CprE 381, Computer Organization and Assembly-Level Programming, Fall 2013

Lab Report for Project A

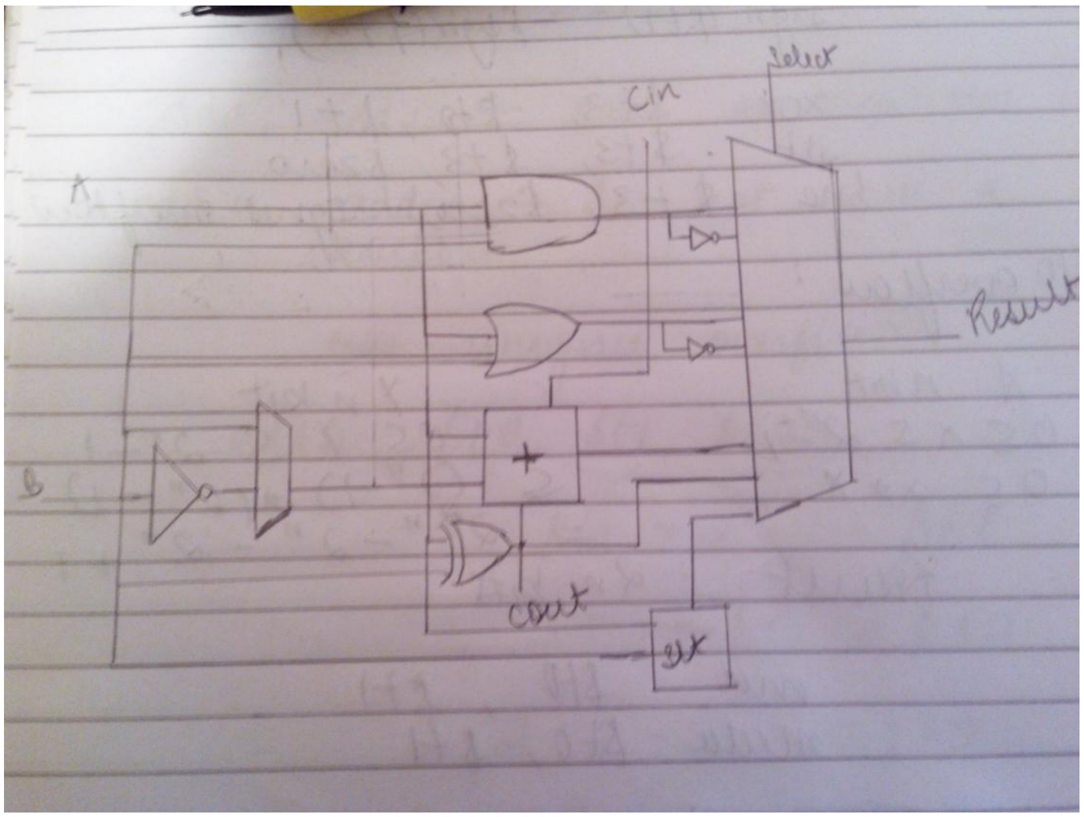
Lab Partners	Deeksha Juneja Asim Verma
Section/Lab Time	Friday 2pm to 4pm
Total Score	________ /20

Refer to the highlighted language in the Project A instructions for the context of the following questions.

a. [Prelab] With your project group members, create a list of best practices / tips for designing, compiling, and testing VHDL modules based on your experiences so far with these labs (for Prelab of the evaluation form.)

- It should be well coded, commented and documented
- use generics to write parameterized models of varying structure and behavior
- You can implement many digital systems, such as memories, as regular iterative compositions of subsystems.
- you should be able to describe the subsystem once and then describe how it is to be repeatedly instantiated, rather than describe each instantiation individually
- Best method to test is using a testbench
- Having a design to start with helps in writing the code

b. [Part 1(a)] Draw a schematic for a 1-bit ALU that supports the following operations: add/sub (both signed and unsigned), slt, and, or, xor, nand, and nor. What are the inputs and outputs that are needed? (2 points)



Inputs
Two inputs A and B are required
A carry in required for adder which is always set to 0
Add_sub is required to determine whether to add or subtract
Selection is required to determine which operation to perform
Output
Carryout is required
Result is required
Signal array for all the outputs is required
Few intermediate signals are required for processing.

