

HOMEWORK II
-------------

**DUE** : October 19, 2011

**READ** :

- Related portions of Chapters 2, 3 and Appendix A of the Hennessy book
- Related portions of Chapter 7 of the Jordan book

**ASSIGNMENT**: There are **three** problems.

**Solve all homework and exam problems as shown in class and past exam solutions**

1) Consider the piece of code studied in Problem 4 of Homework I. This code is for the **DAXPY** application we discussed in class :

```

loop :  L.D      F0, 0(R1)      ; load X[i]
        MUL.D   F0, F0, F2     ; multiply a * X[i]
        L.D      F4, 0(R2)     ; load Y[i]
        ADD.D   F0, F0, F4     ; add a * X[i] + Y[i]
        S.D     F0, 0(R2)     ; store Y[i]
        DADDI   R1, R1, #(-8)10 ; decrement X index
        DADDI   R2, R2, #(-8)10 ; decrement Y index
        BNEZ    R1, loop      ; loop if not done

```

Assume that this is *machine model number 3* : The MIPS uses the **Tomasulo algorithm** of the Hennessy book and as discussed in class. Additional assumptions are as follows : The functional unit timings are as listed on **A-72** of the Hennessy book : ADD.D, MUL.D and DIV.D take 3, 11 and 41 clock periods, respectively ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; Branch instructions take 2 clock periods, but there is **no** delayed branch ; there are enough functional units for integer instructions not to cause stalls ; Store instructions complete in the WR stage ; there is a **perfect** memory with no stalls.

In which clock period, will the second iteration of the loop be completed, assuming there are just two iterations ? That is, what is the last clock period in which the Write-Result stage of an instruction from the **second** iteration be done. Show the forwardings and write-in-the-first-half-read-in-the-second-half cases among the instructions. To answer the question, continue with the following table :

Instruction	IF	ID	EX	WR
L.D F0, 0(R1)	1	2	3-4	5
MUL.D F0, F0, F2	2	3	4/5 - 15	16
Continue	...	...	...	...

2) Consider the same DAXPY code given in Problem 1 above again.

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **One (1)** instruction can be committed per cycle.

a) Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there is a Branch Unit in the EX stage for calculating its effective address and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Assume that there are two iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last. **Show** which instructions are flushed from the pipeline. But, do **not** show the forwardings and write-in-the-first-half-read-in-the-second-half cases among the instructions. To answer the question, continue with the following table :

Instruction	IF	ID	EX	WR	CM
L.D F0, 0(R1)	1	2	3-4	5	6
MUL.D F0, F0, F2	2	3	4/5 - 8	9	10
Continue	...	...	...	...	...

b) Repeat part (a), by assuming that the L1 cache memories take **four** (4) clock periods each and there are **no** cache misses. Do **not** show which instructions are flushed from the pipeline **nor** the forwardings and write-in-the-first-half-read-in-the-second-half cases among the instructions.

c) Repeat part (a) by assuming that the L1 cache memories experience cache misses. Assume that the memory hierarchy is as described in the MIPS CPU PowerPoint presentation. Assume that it is a **cold** start. Clearly show in which clock period the Commit stage of an instruction from the second iteration be done last.

3) Since the reorder buffer of the scalar hardware speculative Tomasulo algorithm machine, *machine model number 4*, contains a value field, it seems it is possible that the value field of the reservation stations can be removed. Discuss how you would change the rules of committing an instruction if the value fields of reservation stations are removed. Are there negative side effects of such a change ? To answer the question focus on the process of fetching operands of an instruction. As discussed in class, there are two alternatives :

→ Fetch the operands of the instruction at issue time if they are ready and place them in an RS : **Issue-bound fetch**. Eventually, the instruction is scheduled for execution on a functional

unit. Obviously, RSs must have value fields to keep the operand values, ( $V_j/V_k$ ), until the instruction is scheduled. This is what the MIPS machine in section 2.6 does.

→ Issue the instruction without fetching the operands : **Schedule-bound fetch**. When the instruction is scheduled for execution the operands are fetched. Therefore, there is no need to have value fields in RSs to keep the operands, only  $Q_j/Q_k$  fields are needed.

The scheme explored in this problem is that if the operands are ready they are fetched during **issue** and used by the functional units immediately, not requiring the value fields in RSs. Otherwise, the operands are pointed by the  $Q_j/Q_k$  fields in RSs to the entries in the ROB.

## RELEVANT QUESTIONS AND ANSWERS

**Q1)** Consider the following **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) :

```

loop :    L.D      F0, 0(R2)      ; Load from vector B
           L.D      F1, 0(R3)      ; Load from vector C
           L.D      F2, 0(R4)      ; Load from vector D
           MUL.D    F3, F1, F2
           ADD.D    F4, F0, F2
           DIV.D    F5, F0, F2
           ADD.D    F6, F3, F4
           SUB.D    F7, F6, F5
           S.D      0(R1), F7      ; Store to vector A
           DADDI   R1, R1, #8      ; Advance the A pointer
           DADDI   R2, R2, #8      ; Advance the B pointer
           DADDI   R3, R3, #8      ; Advance the C pointer
           DADDI   R4, R4, #8      ; Advance the D pointer
           DADDI   R5, R5, #(-1)10 ; Decrement loop counter which has (64)10 initially
           BNEZ    R5, loop        ; Branch back if not the end

```

This code is without delayed branches, and without any consideration for the latencies of functional units, etc.

Assume that this is *machine model number 3* : The MIPS uses the **Tomasulo algorithm** of the Hennessy book and as discussed class. Additional assumptions are as follows : The functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; Branch instructions take 2 clock periods, but there is **no** delayed branch ; there are enough functional units for integer instructions not to cause stalls ; Store instructions complete in the WR stage ; there is a perfect memory with no stalls.

In which clock period, will the second iteration of the loop be completed, assuming there are two iterations ? That is, what is the last clock period in which the Write-Result stage of an instruction from the **second** iteration be done. **Do** show forwardings, etc. without the dependency arrows on the left. Indicate any assumptions made during the execution of the loop, if a situation not discussed in class is encountered.

**A1)** This code has been studied in the context of *machine model number 2*. That is, it is run on the statically scheduled MIPS Int+FP pipeline. The code without unrolling takes 52 clock periods to complete two iterations. If the loop is unrolled twice it takes 27 clock periods to run code to work on the same amount of data elements. However, loop unrolling means that the code size has been increased.

