EC551 Lab1 report
Bin Xu
Yimin Xu

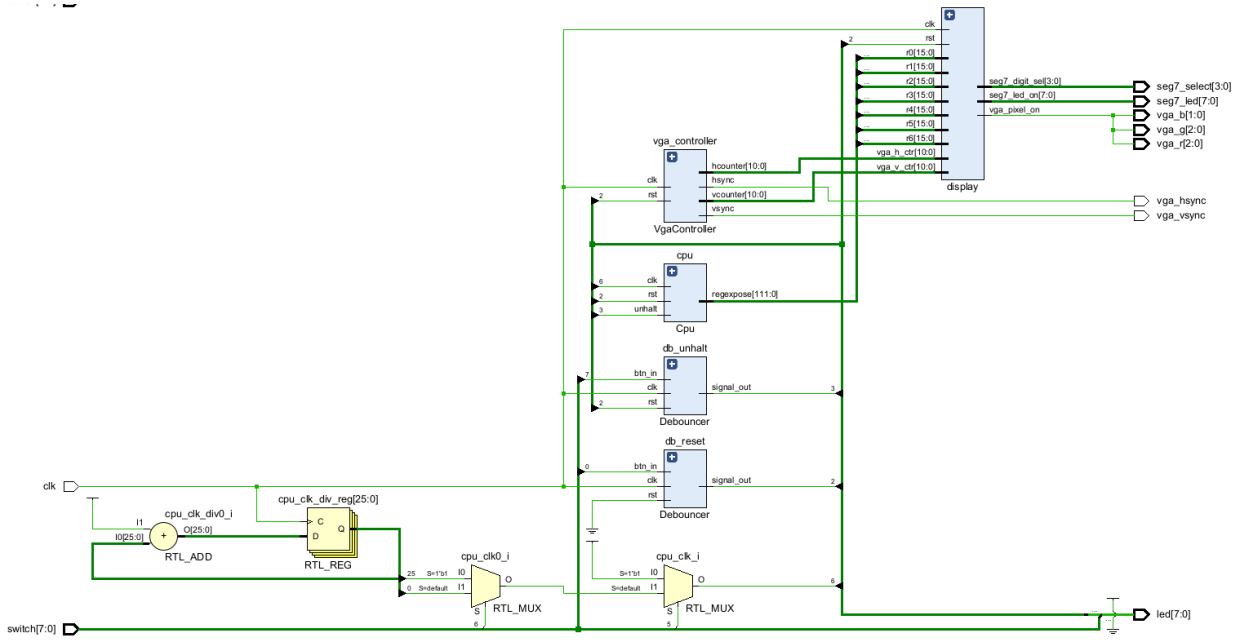
In this project, we build single cycle ALU microproces. There is main.v to contain the CPU logic, 7 segment display and VGA display.
The arithmetic logic part is in CPU.v. In CPU, we need memory, register file, ALU and decode.

- In the ALU, there are two main inputs and one output to do addition, subtraction, or, and and other operations.
- In the decode.v, we need to take an input instruction, and produce the various control signals and register outputs. The instruction decode stage is stateless. The output will tell the CPU where the rest of the component to move is.
- In the memory.v, we need two inputs read_addresses, two outputs read_datas, write_address and write data. One of the reads is to fetch the next instruction to be executed. The other read is for load word operations. The memory module will write output of ALU in every cycle. There is write_enable to control it. In the below simulation, it will show the write_address ad write data.
- In the register file, there is a 6 bits register. We add r6 which is read_only as the seventh register. The registers will read when clk is on posedge. We also need to pass the register to the 7 segment via the main module.

We need to connect to 7 segments and VGA.

- In the 7 segment, we need to show the 7 registers in four digits. We need 4 bits segment select and 8 bits anodes.
- Apart from the 7 segments part, we need to connect to VGA. In the VGA file, we create the char display to represent the different word and digit. Then we use the display terminal to reflect these characters on the VGA port. There is one output in the display terminal.
- Then we need the VGA controller to connect the VGA port. In the VGA controller , we need the clk to control the signal delivering to the screen. There is a counter at the beginning. We need the enable to control the output. We need to consider how to control the vertical and horizontal output.

