**COMPITINO #2 di CALCOLATORI ELETTRONICI del 29-11-01** **MATRICOLA**_____

**COGNOME**_____

**NOME**_____

[ Versione vista durante l'esercitazione del 17/12/2015 ]

Si consideri il seguente programma assembly in cui alla partenza $2=0x100, $4=4, $10=10:

```
    add   $1,   $0,  $0
    add   $5,   $0,  $0
L1: lw    $6, 0($2)
    add   $1,   $1,  $6
    add   $2,   $2,  $4
    addi  $5,   $5,  1
    bne   $5,  $10,  L1
    div   $1,  $10
    mflo  $3
L2: lw    $6, -4($2)
    div   $6,   $3
    mflo  $6
    sw    $6, -4($2)
    sub   $2,   $2,  $4
    addi  $5,   $5,  -1
    bne   $5,   $0,  L2
```

Si supponga che tale programma vada in esecuzione su un calcolatore con processore MIPS a 32-bit (senza pipeline) con frequenza di clock pari a 2GHz. Le istruzioni aritmetico/logiche/jump richiedono 1 ciclo, i salti (branch) 3 cicli e le operazioni di load/store 1 ciclo oltre ai cicli necessari per accedere alla gerarchia di memoria.

Si calcoli il tempo di esecuzione di questo programma nei seguenti casi:

a) nel caso in cui la gerarchia di memoria sia costituita solo dalla memoria principale e questa abbia un tempo di accesso di 50 ns;
b) nel caso in cui nella gerarchia di memoria sia interposto uno write-buffer fra processore e memoria principale. Lo write-buffer consente di memorizzare 16 parole e il tempo di accesso allo write-buffer corrisponde a un ciclo;
c) nel caso in cui nella gerarchia di memoria sia interposta una split-cache (al posto dello write-buffer) fra processore e memoria prinicipale. La split-cache presenta un tempo di accesso corrispondente a un ciclo. Per la cache istruzioni si assuma un miss rate dell' 1% , mentre la cache dati e' set associative a 2 vie di dimensioni 1KB e con blocco di dimensioni 8 Byte.

Riepilogo del significato delle istruzioni

| Instruction | Example | Meaning | Comments |
|---|---|---|---|
| add | `add $1,$2,$3` | $1 = $2 + $3 | 3 operands; exception possible |
| subtract | `sub $1,$2,$3` | $1 = $2 - $3 | 3 operands; exception possible |
| add immediate | `addi $1,$2,100` | $1 = $2 + 100 | + constant; exception possible |
| Division | `div $1, $2` | Hi= $1 % $2, Lo = $1 / $2 | Signed division |
| move from Lo | `mflo $1` | $1 = Lo | Create copy of Lo |
| load word | `lw $1,100($2)` | $1 = Memory[$2+100] | Data from memory to register |
| store word | `sw $1,100($2)` | Memory[$2+100] = $1 | Data from register to memory |
| load upper immediate | `lui $1,100` | $1 = 100 * 256 | Load constant in upper 16bits |
| branch on not equal | `bne $1,$2,100` | if ($1 != $2) go to PC+4+100 | Not equal test; PC relative |

Register Usage

| Name | Register Number | Usage |
|---|---|---|
| $zero | 0 | the constant value 0 |

Si consideri il seguente programma assembly in cui alla partenza $2=0x100, $4=4, $10=10:

```
    add   $1,   $0,  $0      $1 = 0                BB1 x 1
    add   $5,   $0,  $0      $5 = 0
L1: lw    $6, 0($2)      do: $6 = M[$2]            BB2 x 10
    add   $1,   $1,  $6      $1 += $6
    add   $2,   $2,  $4      $2 += $4
    addi  $5,   $5,  1       $5 += 1
    bne   $5,  $10,  L1   while ($5 != $10)
    div   $1,  $10           $1 / $10              BB3 x 1
    mflo  $3                 $3 = LO (mean val)
L2: lw    $6, -4($2)     do: $6 = M[$2-4]          BB4 x 10
    div   $6,   $3           $6 / $3
    mflo  $6                 $6 = LO
    sw    $6, -4($2)         M[$2-4] = $6
    sub   $2,   $2,  $4      $2 -= $4
    addi  $5,   $5,  -1      $5 -= 1
    bne   $5,   $0,  L2   while ($5 != 0)
```

Frequenza di clock CPU: 2*10^9 clk/s
Tempo accesso memoria: 50*10^-9 s == 100 clk

| Basic Block | *Ripe-tizioni* | ALJ (1) | B (3) | LS (1+) | *Cicli solo CPU* | *Cicli Fetch istruzioni (senza cache)* | *Cicli accesso dati (senza cache ne' WB)* | *Cicli solo CPU* | *Cicli Fetch istruzioni (senza cache)* | *Cicli accesso dati (senza cache ma con WB)* | *Cicli solo CPU* | *Cicli Fetch istruzioni (con cache)* | *Hit/Miss in cache dati* | *Cicli accesso dati (con cache e senza WB)* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |
| BB1 | *1* | 2 | 0 | 0 | *2* | *200* | *0* | *2* | *200* | *0* | *2* | *0.01*200* | *0/0* | *0* |
| BB2 | *10* | 3 | 1 | 1 | *70* | *5000* | *1000* | *70* | *5000* | *1000* | *70* | *0.01*5000* | *5/5* | *505* |
| BB3 | *1* | 2 | 0 | 0 | *2* | *200* | *0* | *2* | *200* | *0* | *2* | *0.01*200* | *0/0* | *0* |
| BB4 | *10* | 4 | 1 | 2 | *90* | *7000* | *2000* | *90* | *7000* | *1010* | *90* | *0.01*7000* | *20/0* | *20* |
| **Totale (clk)** | | | | | *164* | *12400* | *3000* | *164* | *12400* | *2010* | *164* | *124* | | *525* |
| **Totale (sec)** | | | | | **CASO a) 15564 cicli (7782 ns)** | | | **CASO b) 14574 cicli (7287 ns)** | | | **CASO c) 813 cicli (406.5 ns)** | | | |

Cache dati
Capacita': 2^10 byte
A: 2 vie
Blocco: 8 byte
→S = C/A/B=64 sets

Note:

- BB2 viene ripetuto 10 volte e contiene una sola load che fa accesso a 10 word consecutive; dato che la cache dati e' sufficientemente grande e ogni blocco contiene 2 word → ho 5 Miss e 5 hit (la seconda via non viene mai usata)
- BB4 viene ripetuto 10 volte e legge e scrive le stesse celle di BB1 ma dall'indirizzo piu' alto verso quello piu' basso → ho 20 hit
- 
- I cicli di accesso alla memoria, anche in caso di hit in cache, sono stati considerati in aggiunta rispetto alla durata di base delle istruzioni.