ECE/CS 552: Introduction to Computer Architecture
ASSIGNMENT #3
Due Date: At the beginning of lecture, October 20th, 2010

<u>This homework is to be done individually.</u> <u>Total 4 Questions, 100 points</u>

1. (20 pts.) In this exercise, we examine how data dependencies affect execution in the basic five-stage pipeline described in the textbook. Problems in this exercise refer to the following sequence of instructions:

a.	lw $1, 40 ($6) add $6, $2, $2 sw $6, 10($3)
b.	lw $5, -16 ($5) sw $5, -16 ($5) add $5, $5, $5

1) (3 pts.) Indicate dependences and their type.

Solution:

	Involved Reg #1	Involved Reg #2	Type of Dependency
Sequence a	Rs of lw	Rd of add	Anti
	Rd of add	Rt of sw	True
Sequence b	Rs of lw	Rt of lw	Anti
	Rt of lw	Rt of sw	True
	<u>Rt of lw</u>	<u>Rs of sw</u>	<u>True</u>
	Rt of lw	Rs of add	True
	Rt of lw	Rt of add	True
	Rt of lw	Rd of add	Output
	Rs of lw	Rd of add	Anti
	Rt of sw	Rd of add	Anti
	Rs of sw	Rd of add	Anti
	Rs of add	Rd of add	Anti
	Rt of add	Rd of add	Anti

2) (3 pts.)Assume there is no forwarding in this pipelined processor. Indicate hazards and add “nop” instructions to eliminate them.

Solution: Hazards exist in true dependencies in the previous table.
After inserting NOP, the hazards are eliminated and the new instruction sequences are:

a.	lw $1, 40 ($6) add $6, $2, $2 NOP NOP sw $6, 10($3)
b.	lw $5, -16 ($5) NOP NOP sw $5, -16 ($5) add $5, $5, $5

3) (3 pts.)Assume there is full forwarding. Indicate hazards and add “nop” instructions to eliminate them.

Solution: With full forwarding, only the True dependency in lw causes hazard (underlined in the dependency table). The new instruction sequences are:

a.	lw $1, 40 ($6) add $6, $2, $2 sw $6, 10($3)
b.	lw $5, -16 ($5) NOP sw $5, -16 ($5) add $5, $5, $5

The remaining problems in this exercise assume the following clock cycle times:

	Without forwarding	With full forwarding	With ALU-ALU forwarding only
a.	300 ps	400 ps	360 ps
b.	200 ps	250 ps	220 ps

4) (4 pts)What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speed-up achieved by adding full forwarding to a pipeline that had no forwarding?

Solution: with full forwarding, a needs 3 cycles and b needs 4 cycles.
With no forwarding, both a and b needs 5 cycles.
Therefore speedup for a = (5*300)/(3*400)=1.25
For b = 5*200 / 4*250 = 1.

5) (3 pts.)Add "nop" instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to EX stage).

Solution:

a.	lw $1, 40 ($6) add $6, $2, $2 sw $6, 10($3)
b.	lw $5, -16 ($5) NOP NOP sw $5, -16 ($5) add $5, $5, $5

6) (4 pts.)What is the total execution time of this instruction sequence with only ALU_ALU forwarding? What is the speed-up over a no-forwarding pipeline?

Solution: Total execution time of a = 360 * 3 = 1080 ps
Total execution time of b = 220 * 5 =1100 ps
Speedup: a = 1500/1080 = 1.39
b = 1000/1100 = 0.91

2. (20 pts.) This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes:

	Branch Outcomes
a.	T, T, NT, T
b.	T, T, T, NT, NT

1) (3 pts.)What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

Solution:

	Always Taken Predictor	Always Not Taken Predictor
a	75%	25%
b	60%	40%

2) (3 pts.)What is the accuracy of the two-bit predictor **for the first four** branches in this pattern, assuming that the predictor starts off in the bottom left state (predict not taken) from Figure 1, or Figure 4.63 from the textbook (4th edition).

