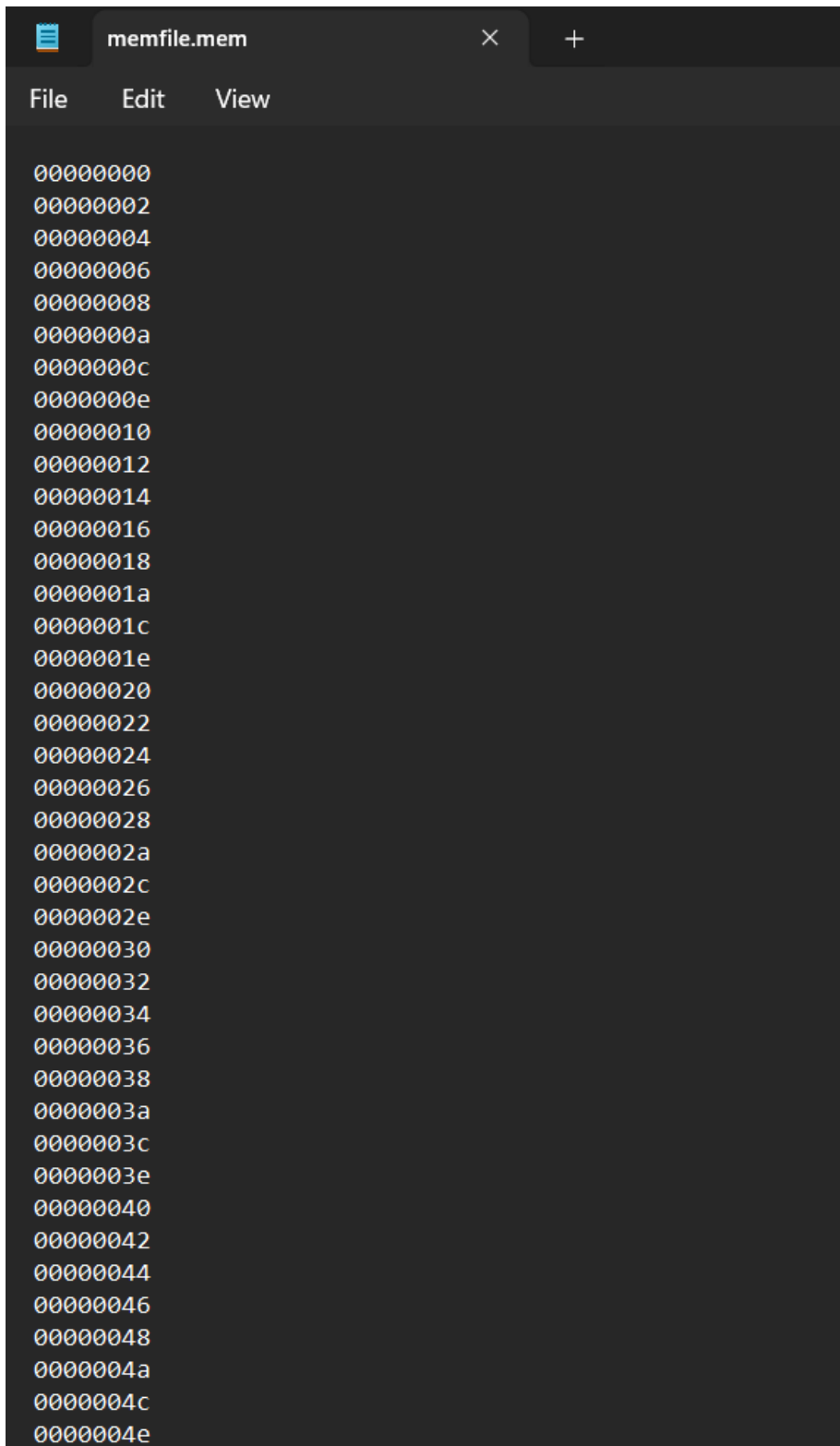


1a.



