cs281: Introduction to Computer Systems

Lab9

1 Arithmetic Logic Unit for the Y86 CPU

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 four operations (addl, subl, andl, xorl) in a combinational circuit that calculates a 32-bit output based on two 32-bit inputs and a 4-bit input (only 2 bits currently significant) specifying the ALU operation to perform. The ALU also computes three flag bits, ZF, SF, and OF.

Func	**ALU Ctrl**	**Semantics**
addl	0000	F = B + A; ZF,SF,OF update
subl	0001	F = B - A; 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

Table 1: Y86 Operations

Table 1 details the specification of the Y86 functional operations. 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 32-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 otherwise. For logical operations (andl and xorl), this bit is always deasserted.

Table 2: Y86 Condition Codes

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 32-bit A and B inputs must come in from the left hand side, the 32-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.

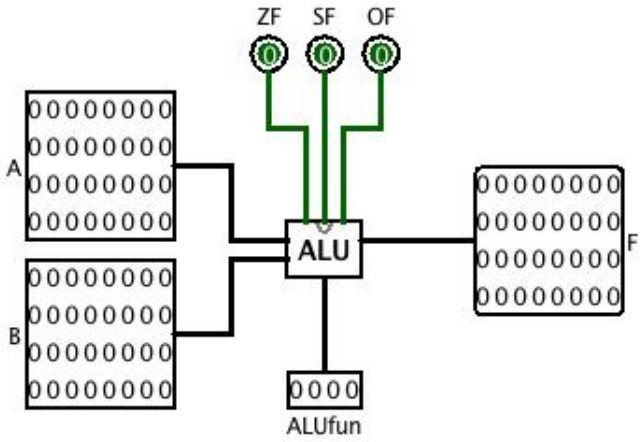

Figure 1: ALU External Specification

1.1 Evaluation

This lab will be graded based on a total of 20 points. 15 of the points will 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 three condition codes over a variety of test inputs, so make sure these are implemented correctly per the specification given above. The final 5 points will be based on your implementation. Two of these points are granted for building your own ripple-carry adder using the basic logic gates and the design from earlier in the semester. One point is granted for using just a single unit for both addition *and* subtraction. The last two implementation points are for clean and clear design and organization.

1.2 Hints

1. 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.

2. 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 4 operations generating four outputs and then use ALUfun to "select" just one that gets routed to the output.

3. You will need to use splitters (perhaps in subcircuits) to get from the 32-bit and 4-bit inputs and/or values in your circuits to get at individual wires giving you boolean 0/1 values.

4. The Logisim circuit gates can be configured to have a large number of inputs, up to 32. So think beyond your "2 single bit input" past experience.

5. 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.