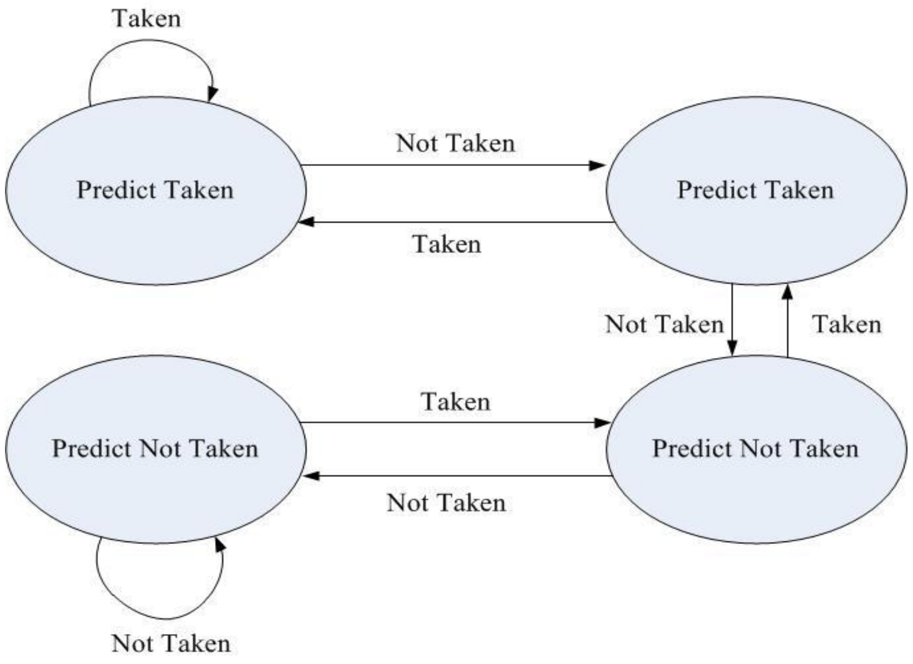


Figure 1: State Machine of 2-bit branch predictor

Solution:

a: 0%; b: 20% (only the third T will be predicted correctly)

3) (3 pts.)What is the accuracy of the two-bit predictor if this pattern is repeated forever?

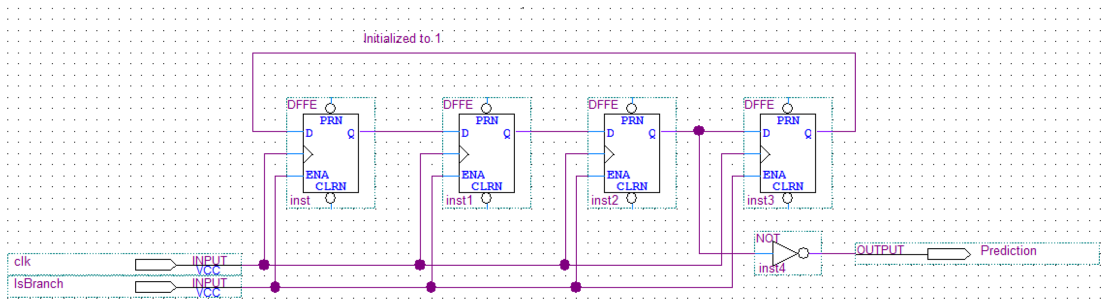
   Solution: a: approximately 75% (NT will be predicted wrong)

   b: approximately 40% (2 NT's and the first T will be predicted wrong)

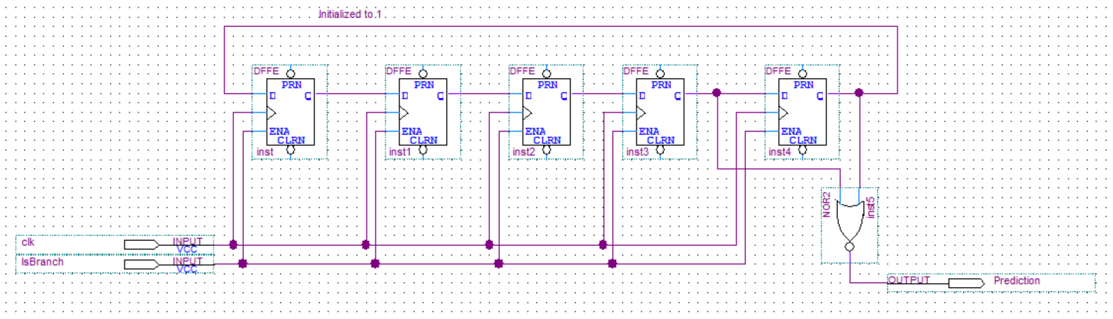
4) (5 pts.) Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever. Your predictor should be a sequential circuit with one output that provides a prediction (1 for taken, 0 for not taken) and no inputs other than the clock and the control signal that indicates that the instruction is a conditional branch.

