Organization of Digital Computers Lab
EECS 112L

Lab 4

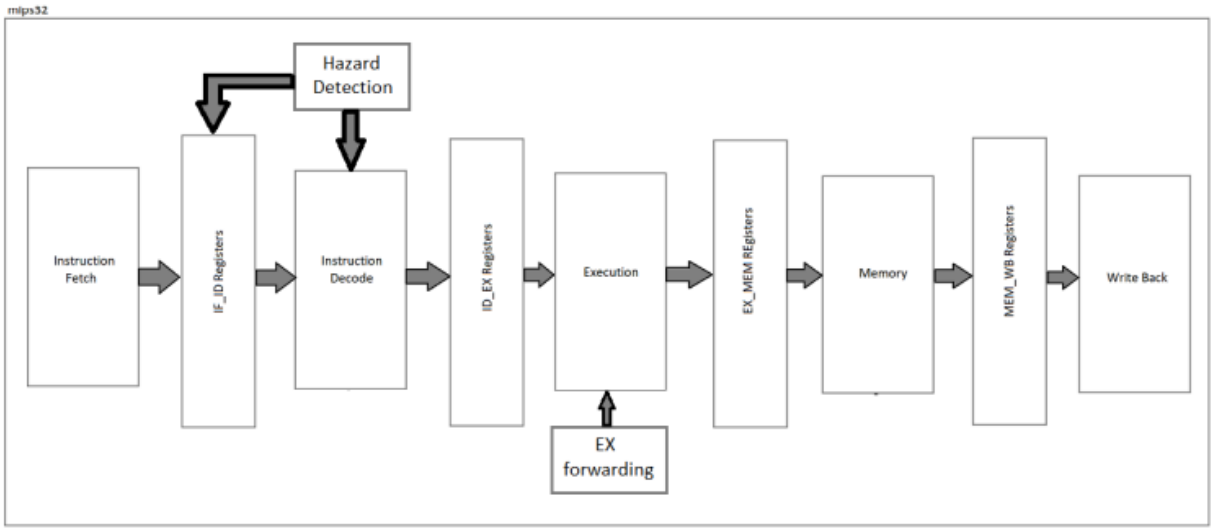
Kyle Zyler Cayanan
80576955

02/13/2022

1 - Objective

The goal of this lab was to convert our single cycle MIPS processor from lab 2 into a processor that was capable of pipelining. To do this we must create registers/flip flops and certain stages of the datapath to save data at specific points so that we may complete several instructions in a single cycle rather than one instruction per cycle. The objective was simply to implement the design shown below.

Figure 1: Pipeline processor



2 - Procedure

We start by breaking down the datapath into five stages: Instruction Fetch, Instruction Decode, Execution, Memory, and Write Back.

2.1 - Instruction Fetch

This stage is implemented using the following diagram below. After wiring all the components together we send the results to a pipelining register shown below which then sends the same data to the next stage. Here we have two muxes, both are responsible for determining whether we have a conditional branch or unconditional branch to an instruction or continuing with the next instruction like normal.

Figure 2: Instruction Fetch.

