CS433 Homework 3
(Chapter 3)

Assigned on 10/3/2017
Due in class on 10/5/2017

Instructions:

1. Please write your name and NetID clearly on the first page.
2. Refer to the course fact sheet for policies on collaboration.
3. Due **IN CLASS** on 10/5/2017.

Problem 1 [50 points]

In this problem, we explore the ability of the compiler to schedule code. Let's assume the pipeline with following characteristics:

- Unless otherwise specified, its properties are like those in the MIPS pipeline we studied in class.
- The pipeline will not stall due to dependences.
- There is 1 integer functional unit, taking 1 cycle to perform integer addition (including effective address calculation for loads/stores), subtraction, and logic operations.
- There is 1 FP/integer multiplier, taking 7 cycles to perform any multiply. It is pipelined.
- There is 1 FP adder, taking 3 cycles to perform FP additions and subtractions. It is pipelined.
- There is 1 FP/integer divider, taking 10 cycles. It is **not** pipelined.
- There is full forwarding and bypassing, including forwarding from the end of an FU to the MEM stage for stores.
- Loads and stores complete in one cycle. That is, they spend one cycle in the MEM stage after the effective address calculation.
- There are as many registers, both FP and integer, as you need.
- Branches are resolved in ID and there is one branch delay slot (after the branch) that you need to fill.
- If multiple instructions finish their EX stages in the same cycle, then we will assume they can all proceed to the MEM stage together. Similarly, if multiple instructions finish their MEM stages in the same cycle, then we will assume they can all proceed to the WB stage together. In other words, for the purpose of this problem, you are to ignore structural hazards on the MEM and WB stages.
- Assume that operands are read in ID stage and written in WB stage.

Consider the following loop where (1) R1 and R2 contain the memory addresses of two arrays of floating point numbers and (2) R5 is initially set to 6.

```
Loop:  L.D      F2    0(R1)
       MUL.D    F4    F2    F2
       L.D      F3    0(R2)
       MUL.D    F5    F3    F3
       ADD.D    F5    F5    F4
       ADD.D    F4    F2    F3
       DIV.D    F5    F5    F4
       S.D      F5    0(R1)
       DADDUI   R1    R1    #8
       DADDUI   R2    R2    #8
       DSUBUI   R5    R5    #1
       BNEZ     R5    Loop
```

Part A [5 points]

Explain what the code does.

[Answer]
It iterates the arrays and updates the first array as below.

```
while (i != 0):
        A[i] = (A[i]^2 + B[i]^2) / (A[i] + B[i])
        i += 1
```

[Grading Scheme]
3 points for the correct loop structure

Part B [10 points]

Rewrite this loop to insert NOPs to explicitly express stalls due to dependences. For this part, do not reorder instructions yet. If there is a dependence between two instructions, insert NOPs to separate the instructions so that the instruction can be issued without stalls. Notice that some dependencies would require you to insert multiple NOPs in a row. Since instruction reordering is not allowed in this part, insert a NOP in the branch delay slot (effectively stalling 1 cycle after the branch). Complete the code below and provide reasons for all NOPs inserted. Some parts are provided as an example.

Inst#	Instruction					Reasons for NOP
1	Loop:	L.D	F2	0(R1)		
2		NOP				RAW on F2
3		MUL.D	F4	F2	F2	
4		L.D	F3	0(R2)		
5		NOP				RAW on F3
6		MUL.D	F5	F3	F3	
7		NOP				RAW on F4 and F5
8		NOP				RAW on F4 and F5
9		NOP				RAW on F4 and F5
10		NOP				RAW on F5
11		NOP				RAW on F5
12		NOP				RAW on F5
13		ADD.D	F5	F5	F4	
14		ADD.D	F4	F2	F3	
15		NOP				RAW on F4 and F5
16		NOP				RAW on F4
17		DIV.D	F5	F5	F4	
18		NOP				RAW on F5
19		NOP				RAW on F5
20		NOP				RAW on F5
21		NOP				RAW on F5
22		NOP				RAW on F5
23		NOP				RAW on F5
24		NOP				RAW on F5
25		NOP				RAW on F5
26		S.D	F5	0(R1)		
27		DADDUI	R1	R1	#8	
28		DADDUI	R2	R2	#8	
29		DSUBUI	R5	R5	#1	
30		NOP				RAW on R5
31		BNEZ	R5	Loop		
32		NOP				Branch delay slot

[Grading Scheme] 0.5 points for each row (listing one dependence is fine); max 10 points

Part C [10 points]

Now reschedule the loop. You can change immediate values and memory offsets and reorder instructions, but do not change anything else (e.g., you cannot use more registers). Complete the code below and explain any remaining NOPs. Some parts are provided as an example.