We need a top module to connect the arithmetic logic to display and 7 segments. In the top module, we set four buttons, including reset, halt, clk speed control, and execution. For the output, we set the contents of the registers in hexadecimal to the monitor via VGA, and display the program counter in decimal on the 7-segment display.



Data Flow diagram:

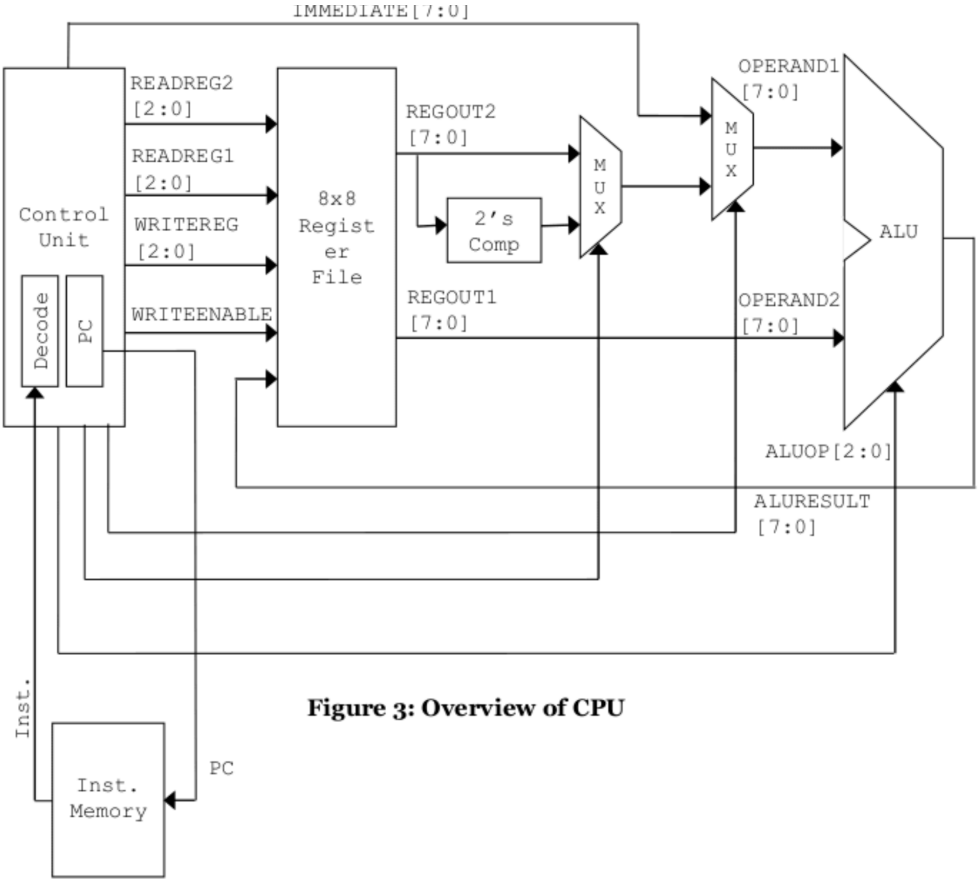
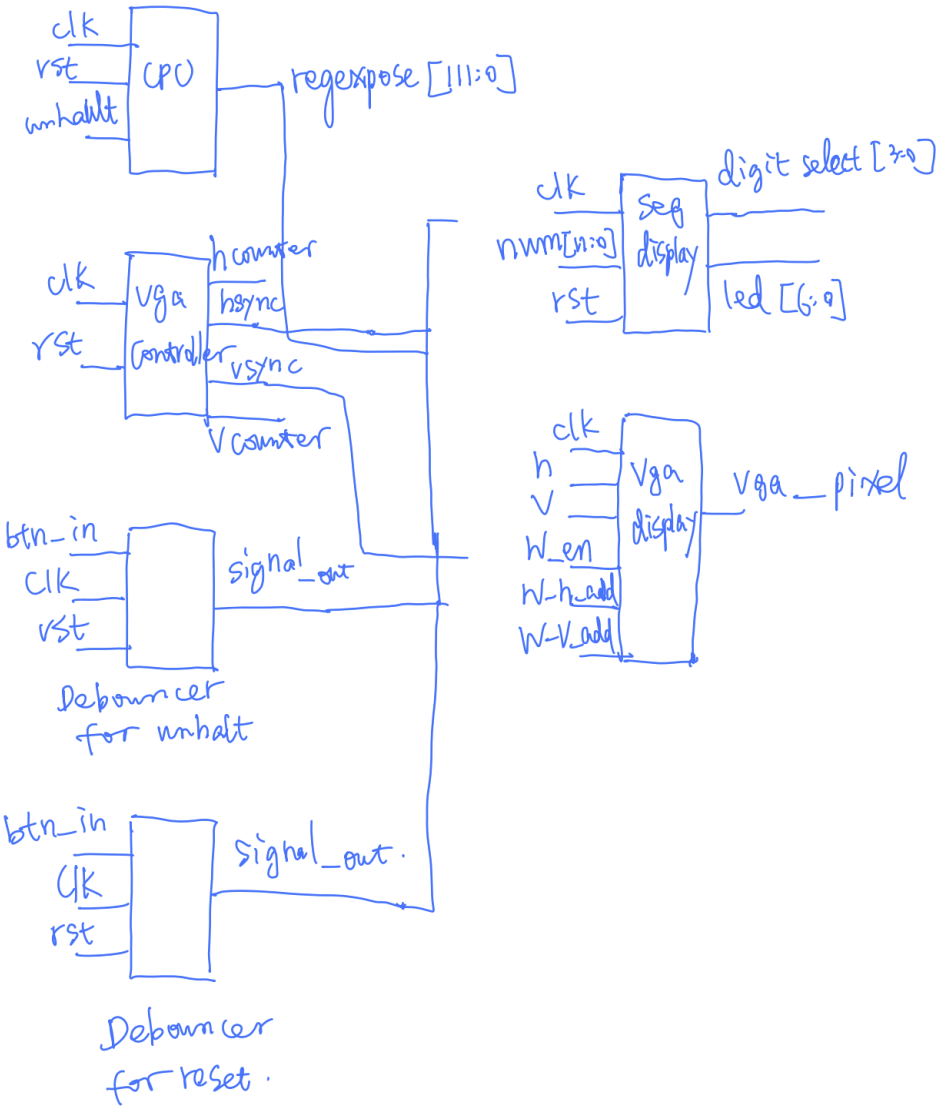


