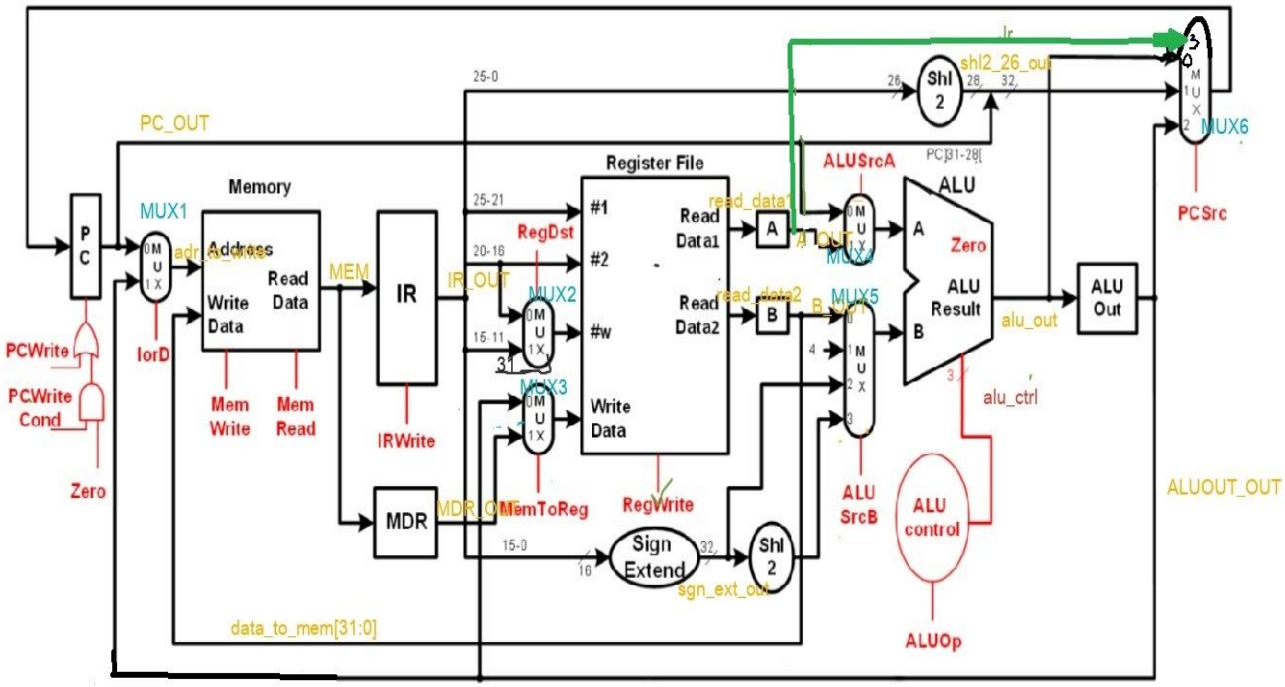
Computer Assignment #3 (Multi Cycle MIPS Processor)

Helia Hosseini (810197491)
Daneshvar Amrollahi (810197685)

Datapath:

Block Diagram:



Verilog Code:

```
module datapath (clk,
                 rst,
                 mem,
                 data_to_mem,
                 reg_dst,
                 mem_to_reg,
                 alu_src_a,
                 alu_src_b,
                 pc_src,
                 alu_ctrl,
                 reg_write,
                 zero_out,
                 zero_in,
                 adr_to_write,
                 IorD,
                 pc_write,
                 pc_write_cond,
                 ir_write,
                 inst);

   input  clk, rst;
   output [31:0] data_to_mem;
   input  [31:0] mem;
   input [1:0] reg_dst;
   input mem_to_reg, alu_src_a, reg_write, ir_write, IorD, pc_write,
pc_write_cond, zero_in, zero_out;
   input [1:0] alu_src_b;
   input [1:0] pc_src;
   input  [2:0] alu_ctrl;
   output [31:0] adr_to_write;
   output [31:0] inst;

   wire [31:0] pc_out;
   wire [31:0] read_data1, read_data2;
   wire [31:0] sgn_ext_out;
   wire [4:0] mux2_out;
   wire [31:0] alu_out;
   wire [31:0] shl2_32_out;
   wire [31:0] mux3_out;
   wire [31:0] mux4_out;
   wire [31:0] mux1_out;
```

```
   wire [31:0] mux5_out;
   wire [27:0] shl2_26_out;
   wire [31:0] mux6_out;
   wire [31:0] ir_out;
   wire [31:0] mdr_out;
   wire [31:0] a_out;
   wire [31:0] b_out;

   wire [31:0] aluout_out;

   reg_32b PC(mux6_out, rst, (zero_in & pc_write_cond) | pc_write, clk,
pc_out);

   mux2to1_32b MUX1(pc_out, aluout_out, IorD, mux1_out);

   reg_32b IR(mem, rst, ir_write, clk, ir_out);

   reg_32b MDR(mem, rst, 1'b1, clk, mdr_out);

   mux3to1_5b MUX2(ir_out[20:16], ir_out[15:11], 5'b11111, reg_dst, mux2_out);

   mux2to1_32b MUX3(aluout_out, mdr_out, mem_to_reg, mux3_out);

   reg_file RF(mux3_out, ir_out[25:21], ir_out[20:16], mux2_out, reg_write,
rst, clk, read_data1, read_data2);

   sign_ext SGN_EXT(ir_out[15:0], sgn_ext_out);
   shl2_32b SHL2_32B(sgn_ext_out, shl2_32_out);

   reg_32b A(read_data1, rst, 1'b1, clk, a_out);
   reg_32b B(read_data2, rst, 1'b1, clk, b_out);

   mux2to1_32b MUX4(pc_out, a_out, alu_src_a, mux4_out);

   mux4to1_32b MUX5(b_out, 32'd4, sgn_ext_out, shl2_32_out, alu_src_b,
mux5_out);

   alu ALU(mux4_out, mux5_out, alu_ctrl, alu_out, zero_out);

   shl2_26b SHL2_26B(ir_out[25:0], shl2_26_out);

   reg_32b ALUOUT(alu_out, rst, 1'b1, clk, aluout_out);

   mux4to1_32b MUX6(alu_out, {pc_out[31:28], shl2_26_out}, aluout_out, a_out,
```

```
pc_src, mux6_out);

   assign adr_to_write = mux1_out;

   assign data_to_mem = b_out;

   assign inst = ir_out;
endmodule
```

Instruction Formats:

Set Less Than Immediate:

```
SLTi  Rt  Rs  data
```

Opcode	R_s	R_t	data
001010	[25:21]	[20:16]	[15:0]

Jump and Link:

```
Jal adr
```

Opcode	adr
000011	[25:0]