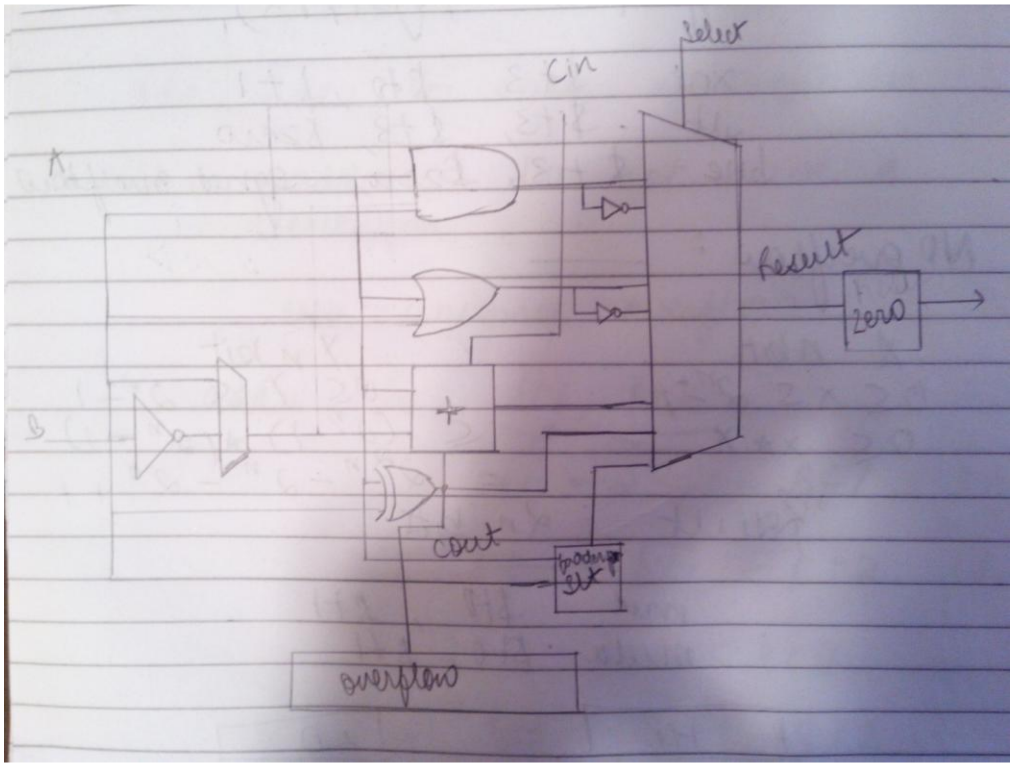
c. [Part 1(b)] In your project writeup, describe your design in terms of the VHDL coding style you chose and the control signals that are required. (2 points)

We used structural VHDL coding style. We made slt, fadder, or, xor, and, inv dataflows. Then we used the components of those in the structural VHDL of alu. For subtraction a control bit was used to determine whether input B will go inside the fadder or 2's complement of B will go. For Nand and nor, and and or gates with inv were used respectively. A sel input was used to select which operation will be performed. An array of signal s1 was used to store the outputs of different operations. S1 was mapped with the input of the mux. The output of mux was mapped with the Result.

d. [Part 1(c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup. (2 points)

To get the waveform, we set the inputs A and B. We then set the carry in to 0. If we want to add or subtract then we set the add_sub bit, otherwise not. In the sel bit, we provide the number of the operation we want to perform. The result for that operation comes in the Result output variable.

e. [Part 2(a)] Draw a simplified schematic for this 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is slt implemented? (2 points)

