

HOMEWORK I
------------

**DUE :** February 9, 2012

**READ :**

- Related portions of Chapters 1, 2, 3 and Appendix A and Appendix B of the Hennessy book
- Related portions of Chapters 1 and 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 following piece of MIPS code :

200	LD	R8, 230(R0)	
204	LD	R9, 238(R0)	
208	L.D	F1, 0(R8)	; Load "k"
20C	L.D	F2, 8(R8)	; Load an element of A
210	DADDI	R9, R9, #(-1) <sub>10</sub>	; Decrement the loop end counter
214	MUL.D	F1, F1, F2	; Multiply "k" and A[i]
218	BNEZ	R9, (-4) <sub>10</sub>	; The end of the loop ?
→ 21C	DADDI	R8, R8, #8	; Branch delay slot instruction
220	JR	R31	; Return from the function
224	S.D	F1, 240(R0)	; Store the result in p
228	---		; This location is not used
22C	---		; This location is not used
230	400		
238	2		
240	?		

Note that this code is a function and the JR instruction is used to return from the function. The JR takes two clock periods as branch instructions do. The delayed branch concept is also applied to the JR. Thus, the instruction that follows the JR (the S.D. in location 228 above) is executed after JR. Then, the instruction pointed by the JR effective address is executed (the instruction pointed by R31).

Note that the code is written for *machine model number 2* (the MIPS Int+FP pipeline) and the compiler determined that "DADDI R8, R8, #8" is an independent instruction for the BNEZ delay slot and so it placed it there. This instruction is pointed by an arrow above. The instruction updates the "A" pointer which is R8. Finally, locations 228 and 22C do not have an instruction. That is, they are not used.

**a)** Write the corresponding piece of high-level code in the style of class discussions and also by using the names of variables mentioned in the comment section above.

**b)** Assume that this is the *machine model number 2* (the MIPS Int+FP pipeline) and the Figure 2.2 latencies are applicable. Clearly show in which clock period the last cycle of the S.D after the JR instruction is performed as done in lecture presentations, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases.

Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**c)** Repeat part (b), by assuming that the L1 cache memories take three (3) clock periods each and there are **no** cache misses.

**d)** Assume that all the addresses in the code above are **physical** addresses. Assume also that the physical Level 1 instruction and data caches are as described in the MIPS CPU PowerPoint presentation.

Clearly **show** which instruction/data is in which cache, which cache block and which physical memory block for **all** instructions and data.

**e)** Repeat part (b), by assuming that the L1 cache memories experience cache misses and using the solution of part (d).

Assume that it is a **cold** start. Clearly show in which clock period the last cycle of the S.D after the JR instruction is performed as done in lecture presentations, i.e. together with all the necessary forwarding and write-in-the-first-half-read-in-the-second-half cases.

**2)** Consider the program in Problem 1 above. Assume that this is the *machine model number 2* (the MIPS Int+FP pipeline) and the ADD.D, MUL.D and DIV.D latencies are 3, 6 and 24, respectively. Clearly show in which clock period the last cycle of the S.D after the JR instruction is performed as done in lecture presentations, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**3)** The following language code computes, DAXPY which is discussed in class :  $Y[i] = a * X[i] + Y[i]$ , the key step in a Gaussian elimination.

```

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

```

The DAXPY code has loop-level parallelism. Thus, the iterations can be performed in parallel. Assume that the loop is executed **even, non-zero number** of times to simplify the solution. The code is written for the unpipelined MIPS processor as there is no instruction after the BNEZ to fill in the branch delay slot. That is, this code is written for *machine model number 0*.

**a)** In order to solve the problem, **first** show the execution of the loop for two iterations on the statically scheduled MIPS Int+FP pipeline (*machine model number 2*), by using Figure 2.2 latencies, as done in lecture presentations, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. Just place a NOP after the BNEZ instruction so that the delayed branch requirement is satisfied. There is **no** other NOP needed for the code. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**b) Then**, modify the loop, by using unrolling and show the timing for **one** iteration, together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. Use Figure 2.2 latencies, when unrolling. Make sure the code is not too long, i.e. the amount of unrolling is not too large. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

## RELEVANT QUESTIONS AND ANSWERS

**Q1)** Consider the following piece of old code :

```

loop :  L.D      F2, 0(R1)      ; Statement S1. R1 points at vector X & is initially (16)10
         ADD.D   F0, F0, F2     ; S2. F0 is variable k & is initially 0
         DADDI  R1, R1, #(-8)10 ; S3
         BNEZ   R1, loop        ; S4
         S.D    F0, 0(R2)      ; S4
    
```

R1 and F0 are already initialized. Assume that R1 is initially (16)<sub>10</sub>.

This code is written for the unpipelined processor, since, for example, the ADD.D is right after the L.D., even though the ADD.D depends on the L.D. Therefore, this code is written for *machine model number 0*.

Write down the corresponding algorithm.

**A1)** The algorithm is as follows :

```

for (i = 1 ; i <= 2 ; i++)
    k = k + X[i] ;
    
```

We are given that the first element of “X” is pointed by R1 initially. Variable “k” is pointed by R2 when the loop is over. There are only two iterations of the loop, since initially R1 is (16)<sub>10</sub>. R1 is decremented by 8 and compared with 0 at end the loop.

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

```

k = 0.0 ;
for (i = 1 ; i<= 100 ; i++)
    k = k + A[i]

```

Assume all numbers except the loop index are double-precision FP numbers and the latencies are as given in Figure 2.2 of the Hennessy book. Write down the code needed so that the **loop instructions** are scheduled without any delay on the statically scheduled MIP Int+FP pipeline. That is, this is *machine model number 2*. Make observations if necessary.

**A2)** This code adds the elements of a vector. There is a loop-carried dependence which is circular. The loop cannot be scheduled without stalls, given the latencies and the short loop length.