In the current question there is a dynamically scheduled pipeline using the Tomasulo algorithm :

	Iteration	Instruction	IF	ID	EX	WR
<b>loop :</b>	1	L.D F0, 0(R2)	1	2	3-4	5
	1	L.D F1, 0(R3)	2	3	4-5	6
	1	L.D F2, 0(R4)	3	4	5-6	7
	1	MUL.D F3, F1, F2	4	5	6/7-10	11
	1	ADD.D F4, F0, F2	5	6	7-10	11
	1	DIV.D F5, F0, F2	6	7	8-11	12
	1	ADD.D F6, F3, F4	7	8	9/11-14	15
	1	SUB.D F7, F6, F5	8	9	10/15-18	19
	1	S.D 0(R1), F7	9	10	11/18	19
	1	DADDI R1, R1, #8	10	11	12	13
	1	DADDI R2, R2, #8	11	12	13	14
	1	DADDI R3, R3, #8	12	13	14	15
	1	DADDI R4, R4, #8	13	14	15	16
	1	DADDI R5, R5, #(-1) <sub>10</sub>	14	15	16	17
	<b>loop :</b>	2	L.D F0, 0(R2)	17	18	19-20
2		L.D F1, 0(R3)	18	19	20-21	22
2		L.D F2, 0(R4)	19	20	21-22	23
2		MUL.D F3, F1, F2	20	21	22/23-26	27
2		ADD.D F4, F0, F2	21	22	23-26	27
2		DIV.D F5, F0, F2	22	23	24-27	28
2		ADD.D F6, F3, F4	23	24	25/27-30	31
2		SUB.D F7, F6, F5	24	25	26/31-34	35
2		S.D 0(R1), F7	25	26	27/34	35
2		DADDI R1, R1, #8	26	27	28	29
2		DADDI R2, R2, #8	27	28	29	30
2		DADDI R3, R3, #8	28	29	30	31

Iteration	Instruction	IF	ID	EX	WR
2	DADDI R4, R4, #8	29	30	31	32
2	DADDI R5, R5, #(-1) <sub>10</sub>	30	31	32	33
2	BNEZ R5, loop	31	32		

The second iteration of the loop ends at clock period 35. Compared with the pipelined execution of the original loop on machine model number 1, this is 17 clock periods faster. It is 8 clock periods slower than the loop unrolled twice.

**Q2)** Consider the following piece of MIPS code :

```

---
LD      R8, 0(R9)           ; R8 is loaded 2 from the memory
loop :  ADD.D   F0, F2, F4    ; F2 and F4 are already initialized
        DIV.D   F6, F8, F10  ; F8 and F10 are already initialized
        MUL.D   F2, F14, F0   ; F14 is already initialized
        DADDI   R8, R8, #(-1)10
        SUB.D   F8, F2, F6
        BNEZ   R8, loop
        S.D    F8, 0(R10)    ; R10 is already initialized

```

The code is an old code, written for the unpipelined MIPS, i.e. there are no delayed branches : *this is machine model number 0.*

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4.* **One (1)** instruction can be committed per cycle.

Assume that the functional unit timings are as listed on **A-72** of the Hennessy book : ADD.D, MUL.D and DIV.D take 3, 11 and 41 clock periods, respectively ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Show the execution of the code as we did in class. That is, what is the last clock period in which the Commit stage of the last non-speculative instruction is done. Do **not** show forwardings, etc., but do show which instructions are flushed from the pipeline.

**A2)** Due to long FP latencies and back-to-back instruction dependencies, instructions are forced to wait in Reservation Stations and the ROB. However, there are limited number of reservation stations and so instructions are stalled in the ID stage due to this structural hazard. The execution is not efficient !

There are two solutions : One is the reduction of the long FP latencies and the other one is increasing the number of reservation stations. In general, the longer the FP functional unit timings, the more reservation stations needed. Try this code for the functional unit latencies listed on page 75 of the Hennessy book.

iter #			IF	ID	EX	WR	CM
	LD	R8, 0(R9)	1	2	3-4	5	6
1	ADD.D	F0, F2, F4	2	3	4-6	7	8
1	DIV.D	F6, F8, F10	3	4	5-45	46	47
1	MUL.D	F2, F14, F0	4	5	6/7-17	18	19/48
1	DADDI	R8, R8, #(-1) <sub>10</sub>	5	6	7	8	9/49
1	SUB.D	F8, F2, F6	6	7	8/46-48	49	50
1	BNEZ	R8, loop	7	8	9	10	11/51
<hr/>							
2	ADD.D	F0, F2, F4	8	9	10/18-20	21	22/52
2	DIV.D	F6, F8, F10	9	10/17	18/49-89	90	91 ← DIV.D is stalled since the 2 RSs are occupied
2	MUL.D	F2, F14, F0	10/17	18/45	46-56	57	58/92 ← MUL.D is stalled since the 2 RSs are occupied
2	DADDI	R8, R8, #(-1) <sub>10</sub>	18/45	46	47	48	49/93
2	SUB.D	F8, F2, F6	46	47	48/90-92	93	94
2	BNEZ	R8, loop	47	48	49	50	51/95
<hr/>							
3	ADD.D	F0, F2, F4	48	49	50/57-59	60	61/95
3	DIV.D	F6, F8, F10	49	50/56	57/93-95		
3	MUL.D	F2, F14, F0	50/56	57/89	90-95		
3	DADDI	R8, R8, #(-1) <sub>10</sub>	57/89	90	91	92	93/95
3	SUB.D	F8, F2, F6	90	91	92/95		
3	BNEZ	R8, loop	91	92	93	94	95
<hr/>							
4	ADD.D	F0, F2, F4	92	93	94/95		
4	DIV.D	F6, F8, F10	93	94	95		
4	MUL.D	F2, F14, F0	94	95			
4	SUBI	R8, R8, #1	95				
	S.D	F8, 0(R10)	96	97	98	99	100 ← The execution completes in 100

These 10 instructions are flushed out in the 95<sup>th</sup> clock period

The code ends at clock period 100 !

