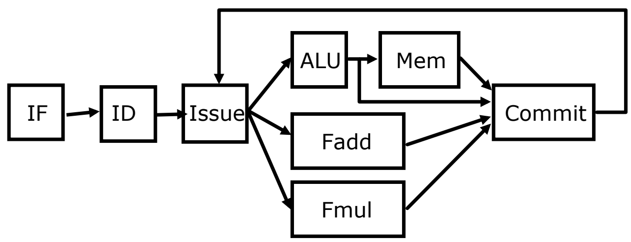
CSE 490/590 Computer Architecture
Homework 2

1. Suppose that you have the following out-of-order datapath with 1-cycle ALU, 2-cycle Mem, 3-cycle Fadd, 5-cycle Fmul, no branch prediction, and in-order fetch and commit.



Consider the following sequence of instructions.

Instruction Number	Instruction
I_1	LD F4, 0 (R1)
I_2	LD F3, 0 (R2)
I_3	FADD F6, F3, F4
I_4	FMUL F1, F6, F3
I_5	LD F5, 0 (R3)
I_6	FADD F2, F5, F4
I_7	FMUL F5, F2, F5
I_8	FADD F3, F1, F4
I_9	FMUL F5, F3, F2
I_{10}	FADD F6, F2, F4

Fill in the ROB and renaming table (next page).

Answer:

Note: Because of the commit stage, the ROB will also hold data (i.e., the answer table does not show the complete picture of the ROB). In other words, we're dealing with the ROB in slides 6-8 in ilp2.pptx. Thus, the renaming table will only hold tags and not the actual data values.

Tag	op	dst	src1	src2
T1	LD	T1	R1	0
T2	LD	T2	R2	0
T3	FADD	T3	T2	T1
T4	FMUL	T4	T3	T2
T5	LD	T5	R3	0
T6	FADD	T6	T5	T1
T7	FMUL	T7	T6	T5
T8	FADD	T8	T4	T1
T9	FMUL	T9	T8	T6
T10	FADD	T10	T6	T1

	R1	R2	R3	F1	F2	F3	F4	F5	F6
I_1							T1		
I_2						T2			
I_3									T3
I_4				T4					
I_5								T5	
I_6					T6				
I_7								T7	
I_8						T8			
I_9								T9	
I_{10}									T10

2. Consider the following instructions. Assume that the initial values for R1, R2, and R3 are all 0.

```
loop:
        SUBI R2, R1, 2
        BNEZ R2, target1
        ADDI R3, R3, 1
target1:
        ADDI R1, R1, 1
        SUBI R4, R1, 3
        BNEZ R4, loop
```

(a) Explain what the code does.

Answer:

Pseudo-code:

if R1 == 2, then R3 = R3 + 1
R1 = R1 + 1
if R1 != 3, then next loop

Thus, the code loops three times (R1 == 0, 1, & 2), and it increment R3 by 1 in the last loop.

(b) Change the code to minimize the number of registers necessary.

Answer: R2 and R4 only hold temporary values, so we can use just one register.

```
loop:
        SUBI R2, R1, 2
        BNEZ R2, target1
        ADDI R3, R3, 1
target1:
        ADDI R1, R1, 1
        SUBI R2, R1, 3
        BNEZ R2, loop
```

(c) Assume that we have a 1-bit branch predictor that stores the result of the last branch and makes the prediction based on the result, i.e., the prediction is "take" if the last branch was taken and the other way round for "not take". Show the results of all predictions throughout the execution.

Answer: Assume that we're starting from "Not Take". In total, the code loops three times, and there are 6 branches.

Branch	Prediction	Actual Result
1st BNEZ (target1)	Not Take	Taken
2nd BNEZ (loop)	Take	Taken
3rd BNEZ (target1)	Take	Taken
4th BNEZ (loop)	Take	Taken
5th BNEZ (target1)	Take	Not Taken
6th BNEZ (loop)	Not Take	Not Taken

(d) Assume that we have one 2-bit branch predictor. Show the results of all predictions throughout the execution.

