**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 | s | \* | \* | \* | \* | \* | \* | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
| s.d f3,v4(r1) |  |  |  |  |  |  |  |  | F | s | 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+100\*22 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 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.

è un DATA HAZARD (istruzioni successive vogliono accedere/modificare i dati prima che il valore nuovo diventi effettivo); qui accade perché la load vuole modificare f1, mentre questo sta venendo usato dalla div precedente. In questo caso è un WAR hazard (scrivo in f1 mentre questo sta ancora venendo usato). Viene risolto, senza cambiare architettura, usando scheduling (ovvero sposto delle istruzioni che non dipendono dagli operandi usati dalla div e dalla load tra di esse, facendo padding e riducendo/evitando gli stalli) o register renaming (uso altro registro nella load e nelle istruzioni dopo la load in modo che non ci sia dipendenza tra div e load)

**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 | / | 9 | 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.

La prima esecuzione in stallo dovrebbe essere la 17esima (ovvero DADDI r2, r2, -1 alla 2^ iterazione), ma quando avviene la ISSUE di questa istruzione (ovvero al clock 9), nel mentre il ROB si è svuotato delle istruzioni che fanno commit prima del clock 9, ovvero le 2 load iniziali. Quindi controllo che la 19esima, ovvero l.d f1,v1(r1) della 3^ iterazione (14 + 2 posti nuovi del ROB) abbia issue prima di 13 (ovvero il commit della 1^ istruzione del ROB) e vedo che ha issue 11, quindi è lei la prima a stallare.