However, the stalls can be avoided if the loop is partitioned to even and odd sections. We add even and odd numbered vector elements separately, by using two instructions : S1 and S2 below. The even and odd parts are independent and can progress in parallel. Finally, we sum the results of the even and odd parts after the loop is over (after we exit the loop) :

	LW	Rc, #(i) <sub>10</sub>	; Rc is the loop counter register : “i”	
	L.D	F0, #0	; F0 accumulates additions on even elements	
	L.D	F2, #0	; F2 accumulates additions on odd elements	
<div style="display: inline-block; vertical-align: middle;">             the loop              {              }              out of the loop              {              }           </div>	<b>loop:</b>	L.D	F4, 0(Ra)	; Register Ra, already initialized, points at vector A
		L.D	F6, 8(Ra)	
		ADD.D	F0, F0, F4;	; Statement S1
		ADD.D	F2, F2, F6	; Statement S2
		DADDI	Rc, Rc, #(-2) <sub>10</sub>	
		BNEZ	Rc, loop	
		DADDI	Ra, Ra, #16	; Branch delay slot
		ADD.D	F0, F0, F2	; Statement S3
		<i>insert two independent instructions here from out of the loop for the S3 latency</i>		
		S.D	F0, 0(Rk)	; Statement S4. Save result in “k”

Two independent instructions have to be inserted between S3 and S4 to prevent data hazards. Note that we assume that there are **even** number of elements in the vector.

**Q3)** Consider the piece of **old** code written for the unpipelined CPU (*machine model number 0*) below.

Write the corresponding piece of high-level code in the style of class discussions and homework assignments and also by using the names of variables mentioned in the comment section above. Mention if this is implementing an application already studied in class.

```

L.D      F8, #0                ; Load value zero (0) to F8
DADDI    R1, R0, #(64)10      ; Memory accesses are commented below :
loop:    L.D      F0, 0(RA)      ; RA points at vector A
        L.D      F2, 0(RB)      ; RB points at vector B
        L.D      F4, 0(RC)      ; RC points at vector C
        MUL.D    F6, F0, F2
        ADD.D    F8, F8, F6
        MUL.D    F10, F4, F8
        S.D      F10, 0(RC)      ; Stores to vector C
        DADDI    R1, R1, #(-1)10
        DADDI    RA, RA, #8      ; RA is advanced
        DADDI    RB, RB, #8      ; RB is advanced
        DADDI    RC, RC, #8      ; RC is advanced
        BNEZ    R1, loop
        S.D      F8, 0(Rk)      ; Stores to scalar k out of the loop.

```

**A3)** The high-level code implements the dot product and another operation :

```

k = 0 ;
for (i = 0 ; i < 64 ; i = i + 1)
{
    k = k + (A[i] * B[i])      /* S1 : Implements the dot product */
    C[i] = k * C[i] ;        /* S2 : Implements the other operation */
}

```

Statement S1 in the loop implements the dot-product operation. The intermediate result of the dot product operation is used for another operation (S2, the second statement in the loop). The algorithm is a **sequential** algorithm.

**Q4)** Consider the following **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) :

```

500    L.D      F1, 0(R4)        ; R4 is passed a value of 2004
504    MUL.D    F3, F2, F1      ; F2 is passed a value
508    DIV.D    F2, F3, F4      ; F4 is passed a value
50C    DADDI    R4, R4, #8
510    DADDI    R5, R5, #(-1)10 ; R5 is passed a value of 2
514    BNEZ    R5, (-6)10
518    S.D      F2, 0(R6)      ; R6 is passed a value of 2000
51C    JR      R31

```

**a)** Rewrite the code to prevent load and branch stalls, by using as few NOPs as possible.

**b)** After rewriting the subroutine, show the execution of the loop, including forwardings, until the code returns on the statically scheduled MIPS Int+FP pipeline (*machine model number 2*) as done in class. Assume that the ADD.D,

MUL.D and DIV.D latencies are 3, 6 and 24, respectively. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

c) Repeat part (b), by assuming that the L1 cache memories take **four** (4) clock period each and there are **no** cache misses.

**A4) a) and b)** The reordered program and its execution timing are below.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
500 L.D F1, 0(R4)	1	2	3	4	5					
504 DADDI R4, R4, #8	2	3	4	5	6	13	14	15	16	17
508 MUL.D F3, F2, F1	3	4	5-11	12	13	14	15/36 H	37-43	44	45
50C DADDI R5, R5, #(-1) <sub>10</sub>	4	5	6	7	8	15/36	37	38	39	40
50C DIV.D F2, F3, F4	5	6/11	12-36	37	38	37	38/43	44-68	69	70
514 BNEZ R5, (-5) <sub>10</sub>	6/11	12				38/43	44			
518 L.D F1, 0(R4)	12	13	14	15	16 H	44	45	46	47	48
51C JR R31						45	46			
520 S.D F2, 0(R6)						46	47/67	68	69	

Note that the branch delay slot contains a L.D instruction which may be problematic when the memory hierarchy is added to the picture : Just when the loop is exited, a cache miss and/or a page fault can be generated by this L.D. In the unpipelined MIPS case this would not happen. What would your action be ?

c) The execution timing with slower cache misses is below.

Instruction	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
500 L.D F1, 0(R4)	1-4	5	6	7-10	11					
504 DADDI R4, R4, #8	5-8	9	10	11	12	29-32	33	34	35	36
508 MUL.D F3, F2, F1	9-12	13	14-20	21	22	33-36	37/46	47-53	54	55
50C DADDI R5, R5, #(-1) <sub>10</sub>	13-16	17	18	19	20	37-40/46	47	48	49	50
50C DIV.D F2, F3, F4	17-20	21	22-46	47	48	46-49	50/53	54-78	79	80
514 BNEZ R5, (-5) <sub>10</sub>	21-24	25				50-53	54			
518 L.D F1, 0(R4)	25-28	29	30	31-34	35	54-57	58	59	60-63	64
51C JR R31						58-61	62			
520 S.D F2, 0(R6)						62-65	66/77	78	79	

**Q5)** 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      F7, 0(R1)      ; 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
    
