Architettura degli Elaboratori (modulo II e B) (Compito 24 Giugno 2009)

Usare un foglio separato per rispondere a due delle domande seguenti, specificando nell'intestazione: **Titolo del corso** (*Architettura degli Elaboratori – modulo II* oppure *Architettura degli Elaboratori B*), **Data esame**, **Cognome e Nome**, **Matricola**

Esercizio 1

Calcolare *numero di insiemi* e *livello di associatività* di una cache di cui siano noti: Cache size=1 MB, TAG size=13 b, PHYSICAL ADDR size = 31 b, INDEX size = 11 b.
Sempre rispetto alla solita gerarchia di memoria, individuare la *dimensione dell'indirizzo virtuale*, considerando che la Page Table ha una dimensione pari a 12 MB, ogni ingresso della Page Table è grande 3 B, e include 3 b di valid, dirty e reference.

Soluzione

La cache ha un numero di insiemi pari a $2^{INDEX\_size} = 2^{11}$. Ogni insieme ha una dimensione pari a $Cache\_size/2^{11} = 2^{20}/2^{11} = 2^9\ B$.
Il block offset è uguale a $ADDR\_size - TAG\_size - INDEX\_size = 31 - 13 - 11 = 7\ b$. Quindi la dimensione di ogni blocco è $2^7 = 128\ B$.
Ogni insieme contiene quindi un numero di blocchi pari a $2^9/BLOCK\_size = 2^9/2^7 = 2^2$. Per cui la cache è associativa a *4 vie*.

Poiché ogni ingresso della PT è grande 3 B, eliminando i 3 bit aggiuntivi, il $PHYSICAL\_page\_number$ è grande 24-3= 21 b. Il $Page\_offset$ è $PHYSICAL\_ADDR\_size - PHYSICAL\_page\_number = 31 - 21 = 10$ b.
Il numero di ingressi della PT è $\frac{12\ MB}{3\ B} = 4 * 2^{20} = 2^{22}$. Sono quindi necessari 22 bit per indirizzare la PT, per cui la $VIRTUAL\_page\_number$ è grande 22 b.
Infine, $VIRTUAL\_ADDR\_size = VIRTUAL\_page\_number + Page\_offset = 22 + 10 = 32$ b.

Esercizio 2 (*Modulo II*)

Considerare la seguente porzione di programma assembler MIPS:

```
        addi $s0, $zero, 0
Loop:
        sll  $t0, $s0, 2   # moltiplico per 4
        add  $t1, $a0, $t0
        lw   $t2, 0($t1)
        addi $t2, $t2, 128
        sw   $t2, 0($t1)
        addi $s0, $s0, 1
        bne  $s0, $a1, Loop
```

Il programma è il cuore della procedura `pippo` che riceve in ingresso due parametri (`$a0` e `$a1`).

- Completare la procedura con il salvataggio e il ripristino dei registri callee-save, e con il ritorno.
- Considerando che `$a0` e `$a1` contengono, rispettivamente, l'indirizzo iniziale di un array di interi e la dimensione dell'array stesso, cosa calcola la procedura?
- Consideranche che `$a1=1024`, calcolare il CPI del loop rispetto alla CPU multiciclo vista a lezione (l'esecuzione dell'istruzione `sll` impiega gli stessi cicli della `add`).

Soluzione

- La procedura è la seguente:

```
pippo:
        addi $sp, $sp, -4
        sw   $s0, 0($sp)

        addi $s0, $zero, 0
loop:
        sll  $t0, $s0, 2   # moltiplico per 4
        add  $t1, $a0, $t0
```

```
lw   $t2, 0($t1)
addi $t2, $t2, 128
sw   $t2, 0($t1)
addi $s0, $s0, 1
bne  $s0, $a1, Loop

lw   $s0, 0($sp)
addi $sp, $sp, 4
jr   $ra
```

- Il programma somma la costante 128 agli elementi (interi=4 B) di un array puntato da `$a0`, la cui dimensione è di `$a1` word.

  Il programma C corrispondente è:

```
void pippo(int *a, int dim) {
     int i;
     for (i=0; i < dim; i++)
        a[i] += 128;
}
```

- Il loop è eseguito per 1024 volte. Il numero di istruzioni eseguite è quindi $IC = 1024 * 7 = 7168$. Considerando che tutte le istruzioni impiegano 4 cicli, escluso `bne` (3 cicli) e `lw` (5 cicli), per ogni ciclo del loop si impiegano: $5 * 4 + 3 + 5 = 28$ *cicli*.

  In totale: $Cicli = 1024 * 28 = 28672$.

  Da cui $CPI = 28672/7168 = 4$.

Esercizio 2 (*Arch. B*)

Sappiamo che, per certo mix di programmi e senza considerare gli stalli dovuti ai cache miss, il CPI delle istruzioni di `lw/sw` è 3, mentre il CPI di tutte le altre istruzioni è 2. Le istruzioni di `lw/sw` sono il 25% di tutte le istruzioni eseguite.
Valutare le seguenti architetture, per le quali i CPI di sopra rimangono invariati:

(a) processore a 250 MHz, data miss rate del 5%, instruction miss rate del 2%, e miss penalty di 10 cicli.

(b) processore a 500 MHz, data miss rate del 3%, instruction miss rate del 2%, e miss penalty di 15 cicli.

Quale delle due architetture esegue più cicli?
Qual è più veloce e di quanto?

Soluzione

$No.\ cicli\_no\_miss = IC * (0.25 * CPI_{lw} + 0.75 * CPI_{altro}) = IC * (0.25 * 3 + 0.75 * 2) = IC * 2.25$

$No.\ cicli_{(a)} = No.\ cicli\_no\_miss + (5\%(0.25 * IC) + 2\%IC) * miss\_penalty = IC * 2.25 + IC(0.0125 + 0.02) * 10 = IC * (2.25 + 0.3250) = 2.575 * IC$

$No.\ cicli_{(b)} = No.\ cicli\_no\_miss + (3\%(0.25 * IC) + 2\%IC) * miss\_penalty = IC * 2.25 + IC(0.0075 + 0.02) * 15 = IC * (2.25 + 0.4125) = 2.6625 * IC$

Quindi l'architettura (b) esegue un numero maggiore di cicli.

Rispetto al tempo di esecuzione:
$T\_exe_{(a)} = 2.575 * IC * ciclo\_clock = 2.575 * IC * \frac{1}{250*10^6} = 0.0103 * \frac{IC}{10^6}\ s.$

$T\_exe_{(b)} = 2.6625 * IC * ciclo\_clock = 2.6625 * IC * \frac{1}{500*10^6} = 0.005325 \frac{IC}{10^6}\ s.$

Da cui l'architettura (b) risulta essere più veloce, con speedup:
$Speedup = \frac{T\_exe_{(a)}}{T\_exe_{(b)}} = 0.0103/0.005325 = 1.934.$