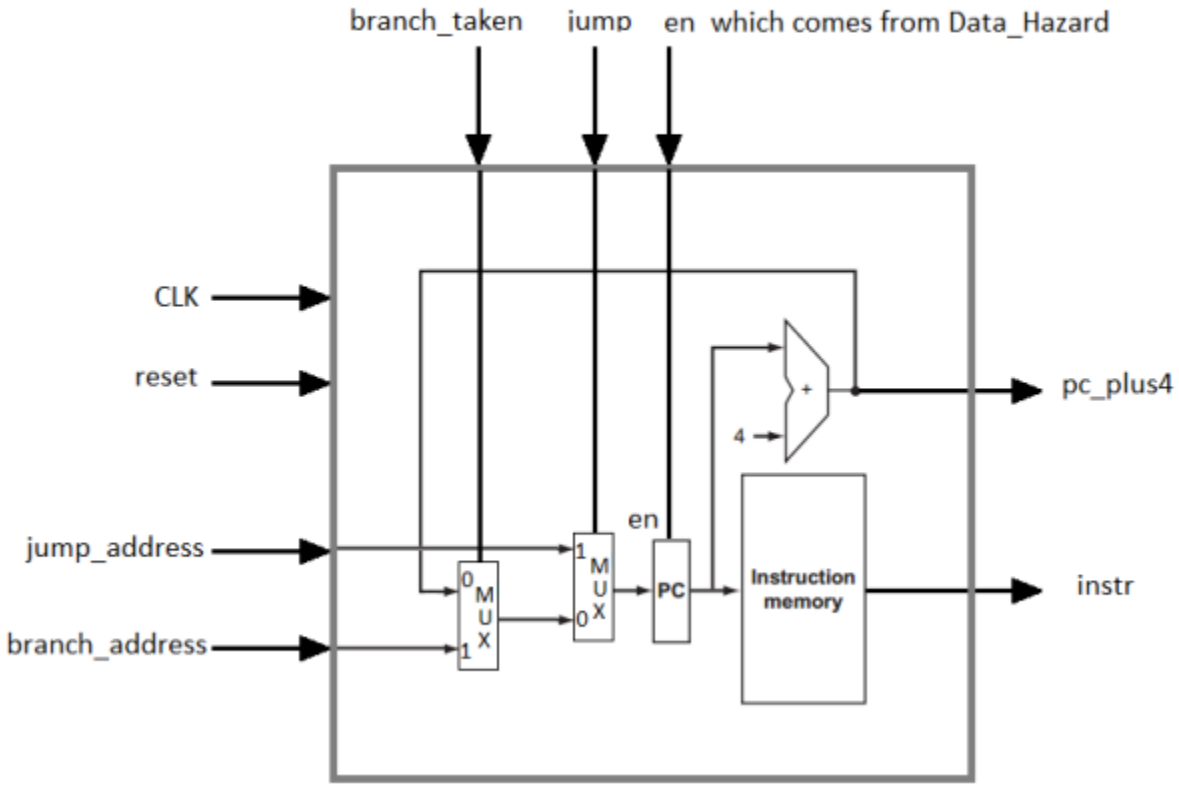
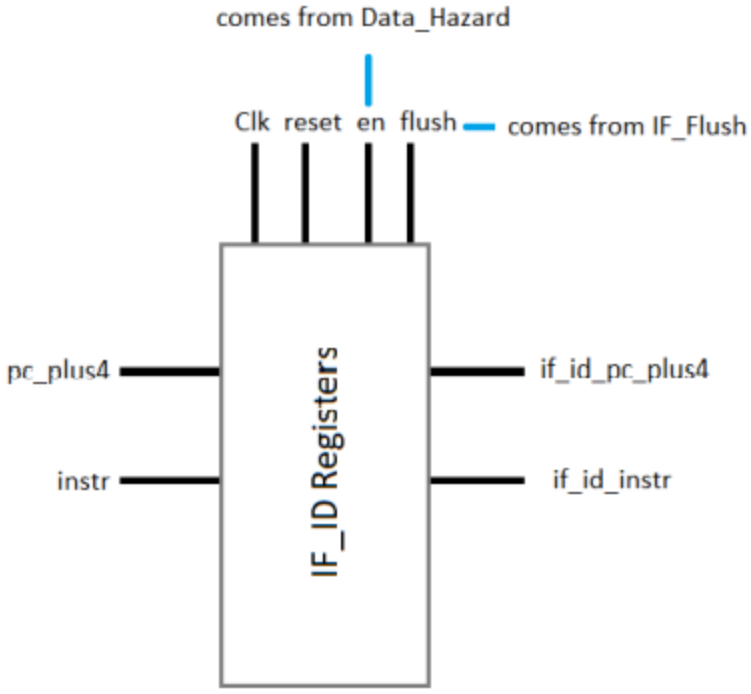


Figure 3: IF/ID Registers.



2.2 - Instruction Decode

This stage gets implemented by creating the diagram shown below in verilog. This stage is dependent on the clk and reset input for the register file module. The entire module takes in the following inputs: The instruction taken from the instruction fetch to instruction decode pipeline register, destination register, read and write signals and the data from the memory/write back pipeline register. In this module we determine the control signals for the execution stage, and retrieve our data for our arithmetic from the register file. The results are then sent to a pipeline register to save data between this stage and the execution stage.

Figure 4: Instruction Decode.