```

a) Write the corresponding piece of **high-level code** in the style of class discussions and homework assignments. Add comments to your code. Is there loop-level parallelism in the high-level code ? Explain.

Note that if your high-level code has more than **six** lines, it may **not** be correct or **not** in the style of class discussions.

b) Show the execution of the loop, including forwardings, for **two** iterations on the statically scheduled MIPS Int+FP pipeline (*machine model number 2*), by using Figure 2.2 latencies, as done in class. When does the second iteration end ? Assume that there is a NOP after the BNEZ instruction to satisfy the delayed branch requirement. **No** other NOP can be added. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

c) Modify the loop, by unrolling it and then show the timing for **one** iteration of the **new** loop on the statically scheduled MIPS Int+FP pipeline (*machine model number 2*), by using Figure 2.2 latencies, as done in class. When does the first iteration end ? Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**A5)**

a) The piece of high-level code is as follows :

```

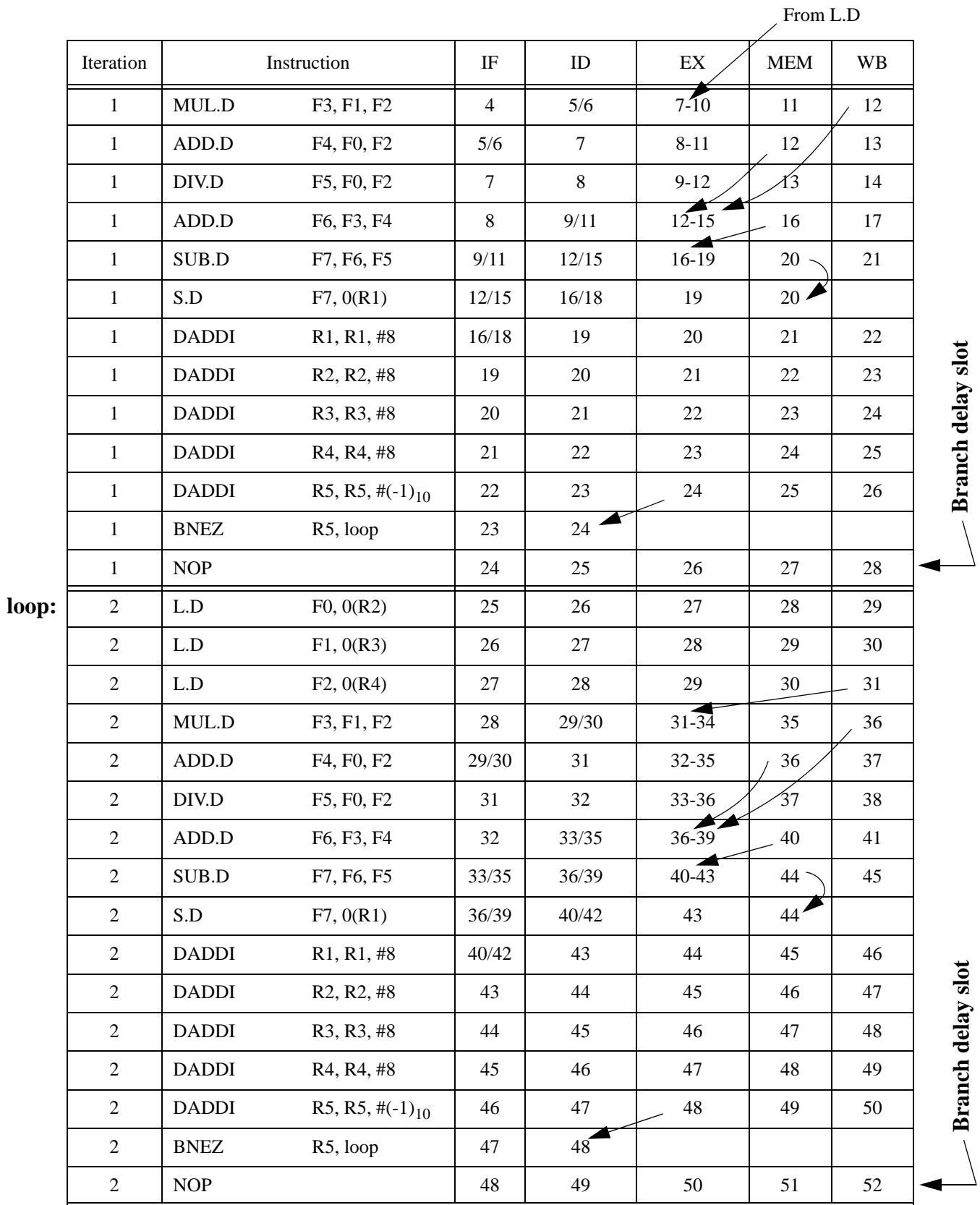
For (i = 0 ; i < 64 ; i++)
  A[i] = (C[i] * D[i]) + (B[i] + D[i]) - (B[i] / D[i]) ;
    
