CSCI 2500 — Computer Organization
Homework 05 (document version 1.0) — Due 9 April 2021
MIPS Circuits in C

- This homework is due by the Midnight EST on the above date via a Submitty gradeable.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- Homeworks are available approximately two weeks before they're due. Plan to start each homework early. You can ask questions during office hours, in the Submitty forum, and during your lab session.

1 Implementing an ALU, Multiplier, and Shifter

For this assignment, you'll be building upon the 1-bit ALU circuit you developed for Lab 5. Specifically, you'll be using your logic gates to implement circuits for a 32-bit ALU, a 16-bit multiplier, and a 32-bit left shifter. Your circuit will be supporting 8 different MIPS instructions given in the table below.

Instruction	Input symbol	OP bits	Operation
and	&	000	Logical AND
or	|	001	Logical OR
nor	n	010	Logical NOR
add	+	011	Integer addition
sub	-	100	Integer subtraction
slt	<	101	Set if less than
sll	s	110	Shift left logical
mul	*	111	Integer multiplication

The OP bits are an array of three BITS that will be sent to your control circuit. These bits will determine which circuit output gets sent back to main. See the `hw05.c` template file for function prototypes and more detail into the design and implementation specifications.

1.1 Data Format

The input/output as well as intermediate format for storing integers will be an extension of the `BIT` type we used for Lab 5. 32-bit 2's complement numbers will be stored as binary in an array of 32 `BIT`s. For these arrays, the most significant bit is at index 31 and the least significant bit is at index 0. The first function you'll need to implement is probably `convert_to_binary`, to convert the input decimal integers to their BIT representations.

1.2 32-bit ALU

See slides 30 and 31 of Chapter-3d-ALU.pdf for logical diagrams of the 1-bit ALU and 32-bit ripple ALU that you'll be implementing. The 1-bit ALU will extend the Lab 5 ALU to also include functionality for set-less-than (slt). You can use a for loop here to simplify the 32 'ripples'. You'll also need to consider logical NOR. You can implement it into your ALU with an additional control bit to flip the OR output, or you can implement the equivalent within your `Control` function.

1.3 16-bit Multiplier

See slides 49 and 50 for logic diagrams for your multiplier. You should use your 32-bit ALU circuit as well as your shifter circuit for this implementation. Note that since we're just implementing a 32-bit ALU, we can effectively only multiply two 16-bit numbers without overflow. You are allowed to use a for loop to simulate the 'Control' of your circuit.

1.4 32-bit Left Shifter

You'll implement two circuits related to shifting. First, the `shifter` function will shift the input by 1-bit either left or right, depending on the control bit. One possible design is given here: `http://www.mathcs.emory.edu/~jallen/Courses/355/Syllabus/1-circuits/shifter.html`. This circuit will be useful for your multiplier. The second function is `left_shifter32`, which will shift input A to the left by the amount specified in input B. There's multiple ways to implement such a circuit. I give you pseudocode one possible approach in the template, which doesn't require a considerable number of gates.

1.5 ALU Control

The control circuit will use the OP bits to determine which operation to perform, or from which operation to return results to main. This circuit can also be used to determine which control bits to set for your ALU. See the template file for details on my approach.

1.6 Assignment Rules

We don't usually have much in the way of rules for our assignments, but this one is a special case. The primary purpose of this assignment is to demonstrate how basic logic gates can be combined together into complex circuits. As a rule, whatever you implement in your solution file should be a loose representation of an implementable circuit. The below are guidelines to this end:

1. No modifications to main().
2. You are allowed to implement additional basic gates – e.g., a three input AND gate – to simplify your other circuits.
3. If desired, you can modify the function prototypes given in the template. For example, you can pass an additional control bit to your ALU for the sake of outputting NOR instead of OR.

4. No if() or if()-else() logical control statements, or equivalent.
5. You can use basic for loops only (for int i = 0; i < N; ++i). Different loop structures are not allowed, as they can easily be used as a logical equivalent to if-else statements.
6. No calls to any external or library functions.
7. The above rules are to basically ensure that you're sticking to using gates for all logical control and operations. No tricky workarounds are allowed.

2 Input and output

Below are examples of inputs and expected outputs for the assignment. Generally, you should expect positive or negative numbers for all inputs *except* for the shift amount for shift left logical (note: if you implement the circuits as given in the Chapter-3d pdf, you should have to do **no** additional work to account for negative inputs).

```
bash$ ./hw05
12 & 5
00000000000000000000000000001100
&
00000000000000000000000000000101
=
00000000000000000000000000000100

bash$ ./hw05
12 | 5
00000000000000000000000000001100
|
00000000000000000000000000000101
=
00000000000000000000000000001101

bash$ ./hw05
12 n 5
00000000000000000000000000001100
n
00000000000000000000000000000101
=
11111111111111111111111111110010

bash$ ./hw05
12 + 5
00000000000000000000000000001100
+
00000000000000000000000000000101
=
00000000000000000000000000010001

bash$ ./hw05
12 + -5
00000000000000000000000000001100
+
11111111111111111111111111111011
=
00000000000000000000000000000111

bash$ ./hw05
12 - 5
00000000000000000000000000001100
-
00000000000000000000000000000101
=
00000000000000000000000000000111

bash$ ./hw05
12 < 5
00000000000000000000000000001100
<
00000000000000000000000000000101
=
00000000000000000000000000000000

bash$ ./hw05
-12 < 5
11111111111111111111111111110100
<
00000000000000000000000000000101
=
00000000000000000000000000000001

bash$ ./hw05
12 s 5
00000000000000000000000000001100
s
00000000000000000000000000000101
=
00000000000000000000000110000000

bash$ ./hw05
12 * 5
00000000000000000000000000001100
*
00000000000000000000000000000101
=
00000000000000000000000000111100

bash$ ./hw05
12 * -5
00000000000000000000000000001100
*
11111111111111111111111111111011
=
11111111111111111111111111000100
```

3 Submission and Grading Criteria

For this assignment, you will submit your code to the Submitty gradeable. A code template `hw05.c` is provided for your convenience. Do your best to stick to the above rules, as the TA grade is mostly based on your adherence to them. The below will be the grading criteria for the assignment.

1. Autograding: 60%
   - Standard visible and hidden test cases
2. TA grading: 40%
   - Solution is representative of an implementable circuit
   - Solution otherwise adheres to the rules given above