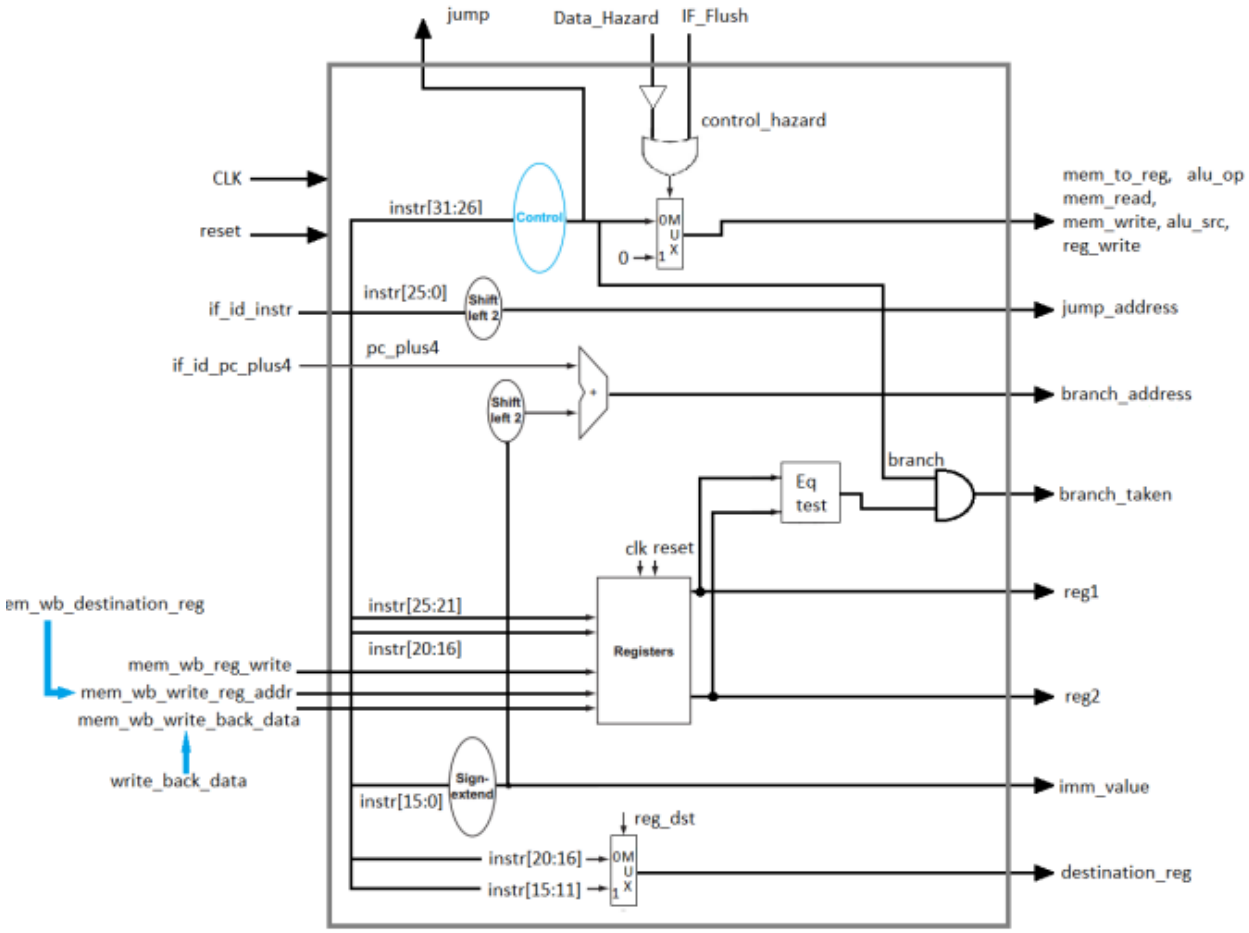
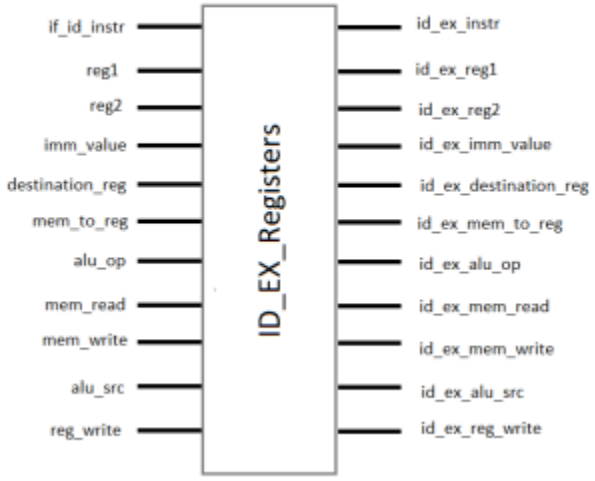


Figure 5: ID/EX Registers.



