrace{4}_{OFFSET}$$

   Sappiamo che ci sarà un conflitto nella cache quando due blocchi, indicizzati sullo stesso `INDEX`, vengono acceduti. Il programma accede alla cache con la seguente sequenza di indirizzi fisici:

```
0x 1710 000 4  //INDEX= 0
0x 1710 008 4  //INDEX= 8
0x 1710 010 4  //INDEX=16
0x 1710 018 4  //INDEX=24
0x 1710 020 4  //INDEX=32
0x 1710 028 4  //INDEX=40
0x 1710 030 4  //INDEX=48
0x 1710 038 4  //INDEX=56
0x 1710 040 4  //INDEX=64
...
0x ???? ??? ?  //INDEX= 0  Conflitto!
```

   Tenendo conto che l'incremento di 0x80 corrisponde ad un crimento di 8 dell'`INDEX`, e sapendo che la dimensione della cache è $4096 = 2^{12}$ linee, significa che il numero di accessi prima che si verifichi un conflitto[1] sarà $n \cdot 2^3 = 2^{12}$ ossia $n = 2^9 = 0x200$.

   Tradotto in esadecimale l'indirizzo del primo conflitto sarà:

[1] Nell'esercizio consideriamo il numero di accessi necessari prima del primo conflitto, più in generale i conflitti sono dati dagli n che soddisfano l'equazione

$$0x17100004 + 0x200 \cdot 0x80 = 0x17100004 + 0x10000 = 0x17110004$$

index	V	Tag	data
0x000	0	/	/
...	...	...	...
0x008	0	/	/
...	...	...	...
0x010	0	/	/
...	...	...	...
0x018	0	/	/
...	...	...	...
...	...	...	...
...	...	...	...
0xff8	0	/	/

(a) stato iniziale

index	V	Tag	data
0x000	1	0x1710	.....
...	...	...	...
0x008	0	/	/
...	...	...	...
0x010	0	/	/
...	...	...	...
0x018	0	/	/
...	...	...	...
...	...	...	...
...	...	...	...
0xff8	0	/	/

(b) stato dopo l'accesso a 0x17100004

index	V	Tag	data
0x000	1	0x1710	.....
...	...	...	...
0x008	1	0x1710	.....
...	...	...	...
0x010	0	/	/
...	...	...	...
0x018	0	/	/
...	...	...	...
...	...	...	...
...	...	...	...
0xff8	0	/	/

(c) stato dopo l'accesso a 0x17100084

index	V	Tag	data
0x000	1	0x1710	.....
...	...	...	...
0x008	1	0x1710	.....
...	...	...	...
0x010	1	0x1710	.....
...	...	...	...
0x018	1	0x1710	.....
...	...	...	...
...	...	...	...
...	...	...	...
0xff8	1	0x1710	.....

(d) stato dopo l'accesso a 0x1710ff84

index	V	Tag	data
0x000	1	0x1711	.....
...	...	...	...
0x008	1	0x1710	.....
...	...	...	...
0x010	1	0x1710	.....
...	...	...	...
0x018	1	0x1710	.....
...	...	...	...
...	...	...	...
...	...	...	...
0xff8	1	0x1710	.....

(e) stato dopo l'accesso a 0x17110004

Figura 2: Stato della cache durante l'esecuzione del programma

Nella figura 2 possiamo vedere quello che succede nella cache. All'inizio la cache è vuota (Figura 2a), quindi tutti i bit di Validità sono settati a 0. Viene effettuato il primo accesso usando l'indirizzo 0x17100004, l'`INDEX` è 0x000 quindi verrà utilizzata la prima linea (set) della cache. La CPU nota che il bit di Validità è settato a 0, quindi deve recuperare il blocco in memoria (cache miss) e portarlo nella cache. Dopo che la CPU ha fatto ciò, la cache appare come in figura 2b.

Il puntatore p viene incrementato di 0x80, il nuovo indirizzo a cui accedere è 0x17100084. Il procedimento è lo stesso che abbiamo appena visto, però questa volta l'`INDEX` è 0x008, quindi utilizziamo la linea con indice 8. Dopo aver portato il secondo blocco in cache, lo stato è quello della figura 2c.

Il programma procede in questo modo fino a raggiungere (e riempire la linea di cache relativa) l'indirizzo 0x1710ff84 (`INDEX`: 0xff8). Dopo aver portato anche questo blocco in cache, lo stato sarà quello della Figura 2d.

Il puntatore p viene nuovamente incrementato 0x80, ora il suo valore sarà 0x17110004. L'`INDEX` risultante è 0x000, la CPU controlla quindi la prima linea della cache (`INDEX` 0) e nota che il bit di Validità è settato a 1, quindi la linea è occupata da un blocco.