Solution: The predictor can be built as a Moore state machine. Here are examples using one-hot state machine:

a:



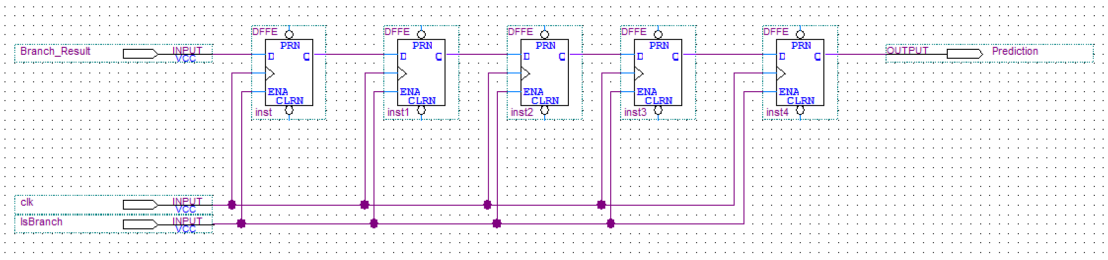
b:



(6 pts.) Repeat 4), but now your should be able to eventually (after a warm-up period during which it can make wrong predictions) start perfectly predicting both this pattern and its exact opposite (e.g., a: NT, NT, T, NT). Your predictor should have an input that tells it what the real outcome was. Hint: this input lets your predictor determine which of the two repeating patterns it is given.

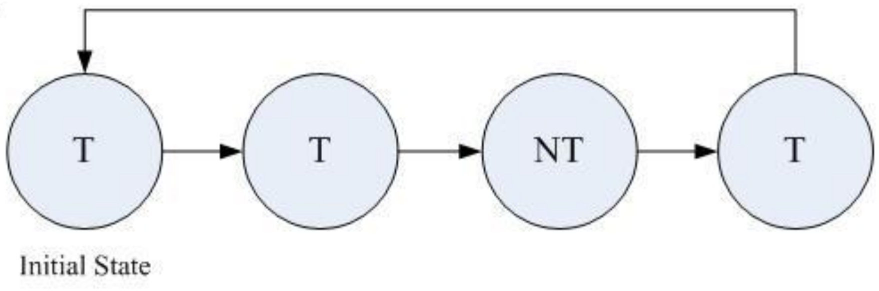
Solution: The easiest way is to use a shift register to record the history. The length of the shift register should be the same (or longer) as the period T of branch patterns. The predictor's output is the same as the branch outcome T cycles before.

Here is an example for sequence b.

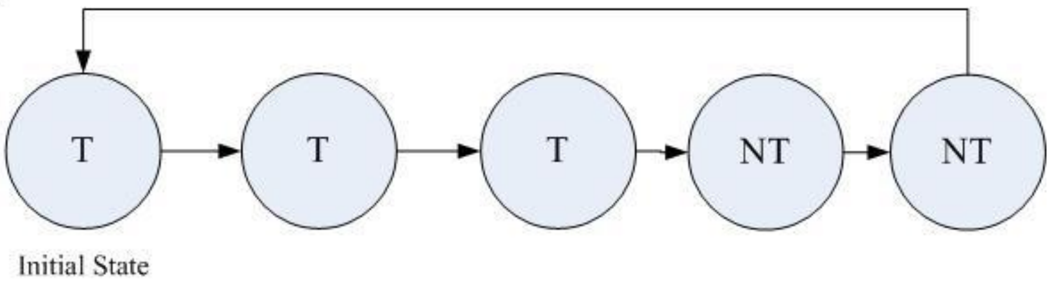


If you are using state diagram to solve problem 2(4) and 2(5), you must specify the input/output of the states, conditions that triggers the state to change, and the initial state. The examples of state diagrams are shown below (for (5), assuming the pattern changes only after the previous pattern finishes):