**Q3)** Consider the following piece of MIPS code written for its unpipelined version :

```

DADDI    R1, R0, #(64)10           ; Memory accesses are commented below :
L.D      F0, 0(Rk)                   ; Rk points at constant k
loop:    LD      Ra, 0(Rindexa)        ; Rindexa points at index vector for vector A
L.D      F2, 0(Rb)                   ; Rb points at vector B
L.D      F4, 0(Rd)                   ; Rd points at vector D
DADD.D   F6, F2, F0
MUL.D    F8, F6, F4
S.D      F6, 0(Ra)                   ; Stores to vector A
S.D      F8, 0(Rc)                   ; Stores to vector C
DADDI    R1, R1, #(-1)10
DADDI    Rindexa, Rindexa, #4        ; Rindexa is advanced
DADDI    Rb, Rb, #8                  ; Rb is advanced
DADDI    Rc, Rc, #8                  ; Rc is advanced
DADDI    Rd, Rd, #8                  ; Rd is advanced
BNEZ     R1, loop

```

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **One (1)** instruction can be committed per cycle.

Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating the effective address of the branch and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Assume that there are 64 iterations. Show the execution of the loop as we did in class. That is, what is the last clock period in which the Commit stage of an instruction from the first iteration be done last. Do **not** show forwardings, etc., but do show which instructions are flushed from the pipeline.

**A3)** The execution is as follows :

Iter #	Instruction	IF	ID	EX	WR	CM
	DADDI R1, R0, #(64) <sub>10</sub>	1	2	3	4	5
	L.D F0, 0(Rk)	2	3	4-5	6	7
<b>loop :</b> 1	LD Ra, 0(R <sub>indexa</sub> )	3	4	5-6	7	8
1	L.D F2, 0(Rb)	4	5	6-7	8	9
1	L.D F4, 0(Rd)	5	6	7-8	9	10
1	DADD.D F6, F2, F0	6	7	8-11	12	13
1	MUL.D F8, F6, F4	7	8	9/12-15	16	17
1	S.D F6, 0(Ra)	8	9	10	11	12/18
1	S.D F8, 0(Rc)	9	10	11	12	13/19
1	DADDI R1, R1, #(-1) <sub>10</sub>	10	11	12	13	14/20

	1	DADDI	$R_{indexa}, R_{indexa}, \#4$	11	12	13	14	15/21
	1	DADDI	$Rb, Rb, \#8$	12	13	14	15	16/22
	1	DADDI	$Rc, Rc, \#8$	13	14	15	16	17/23
	1	DADDI	$Rd, Rd, \#8$	14	15	16	17	18/24
	1	BNEZ	$R1, loop$	15	16	17	18	19/25
<b>loop :</b>	2	LD	$Ra, 0(R_{indexa})$	16	17	18-19	20	21/26
	2	L.D	$F2, 0(Rb)$	17	18	19-20	21	22/27
	2	L.D	$F4, 0(Rd)$	18	19	20-21	22	23/28
	2	DADD.D	$F6, F2, F0$	19	20	21-24	25	26/29
	2	MUL.D	$F8, F6, F4$	20	21	22/25-28	29	30
	2	S.D	$F6, 0(Ra)$	21	22	23	24	25/31
	2	S.D	$F8, 0(Rc)$	22	23	24	25	26/32
	2	DADDI	$R1, R1, \#(-1)_{10}$	23	24	25	26	27/33
	2	DADDI	$R_{indexa}, R_{indexa}, \#4$	24	25	26	27	28/34
	2	DADDI	$Rb, Rb, \#8$	25	26	27	28	29/35
	2	DADDI	$Rc, Rc, \#8$	26	27	28	29	30/36
	2	DADDI	$Rd, Rd, \#8$	27	28	29	30	31/37
	2	BNEZ	$R1, loop$	28	29	30	31	32/38
<b>loop :</b>	3	LD	$Ra, 0(R_{indexa})$	29	30	31-32	33	34/39
	...	....	...	...	...	...	...	..
	...	....	...	...	...	...	...	..
<b>loop :</b>	64	LD	$Ra, 0(R_{indexa})$	822	823	824-825	826	827/832
	64	L.D	$F2, 0(Rb)$	823	824	825-826	827	828/833
	64	L.D	$F4, 0(Rd)$	824	825	826-827	828	829/834
	64	DADD.D	$F6, F2, F0$	825	826	827-830	831	832/835
	64	MUL.D	$F8, F6, F4$	826	827	828/831-834	835	836
	64	S.D	$F6, 0(Ra)$	827	828	829	830	831/837
	64	S.D	$F8, 0(Rc)$	828	829	830	831	832/838
	64	DADDI	$R1, R1, \#(-1)_{10}$	829	830	831	832	833/839
	64	DADDI	$R_{indexa}, R_{indexa}, \#4$	830	831	832	833	834/840
	64	DADDI	$Rb, Rb, \#8$	831	832	833	834	835/841
	64	DADDI	$Rc, Rc, \#8$	832	833	834	835	836/842
	64	DADDI	$Rd, Rd, \#8$	833	834	835	836	837/843
	64	BNEZ	$R1, loop$	834	835	836	837	838/844

The execution completes in clock period 844



loop :

65	LD	Ra, 0(R <sub>indexa</sub> )	835	836	837-838	839	840/844
65	L.D	F2, 0(Rb)	836	837	838-839	840	841/844
65	L.D	F4, 0(Rd)	837	838	839-840	841	842/844
65	DADD.D	F6, F2, F0	838	839	840-843	844	
65	MUL.D	F8, F6, F4	839	840	841/844		
65	S.D	F6, 0(Ra)	840	841	842/844		
65	S.D	F8, 0(Rc)	841	842	843/844		
65	DADDI	R1, R1, #(-1) <sub>10</sub>	842	843	844		
65	DADDI	R <sub>indexa</sub> , R <sub>indexa</sub> , #4	843	844			
65	DADDI	Rb, Rb, #8	844				

These instructions are flushed out in the 844<sup>th</sup> clock period

The loop ends at clock period 844. At that point in time, there are 10 speculatively executed instructions of iteration 65 in the pipeline. They are discarded (flushed out). Thus, only **one iteration** of instructions are flushed out.

Since FP latencies are short and successive instruction dependencies are few, not too many instructions “accumulate” in the Reservation Stations and the ROB.