Jump Register:

```
Jr Rs
```

Opcode	R_s	X
000110	[25:21]	X

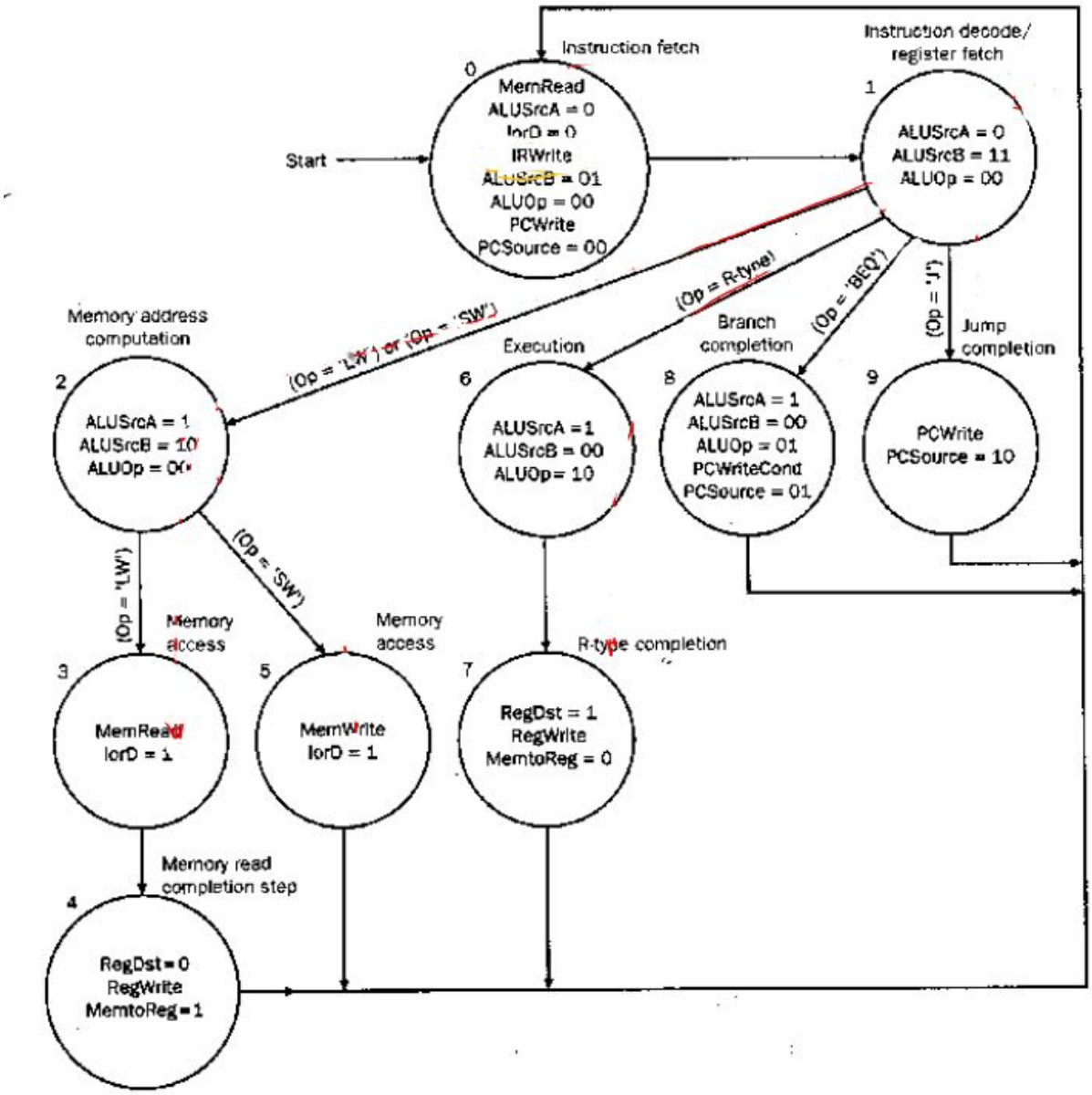
Jump:

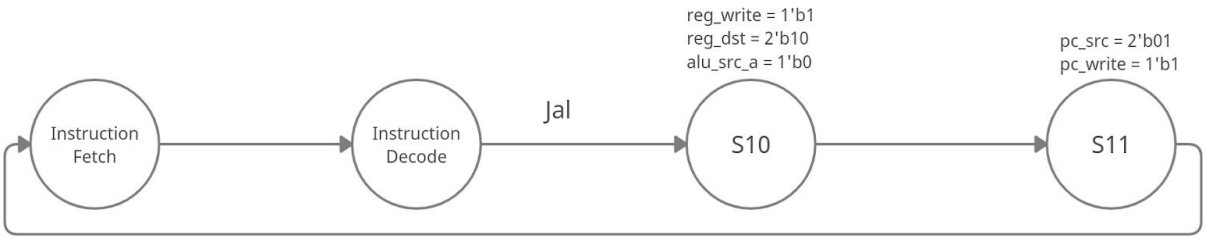
```
J adr
```

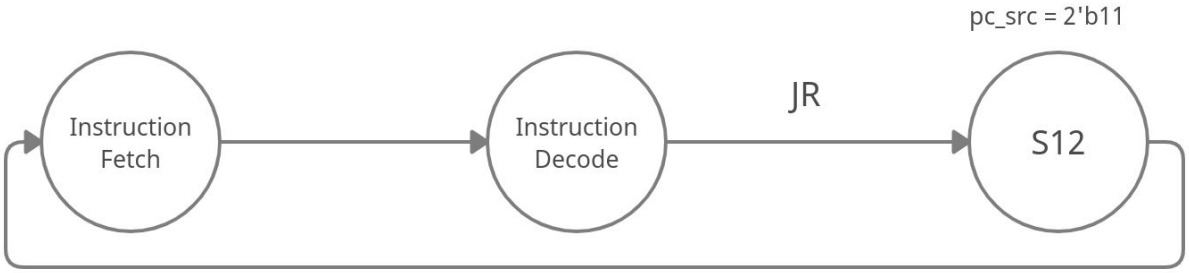
Opcode	adr
000010	[25:0]