Il controllo del `TAG` causa un conflitto poiché il `TAG` salvato nella cache è 0x1710 mentre quello del puntatore è 0x1711. In questo caso la CPU procede a risolvere il conflitto, aggiornando il valore in memoria del blocco "vecchio" se necessario (indirizzo 0x17100004, vedi write-back/write-through) e importando il blocco "nuovo" (indirizzo 0x17110004) nella cache. Lo stato finale, dopo questo accesso, sarà quello mostrato in figura 2e.

2 Prestazioni Caching

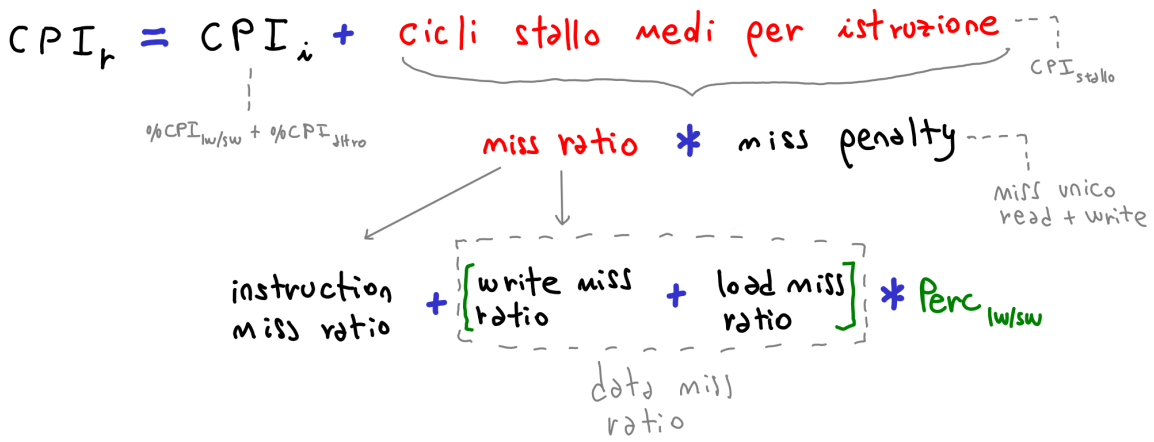

Figura 3: Reminder relazione fondamentale.

Esercizio 4

Considerare l'esecuzione di un programma P su di una data CPU. Calcolare il CPI ideale (CPI_i), considerando che, senza gli effetti della cache, il CPI medio delle load/store sarebbe 4.5, il CPI medio delle altre istruzioni sarebbe 2, mentre la percentuale di load/store è del 40%. Considerando i miss della cache si ottiene un CPI reale (CPI_r) pari a 3.6. Sapendo che $InstructionMissRate = 4\%$, $DataMissRate = 2.5\%$, determinare il $MissPenalty$ (in cicli). Calcolare i tempi, reali e ideali, per eseguire P, considerando che $IC = 200$ Milioni, mentre la frequenza della CPU è di 500 MHz.

Soluzione

Iniziamo ricavando il CPI_i considerando le istruzioni di load/store e le restanti istruzioni.

$$\begin{aligned} CPI_i =& 0.4 \cdot CPI_{lw,sw} + 0.6 \cdot CPI_{altro} \\ =& (0.4 \cdot 4.5) + (0.6 \cdot 2) = 3 \end{aligned}$$

Per alleggerire la notazione sia $x = MissPenalty$. Abbiamo che:

$$\begin{aligned} CPI_r =& CPI_i + 0.04 \cdot x + 0.4 \cdot 0.025 \cdot x \\ 3.6 =& 3 + 0.04 \cdot x + 0.4 \cdot 0.025 \cdot x \\ x =& \frac{0.6}{0.04 + 0.4 \cdot 0.025} = 12 \text{ cicli} \end{aligned}$$

Sapendo che la CPU è a 500 MHz, allora:

$$T = \frac{1}{500 \cdot 10^6}$$

Il tempo di esecuzione ideale è quindi:

$$\begin{aligned} T_{exe_i} =& CPI_i \cdot IC \cdot T \\ =& 3 \cdot 200 \cdot 10^6 \cdot \frac{1}{500 \cdot 10^6} \\ =& \frac{600}{500} = 1.2 sec. \end{aligned}$$

Mentre il tempo di esecuzione reale è:

$$\begin{aligned} T_{exe} =& CPI_r \cdot IC \cdot T \\ =& 3.6 \cdot 200 \cdot 10^6 \cdot \frac{1}{500 \cdot 10^6} \\ =& \frac{720}{500} = 1.44 sec. \end{aligned}$$