**Q4)** Consider the following piece of **old** MIPS code for the unpipelined MIPS processor. That is, this is a code without delayed branches, without any consideration for the latencies of functional units, etc. :

```

loop :   L.D      F0, 0(R2)      ; Load from M
        MUL.D   F0, F0, F0    ; M[i] * M[i]
        L.D     F1, 0(R3)    ; Load from N
        L.D     F2, 0(R4)    ; Load from Q
        MUL.D   F3, F1, F2    ; N[i] * Q[i]
        ADD.D   F4, F0, F3    ; M[i] * M[i] + N[i] * Q[i]
        S.D     F4, 0(R1)    ; Store to K
        DADDI   R1, R1, #8    ; Advance the K pointer
        DADDI   R2, R2, #8    ; Advance the M pointer
        DADDI   R3, R3, #8    ; Advance the N pointer
        DADDI   R4, R4, #8    ; Advance the N pointer
        DADDI   R5, R5, #(-1)10 ; Decrement the loop counter
        BNEZ    R5, loop      ; Branch back if not the end

```

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **One (1)** instruction can be committed per cycle.

**a)** Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Assume that the loop has **two (2)** iterations. In which clock period, will the second iteration of the loop be completed in this machine model : what is the last clock period in which the Commit stage of an instruction from the second iter-

ation is done ? Do **not** show forwardings, etc., but do show which instructions are flushed from the pipeline. If a new situation is encountered, indicate the assumption made and/or how it is handled.

**b)** Repeat part (a), by assuming that the L1 cache memories take **two** (2) clock periods each and there are **no** cache misses.

**A4) a)** The execution of the loop for two iterations and the flushed out instructions are as follows :

Iteration	Instruction	IF	ID	EX	WR	CM
<b>loop :</b> 1	L.D F0, 0(R2)	1	2	3-4	5	6
1	MUL.D F0, F0, F0	2	3	4/5-8	9	10
1	L.D F1, 0(R3)	3	4	5-6	7	8/11
1	L.D F2, 0(R4)	4	5	6-7	8	9/12
1	MUL.D F3, F1, F2	5	6	7/8-11	12	13
1	ADD.D F4, F0, F3	6	7	8/12-15	16	17
1	S.D F4, 0(R1)	7	8	9	10	11/18
1	DADDI R1, R1, #8	8	9	10	11	12/19
1	DADDI R2, R2, #8	9	10	11	12	13/20
1	DADDI R3, R3, #8	10	11	12	13	14/21
1	DADDI R4, R4, #8	11	12	13	14	15/22
1	DADDI R5, R5, #(-1) <sub>10</sub>	12	13	14	15	16/23
1	BNEZ R5, loop	13	14	15	16	17/24
<b>loop :</b> 2	L.D F0, 0(R2)	14	15	16-17	18	19/25
2	MUL.D F0, F0, F0	15	16	17/18-21	22	23/26
2	L.D F1, 0(R3)	16	17	18-19	20	21/27
2	L.D F2, 0(R4)	17	18	19-20	21	22/28
2	MUL.D F3, F1, F2	18	19	20/21-24	25	26/29
2	ADD.D F4, F0, F3	19	20	21/25-28	29	30
2	S.D F4, 0(R1)	20	21	22	23	24/31
2	DADDI R1, R1, #8	21	22	23	24	25/32
2	DADDI R2, R2, #8	22	23	24	25	26/33
2	DADDI R3, R3, #8	23	24	25	26	27/34
2	DADDI R4, R4, #8	24	25	26	27	28/35

loop :

Iteration	Instruction	IF	ID	EX	WR	CM
2	DADDI R5, R5, #(-1) <sub>10</sub>	25	26	27	28	29/36
2	BNEZ R5, loop	26	27	28	29	30/37
3	L.D F0, 0(R2)	27	28	29-30	31	32/37
3	MUL.D F0, F0, F0	28	29	30/31-34	35	36/37
3	L.D F1, 0(R3)	29	30	31-32	33	34/37
3	L.D F2, 0(R4)	30	31	32-33	34	35/37
3	MUL.D F3, F1, F2	31	32	33/34-37		
3	ADD.D F4, F0, F3	32	33	34/37		
3	S.D F4, 0(R1)	33	34	35	36	37
3	DADDI R1, R1, #8	34	35	36	37	
3	DADDI R2, R2, #8	35	36	37		
3	DADDI R3, R3, #8	36	37			
3	DADDI R4, R4, #8	37				

These 11 instructions are flushed out at the end of the 37<sup>th</sup> clock period

The second iteration of the loop ends at clock period 37. 11 instructions of the third iteration are flushed out of the ROB when the loop completes.

Running the old code on the new processor shows why some old code runs slower than the new code for the same application : instructions wait for each other due to data dependencies (FP instructions above) while other instructions can be executed in the meantime (DADDI instructions above). A contemporary compiler would move the DADDI instructions up between FP instructions so that both stall cycles are reduced and useful work is done.

**b)** The execution of the loop for two iterations and a different memory speed together with the flushed out instructions are as follows :

loop :

Iteration	Instruction	IF	ID	EX	WR	CM
1	L.D F0, 0(R2)	1-2	3	4-6	7	8
1	MUL.D F0, F0, F0	3-4	5	6/7-10	11	12
1	L.D F1, 0(R3)	5-6	7	8-10	11	12/13
1	L.D F2, 0(R4)	7-8	9	10-12	13	14
1	MUL.D F3, F1, F2	9-10	11	12/13-16	17	18
1	ADD.D F4, F0, F3	11-12	13	14/17-20	21	22
1	S.D F4, 0(R1)	13-14	15	16	17	18/23
1	DADDI R1, R1, #8	15-16	17	18	19	20/24

Iteration	Instruction	IF	ID	EX	WR	CM
1	DADDI R2, R2, #8	17-18	19	20	21	22/25
1	DADDI R3, R3, #8	19-20	21	22	23	24/26
1	DADDI R4, R4, #8	21-22	23	24	25	26/27
1	DADDI R5, R5, #(-1) <sub>10</sub>	23-24	25	26	27	28
1	BNEZ R5, loop	25-26	27	28	29	30
<b>loop :</b>	L.D F0, 0(R2)	27-28	29	30-32	33	34
	MUL.D F0, F0, F0	29-30	31	32/33-36	37	38
	L.D F1, 0(R3)	31-32	33	34-36	37	38/39
	L.D F2, 0(R4)	33-34	35	36-38	39	40
	MUL.D F3, F1, F2	35-36	37	38/39-42	43	44
	ADD.D F4, F0, F3	37-38	39	40/43-46	47	48
	S.D F4, 0(R1)	39-40	41	42	43	44/49
	DADDI R1, R1, #8	41-42	43	44	45	46/50
	DADDI R2, R2, #8	43-44	45	46	47	48/51
	DADDI R3, R3, #8	45-46	47	48	49	50/52
	DADDI R4, R4, #8	47-48	49	50	51	52/53
	DADDI R5, R5, #(-1) <sub>10</sub>	49-50	51	52	53	54
	BNEZ R5, loop	51-52	53	54	55	56
<b>loop :</b>	L.D F0, 0(R2)	53-54	55	56		
	MUL.D F0, F0, F0	55-56				

These 2 instructions are flushed out at the end of the 56<sup>th</sup> clock period

The second iteration of the loop ends at clock period 56. 2 instructions of the third iteration are flushed out of the ROB when the loop completes.

