Lab 8 : ALU Control and Datapath

Name:

Your Name

Sign the following statement:

On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work

Type Name

1 Objective

The main objectives of this lab are

- Build the basic elements of the MIPS datapath that will eventually be used to build a complete processor.

2 Pre-requisite

For this lab you are expected to be familiar with MIPS single cycle processor from sections 4.2, 4.3 and 4.4 of the textbook.

3 Data Memory

You will be implementing the memory components, ALU control block, and next PC logic in this lab. Instruction memory and data memory are the most crucial components since they are used to store instructions and data respectively. Instructions are read from instruction memory that process data stored in the data memory. Under normal circumstances, instructions and data are stored in different parts of the same large memory blocks. For ease in this lab, we will use different memory blocks for data and instructions.

Write a behavioral Verilog code to implement the data memory of size 64 words, where a word is 32 bits in size. Remember that your code must be synthesizable. A memory write only commits to the memory on the negative edge of a clock, where read operations occur on the positive edge of the clock. In addition, memory reads and writes must have a delay of 20. Be sure to use non-blocking assignments

The data memory must have the following inputs:

Address The 32-bit address where the data is either written or read.

Write Data The 32-bit data that is to be written at the address specified by the *Address* input above.

MemRead A single bit signal which should be set when you wish to read data from the memory.

MemWrite A single bit signal which should be set when you wish to write data to the memory.

Clock The clock signal to synchronize data writes.

The data memory must have the following outputs:

Read Data The 32-bit data that is read from the address specified by the *Address* input when a memory read operation is performed.

Note: `MemRead` and `MemWrite` must not be active at the same time.

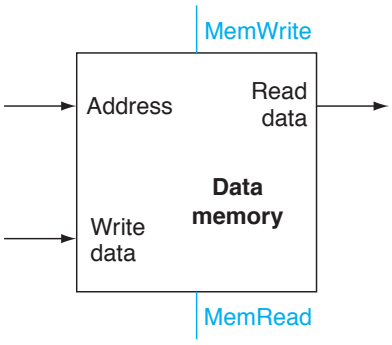

Fig. 1: Memory Unit

Your Data memory module must have the port definition.

```
module DataMemory(ReadData, Address, WriteData,
MemoryRead, MemoryWrite, Clock);
```

```
input [31:0] Address, WriteData;
input MemoryRead, MemoryWrite, Clock;
output [31:0] ReadData;
/* write your code here */
endmodule
```

Use the Verilog code `DataMemoryTest.v` for testing your code.

Demonstrate your program to the TA;

4 ALU Control

As described in the previous lab, the ALU control block, takes the following 2 as inputs:

- ALUOP(4 bits) from the main control unit
- Function field in an R-type instruction (bits [5:0] of the instruction).

The output of the ALU control unit is a 4-bit signal that directly controls the ALU block.

Use the following form as the module interface:

```
module ALUControl(ALUCtrl, ALUop, FuncCode);
input [3:0] ALUop;
input [5:0] FuncCode;
output ALUCtrl[3:0];
/* write your code here */
endmodule
```

Note: The `ALUop` you will use here is different from previous labs. For an R-type instruction, `ALUop` should be 1111. For I-type instructions, `ALUop` should be the value of `ALUCtrl` for that instruction. Your output should have a delay of 2.

Write a test bench to test your code with the following cases. Fill in the expected ALUCtrl for the given inputs before you proceed. Remember that your code must be synthesizable.

Test Case	ALUop	FuncCode	Expected ALUCtrl
1	0010	XXXXXX	
2	0110	XXXXXX	
3	1111	000000	
4	1111	000010	
5	1111	100000	
6	1111	100010	
7	1111	100100	
8	1111	100101	
9	1111	101010	

Use `$monitor` or `$display` commands to print the inputs (`ALUop`, `FuncCode`) and the output (`ALUCtrl`) values for each test case. Copy and paste the test result output:

5 Next PC Logic

Write a behavior model for calculating the next PC for an instruction. It will use information from the processor control module and the ALU to determine the destination for the next PC.

Use the following module interface:

```
module NextPClogic(NextPC, CurrentPC, JumpField,
                   SignExtImm32, Branch, ALUZero, Jump);
input [31:0] CurrentPC, SignExtImm32;
input [25:0] JumpField;
input Branch, ALUZero, Jump;
output [31:0] NextPC;
/* write your code here */
endmodule
```

Where `JumpField` is the jump field from the current instruction and `SignExtImm32` is the sign extended lower 16 bits of the current instruction. `Branch` is true if the current instruction is a branch instruction, `Jump` is true if the current instruction is a jump, and `ALUZero` is the `Zero` output of the ALU.

Any additions with a constant should have a delay of 1, general addition should have a delay of 2, and any multiplexers should have a delay of 1 (This includes statements inside if/else statements). Write a test module to test your module's correct operation. In your test code, use `$display` command to print the inputs (CurrentPC, JumpField, SignExtImm32, Branch, ALUZero, Jump) and the output (NextPC). Copy and paste the test result output:

6 Deliverables

Please turn-in the following:

- Your Verilog code along with the test code for all three components. Ensure that your program is clearly commented.
- Files with waveform trace from DVE (print to file) when running your testbench for each of the components. Be sure to set it to require enough pages that the full signal values are readable in the wavefore trace. Postscript format is fine.