1a.

memfile.mem

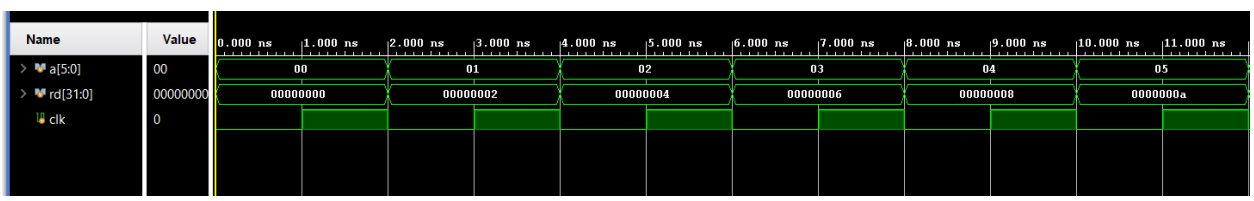
File Edit View

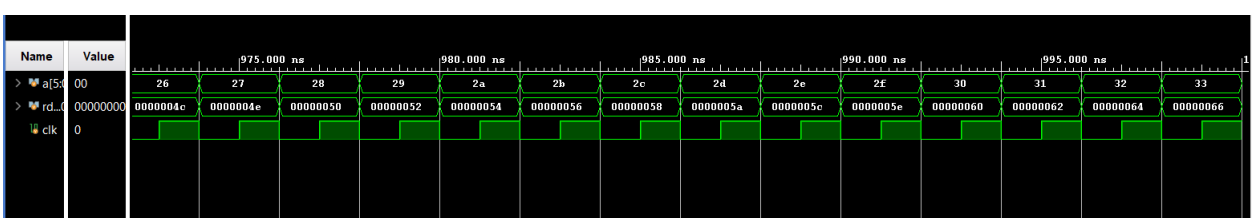
```
00000000
00000002
00000004
00000006
00000008
0000000a
0000000c
0000000e
00000010
00000012
00000014
00000016
00000018
0000001a
0000001c
0000001e
00000020
00000022
00000024
00000026
00000028
0000002a
0000002c
0000002e
00000030
00000032
00000034
00000036
00000038
0000003a
0000003c
0000003e
00000040
00000042
00000044
00000046
00000048
0000004a
0000004c
0000004e
```

1b.

```
`timescale 1ns / 1ps

module dmem_testbench();

logic [5:0] a;      // 6 bit input
logic [31:0] rd;    // 32 bit output
logic clk;          // clock

imem instantiated_imem(a, rd);
    // setup initial values
  initial begin
    clk = 0;    // set clock to 0

    // since we are only going to implement a lookup table the
    // output are hardcoded in the memfile. The 32 bit output are
    // written in hex values.
    // In order to output properly in vivado, I used a for loop to avoid
    // repeteated lines of code. This for loop would iterate through all
    // possible 6 bit combinations or 0 - 63 in decimal
    for(a = 0; a <= 63; a = a + 1)begin
        #2;     // wait for 1 clock cycle before checking next value
    end
  end

    // clock simulator
    // Simulate clock, 2ticks for clock cycle
    // hence 2ns each clock cycle
    always begin
        #1 clk = ~clk;
    end
endmodule
```

1c.