The old code on the new processor takes 56 clock periods. Moving instructions around to reduce the dependencies would reduce the execution time only slightly !

**Q5)** Consider the **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) below.

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **Two (2)** instructions can be committed per cycle.

```

loop :   L.D      F0, 0(R1)      ; Load from vector A
        ADD.D   F0, F0, F1     ; F1 is already initialized with the constant value "b"
        MUL.D   F0, F0, F2     ; F2 is already initialized with the constant value "c"
        DIV.D   F0, F0, F3     ; F3 is already initialized with the constant value "d"
        S.D     F0, 0(R1)     ; Store to vector A
        DADDI   R1, R1, #8     ; Advance the vector A pointer
        DADDI   R2, R2, #(-1)10 ; Decrement the loop counter
        BNEZ    R2, loop      ; Branch back if not the end

```

Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address of the branch and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; assume that the L1 cache memories take **three** (3) clock periods each and there are **no** cache misses.

Assume that there are **two** (2) iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last ? **Do** show forwardings, etc. without the dependency arrows on the left, but **not** flushed out instructions.

**A5)** The execution of the loop with **two** iterations on the *version 4 MIPS* is as follows :

Iteration	Instruction	IF	ID	EX	WR	CM
<b>loop :</b> 1	L.D F0, 0(R1)	1-3	4	5-8	9	10
1	ADD.D F0, F0, F1	4-6	7	7/9-13	13	14
1	MUL.D F0, F0, F2	7-9	10	11/13-16	17	18
1	DIV.D F0, F0, F3	10-12	13	14/17-20	21	22
1	S.D F0, 0(R1)	13-15	16	17	18	19/22
1	DADDI R1, R1, #8	16-18	19	20	21	22/23
1	DADDI R2, R2, #(-1) <sub>10</sub>	19-21	22	23	24	25 H
1	BNEZ R2, loop	22-24	25 H	26	27	28
<b>loop :</b> 2	L.D F0, 0(R1)	25-27	28	29-32	33	34
2	ADD.D F0, F0, F1	28-30	31	32/33-36	37	38
2	MUL.D F0, F0, F2	31-33	34	35/37-40	41	42
2	DIV.D F0, F0, F3	34-36	37	38/41-44	45	46
2	S.D F0, 0(R1)	37-39	40	41	42	43/46
2	DADDI R1, R1, #8	40-42	43	44	45	46/47
2	DADDI R2, R2, #(-1) <sub>10</sub>	43-45	46	47	48	49 H
2	BNEZ R2, loop	46-48	49 H	50	51	52

The second iteration of the loop ends at clock period 52. Note that committing two instructions per clock periods does not reduce the execution time much due to slow cache memories. Only the S.D instructions makes use this feature and commits together with the previous instruction.

**Q6)** Consider the following **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) :

```

loop :  L.D      F1, 0(R1)      ; Load from vector A
         ADD.D   F3, F2, F1    ; F2 is already initialized with "k"
         S.D     F3, 0(R1)    ; Store to vector A
         L.D     F4, 0(R4)    ; Load from vector B
         SUB.D   F3, F4, F3
         MUL.D   F5, F3, F1
         S.D     F5, 0(R5)    ; Store to vector C
         DADDI   R1, R1, #8    ; Advance the vector A pointer
         DADDI   R4, R4, #8    ; Advance the vector B pointer
         DADDI   R5, R5, #8    ; Advance the vector C pointer
         DADDI   R6, R6, #(-1)10 ; R6 is the loop end counter and initialized to a value
         BNEZ    R6, loop     ; Branch back if not the end
    
```

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **Two (2)** instructions can be committed per cycle.

Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address of the branch and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Assume that there are **two** iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last ? **Do** show forwardings, etc. without the dependency arrows on the left. **But, do not** show the flushed out instructions.

**A6)** The execution of the loop with **two** iterations on the *version 4 MIPS* is as follows :

Iteration	Instruction	IF	ID	EX	WR	CM
<b>loop :</b> 1	L.D F1, 0(R1)	1	2	3-4	5	6
1	ADD.D F3, F2, F1	2	3	4/5-8	9	10
1	S.D F3, 0(R1)	3	4	5	6	7/10
1	L.D F4, 0(R4)	4	5	6-7	8	9/11
1	SUB.D F3, F4, F3	5	6	7/9-12	13	14
1	MUL.D F5, F3, F1	6	7	8/13-16	17	18
1	S.D F5, 0(R5)	7	8	9	10	11/18

loop :

Iteration	Instruction	IF	ID	EX	WR	CM
1	DADDI R1, R1, #8	8	9	10	11	12/19
1	DADDI R4, R4, #8	9	10	11	12	13/19
1	DADDI R5, R5, #8	10	11	12	13	14/20 <sup>H</sup>
1	DADDI R6, R6, #(-1) <sub>1</sub>	11	12	13	14	15/20
1	BNEZ R6, loop	12	13	14	15	16/21
2	L.D F1, 0(R1)	13	14	15-16	17	18/21
2	ADD.D F3, F2, F1	14	15	16/17-20	21	22
2	S.D F3, 0(R1)	15	16	17	18	19/22
2	L.D F4, 0(R4)	16	17	18-19	20	21/23
2	SUB.D F3, F4, F3	17	18	19/21-24	25	26
2	MUL.D F5, F3, F1	18	19	20/25-28	29	30
2	S.D F5, 0(R5)	19	<sup>H</sup> 20	21	22	23/30
2	DADDI R1, R1, #8	20	21	22	23	24/31
2	DADDI R4, R4, #8	21	22	23	24	25/31
2	DADDI R5, R5, #8	22	23	24	25	26/32
2	DADDI R6, R6, #(-1) <sub>1</sub>	23	24	25	26	27/32
2	BNEZ R6, loop	24	25	26	27	28/33

The second iteration of the loop ends at clock period 33.