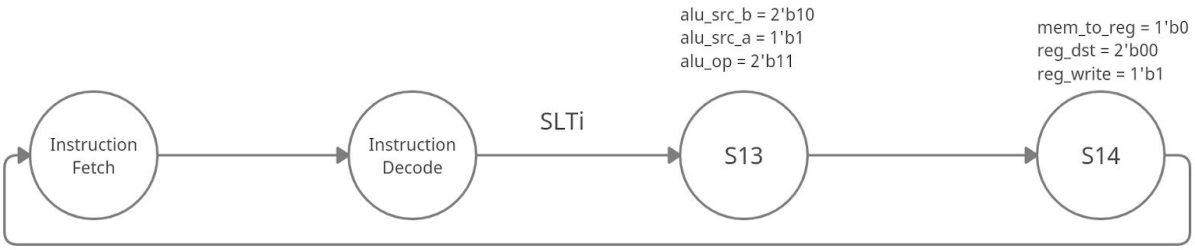
Controller:

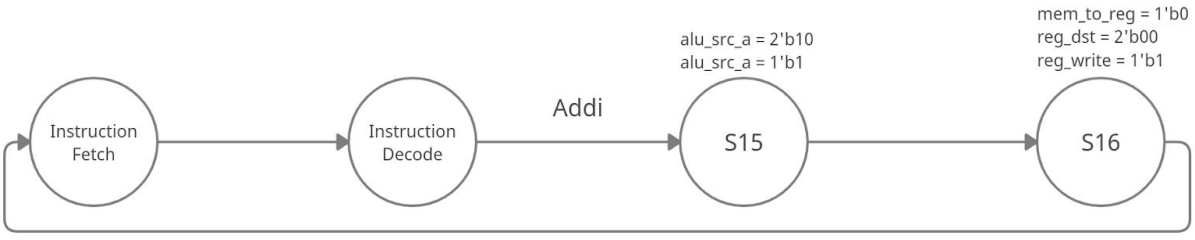
The controller can be shown as the union of FSMs below:




reg_write = 1'b1
reg_dst = 2'b10
alu_src_a = 1'b0
pc_src = 2'b01
pc_write = 1'b1
Instruction Fetch
Instruction Decode
Jal
S10
S11


pc_src = 2'b11
Instruction Fetch
Instruction Decode
JR
S12


alu_src_b = 2'b10
alu_src_a = 1'b1
alu_op = 2'b11
mem_to_reg = 1'b0
reg_dst = 2'b00
reg_write = 1'b1
Instruction Fetch
Instruction Decode
SLTi
S13
S14


alu_src_a = 2'b10
alu_src_a = 1'b1
mem_to_reg = 1'b0
reg_dst = 2'b00
reg_write = 1'b1
Instruction Fetch
Instruction Decode
Addi
S15
S16

Verilog Code:

The controller is implemented using huffman coding style:

```
`define   S0      5'b00000
`define   S1      5'b00001
`define   S2      5'b00010
`define   S3      5'b00011
`define   S4      5'b00100
`define   S5      5'b00101
`define   S6      5'b00110
`define   S7      5'b00111
`define   S8      5'b01000
`define   S9      5'b01001
`define   S10     5'b01010
`define   S11     5'b01011
`define   S12     5'b01100
`define   S13     5'b01101
`define   S14     5'b01110
`define   S15     5'b01111
`define   S16     5'b10000

`define J       6'b000010
`define JAL     6'b000011
`define SLTI    6'b001010
`define JR      6'b000110
`define ADDI    6'b001001
`define BEQ     6'b000100
`define RTYPE   6'b000000
`define LW      6'b100011
`define SW      6'b101011


module controller (opcode,
                   func,
                   zero_out,
                   zero_in,
                   reg_dst,
                   mem_to_reg,
                   reg_write,
                   mem_read,
                   mem_write,
                   pc_src,
```

```
                operation,
                pc_write,
                pc_write_cond,
                IorD,
                ir_write,
                alu_src_a,
                alu_src_b,
                clk,
                rst);

output zero_out;
input zero_in, clk, rst;
output reg pc_write, pc_write_cond, IorD, ir_write, alu_src_a;
output reg [1:0] alu_src_b;

input [5:0] opcode;
input [5:0] func;
output  reg mem_to_reg, reg_write, mem_read, mem_write;
output reg [1:0] pc_src;

output [2:0] operation;

output reg [1:0] reg_dst;

reg [1:0] alu_op;

alu_controller ALU_CTRL(alu_op, func, operation);

reg[4:0] ps, ns;

always @(posedge clk)
begin
    if (rst)
        ps <= `S0;
    else
        ps <= ns;
end

always @(ps, opcode)
begin
    case (ps)
        `S0:  ns = `S1;

        `S1: begin
            case(opcode)
```

```
                `J:  ns       = `S9;
                `BEQ:    ns   = `S8;
                `RTYPE:    ns = `S6;
                `LW:     ns   = `S2;
                `SW:      ns  = `S2;
                `JAL:    ns   = `S10;
                `JR:    ns    = `S12;
                `SLTI:  ns    = `S13;

                default:ns = `S15; //ADDI
            endcase
        end
        `S2: begin
            case(opcode)
                `LW: ns    = `S3;
                default:ns = `S5;
            endcase
        end


        `S3:  ns = `S4;
        `S4:  ns = `S0;
        `S5:  ns = `S0;
        `S6:  ns = `S7;
        `S7:  ns = `S0;
        `S8:  ns = `S0;
        `S9:  ns = `S0;
        `S10: ns = `S11;
        `S11: ns = `S0;
        `S12: ns = `S0;
        `S13: ns = `S14;
        `S14: ns = `S0;
        `S15: ns = `S16;
        `S16: ns = `S0;

    endcase
end

always @(ps)
begin
    {pc_write, pc_write_cond, IorD, ir_write, alu_src_a} = {5'b0};
    {alu_src_b}                                          = {2'b0};
    {mem_to_reg, reg_write, mem_read, mem_write, pc_src} = {6'b0};
    {reg_dst}                                            = {2'b0};
    {alu_op}                                             = {2'b00};
```

```
       case (ps)
           `S0:    {mem_read, ir_write, alu_src_b, pc_write} = {1'b1, 1'b1,
2'b01, 1'b1};
           `S1:    {alu_src_b}                               = {2'b11};
           `S2:    {alu_src_a, alu_src_b}                    = {1'b1, 2'b10};
           `S3:    {IorD, mem_read}                          = {1'b1, 1'b1};
           `S4:    {reg_write, mem_to_reg}                   = {1'b1, 1'b1};
           `S5:    {mem_write, IorD}                         = {1'b1, 1'b1};
           `S6:    {alu_src_a, alu_op}  = {1'b1, 2'b10};
           `S7:    {reg_dst, reg_write} = {1'b1, 1'b1};
           `S8:    {alu_src_a, alu_op, pc_write_cond, pc_src} = {1'b1, 2'b01,
1'b1, 2'b10};
           `S9:    {pc_write, pc_src} = {1'b1, 2'b01};
           `S10:   {reg_write, reg_dst} = {1'b1, 2'b10};
           `S11:   {pc_src, pc_write} = {2'b01, 1'b1};
           `S12:   {pc_src} = {2'b11}; //Etmaame JR
           `S13:   {alu_src_b, alu_src_a, alu_op} = {2'b10, 1'b1, 2'b11};
           `S14:   {reg_write} = {1'b1};
           `S15:   {alu_src_a, alu_src_b} = {1'b1, 2'b10};
           `S16:   {reg_write} = {1'b1};
       endcase
   end

   assign zero_out = zero_in;

endmodule
```

Testing

The processor is tested using the following test bench:

Load #20 32bit numbers in 2's complement format from a .mem file into Data Memory and find their minimum and it's index and finally store them at address 2000 and 2004 of Data Memory.

Pseudocode:

```
min = a[0]
min_idx = 0

for (int i = 1 ; i < 20 ; i++)
      if (a[i] < mn)
      {
            min = a[i]
            min_idx = i
      }
```

The pseudocode above is converted to an equivalent set of Assembly instructions:

Assembly Instructions:

```
            LW R4, 1000(R0)
            ADDi R5, R0, 0
            ADDi R1, R0, 1
            ADDi R2, R0, 20

iLoop:      Beq R1, R2, AFTER_LOOP
            ADD R3, R0, R1
            ADD R3, R3, R3
            ADD R3, R3, R3
```

```
            LW R7, 1000(R3)
            SLT R6, R7, R4
            Beq R6, R0, END_LOOP
            ADD R4, R0, R7
            ADD R5, R0, R1
END_LOOP:   ADDi R1, R1, 1
            J iLoop
AFTER_LOOP:SW R4, 2000(R0)
            SW R5, 2004(R0)
```

