Phase 3 Report

Luka Gobovic - 20215231

Daniel Joseph - 20217305

ELEC 374

March 24, 2023

Academic Integrity Statement

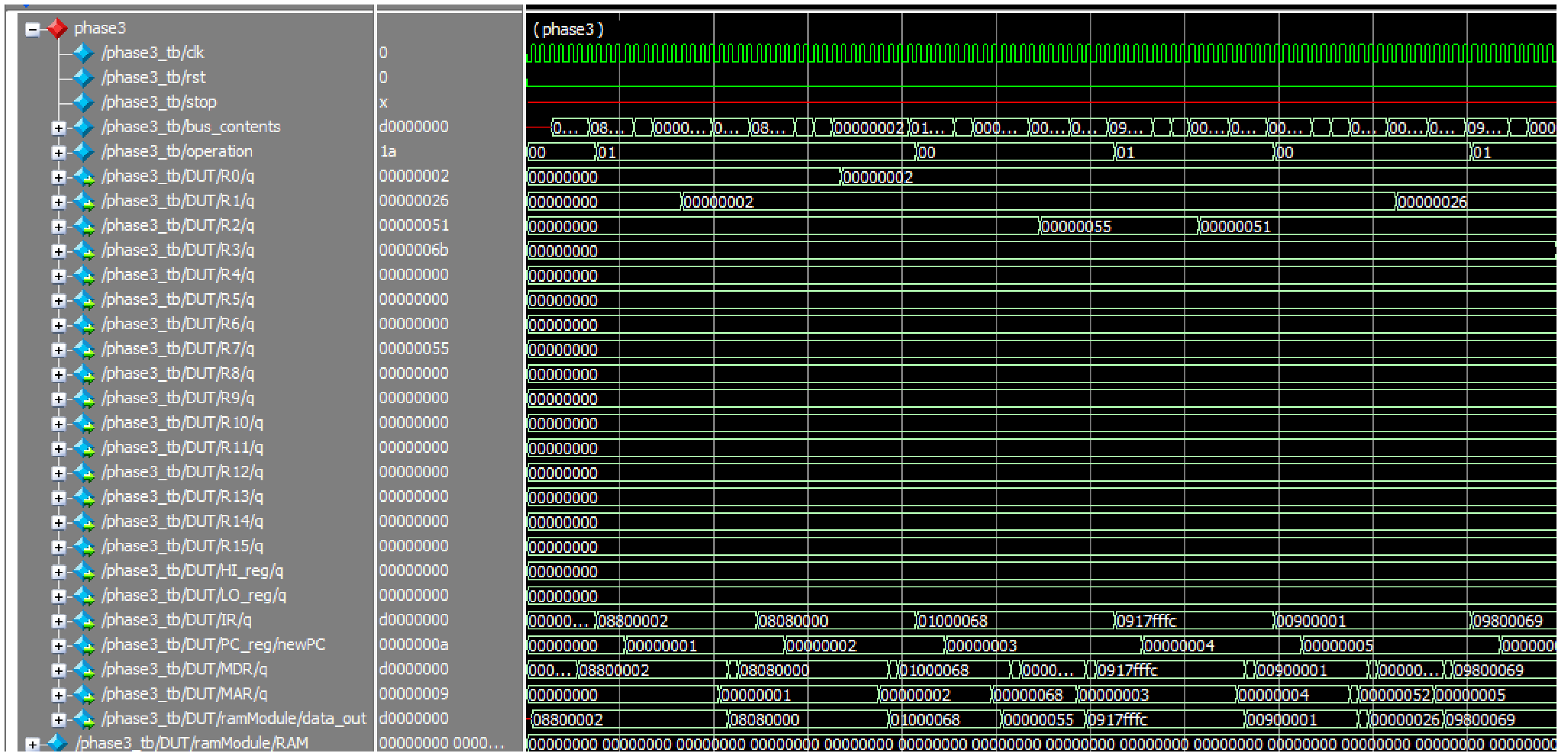
We do hereby verify that this written lab report is our own work and contains our own original ideas, concepts, and designs. No portion of this report has been copied in whole or in part from another source, with the possible exception of properly referenced material.

What Was Built Upon

The group did not add any extra functionality other than what was requested in the Phase 3 document. The only other aspect that was tweaked was that each of the instructions only ran for how ever many signals were present in its original control sequence. So for example, move from high only had one extra cycle after the fetch and decode. This way we avoided having extra idle cycles. Additionally, the group needed to add extra delays between each sequence to ensure that registers were correctly capturing newly updated data, and that the RAM also functioned as expected.

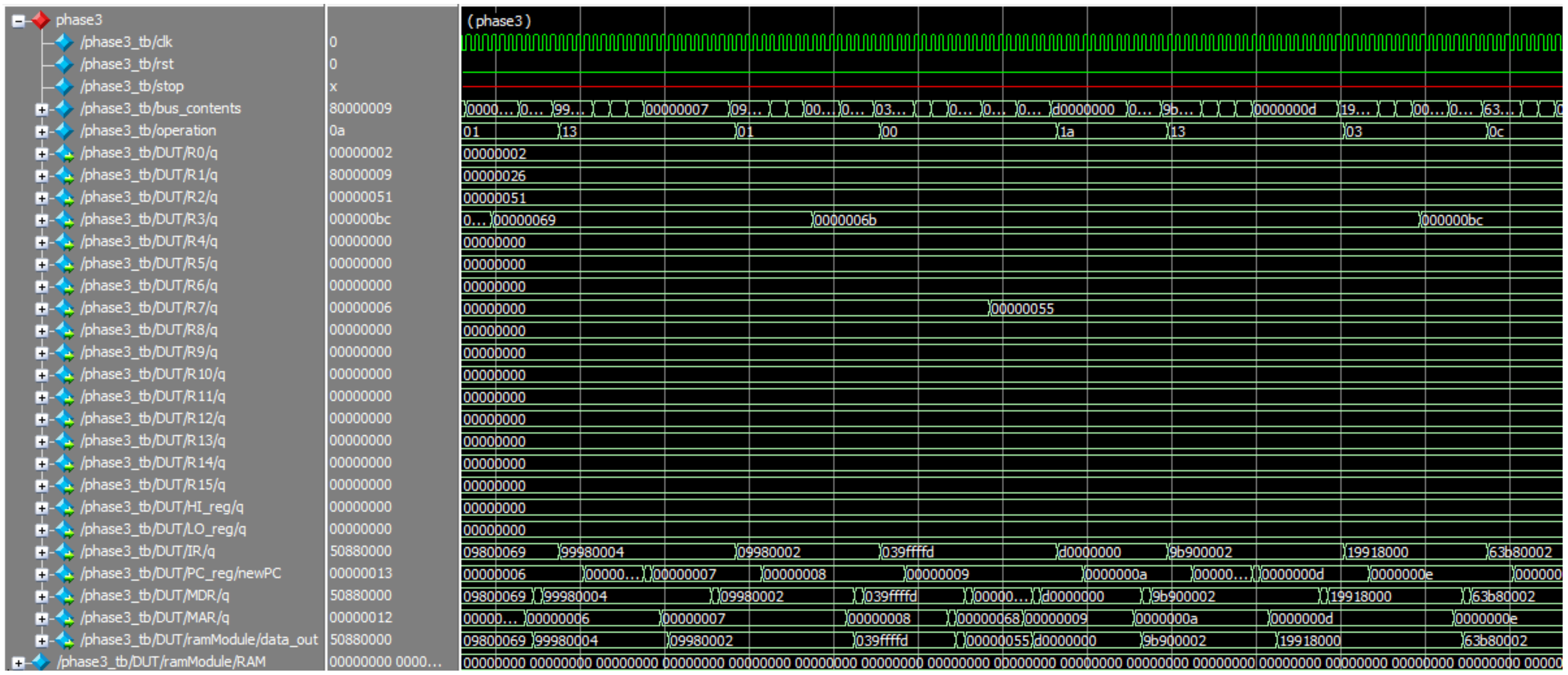
Simulation Waveforms

Included below are a series of ModelSim outputs for the set of instructions outlined in 3.2 of the lab procedure. The screenshots include the contents of the PC, IR, MAR, MDR, HI, LO and R0 to R15 registers, and how each are effected by each instruction can be seen. The IR shows the current instructions being executed. The figure captions below each of the images outlines which instructions are currently being displayed on the waveforms.