**Q7)** The Section 2.4 - 2.5 Tomasulo algorithm we discussed in class (*machine model number 3*) is for a scalar processor with dynamic scheduling. It has drawbacks, two of which are that **i)** the CDB bus can carry only one value at a time, becoming a bottleneck and **ii)** while an instruction is issued to a reservation station, it is possible that the issued instruction misses to “take along” an operand with it since the operand may have just been put on the CDB by a functional unit and the issue logic is not aware of that. Thus, the instruction would wait indefinitely or would get an incorrect value. Suggest reasonable solutions for these two cases.

**A7) i)** In the first Tomasulo algorithm, seven functional units are used : 1 Load, 1 Store, 3 FP Add/Sub and 2 FP Mul/Div. We are not told about the number of other integer functional units (other than Load and Store integer functional units), so we will ignore them for this discussion.

All, except the Store unit, need the CDB. Technically, the Store unit needs the CDB since a store in transit to memory can respond to a subsequent Load from the same location. All these seven units can complete simultaneously and want to connect their results to the CDB. So, the CDB needs to have a width of 7 (seven) “words” as opposed to 1 “word.” This way, it can carry up to seven values at the same time. This, however, means that there must be **six write ports** to the FP registers, **six write ports** to each of the FP Reservation Stations and **six write ports** for the Load unit and the Store unit so that six different values can be written to FP registers, RSs, load buffer and store buffer entries at the same time. Similarly, for the integer registers (the GPRs), the number of write ports should be increased while the width of the **integer CDB** is increased.

ii) In our algorithm, the operand fetch is made during an issue, not during scheduling. So, when the ID stage decodes an instruction in a particular clock period, it checks the Register Status table to see if the  $Q_i$  field of an operand register needed by the instruction is blank. In the original implementation, if the field is not blank then the value of  $Q_i$  is  $FU_n$  (Functional Unit  $n$ ) which means the value is being computed by  $FU_n$ . Thus that the current instruction in ID waits in its RS until that value is computed by  $FU_n$ . However, in the clock period that the ID stage is working on that particular instruction, that same  $FU_n$  places its result on the CDB and is in the process of updating the Register Status table to indicate that it has finished, i.e. the  $Q_i$  entry is being written blank and the result is being written to the register. As you remember, reservation stations and the ROB are written at the end of the clock period. Thus, in that critical clock period, this  $Q_i$  is not blank.

So, the ID stage places  $FU_n$  in the  $Q_x$  field of the reservation station buffer entry when it issues the instruction, not the value of the operand in the  $V_x$  field even though the value is now on the CDB. The instruction would get a wrong value when the same  $FU_n$  computes a result for another instruction. In the worst case, if that  $FU_n$  is never needed again by the program, the instruction will wait indefinitely, that is the program will wait indefinitely. There are a number of solutions !

One solution is that each functional unit has a new output called “valid” which is 1 in the clock period the result is placed on the CDB. The issue stage would have to check the Register Status table and the special output lines. If the  $Q_i$  value of the needed register matches the functional unit number whose valid output line is 1, the ID stage instructs the RS to store the result on the CDB to the  $V_x$  field for the instruction.

Finally, it must be noted that this solution works with any number of CDBs, one or seven.

**Q8)** Consider the piece of **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) below. This code is written without delayed branches and without any consideration for the latencies of functional units, etc.

```

loop :      L.D      F0, 0(R2)      ; Load from vector B
             MUL.D   F2, F0, F1    ; F1 is already initialized with “c”
             L.D      F3, 0(R3)    ; Load from vector D
             ADD.D   F4, F2, F3
             S.D      F4, 0(R1)    ; Store to vector A
             DIV.D   F6, F4, F5    ; F5 is already initialized with “e”
             S.D      F6, 0(R3)    ; Store to vector D
             DADDI   R1, R1, #(-8)10 ; Advance the vector A pointer
             DADDI   R2, R2, #(-8)10 ; Advance the vector B pointer
             DADDI   R3, R3, #(-8)10 ; Advance the vector D pointer
             BNEZ    R1, loop      ; Branch back if not the end

```

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **Two (2)** instructions can be committed per cycle. Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses.

Assume that there are **two** iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the **second** iteration be done last ? Also, **do** show forwardings, etc. without the dependency arrows on the left **and** which instructions are flushed out of the pipeline.

**A8)** The second iteration of the loop ends at clock period 31. Only nine instructions are flushed out of the ROB since the latencies are short, two instructions are committed per clock period and the number of instructions that depend on each other's result is small. The execution of the loop with **two** iterations on the *version 4 MIPS* is as follows:

Iteration	Instruction	IF	ID	EX	WR	CM		
<b>loop :</b>	1	L.D	F0, 0(R2)	1	2	3-4	5	6
	1	MUL.D	F2, F0, F1	2	3	4/5-8	9	10
	1	L.D	F3, 0(R3)	3	4	5-6	7	8/10
	1	ADD.D	F4, F2, F3	4	5	6/9-12	13	14
	1	S.D	F4, 0(R1)	5	6	7	8	9/14
	1	DIV.D	F6, F4, F5	6	7	8/13-16	17	18
	1	S.D	F6, 0(R3)	7	8	9	10	11/18
	1	DADDI	R1, R1, #(-8) <sub>10</sub>	8	9	10	11	12/19
	1	DADDI	R2, R2, #(-8) <sub>10</sub>	9	10	11	12	13/19
	1	DADDI	R3, R3, #(-8) <sub>10</sub>	10	11	12	13	14/20
	1	BNEZ	R2, loop	11	12	13	14	15/20
<b>loop :</b>	2	L.D	F0, 0(R2)	12	13	14-15	16	17/21
	2	MUL.D	F2, F0, F1	13	14	15/16-19	20	21
	2	L.D	F3, 0(R3)	14	15	16-17	18	19/22
	2	ADD.D	F4, F2, F3	15	16	17/20-23	24	25
	2	S.D	F4, 0(R1)	16	17	18	19	20/25
	2	DIV.D	F6, F4, F5	17	18	19/24-27	28	29
	2	S.D	F6, 0(R3)	18	19	20	21	22/29
	2	DADDI	R1, R1, #(-8) <sub>10</sub>	19	20	21	22	23/30
	2	DADDI	R2, R2, #(-8) <sub>10</sub>	20	21	22	23	24/30
	2	DADDI	R3, R3, #(-8) <sub>10</sub>	21	22	23	24	25/31
	2	BNEZ	R2, loop	22	23	24	25	26/31
<b>loop :</b>	3	L.D	F0, 0(R2)	23	24	25-26	26	28/31
	3	MUL.D	F2, F0, F1	24	25	26/27-30	31	
	3	L.D	F3, 0(R3)	25	26	27-28	29	30/31
	3	ADD.D	F4, F2, F3	26	27	28/31		
	3	S.D	F4, 0(R1)	27	28	29	30	31
	3	DIV.D	F6, F4, F5	28	29	30/31		
	3	S.D	F6, 0(R3)	29	30	31		
	3	DADDI	R1, R1, #(-8) <sub>10</sub>	30	31			
	3	DADDI	R2, R2, #(-8) <sub>10</sub>	31				