2.3 - Execution

This stage handles the arithmetic of our program. It takes in the immediate value, registers 1 and 2, the instruction, and our ALU operation from the ID/EX pipeline register and outputs our alu_results. These results end up in the ex/mem pipeline registers. Alongside handling the arithmetic we also must account for forwarding data and hazard detection in case we run into data that has dependencies. These hazards might not allow the data to be accessed at the right time and result in data that is too old or too early. This gets solved by forwarding and hazard detection modules shown below

Figure 7: Execution.

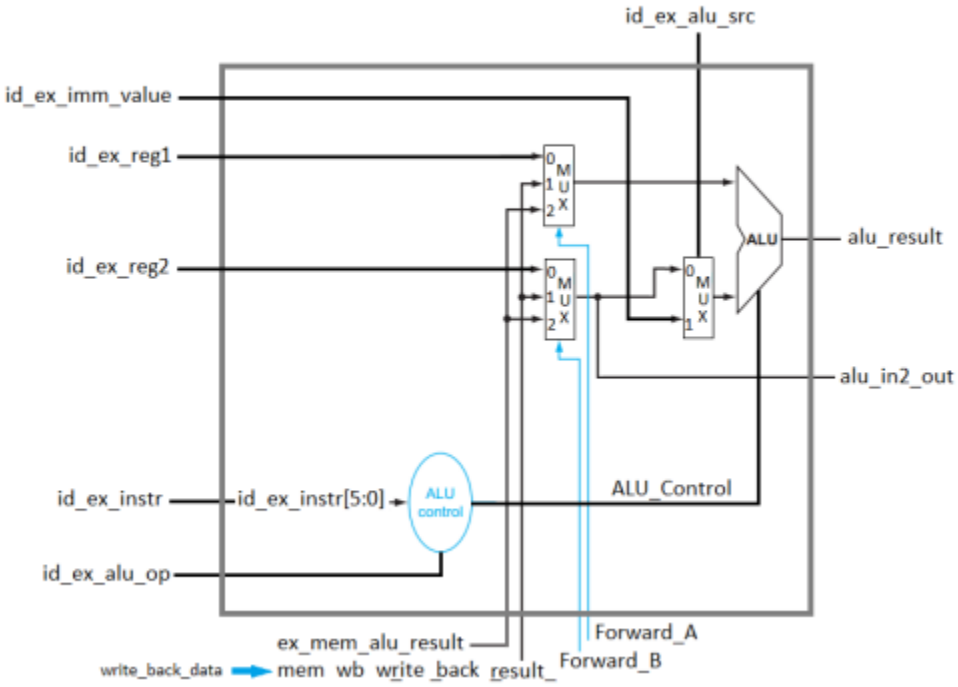


Figure 8: EX/MEM Registers.

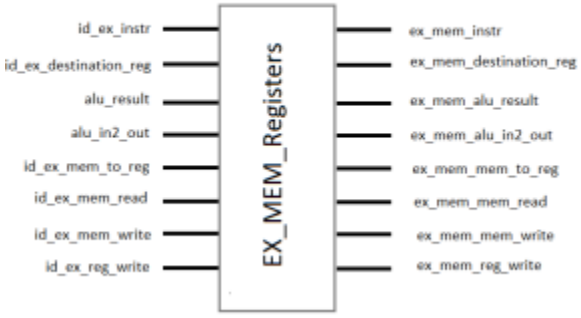
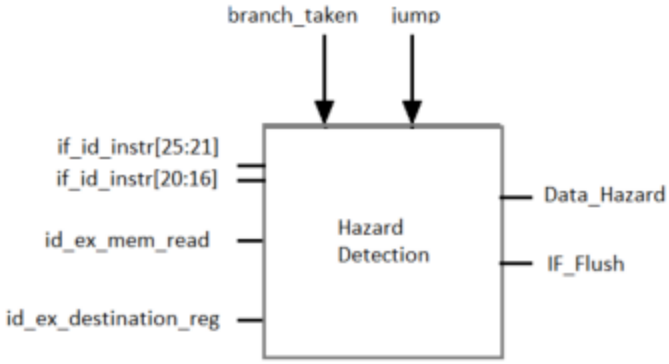


Figure 6: Hazard Detection.



The forwarding stage chooses if we get the data from the registers during the EX/MEM registers or MEM/WB registers. The registers are chosen under the following conditions.