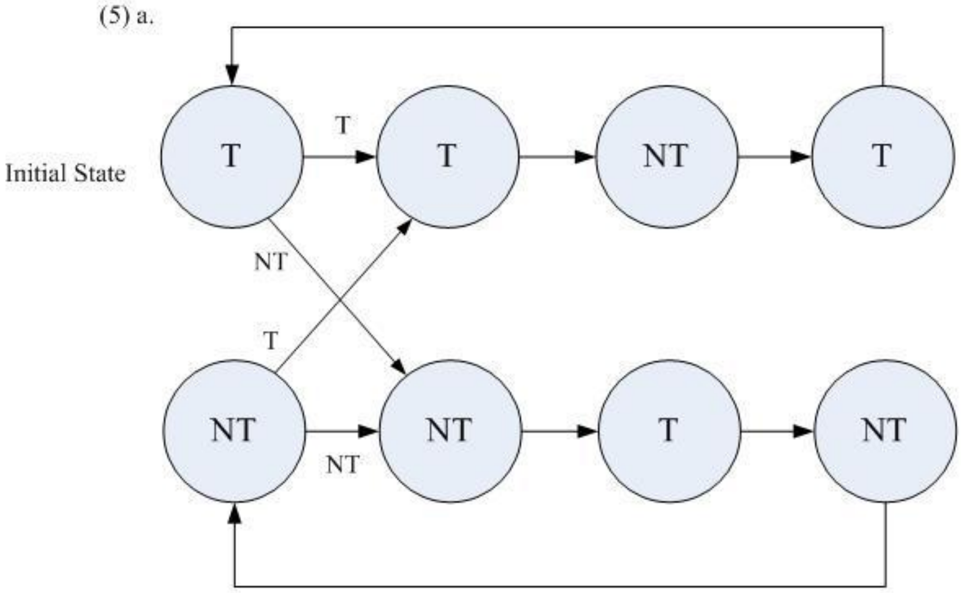
(4) a.



(4) b.

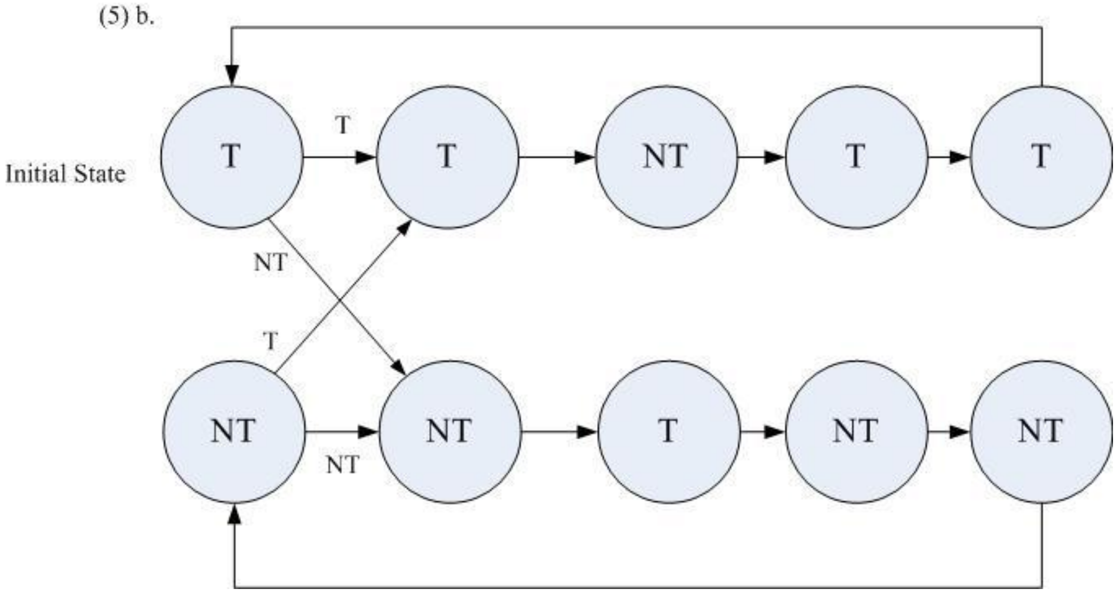


(5) a.



Initial State
T
T
NT
T
T
NT
NT
T
T
NT
NT
T
NT

(5) b.



Initial State
T
T
NT
T
T
T
NT
NT
T
NT
NT
T
NT
NT

(30 pts.) Barrel Shifter Design in Quartus II
Design a 16-bit four-level Barrel Shifter. It has three control signals: **Shift/Rotate, A/L** (Arithmetic/Logic) and **Direction,** one 4-bit control input **Shamt** (Shift amount), sixteen-bit data inputs **D[15:0]**, sixteen-bit data outputs **Q[15:0]**. Figure 2 shows its block diagram.