These nine instructions are flushed out in the 31<sup>th</sup> clock period

**Q9)** Consider the following **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) :

**loop :** L.D F2, 0(R8)  
 ADD.D F2, F2, F0 ; F0 has a constant  
 MUL.D F2, F2, F1 ; F1 has a constant  
 S.D F2, 0(R8)  
 L.D F3, 0(R9)  
 DIV.D F4, F2, F3  
 S.D F4, 0(R9)  
 DADDI R8, R8, #(-8)<sub>10</sub>  
 DADDI R9, R9, #(-8)<sub>10</sub>  
 BNEZ R8, loop

The MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine discussed in class : *Machine model number 4*. **Two (2)** instruction can be committed per cycle. Assume that the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book ; the number of reservation station buffers for FP operations is as given in class ; there are enough number of CDB buses to eliminate bottlenecks ; there is a Branch Unit in the EX stage for calculating its effective address and determining the condition ; there is also additional branch prediction hardware in and out of the pipeline ; there are enough functional units for integer instructions not to cause

stalls ; the L1 cache memories take **one** clock period each and there are **no** cache misses. Assume that there are **two (2)** iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last ? Do **not** show forwardings, etc. **nor** the flushed out instructions.

**A9)** The execution of the loop with **two** iterations on the *version 4 MIPS* is as follows :

Iteration	Instruction	IF	ID	EX	WR	CM
<b>loop :</b> 1	L.D F2, 0(R8)	1	2	3-4	5	6
1	ADD.D F2, F2, F0	2	3	4/5-8	9	10
1	MUL.D F2, F2, F1	3	4	5/9-12	13	14
1	S.D F2, 0(R8)	4	5	6	7	8/14
1	L.D F3, 0(R9)	5	6	7-8	9	10/15
1	DIV.D F4, F2, F3	6	7	8/13-16	17	18
1	S.D F4, 0(R9)	7	8	9	10	11/18
1	DADDI R8, R8, #(-8) <sub>10</sub>	8	9	10	11	12/19
1	DADDI R9, R9, #(-8) <sub>10</sub>	9	10	11	12	13/19
1	BNEZ R8, loop	10	11	12	13	14/20
<b>loop :</b> 2	L.D F2, 0(R8)	11	12	13-14	15	16/20
2	ADD.D F2, F2, F0	12	13	14/15-18	19	20/21
2	MUL.D F2, F2, F1	13	14	15/19-22	23	24
2	S.D F2, 0(R8)	14	15	16	17	18/24
2	L.D F3, 0(R9)	15	16	17-18	19	20/25
2	DIV.D F4, F2, F3	16	17	18/23-26	27	28
2	S.D F4, 0(R9)	17	18	19	20	21/28
2	DADDI R8, R8, #(-8) <sub>10</sub>	18	19	20	21	22/29
2	DADDI R9, R9, #(-8) <sub>10</sub>	19	20	21	22	23/29
2	BNEZ R8, loop	20	21	22	23	24/30

The second iteration of the loop ends at clock period 30.

**Q10)** Consider the following piece of **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*):

```

loop :  L.D      F0, 0(R1)      ; Load from vector A
         L.D      F1, 8000(R1)  ; Load from vector B
         MUL.D    F0, F0, F2    ; F2 is already initialized with constant "k"
         DIV.D    F1, F1, F4    ; F4 is already initialized with constant "p"
         MUL.D    F5, F1, F0
         S.D      F1, 0(R1)
         S.D      F5, 8000(R1)
         DADDI   R1, R1, #8
         DADDI   R2, R2, #(-1)10
         BNEZ    R2, loop

```

Assume that the MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine as discussed in class. That is, this is *machine model number 4*. **One (1)** instruction can be committed per cycle.

**a)** Assume that the functional unit timings are as given in Question 1 above ; the number of FP reservation station buffers is as given in class. The L1 cache memories take **one** clock period each and there are no cache misses.

Assume that there are **two (2)** iterations. In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last ?

Do **not** show forwardings, etc. **nor** the flushed out instructions.

**b)** Repeat part (a) such that the L1 cache memories experience cache misses ; the memory hierarchy is as described in the MIPS CPU PowerPoint presentation ; it is a **cold** start ; the loads initially generate addresses that are divisible by the block length size.

**A10) a)** The execution of the loop with **two** iterations on the *version 4 MIPS* with ideal memory is below. The second iteration of the loop ends at clock period 32.

**b)** The execution of the loop with **two** iterations on the *version 4 MIPS* with non-ideal memory is below. The second iteration of the loop ends at clock period 44.

a)

	Instruction	IF	ID	EX	MEM	WB
loop :	L.D F0, 0(R1)	1	2	3-4	5	6
	L.D F1, 8000(R1)	2	3	4-5	6	7
	MUL.D F0, F0, F2	3	4	5-8	9	10
	DIV.D F1, F1, F4	4	5	6-9	10	11
	MUL.D F5, F1, F0	5	6/8	9/10-13	14	15
	S.D F1, 0(R1)	6/8	9	10	11	12/16
	S.D F5, 8000(R1)	9	10	11	12	13/17
	DADDI R1, R1, #8	10	11	12	13	14/18
	DADDI R2, R2, #(-1) <sub>10</sub>	11	12	13	14	15/19
	BNEZ R2, loop	12	13	14	15	16/20
loop :	L.D F0, 0(R1)	13	14	15-16	17	18/21
	L.D F1, 8000(R1)	14	15	16-17	18	19/22
	MUL.D F0, F0, F2	15	16	17-20	21	22/23
	DIV.D F1, F1, F4	16	17	18-21	22	23/24
	MUL.D F5, F1, F0	17	18/20	21/22-25	26	27
	S.D F1, 0(R1)	18/20	21	22	23	24/28
	S.D F5, 8000(R1)	21	22	23	24	25/29
	DADDI R1, R1, #8	22	23	24	25	26/30
	DADDI R2, R2, #(-1) <sub>10</sub>	23	24	25	26	27/31
	BNEZ R2, loop	24	25	26	27	28/32