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠  0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠  0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠  0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠  0)
       and (EX/MEM.RegisterRd =  ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠  0)
and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠  0)
       and (EX/MEM.RegisterRd =  ID/EX.RegisterRt))
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```

2.4 - Memory

This stage is simply the data memory where we store our results from the ALU. We get our inputs from the ex/mem registers and outputs get sent into the mem/wb pipeline registers

Figure 9: Memory.

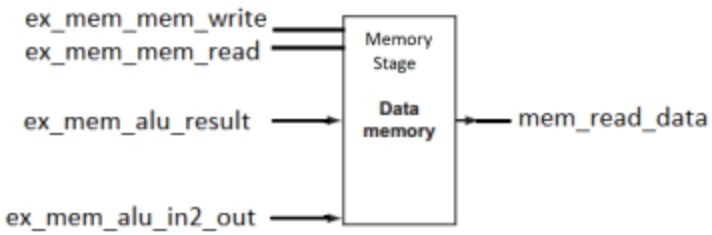
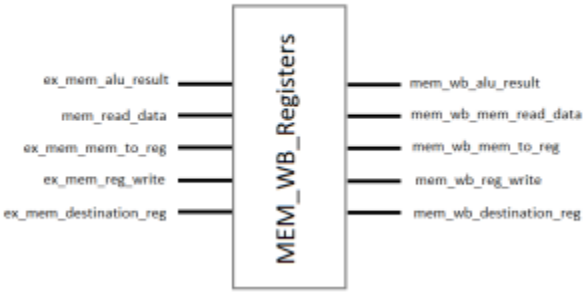


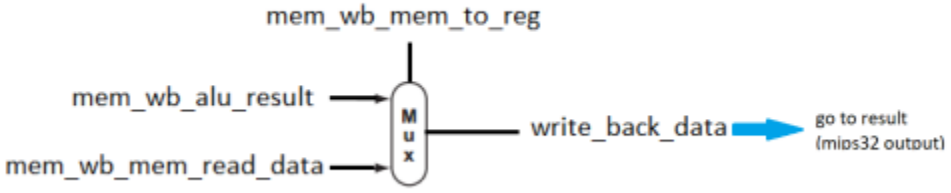
Figure 10: MEM/WB Registers.



2.5 - Write Back

This stage is simply just a mux that writes back data to the execution stage so it may be used for the next instruction.

Figure 11: Write Back.