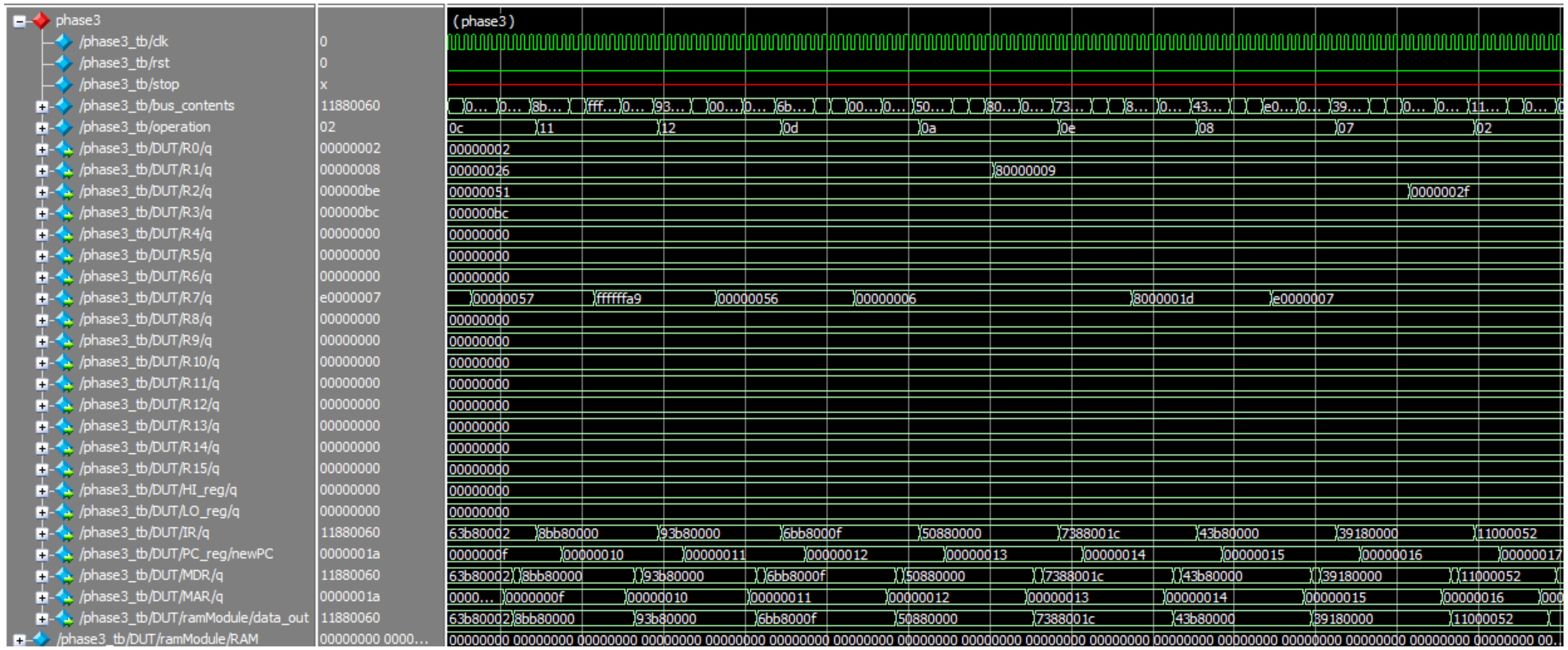
```
phase3                                                (phase3)
/phase3_tb/clk                       0
/phase3_tb/rst                       0
/phase3_tb/stop                      x
/phase3_tb/bus_contents              d0000000
/phase3_tb/operation                 1a           00  01  00  01  00  01
/phase3_tb/DUT/R0/q                  00000002     00000000  00000002
/phase3_tb/DUT/R1/q                  00000026     00000000  00000002  00000026
/phase3_tb/DUT/R2/q                  00000051     00000000  00000055  00000051
/phase3_tb/DUT/R3/q                  0000006b     00000000
/phase3_tb/DUT/R4/q                  00000000     00000000
/phase3_tb/DUT/R5/q                  00000000     00000000
/phase3_tb/DUT/R6/q                  00000000     00000000
/phase3_tb/DUT/R7/q                  00000055     00000000
/phase3_tb/DUT/R8/q                  00000000     00000000
/phase3_tb/DUT/R9/q                  00000000     00000000
/phase3_tb/DUT/R10/q                 00000000     00000000
/phase3_tb/DUT/R11/q                 00000000     00000000
/phase3_tb/DUT/R12/q                 00000000     00000000
/phase3_tb/DUT/R13/q                 00000000     00000000
/phase3_tb/DUT/R14/q                 00000000     00000000
/phase3_tb/DUT/R15/q                 00000000     00000000
/phase3_tb/DUT/HI_reg/q              00000000     00000000
/phase3_tb/DUT/LO_reg/q              00000000     00000000
/phase3_tb/DUT/IR/q                  d0000000     00000...  08800002  08080000  01000068  0917fffc  00900001  09800069
/phase3_tb/DUT/PC_reg/newPC          0000000a     00000000  00000001  00000002  00000003  00000004  00000005  000000
/phase3_tb/DUT/MDR/q                 d0000000     000...  08800002  08080000  01000068  0000...  0917fffc  00900001  00000...  09800069
/phase3_tb/DUT/MAR/q                 00000009     00000000  00000001  00000002  00000068  00000003  00000004  00000052  00000005
/phase3_tb/DUT/ramModule/data_out    d0000000     08800002  08080000  01000068  00000055  0917fffc  00900001  00000026  09800069
/phase3_tb/DUT/ramModule/RAM         00000000 0000...  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000000
```

Figure 1: The waveform for instructions 1 (ldi R1, 2) to 5 (ld R1, 1(R2)).



```
phase3
  /phase3_tb/clk                      0
  /phase3_tb/rst                      0
  /phase3_tb/stop                     x
  /phase3_tb/bus_contents             80000009
  /phase3_tb/operation                0a
  /phase3_tb/DUT/R0/q                 00000002
  /phase3_tb/DUT/R1/q                 80000009
  /phase3_tb/DUT/R2/q                 00000051
  /phase3_tb/DUT/R3/q                 000000bc
  /phase3_tb/DUT/R4/q                 00000000
  /phase3_tb/DUT/R5/q                 00000000
  /phase3_tb/DUT/R6/q                 00000000
  /phase3_tb/DUT/R7/q                 00000006
  /phase3_tb/DUT/R8/q                 00000000
  /phase3_tb/DUT/R9/q                 00000000
  /phase3_tb/DUT/R10/q                00000000
  /phase3_tb/DUT/R11/q                00000000
  /phase3_tb/DUT/R12/q                00000000
  /phase3_tb/DUT/R13/q                00000000
  /phase3_tb/DUT/R14/q                00000000
  /phase3_tb/DUT/R15/q                00000000
  /phase3_tb/DUT/HI_reg/q             00000000
  /phase3_tb/DUT/LO_reg/q             00000000
  /phase3_tb/DUT/IR/q                 50880000
  /phase3_tb/DUT/PC_reg/newPC         00000013
  /phase3_tb/DUT/MDR/q                50880000
  /phase3_tb/DUT/MAR/q                00000012
  /phase3_tb/DUT/ramModule/data_out   50880000
/phase3_tb/DUT/ramModule/RAM          00000000 0000...
```

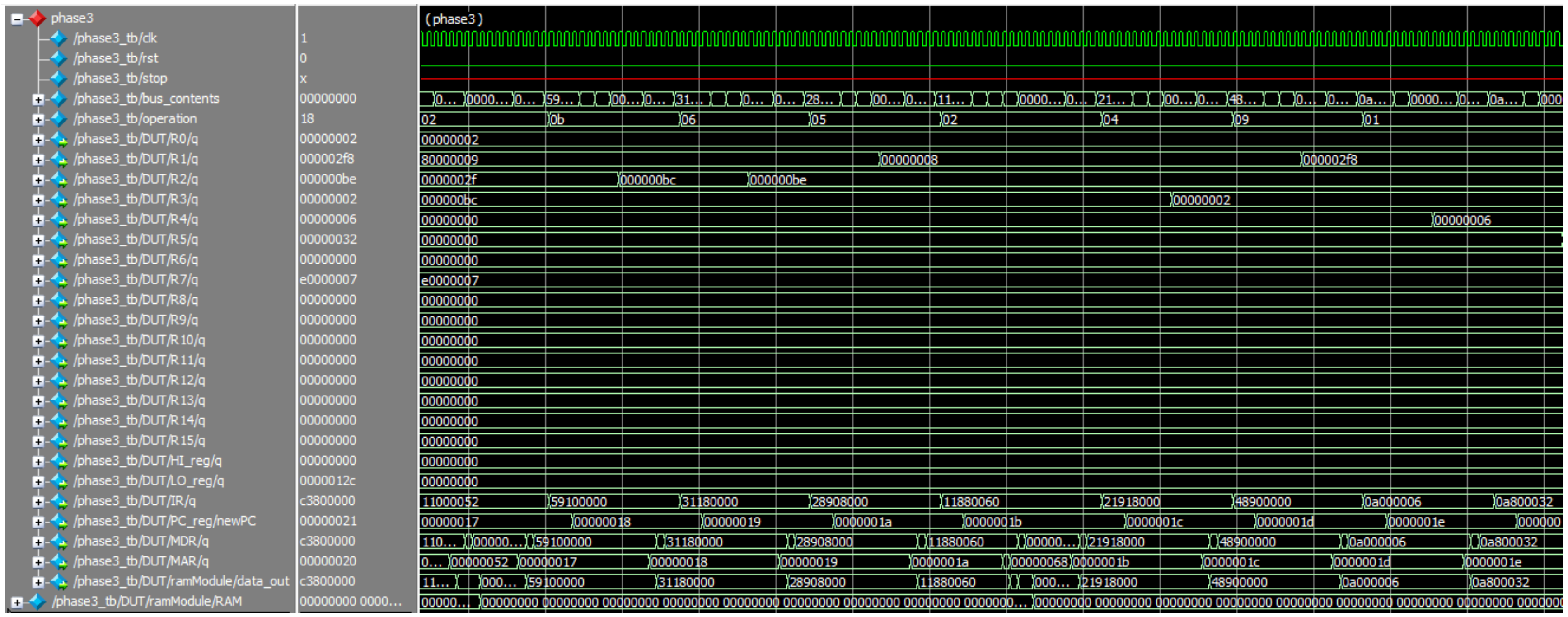
Figure 2: The waveforms starting with instruction 6 (ldi R3, $69), and ending with instruction 14 at target (add R3, R2, R3).



```
phase3
/phase3_tb/clk                      0
/phase3_tb/rst                      0
/phase3_tb/stop                     x
/phase3_tb/bus_contents             11880060
/phase3_tb/operation                02
/phase3_tb/DUT/R0/q                 00000002
/phase3_tb/DUT/R1/q                 00000008
/phase3_tb/DUT/R2/q                 000000be
/phase3_tb/DUT/R3/q                 000000bc
/phase3_tb/DUT/R4/q                 00000000
/phase3_tb/DUT/R5/q                 00000000
/phase3_tb/DUT/R6/q                 00000000
/phase3_tb/DUT/R7/q                 e0000007
/phase3_tb/DUT/R8/q                 00000000
/phase3_tb/DUT/R9/q                 00000000
/phase3_tb/DUT/R10/q                00000000
/phase3_tb/DUT/R11/q                00000000
/phase3_tb/DUT/R12/q                00000000
/phase3_tb/DUT/R13/q                00000000
/phase3_tb/DUT/R14/q                00000000
/phase3_tb/DUT/R15/q                00000000
/phase3_tb/DUT/HI_reg/q             00000000
/phase3_tb/DUT/LO_reg/q             00000000
/phase3_tb/DUT/IR/q                 11880060
/phase3_tb/DUT/PC_reg/newPC         0000001a
/phase3_tb/DUT/MDR/q                11880060
/phase3_tb/DUT/MAR/q                0000001a
/phase3_tb/DUT/ramModule/data_out   11880060
/phase3_tb/DUT/ramModule/RAM        00000000 0000...
```

