Computer Architecture Project #2

Object： Implement a simulator for Tomasulo algorithm. In this simulator, there are three adders, two multipliers, two load buffers, and two store buffers. The clock cycles in execution steps of different instructions are according to the slides. The cycles of execution step of L.D, **S.D**., ADD.D, SUB.D, MUL.D, and DIV.D are two, **one**, two, two, 10, and 40, respectively. Your simulator must be able to execute such the six instructions: L.D、S.D、ADD.D、SUB.D、MUL.D、DIV.D. (浮點數暫存器有16個，編號為F0、F2、F4、…、F30，初始值為1；整數暫存器有32個，編號為R0、R1、…、R31，除R1的初始值為16外，其餘整數暫存器初始值為0；記憶體為8個雙精準的空間(64 Bytes)，初始值為1)

Input：read a file which contains the content such as:

test1.txt (20%) no dependence

MUL.D F0, F2, F4

SUB.D F6, F8, F10

DIV.D F12, F14, F16

ADD.D F18, F20, F22

test2.txt (20%) true dependence

L.D F6, 8(R1)

ADD.D F0, F2, F6

DIV.D F8, F0, F4

SUB.D F10, F6, F8

MUL.D F12, F10, F30

S.D F12, 40(R1)

test3.txt (20%) true dependence + anti dependence

L.D F6, 8(R2)

L.D F2, 40(R3)

ADD.D F4, F2, F6

DIV.D F8, F0, F4

SUB.D F0, F6, F2

MUL.D F12, F6, F4

S.D F12, 40(R3)

test4.txt (20%) true dependence + anti dependence + output dependence

L.D F0, 0(R1)

L.D F2, 0(R1)

ADD.D F2, F2, F2

MUL.D F12, F6, F4

DIV.D F8, F12, F4

SUB.D F8, F6, F2

S.D F8, 16(R1)

S.D F12, 8(R1)

test5.txt (20%) true dependence + anti dependence + output dependence

L.D F0, 0(R1)

L.D F2, 0(R0)

DIV.D F4, F0, F2

DIV.D F8, F6, F4

DIV.D F10, F8, F4

SUB.D F8, F0, F2

Output: Show the contents of Instruction status、load buffer、store buffer、Reservation Stations、Register result status at different clock cycles、暫存器的值

. You must write the output results into the output file “output.txt”.

Turn in： You must turn in the source code, the executable file, and a report with about 3 pages (10%).

How to test your executable file: Your executable file whose name is “project.exe” must read an input file, as shown in the following：

project.exe test1.txt

Due day：6月4日

PS :一人一組

針對Load與Store的execute說明，在投影片中，pseudocode的execute階段要能夠執行需要等待Base register的value已經得到，且要是Load-Store queue的第一個，這種作法將讓load與store指令會依據順序執行，而且要等待前面的load或是store完成後，從queueu中刪除，才能執行下一個load或是store指令的execute。如此作法，可以避免因為memory address存取發生的WAW、WAR、RAW的hazard。

但是依據例子，卻又不需要等到前面的load或是store指令完成，才能進行execute階段，所以在實作時，不考慮要是load-store queue的第一個，這樣才會與範例中的各階段執行時間一致。這樣的修改將會可能在write result階段產生hazard，所以可以利用in-order方式去完成load與store指令的write result階段。





Sample 1

L.D F6, 8(R2)

L.D F2, 40(R3)

MUL.D F0, F2, F4

SUB.D F8, F6, F2

DIV.D F10, F0, F6

ADD.D F6, F8, F2

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 15 | 16 |
| 4 | 7 | 8 |
| 5 | 56 | 57 |
| 6 | 10 | 11 |

Sample 2

L.D F6, 8(R2)

L.D F2, 40(R3)

ADD.D F0, F2, F6

SUB.D F8, F6, F2

MUL.D F10, F0, F8

S.D F10, 0(R3)

DIV.D F8, F0, F2

S.D F8, 8(R3)

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 7 | 8 |
| 4 | 7 | 8 |
| 5 | 18 | 19 |
| 6 | 7 | 20 |
| 7 | 48 | 49 |
| 8 | 9 | 50(等待DIV.D所產生出的結果) |

Sample 3

L.D F6, 8(R2)

L.D F2, 40(R3)

