**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 6 stages | * + FP divider unit: not pipelined unit that requires 6 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++) {

; v4[i] = v1[i]/v2[i] \* v3[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 f3,f1,f2 |  |  |  |  | F | D | s | / | / | / | / | / | / | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 7 |
| l.d f1,v3(r1) |  |  |  |  |  | F | s | D | E | s | s | s | s | S | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| mul.d f3,f3,f1 |  |  |  |  |  |  |  | F | D | s | s | s | s | \* | \* | \* | \* | \* | \* | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
| s.d f3,v4(r1) |  |  |  |  |  |  |  |  | F | s | s | s | s | D | E | s | s | s | s | S | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | s | s | s | s | S | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | s | s | s | S | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 2 |
| Halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | - | - | - | - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Total |  |  |  |  | 6+22\*100 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2206 |

**Domanda 2**

Considerando il programma precedente, e in particolare la coppia di istruzioni:

div.d f3,f1,f2

l.d f1,v3(r1)

che tipo di hazard crea l’utilizzo di f1 e come viene risolto? motivare la risposta.

Si crea un WAR hazard in cui la load deve leggere il dato dopo che sia terminata la fase di execute della div. Si può risolvere con stalli (HW) o register renaming (SW).

**Domanda 3**

Considerando il programma precedente e l’architettura del processore superscalare descritto in seguito; completare la tabella relativa alle prime 3 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: 6 stages
    5. 1 FP divider unit, which is not pipelined: 6 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 | 2ea | 3 | 4 | 5 |
| 1 | l.d f2,v2(r1) | 1 | 3ea | 4 | 5 | 6 |
| 1 | div.d f3,f1,f2 | 2 | 6d-11d |  | 12 | 13 |
| 1 | l.d f1,v3(r1) | 2 | 4ea | 5 | 6 | 13 |
| 1 | mul.d f3,f3,f1 | 3 | 13m-18m |  | 19 | 20 |
| 1 | s.d f3,v4(r1) | 3 | 5ea |  |  | 20 |
| 1 | daddui r1,r1,8 | 4 | 5i |  | 6 | 21 |
| 1 | daddi r2,r2,-1 | 4 | 6i |  | 7 | 21 |
| 1 | bnez r2,loop | 5 | 8j |  |  | 22 |
| 2 | l.d f1,v1(r1) | 6 | 7ea | 8 | 9 | 22 |
| 2 | l.d f2,v2(r1) | 6 | 8ea | 9 | 10 | 23 |
| 2 | div.d f3,f1,f2 | 7 | 12d-17d |  | 18 | 23 |
| 2 | l.d f1,v3(r1) | 7 | 9ea | 10 | 11 | 24 |
| 2 | mul.d f3,f3,f1 | 8 | 19m-24m |  | 25 | 26 |
| 2 | s.d f3,v4(r1) | 8 | 10ea |  |  | 26 |
| 2 | daddui r1,r1,8 | 9 | 10i |  | 11 | 27 |
| 2 | daddi r2,r2,-1 | 9 | 11i |  | 12 | 27 |
| 2 | bnez r2,loop | 10 | 13j |  |  | 28 |
| 3 | l.d f1,v1(r1) | 11 | 12ea | 13 | 14 | 28 |
| 3 | l.d f2,v2(r1) | 11 | 13ea | 14 | 15 | 29 |
| 3 | div.d f3,f1,f2 | 12 | 18d-23d |  | 24 | 29 |
| 3 | l.d f1,v3(r1) | 12 | 14ea | 15 | 16 | 30 |
| 3 | mul.d f3,f3,f1 | 13 | 25m-30m |  | 31 | 32 |
| 3 | s.d f3,v4(r1) | 13 | 15ea |  |  | 32 |
| 3 | daddui r1,r1,8 | 14 | 15i |  | 16 | 33 |
| 3 | daddi r2,r2,-1 | 14 | 16i |  | 17 | 33 |
| 3 | bnez r2,loop | 15 | 18j |  |  | 34 |

**Domanda 4**

Considerando il segmento di codice presentato nella tabella precedente, se assumiamo che il ROB ha una dimensione di 16 elementi e alla partenza del codice in tabella è vuoto, qual è la prima istruzione che dovrebbe stallare durante l’esecuzione del programma? motivare la risposta.

Il ROB si dovrebbe riempire all’istruzione 16 e al CC9, intanto però ha liberato 2 posti dalle commit delle prime istruzioni ai CC 5 e 6 quindi fa spazio ad altre due istruzioni e la prima a stallare è la 19esima:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3 | l.d f1,v1(r1) | 11 | 12ea | 13 | 14 | 28 |

in quanto siamo al CC11 e le prime del ROB verranno committate al CC13.