Esercizio 5

Abbiamo le seguenti misure relative all'esecuzione di un certo programma su un processore a 2 GHz:

- $CPI_i = 2$
- $Perc_{lw,sw} = 20\%$
- $DataMissRate = 30\%$
- $InstructionMissRate = 5\%$
- $IC = 10$ Milioni
- $T_{exe} = 65ms$

1. Si richiede di calcolare il $MissPenalty$ espresso in ns.
2. Modificando la cache, e mantenendo inalterato il resto del sottosistema di memoria, si osserva un miglioramento del $DataMissRate$. Se otteniamo un tempo di esecuzione $T_{exe} = 40ms$, quanto vale il nuovo $DataMissRate$?

Soluzione

1. Per alleggerire la notazione poniamo $x = MissPenalty$.
   Sappiamo che:

$$\begin{aligned} CPI_{stallo} =& (InstructionMissRate + DataMissRate \cdot Perc_{lw,sw}) \cdot x \\ =& (0.05 + 0.06)x = 0.11x \end{aligned}$$

   Possiamo ora ricavare il CPI_r:

$$CPI_r = CPI_i + CPI_{stallo} = 2 + 0.11x$$

   A questo punto sembrerebbe impossibile ricavare x in quanto non sappiamo quanto è CPI_r. Guardando bene i dati del problema possiamo sfruttare T_{exe} per proseguire.
   Ricordiamo che $T_{exe} = CPI_r \cdot IC \cdot T$ di conseguenza abbiamo:

$$T_{exe} = CPI_r \cdot IC \cdot T = (2 + 0.11x) \cdot 10^7 \cdot T$$

   Ricaviamo T dalla frequenza del processore (2GHz), nel seguente modo:

$$T = \frac{1}{2 \cdot 10^9} = 0.5ns$$

   Ora possiamo ricavare x dall'equazione, visto che $T_{exe} = 65ms$. Infatti:

$$\begin{aligned} T_{exe} =& CPI_r \cdot IC \cdot T \\ 65 \cdot 10^{-3} =& (2 + 0.11x) \cdot 10^7 \cdot 0.5 \cdot 10^{-9} \\ 65 =& (2 + 0.11x) \cdot 5 \\ 65 =& 10 + 0.55x \\ x =& \frac{65 - 10}{0.55} = 100 \text{ cicli} \end{aligned}$$

Per esprimere il miss penalty in ns, basta moltiplicare per T:

$$MissPenalty \cdot T = 100 \cdot 0.5ns = 50ns$$

Alternativamente si può procedere calcolando prima CPI_r dalla formula:

$$T_{exe} = IC \cdot CPI_r \cdot \frac{1}{Freq}$$

$$CPI_r = \frac{T_{exe} \cdot Freq}{IC} = \frac{65 \cdot 10^{-3} \cdot 2 \cdot 10^9}{10^7} = 13 \; cicli$$

E successivamente utilizzare la seguente formula per ottenere il $MissPenalty$:

$$CPI_r = CPI_i + InstructionMissRate \cdot MissPenalty + Perc_{lw,sw} \cdot DataMissRate \cdot MissPenalty$$

$$MissPenalty = \frac{CPI_r - CPI_i}{InstructionMissRate + Perc_{lw,sw} \cdot DataMissRate} =$$
$$= \frac{13 - 2}{0.05 + 0.2 \cdot 0.3} = \frac{11}{0.11} = 100 \; cicli$$

2. Sia x il nuovo valore del $DataMissRate$:

$$CPI_{stallo} = (InstructionMissRate + x \cdot Perc_{lw,sw}) \cdot MissPenalty = (0.05 + 0.2x)100 = 5 + 20x$$

$$CPI_r = CPI_i + CPI_{stallo} = 2 + (5 + 20x) = 7 + 20x$$

Sfruttiamo ancora la definizione di T_{exe}.

$$\begin{aligned} T_{exe}^{'} &= CPI_r \cdot IC \cdot T \\ &= (7 + 20x) \cdot 10^7 \cdot 0.5 \cdot 10^{-9} \\ &= (7 + 20x) \cdot 5 \cdot 10^6 \cdot 10^{-9} \\ &= (7 + 20x) \cdot 5 \cdot 10^{-3} \end{aligned}$$

Metto nell'equazione il tempo di esecuzione dato dalla consegna e calcolo il nuovo $DataMissRate$ risolvendo l'equazione per x.

$$\begin{aligned} T_{exe}^{'} &= (7 + 20x) \cdot 5 \cdot 10^{-3} \\ 40 \cdot 10^{-3} &= 35 \cdot 10^{-3} + 10^{-1}x \\ 10^{-1}x &= 5 \cdot 10^{-3} \\ x &= 5 \cdot 10^{-2} = 0.05 = 5\% \end{aligned}$$

Il nuovo $DataMissRate$ è del 5%.

