CS281, Computer Systems

Lab 9: Y86 ALU

1. The purpose of this laboratory is to:

   - move from a programmer's perspective to a hardware designer's perspective.
   - develop more expertise with Logisim.

2. This lab is to be completed with a partner. The lab is due by Monday, November 4, 9:30am. You will only submit a copy of your logisim file; the accompanying lab report is not required but may help document your work if necessary when seeking partial credit.

3. The following ALU description specifies an Arithmetic and Logic Unit that can serve the needs of our hardware realization of the Y86 CPU datapath. It initially supports six operations (addl, subl, andl, xorl, two shifts) in a combinational circuit that calculates a 8-bit output based on two 8-bit inputs and a 4-bit input (only 3 bits currently significant) specifying the ALU operation to perform. The ALU also computes a Cout bit and three flag bits, ZF, SF, and OF.

Func	**ALU Ctrl**	**Semantics**
addl	0000	F = B + A + Cin; Cout, ZF, SF, OF update
subl	0001	F = B + A + Cin; Cout, ZF, SF, OF update
andl	0010	F = A & B (bitwise AND); ZF, SF update, OF=0
xorl	0011	F = A ^ B (bitwise XOR); ZF, SF update, OF=0
asrl	0100	F = A >> 1 (arithmetic); ZF, SF update, OF=0, Cout= A_0
lsll	0101	F = A << 1 (zero fill); ZF, SF update, OF=0, Cout = A_7

Table 1: Y86 Operations

4. Table 1 details the specification of the Y86 functional operations. Op is a four bit wide input, so in Table 1 we give a mneumonic for the operation, the bit pattern for Op, and then the computation specifying the F output in terms of A, B, and Cin. Since we all know C/C++ bitwise operations now, we will borrow that notation. Note that the addl and subl also add in the value of Cin. Cin is also used as the bit to "fill in" for the shift right (as the

most significant bit) and shift left (as the least significant bit). The reason for this design is to allow this ALU8 to be chained together with three more ALU8 chips to build a 32 bit ALU. Note also that we do not need explicit subtraction, because we can obtain that operation by using the subl operation and setting Cin to 1, effectively getting the "complement and adding one" semantics we need for two's complement.

5. Table 2 gives the meanings for the three condition code flag bits.

Flag	Width	Description
ZF	1	Zero Flag: If the result of the current ALU operation is zero, then Z is asserted. If the result of the operation is not zero, then Z is 0.
SF	1	Sign Flag: Reflects the most significant bit of the result of the current ALU operation. When interpreted as 8-bit two's complement, the sign bit indicates a negative result.
OF	1	Overflow Flag: For arithmetic operations (addl and subl), this bit is asserted if the current operation caused a two's complement overflow–either positive or negative, and is deasserted (0) otherwise. For logical and shift operations (andl, xorl, asrl, lsll), this bit is always deasserted.
Cout	1	Carry Flag: This bit is 1 if addition or subtraction caused a carry-out from the most significant bit position, so an unsigned overflow. For a right shift, this is the bit "shifted off" from operand A, aka A_0. For a left shift, this is the bit "shifted off" from operand A, aka A_7. This bit is always 0 for the and and xor operations.

Table 2: Y86 Condition Codes

6. Using the **Logisim** digital logic design and simulator package, design and implement the above-described Y86 ALU. You must package the ALU so that it conforms to the interface given in Figure 1. In particular, the 8-bit A and B inputs must come in from the left hand side, the 8-bit output, F, must come out the right hand side, the condition codes must be in the correct order across the top as single bit input pins, and the ALU function must come in as a single 4-bit input at the bottom. Furthermore, your file should be named ALU.circ and the name of the circuit itself should be ALU (not main). You may define any other subordinate circuits that you wish to keep your design clean and organized.

7. This lab will be graded based on a total of 50 points. 40 of the points will

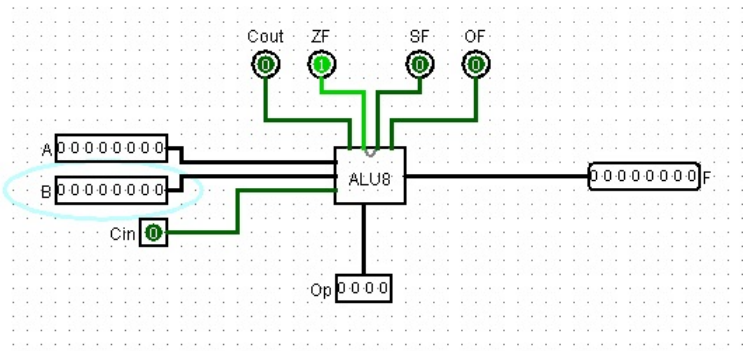

Figure 1: ALU External Specification

derive from correct operation of the ALU for all of my test cases. I will be grading for both the output F as well as *all four condition codes* over a variety of test inputs, so make sure these are implemented correctly per the specification given above. The final 10 points will be based on your circuit design. Some of the grading criteria for the 10 points include these design elements:

(a) Following conventions of inputs on the left, outputs on the right, and being able to read the "flow" of the circuit from left to right.

(b) Abstraction — appropriate use of sub circuits with appropriate pin interfaces and labels so that a problem's decomposition is reflected in the design and hierarchical use of circuits.

(c) Neatness — appropriate "clean" routing of wires and placement of sub circuits to convey an organized and orderly design.

8. If you're pressed for time: To allow prioritization of the elements of the design and implementation, we give two levels of deliverables corresponding roughly to B-level work versus A-level work. Both levels have in common the 10 points for circuit design criteria.

   (a) For the B-level work, 32 of the 40 correctness points will assess the operation of add, sub, and, xor for the result F as well as the condition codes appropriate to these operations.

   (b) For the A-level work, the remaining 8 correctness points will assess the operation of asr and lsl for the result F, and the condition codes appropriate to these operations.

9. **Hints**

(a) Begin with a basic design that uses the built-in adder and subtractor circuits within Logisim. The key is to get up and running with inputs generating outputs, and then work on refining your design. Your final design should include only and, or, not, xor, and multiplexing for this step of ALU design. You are further allowed to use bit extensions to increase the size of Cin, etc, where necessary. You will need to use splitters (perhaps in subcircuits) to get from the 8-bit and 4-bit inputs and/or values in your circuits to get at individual wires giving you boolean 0/1 values.

(b) The Logisim circuit gates can be configured to have a large number of inputs, up to 32. Each of those inputs can have more than one bit (for instance, eight). Thinking beyond your "2 single bit input" past experience may reduce your design time.

(c) For the basic inputs-to-output design, consider the following: In sequential programming in C/C++, we would think about implementing ALUfun as a chained conditional or a switch which, depending on the value of ALUfun, would perform just one operation. In hardware, we tend to think more parallel – go ahead and perform all six operations generating six outputs and then use ALUfun to "select" just one that gets routed to the output.

(d) When working on the condition codes, take the time to enumerate the input category combinations that result in different values of the condition codes (particularly the OF bit). Then make sure you test these combinations to make sure your circuit works for *all* of them. Most students lose points here because they test one or two cases and are then satisfied.