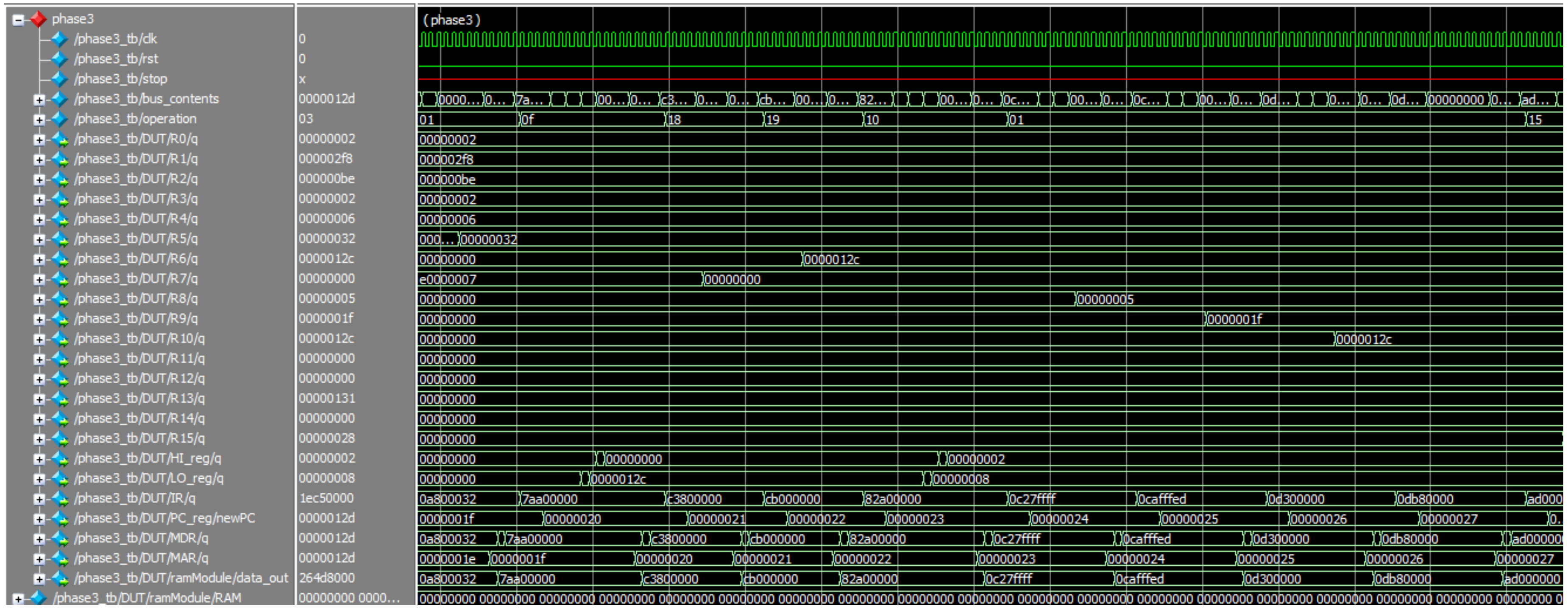
(phase3)
operation: 0c 11 12 0d 0a 0e 08 07 02
R0: 00000002
R1: 00000026 80000009
R2: 00000051 0000002f
R3: 000000bc
R7: 00000057 ffffffa9 00000056 00000006 8000001d e0000007
IR: 63b80002 8bb80000 93b80000 6bb8000f 50880000 7388001c 43b80000 39180000 11000052
PC: 0000000f 00000010 00000011 00000012 00000013 00000014 00000015 00000016 00000017
MDR: 63b80002 8bb80000 93b80000 6bb8000f 50880000 7388001c 43b80000 39180000 11000052
MAR: 0000000f 00000010 00000011 00000012 00000013 00000014 00000015 00000016
data_out: 63b80002 8bb80000 93b80000 6bb8000f 50880000 7388001c 43b80000 39180000 11000052

Figure 3: The waveforms for the instruction starting with addi R7, R7, 2 to instruction 22 (shr R2, R3, R0).



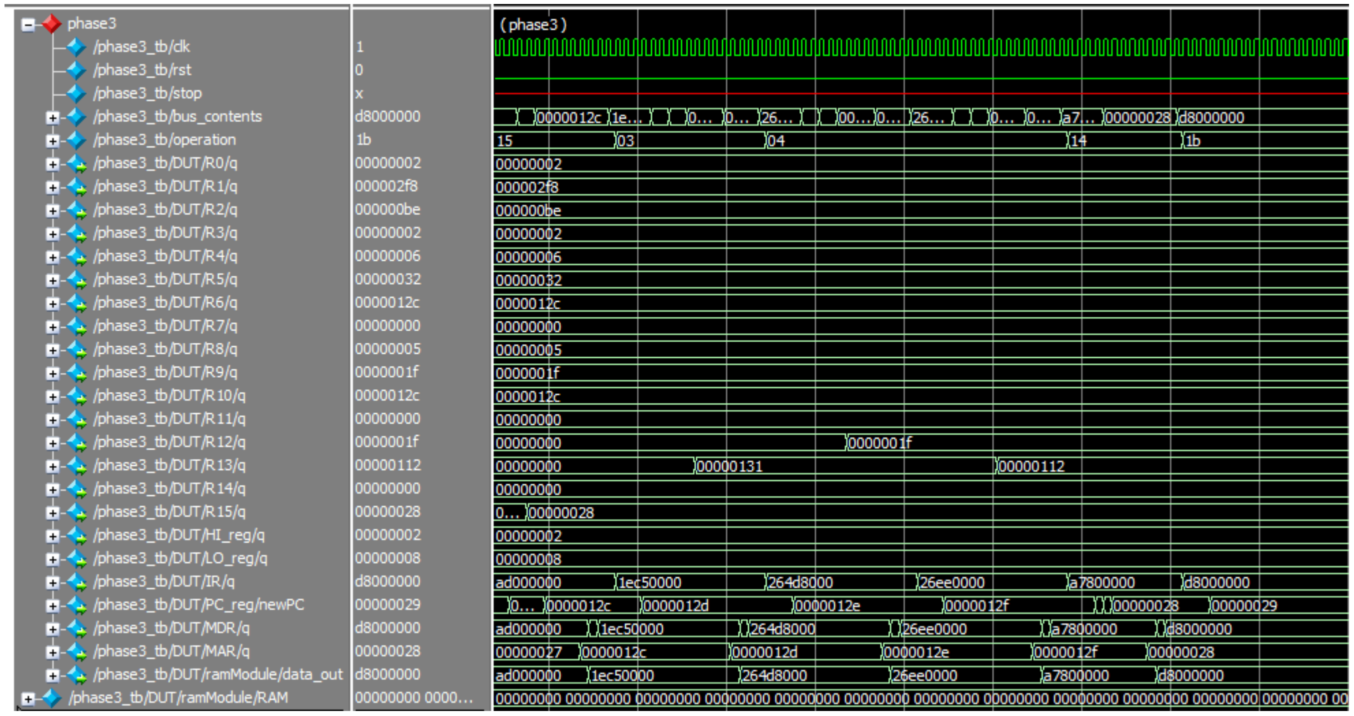
phase3	
/phase3_tb/clk	1
/phase3_tb/rst	0
/phase3_tb/stop	x
/phase3_tb/bus_contents	00000000
/phase3_tb/operation	18
/phase3_tb/DUT/R0/q	00000002
/phase3_tb/DUT/R1/q	000002f8
/phase3_tb/DUT/R2/q	000000be
/phase3_tb/DUT/R3/q	00000002
/phase3_tb/DUT/R4/q	00000006
/phase3_tb/DUT/R5/q	00000032
/phase3_tb/DUT/R6/q	00000000
/phase3_tb/DUT/R7/q	e0000007
/phase3_tb/DUT/R8/q	00000000
/phase3_tb/DUT/R9/q	00000000
/phase3_tb/DUT/R10/q	00000000
/phase3_tb/DUT/R11/q	00000000
/phase3_tb/DUT/R12/q	00000000
/phase3_tb/DUT/R13/q	00000000
/phase3_tb/DUT/R14/q	00000000
/phase3_tb/DUT/R15/q	00000000
/phase3_tb/DUT/HI_reg/q	00000000
/phase3_tb/DUT/LO_reg/q	0000012c
/phase3_tb/DUT/IR/q	c3800000
/phase3_tb/DUT/PC_reg/newPC	00000021
/phase3_tb/DUT/MDR/q	c3800000
/phase3_tb/DUT/MAR/q	00000020
/phase3_tb/DUT/ramModule/data_out	c3800000
/phase3_tb/DUT/ramModule/RAM	00000000 0000...