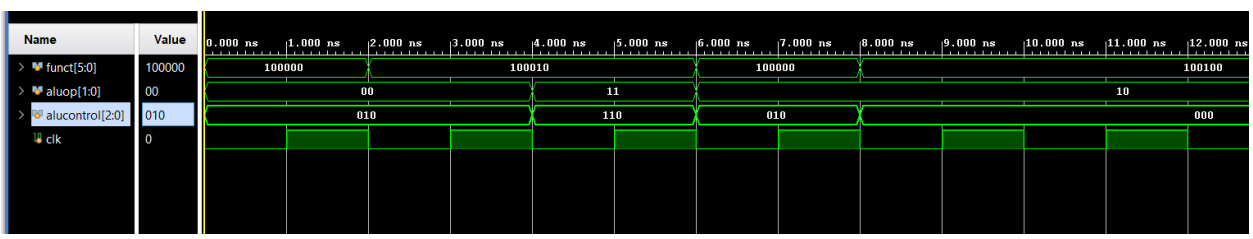




2a.

```
`timescale 1ns / 1ps

module aludec_testbench();

logic [5:0] funct;      // 6 bit funct code
logic [1:0] aluop;    // 1 bit aluop code
logic [2:0] alucontrol; // 3 bit output of alu
logic clk;          // clock

aludec instantiated_aludec(funct, aluop, alucontrol);
    // setup initial values
  initial begin
    clk = 0;    // set clock to 0

    // The aludec module will always read both values of funct and aluop
    // However, it will always check first the value of aluop.
    // If unsatisfied from the conditionals it would check the funct code.
    // Specific outputs for the alucontrol are given for every condition

    // Case 1: aluop: 00 -> valid output immediately to alucontrol 010
    funct = 'b100000; aluop = 'b00; #2;

    // wait for 1 clock cycle
    // Case 2: aluop: 00 -> valid output immediately to alucontrol 010
    funct = 'b100010; aluop = 'b00; #2;

    // wait for 1 clock cycle
    // Case 3: aluop: 11 -> invalid aluop check funct code: 100010 -> valid, output immediately alucontrol 110
    funct = 'b100010; aluop = 'b11; #2;

    // wait for 1 clock cycle
    // Case 4: aluop: 10 -> invalid aluop check funct code: 100000 -> valid, output immediately alucontrol 010
    funct = 'b100000; aluop = 'b10; #2;

    // wait for 1 clock cycle
    // Case 5: aluop: 10 -> invalid aluop check funct code: 100100 -> valid, output immediately alucontrol 000
    funct = 'b100100; aluop = 'b10; #2;
  end

    // clock simulator
    // Simulate clock, 2ticks for clock cycle
    // hence 2ns each clock cycle
    always begin
        #1 clk = ~clk;
    end
endmodule
```

2b.



3.

