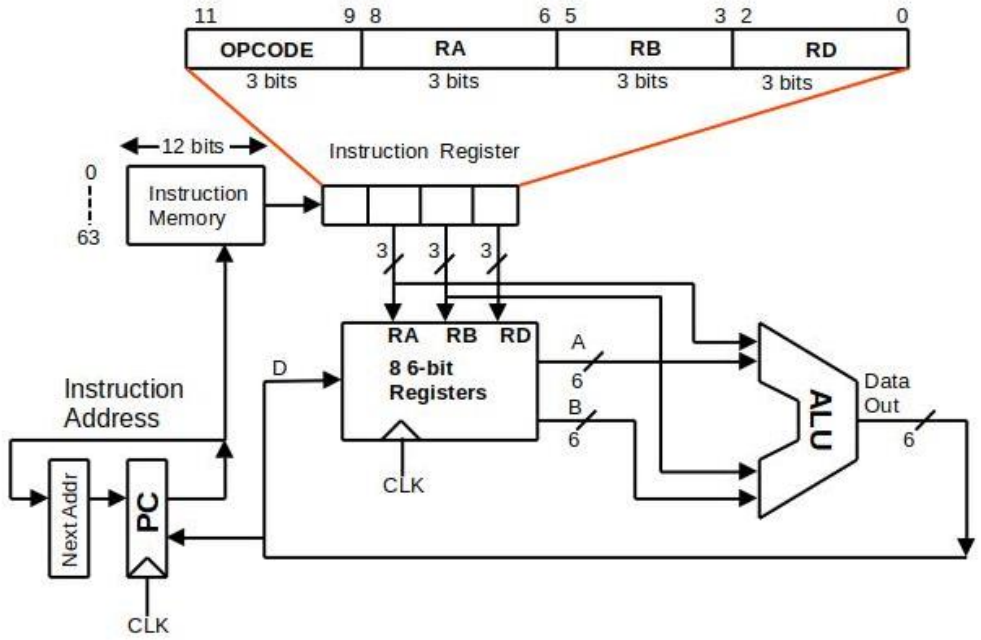
Lab 4 – EEE333 Hardware Design Language
Programming Logic Microprocessor Design

In this lab, you will write a Verilog code to implement a simple microprocessor as shown below:



DESCRIPTION:

You will design a CPU with a 12 bit instruction and 64 instruction memory locations. The register file will contain 8 registers, and each register will be 6 bits wide.

CODING BLOCK:

The machine will implement 2 basic integer operations (+, *). For multiplication, you should produce an output that returns the lower 8 bits of the total product bits. Multiplication may be **unsigned**.

OPCODE	Task to be Executed	Description	Category
001	LDI	Contents of RD = {RA, RB} (load immediate)	Load
010	ADD	Contents of RD = (contents of RA) + (contents of RB)	Math
011	ADI	Contents of RD = (contents of RA) + RB (add immediate)	
100	MUL	Contents of RD = (contents of RA) * (contents of RB)	
101	CMPJ	If (contents RA) >= (contents of RB) increment PC by RD immediate	Jump
110	JMP	Jump to immediate address location {RA, RB} for next instruction	
111	NOP	No operation (Do nothing during the instruction cycle	NOP
000	HALT	HALT Program Execution	Halt

Register File
The register file must be implemented in a **separate module**. It can be read from and written to.

Hex Display
Use your lab 1 file to display to the FPGA. Your ALU results should be displayed using the **OCTAL** number system, so 3 bits per 7 segment display.

ROM
Read Only Memory. Should contain the instruction set you want the microprocessor to run.
The code below is what will be needed to implement the Quartus ROM file into your design.

```
(* ram_init_file = "Lab4.mif" *) logic [11:0] mem[63:0];
........
assign *Instruction Code* = mem[*Program Counter*];
```

Where Lab4 is the name you have chosen for your lab module and must be inside quotes.

Block Description:

1. The Program Counter (PC) provides the address for the instruction to be executed. It should be incremented (PC=PC+1) in the "Write Back" cycle to point to the next instruction in the Instruction Memory (IM). If the current instruction is a JMP instruction, the PC will be updated during the "Write Back" cycle to point to the 'jump' address provided to it by the ALU output. The initial value of the PC must be 0.

2. The Instruction Memory is a 12-bit wide ROM with 64 locations. It outputs the instructions pointed to by the PC. This is to be created as a **memory initialization file using on chip memory (defined on Quartus).**

3. The Instruction Register contains the value being output from the Instruction Memory during the Instruction Fetch Cycle. It has four fields as indicated in the top diagram. OPCODE indicates which instruction is to be executed. RA points to a location in the register file to be used as data value A (or, in the case of the LDI instruction, contains the immediate data used by that instruction). RB points to a location in the register file to be used as data value B (or, in the case of the LDI and ADI instructions, contains the immediate value (data) used by those instructions). RD points to the location in the register file to which results will be written by those instructions which need to store a result.

4. The Register File is a three-ported memory which contains 8 8-bit values. The three ports consist of two read ports which are addressed by RA and RB and can be read simultaneously during the Instruction Decode cycle. The third port is a write port addressed by RD and is written to in the Write Back cycle.

5. The ALU is a logic block that takes data from any of four inputs and outputs a result as indicated by the opcode.

Board Specifications:

1. SW0 should be used as the **asynchronous active-high** reset signal. When on, you should be in reset mode. (FSMs should start in their reset state and you should clear the contents of the register file on reset).

2. CLOCK: The clock used for your state machine(s) should be the internal 50MHz clock (PIN_M9 for boards in the lab). See the next instruction regarding single-step mode.

3. SW1 will be used as a single-step mode switch. If SW1 is off, then once we come out of reset our microprocessor will execute whatever sequence exists in the ROM until it reaches a HALT instruction. If SW1 is on, the microprocessor will stop at the end of every EXECUTE cycle so that we can monitor various contents of the microprocessor as specified below.

4. KEY0 will be used as the **single-step advance** button. This means that when in single-step mode (that is, SW1 is high) every time we push KEY0 we will advance to the next instruction. The **8 lowest LEDs (LED [7:0]) should normally indicate the address of the current instruction**, which is the value stored in the PC. That is, if we are executing the instruction at 8'b00011010, then the LEDs should display off-off-off-on-on-off-on-off. This is the default for LED [7:0]. {SW4, SW3, SW2} will be used to select which output goes to the 7-segment display based on the table below. Align the display to the right using the number of segments required for the desired display. The last 3 sets of switches may be used as you see fit to help debug your code.

SW4	SW3	SW2	Display
0	0	0	Display RF {SW7, SW6, SW5}
0	0	1	Display IR
0	1	0	Display PC
0	1	1	Display OPCODE
1	0	0	Display ALU_out
1	0	1	Debug
1	1	0	Debug
1	1	1	Debug

Note 1: RF[addr] is the register file's data stored at the address which is indicated by switches 7,6,5 – in that order.

Note 2: It is **not expected** that we will see the 7-segment display values and LEDs when in full speed mode. It is, however, expected that these displays will be readable and display the correct data when halted at a halt instruction, halted between instructions in single-step mode, or at the end of execution.