Figure 4: The waveforms for instructions 22 (st $52, R2) to instruction 30 (ldi r4, 6).



```
phase3
/phase3_tb/clk                      0
/phase3_tb/rst                      0
/phase3_tb/stop                     x
/phase3_tb/bus_contents             0000012d
/phase3_tb/operation                03
/phase3_tb/DUT/R0/q                 00000002
/phase3_tb/DUT/R1/q                 000002f8
/phase3_tb/DUT/R2/q                 000000be
/phase3_tb/DUT/R3/q                 00000002
/phase3_tb/DUT/R4/q                 00000006
/phase3_tb/DUT/R5/q                 00000032
/phase3_tb/DUT/R6/q                 0000012c
/phase3_tb/DUT/R7/q                 00000000
/phase3_tb/DUT/R8/q                 00000005
/phase3_tb/DUT/R9/q                 0000001f
/phase3_tb/DUT/R10/q                0000012c
/phase3_tb/DUT/R11/q                00000000
/phase3_tb/DUT/R12/q                00000000
/phase3_tb/DUT/R13/q                00000131
/phase3_tb/DUT/R14/q                00000000
/phase3_tb/DUT/R15/q                00000028
/phase3_tb/DUT/HI_reg/q             00000002
/phase3_tb/DUT/LO_reg/q             00000008
/phase3_tb/DUT/IR/q                 1ec50000
/phase3_tb/DUT/PC_reg/newPC         0000012d
/phase3_tb/DUT/MDR/q                0000012d
/phase3_tb/DUT/MAR/q                0000012d
/phase3_tb/DUT/ramModule/data_out   264d8000
/phase3_tb/DUT/ramModule/RAM        00000000 0000...
```

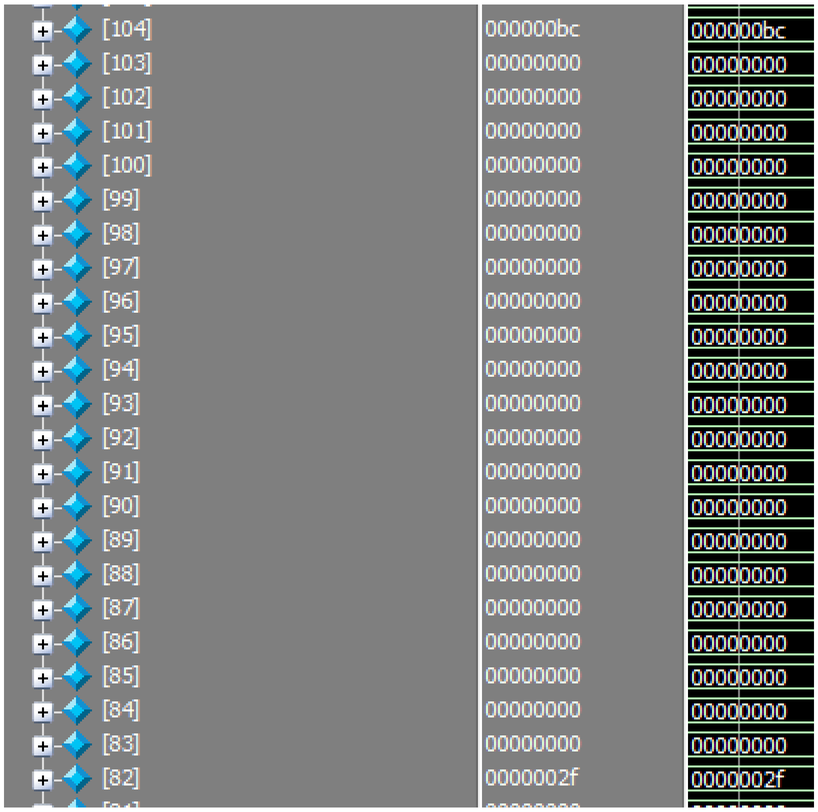
Figure 5: The waveforms for instruction 30 (ldi R5, $32) to instruction 38 (ldi R10, 0(R6)).



phase3	
/phase3_tb/clk	1
/phase3_tb/rst	0
/phase3_tb/stop	x
/phase3_tb/bus_contents	d8000000
/phase3_tb/operation	1b
/phase3_tb/DUT/R0/q	00000002
/phase3_tb/DUT/R1/q	000002f8
/phase3_tb/DUT/R2/q	000000be
/phase3_tb/DUT/R3/q	00000002
/phase3_tb/DUT/R4/q	00000006
/phase3_tb/DUT/R5/q	00000032
/phase3_tb/DUT/R6/q	0000012c
/phase3_tb/DUT/R7/q	00000000
/phase3_tb/DUT/R8/q	00000005
/phase3_tb/DUT/R9/q	0000001f
/phase3_tb/DUT/R10/q	0000012c
/phase3_tb/DUT/R11/q	00000000
/phase3_tb/DUT/R12/q	0000001f
/phase3_tb/DUT/R13/q	00000112
/phase3_tb/DUT/R14/q	00000000
/phase3_tb/DUT/R15/q	00000028
/phase3_tb/DUT/HI_reg/q	00000002
/phase3_tb/DUT/LO_reg/q	00000008
/phase3_tb/DUT/IR/q	d8000000
/phase3_tb/DUT/PC_reg/newPC	00000029
/phase3_tb/DUT/MDR/q	d8000000
/phase3_tb/DUT/MAR/q	00000028
/phase3_tb/DUT/ramModule/data_out	d8000000
/phase3_tb/DUT/ramModule/RAM	00000000 0000...

Figure 6: The waveforms for instruction 38 (ldi R11, 0(R7)) to the last instruction (jr R15). After this instruction 41 (halt) is executed.

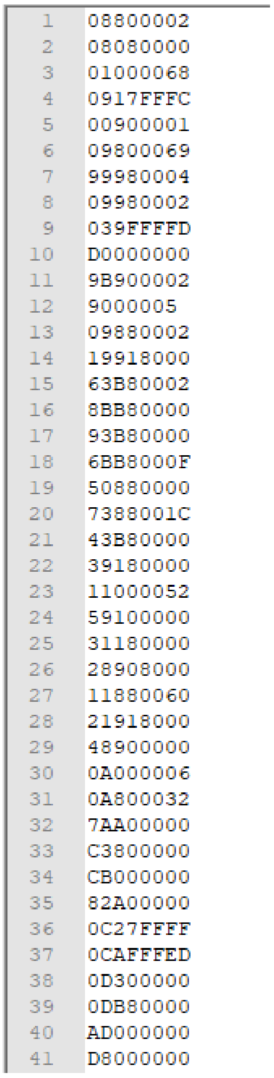
RAM Contents After Execution



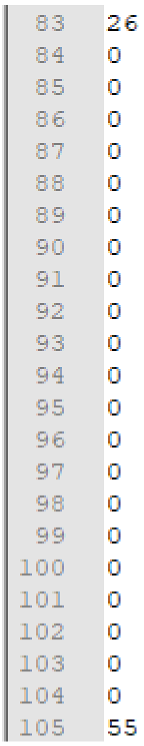
```
[104]    000000bc    000000bc
[103]    00000000    00000000
[102]    00000000    00000000
[101]    00000000    00000000
[100]    00000000    00000000
[99]     00000000    00000000
[98]     00000000    00000000
[97]     00000000    00000000
[96]     00000000    00000000
[95]     00000000    00000000
[94]     00000000    00000000
[93]     00000000    00000000
[92]     00000000    00000000
[91]     00000000    00000000
[90]     00000000    00000000
[89]     00000000    00000000
[88]     00000000    00000000
[87]     00000000    00000000
[86]     00000000    00000000
[85]     00000000    00000000
[84]     00000000    00000000
[83]     00000000    00000000
[82]     0000002f    0000002f
```

Locations are in decimal, so 104 would be hex $68, 82 would be $52, both the locations specified by the store instruction.