Answer: Assume that we're starting from "Not Take" & "Right".

Branch	Prediction	Actual Result
1st BNEZ (target1)	Not Take & Right	Taken
2nd BNEZ (loop)	Not Take & Wrong	Taken
3rd BNEZ (target1)	Take & Right	Taken
4th BNEZ (loop)	Take & Right	Taken
5th BNEZ (target1)	Take & Right	Not Taken
6th BNEZ (loop)	Take & Wrong	Not Taken

(e) Assume that we have four 2-bit branch predictors per branch instruction as well as one 2-bit shift register that stores the result of the last two branch instructions (i.e., we have a two-level branch predictor). Show the results of all predictions throughout the execution.

Answer: Assume that we're starting from "Not Take" & "Right" for predictors and "Not Taken" & "Not Taken" for the global history. We use two tables, one table per branch instruction. Each instruction has four predictors since there are four possible cases from the global history (the tables below do not show these cases separately though). Thus, out of 4 branches, the 3rd and 5th branches share the same predictor; all others use different ones.

Branch	History (Last Two)	Prediction	Actual Result
1st BNEZ (target1)	Not Taken & Not Taken	Not Take & Right	Taken
3rd BNEZ (target1)	Taken & Taken	Not Take & Right	Taken
5th BNEZ (target1)	Taken & Taken	Not Take & Wrong	Not Taken

Branch	History (Last Two)	Prediction	Actual Result
2nd BNEZ (loop)	Not Taken & Taken	Not Take & Right	Taken
4th BNEZ (loop)	Taken & Taken	Not Take & Right	Taken
6th BNEZ (loop)	Taken & Not Taken	Not Take & Right	Not Taken

3. (Example on p.77) Consider the following code:

```
loop:
        LD F0, 0(R1)
        FADD F4, F0, F2
        ST F4, 0(R1)
        ADDI R1, R1, -8
        BNE R1, R2 loop
```

Show how to unroll the loop so that there are four copies of the loop body, assuming that R1 - R2 is initially a multiple of 32, which means that the number of loop iterations is a multiple of 4. Eliminate any obviously redundant computations and do not reuse any of the registers.

Answer: Please refer to the textbook.

4. (Example on p.116) Suppose we have a VLIW that could issue two memory references, two FP operations, and one integer operation or branch in every clock cycle. Show an unrolled version of the loop $x[i] = x[i] + s$ (see p.76 for the MIPS code) for such a processor. Unroll as many times as necessary to eliminate any stalls. Ignore delayed branches.

   Answer: Please refer to the textbook.

5. Suppose that you have a multithreading single-issue in-order datapath with 1-cycle ALU, 2-cycle Mem, 3-cycle Fadd, 5-cycle Fmul, no branch prediction, and in-order fetch and commit. Consider the following instructions:

```
loop:
        LD F0, 0(R1)
        FADD F3, F3, F2
        ADDI R1, R1, -8
        BNE F0, F2, loop
```

What is the minimum number of threads necessary to fully utilize the datapath for each of the following strategies?

(a) Fixed switching: the CPU switches to a different thread every cycle in a round-robin fashion.

Answer: We need to fill in the pipeline bubbles with useful instructions. Thus, the general strategy is to see the worst case scenario — see where the pipeline bubbles are and how many cycles we are wasting because of the bubbles. That's the minimum number of threads we need to keep the pipeline busy.

In general, problems can be memory operations, control hazards, and data hazards. In our code above, the problematic one is not BNE because we assume that there is no branch prediction. In the worst case, during EX for the BNE, we might know that we need to insert pipeline bubbles for IF and ID. Thus, we need at least 3 threads to replace the bubbles.

(b) Data-dependent switching: the CPU switches to a different thread when an instruction cannot proceed due to a dependency.

Answer: The general strategy is the same. We consider the worst case and see how many threads we need to fill in the possible bubbles.

There are two dependencies. One is between LD and ADDI on R1, and the other is LD and BNE on F0. However, these dependencies do not lead to stalls in a simple pipeline. Thus, we are not able to improve the performance even with more threads.