Inst#	Instruction					Reasons for NOP
1	Loop:	L.D	F2	0(R1)		
2		L.D	F3	0(R2)		
3		MUL.D	F4	F2	F2	
4		MUL.D	F5	F3	F3	
5		NOP				RAW on F4 and F5
6		NOP				RAW on F4 and F5
7		NOP				RAW on F4 and F5
8		NOP				RAW on F4 and F5
9		NOP				RAW on F5
10		ADD.D	F4	F2	F3	
11		ADD.D	F5	F5	F4	
12		NOP				RAW on F4 and F5
13		NOP				RAW on F5
14		DIV.D	F5	F5	F4	
15		DADDUI	R1	R1	#8	
16		DADDUI	R2	R2	#8	
17		DSUBUI	R5	R5	#1	
18		NOP				RAW on F5 and R5
19		NOP				RAW on F5
20		NOP				RAW on F5
21		NOP				RAW on F5
22		BNEZ	R5	Loop		
23		S.D	F5	-8(R1)		

[Grading Scheme]
0.5 points for each row; max 10 points
The above is only an example of possible solutions. Other correct solutions are accepted (e.g., placing integer operations in different slots).

Part D [15 points]

Now unroll and reschedule the loop for 3 iterations and minimize the number of NOPs. You can remove redundant instructions and use as many registers as you need. Complete the code below and explain any remaining NOPs. Some parts are provided as an example.

Inst#	Instruction					Reasons for NOP
1	Loop:	L.D	F2	0(R1)		
2		L.D	F3	0(R2)		
3		MUL.D	F4	F2	F2	
4		MUL.D	F5	F3	F3	
5		ADD.D	F6	F2	F3	
6		L.D	F2-1	8(R1)		
7		L.D	F3-1	8(R2)		
8		MUL.D	F4-1	F2-1	F2-1	
9		MUL.D	F5-1	F3-1	F3-1	
10		ADD.D	F6-1	F2-1	F3-1	
11		ADD.D	F5	F5	F4	
12		L.D	F2-2	16(R1)		
13		L.D	F3-2	16(R2)		
14		DIV.D	F5	F5	F6	
15		MUL.D	F4-2	F2-2	F2-2	
16		MUL.D	F5-2	F3-2	F3-2	
17		ADD.D	F5-1	F5-1	F4-1	
18		ADD.D	F6-2	F2-2	F3-2	
19		DADDUI	R1	R1	#24	
20		DADDUI	R2	R2	#24	
21		DSUBUI	R5	R5	#3	
22		NOP				RAW on F5-2
23		ADD.D	F5-2	F5-2	F4-2	
24		DIV.D	F5-1	F5-1	F6-1	
25		S.D	F5	-24(R1)		
26		NOP				RAW on F5-1; Structural hazard on DIV
27		NOP				
28		NOP				
29		NOP				
30		NOP				
31		NOP				
32		NOP				
33		NOP				
34		DIV.D	F5-2	F5-2	F6-2	
35		S.D	F5-1	-16(R1)		

36		NOP			RAW on F5-2
37		NOP			RAW on F5-2
38		NOP			RAW on F5-2
39		NOP			RAW on F5-2
40		NOP			RAW on F5-2
41		NOP			RAW on F5-2
42		BNEZ	R5	Loop	
43		S.D	F5-2	-8(R1)	

[Grading Scheme]
0.5 points for each row; max 15 points
The above is only an example of possible solutions. Other correct solutions are accepted.

Part E [10 points]

Unroll the loop 3 times and schedule it for a VLIW machine to take as few cycles as possible. Each VLIW instruction can contain one memory reference, one FP operation, and one integer operation. Leave a row blank if no operation can be scheduled. Again, you can change immediate values and memory offsets, reorder instructions, remove redundant instructions, and use as many registers as you need.

MEM op	FP op	INT op
L.D F2 0(R1)		
L.D F3 0(R1)		
L.D F2-1 8(R1)	MUL.D F4 F2 F2	
L.D F3-1 8(R2)	MUL.D F5 F3 F3	
L.D F2-2 16(R1)	ADD.D F6 F2 F3	
L.D F3-2 16(R2)	MUL.D F4-1 F2-1 F2-1	DADDUI R1 R1 #24
	MUL.D F5-1 F3-1 F3-1	DADDUI R2 R2 #24
	ADD.D F6-1 F2-2 F3-2	DSUBUI R5 R5 #3
	MUL.D F4-2 F2-2 F2-2	
	MUL.D F5-2 F3-2 F3-2	
	ADD.D F5 F5 F4	
	ADD.D F6-2 F2-2 F3-2	
	ADD.D F5-1 F5-1 F4-1	
	DIV.D F5 F5 F6	
	ADD.D F5-2 F5-2 F4-2	
S.D F5 -24(R1)		
	DIV.D F5-1 F5-1 F6-1	
S.D F5-1 -16(R1)		
	DIV.D F5-2 F5-2 F6-2	

		BNEZ R5 Loop
S.D F5-2 -8(R1)		

[Grading Scheme]
0.5 points for each cell; max 10 points
The above is only an example of possible solutions. Other correct solutions are accepted.

Problem 2 [15 points]

In this problem, you will compare the loop unrolling and the software pipelining. Assume the followings:

(1) The processor is a traditional 5-stage pipeline with an in-order core.
(2) There are forwarding paths into the EX stage.
(3) A branch is resolved in the ID stage and it has one branch delay slot.
(4) R1 initially contains the address of an array.
(5) R2 contains the address of the array's last element.
(6) The array contains many elements, so don't worry about the branch not taken.