MUL.D stalled due to limited RSs

MUL.D stalled due to limited RSs

b)

	Instruction	IF	ID	EX	MEM	WB
loop :	L.D F0, 0(R1)	1/5	6	7-8/12	13	14
	L.D F1, 8000(R1)	6	7	8-9/17	18	19
	MUL.D F0, F0, F2	7	8	9/13-16	17	18/20
	DIV.D F1, F1, F4	8	9	10/18-21	22	23
	MUL.D F5, F1, F0	9	10/16	17/22-25	26	27
	S.D F1, 0(R1)	10/16	17	18	19	20/28
	S.D F5, 8000(R1)	17	18	19	20	21/29
	DADDI R1, R1, #8	18	19	20	21	22/30
	DADDI R2, R2, #(-1) <sub>10</sub>	19/23	24	25	26	27/31
	BNEZ R2, loop	24	25	26	27	28/32
loop :	L.D F0, 0(R1)	25	26	27-28	29	30/33
	L.D F1, 8000(R1)	26	27	28-29	30	31/34
	MUL.D F0, F0, F2	27	28	29-32	33	34/35
	DIV.D F1, F1, F4	28	29	30-33	34	35/36
	MUL.D F5, F1, F0	29	30/32	33/34-37	38	39
	S.D F1, 0(R1)	30/32	33	34	35	36/40
	S.D F5, 8000(R1)	33	34	35	36	37/41
	DADDI R1, R1, #8	34	35	36	37	38/42
	DADDI R2, R2, #(-1) <sub>10</sub>	35	36	37	38	39/43
	BNEZ R2, loop	36	37	38	39	40/44

MUL.D stalled due to limited RSs

MUL.D stalled due to limited RSs

**Q11)** Consider the following piece of **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*):

```
loop :  L.D      F1, 0(R1)      ; Load from vector A
        L.D      F2, 0(R2)      ; Load from vector B
        ADD.D   F3, F1, F2
        MUL.D   F5, F1, F4      ; F4 is already initialized
        DIV.D   F5, F3, F5
        S.D     F5, 0(R2)
        DADDI   R1, R1, #8
        DADDI   R2, R2, #8
        DADDI   R3, R3, #(-1)10
        BNEZ   R3, loop
        S.D     F3, 0(R1)
```

Assume that the MIPS is implemented as the **scalar hardware-speculative** Tomasulo algorithm machine as discussed in class. That is, this is *machine model number 4*. **One (1)** instruction can be committed per cycle.

Assume that the functional unit timings are as shown in Figure 2.2 on page 75 of the Hennessy book ; the number of FP reservation station buffers is as given in class. Assume that there are **two (2)** iterations.

Assume also that the L1 cache memories experience **cache misses** ; the memory hierarchy is as described in the MIPS CPU PowerPoint presentation ; it is a **cold** start ; the loads initially generate addresses that are divisible by the block length size.

In which clock period, will the **second** iteration of the loop be completed ? That is, what is the last clock period in which the Commit stage of an instruction from the second iteration be done last ? Do **not** show forwardings, etc. **nor** the flushed out instructions.

**A11)** The execution of the loop with **two** iterations on the *version 4 MIPS* with non-ideal memory is below. The execution of the code ends at clock period 47.

13 instructions are flushed out and then the last instruction, the S.D instruction is executed.

Note that we cannot reorder the instructions since the pipeline is designed to handle the old code !

	Iteration	Instruction	IF	ID	EX	WR	CM
<b>loop :</b>	1	L.D F1, 0(R1)	1/5	6	7-8/12	13	14
	1	L.D F2, 0(R2)	6	7	8-9/17	18	19
	1	ADD.D F3, F1, F2	7	8	9/18-21	22	23
	1	MUL.D F5, F1, F4	8	9	10/13-16	17	18/24
	1	DIV.D F5, F3, F5	9	10	11/22-25	26	27
	1	S.D F5, 0(R2)	10	11	12	13	14/28
	1	DADDI R1, R1, #8	11	12	13	14	15/29
	1	DADDI R2, R2, #8	12	13	14	15	16/30
	1	DADDI R3, R3, #(-1) <sub>10</sub>	13/17	18	19	20	21/31
	1	BNEZ R3, loop	18	19	20	21	22/32
<b>loop :</b>	2	L.D F1, 0(R1)	19	20/16	21-22	23	24/33
	2	L.D F2, 0(R2)	20	21	22-23	24	25/34
	2	ADD.D F3, F1, F2	21	22	23/24-27	28	29/35
	2	MUL.D F5, F1, F4	22	23	24-27	28	29/36
	2	DIV.D F5, F3, F5	23	24/25	26/28-31	32	33/37
	2	S.D F5, 0(R2)	24/25	26	27	28	29/38
	2	DADDI R1, R1, #8	26	27	28	29	30/39
	2	DADDI R2, R2, #8	27	28	29	30	31/40
	2	DADDI R3, R3, #(-1) <sub>10</sub>	28	29	30	31	32/41
	2	BNEZ R3, loop	29	30	31	32	33/42
<b>loop :</b>	3	L.D F1, 0(R1)	30	31/29	30-31	32	33/42
	3	L.D F2, 0(R2)	31	32	33-34	35	36/42
	3	ADD.D F3, F1, F2	32	33	34/35-38	39	40/42
	3	MUL.D F5, F1, F4	33	34	35-38	39	40/42
	3	DIV.D F5, F3, F5	34	35	36/39-42		
	3	S.D F5, 0(R2)	35	36	37	38	39/42
	3	DADDI R1, R1, #8	36	37	38	39	40/42
	3	DADDI R2, R2, #8	37	38	39	40	41/42
	3	DADDI R3, R3, #(-1) <sub>10</sub>	38	39	40	41	42
	3	BNEZ R3, loop	39	40	41	42	
<b>loop :</b>	4	L.D F1, 0(R1)	40	41	42		
	4	L.D F2, 0(R2)	41	42			
	4	ADD.D F3, F1, F2	42				
		S.D F3, 0(R1)	43	44	45	46	47

DIV.D stalled due to limited RSs

These 13 instructions are flushed out