3 - Simulation Results

We use the following instructions to test our processor and confirm that it is working. The test bench provided has a score keeper that tallies a successful instruction versus a failed instruction. We must note that certain jump and beq instructions will not work because of the hazards mentioned previously.

```
        ////////////////////////////////////////////////////// grading
        // load to registers 1 to 10
                                                  // instruction            alu result in hex       register content       mem content
        rom[0] = 32'b10001100000000010000000000000000; // r1  = mem[0]              4 (add 0)          r1 = 00000005            -
        rom[1] = 32'b10001100000000010000000000000100; // r2  = mem[1]              8 (add 1)          r2 = 0fdf6e91            -
        rom[2] = 32'b10001100000000011000000000001000; // r3  = mem[2]              c (add 2)          r3 = 6a31439b            -
        rom[3] = 32'b10001100000000100000000000001100; // r4  = mem[3]             10(add 3)           r4 = 56343ffd            -
        rom[4] = 32'b10001100000001010000000000010000; // r5  = mem[4]             14(add 4)           r5 = 429eeddb            -
        rom[5] = 32'b10001100000001100000000000010100; // r6  = mem[5]             18(add 5)           r6 = 90000000            -
        rom[6] = 32'b10001100000001110000000000011000; // r7  = mem[6]             1c(add 6)           r7 = 9134fd75            -
        rom[7] = 32'b10001100000010000000000000011100; // r8  = mem[7]             20(add 7)           r8 = bcd11247            -
        rom[8] = 32'b10001100000010010000000000100000; // r9  = mem[8]             24(add 8)           r9 = b55bd831            -
        rom[9] = 32'b10001100000010100000000000100100; // r10 = mem[9]             28(add 9)           r10= d18fa600            -

        // two positive operands
        rom[10] = 32'b00110000011010111111111101100011; // andi r11,r3,#ff63        6a314303            r11= 6a314303            -
        rom[11] = 32'b00000000001000100110000000100111; // nor  r12,r1,r2           f020916a            r12= f020916a            -
        rom[12] = 32'b00000000001000100110100000101010; // slt  r13,r1,r2           00000001            r13= 00000001            -
        rom[13] = 32'b11000000010000000111000011000000; // sll  r14,r2,#3           7efb7488            r14= 7efb7488            -
        rom[14] = 32'b11000000001000000111100101000010; // srl  r15,r1,#5           00000000            r15= 00000000            -
        rom[15] = 32'b11000000110000001000000110000011; // sra  r16,r6,#6           fe400000            r16= fe400000            -
        rom[16] = 32'b00000000010000111000100000100110; // xor  r17,r2,r3           65ee2d0a            r17= 65ee2d0a            -
        rom[17] = 32'b00000000001000101001000000011000; // mult r17,r1,r2           4f5d28d5            r18= 4f5d28d5            -
        rom[18] = 32'b00000000010000011001100000011010; // div  r19,r2,r1           032cafb6            r19= 032cafb6            -
        // store the result in memory
        rom[19] = 32'b10101100000010110000000000101100; // sw mem[r0+11] <= r11     2c(add 11)              -             mem[11]= 6a314303
        rom[20] = 32'b10101100000011000000000000110000; // sw mem[r0+12] <= r12     30(add 12)              -             mem[12]= f020916a
        rom[21] = 32'b10101100000011010000000000110100; // sw mem[r0+13] <= r13     34(add 13)              -             mem[13]= 00000001
        rom[22] = 32'b10101100000011100000000000111000; // sw mem[r0+14] <= r14     38(add 14)              -             mem[14]= 7efb7488
        rom[23] = 32'b10101100000011110000000000111100; // sw mem[r0+15] <= r15     3c(add 15)              -             mem[15]= 00000000
        rom[24] = 32'b10101100000100000000000001000000; // sw mem[r0+16] <= r16     40(add 16)              -             mem[16]= fe400000
        rom[25] = 32'b10101100000100001000000000001000100; // sw mem[r0+17] <= r17     44(add 17)              -             mem[17]= 65ee2d0a
        rom[26] = 32'b10101100000100100000000000001001000; // sw mem[r0+18] <= r18     48(add 18)              -             mem[18]= 4f5d28d5
        rom[27] = 32'b10101100000100110000000000001001100; // sw mem[r0+19] <= r19     4c(add 19)              -             mem[19]= 032cafb6

// one positive and one negative operand
rom[28] = 32'b00110000111010110000111101100011; // andi r11,r7,#f63          00000d61            r11= 00000d61            -
rom[29] = 32'b00000000010001110110000000100111; // nor  r12,r2,r7            6000000a            r12= 6000000a            -
rom[30] = 32'b00000000010001110110100000101010; // slt  r13,r2,r7            00000001            r13= 00000001            -
rom[31] = 32'b11000000111000000111001101000000; // sll  r14,r2,#13           9faea000            r14= 9faea000            -