The image shows a screenshot of a text editor window with the title bar 'memfile.mem'. The window has a menu bar with 'File', 'Edit', and 'View'. The main content area displays a list of hexadecimal addresses, starting from 00000000 and ending at 0000004e, with increments of 2. The addresses are listed on the left side of the editor, and the rest of the editor area is empty.

```
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
```

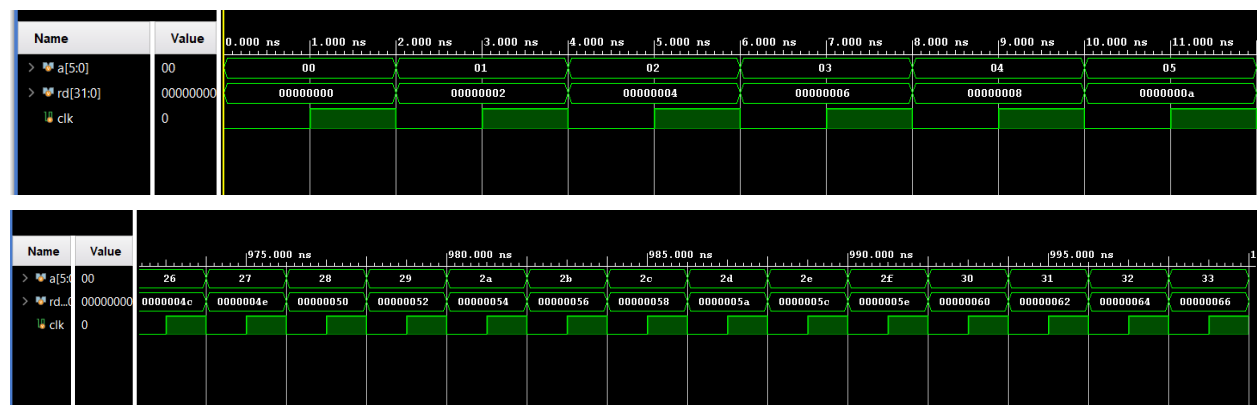
1b.

```

1  `timescale 1ns / 1ps
2
3  module dmem_testbench();
4
5  logic [5:0] a;      // 6 bit input
6  logic [31:0] rd;    // 32 bit output
7  logic clk;         // clock
8
9  imem instantiated_imem(a, rd);
10
11  // setup initial values
12  initial begin
13
14      // since we are only going to implement a lookup table the
15      // output are hardcoded in the memfile. The 32 bit output are
16      // written in hex values.
17      // In order to output properly in vivado, I used a for loop to avoid
18      // repeated lines of code. This for loop would iterate through all
19      // possible 6 bit combinations or 0 - 63 in decimal
20      for(a = 0; a <= 63; a = a + 1)begin
21          #2;      // wait for 1 clock cycle before checking next value
22      end
23  end
24
25  // clock simulator
26  // Simulate clock, 2 ticks for clock cycle
27  // hence 2ns each clock cycle
28  always begin
29      #1 clk = ~clk;
30  end
31 endmodule
32

```

1c.



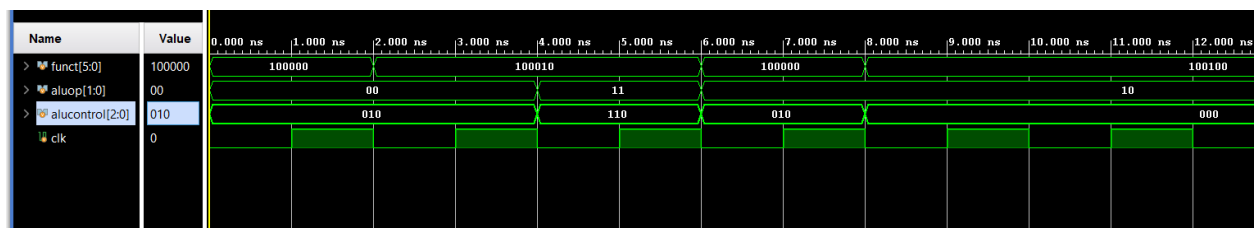
2a.

```

1  `timescale 1ns / 1ps
2
3  module aludec_testbench();
4
5  logic [5:0] funct;    // 6 bit funct code
6  logic [1:0] aluop;    // 1 bit aluop code
7  logic [2:0] alucontrol; // 3 bit output of alu
8  logic clk;           // clock
9
10 aludec instantiated_aludec(funct, aluop, alucontrol);
11 // setup initial values
12 initial begin
13     clk = 0; // set clock to 0
14
15     // The aludec module will always read both values of funct and aluop
16     // However, it will always check first the value of aluop.
17     // If unsatisfied from the conditionals it would check the funct code.
18     // Specific outputs for the alucontrol are given for every condition
19
20     // Case 1: aluop: 00 -> valid output immediately to alucontrol 010
21     funct = 'b100000; aluop = 'b00; #2;
22
23     // wait for 1 clock cycle
24     // Case 2: aluop: 00 -> valid output immediately to alucontrol 010
25     funct = 'b100010; aluop = 'b00; #2;
26
27     // wait for 1 clock cycle
28     // Case 3: aluop: 11 -> invalid aluop check funct code: 100010 -> valid, output immediately alucontrol 110
29     funct = 'b100010; aluop = 'b11; #2;
30
31     // wait for 1 clock cycle
32     // Case 4: aluop: 10 -> invalid aluop check funct code: 100000 -> valid, output immediately alucontrol 010
33     funct = 'b100000; aluop = 'b10; #2;
34
35     // wait for 1 clock cycle
36     // Case 5: aluop: 10 -> invalid aluop check funct code: 100100 -> valid, output immediately alucontrol 000
37     funct = 'b100100; aluop = 'b10; #2;
38 end
39
40 // clock simulator
41 // Simulate clock, 2ticks for clock cycle
42 // hence 2ns each clock cycle
43 always begin
44     #1 clk = ~clk;
45 end
46 endmodule