```

The code has loop-level parallelism that **all** 64 loop iterations can be run in parallel.

b) The execution of the loop for the first two iterations is as follows :

Iteration	Instruction	IF	ID	EX	MEM	WB
<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

To MUL.D ←



The second iteration of the loop ends at clock period 52.

c) The execution of the new loop with loop unrolling (**twice**) and instruction movements for the first iteration is shown below :

**loop:**

Iteration	Instruction		IF	ID	EX	MEM	WB
1	L.D	F1, 0(R3)	1	2	3	4	5 H
1	L.D	F2, 0(R4)	2	3	4	5	6 H
1	L.D	F0, 0(R2)	3	4	5	6	7 H
1	MUL.D	F3, F1, F2	4	5 H	6-9	10	11
1	ADD.D	F4, F0, F2	5	6 H	7-10	11	12
1	DIV.D	F5, F0, F2	6	7 H	8-11	12	13
1	L.D	F9, 8(R3)	7	8	9	10	11
1	L.D	F10, 8(R4)	8	9	10	11	12
1	ADD.D	F6, F3, F4	9	10	11-14	15	16
1	L.D	F8, 8(R2)	10	11	12	13	14
1	MUL.D	F11, F9, F10	11	12	13-16	17	18
1	ADD.D	F12, F8, F10	12	13	14-17	18	19
1	DIV.D	F13, F8, F10	13	14	15-18	19	20
1	SUB.D	F7, F6, F5	14	15	16-19	20	21
1	DADDI	R2, R2, #(16) <sub>10</sub>	15	16	17	18	19
1	ADD.D	F14, F11, F12	16	17	18-21	22	23
1	DADDI	R3, R3, #(16) <sub>10</sub>	17	18	19	20	21
1	DADDI	R4, R4, #(16) <sub>10</sub>	18	19	20	21	22
1	DADDI	R5, R5, #(-2) <sub>10</sub>	19	20	21	22	23
1	SUB.D	F15, F14, F13	20	21	22-25	26	27
1	S.D	F7, 0(R1)	21	22	23	24	
1	DADDI	R1, R1, #(16) <sub>10</sub>	22	23	24	25	26
1	BNEZ	R5, loop	23	24			
1	S.D	(F15, -8) <sub>10</sub> (R1)	24	25	26	27	

Branch delay slot

The first iteration of the modified loop ends at clock period 27. One iteration of the new loop corresponds to two iterations of the original loop, which takes 52 clock periods. The changes on the loop saved 25 clock periods !

**Q6)** 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
          MUL.D   F0, F0, F1    ; F1 is already initialized with constant "k"
          S.D     F0, 0(R1)
          L.D     F2, 0(R2)    ; Load from vector B
          DIV.D   F2, F2, F3    ; F3 is already initialized with constant "p"
          L.D     F4, 0(R3)    ; Load from vector C
          ADD.D   F2, F2, F4
          ADD.D   F2, F2, F0
          S.D     F2, 0(R2)
          DADDI  R1, R1, #(-8)10
          DADDI  R2, R2, #(-8)10
          DADDI  R3, R3, #(-8)10
          DADDI  R4, R4, #(-1)10
          BNEZ   R4, loop

```

Assume that this is *machine model number 2* (the MIPS Int+FP pipeline). Assume that the functional unit timings are as follows :

- ADD.D takes 3 clock periods in the EX stage
- MUL.D takes 5 clock periods in the EX stage
- DIV.D takes 7 clock periods in the EX stage

**a)** Write the corresponding piece of high-level code as shown in class. Use the names of variables given in the comment section above. Is there loop-level parallelism ? Why ?

**b)** Assume that there are **two (2)** iterations. In which clock period, will the **second** iteration of the loop be completed? Clearly show in which clock period the last iteration ends as done in class, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. To solve the problem, move an instruction down to the branch delay slot. Do **not** do any other reordering!

Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**A6) a)** The corresponding high-level code is as follows :

```

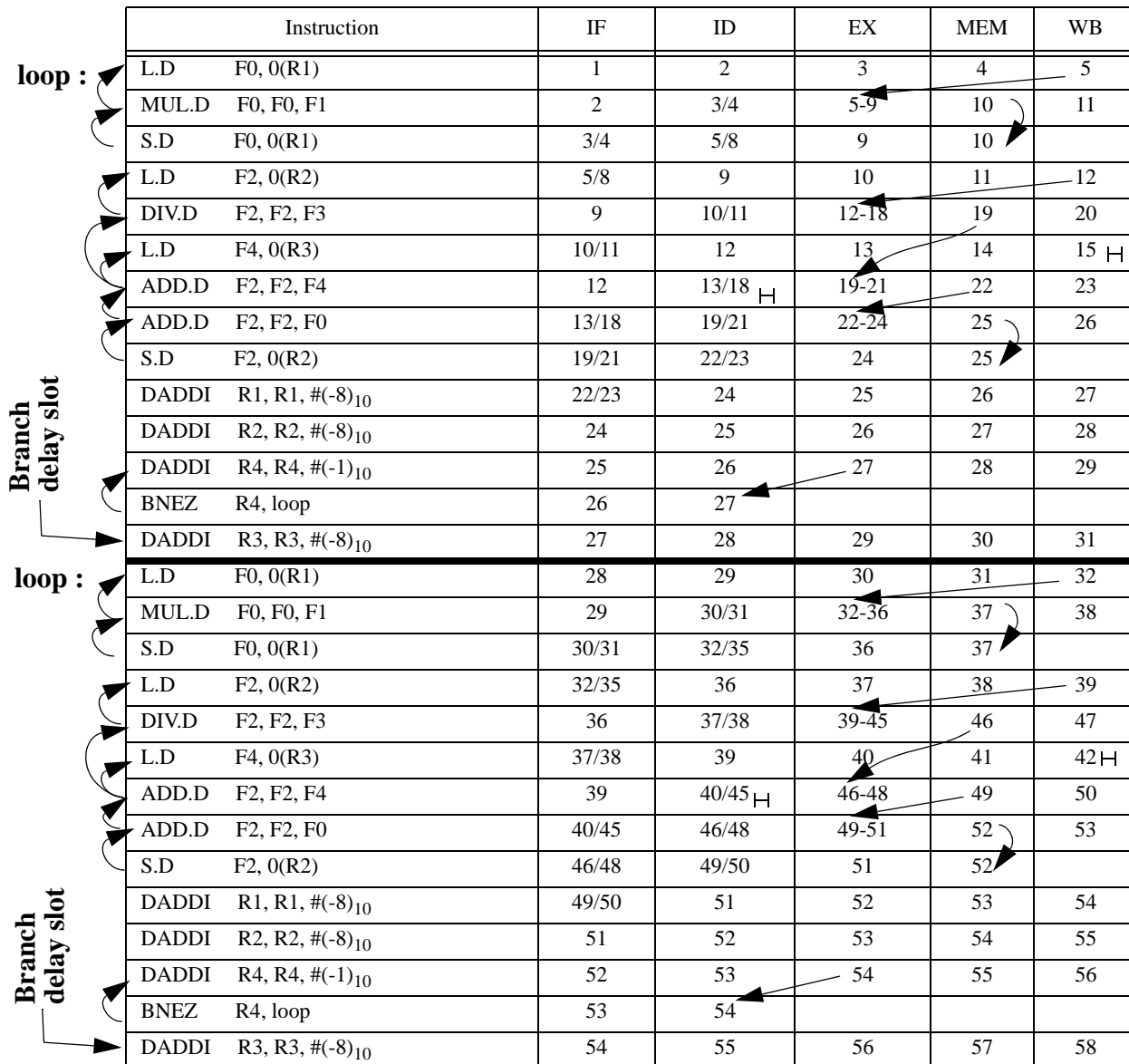
for (i = 0 ; i < n ; i = i + 1)
{ A[i] = A[i] * k ;
  B[i] = (B[i] / p) + C[i] + A[i] ; }

```

There is loop-level parallelism since no value computed in one iteration is used by any other iteration. All iterations are independent of each other. Therefore, there is loop-level parallelism.