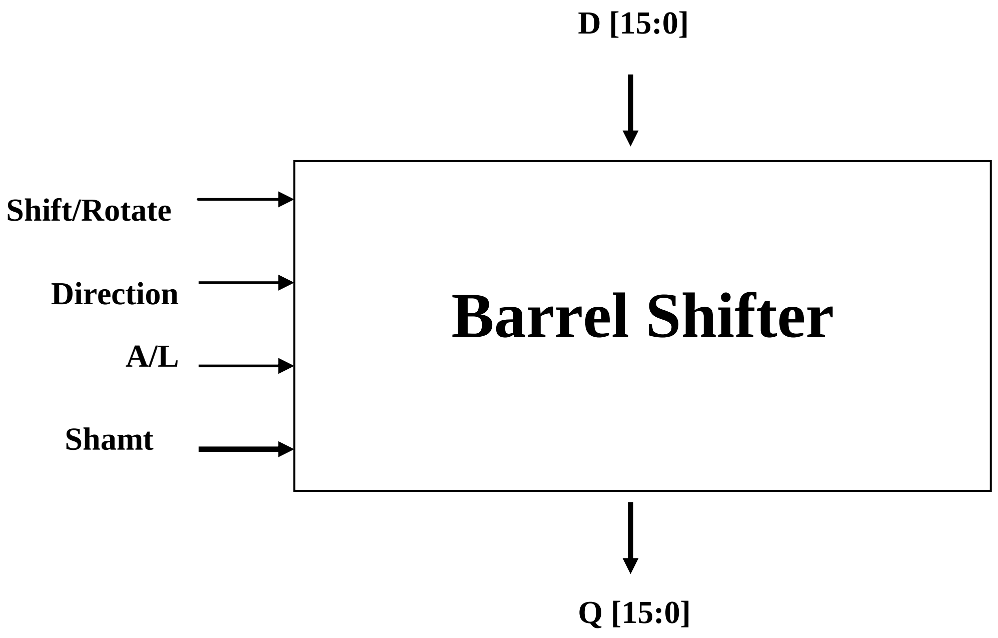


Figure 2: Block diagram of Barrel Shifter

When **Shift/Rotate** signal is low, "0" is shifted in Q. The number of bits shifted in is specified in **Shamt** and the direction of shift operation is specified in **Direction** (Left = 0, Right = 1). If **A/L** is low, the shifter performs Logic Shift operation; If **A/L** is high, the shifter performs Arithmetic shift. When **Shift/Rotate** is high, those bits shifted out are shifted in on the other side (instead of "0").
For example (**A/L** = 0, **Shamt** = 1 and **Direction** = Left):

Shift/Rotate	D[15:0]	Q[15:0]
0	8000	0000
1	8000	0001

You should turn in:
1) A schematic of your design. If your top level design contains low level functional blocks (e.g. you generate symbols and use them in the top level), you should also provide the schematic of those functional blocks. You are only allowed to use either schematic or structural HDL in your design.
2) A testbench (vector waveform file .vwf) for your design. You should provide enough test cases in your testbench in order to test the full functionality of your design. It must contain the following cases:

Shift/Rotate	Direction	A/L	Shamnt
0	0	0	0
0	0	0	1
0	0	0	2
0	1	0	8
1	0	0	1
1	0	0	2
1	1	0	8
0	1	1	4
0	1	1	4

You may choose different values for D in order to prove the correctness of your design.
3) A simulation result of your design. Functional simulation is required. You should include all inputs and outputs in the simulation file, and highlight **D[15:0]** and **Q[15:0]** in the waveform.

4. (30 pts.) Datapath and Control Logic Design in Quartus II.
Build a single-cycle 16-bit datapath with your ALU, Register File and the Barrel Shifter (shifter can be incorporated into the ALU). Then design a control logic that correctly handles the execution of arithmetic instructions in your datapath. The arithmetic instructions are **ADD**, **SUB**, **AND OR**, **SLL**, **SRL** (Shift Left/Right Logic,), **SRA** (Shift Left/Right Arithmetic) and **RL** (Rotate Left, **Shift/Rotate** = 1, **Direction** = 0). The machine code of each instruction is shown in the following table:

Operation	ADD	SUB	AND	OR	SLL	SRL	SRA	RL
Machine Code	0000	0001	0010	0011	0100	0101	0110	0111

Figure 3 shows the block diagram of the datapath and control logic.

