COMP 303 MIPS Processor Design
Project 2: 32-bit ALU Implementation

Due Date: 23:59 October 30, 2009 (Late submissions will not be accepted)

Overview:
In the next three projects for COMP 303, you will design and implement a subset of the MIPS32 architecture in Logisim, a software logic simulator. The goal of these projects is to move you from designing small special-purpose circuits, such as those you designed as homework exercises, to building complex, general-purpose CPUs. We will be implementing a basic 32-bit MIPS processor. We will ignore more advanced features such as the MIPS coprocessor instructions and traps/exceptions. Read this document entirely, paying special attention to the section "Deviation from the MIPS standard".

We will be using Logisim, a free hardware design and circuit simulation tool. We have implemented a library of components to help you, and Logisim comes with libraries containing basic gates, memory chips, multiplexors and decoders, and other simple components. You may use any of these basic components in your designs, but ***you may not use the Logisim arithmetic library*** anywhere in your design. If you wish to download additional libraries from the web to use in your design, please check with the course staff for approval.

***Academic Integrity*. As one of the most widely studied architectures, MIPS has a wealth of information available on the web and in textbooks. You may consult any of the MIPS architecture documentation available to you in order to learn about the instruction set, what each instruction does, etc. But we expect your design to be entirely your own. If you are unsure if it is okay to borrow from some other source, just ask the TAs, and give credit in your final writeup. If you are unsure about asking the TAs, then it is probably not okay. Plagiarism in any form will not be tolerated.**

Project 2: 32-bit ALU Implementation
The MIPS ALU (arithmetic and logic unit) performs all of the core computations dictated by the assembly language.

Begin by implementing the following circuits (numbers in brackets give the number of bits in each input/output).

Full Adder: C = A + B + cin
Input: A[32], B[32], cin
Ouput: C[32], cout
The output *C* is computed by adding *A*, *B*, and *cin*. Any remaining carry bit is output on *cout*. *Hint:* Use sub-components to make wiring easier, by building a 1-bit adder, then 4-bit adder, and so on up to 32-bits.

Left Shifter: C = (B << sa) | $(cin)^{sa}$

Input: B[32], sa[5], cin

Ouput: C[32]

The output *C* is computed by shifting *B* to the left *sa* bits, filling the bits on the right with *sa* copies of *cin*. The shift amount *sa* can be anything from 0 to 31, encoded as an unsigned integer. One efficient way to implement such a shifter is to perform the shift as a series of stages: the first stage shifts either 0 or 16 bits, the second stage either 0 or 8 bits, the third stage either 0 or 4 bits, and so on. By enabling different combinations of stages the circuit can shift any desired amount. *Hint*: Note that shifting a value on a 32-bit bus, by a *constant* amount, either left or right, is simply a matter of adding and removing and renaming the wires on the bus.

ALU: C= f(A, B, sa)

Input: A[32], B[32], op[4], sa[5]

Output: C[32]

The function is selected according the value of the *op* input:

op	*C*	*Function name*
0000	C = B >> sa	shift right logical
0001	C = B >>> sa	shift right arithmetic
001x	C = B << sa	shift left logical
010x	C = A & B	and
011x	C = A | B	or
100x	C = ~(A | B)	nor
101x	C = A ^ B	xor
110x	C = A + B	add
111x	C = A - B	subtract

Table 1

Note the difference between logical right shift (which fills with zero bits), and arithmetic right shift (which fills the empty bits with the copies of the sign bit of B). The right shift operations can be implemented using a dedicated right-shift component, or by re-using other circuits (as was done for add/subtract in class).

Getting started with Logisim:

Logisim is available from the course web site. You can get familiar with Logisim by making some simple circuits and testing them out. There will be a Problem Session on how to use Logisim, but the interface is quite straightforward – you should be able to get going with just the binary. You can also check out the official Logisim pages for help if necessary.

What to submit:

Submit a single zip file containing your ALU. Please name your main circuit as project2.circ. Please provide the necessary library loadings to the top hierarchy file, i.e. project2.circ, so that your project2.circ file can be opened without asking to locate other .circ files.

Your ALU should be oriented such that there are two 32-bit buses on the west side, a 5-bit SA bus on the north side, a 4-bit opcode bus on the south side, and a 32-bit output bus on the east side. We will test your ALU by inserting it into a test circuit, therefore it is crucial for you to get the pins oriented correctly.

For the Adventurous:

Note: These suggestions for an extra challenge will be examined (and commented on, if your project works well) but not graded. They will have no impact on the class grades. They are here to provide some direction to those who finish their assignments early and are looking for a way to impress friends and family.

- Implement a carry-lookahead adder
- Minimize the number of gates used by the ALU

Warnings, Help and Hints:

Ask the TAs for help. We expect to see most students in office hours during the course of the project. Extra hours will be scheduled as needed.

Do a pencil and paper design for all components before implementing in Logisim.

Use buses in Logisim wherever possible, instead of drawing 32 parallel wires.

Data should flow from left to right as much as possible.

Keep in mind that most Logisim components can be adjusted to fit your needs. You can get a MUX of the right size by adjusting the number of inputs and outputs.

Keep in mind that most Logisim components can be adjusted to fit your needs.

You may not use the pre-built arithmetic components (e.g. adders) in Logisim for this assignment; you'll need to develop them from basic gates.

Any deviations from Table 1 result in grade degradation.

Check your circuit, using Logisim simulation and be sure that all instructions work. A failure of an ALU function means nothing whether you have very small errors or big errors. Therefore please correct small errors to have all functions working well.

Use one adder, designed by you, for both addition and subtraction.

Is optimal always best?

It is not enough to simply build a working circuit. We want you to build a *good* working circuit. But we leave it up to you to define what *good* is, and we expect you to document your criteria and how you fulfill them. For example, you might aim to make the fastest processor possible, at the expense of simplicity and number of gates. Or you might aim for a minimalist design using the fewest gates possible, even if it makes your processor slower. Eventually, in Project 3, you will be asked to document your goals and justify the choices you made. All of your designs should be clear and easy to follow, and should be annotated (with text labels) for any unusual or difficult parts of the circuit.

Deviation from the MIPS Standard:

You can ignore any MIPS instruction or feature not mentioned in this document.

In addition:

- Ignore overflow conditions
- Assume traps or exceptions will not occur (i.e., do not implement them).
- Do not implement floating point, multiply/divide, or any other arithmetic used by the MIPS processor.