rom[32] = 32'b11000001000000000011110011000010; // srl  r15,r8,#7            0179a224            r15= 0179a224            -
rom[33] = 32'b11000001001000001000000010000011; // sra  r16,r9,#2            ed56f60c            r16= ed56f60c            -
rom[34] = 32'b00000000010001111000100000100110; // xor  r17,r2,r7            9eeb93e4            r17= 9eeb93e4            -
rom[35] = 32'b00000000010001111001000000011000; // mult r17,r2,r7            a7d6d545            r18= a7d6d545            -
rom[36] = 32'b00000000111000101001100000011010; // div  r19,r7,r2            00000009            r19= 00000009            -
// store the result in memory
rom[37] = 32'b10101100000010110000000000101000; // sw mem[r0+20] <= r11      50(add 20)              -             mem[20]= 00000d61
rom[38] = 32'b10101100000011000000000000101100; // sw mem[r0+21] <= r12      54(add 21)              -             mem[21]= 6000000a
rom[39] = 32'b10101100000011010000000000101100; // sw mem[r0+22] <= r13      58(add 22)              -             mem[22]= 00000001
rom[40] = 32'b10101100000011100000000000101110; // sw mem[r0+23] <= r14      5c(add 23)              -             mem[23]= 9faea000
rom[41] = 32'b10101100000011110000000000101100; // sw mem[r0+24] <= r15      60(add 24)              -             mem[24]= 0179a224
rom[42] = 32'b10101100000100000000000000101100; // sw mem[r0+25] <= r16      64(add 25)              -             mem[25]= ed56f60c
rom[43] = 32'b10101100000100010000000000101000; // sw mem[r0+26] <= r17      68(add 26)              -             mem[26]= 9eeb93e4
rom[44] = 32'b10101100000100100000000000101100; // sw mem[r0+27] <= r18      6c(add 27)              -             mem[27]= a7d6d545
rom[45] = 32'b10101100000100110000000000101000; // sw mem[r0+28] <= r19      70(add 28)              -             mem[28]= 00000009

// one positive and one negative operand
rom[46] = 32'b00110001010010111110000100100111; // andi r11,r10,#e127        d18fa000            r11= d18fa000            -
rom[47] = 32'b00000000011010000110000000100111; // nor  r12,r3,r8            010eac20            r12= 010eac20            -
rom[48] = 32'b00000001000000110110100000101010; // slt  r13,r8,r3            00000000            r13= 00000000            -
rom[49] = 32'b11000000011000000111010001000000; // sll  r14,r3,#17           87360000            r14= 87360000            -
rom[50] = 32'b11000001000000000111110100000010; // srl  r15,r8,#20           00000bcd            r15= 00000bcd            -
rom[51] = 32'b11000001000000001000000001000011; // sra  r16,r8,#1            de688923            r16= de688923            -
rom[52] = 32'b00000000011010001000100000100110; // xor  r17,r3,r8            d6e051dc            r17= d6e051dc            -
rom[53] = 32'b00000000011010001001000000011000; // mult r17,r3,r8            eff5a5fd            r18= eff5a5fd            -
rom[54] = 32'b00000001000000111001100000011010; // div  r19,r8,r3            00000001            r19= 00000001            -
// store the result in memory
rom[55] = 32'b10101100000001011000000000101000; // sw mem[r0+29] <= r11      74(add 29)              -             mem[29]= d18fa000
rom[56] = 32'b10101100000011000000000000101100; // sw mem[r0+30] <= r12      78(add 30)              -             mem[30]= 010eac20
rom[57] = 32'b10101100000011010000000000101100; // sw mem[r0+31] <= r13      7c(add 31)              -             mem[31]= 00000000
rom[58] = 32'b10101100000011100000000010000000; // sw mem[r0+32] <= r14      80(add 32)              -             mem[32]= 87360000
rom[59] = 32'b10101100000011110000000010000100; // sw mem[r0+33] <= r15      84(add 33)              -             mem[33]= 00000bcd
rom[60] = 32'b10101100000100000000000010001000; // sw mem[r0+34] <= r16      88(add 34)              -             mem[34]= de688923
rom[61] = 32'b10101100000100010000000010001100; // sw mem[r0+35] <= r17      8c(add 35)              -             mem[35]= d6e051dc
rom[62] = 32'b10101100000100100000000010010000; // sw mem[r0+36] <= r18      90(add 36)              -             mem[36]= eff5a5fd
rom[63] = 32'b10101100000100110000000010010100; // sw mem[r0+37] <= r19      94(add 37)              -             mem[37]= 00000001
```

```
// two negative operands (not for shift)
rom[64] = 32'b00110001000010111101000000000010; // andi r11,r8,#d002          bcd11002          r11= bcd11002             -
rom[65] = 32'b00000000111010000110000000100111; // nor  r12,r7,r8             420a0088          r12= 420a0088             -
rom[66] = 32'b00000001000001110110100000101010; // slt  r13,r8,r7             00000000          r13= 00000000             -
rom[67] = 32'b11000000111000000111000111000000; // sll  r14,r7,#7             9a7eba80          r14= 9a7eba80             -
rom[68] = 32'b11000001001000000111100011000010; // srl  r15,r9,#3             16ab7b06          r15= 16ab7b06             -