```

2b.



3.

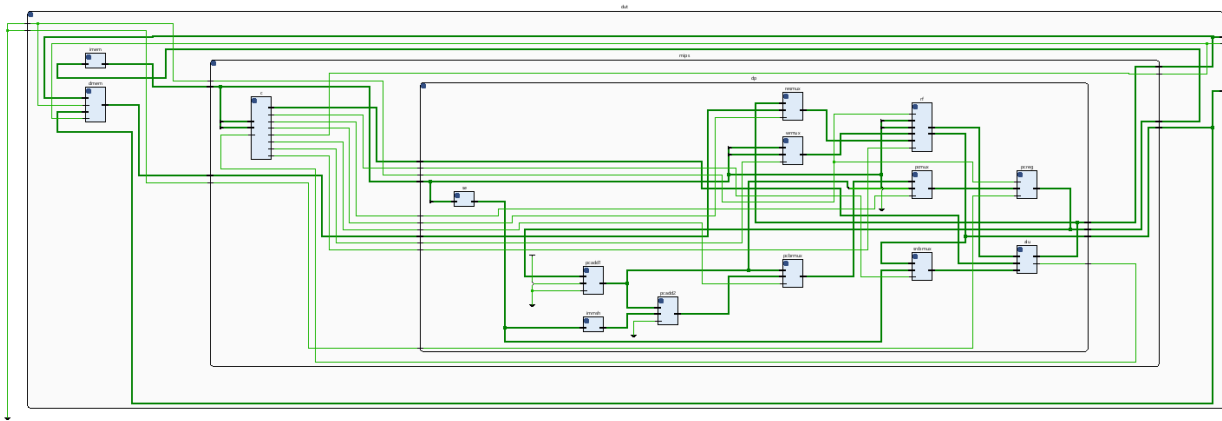


Figure 1. Wide Shot of the schematic generated by Vivado

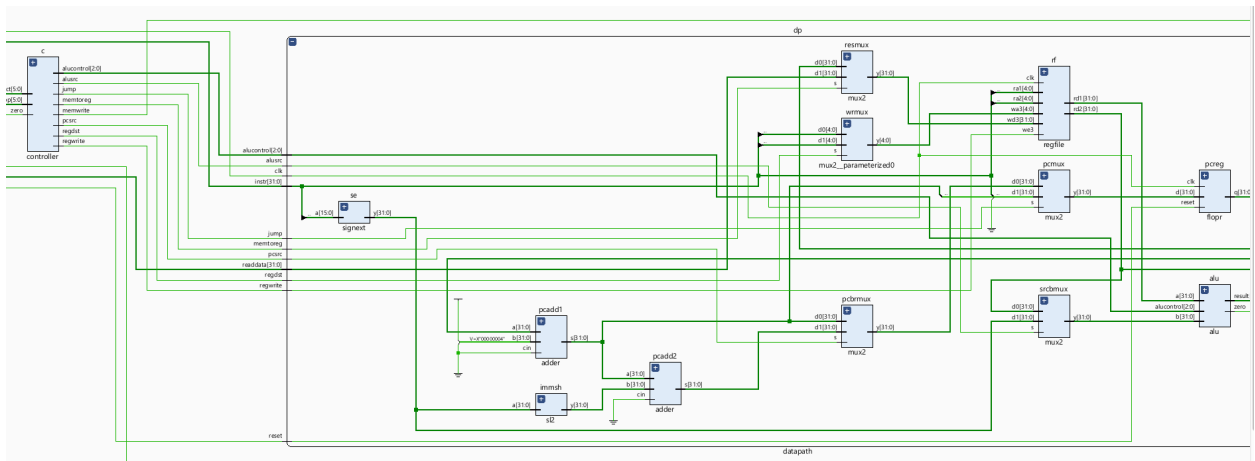