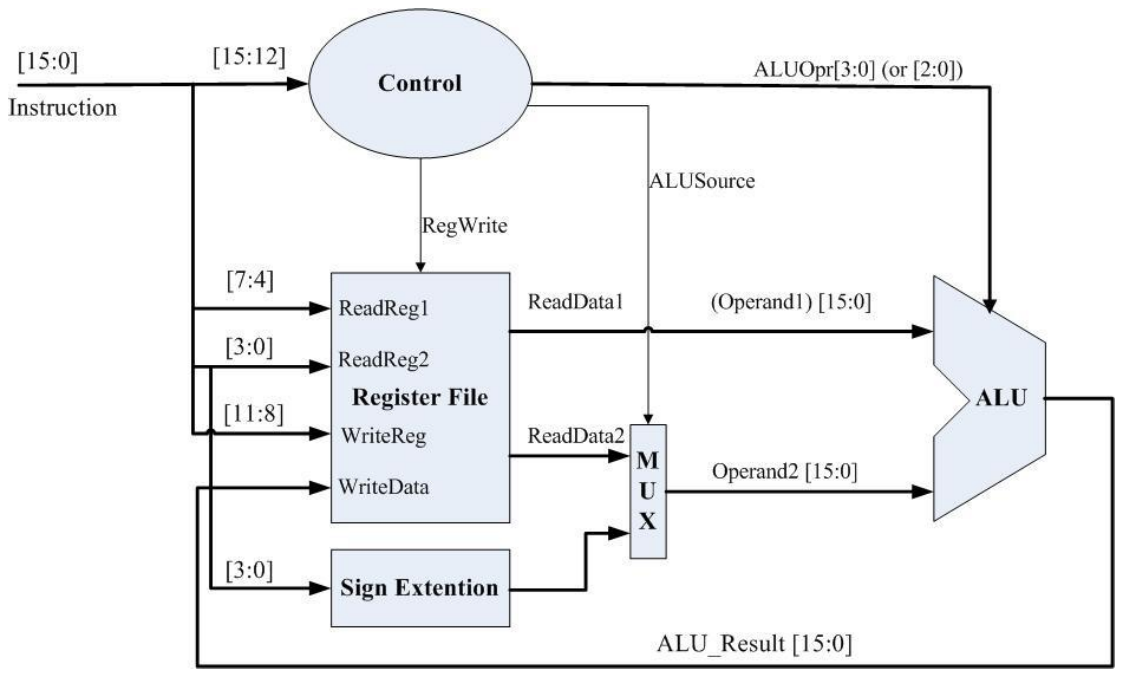


Figure 3: Block diagram of single-cycle datapath and control logic

The **ADD, SUB, AND, OR** have a three address format. The assembly level syntax for these instructions is:

Opcode	Rd	Rs	Rt
15 12	11 8	7 4	3 0

These instructions will execute: Rd <== Rs (Opr) Rt. The two operands are Rs and Rt and the destination is register Rd.

The **SLL, SRL, SRA** and **RL** have a two address and one immediate format. The assembly level syntax for these instructions is:

Opcode	Rt	Rs	imm
15 12	11 8	7 4	3 0

These instructions will execute: Rt <== Rs (Opr) imm. The two operands are Rs and imm, and the destination is register Rt. The 4-bit immediate value imm is used as the **Shamnt** input of the Barrel Shifter.

How to test your design:

Since we do not have memory operations yet, we use an external input bus to initialize your ALU. Use a multiplexor to select the source of **WriteData**, and use another input signal to control the multiplexor: "0" indicates initialization phase, in which the external input value is written into the registers; "1" indicates the testing phase, in which ALU result is written into the registers.

Note: This external initialization scheme is used only for testing purpose. Later in your project you will need to remove them from your design.

You should turn in:

1) A schematic of your design. If your top level design contains low level functional blocks (e.g. you generate symbols and use them in the top level), you should also provide the schematic of those functional blocks. You are allowed to use schematic, structural HDL or behavioral HDL. Behavioral HDL is only allowed to use in control logic, not datapath elements. In the schematic, you may use one megafuntion block BUSMUX as your multiplexors for buses in your datapath. Other symbols must be primitives.

2) A testbench (vector waveform file .vwf) for your design. You should provide enough test cases that cover functionality verification for all of the 8 operations.

3) A simulation result of your design. Functional simulation is required. Include all the data values and control signals, and highlight each instruction code and its ALU result value.