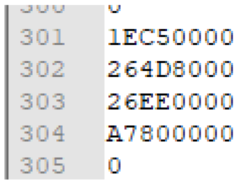
RAM Contents Before



```
1   08800002
2   08080000
3   01000068
4   0917FFFC
5   00900001
6   09800069
7   99980004
8   09980002
9   039FFFFD
10  D0000000
11  9B900002
12  9000005
13  09880002
14  19918000
15  63B80002
16  8BB80000
17  93B80000
18  6BB8000F
19  50880000
20  7388001C
21  43B80000
22  39180000
23  11000052
24  59100000
25  31180000
26  28908000
27  11880060
28  21918000
29  48900000
30  0A000006
31  0A800032
32  7AA00000
33  C3800000
34  CB000000
35  82A00000
36  0C27FFFF
37  0CAFFFED
38  0D300000
39  0DB80000
40  AD000000
41  D8000000
```



```
83   26
84   0
85   0
86   0
87   0
88   0
89   0
90   0
91   0
92   0
93   0
94   0
95   0
96   0
97   0
98   0
99   0
100  0
101  0
102  0
103  0
104  0
105  55
```

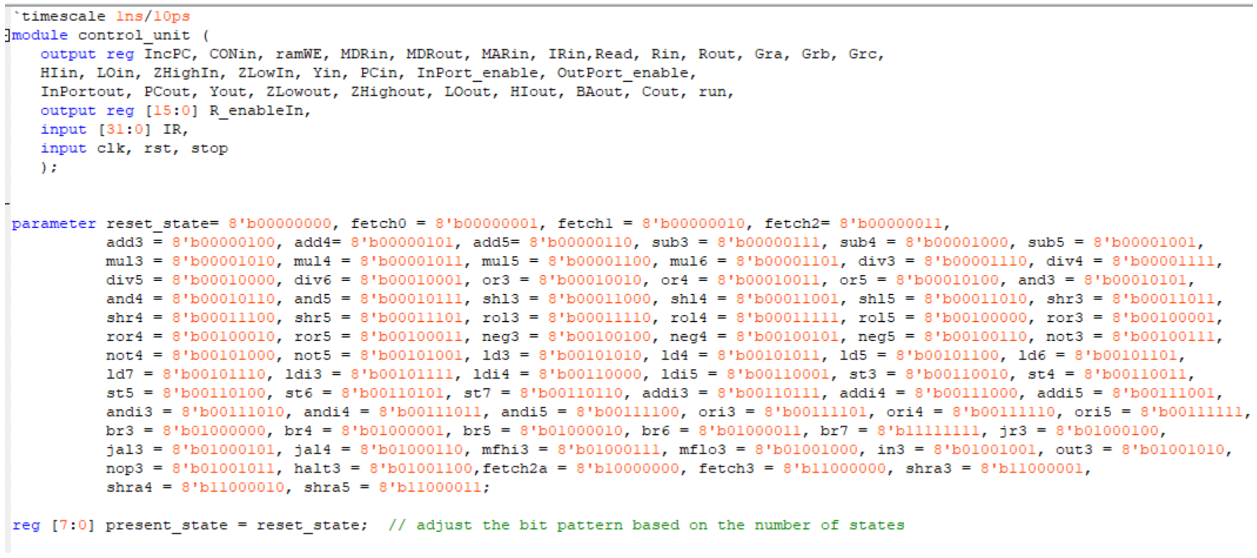


```
300  0
301  1EC50000
302  264D8000
303  26EE0000
304  A7800000
305  0
```

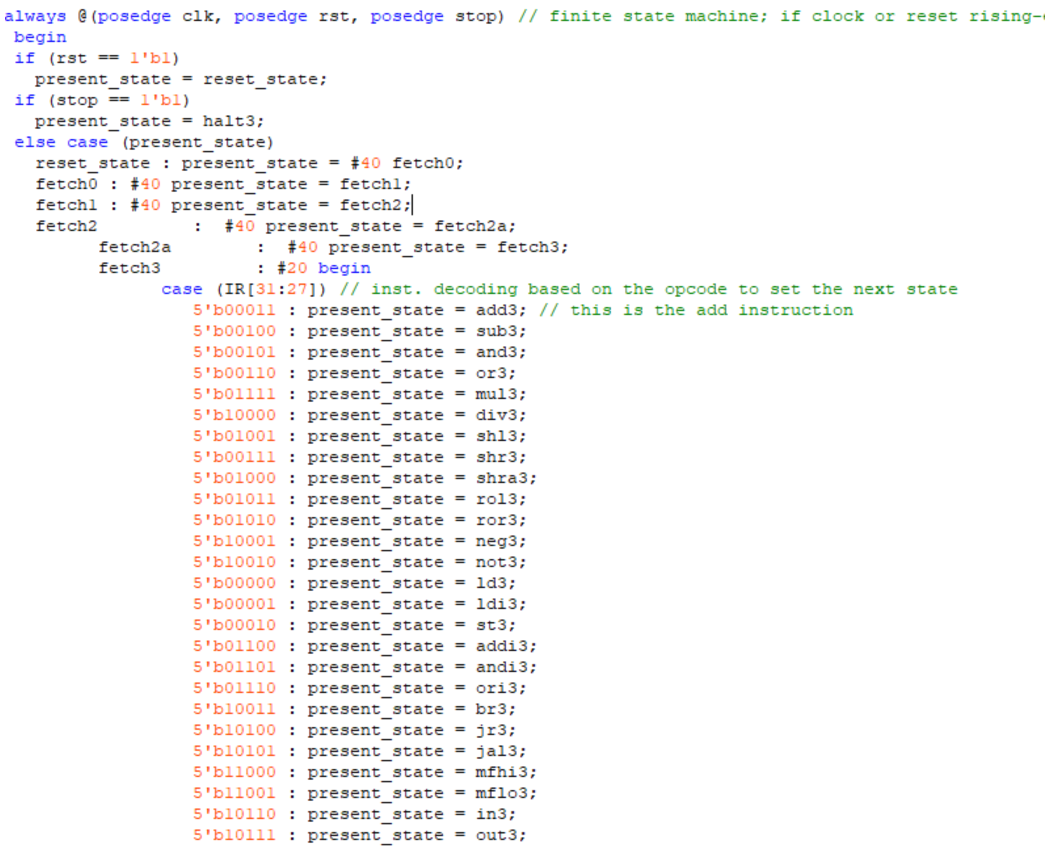
Verilog Code

This section only includes revised or newly added code, refer to phase 1 and 2 reports for other code.

Control Unit



```
`timescale 1ns/10ps
]module control_unit (
    output reg IncPC, CONin, ramWE, MDRin, MDRout, MARin, IRin,Read, Rin, Rout, Gra, Grb, Grc,
    HIin, LOin, ZHighIn, ZLowIn, Yin, PCin, InPort_enable, OutPort_enable,
    InPortout, PCout, Yout, ZLowout, ZHighout, LOout, HIout, BAout, Cout, run,
    output reg [15:0] R_enableIn,
    input [31:0] IR,
    input clk, rst, stop
    );

-
 parameter reset_state= 8'b00000000, fetch0 = 8'b00000001, fetch1 = 8'b00000010, fetch2= 8'b00000011,
           add3 = 8'b00000100, add4= 8'b00000101, add5= 8'b00000110, sub3 = 8'b00000111, sub4 = 8'b00001000, sub5 = 8'b00001001,
           mul3 = 8'b00001010, mul4 = 8'b00001011, mul5 = 8'b00001100, mul6 = 8'b00001101, div3 = 8'b00001110, div4 = 8'b00001111,
           div5 = 8'b00010000, div6 = 8'b00010001, or3 = 8'b00010010, or4 = 8'b00010011, or5 = 8'b00010100, and3 = 8'b00010101,
           and4 = 8'b00010110, and5 = 8'b00010111, shl3 = 8'b00011000, shl4 = 8'b00011001, shl5 = 8'b00011010, shr3 = 8'b00011011,
           shr4 = 8'b00011100, shr5 = 8'b00011101, rol3 = 8'b00011110, rol4 = 8'b00011111, rol5 = 8'b00100000, ror3 = 8'b00100001,
           ror4 = 8'b00100010, ror5 = 8'b00100011, neg3 = 8'b00100100, neg4 = 8'b00100101, neg5 = 8'b00100110, not3 = 8'b00100111,
           not4 = 8'b00101000, not5 = 8'b00101001, ld3 = 8'b00101010, ld4 = 8'b00101011, ld5 = 8'b00101100, ld6 = 8'b00101101,
           ld7 = 8'b00101110, ldi3 = 8'b00101111, ldi4 = 8'b00110000, ldi5 = 8'b00110001, st3 = 8'b00110010, st4 = 8'b00110011,
           st5 = 8'b00110100, st6 = 8'b00110101, st7 = 8'b00110110, addi3 = 8'b00110111, addi4 = 8'b00111000, addi5 = 8'b00111001,
           andi3 = 8'b00111010, andi4 = 8'b00111011, andi5 = 8'b00111100, ori3 = 8'b00111101, ori4 = 8'b00111110, ori5 = 8'b00111111,
           br3 = 8'b01000000, br4 = 8'b01000001, br5 = 8'b01000010, br6 = 8'b01000011, br7 = 8'b11111111, jr3 = 8'b01000100,
           jal3 = 8'b01000101, jal4 = 8'b01000110, mfhi3 = 8'b01000111, mflo3 = 8'b01001000, in3 = 8'b01001001, out3 = 8'b01001010,
           nop3 = 8'b01001011, halt3 = 8'b01001100,fetch2a = 8'b10000000, fetch3 = 8'b11000000, shra3 = 8'b11000001,
           shra4 = 8'b11000010, shra5 = 8'b11000011;

 reg [7:0] present_state = reset_state;  // adjust the bit pattern based on the number of states
```



```
always @(posedge clk, posedge rst, posedge stop) // finite state machine; if clock or reset rising-
 begin
 if (rst == 1'b1)
   present_state = reset_state;
 if (stop == 1'b1)
   present_state = halt3;
 else case (present_state)
   reset_state : present_state = #40 fetch0;
   fetch0 : #40 present_state = fetch1;
   fetch1 : #40 present_state = fetch2;
   fetch2          :  #40 present_state = fetch2a;
         fetch2a         :  #40 present_state = fetch3;
         fetch3          : #20 begin
                case (IR[31:27]) // inst. decoding based on the opcode to set the next state
                    5'b00011 : present_state = add3; // this is the add instruction
                    5'b00100 : present_state = sub3;
                    5'b00101 : present_state = and3;
                    5'b00110 : present_state = or3;
                    5'b01111 : present_state = mul3;
                    5'b10000 : present_state = div3;
                    5'b01001 : present_state = shl3;
                    5'b00111 : present_state = shr3;
                    5'b01000 : present_state = shra3;
                    5'b01011 : present_state = rol3;
                    5'b01010 : present_state = ror3;
                    5'b10001 : present_state = neg3;
                    5'b10010 : present_state = not3;
                    5'b00000 : present_state = ld3;
                    5'b00001 : present_state = ldi3;
                    5'b00010 : present_state = st3;
                    5'b01100 : present_state = addi3;
                    5'b01101 : present_state = andi3;
                    5'b01110 : present_state = ori3;
                    5'b10011 : present_state = br3;
                    5'b10100 : present_state = jr3;
                    5'b10101 : present_state = jal3;
                    5'b11000 : present_state = mfhi3;
                    5'b11001 : present_state = mflo3;
                    5'b10110 : present_state = in3;
                    5'b10111 : present_state = out3;
```

```
                  5'b10111 : present_state = out3;
                  5'b11010 : present_state = nop3;
                  5'b11011 : present_state = halt3;
               endcase
            end
         add3            :  #40 present_state = add4;
         add4            :  #40 present_state = add5;
         add5            :  #40 present_state = reset_state;

         addi3           :  #40 present_state = addi4;
         addi4           :  #40 present_state = addi5;
         addi5           :  #40 present_state = reset_state;

         sub3            :  #40 present_state = sub4;
         sub4            :  #40 present_state = sub5;
         sub5            :  #40 present_state = reset_state;

         mul3            :  #40 present_state = mul4;
         mul4            :  #40 present_state = mul5;
         mul5            :  #40 present_state = mul6;
         mul6            :  #40 present_state = reset_state;

         div3            :  #40 present_state = div4;
         div4            :  #40 present_state = div5;
         div5            :  #40 present_state = div6;
         div6            :  #40 present_state = reset_state;

         or3             :  #40 present_state = or4;
         or4             :  #40 present_state = or5;
         or5             :  #40 present_state = reset_state;

         and3            :  #40 present_state = and4;
         and4            :  #40 present_state = and5;
         and5            :  #40 present_state = reset_state;

         shl3            :  #40 present_state = shl4;
         shl4            :  #40 present_state = shl5;
         shl5            :  #40 present_state = reset_state;
shr3            :  #40 present_state = shr4;
shr4            :  #40 present_state = shr5;
shr5            :  #40 present_state = reset_state;

shra3           :  #40 present_state = shra4;
shra4           :  #40 present_state = shra5;
shra5              :  #40 present_state = reset_state;

rol3            :  #40 present_state = rol4;
rol4            :  #40 present_state = rol5;
rol5            :  #40 present_state = reset_state;

ror3            :  #40 present_state = ror4;
ror4            :  #40 present_state = ror5;
ror5            :  #40 present_state = reset_state;

neg3            :  #40 present_state = neg4;
neg4            :  #40 present_state = reset_state;

not3            :  #40 present_state = not4;
not4            :  #40 present_state = reset_state;

ld3             :  #40 present_state = ld4;
ld4             :  #40 present_state = ld5;
ld5             :  #40 present_state = ld6;
ld6             :  #40 present_state = ld7;
ld7             :  #40  present_state = reset_state;

ldi3            :  #40 present_state = ldi4;
ldi4            :  #40 present_state = ldi5;
ldi5            :  #40 present_state = reset_state;

st3             :  #40 present_state = st4;
st4             :  #40 present_state = st5;
st5             :  #40 present_state = st6;
st6             :  #40 present_state = st7;
st7             :  #40 present_state = reset_state;
```

