3st Homework for Computer Architecture

Submission deadline: Dec. 2 , 11：55pm

Read Chapter 3 Appendix C then do the following problems.

(Total 105 points)

In 6th Edition

3.11 3.15 3.16 3.17(不做要求，建议自己推一下) C.10

```
Loop:   lw x1,0(x2)
        addi    x1,x1, 1
        sw      x1,0(x2)
        addi    x2,x2,4
        sub     x4,x3,x2
        bnz     x4,Loop
```

Figure 3.53 Code loop for Exercise 3.11.

3.11 [10/10/10] <3.3> Assume a five-stage single-pipeline microarchitecture (fetch, decode, execute, memory, write-back) and the code in Figure 3.53. All ops are one cycle except LW and SW, which are 1+2 cycles, and branches, which are 1+1 cycles. There is no forwarding. Show the phases of each instruction per clock cycle for one iteration of the loop.

a. [10] <3.3> How many clock cycles per loop iteration are lost to branch overhead?

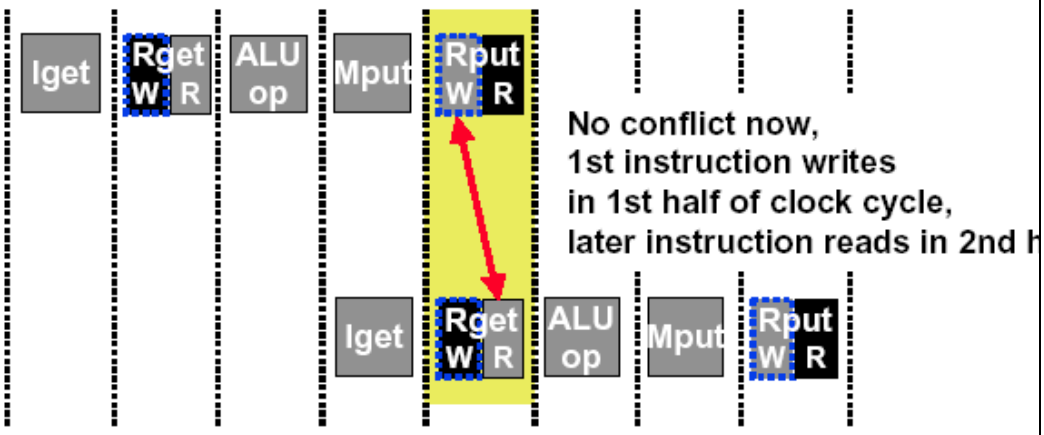
b. [10] <3.3> Assume a static branch predictor, capable of recognizing a backward branch in the Decode stage. Now how many clock cycles are wasted on branch overhead?

c. [10] <3.3> Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?

Answer

a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
lw x1, 0(x2)	F	D	E	M	-	-	W																
addi x1,x1,1		F	D	-	-	-	-	E	M	W													
sw x1, 0(x2)			F	-	-	-	-	D	-	-	E	M	-	-	W								
addi x2,x2,4								F	-	-	D	E	-	-	M	W							
sub x4,x3,x2											F	D	-	-	-	-	E	M	W				
bnz x4,Loop												F	-	-	-	-	D	-	-	E	-	M	W
Lw x1,0(x2)																						F	D

	5 clock cycles per loop iteration are lost to branch overhead. 如果没有 overhead 下一个迭代的 F 应该在 bnz 的 D 状态下面，由于 bnz 需要执行完 E 后 lw 才能够读入，那么 sub 和 bnz 的数据冲突导致 D 执行 3 个周期，之后题干说 bnz 的 E 是 2 个周期。注意：其中如果发生数据冲突，在有 double bump 的情况下同一个周期的上一条指令的 W 能够在下降沿将寄存器传到下一条指令的 D 状态。 
b	1 cycle lost with static predictor.（说法合理即可）如果偏移地址能够在 D 阶段被计算，那么 D 的下一个周期能够 fetch 下一轮的 instruction（lw）所以相比之下只需要停 1 个周期。
c	0 cycle lost. 因为动态预测器能够记录上次迭代 bnz 指令跳转的地址，直接去取，不用停顿。

```
foo: fld      F2,0(x1)   ; (F2) = X(i)
     fmul.d   F4,F2,F0   ; (F4) = a*X(i)
     fld      F6,0(x2)   ; (F6) = Y(i)
     fadd.d   F6,F4,F6   ; (F6) = a*X(i) + Y(i)
     fsd      F6,0(x2)   ; Y(i) = a*X(i) + Y(i)
     addi     x1,x1,#8   ; increment X index
     addi     x2,x2,#8   ; increment Y index
     sltu     x3,x1,x4   ; test: continue loop?
     bnez     x3,foo     ; loop if needed
```

3.15 [20/20] <3.4, 3.5, 3.7, 3.8> In this exercise, we will look at how variations on Tomasulo's algorithm perform when running the loop from Exercise 3.14. The functional units (FUs) are described in the following table.