Figure 3: Overview of CPU

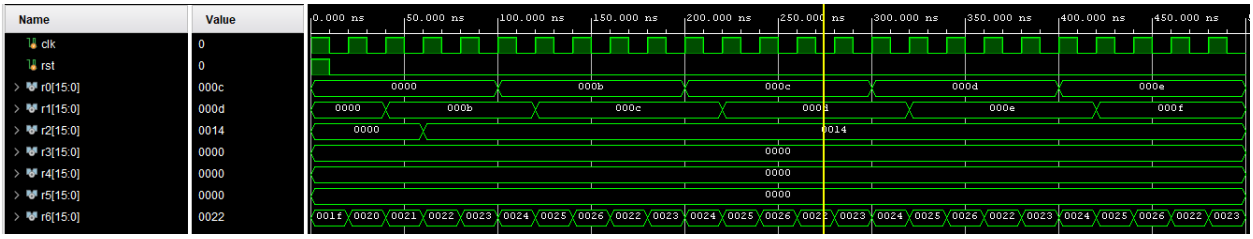


Below I will show some testwave for the different components.

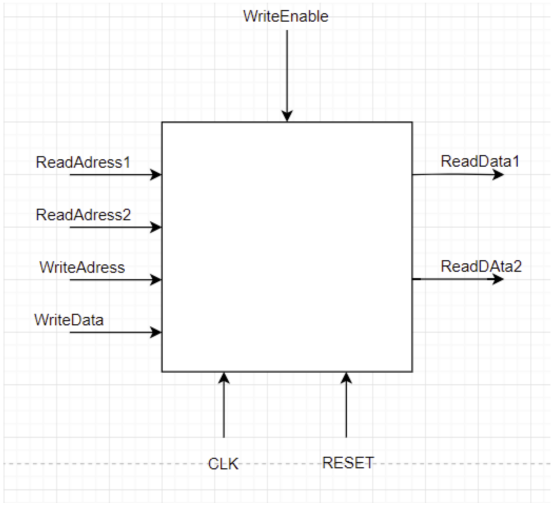
CPU:

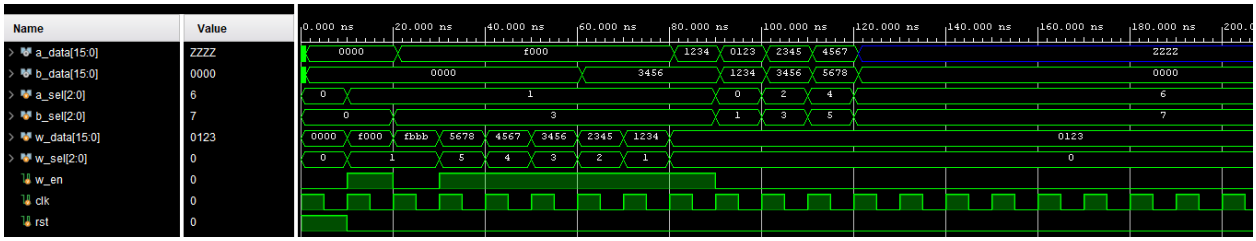
The first important part is the decoder. In the decoder, we have

- instruction decode: recognize different instructions, produce different control signal, then tell the CPU which component will be used in the rest of state.
- register select: tell which register data get from
- write enable: tell the data write to which location(memory, register)
- ALU opcode: do the calculation for some instruction
- Counter select: tell CPU what the counter from for some instruction(jump target, increment, or halt)



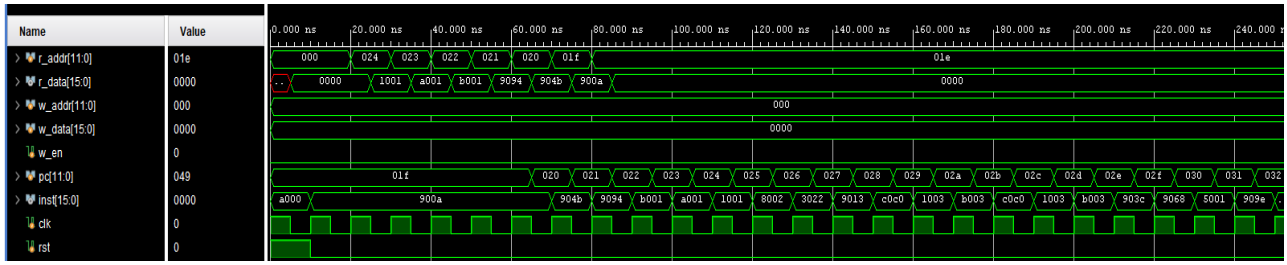
Register File is a memory space present within the CPU. It is used by the CPU to fetch and hold the data from the secondary memory devices. It detch data, then decode, then Execute. When we run the instruction, some value will be passed to register. We set 7 registers, one to six are 16 bits. The last one shows the pc counter. When the reg_enable is active, the register will write in every clk cycle.

