ECEN 651: Microprogrammed Control of Digital Systems
Department of Electrical and Computer Engineering
Texas A&M University

Laboratory Exercise #4

Objective

The objective of lab this week is to complete the design of ALU module and ALU Control module of a single-cycle processor.

Procedure

1. Design the ALU. The Arithmetic Logic Unit (ALU) in MIPS is a combinational logic module that produces an arithmetic or logic result from two input operands based on the control signals input to the module. Figure 1 provides the port interface diagram of the ALU you will build in this lab. The two operand buses, BusA and BusB, are each 32-bits wide, while the control bus is 4-bits wide. The outputs from the ALU module include a 32-bit result bus, BusW, and a Zero signal, which is HIGH when the result equals zero. Please note that this ALU does not provide support for arithmetic exceptions; however, we will fix this in later labs. The ALU can be thought of as combination of arithmetic and logic blocks, each sharing the same inputs, such that the outputs of each of the blocks are multiplexed to create the final result. In this illustration, the ALU control signals act as the selection lines of the multiplexer.

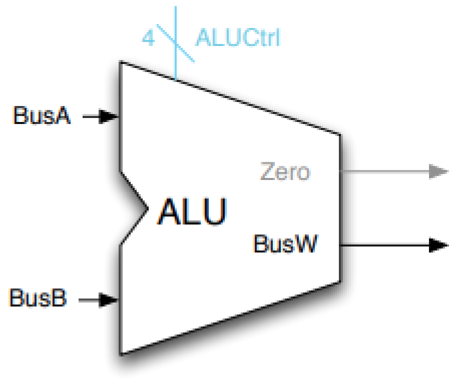
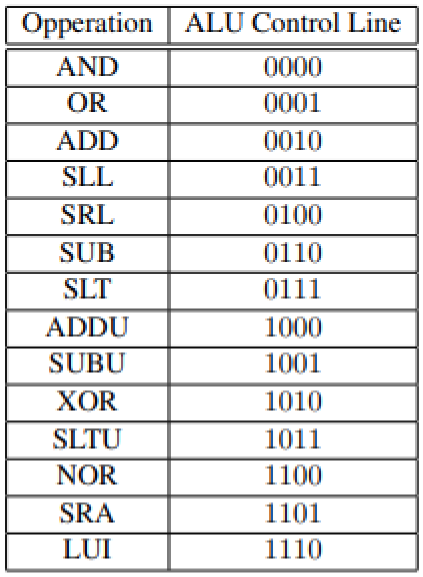


Figure 1: ALU port interface

(1) Write a Verilog module to implement the ALU discussed above. Use the following module interface:
module ALU(BusW, Zero, BusA, BusB, ALUCtrl);
Helpful Hints:

• Use Figure 2 in conjunction with the MIPS reference card on the laboratory website as a guide.
• Your code should make use of the case statement.
• Since exceptions are not being supported initially, ADD and ADDU are identical operations. The same goes for SUB and SUBU.
• SLT must compare 2 two's complement numbers. This can be done by first negating the sign bits of the two operands and then performing an unsigned comparison



Opperation	ALU Control Line
AND	0000
OR	0001
ADD	0010
SLL	0011
SRL	0100
SUB	0110
SLT	0111
ADDU	1000
SUBU	1001
XOR	1010
SLTU	1011
NOR	1100
SRA	1101
LUI	1110

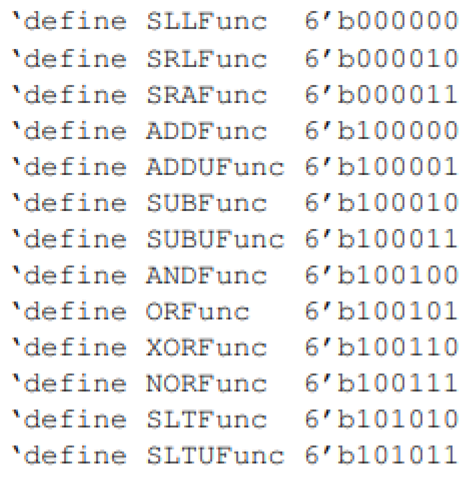
Figure 2: Arithmetic Logic functions and associated control signals

(2) Simulate the operation of your hardware using ISE and the ALU test bench provided on the laboratory website. Paste the output of the simulator into your lab write-up.
(3) Synthesize your design using the Device properties provided in Lab 3. Fix all warnings and errors and ensure the compiled hardware contains no latches or flip-flops. Provide the summary results of the synthesis process in your lab write-up.

2. Design the ALU control logic. The process of decoding an instruction into a control code that the ALU can understand (Figure 2) is split up into two blocks. The first block is contained within the control unit and will be addressed shortly. The second block is the ALU control block shown in Figure 1, which takes in an ALU op control code (Note the control signal is thicker indicating a bus) from the control unit and the function field from the current instruction. The ALU control logic should asynchronously function as follows:
• When the ALU op bus is not equal to 4'b1111, the ALU op code should be passed directly to the ALU.
• When the ALU op bus is equal to 4'b1111, the function code should be used to determine the ALU control code.
For R-type instructions, the control unit simply sets the ALU op to 4'b1111 and the ALU control logic will interpret the arithmetic or logic function needed to carry out the instruction and pass this information on to the ALU. However, when an instruction other

than an R-type instruction is executing, the control unit must specify the correct ALU control signal on the ALU op bus.

(1) Describe the ALU control logic in Verilog using the following module interface:
module ALUControl(ALUCtrl, ALUop, FuncCode);
Helpful Hints:
• Case statements can be easily nested within if else statements within always@(*) blocks.
• Use the following define statements (in addition to the ones provided above) for code readability



```
`define SLLFunc  6'b000000
`define SRLFunc  6'b000010
`define SRAFunc  6'b000011
`define ADDFunc  6'b100000
`define ADDUFunc 6'b100001
`define SUBFunc  6'b100010
`define SUBUFunc 6'b100011
`define ANDFunc  6'b100100
`define ORFunc   6'b100101
`define XORFunc  6'b100110
`define NORFunc  6'b100111
`define SLTFunc  6'b101010
`define SLTUFunc 6'b101011
```

(2) Simulate the operation of your hardware using ISE and the ALU control test bench provided on the laboratory website. Paste the output of the simulator into your lab write-up.

(3) Synthesize your design using the same Device properties as in Section 2. Fix all warnings and errors and ensure the compiled hardware contains no latches or flip-flops. Provide the summary results of the synthesis process in your lab write-up.