Anche in questo caso si può usare un metodo alternativo. Prima si usa la formula di T_{exe} per calcolare CPI_r:

$$T_{exe} = IC \cdot CPI_r \cdot \frac{1}{Freq}$$

$$CPI_r = \frac{T_{exe} \cdot Freq}{IC} = \frac{40 \cdot 10^{-3} \cdot 2 \cdot 10^9}{10^7} = 8 \; cicli$$

E poi si usa la formula di CPI_r per calcolare id $DataMissRate$:

$$CPI_r = CPI_i + InstructionMissRate \cdot MissPenalty + Perc_{lw,sw} \cdot DataMissRate \cdot MissPenalty$$

$$DataMissRate = \frac{CPI_r - CPI_i - InstructionMissRate \cdot MissPenalty}{Perc_{lw,sw} \cdot MissPenalty} =$$
$$= \frac{8 - 2 - 0.05 \cdot 100}{0.2 \cdot 100} = \frac{1}{20} = 0.05 = 5\%$$

Esercizio 6

Un computer a 1 GHz, nell'eseguire un certo programma, ha una prestazione ideale di 500 MIPS (senza considerare la cache e i miss relativi). Considerando la cache, il CPI_r misurato diventa uguale a 2.48. Conoscendo che il data e l'instruction miss rate sono entrambi uguali al 2% e che la percentuale di load/store è del 20%, si chiede di:

1. Calcolare il CPI_i
2. Calcolare il $MissPenalty$

Soluzione

1. Poiché la prestazione ideale è di 500 MIPS ed il processore è a 1GHz significa che, in media, ci impiegheremmo 2 cicli a completare una istruzione[2].

   Possiamo riformulare più formalmente la soluzione, in particolare ricordandoci che :

   $$MIPS = \frac{IC}{T_{exe_i} \cdot 10^6}$$

   e che:

   $$T = \frac{1}{F}$$

   abbiamo:

   $$\begin{aligned} MIPS =& \frac{IC \cdot F}{IC \cdot CPI_i \cdot 10^6} = \frac{F}{CPI_i \cdot 10^6} \\ 500 =& \frac{10^9}{CPI_i \cdot 10^6} \\ 500 =& \frac{10^3}{CPI_i} \\ CPI_i =& \frac{10^3}{500} = 2 \end{aligned}$$

2. Quello appena calcolato è il CPI_i, possiamo dunque ricavare $MissPenalty$ nel seguente modo:

   $$\begin{aligned} CPI_{stallo} =& CPI_r - CPI_i \\ =& 2.48 - 2 \\ =& 0.48 \end{aligned}$$

   Da quì possiamo sfruttare il fatto che $CPI_{stallo} = MissRatio \cdot MissPenalty$ e ricavare $MissPenalty$ attraverso la composizione di $MissRatio$, in particolare:

   $$\begin{aligned} 0.48 =& (2\% + 2\% \cdot 20\%) \cdot MissPenalty \\ 0.48 =& (0.02 + 0.02 \cdot 0.2) \cdot MissPenalty \\ MissPenalty =& 20 \end{aligned}$$

Nota Bene: differenze tra kB e KiB

In questa lezione abbiamo considerato 1 kB = 2^{10} Byte. Qesta NON sarebbe la definizione standard. Si tratta di una convenzione che viene usata comunemente dagli informatici per facilitare i conti. Infatti:

- Definizione standard: 1 kB = 1000 Byte.
- Definizione NON standard: 1 kB = 1024 Byte.

[2] La capacità di esecuzione è di 1 miliardo di istruzioni al secondo, se la prestazione ideale prevede 500 milioni di istruzioni al secondo, significa che ogni due cicli finiamo un'instruzione.

Il nome corretto per la definizione non standard sarebbe il kibibyte (KiB). Di fatto, nell'ambito dell'informatica, si scrive impropriamente kB per intendere KiB. Da poco il termine kibibyte sta iniziando ad essere sempre più comune, questo è importante in quanto la differenza tra 1 kB e 1 KiB non è trascurabile (sono 192 bit persi per sempre!). Lo stesso discorso si estende ai MB ed ai MiB e via dicendo.

Risorse

- Wikipedia: kB vs. KiB, prefissi binari vs decimali
- Struttura e progetto dei calcolatori - David A. Paterson, John L. Hennessy, Capitolo 5.