**b)** The execution of the loop with a moved instruction and **two** iterations on the *version 2 MIPS* is below.

The second iteration ends in clock period 58 ! All hazards are RAW. Note that there are other instructions that can be moved down besides that specific ADDI instruction.



**Q7)** Consider the piece of **old** MIPS code below for the **unpipelined** MIPS processor (*machine model number 0*) discussed in class. Assume that this is *machine model number 2* (the MIPS Int+FP pipeline) and the functional unit timings are as listed in **Figure 2.2** on page 75 of the Hennessy book.

a) Move an instruction down to deal with the branch delay slot. Do **not** do any other reordering!

b) Assume that there are only **two (2)** iterations. In which clock period, will the **second** iteration of the loop be completed? Clearly show in which clock period the last iteration ends as done in class, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

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

```

loop :   L.D      F2, 0(R8)
        ADD.D   F2, F2, F0      ; F0 is already initialized
        MUL.D  F2, F2, F1      ; F1 is already initialized
        S.D    F2, 0(R8)
        L.D    F3, 0(R9)
        DIV.D  F4, F2, F3
        S.D    F4, 0(R9)
        DADDI  R8, R8, #(-8)10
        DADDI  R9, R9, #(-8)10
        BNEZ   R8, loop

```

**A7) a) & b)** The execution of the loop with a moved instruction and **two** iterations on the *version 2 MIPS* is below.

	Instruction	IF	ID	EX	MEM	WB
	L.D F2, 0(R8)	1	2	3	4	5
	ADD.D F2, F2, F0	2	3/4	5-8	9	10
	MUL.D F2, F2, F1	3/4	5/8	9-12	13	14 H
	S.D F2, 0(R8)	5/8	9/11	12	13	
	L.D F3, 0(R9)	9/11	12	13	14	15
	DIV.D F4, F2, F3	12	13/14 H	15-18	19	20
	S.D F4, 0(R9)	13/14	15/17	18	19	
	DADDI R8, R8, #(-8) <sub>10</sub>	15/17	18	19	20	21 H
	BNEZ R8, loop	18	19			
	DADDI R9, R9, #(-8) <sub>10</sub>	19	20	21	22	23
	L.D F2, 0(R8)	20	21 H	22	23	24
	ADD.D F2, F2, F0	21	22/23	24-27	28	29
	MUL.D F2, F2, F1	22/23	24/27	28-31	32	33 H
	S.D F2, 0(R8)	24/27	28/30	31	32	
	L.D F3, 0(R9)	28/30	31	32	33	34
	DIV.D F4, F2, F3	31	32/33 H	34-37	38	39
	S.D F4, 0(R9)	32/33	34/36	37	38	
	DADDI R8, R8, #(-8) <sub>10</sub>	34/36	37	38	39	40
	BNEZ R8, loop	37	38			
	DADDI R9, R9, #(-8) <sub>10</sub>	38	39	40	41	42

The second iteration ends in clock period 42 ! All hazards are RAW. Note that there are other instructions that can be moved down besides the ADDI instruction.

c) The execution timing with slower cache misses is below.

	Instruction	IF	ID	EX	MEM	WB
	L.D F2, 0(R8)	1-2	23	4	5-6	7
	ADD.D F2, F2, F0	3-4	5/6	7-10	11	12
	MUL.D F2, F2, F1	5-6	7/10	11-14	15	16 H
	S.D F2, 0(R8)	7-8/10	11/13	14	15	
	L.D F3, 0(R9)	11-12/13	14	15	16-17	18
	DIV.D F4, F2, F3	14-15	16/17 H	18-21	22	23
	S.D F4, 0(R9)	16-17	18/20	21	22	
	DADDI R8, R8, #(-8) <sub>10</sub>	18-19/20	21	22	23	24
	BNEZ R8, loop	21-22	23			
	DADDI R9, R9, #(-8) <sub>10</sub>	23-24	25	26	27	28
	L.D F2, 0(R8)	25-26	27	28	29-30	31
	ADD.D F2, F2, F0	27-28	29/30	31-34	35	36
	MUL.D F2, F2, F1	29-30	31/34	35-38	39	40 H
	S.D F2, 0(R8)	31-32/34	35/37	38	39	
	L.D F3, 0(R9)	35-36/37	38	39	40-41	42
	DIV.D F4, F2, F3	38-39	40/41 H	42-45	46	47
	S.D F4, 0(R9)	40-41	42/44	45	46	
	DADDI R8, R8, #(-8) <sub>10</sub>	42-43/44	45	46	47	48
	BNEZ R8, loop	45-46	47			
	DADDI R9, R9, #(-8) <sub>10</sub>	47-48	49	50	51	52

The second iteration ends in clock period 52 ! All hazards are RAW. Note that slower L1 cache memories resulted in a 10 clock period delay in completion.

**Q8)** Consider the **old** MIPS code below written for the unpipelined MIPS processor (*machine model number 0*). Assume that this is *machine model number 2* (the MIPS Int+FP pipeline). Assume that the functional unit timings are as shown in Figure 2.2 on page 75 of the Hennessy book. Assume that there are **two (2)** iterations. In which clock period, will the **second** iteration of the loop be completed ? The L1 cache memories take **one** clock period each and there are no cache misses.