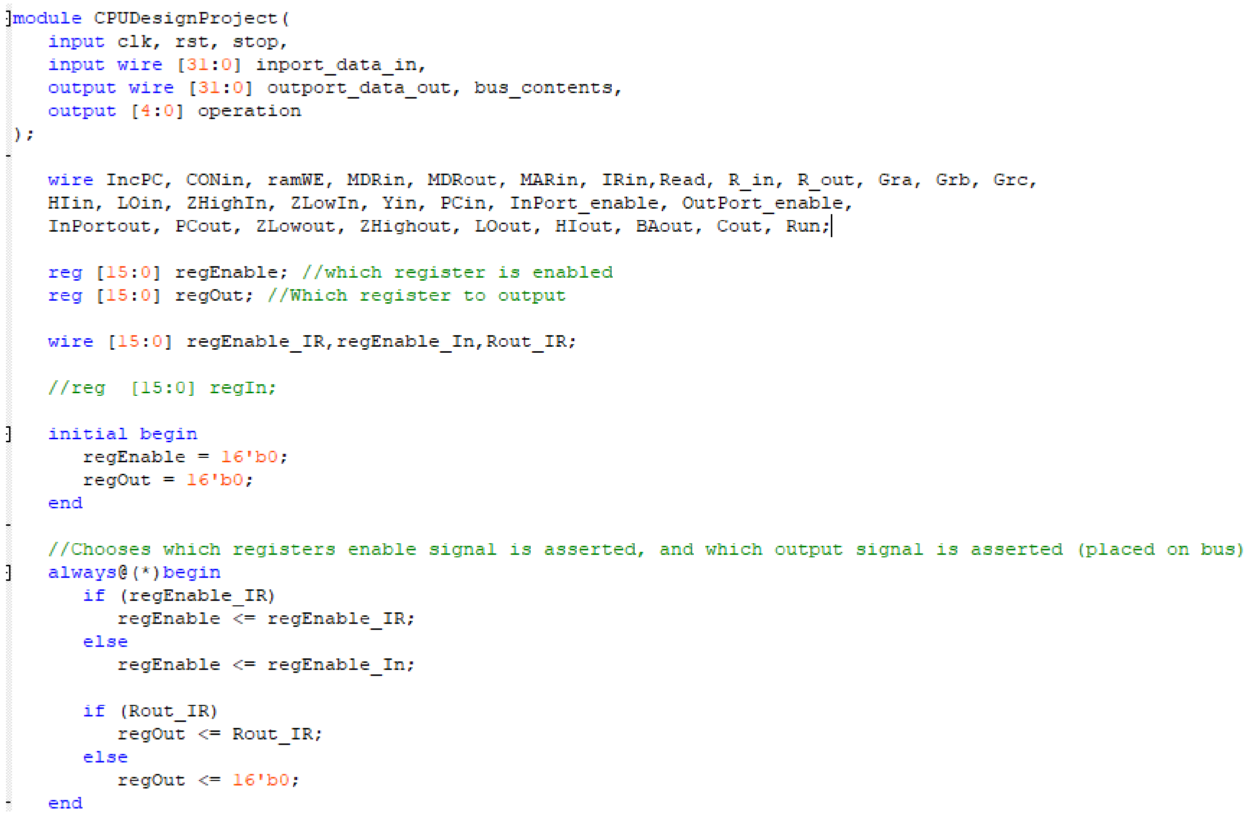
```
 andi3          :  #40 present_state = andi4;
 andi4          :  #40 present_state = andi5;
 andi5          :  #40 present_state = reset_state;

 ori3           :  #40 present_state = ori4;
 ori4           :  #40 present_state = ori5;
 ori5           :  #40 present_state = reset_state;

 jal3           :  #40 present_state = jal4;
 jal4           :  #40 present_state = reset_state;

 jr3            :  #40 present_state = reset_state;

 br3            :  #40 present_state = br4;
 br4            :  #40 present_state = br5;
 br5            :  #40 present_state = br6;
 br6            :  #40 present_state = br7;
 br7            :  #40 present_state = reset_state;

 out3           :  #40 present_state = reset_state;

 in3            :  #40 present_state = reset_state;

 mflo3          :  #40 present_state = reset_state;

 mfhi3          :  #40 present_state = reset_state;

 nop3           :  #40 present_state = reset_state;
always @(present_state) begin // do the job for each state
   case (present_state) // assert the required signals in each state
      reset_state: begin
         run <= 1;
         R_enableIn <= 0;
         PCout <= 0; ZLowout <= 0; MDRout <= 0;
            MARin <= 0; ZHighIn <= 0; ZLowIn <= 0; CONin<=0;
            InPort_enable<=0; OutPort_enable<=0;
            PCin <=0; MDRin <= 0; IRin <= 0;
            Yin <= 0; IncPC <= 0; ramWE <=0;
            Gra<=0; Grb<=0; Grc<=0; BAout<=0; Cout<=0;
            InPortout<=0; ZHighout<=0; LOout<=0; HIout<=0;
            HIin<=0; LOin<=0; Rout<=0; Rin<=0; Read<=0;

      end
      fetch0: begin
            #5 PCout <= 1; MARin <= 1;
      end
      fetch1: begin
            PCout <= 0; MARin <= 0; ZLowIn <= 0;
            Read <= 1; MDRin <= 1;
      end
      fetch2: begin
            ZLowout <= 0; Read <= 0; MDRin <= 0;
            #5 MDRout <= 1; IRin <= 1;
      end
      fetch2a : begin

      end
      fetch3 : begin
            MDRout <= 0; IRin <= 0;
            PCin <= 1; IncPC <= 1;
      end
      //END OF FETCH
```

```
//MUL AND DIVIDE
mul3, div3 : begin
     PCin <= 0; IncPC <= 0;
    Gra <= 1; Rout <= 1;Yin <= 1;
end
mul4, div4: begin
    Gra <= 0; Rout <= 0;Yin <= 0;
    Grb <= 1; Rout <= 1; ZLowIn <= 1; ZHighIn <= 1;
end
mul5, div5: begin
    Grb <= 0; Rout <= 0; ZLowIn <= 0; ZHighIn <= 0;
    ZLowout<= 1; LOin <= 1;
end
mul6, div6: begin
    ZLowout<= 0; LOin <= 0;
    ZHighout <= 1; HIin <= 1;
    //#40 Zhighout <= 0; HIin <= 0;
end
//Intermediate Inst
andi3,ori3,addi3: begin
      PCin <= 0; IncPC <= 0;
     Grb <= 1; Rout <= 1; Yin <= 1;
end
andi4,ori4,addi4: begin
     Grb <= 0; Rout <= 0; Yin <= 0;
     Cout <= 1; ZHighIn <= 1; ZLowIn <= 1;
end
andi5,ori5,addi5: begin
      Cout <= 0; ZHighIn <= 0; ZLowIn <= 0;
      ZLowout<= 1; Gra <= 1; Rin <= 1;
      //#40 ZLowout<= 0; Gra <= 0; Rin <= 0;
end

//Not Neg
not3,neg3: begin
      PCin <= 0; IncPC <= 0;
      Grb <= 1; Rout <= 1; ZLowIn <= 1;
end
not4,neg4: begin
      Grb <= 0; Rout <= 0; ZLowIn <= 10;
      ZLowout<= 1; Gra <= 1; Rin <= 1;
end
//LOAD
ld3: begin
       PCin <= 0; IncPC <= 0;
      Grb <= 1; BAout <= 1; Yin <= 1;
end
ld4: begin
      Grb <= 0; BAout <= 0; Yin <= 0;
      Cout <= 1; ZHighIn <= 1; ZLowIn <= 1;
end
ld5: begin
      Cout <= 0; ZHighIn <= 0; ZLowIn <= 0;
      ZLowout<= 1; MARin <= 1;
end
ld6: begin
      ZLowout<= 0; MARin <= 0;
      Read <= 1; MDRin <= 1;
end
ld7: begin
      Read <= 0; MDRin <= 0;
      MDRout <= 1; Gra <= 1; Rin <= 1;
      //#40 MDRout <=0; Gra<=0; Rin<=0;
end
```

