## 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++) {

; 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 |
| add.d f3,f1,f2 |  |  |  |  | F | D | s | + | + | + | + | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 5 |
| l.d f1,v3(r1) |  |  |  |  |  | F | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| l.d f2,v4(r1) |  |  |  |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0 |
| add.d f4,f1,f2 |  |  |  |  |  |  |  |  | F | D | s | + | + | + | + | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 4 |
| mul.d f4,f4,f3 |  |  |  |  |  |  |  |  |  | F | s | D | s | s | s | \* | \* | \* | \* | \* | \* | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 6 |
| s.d f4,v4(r1) |  |  |  |  |  |  |  |  |  |  |  | F | 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\*23 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2306 |

**Domanda 2**

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

l.d f2,v2(r1)

add.d f3,f1,f2

che tipo di hazard crea l’utilizzo di f2 e come viene risolto? motivare la risposta.

DATA HAZARD, in particolare RAW (Read After Write) HAZARD in quanto f2 viene prima scritto dalla load e quindi la add.d deve aspettare che finisca lo stadio di Mem della l.d per avere l’operando. Viene infatti messo uno stallo in modo che la E della add parta un clock dopo la M della load (si potrebbe risolvere anche attraverso rescheduling se possibile, in quanto potremmo mettere un’istruzione [es. la daddi su r2 in quanto indipendente dalle 2 istruzioni in questione] che faccia da cuscinetto/padding in modo da evitare lo stallo]).

**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: 7 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 | add.d f3,f1,f2 | 2 | 6a-9a | / | 10 | 11 |
| 1 | l.d f1,v3(r1) | 2 | 4ea | 5 | 6 | 11 |
| 1 | l.d f2,v4(r1) | 3 | 5ea | 6 | 7 | 12 |
| 1 | add.d f4,f1,f2 | 3 | 8a-11a | / | 12 | 13 |
| 1 | mul.d f4,f4,f3 | 4 | 13m-18m | / | 19 | 20 |
| 1 | s.d f4,v4(r1) | 4 | 6ea | / | / | 20 |
| 1 | daddui r1,r1,8 | 5 | 6i | / | 7 | 21 |
| 1 | daddi r2,r2,-1 | 5 | 7i | / | 8 | 21 |
| 1 | bnez r2,loop | 6 | 9j | / | / | 22 |
| 2 | l.d f1,v1(r1) | 7 | 8ea | 9 | 10 | 22 |
| 2 | l.d f2,v2(r1) | 7 | 9ea | 10 | 11 | 23 |
| 2 | add.d f3,f1,f2 | 8 | 12a-15a | / | 16 | 23 |
| 2 | l.d f1,v3(r1) | 8 | 10ea | 11 | 12 | 24 |
| 2 | l.d f2,v4(r1) | 9 | 11ea | 12 | 13 | 24 |
| 2 | add.d f4,f1,f2 | 9 | 14a-17a | / | 18 | 25 |
| 2 | mul.d f4,f4,f3 | 10 | 19m-24m | / | 25 | 26 |
| 2 | s.d f4,v4(r1) | 10 | 12ea | / | / | 26 |
| 2 | daddui r1,r1,8 | 11 | 12i | / | 13 | 27 |
| 2 | daddi r2,r2,-1 | 11 | 13i | / | 14 | 27 |
| 2 | bnez r2,loop | 12 | 15j | / | / | 28 |
| 3 | l.d f1,v1(r1) | 13 | 14ea | 15 | 16 | 28 |
| 3 | l.d f2,v2(r1) | 13 | 15ea | 16 | 17 | 29 |
| 3 | add.d f3,f1,f2 | 14 | 18a-21a | / | 22 | 29 |
| 3 | l.d f1,v3(r1) | 14 | 16ea | 17 | 18 | 30 |
| 3 | l.d f2,v4(r1) | 15 | 17ea | 18 | 19 | 30 |
| 3 | add.d f4,f1,f2 | 15 | 20a-23a | / | 24 | 31 |
| 3 | mul.d f4,f4,f3 | 16 | 25m-30m | / | 31 | 32 |
| 3 | s.d f4,v4(r1) | 16 | 18ea | / | / | 32 |
| 3 | daddui r1,r1,8 | 17 | 18i | / | 20 | 33 |
| 3 | daddi r2,r2,-1 | 17 | 19i | / | 21 | 33 |
| 3 | bnez r2,loop | 18 | 22j | / | / | 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.

Se ROB da 16, la 1^ che stalla dovrebbe essere la 17esima operazione, ovvero la add.d f4,f1,f2, ma visto che il suo ISSUE avviene a 9, nel contempo le 2 istruzioni iniziali del ROB (l.d f1,v1(r1) e l.d f2,v2(r1)) fanno commit (a clock 5 e 6); quindi si svuotano 2 posti nel ROB e vado avanti di 2, quindi prendo la 19esima istruzione, ovvero la s.d f4,v4(r1) e vedo che il suo ISSUE = 10 < COMMIT della prima istruzione ora nel ROB = 11 (ovvero la add.d f3,f1,f2). Perciò questa istruzione va bene ed è proprio la 19esima operazione a stallare.