Consider the following loop.

Inst#	Instruction				
1	Loop:	L.D	F1	0(R1)	
2		ADD.D	F2	F1	F1
3		S.D	F2	0(R1)	
4		DADDUI	R1	R1	#8
5		BNE	R1	R2	Loop
6		NOP			

Part A [3 points]

List all dependencies causing stalls in the code.

[Answer]
2->1: RAW on F1
3->2: RAW on F2
5->4: RAW on R1

[Grading Scheme]
1 point for each dependence

Part B [4 points]

Complete the code below which unroll the loop for 3 iterations and reorder instructions of the loop body to reduce stalls. You can use as many registers as you want and change immediate values and memory offsets. List any remaining dependence causing stalls.

Inst#	Instruction				
1	Loop:	L.D	F1	0(R1)	
2		L.D	F1-1	8(R1)	
3		L.D	F1-2	16(R1)	
4		ADD.D	F2	F1	F1
5		ADD.D	F2-1	F1-1	F1-1
6		ADD.D	F2-2	F1-2	F1-2
7		S.D	F2	0(R1)	
8		S.D	F2-1	8(R1)	
9		S.D	F2-2	16(R1)	
10		DADDUI	R1	R1	#24
11		BNE	R1	R2	Loop
12		NOP			

[Answer]
11->10: RAW on R1

[Grading Scheme]
0.5 point for each row; reordering instructions differently is okay as long as instructions from the same iteration are in order and separated enough; 1 point for correctly identifying dependence

Part C [4 points]

Complete the steady-state code for a software pipelined version below which pipeline over 3 iterations. List any remaining dependence causing stalls.

Inst#	Instruction				
1	Loop:	S.D	F2	0(R1)	
2		ADD.D	F2	F1	F1
3		L.D	F1	16(R1)	
4		DADDUI	R1	R1	#8
5		BNE	R1	R2	Loop
6		NOP			

[Answer]
11->10: RAW on R1

[Grading Scheme]
1 point for each row; 1 point for correctly identifying dependence
Memory offsets of -16 for S.D and 0 for L.D are accepted.

Part D [4 points]

In order to execute 6 iterations of the original loop, how many instructions does each technique have to execute?

[Answer]
Loop unrolling: 24, Software pipelining: 36

[Grading Scheme] 2 points for each

Problem 3 (*for graduate students only*) [10 points]

In this problem, you need to consider the following format for predicated MIPS instructions:

```
(pT) ADD R1, R2, R3
```

where the ADD instruction is predicated on the predicate register pT. Assume a set of 1-bit predicate registers, and compare instructions which set a pair of predicate registers to complementary values:

```
CMP.NE pT, pF = R8, R0
```

The above instruction compare sets the 1-bit predicate registers, pT and pF, based on the "not equal" (NE) comparison relation as follows:

```
pT = (R8 != R0)
pF = !(R8!= R0)
```

So pT is true if R8 is not equal to R0, and pF is the complement of pT. For the following problem, you can assume the availability of any comparison relation with two operands; e.g., .LE for less than or equal to and .GT for greater than.

Part A [4 points]

Using the predicated instructions described above, write the three basic blocks of the following code fragment as a single basic block (i.e., eliminate all branches using predicated instructions).

```
        SUB       R1      R13     R14
        BLT       R1      R4      L1      (branch if R1 < R4)
        ADDI      R2      R2      #1
        SW        R2      0(R7)
        J         L2
L1:     DIV.D     F0      F0      F2
        ADD.D     F0      F4      F2
        S.D       F0      0(R8)
L2:     …
```

[Answer]

```
        SUB       R1      R13     R14
        CMP.LT    pT, pF = R1, R4
        (pF) ADDI R2      R2      #1
        (pF) SW   0(R7)   R2
        (pT) DIV.D F0     F0      F2
        (pT) ADD.D F0     F4      F2
        (pT) S.D  0(R8)   F0
L2:     …
```

[Grading Scheme]

0.5 points for each correctly translated instruction (Note that for the jump instruction, J, the correct translation is to not have any instruction)

Part B [6 points]

What are all the data dependencies in your new code? Indicate the type(RAW, WAW, WAR) of each dependence. Notice that the predicate registers may also cause dependences.

[Answer]

The data dependencies are:

(1) RAW dependency due to R1 between SUB and CMP.
(2) RAW dependency due to pF between CMP and ADDI.
(3) RAW dependency due to pF between CMP and SW.
(4) RAW dependency due to R2 between ADDI and SW.
(5) RAW dependency due to pT between CMP and DIV.D.
(6) RAW dependency due to pT between CMP and ADD.D.
(7) RAW dependency due to pT between CMP and S.D.
(8) RAW dependency due to F0 between ADD.D and S.D.
(9) WAW dependency due to F0 between DIV.D and ADD.D.
(10) WAR dependency due to F0 between DIV.D and ADD.D.

[Grading Scheme]

0.5 points for each correct dependence