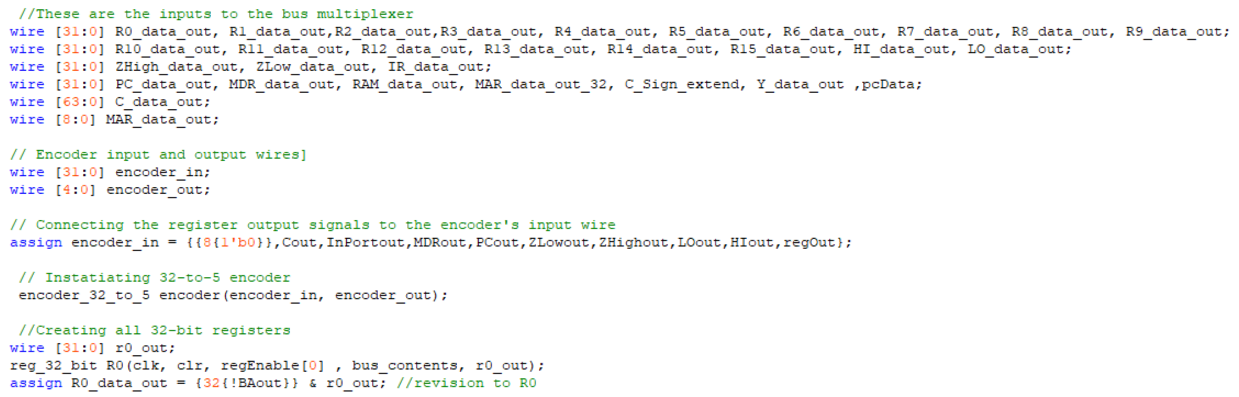
```
//Load Intermediate
ldi3: begin
      PCin <= 0; IncPC <= 0;
      Grb <= 1; BAout <= 1; Yin <= 1;
end
ldi4: begin
      Grb <= 0; BAout <= 0; Yin <= 0;
      Cout <= 1; ZHighIn <= 1; ZLowIn <= 1;
end
ldi5: begin
      Cout <= 0; ZHighIn <= 0; ZLowIn <= 0;
      ZLowout<= 1; Gra <= 1; Rin <= 1;
end
//Store
st3: begin
      PCin <= 0; IncPC <= 0;
     Grb <= 1; BAout <= 1; Yin <= 1;
end
st4: begin
     Grb <= 0; BAout <= 0; Yin <= 0;
     Cout <= 1; ZHighIn <= 1; ZLowIn <= 1;
end
st5: begin
     Cout <= 0; ZHighIn <= 0; ZLowIn <= 0;
     ZLowout<= 1; MARin <= 1;
end
st6: begin
     ZLowout<= 0; MARin <= 0;
     Read <= 0; Gra <= 1; Rout <= 1;  MDRin <= 1;
end
st7: begin
     Gra <= 0; Rout <= 0;  MDRin <= 0; MDRout <= 1;
     ramWE <= 1;
end

//Jump
jr3: begin
     PCin <= 0; IncPC <= 0;
     Gra <= 1; Rout <= 1; PCin <= 1;
end
//Jump and Link
jal3: begin
      PCin <= 0; IncPC <= 0;
      PCout <= 1; R_enableIn <= 16'h8000;
end
jal4: begin
     PCout <= 0; R_enableIn <= 16'h0;
     Gra <= 1; Rout <= 1; PCin <= 1;
end
//Outport
out3: begin
      PCin <= 0; IncPC <= 0;
     Gra <= 1; Rout <= 1; OutPort_enable <= 1;
end
//Inport
in3: begin
     PCin <= 0; IncPC <= 0;
     Gra <= 1; Rin <= 1; InPortout <= 1;
end
//Move from HI
mfhi3: begin
      PCin <= 0; IncPC <= 0;
     Gra <= 1; Rin <= 1; HIout <= 1;
end
//Move from LO
mflo3: begin
      PCin <= 0; IncPC <= 0;
     Gra <= 1; Rin <= 1; LOout <= 1;
end
```

```
//Branch
br3: begin
     PCin <= 0; IncPC <= 0;
     Gra <= 1; Rout <= 1; CONin <= 1;
end
br4: begin
     Gra <= 0; Rout <= 0; CONin <= 0;
     PCout <= 1; Yin <= 1;
end
br5: begin
     PCout <= 0; Yin <= 0;
     Cout <= 1; ZLowIn <= 1; ZHighIn <= 1;
end
br6: begin
     Cout <= 0; ZLowIn<= 0; ZHighIn <= 0;
     ZLowout <= 1; PCin <= 1;
end
br7: begin
     ZLowout<=0; PCin<=0;
     PCout<=1; MARin <= 1;
end
//No OP
nop3 : begin
   PCin <= 0; IncPC <= 0;
end
//Halt
halt3: begin
  PCin <= 0; IncPC <= 0;
  run <= 0;
end
```

Datapath

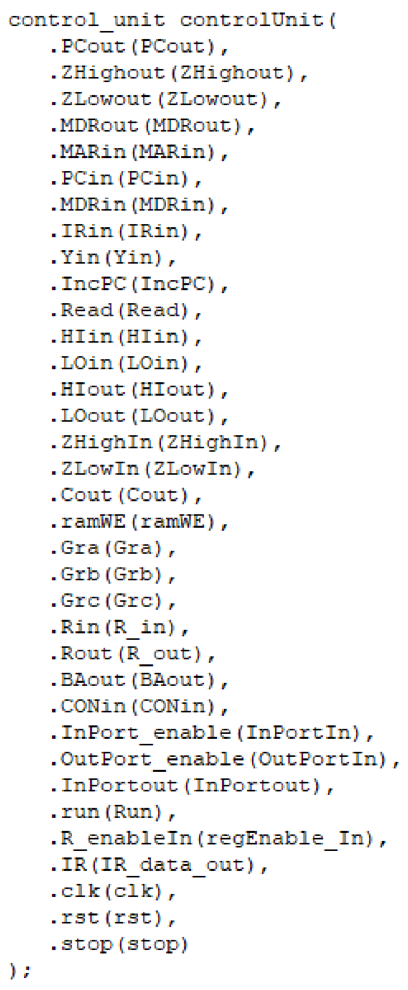


```
module CPUDesignProject(
   input clk, rst, stop,
   input wire [31:0] inport_data_in,
   output wire [31:0] outport_data_out, bus_contents,
   output [4:0] operation
);

   wire IncPC, CONin, ramWE, MDRin, MDRout, MARin, IRin,Read, R_in, R_out, Gra, Grb, Grc,
   HIin, LOin, ZHighIn, ZLowIn, Yin, PCin, InPort_enable, OutPort_enable,
   InPortout, PCout, ZLowout, ZHighout, LOout, HIout, BAout, Cout, Run;

   reg [15:0] regEnable; //which register is enabled
   reg [15:0] regOut; //Which register to output

   wire [15:0] regEnable_IR,regEnable_In,Rout_IR;

   //reg  [15:0] regIn;

   initial begin
      regEnable = 16'b0;
      regOut = 16'b0;
   end

   //Chooses which registers enable signal is asserted, and which output signal is asserted (placed on bus)
   always@(*)begin
      if (regEnable_IR)
         regEnable <= regEnable_IR;
      else
         regEnable <= regEnable_In;

      if (Rout_IR)
         regOut <= Rout_IR;
      else
         regOut <= 16'b0;
   end
```