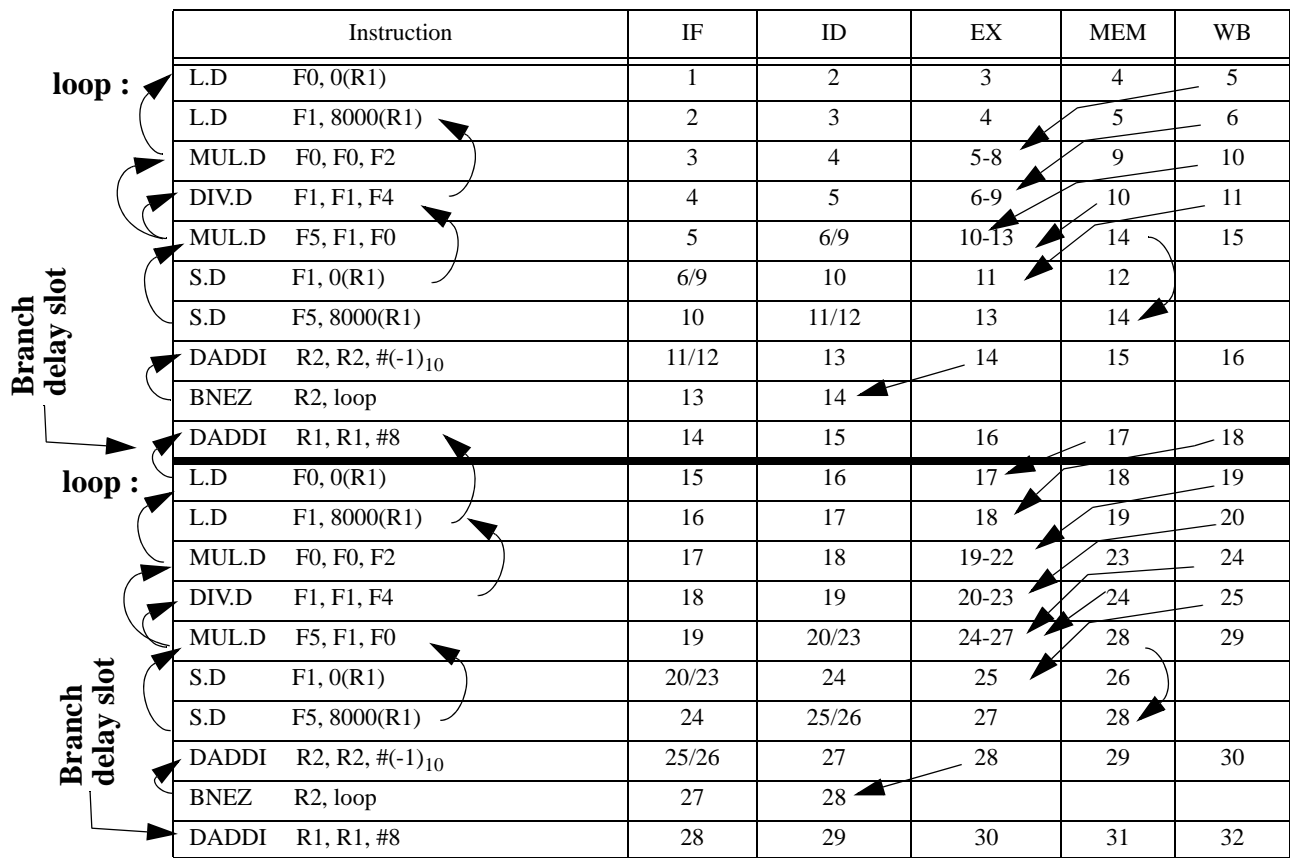
Clearly show in which clock period the last iteration ends as done in class, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. To solve the problem, move an instruction down to the branch delay slot. Do **not** do any other reordering!

```

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

```

A8) The execution of the loop with a moved instruction and **two** iterations on the *version 2 MIPS* is as follows :



The second iteration ends in clock period 32. All data hazards are RAW. Note that there are other instructions that can be moved down besides that specific DADDI instruction.

Q9) Consider the **old** MIPS code for the unpipelined MIPS processor (*machine model number 0*) below. Assume that this is *machine model number 2* (the MIPS Int+FP pipeline). Assume that the functional unit timings are as shown in Figure 2.2 on page 75 of the Hennessy book. Assume that there are **two (2)** iterations.

a) The L1 cache memories take **one** clock period each and there are **no** cache misses. Clearly show in which clock

```

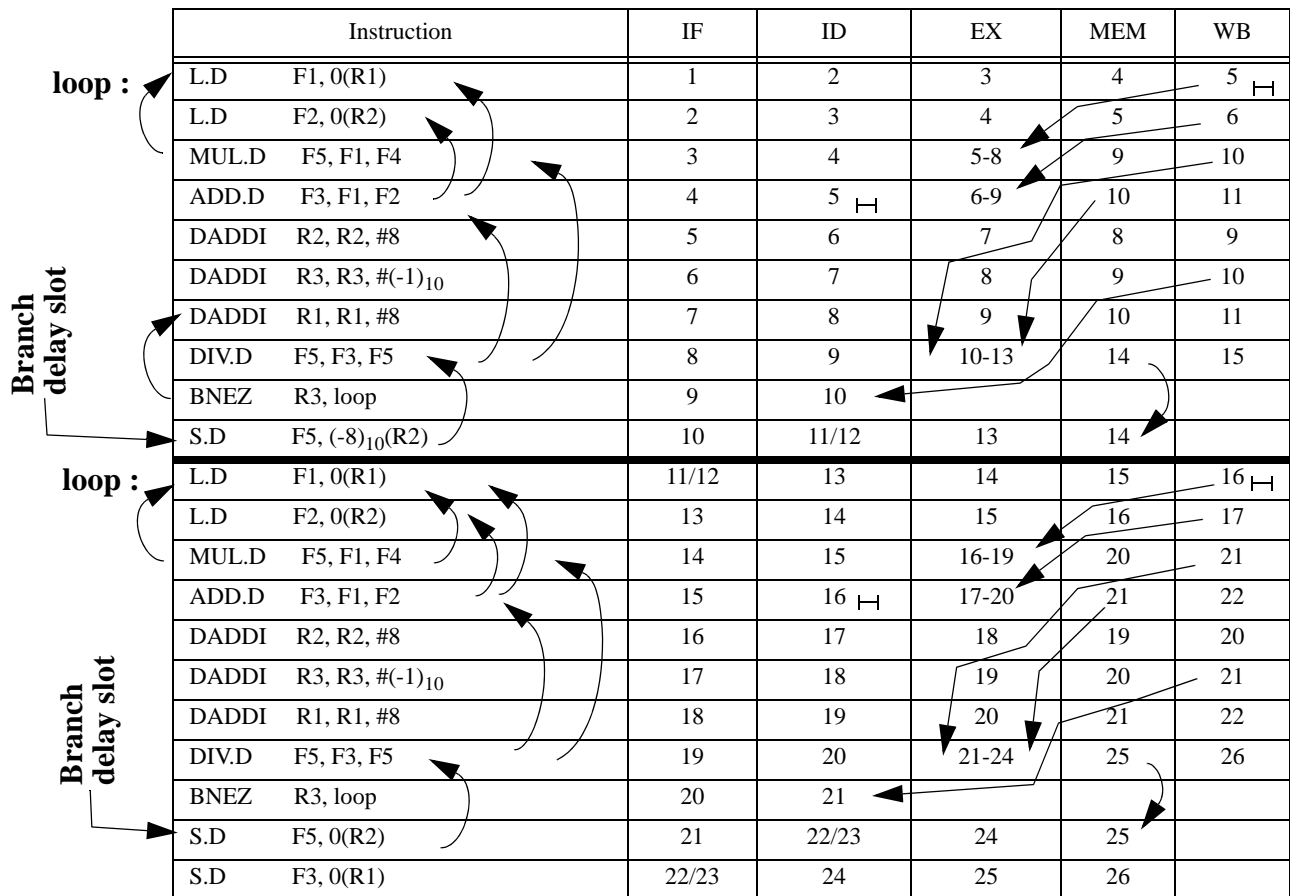
loop :   L.D      F1, 0(R1)      ; Load from vector A
        L.D      F2, 0(R2)      ; Load from vector B
        ADD.D    F3, F1, F2
        MUL.D    F5, F1, F4      ; F4 is already initialized
        DIV.D    F5, F3, F5
        S.D      F5, 0(R2)
        DADDI    R1, R1, #8
        DADDI    R2, R2,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