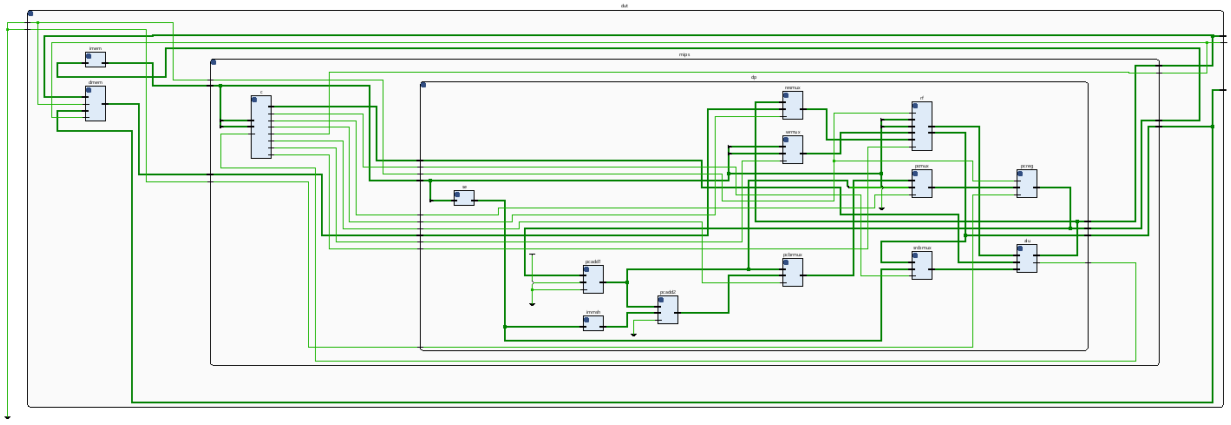


Figure 1. Wide Shot of the schematic generated by Vivado

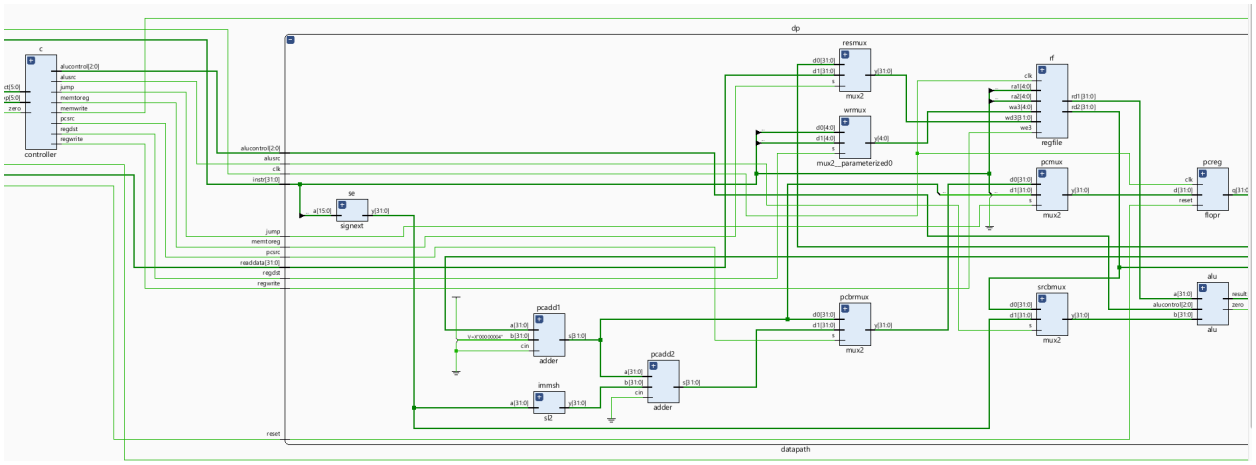


Figure 2. Close up shot on the modules used. Schematic generated using Vivado

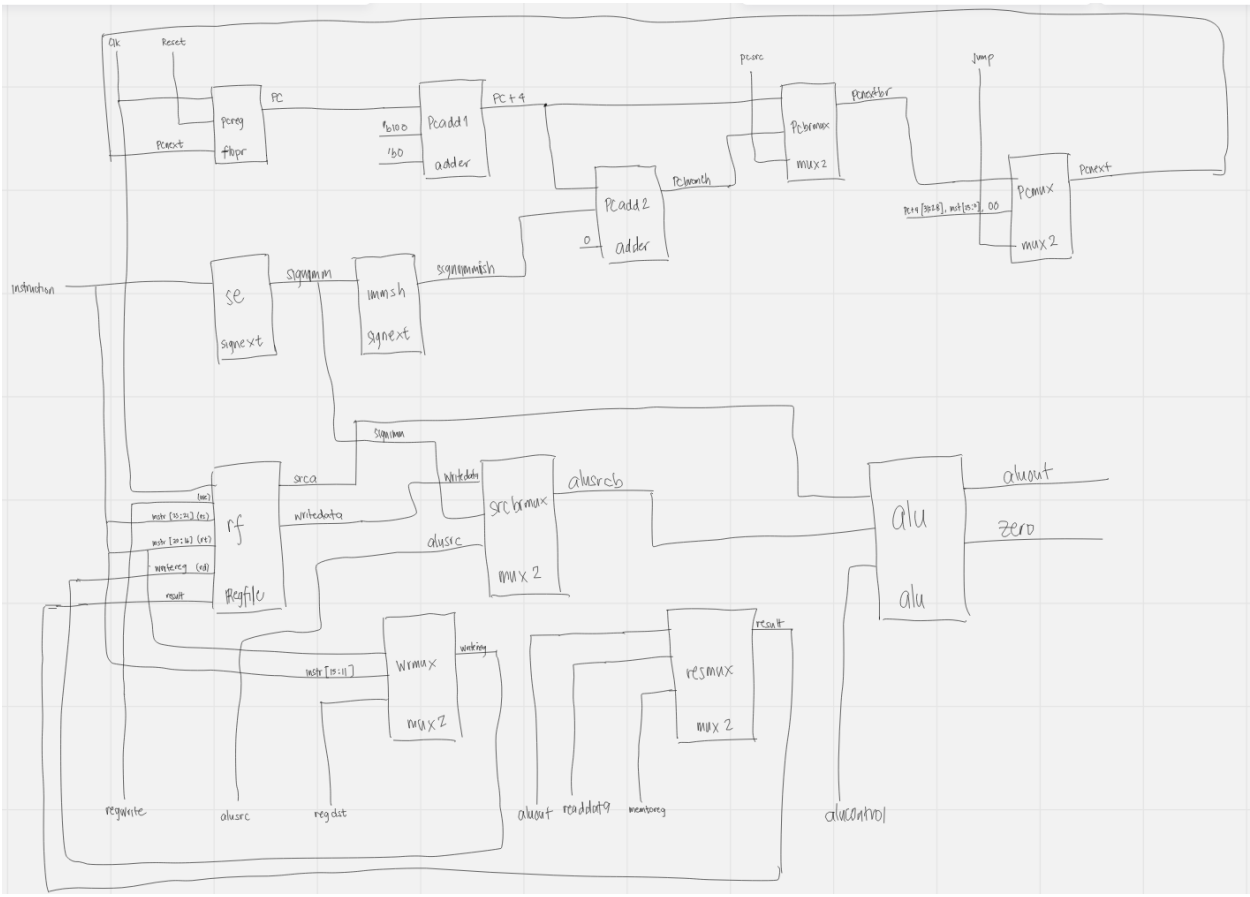


Figure 3. Drawn Schematic of the datapath module