**Domanda 1**

Considerando il processore MIPS64 e l’architettura descritta in seguito:

|  |  |  |
| --- | --- | --- |
| * + Integer ALU: 1 clock cycle   + Data memory: 1 clock cycle   + FP multiplier unit: pipelined 8 stages | * + FP divider unit: not pipelined unit that requires 8 clock cycles   + FP arithmetic unit: pipelined 4 stages   + branch delay slot: 1 clock cycle, and the branch delay slot disabled | * + forwarding enabled   + it is possible to complete instruction EXE stage in an out-of-order fashion. |

Usando il frammento di codice riportato, si calcoli il tempo di esecuzione dell’intero programma in colpi di clock e si completi la seguente tabella.

; for (i = 0; i < 100; i++) {

; v5[i] = v1[i]/v2[i] + v3[i] + v4[i] ;

;}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| .data |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Clock  cycles |
| V1: .double “100 values” |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V2: .double “100 values” |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V3: .double “100 values”  …  V5: .double “100 zeros” |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V4: .double “100 values” |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| V5: .double “100 values” |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| .text |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| main: daddui r1,r0,0 | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| daddui r2,r0,100 |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| loop: l.d f1,v1(r1) |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| l.d f2,v2(r1) |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| div.d f4,f1,f2 |  |  |  |  | F | D | s | : | : | : | : | : | : | : | : | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 9 |
| l.d f3,v3(r1) |  |  |  |  |  | F | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| add.d f5,f4,f3 |  |  |  |  |  |  |  | F | D | s | s | s | s | s | s | + | + | + | + | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 |
| l.d f4,v4(r1) |  |  |  |  |  |  |  |  | F | s | s | s | s | s | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| add.d f5,f4,f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | s | s | + | + | + | + | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 |
| s.d f5,v5(r1) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | s | D | E | s | s | s | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | s | s | s | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | s | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| Halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Total |  |  |  |  | 6+25\*10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2506 |

**Domanda 2**

Considerando il programma precedente, quali sono le istruzioni che beneficiano principalmente del meccanismo di FORWARDING del processore e perché? motivare la risposta.

Generalmente le istruzioni che beneficiano maggiormente del meccanismo di forwarding sono quelle con stalli dovuti a dipendenze di dato (RAW, WAW) perché possono intercettare un valore subito dopo la fase di execute invece che dopo la fase di write back (2 colpi di clock in meno). Un esempio è add.d f5, f4, f3 che è costretto ad attendere per 6 c.c la conclusione dell’operazione di divisione ma almeno il forwarding riesce a ridurre il numero di stalli rendendo il valore di f4 disponibile appena finisce la fase di execute della divisione.

**Domanda 3**

Considerando il programma precedente e l’architettura del processore superscalare descritto in seguito; completare la tabella relativa alle prime 2 iterazioni.

Processor architecture:

* + Issue 2 instructions per clock cycle
  + jump instructions require 1 issue
  + handle 2 instructions commit per clock cycle
  + timing facts for the following separate functional units:
    1. 1 Memory address 1 clock cycle
    2. 1 Integer ALU 1 clock cycle
    3. 1 Jump unit 1 clock cycle
    4. 1 FP multiplier unit, which is pipelined: 8 stages
    5. 1 FP divider unit, which is not pipelined: 8 clock cycles
    6. 1 FP Arithmetic unit, which is pipelined: 4 stages
  + Branch prediction is always correct
  + There are no cache misses
  + There are 2 CDB (Common Data Bus).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # iteration |  | Issue | EXE | MEM | CDB x2 | COMMIT x2 |
| 1 | l.d f1,v1(r1) | 1 | 2m | 3 | 4 | 5 |
| 1 | l.d f2,v2(r1) | 1 | 3m | 4 | 5 | 6 |
| 1 | div.d f4,f1,f2 | 2 | 6d | - | 14 | 15 |
| 1 | l.d f3,v3(r1) | 2 | 4m | 5 | 6 | 15 |
| 1 | add.d f5,f4,f3 | 3 | 15a | - | 19 | 20 |
| 1 | l.d f4,v4(r1) | 3 | 5m | 6 | 7 | 20 |
| 1 | add.d f5,f4,f5 | 4 | 20a | - | 24 | 25 |
| 1 | s.d f5,v5(r1) | 4 | 6m | - | - | 25 |
| 1 | daddui r1,r1,8 | 5 | 6i | - | 7 | 26 |
| 1 | daddi r2,r2,-1 | 5 | 7i | - | 8 | 26 |
| 1 | bnez r2,loop | 6 | 9j | - | - | 27 |
| 2 | l.d f1,v1(r1) | 7 | 8m | 9 | 10 | 27 |
| 2 | l.d f2,v2(r1) | 7 | 9m | 10 | 11 | 28 |
| 2 | div.d f4,f1,f2 | 8 | 14d | - | 22 | 28 |
| 2 | l.d f3,v3(r1) | 8 | 10m | 11 | 12 | 29 |
| 2 | add.d f5,f4,f3 | 9 | 23a | - | 27 | 29 |
| 2 | l.d f4,v4(r1) | 9 | 11m | 12 | 13 | 30 |
| 2 | add.d f5,f4,f5 | 10 | 28a | - | 32 | 33 |
| 2 | s.d f5,v5(r1) | 10 | 12m | - | - | 33 |
| 2 | daddui r1,r1,8 | 11 | 12i | - | 13 | 34 |
| 2 | daddi r2,r2,-1 | 11 | 13i | - | 14 | 34 |
| 2 | bnez r2,loop | 12 | 15j | - | - | 35 |

**Domanda 4**

Considerando il segmento di codice presentato nella tabella precedente, se assumessimo che il ROB abbia una dimensione di 8 elementi, quale sarebbe la prima istruzione che dovrebbe stallare durante la esecuzione del programma? motivare la risposta.

La prima istruzione che stallerebbe è daddi r2,r2,-1. Si estraggono 9 istruzioni, ma nel ROB ce ne saranno 8 al quinto c.c. perché vi è il commit della prima istruzione.