ADD.D F4, F2, F6

DIV.D F8, F0, F4

SUB.D F6, F6, F2

MUL.D F10, F6, F4

S.D F10, 40(R3)

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 7 | 8 |
| 4 | 48 | 49 |
| 5 | 7 | 8 |
| 6 | 18 | 19 |
| 7 | 8 | 20 |

Sample 4

L.D F6, 8(R2)

L.D F2, 40(R3)

ADD.D F4, F2, F6

DIV.D F8, F0, F4

MUL.D F6, F8, F4

SUB.D F10, F2, F4

SUB.D F14, F8, F4

SUB.D F10, F6, F4

ADD.D F12, F6, F4

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 ADD.D | 7 | 8 |
| 4 | 48 | 49 |
| 5 | 59 | 60 |
| 6 SUD.D | 10 | 11 |
| 7 SUD.D | 51 | 52 |
| 9 SUD.D | 62 | 63 |
| 12 ADD.D | 62 | 63 |

Sample 5

L.D F6, 8(R2)

L.D F2, 40(R3)

MUL.D F4, F2, F6

DIV.D F8, F0, F4

MUL.D F6, F6, F2

DIV.D F10, F6, F4

S.D F10, 40(R3)

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 15 | 16 |
| 4 | 56 | 57 |
| 17 | 27 | 28 |
| 29 | 69 | 70 |
| 30 | 31 | 71 |

Sample 6

L.D F0, 0(R1)

L.D F2, 0(R0)

DIV.D F4, F0, F2

DIV.D F8, F6, F4

SUB.D F8, F0, F2

DIV.D F10, F8, F4

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 45 | 46 |
| 4 | 86 | 87 |
| 5 | 7 | 8 |
| 47 | 87 | 88 |

Sample 7

L.D F2, 8(R1) F2=1

L.D F4, 16(R0) F4=1

ADD.D F4, F2, F4 F4=2

DIV.D F4, F4, F4 F4=1

DIV.D F4, F4, F4 F4=1

DIV.D F4, F4, F4 F4=1

DIV.D F4, F4, F4 F4=1

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 4 |
| 2 | 4 | 5 |
| 3 | 7 | 8 |
| 4 | 48 | 49 |
| 5 | 89 | 90 |
| 50 | 130 | 131 |
| 91 | 171 | 172 |

L.D F2, 8(R1)

L.D F4, 16(R0)

ADD.D F4, F2, F4

DIV.D F4, F4, F4

DIV.D F4, F4, F4

DIV.D F4, F4, F4

DIV.D F4, F4, F4

**注意：以下只是提供給各位參考，並非要完全符合，可依據個人需要做修改。**

#define LoadBuffer0 0

#define LoadBuffer1 1

#define StoreBuffer0 2

#define StoreBuffer1 3

#define Adder0 4

#define Adder1 5

#define Adder2 6

#define Multiplier0 7

#define Multiplier1 8

//由於包含各種指令所用的欄位，因此實際在執行時可能有些欄位用不到

struct Instruction

{

int opcode;

int rs;

int rt;

int rd;

int offset;

int Issue; //紀錄完成該步驟的迴圈

int Execution;

int Write;

};

//由於包含各種指令所用的欄位，因此實際在執行時可能有些欄位用不到

struct ReservationStation

{

int busy;

int opcode;

int FU; //記錄使用哪個function unit

float Vj;

float Vk;

int Qi;

int Qj;

int Qk;

int A;

};

struct ReservationStation LoadBuffer[2];

struct ReservationStation StoreBuffer[2];

struct ReservationStation Adder[3];

struct ReservationStation Multiplier[2];

struct RegisterStatus

{

int Qi;

};

struct RegisterStatus FRegister[16];

struct RegisterStatus IRegister[32];

讀入欲執行之檔案中的每行MIPS組合語言，將每個指令放入struct Instruction中

//初始化各個結構

Clock=0;

//何時離開主迴圈? WriteResult()、Execute()、Issue()皆無推進任何一個執令，或是struct Instruction中的Issue、Execution、Write皆非原初始值。

while(1)

{

WriteResult(); //執行已完成Execution步驟的指令

Execute();//執行已完成Issue步驟的指令，並在維護每個正在本步驟的指令

Issue();//issue下一個未被issue的指令

Clock++;

}