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

|  |  |  |
| --- | --- | --- |
| * + Integer ALU: 1 clock cycle   + Data memory: 1 clock cycle   + FP multiplier unit: pipelined 7 stages | * + FP arithmetic unit: pipelined 2 stages   + FP divider unit: not pipelined unit that requires 6 clock cycles   + 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. |

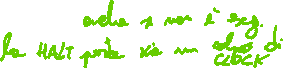
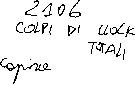
Usando il frammento di codice riportato, si calcoli il tempo di esecuzione dell’intero programma in colpi di clock e si completi la seguente tabella.

; 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 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r2,r0,100 |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| loop: l.d f1,v1(r1) |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f2,v2(r1) |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| l.d f3,v3(r1) |  |  |  |  | F | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f4,f1,f2 |  |  |  |  |  | F | D | / | / | / | / | / | / | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| div.d f4,f4,f3 |  |  |  |  |  |  | F | D | s | s | s | s | s | / | / | / | / | / | / | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| s.d f4,v4(r1) |  |  |  |  |  |  |  | F | s | s | s | s | s | D | E | s | s | s | s | S | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddui r1,r1,8 |  |  |  |  |  |  |  |  |  |  |  |  |  | F | D | s | s | s | s | s | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| daddi r2,r2,-1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | s | s | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| bnez r2,loop |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | s | D | E | M | W |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Halt |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | F | S | N | N | N | N |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Total |  |  |  |  |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |



**Domanda 2**

Considerando il programma precedente, se l’unità di divisione fosse pipelined a 6 stadi, come cambiano le prestazioni del programma? Migliorano, peggiorano? motivare la risposta.

**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



* + 1. 1 Integer ALU 1 clock cycle



* + 1. 1 Jump unit 1 clock cycle



* + 1. 1 FP multiplier unit, which is pipelined: 6 stages



* + 1. 1 FP divider unit, which is not pipelined: 6 clock cycles



* + 1. 1 FP Arithmetic unit, which is pipelined: 2 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 (effective address) | 3 | 4 | 5 |
| 1 | l.d f2,v2(r1) | 1  (no vincoli su ROB e RESERV. STAT.) | 3ea (stallo, ea fa un’istr. Per CC) | 4 | 5 | 6 |
| 1 | l.d f3,v3(r1) | 2 | 4ea | 5 | 6 | 7 |
| 1 | div.d f4,f1,f2 | 2 | 6d-11d (divisione, aspetta scritture su CDB degli operandi) | (c’è memory solo per LOAD, non per la STORE) | 12 | 13 |
| 1 | div.d f4,f4,f3 | 3 | (UNITà FP div è occupata dal CC 6 all’11 compresi, dal 12 posso partire,ma non questa, dato che gli serve f4, che avrà dopo la sua scrittura sul CDB, quindi dal CC 13)  18d-23d |  | 24 | 25 |
| 1 | s.d f4,v4(r1) | 3 | 5ea |  |  | 25 |
| 1 | daddui r1,r1,8 | 4 | 5i (integer, usa unità diversa dall’ea) |  | 6 | 26 |
| 1 | daddi r2,r2,-1 | 4 | 6i |  | 7 | 26 |
| 1 | bnez r2,loop | 5  (la branch viene eseguita sempre da sola, sia se è nel primo slot, sia se è nel secondo. Qui era nel primo, se fosse stata nel secondo, anche l’istr. Succ. Sarebbe stata sola e poi si sarebbe ricominciato con le coppie) | 8j (jump, aspetta scrittura R2 su CDB) | 8 | 9 | 27 |
| 2 | l.d f1,v1(r1) | 6 | 7ea | 8 | 9 | 27 |
| 2 | l.d f2,v2(r1) | 6 | 8ea | 9 | 10 | 28 |
| 2 | l.d f3,v3(r1) | 7 | 9ea | 10 | 11 | 28 |
| 2 | div.d f4,f1,f2 | 7 | (ho già tutti gli operandi al CC 10, quando si libera la FP DIV, ovvero dal 12esimo CC, posso già iniziare. Dato che può partire 1 CC prima della div precedente, parte prima questa, anche se è scritta dopo (Out-of-Order), così si guadagna 1 CC)  12d-17d |  | 18 | 29 |
| 2 | div.d f4,f4,f3 | 8 | 24d-29d |  | 30 | 31 |
| 2 | s.d f4,v4(r1) | 8 | 10ea |  |  | 31 |
| 2 | daddui r1,r1,8 | 9 | 10i |  | 11 | 32 |
| 2 | daddi r2,r2,-1 | 9 | 11i |  | 12 | 32 |
| 2 | bnez r2,loop | 10 | 13j |  |  | 33 |
| 3 | l.d f1,v1(r1) | 11 | 12ea | 13 | 14 | 33 |
| 3 | l.d f2,v2(r1) | 11 | 13ea | 14 | 15 | 34 |
| 3 | l.d f3,v3(r1) | 12 | 14ea | 15 | 16 | 34 |
| 3 | div.d f4,f1,f2 | 12 | 30d |  | 36 | 37 |
| 3 | div.d f4,f4,f3 | 13 | 37d |  | 43 | 44 |
| 3 | s.d f4,v4(r1) | 13 | 15ea |  |  | 44 |
| 3 | daddui r1,r1,8 | 14 | 15i |  | 16 | 45 |
| 3 | daddi r2,r2,-1 | 14 | 16i |  | 17 | 45 |
| 3 | bnez r2,loop | 15 | 18j |  |  | 46 |

**Domanda 4**

Considerando il segmento di codice presentato nella tabella precedente, se assumessimo che si volessero migliorare le prestazioni del programma duplicando una delle seguenti unità funzionali:



* FP multiplier unit



* FP divider unit
* FP Arithmetic unit
* Unità di Load/Store

Quale permetterebbe di ottenere un miglioramento maggiore nelle prestazioni del processore? motivare la risposta.



Sicuramente la divisione dato che è quell ache occupa più CC.

2) Qual’è la prima istruzione che stalla se il ROB avesse 16 SLOT?

Arriveremmo senza problem all’istr. 16, over “daddui r1, r1, 8”, intanto, essendo passati 8 CC di issue, si liberano le prime 3 istr. Che si tolgono dal ROB (commit a 5,6 e 7), quindi si va avanti di 3 istr. fino a l.d f1, v1(r1), siamo al CC 11, non ha fatto COMMIT nessun’altra istr. Fià in ROB -> STALLO al CC 11.

3) Ci potrebbero essere domande simili alla 2) relative alla disponibilità delle RESERVATION STATION che, analogamente al ROB, possono causare stalli se ancora piene.