FU type	Cycles in EX	Number of FUs	Number of reservation stations
Integer	1	1	5
FP adder	10	1	3
FP multiplier	15	1	2

Assume the following:

- Functional units are not pipelined.
- There is no forwarding between functional units; results are communicated by the common data bus (CDB).
- The execution stage (EX) does both the effective address calculation and the memory access for loads and stores. Thus, the pipeline is IF/ID/IS/EX/WB.
- Loads require one clock cycle.
- The issue (IS) and write-back (WB) result stages each require one clock cycle.
- There are five load buffer slots and five store buffer slots.
- Assume that the Branch on Not Equal to Zero (BNEZ) instruction requires one clock cycle.

a. [20] <3.4–3.5> For this problem use the single-issue Tomasulo MIPS pipeline of Figure 3.10 with the pipeline latencies from the preceding table. Show the number of stall cycles for each instruction and what clock cycle each instruction begins execution (i.e., enters its first EX cycle) for three iterations of the loop. How many cycles does each loop iteration take? Report your answer in the form of a table with the following column headers:

- Iteration (loop iteration number)
- Instruction
- Issues (cycle when instruction issues)
- Executes (cycle when instruction executes)
- Memory access (cycle when memory is accessed)
- Write CDB (cycle when result is written to the CDB)
- Comment (description of any event on which the instruction is waiting)

Show three iterations of the loop in your table. You may ignore the first instruction.

b. [20] <3.7, 3.8> Repeat part (a) but this time assume a two-issue Tomasulo algorithm and a fully pipelined floating-point unit (FPU).

(a)

No.	Iter.	Inst.	Issue	Execute /Memory	Write CDB	Comment
1	1	fld F2, 0(x1)	1	2	3	
2	1	fmul.d F4, F2, F0	2	4	19	等指令 1 的 F2
3	1	fld F6, 0(x2)	3	4	5	
4	1	fadd.d F6, F4, F6	4	20	30	等指令 2 的 F4

5	1	fsd F6, 0(x2)	5	31		等指令 4 的 F6
6	1	addi x1, x1, #8	6	7	8	
7	1	addi x2, x2, #8	7	8	9	
8	1	sltu x3, x1, x4	8	9	10	
9	1	bnez x3, foo	9	11		等指令 8 的 x3
10	2	fld F2, 0(x1)	10	12	13	等指令 bnez
11	2	fmul.d F4, F2, F0	11	14（19）	34	等指令 10 的 F2，之后等指令 2 的 Mul FU
12	2	fld F6, 0(x2)	12	13	14	
13	2	fadd.d F6, F4, F6	13	35	45	等指令 11 的 F4
14	2	fsd F6, 0(x2)	14	46		等指令 13 的 F6
15	2	addi x1, x1, #8	15	16	17	
16	2	addi x2, x2, #8	16	17	18	
17	2	sltu x3, x1, x4	17	18	20	
18	2	bnez x3, foo	18	20		等指令 17 的 x3
19	3	fld F2, 0(x1)	19	21	22	等指令 bnez
20	3	fmul.d F4, F2, F0	20	23（34）	49	等指令 19 的 F2，之后等指令 11 的 Mul FU
21	3	fld F6, 0(x2)	21	22	23	
22	3	fadd.d F6, F4, F6	22	50	60	等指令 20 的 F4
23	3	fsd F6, 0(x2)	23	61		等指令 22 的 F6
24	3	addi x1, x1, #8	24	25	26	
25	3	addi x2, x2, #8	25	26	27	
26	3	sltu x3, x1, x4	26	27	28	
27	3	bnez x3, foo	27	29		等指令 26 的 x3

(b)

No.	Iter.	Inst.	Issue	Execute/ Memory	Write CDB	Comment
1	1	fld F2, 0(x1)	1	2	3	
2	1	fmul.d F4, F2, F0	1	4	19	等指令 1 的 F2
3	1	fld F6, 0(x2)	2	3	4	
4	1	fadd.d F6, F4, F6	2	20	30	等指令 2 的 F4
5	1	fsd F6, 0(x2)	3	31		等指令 4 的 F6
6	1	addi x1, x1, #8	3	4	5	
7	1	addi x2, x2, #8	4	5	6	
8	1	sltu x3, x1, x4	4	6	7	等指令 7 的 Int FU
9	1	bnez x3, foo	5	7		等指令 8 的 x3
10	2	fld F2, 0(x1)	6	8	9	指令 9 为跳转，发射在下一周期。等指令 9 的 Int FU
11	2	fmul.d F4, F2, F0	6	10	25	等待指令 10 的 F2，全流水，Mul FU 不冲突
12	2	fld F6, 0(x2)	7	9	10	等指令 10 的 Int FU
13	2	fadd.d F6, F4, F6	7	26	36	等指令 11 的 F4