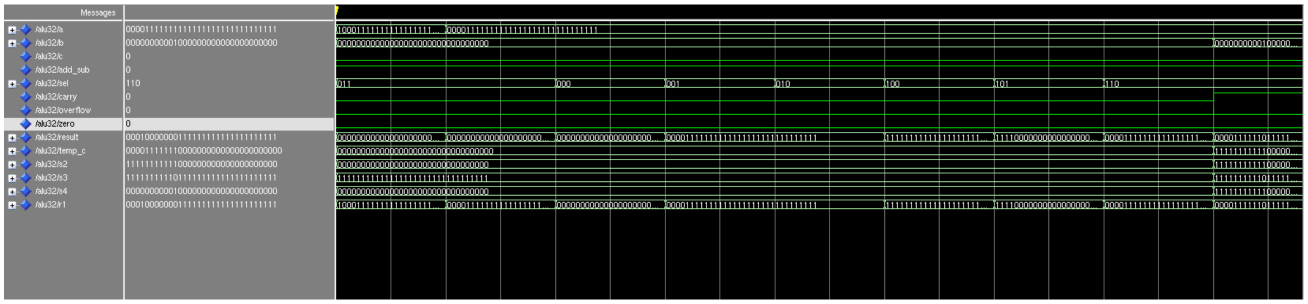


To calculate the overflow, we use the carry in bit. We made a temp_c signal which is std_logic_vector(32 downto 0). In its 0th position we but the carry in and its 32nd position was put in the carryout. It is mapped to the carryout of alu (1bit). To check for overflow, we XOR the 31st and 32nd bit temp_c. Note that overflow only occurs when both the inputs have the same sign. If the sum of two numbers with the sign bits off yields a result number with the sign bit on, the "overflow" flag is turned on and vice versa.
To calculate zero, we put the result in with select. If the result is zero then zero bit is turned on else its turned off. To implement slt, we used fadder. We set add_sub to 1 to perform subtraction. We check if our result is less than 0 (r1<0). If it is then slt is set to "00000000000000000000000000000001" otherwise it is set to "00000000000000000000000000000000".

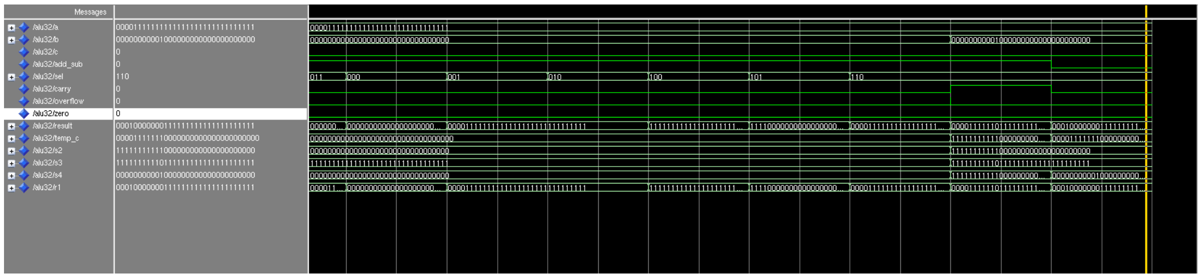
f. [Part 2(b)] In your writeup, describe what challenges (if any) you faced in implementing this module. (2 points)

While implementing 32 bit ALU one major challenge was the implementation of slt function. In one bit we easily did it by creating a separate slt module, however to implement that in 32bit was not easy. So we finally made use of our adder and used the concept of A-B<0 to implement it.

