CS 2321 Laboratory Problem Set 6

Due April 5, 2006

April 7, 2006

These problems are to be done on an individual basis following the Trinity University Academic Integrity Policy or Trinity University Honor Code.

Academic Integrity and Honor Code

All students are covered by a policy that prohibits dishonesty in academic work. The Academic Integrity Policy (AIP) covers all students who entered Trinity before the Fall of 2004. The Academic Honor Code covers all those who entered the Fall of 2004 or later. The Integrity Policy and the Code share many features: each asserts that the academic community is based on honesty and trust; each contains the same violations; each provides for a procedure to determine if a violation has occurred and what the punishment will be; each provides for an appeal process. The main difference is that the faculty implements the AIP while the Honor Code is implemented by the Academic Honor Council. Under the Academic Integrity Policy, the faculty member determines whether a violation has occurred as well as the punishment for the violation (if any) within certain guidelines. Under the Honor Code, a faculty member will (or a student may) report an alleged violation to the Academic Honor Council. It is the task of the Council to investigate, adjudicate, and assign a punishment within certain guidelines if a violation has been verified. Students who are under the Honor Code are required to pledge all written work that is submitted for a grade: On my honor, I have neither given nor received any unauthorized assistance on this work and their signature. The pledge may be abbreviated pledged with a signature.

Laboratory problems should be submitted electronically (e-mail to cs2321@ariel.cs.trinity.edu) on or before the due date and should contain a problem write-up, source code to any programs and data sets used in solving the problem. The submitted files should be ASCII text files having Unix end-of-line characters (please convert all Windows and Mac text files to Unix format–I have found that Emacs seems to do a reasonable job of such conversions). If several files need to be submitted, put them in a directory having name *your-last-name-problem-set-number* and create a tar archive of this file system and attach it to your e-mail problem submission.

Figure 1 shows a circuit diagram of a 1-bit ALU which performs *AND, OR, addition, subtraction*, a *AND NOT* b, a *OR NOT* b, and a *LESS* b.

Build a model of this circuit using the J based circuit modeling language we have developed in class.

The inputs of the One Bit ALU are *a, b, less, binvert, carryin*, and *operation*. The inputs *a, b, less, binvert, carryin* are one bit values, while the input *operation* ranges in value from 0 to 3 and thus requires at least 2 bits. The outputs of the One Bit Alu are *result* and *carryout*, each one bit in size.

Build a second model of a one bit ALU with overflow detection using the circuit diagram shown in Figure 2. Note that this circuit is very similar to Figure 1 with the exception that there is no *carryout*, but two new outputs have been introduced, *set* and *overflow*. Do not worry about how the overflow detection circuit computes *overflow*; just set that output to zero (no overflow) for now.

The vertical symbol (having input of operation, producing the output result) is called a multiplexer. This circuit uses the operation input as a multi-bit input and uses that binary number to select which, of several, inputs should be used for the output. A multiplexer can be modeled (non circuit based) by:

```
NB. a multiplexer

mux =: monad define
```

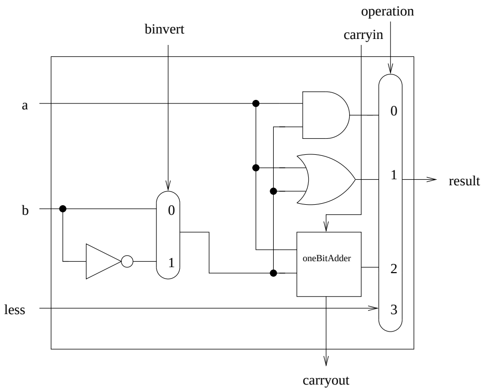

Figure 1: Circuit Diagram for a One Bit ALU

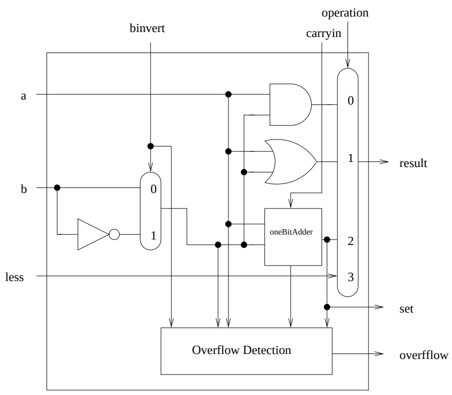

Figure 2: Circuit Diagram for a One Bit ALU with Overflow Detection

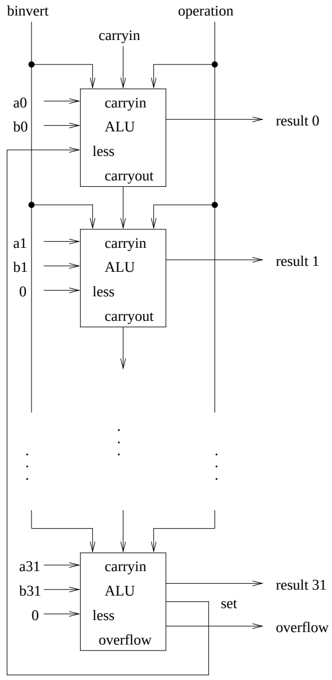

Figure 3: Circuit Diagram for a 32 bit ALU

```
'selector inputs' =. y.
selector from inputs
)
```

This model assumes the `selector` has been converted from binary to decimal, however, you may modify the multiplexer to do that conversion internally if you wish. Both of your oneBitALU models should use a multiplexer to compute the output *result*.

Use these two oneBitALU models to build a model of a 32 bit ALU. The least 31 bits of the 32 bit ALU should use the circuit from Figure 1 and the most significant bit should use the circuit shown in Figure 2. A circuit diagram for the 32 bit ALU is shown in Figure 3

Notice that this model will have in excess of 64 inputs so it will be advantageous to allow the a and b inputs to be decimal integers which are converted to 32-bit binary values inside the ALU model. The 32-bit result should be returned in decimal format as well. All other inputs and outputs should be handled separately in binary form as in the other modeling we have done.

Note that the *set on less* operation uses a binary input of 1 1 1, so this means that the operation multiplexer of the 1-bit ALU requires an 8-way select. Other values for operation are given in Table 1.

Operation	Function
0 0 0	and
0 0 1	or
0 1 0	add
1 1 0	subtract
1 1 1	set on less

Table 1: ALU Operation Control Lines

Our present multiplexer will accomplish this, but this is not based on a circuit. Try to develop a circuit for this multiplexer and build a circuit based model for it.

You should build a suite of test values for each of the one bit ALU's and a test suite for the 32 bit ALU and demonstrate the correct operation of your circuit models.

Finally, try to build a circuit for overflow detection, and count the number of transistors, diodes and resistors in your 32-bit ALU.

Table 2 gives the transistor count for each of the circuit types.

Circuit	Resistors	Diodes	Transistors
AND	6	4	6
OR	7	4	8
NOT	4	1	4

Table 2: Transistor Count for Various Logic Circuits

Problem Set 6 Solution [HTML] [PS] [PDF]