14	2	fsd F6, 0(x2)	8	37		等指令 13 的 F6
15	2	addi x1, x1, #8	8	10	11	等指令 12 的 Int FU
16	2	addi x2, x2, #8	9	11	12	等指令 15 的 Int FU
17	2	sltu x3, x1, x4	9	12	13	等指令 16 的 Int FU
18	2	bnez x3, foo	10	14		等指令 17 的 x3
19	3	fld F2, 0(x1)	11	15	16	等指令 bnez
20	3	fmul.d F4, F2, F0	20	21	36	等待指令 2 退出保留站，否则结构冲突不能发射
21	3	fld F6, 0(x2)	20	21	22	
22	3	fadd.d F6, F4, F6	21	37	47	等指令 20 的 F4
23	3	fsd F6, 0(x2)	21	48		等指令 22 的 F6
24	3	addi x1, x1, #8	22	23	24	
25	3	addi x2, x2, #8	22	24	25	等指令 24 的 Int FU
26	3	sltu x3, x1, x4	23	25	26	等指令 24 的 x1，等指令 25 的 Int FU
27	3	bnez x3, foo	23	27		等指令 26 的 x3

3.16 [10] <3.4> Tomasulo's algorithm has a disadvantage: only one result can compute per clock per CDB. Use the hardware configuration and latencies from the previous question and find a code sequence of no more than 10 instructions where Tomasulo's algorithm must stall due to CDB contention. Indicate where this occurs in your sequence.

Answer

a

Instruction	Issues at	Executes/memory	Write CDB at
fadd.d F2,F4,F6	1	2	12
add x1,x1,x2	2	3	4
add x1,x1,x2	3	5	6
add x1,x1,x2	4	7	8
add x1,x1,x2	5	9	10
add x1,x1,x2	6	11	12 (CDB conflict)

C.10 [25] <C.8> It is critical that the scoreboard be able to distinguish RAW and WAR hazards, because a WAR hazard requires stalling the instruction doing the writing until the instruction reading an operand initiates execution, but a RAW hazard requires delaying the reading instruction until the writing instruction finishes—just the opposite. For example, consider the sequence:

```
fmul.d    f0,f6,f4
fsub.d    f8,f0,f2
fadd.d    f2,f10,f2
```

The `fsub.d` depends on the `fmul.d` (a RAW hazard), thus the `fmul.d` must be allowed to complete before the `fsub.d`. If the `fmul.d` were stalled for the `fsub.d` due to the inability to distinguish between RAW and WAR hazards, the processor will deadlock. This sequence contains a WAR hazard between the `fadd.d` and the `fsub.d`, and the `fadd.d` cannot be allowed to complete until the `fsub.d` begins execution. The difficulty lies in distinguishing the RAW hazard between `fmul.d` and `fsub.d`, and the WAR hazard between the `fsub.d` and `fadd.d`. To see just why the three-instruction scenario is important, trace the handling of each instruction stage by stage through issue, read operands, execute, and write result. Assume that each scoreboard stage other than execute takes 1 clock cycle. Assume that the `fmul.d` instruction requires 3 clock cycles to execute and that the `fsub.d` and `fadd.d` instructions each take 1 cycle to execute. Finally, assume that the processor has two multiply function units and two add function units. Present the trace as follows.

1. Make a table with the column headings Instruction, Issue, Read Operands, Execute, Write Result, and Comment. In the first column, list the instructions in program order (be generous with space between instructions; larger table cells will better hold the results of your analysis). Start the table by writing a 1 in the Issue column of the `fmul.d` instruction row to show that `fmul.d` completes the issue stage in clock cycle 1. Now, fill in the stage columns of the table through the cycle at which the scoreboard first stalls an instruction.
2. For a stalled instruction write the words "waiting at clock cycle X," where X is the number of the current clock cycle, in the appropriate table column to show that the scoreboard is resolving an RAW or WAR hazard by stalling that stage. In the Comment column, state what type of hazard and what dependent instruction is causing the wait.

3. Adding the words "completes with clock cycle Y" to a "waiting" table entry, fill in the rest of the table through the time when all instructions are complete. For an instruction that stalled, add a description in the Comments column telling why the wait ended when it did and how deadlock was avoided (Hint: Think about how WAW hazards are prevented and what this implies about active instruction sequences.). Note the completion order of the three instructions as compared to their program order.

Ans:

Instruction	Issue	Read Operands	Execute	Write Result	Comment
fmul.d f0, f6, f4	1	2	3	6	

fsub.d f8, f0, f2	2	waiting at clock cycle 2, completes at clock cycle 7	8	9	Wait for f0. RAW
fadd.d f2, f10, f2	3	4	5	waiting at clock cycle 6, complete at clock cycle 8	wait for f2 to be read. WAW

Completion order: fmul.d, fadd.d, fsub.d