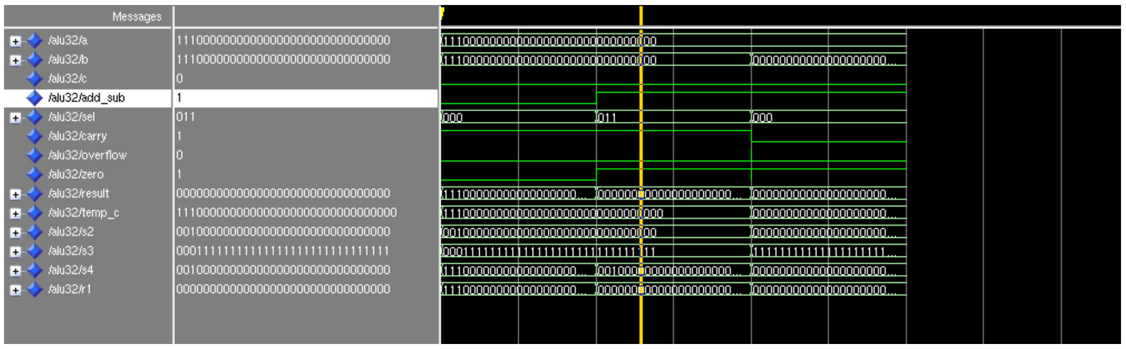
g. [Part 2(c)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup. (2 points)



```
Messages
/alu32/a          00001111111111111111111111111111
/alu32/b          00000000001000000000000000000000
/alu32/c          0
/alu32/add_sub    0
/alu32/sel        110
/alu32/carry      0
/alu32/overflow   0
/alu32/zero       0
/alu32/result     00010000000111111111111111111111
/alu32/temp_c     00001111111000000000000000000000
/alu32/s2         11111111111000000000000000000000
/alu32/s3         11111111110111111111111111111111
/alu32/s4         00000000001000000000000000000000
/alu32/r1         00010000000111111111111111111111
```



```
Messages
/alu32/a          00001111111111111111111111111111
/alu32/b          00000000001000000000000000000000
/alu32/c          0
/alu32/add_sub    0
/alu32/sel        110
/alu32/carry      0
/alu32/overflow   0
/alu32/zero       0
/alu32/result     00010000000111111111111111111111
/alu32/temp_c     00001111111000000000000000000000
/alu32/s2         11111111111000000000000000000000
/alu32/s3         11111111110111111111111111111111
/alu32/s4         00000000001000000000000000000000
/alu32/r1         00010000000111111111111111111111
```



```
Messages
/alu32/a          11100000000000000000000000000000
/alu32/b          11100000000000000000000000000000
/alu32/c          0
/alu32/add_sub    1
/alu32/sel        011
/alu32/carry      1
/alu32/overflow   0
/alu32/zero       1
/alu32/result     00000000000000000000000000000000
/alu32/temp_c     111000000000000000000000000000000
/alu32/s2         00100000000000000000000000000000
/alu32/s3         00011111111111111111111111111111
/alu32/s4         00100000000000000000000000000000
/alu32/r1         00000000000000000000000000000000
```

To get the waveform, we set the inputs A and B. We then set the carry in to 0. If we want to subtract or do slt then we set the add_sub bit to 1, otherwise 0. In the sel bit, we provide the number of the operation we want to perform. Overflow tells if overflow has occurred or not. The result for that operation comes in the Result output variable. The result is then used to check for zero. If the result is 0, the zero is set to 1.

h. [Part 3(a)] Describe the difference between logical (srl) and arithmetic (sra) shifts. Why does MIPS not have a sla instruction? (2 points)

Shift Right Logical shifts zeros in from the left.

Shift Right Arithmetic replicates the sign bit on the left. The new MSB after the SRA operation will be the same as the old MSB. It can be used for division by 2^shiftbit.

MIPS does not have sla because it is not required as there is no sign bit on the left side. It is same as logical left.

i. [Part 3(b)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations? (2 points)

   Our code has a control bit 'logical' which decides whether there arithmetic operation will be performed or the logical. When set to 0 arithmetic operation is performed and when set to 1, logical operation is performed. Shift bit is the input which decides by how many bits operation will be performed.
   Eg. When logical input is set for 1 and left is set to 0, it performs shift right logical (srl).
   Also the entire process of shifting is divided into stages i.e 1-bit shift, 2 bit shift, 4 bit shift, 8 bit shift, 16 bit shift. This entire shifting operation is done with the help of mux.

j. [Part 3(c)] In your writeup, explain how the right barrel shifter from part b) can be enhanced to also support left shifting operations? (2 points)

   To perform left shifting operation we add another control bit called left which decides whether right shift or left shift operation will be performed. When set to 0 right bit shifting is done and when set to 1 left bit shifting is done. Also since there is no left arithmetic shift in mips so even if left is set to 1 and logical is set to 0 our code will perform right arithmetic shifting instead of left.

k. [Part 3(d)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup? (2 points)

   There are 3 different waveforms.
   1. Left = 0, logical = 0. This performs arithmetic right shift.
   2. Left = 0, logical = 1. This performs logical right shift.
   3. Left = 1, logical = 1. This performs logical left shift.
   4. Left = 1, logical = 0. This performs arithmetic right shift as well because arithmetic left shift is not required.