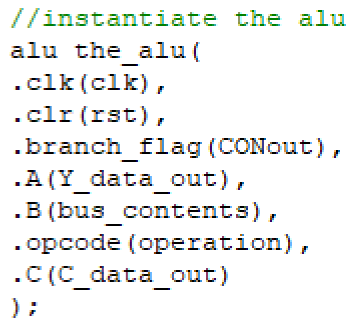


```
 //These are the inputs to the bus multiplexer
wire [31:0] R0_data_out, R1_data_out,R2_data_out,R3_data_out, R4_data_out, R5_data_out, R6_data_out, R7_data_out, R8_data_out, R9_data_out;
wire [31:0] R10_data_out, R11_data_out, R12_data_out, R13_data_out, R14_data_out, R15_data_out, HI_data_out, LO_data_out;
wire [31:0] ZHigh_data_out, ZLow_data_out, IR_data_out;
wire [31:0] PC_data_out, MDR_data_out, RAM_data_out, MAR_data_out_32, C_Sign_extend, Y_data_out ,pcData;
wire [63:0] C_data_out;
wire [8:0] MAR_data_out;

// Encoder input and output wires]
wire [31:0] encoder_in;
wire [4:0] encoder_out;

// Connecting the register output signals to the encoder's input wire
assign encoder_in = {{8{1'b0}},Cout,InPortout,MDRout,PCout,ZLowout,ZHighout,LOout,HIout,regOut};

 // Instatiating 32-to-5 encoder
 encoder_32_to_5 encoder(encoder_in, encoder_out);

 //Creating all 32-bit registers
wire [31:0] r0_out;
reg_32_bit R0(clk, clr, regEnable[0] , bus_contents, r0_out);
assign R0_data_out = {32{!BAout}} & r0_out; //revision to R0
```

```verilog
//Registers
reg_32_bit R1(clk, clr, regEnable[1], bus_contents, R1_data_out);
reg_32_bit R2(clk, clr, regEnable[2], bus_contents, R2_data_out);
reg_32_bit R3(clk, clr, regEnable[3], bus_contents, R3_data_out);
reg_32_bit R4(clk, clr, regEnable[4], bus_contents, R4_data_out);
reg_32_bit R5(clk, clr, regEnable[5], bus_contents, R5_data_out);
reg_32_bit R6(clk, clr, regEnable[6], bus_contents, R6_data_out);
reg_32_bit R7(clk, clr, regEnable[7], bus_contents, R7_data_out);
reg_32_bit R8(clk, clr, regEnable[8], bus_contents, R8_data_out);
reg_32_bit R9(clk, clr, regEnable[9], bus_contents, R9_data_out);
reg_32_bit R10(clk, clr, regEnable[10], bus_contents, R10_data_out);
reg_32_bit R11(clk, clr, regEnable[11], bus_contents, R11_data_out);
reg_32_bit R12(clk, clr, regEnable[12], bus_contents, R12_data_out);
reg_32_bit R13(clk, clr, regEnable[13], bus_contents, R13_data_out);
reg_32_bit R14(clk, clr, regEnable[14], bus_contents, R14_data_out);
reg_32_bit R15(clk, clr, regEnable[15], bus_contents, R15_data_out);
reg_32_bit Y(clk, clr, Yin, bus_contents, Y_data_out);
reg_32_bit HI_reg(clk, clr, HIin, bus_contents, HI_data_out);
reg_32_bit LO_reg(clk, clr, LOin, bus_contents, LO_data_out);
reg_32_bit ZHigh_reg(clk, clr, ZHighIn, C_data_out[63:32], ZHigh_data_out);
reg_32_bit ZLow_reg(clk, clr, ZLowIn, C_data_out[31:0], ZLow_data_out);

reg_32_bit IR(clk, rst, IRin, bus_contents, IR_data_out);

IncPC_32_bit PC_reg(clk, IncPC, PCin, bus_contents, PC_data_out);

reg_32_bit OutPort(clk, clr, OutPortIn, bus_contents, outport_data_out);
reg_32_bit InPort(clk, clr, InPortIn, Inport_data_out, BusMuxIn_In_Port);

//Select and encode Logic and CON FF
selectencodelogic selEn(IR_data_out, Gra, Grb, Grc, R_in, R_out, BAout, C_Sign_extend,
                  regEnable_IR, Rout_IR, operation);
CONFF_logic conff(IR_data_out[20:19],bus_contents, CONin, CONout);

//MDR
wire [31:0] MDR_mux_out;
//First create the 2-1 mux that selects either the RAM or the bus contents
mux_2_to_1 MDMux(bus_contents,RAM_data_out, Read,MDR_mux_out);
//Create the actual MDR itself by instantiating a regular 32 bit reg
reg_32_bit MDR(clk,rst,MDRin,MDR_mux_out,MDR_data_out);

//This is done to avoid having to make an MAR unit module
reg_32_bit MAR(clk,rst,MARin, bus_contents, MAR_data_out_32);
assign MAR_data_out = MAR_data_out_32[8:0];

//memRAM ramModule(MAR_data_out,clk,MDR_data_out,Read,ramWE,RAM_data_out);
ram ramModule(MDR_data_out,MAR_data_out,clk,ramWE,RAM_data_out);

// Multiplexer to select which data to send out on the bus
mux_32_to_1 BusMux(
   .BusMuxIn_R0(R0_data_out),.BusMuxIn_R1(R1_data_out), .BusMuxIn_R2(R2_data_out),.BusMuxIn_R3(R3_data_out),
   .BusMuxIn_R4(R4_data_out),.BusMuxIn_R5(R5_data_out), .BusMuxIn_R6(R6_data_out),.BusMuxIn_R7(R7_data_out),
   .BusMuxIn_R8(R8_data_out),.BusMuxIn_R9(R9_data_out),.BusMuxIn_R10(R10_data_out),.BusMuxIn_R11(R11_data_out),
   .BusMuxIn_R12(R12_data_out),.BusMuxIn_R13(R13_data_out),.BusMuxIn_R14(R14_data_out),.BusMuxIn_R15(R15_data_out),
   .BusMuxIn_HI(HI_data_out),.BusMuxIn_LO(LO_data_out),.BusMuxIn_Z_HI(ZHigh_data_out),.BusMuxIn_Z_LO(ZLow_data_out),
   .BusMuxIn_PC(PC_data_out),.BusMuxIn_MDR(MDR_data_out),.BusMuxIn_In_Port(Inport_data_out),.C_Sign_Extended(C_Sign_extend),
   .BusMuxOut(bus_contents),.select(encoder_out)
   );
```

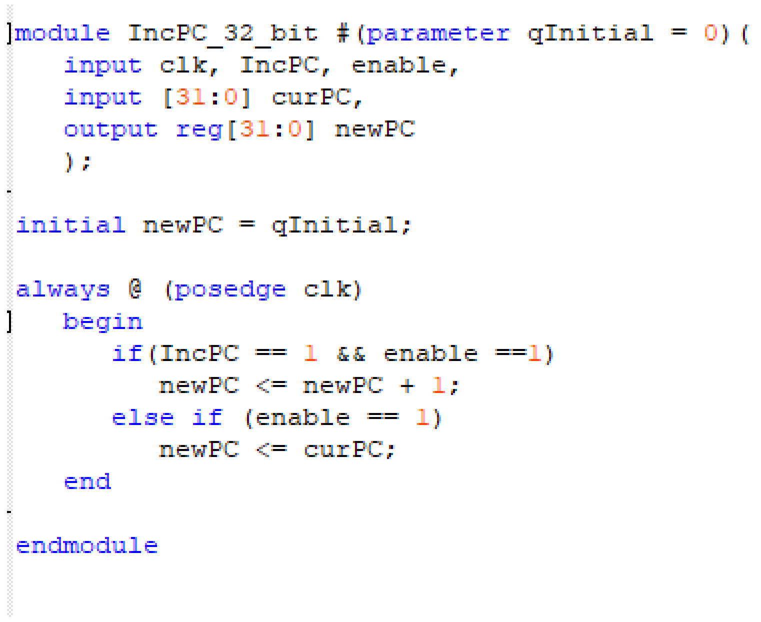


```
control_unit controlUnit(
    .PCout(PCout),
    .ZHighout(ZHighout),
    .ZLowout(ZLowout),
    .MDRout(MDRout),
    .MARin(MARin),
    .PCin(PCin),
    .MDRin(MDRin),
    .IRin(IRin),
    .Yin(Yin),
    .IncPC(IncPC),
    .Read(Read),
    .HIin(HIin),
    .LOin(LOin),
    .HIout(HIout),
    .LOout(LOout),
    .ZHighIn(ZHighIn),
    .ZLowIn(ZLowIn),
    .Cout(Cout),
    .ramWE(ramWE),
    .Gra(Gra),
    .Grb(Grb),
    .Grc(Grc),
    .Rin(R_in),
    .Rout(R_out),
    .BAout(BAout),
    .CONin(CONin),
    .InPort_enable(InPortIn),
    .OutPort_enable(OutPortIn),
    .InPortout(InPortout),
    .run(Run),
    .R_enableIn(regEnable_In),
    .IR(IR_data_out),
    .clk(clk),
    .rst(rst),
    .stop(stop)
);
```



```
//instantiate the alu
alu the_alu(
.clk(clk),
.clr(rst),
.branch_flag(CONout),
.A(Y_data_out),
.B(bus_contents),
.opcode(operation),
.C(C_data_out)
);
```

Increment PC



```
module IncPC_32_bit #(parameter qInitial = 0)(
    input clk, IncPC, enable,
    input [31:0] curPC,
    output reg[31:0] newPC
    );

initial newPC = qInitial;

always @ (posedge clk)
    begin
        if(IncPC == 1 && enable ==1)
            newPC <= newPC + 1;
        else if (enable == 1)
            newPC <= curPC;
    end

endmodule
```