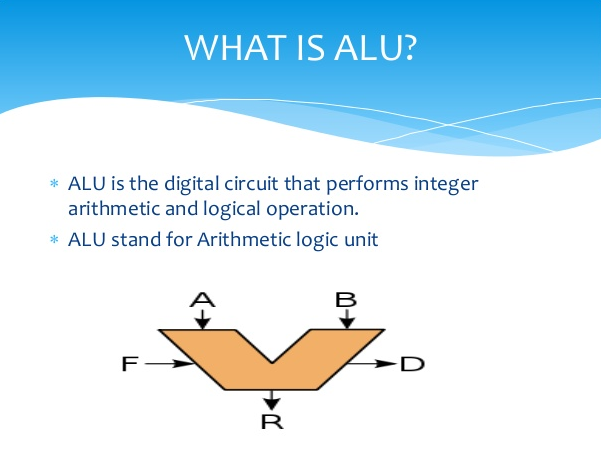
# Lab #9: Microprocessor

In this week you need to implement a microprocessor with a register file and an ALU.

Just to refresh the ALU, the following is our ALU specifications. It had been used for the lab, in the quiz and in the class.

The ALU (as given below) takes two 8-bit numbers as inputs known as operands (A and B). It also provides one output (R, 8 bits) of the ALU which is derived after performing the computations as desired. A function code (F, 4-bit wide to represent 16 operations) is also provided as an input to the ALU to indicate the operation to be performed on the operands.



D is the collection of two output status bits D1D0. D0 bit is set to 1 if R is all zeros. It remains 0 otherwise. D1 bit is set as given in the table below (it roughly represents the Carry out)

The ALU operations are indicated by the following table with reference to the function code.

|  |  |  |
| --- | --- | --- |
| Function Code (F) | Operation to perform | D1 |
| 4’b0000 |  | 0 |
| 4’b0001 | (only 2’s complement representation) | if is (i.e. n’b10…00), cannot be represented (since the max positive value representable is ). Hence the output shall not be correct. D1 is set to 1 to indicate overflow. |
| 4’b0010 |  | D1 is Carry out of the summation |
| 4’b0011 |  | D1 is 1 if cannot be represented in bits (i.e. there is a borrow into the most significant bit) |
| 4’b0100 | (*aka* arithmetic shift to left by one bit) | D1 is set to 1 if cannot be represented in bit. |
| 4’b0101 | (*aka* arithmetic right shift) | 0 |
| 4’b0110 |  | D1 is Carry out of the summation |
| 4’b0111 |  | D1 is 1 if the cannot be represented in bits. |
| 4’b1000 |  | 0 |
| 4’b1001 |  | 0 |
| 4’b1010 |  | 0 |
| 4’b1011 |  | 0 |
| 4’b1100 |  | 0 |
| 4’b1101 |  | 0 |
| 4’b1110 |  | 0 (with output R on the 7-segment LED display as two HEX digits |
| 4’b1111 | (i.e. all 1’s) | 0 |

You will also need a register file as follows. The Register file is 8 bit wide (n = 8) whereas it has 16 registers called R1 to R16. *k* is therefore 4 bit and fields RS1, RS2 ad WR are *k* bit wide to be able to address all 16 registers.

Register File

D

Q1

Q2

WR

WE

RS1

RS2

*n*

*n*

*n*

*k*

*k*

*k*

In order to read a register (one of 16), the address (i.e. the register number) is to be given on RS1 or on RS2 and the corresponding register is read on Q1 or on Q2 respectively. In order to write a value in a specific register in the register file, the data is presented on D lines, register number to which the data is destined to be written is presented on WR lines and the signal WE is set to 1 which makes the corresponding register latches transparent. It must be ensured that the data is not changed while WE is held 1. The values (i.e. the last value just before WE went from 1 to 0) would be written in the register.

The lab is to implement the following.

1. Create registers for storing the values of RS1, RS2, WR, WE and F. The RS1 will be provided on the flip switches. We use a push button switch to latch RS1. The same flip switches will be used to provide RS2 and WR and different flip switches will be used for RS2 and WR respectively.
2. Create a 1 KHz clock by division of 25MHz clock by 25000 on the FPGA. In this lab, we shall use this 1KHz clock as clock for the rest of the signals.
3. Use a push down switch to generate a WE signal. It shall be one pulse of the clock whenever the push down switch is pressed. For this use the clock to count from 0 to 9. Whenever counter carried a value of 9 and the clock is 1 the WE is set to 1 provided the corresponding press switch was pressed and was detected as a 1 when a positive edge of the clock happened and the counter had a value of 0 or 1. This is a de-bouncing logic for the switch. WE should not become 1 again till the push down switch is released for at least 10ms (the same logic implemented with 1KHz clock).
4. Use the multiplexed display to show the value from the ALU (i.e. R) which must be stored in a register (other than the register file) for this purpose whenever the ALU function code was 4’b1110.
5. Show the D1 and D0 on two LEDs.
6. Perform a few instructions to show the outputs.

Instructions:

1. Set R1 to 8’b11111111.
2. Show the value R1 on the display (which shall show FF).
3. Set R3 to R1/2.
4. Show the value R3 on the display (which shall show 7F).
5. Set R4 to R3 + 1.
6. Show the value R4 on the display (which shall show 80)
7. Set R5 to R4/2.
8. Show the value of R5 on the display (which shall show 40).

How to execute an instruction?

1. Set RS1 on four flip switches. Press press-button for latching RS1. Note that these need not be de-bounced as the same value will be stored in the register carrying 4 bit RS1.
2. Set RS2 on the same flip switches. Press the press-button for latching RS2. RS2 will get latched in the 4-bit register carrying RS2 value.
3. Set WR on the same flip switches. Press the press-button for latching WR. WR will get latched in the 4-bit register carrying WR value
4. Set F on the same flip switches. Press the press-button for latching F. F will get latched in the 4-bit register carrying F value.
5. Press the press-switch earmarked for providing WE. The instruction is then computed using the stored values of RS1, RS2, WR and F. Please note that since you do not need to store the value in register file for displaying the contents of a register, the register output shall be visible as soon as the function code is latched.