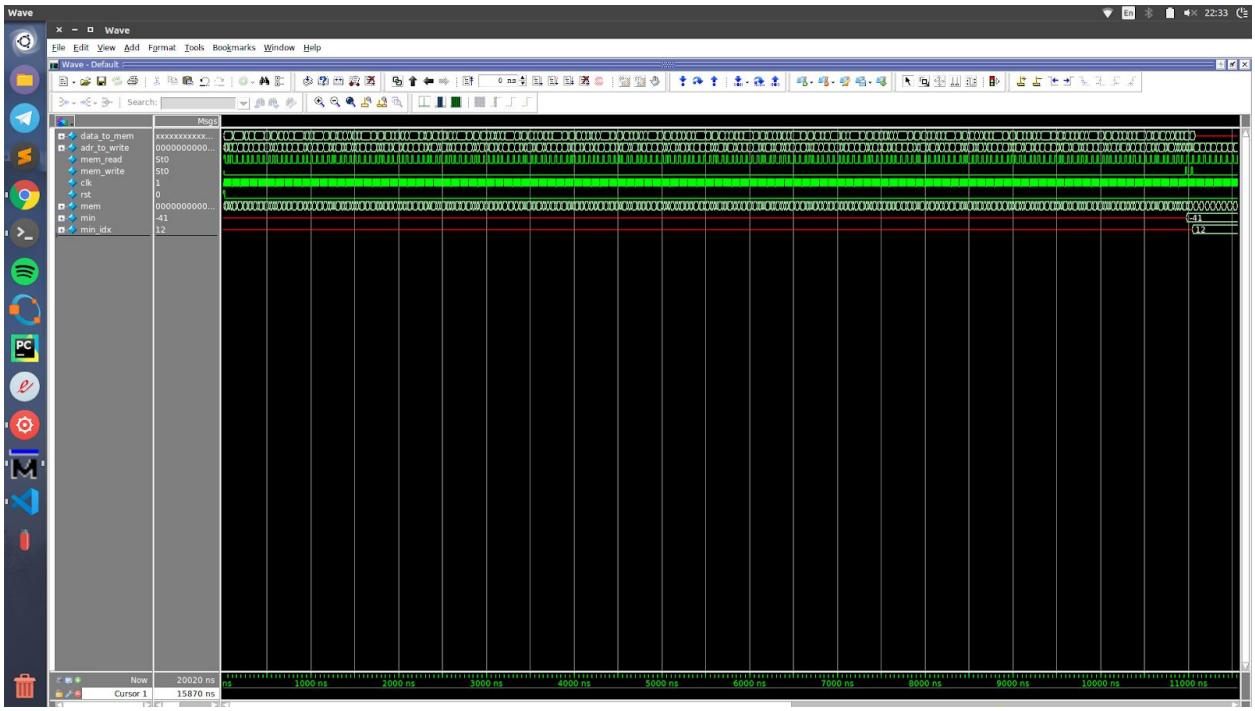
The loop variable (i) is stored in R1.
Minimum value (min) is stored in R4.
Minimum value's index (min_idx) is stored in R5.

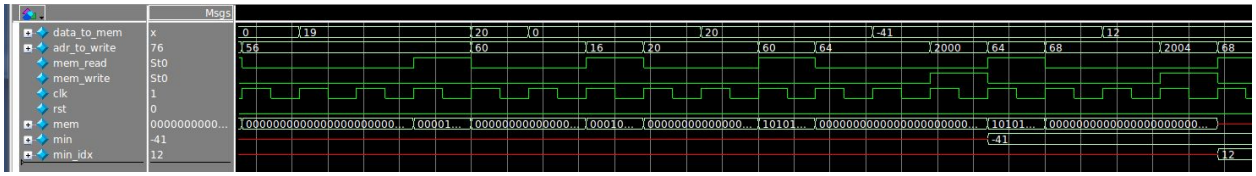
The array of 20 random signed numbers loaded into Data Memory is as followed:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
37	27	49	94	42	28	83	113	62	109	73	82	-41	102	63	111	-27	25	68	32

The first row corresponds to the indices while the second row corresponds to the values stored in that index.

Waveforms:





As it can be seen, the minimum number in the test set is A[12] = -41 which is shown on *min* and *min_idx* in the waveforms after executing all instructions.