Figure 2. Close up shot on the modules used. Schematic generated using Vivado

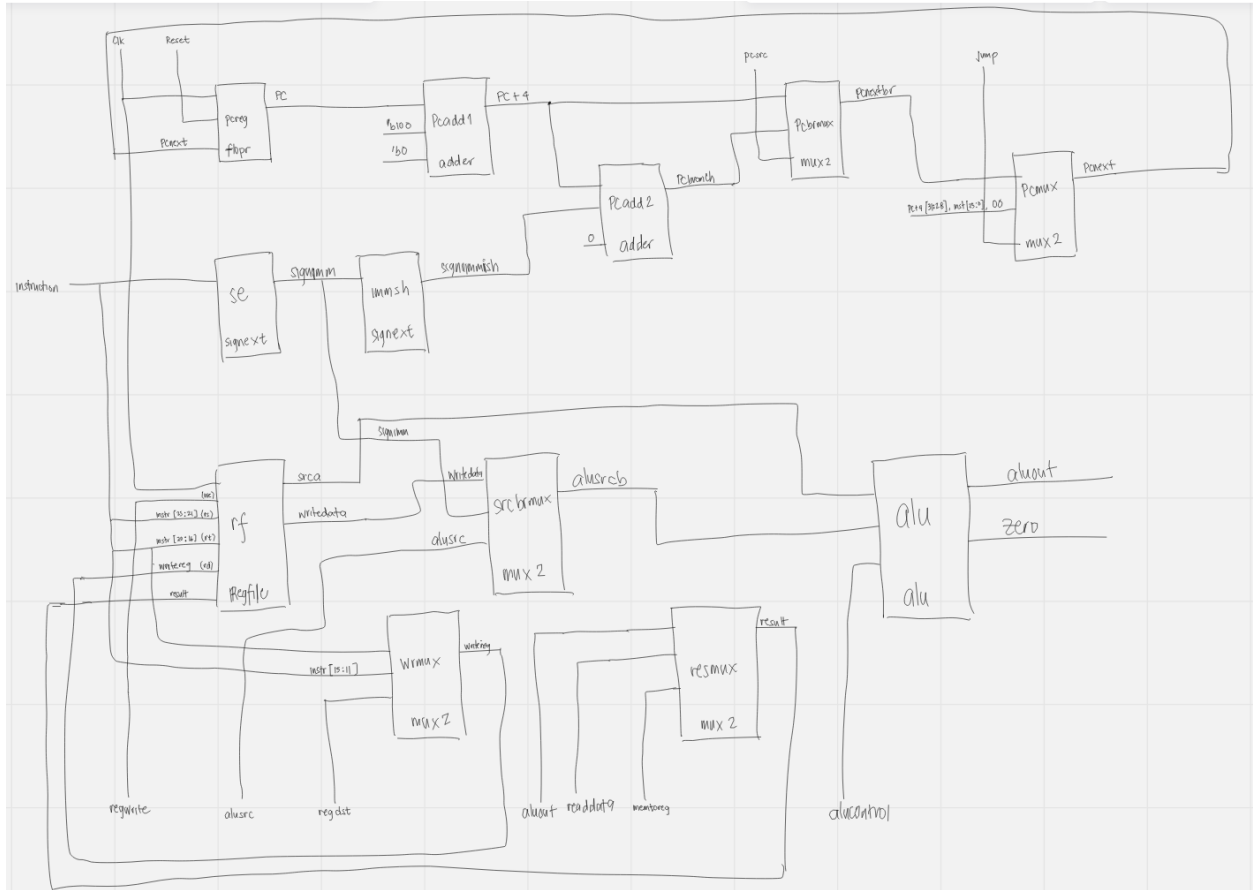


Figure 3. Drawn Schematic of the datapath module