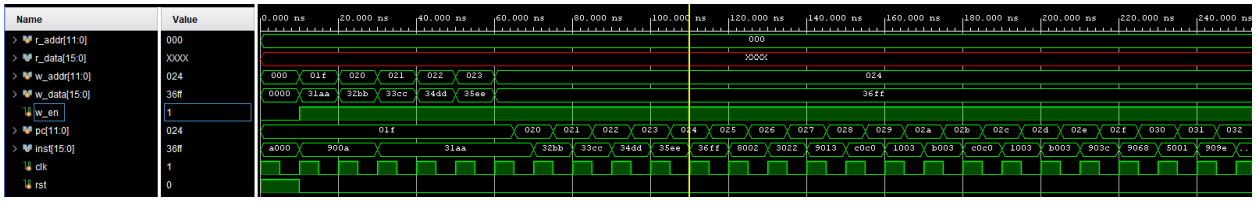




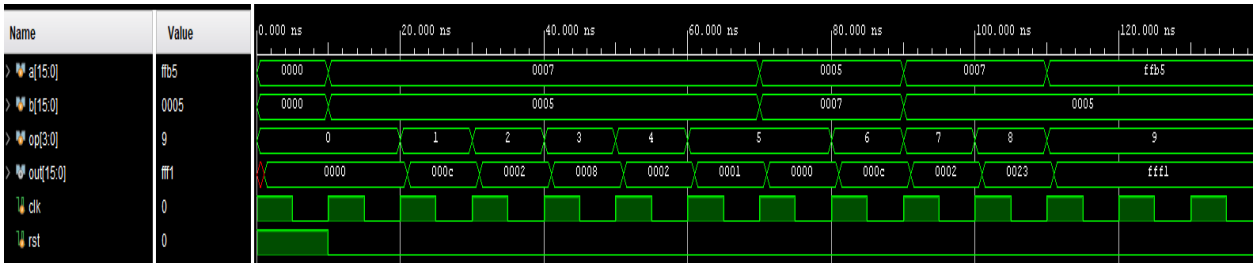
Memory uses the same logic as register. We need to set read data, read address, write data, write address. In memory we need 12-bit addressable space.
We need to use a $readmemh() to specify initial memory contents from a hex file.
We perform 2 reads and 1 write in every clk cycle.
One read is connected to the counter, then fetched the next instruction for execution. The other read is for loading the word. When there is a falling edge, memory will read the word. It is an asynchronous design, because we need to address the data from the register.

In the below testwave, we can find the write_data, write_address, read_data and read_address waveform. They show that my memory works.
In the memory design, when the write_enable is 1, memory will write the output from ALU.





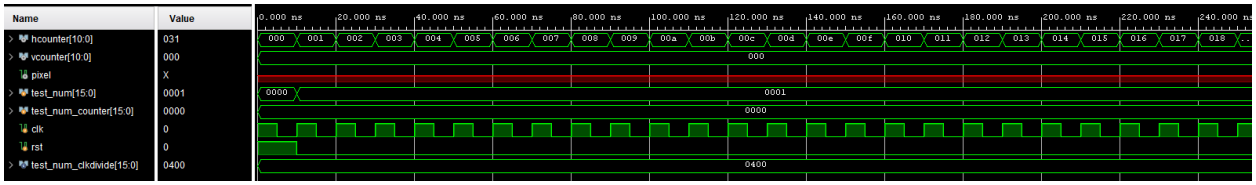
ALU: We need to finish four operations, ADD, SUB, INC, XOR instructions. We use behavioral verilog.



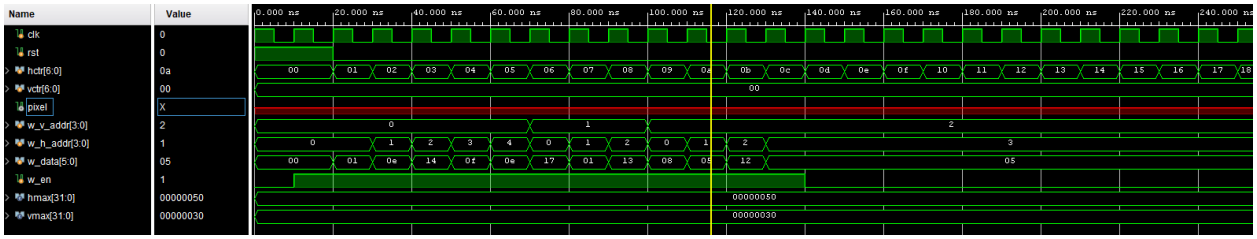
After we finish the arithmetic part, we need to design our display part.

- 7 segment display: In this file, we want to display the pc counter in the 7 segment on the FPGA board. On the FPGA board, we can get the decimal number of program count which is the same as the R6 from the register file.
- Vga display: in the vga display design, we need to reflect the R0 to R6 on the screen. Firstly we design the display char setting. We need the char.list, there are letters A-Z, digits 0-9, and a space character
- In the vga display, we hold a 2d memory array controlling which character is located at each square in a grid composed of 8x8 squares. The display initializes the register name on the left side, then shows the hex value of each register. The display reads which character is stored at the location, then passes the information to the char setting to decide if it should be active or not. There is default char stored in the char.list
- Vga controller: I download it from the support material and use it in the lab.

Display:



vga display:



For the assembly part, I upload all the files used in the assembly file. You can find tohow to use it in the txt file.