```

period the last iteration ends as done in class, i.e. together with all the **necessary** forwarding and write-in-the-first-half-read-in-the-second-half cases. Order the instructions so that the new code can run on this model correctly and fast. But, do **not** unroll the code.

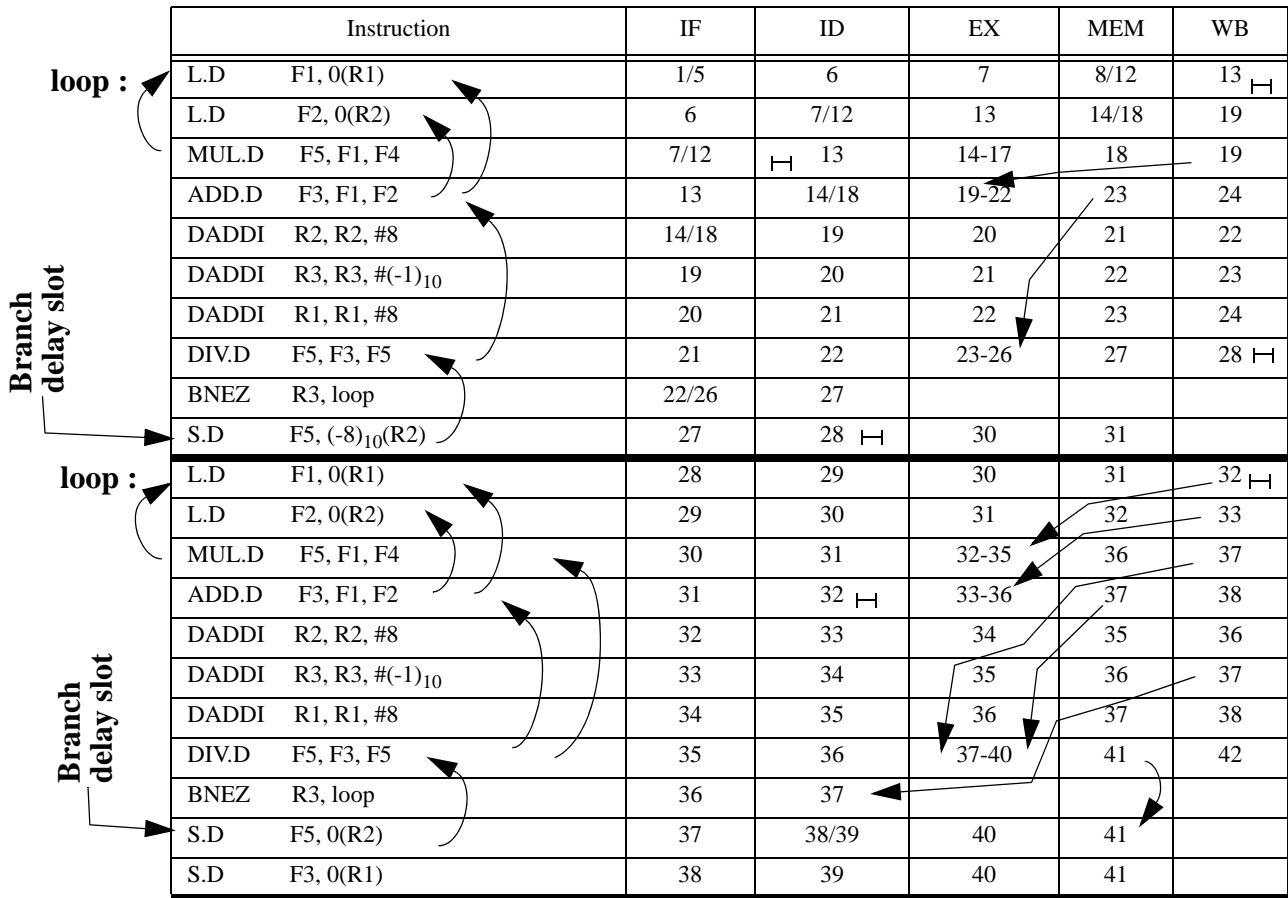
**b)** The L1 cache memories experience cache misses. The memory hierarchy is as described in the MIPS CPU PowerPoint presentation. Assume that it is a **cold** start. Show in which clock period the execution ends as done in lecture presentations, i.e. together with all the necessary forwarding and write-in-the-first-half-read-in-the-second-half cases.

**A9) a)** The execution of the loop with a reordered code and **two** iterations on the *version 2 MIPS* is as follows :



The code execution ends in clock period 26 ! All data hazards are RAW.

b) The execution with cache misses is below. The execution ends in clock period 41 ! All data hazards are RAW.



**Q10)** Consider the following MIPS code below. This code is written for the statically scheduled MIPS Int+FP pipeline (*machine model number 2*).

```

200 DADD R8, R9, R10 ; R9 = 1000 and R10 = 400 initially
204 LD R11, 0(R8)
208 L.D F0, 0(R12) ; R12 initially has 2400
20C L.D F2, 0(R11)
210 L.D F4, 0(R13) ; R13 initially has 3400
214 DIV.D F6, F0, F2
218 MUL.D F8, F6, F4
21C DADDI R10, R10, #8
220 DADDI R14, R14, #(-1)10 ; R14 initially has 2
224 DADDI R12, R12, #8
228 ADD.D F10, F8, F10 ; F10 has already been initialized
22C BNEZ R14, (-12)10
230 DADDI R13, R13, #8
234 S.D F10, 0(R13)

```

Show the execution of the loop, including forwardings, until the code completes on the statically scheduled MIPS Int+FP pipeline (*machine model number 2*) as done in class. Assume that the ADD.D, MUL.D and DIV.D latencies

are 3, 6 and 24, respectively. When does the loop end ? Assume that the L1 cache memories take one clock period each and there are **no** cache misses.

**A10)** The reordered program and its execution timing are below.

Instruction			IF	ID	EX	MEM	WB
200	DADD	R8, R9, R10	1	2	3	4	5
204	LD	R11, 0(R8)	2	3	4	5	6
208	L.D	F0, 0(R12)	3	4	5	6	7
20C	L.D	F2, 0(R11)	4	5	6	7	8
210	L.D	F4, 0(R13)	5	6	7	8	9
214	DIV.D	F6, F0, F2	6	7	8 - 32	33	34
218	MUL.D	F8, F6, F4	7	8/32	33 - 39	40	41
21C	DADDI	R10, R10, #8	8/32	33	34	35	36
220	DADDI	R14, R14, #(-1) <sub>10</sub>	33	34	35	36	37
224	DADDI	R12, R12, #8	34	35	36	37	38
228	ADD.D	F10, F8, F10	35	36/39	40 - 43	44	45
22C	BNEZ	R14, (-12) <sub>10</sub>	36/39	40			
230	DADDI	R13, R13, #8	40	41	42	43	44
200	DADD	R8, R9, R10	41	42	43	44	45
204	LD	R11, 0(R8)	42	43	44	45	46
208	L.D	F0, 0(R12)	43	44	45	46	47
20C	L.D	F2, 0(R11)	44	45	46	47	48
210	L.D	F4, 0(R13)	45	46	47	48	49
214	DIV.D	F6, F0, F2	46	47	48 - 72	73	74
218	MUL.D	F8, F6, F4	47	48/72	73 - 79	80	81
21C	DADDI	R10, R10, #8	48/72	73	74	75	76
220	DADDI	R14, R14, #(-1) <sub>10</sub>	73	74	75	76	77
224	DADDI	R12, R12, #8	74	75	76	77	78
228	ADD.D	F10, F8, F10	75	76/79	80 - 83	84	85
22C	BNEZ	R14, (-12) <sub>10</sub>	76/79	80			
230	DADDI	R13, R13, #8	80	81	82	83	84
234	S.D	F10, 0(R13)	81	82	83	84	

**Q11)** Consider the following MIPS-64 piece of code written for the **pipelined** processor :

```

0  LD      R8, 0(R9)           ; R8 is loaded from vector A
4  DADDI   R12, R12, #(-1)10 ; R12 is the loop-end counter
8  DSUB    R8, R8, R10        ; R10 is already initialized with a value
C  SD      R8, 0(R11)        ; Store to vector B
10 DADDI   R9, R9, #8
14 BNEZ    R12, (-6)10
18 DADDI   R11, R11, #8      ; Instruction in the branch delay slot

```

Assume that there is a **memory hierarchy**. Vector **A** starts at **2000** and vector **B** starts at **A000**.

There are physical Level 1 instruction and data caches. Each L1 cache is a 16 KB cache with Direct mapping and with write-back and write allocate. The block length is 32 bytes. The physical memory contains 1 GBytes.

Note that the above code is written for the **pipelined** processor. Assume that **R12 is 2** initially. Clearly **show** which instruction/data is in which cache, which cache block and which physical memory block for **all** instructions and data.

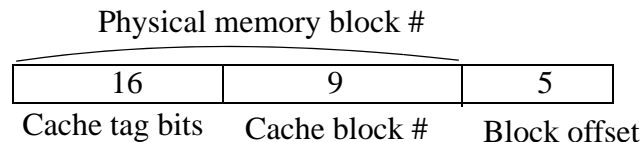
**A11)** The physical memory (PM) has 1 GB =  $2^{30}$  bytes. Therefore, the physical address has **30** bits.

The number of physical memory blocks = (PM size in bytes)/(block size in bytes)  $2^{30}/2^5 = 2^{25}$  blocks.

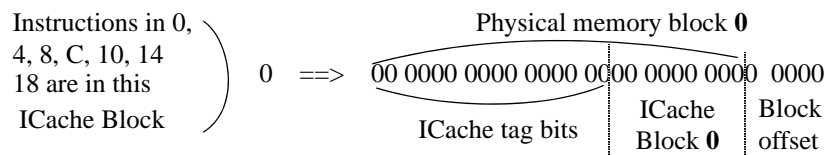
The cache memories have 16KB that is  $2^{14}$  bytes.

The number of cache blocks = (cache size in bytes)/(block size in bytes) =  $2^{14}/2^5 = 2^9 = 512$  blocks in each cache.

The physical address from the point of physical memory and cache memory blocks is as follows :



**The instruction cache, the ICache :**



**The data cache, the DCache :**