rom[69] = 32'b11000001001000001000000101000011; // sra  r16,r9,#5             fdaadec1          r16= fdaadec1             -
rom[70] = 32'b00000000111010001000100000100110; // xor  r17,r7,r8             2de5ef32          r17= 2de5ef32             -
rom[71] = 32'b00000000111010001001000000011000; // mult r17,r7,r8             d8098573          r18=d8098573              -
rom[72] = 32'b00000001000001111001100000011010; // div  r19,r8,r7             00000001          r19= 00000001             -
// store the result in memory
rom[73] = 32'b10101100000010110000000010011000; // sw mem[r0+38] <= r11       98(add 38)              -             mem[38]= bcd11002
rom[74] = 32'b10101100000011000000000010011100; // sw mem[r0+39] <= r12       9c(add 39)              -             mem[39]= 420a0088
rom[75] = 32'b10101100000011010000000010100000; // sw mem[r0+40] <= r13       a0(add 40)              -             mem[40]= 00000000
rom[76] = 32'b10101100000011100000000010100100; // sw mem[r0+41] <= r14       a4(add 41)              -             mem[41]= 9a7eba80
rom[77] = 32'b10101100000011110000000010101000; // sw mem[r0+42] <= r15       a8(add 42)              -             mem[42]= 16ab7b06
rom[78] = 32'b10101100000100000000000010101100; // sw mem[r0+43] <= r16       ac(add 43)              -             mem[43]= fdaadec1
rom[79] = 32'b10101100000100010000000010110000; // sw mem[r0+44] <= r17       b0(add 44)              -             mem[44]= 2de5ef32
rom[80] = 32'b10101100000100100000000010110100; // sw mem[r0+45] <= r18       b4(add 45)              -             mem[45]= d8098573
rom[81] = 32'b10101100000100110000000010111000; // sw mem[r0+46] <= r19       b8(add 46)              -             mem[46]= 00000001

// zero result or overflow
rom[82] = 32'b00110001000010110000000000000000; // andi r11,r8,#0             00000000          r11= 00000000             -
rom[83] = 32'b00000000110010100110000000100111; // nor  r12,r6,r10            2e7059ff          r12= 2e7059ff             -
rom[84] = 32'b00000000110010100110100000101010; // slt  r13,r6,r10            00000001          r13= 00000001             -
rom[85] = 32'b11000000111000000111011111000000; // sll  r14,r7,#31            80000000          r14= 80000000             -
rom[86] = 32'b11000001001000000111111111000010; // srl  r15,r9,#31            00000001          r15= 00000001             -
rom[87] = 32'b11000001001000001000011111000011; // sra  r16,r9,#31            ffffffff          r16= ffffffff             -
rom[88] = 32'b00000001001010101000100000100110; // xor  r17,r9,r10            64d47e31          r17= 64d47e31             -
rom[89] = 32'b00000001001010101001000000011000; // mult r17,r9,r10            528ec600          r18= 528ec600             -
rom[90] = 32'b00000001001010101001100000011010; // div  r19,r9,r10            00000000          r19= 00000000             -
// store the result in memory
rom[91] = 32'b10101100000010110000000010111100; // sw mem[r0+47] <= r11       bc(add 47)              -             mem[47]= 00000000
rom[92] = 32'b10101100000011000000000011000000; // sw mem[r0+48] <= r12       c0(add 48)              -             mem[48]= 2e7059ff
rom[93] = 32'b10101100000011010000000011000100; // sw mem[r0+49] <= r13       c4(add 49)              -             mem[49]= 00000001
rom[94] = 32'b10101100000011100000000011001000; // sw mem[r0+50] <= r14       c8(add 50)              -             mem[50]= 80000000
rom[95] = 32'b10101100000011110000000011001100; // sw mem[r0+51] <= r15       cc(add 51)              -             mem[51]= 00000001
rom[96] = 32'b10101100000100000000000011010000; // sw mem[r0+52] <= r16       d0(add 52)              -             mem[52]= ffffffff
rom[97] = 32'b10101100000100010000000011010100; // sw mem[r0+53] <= r17       d4(add 53)              -             mem[53]= 64d47e31
rom[98] = 32'b10101100000100100000000011011000; // sw mem[r0+54] <= r18       d8(add 54)              -             mem[54]= 528ec600
rom[99] = 32'b10101100000100110000000011011100; // sw mem[r0+55] <= r19       dc(add 55)              -             mem[55]= 00000000
```

```
rom[100] = 32'b00100000001011011111111111111101; // addi r13,r1,#fffd         00000002           r13 = 00000002             -
rom[101] = 32'b10001100000011100000000000000000; // r14  = mem[0]             00000000           r14 = 00000005             -
rom[102] = 32'b00100000001011111111111111111110; // addi r15,r1,#fffe         00000003           r15 = 00000003             -


// branch forward taken
rom[103] = 32'b00010000001011100000000000000001; // beq r1,r14,#1             00000000           branch to instruction rom[105]
rom[104] = 32'b00000000001000100101000000100000; // add r10,r1,r2             0fdf6e96           r10= 0fdf6e96         doesnt run
rom[105] = 32'b00000000001000111001100000100000; // add r19,r1,r3             6a3143a0           r19= 6a3143a0              -
rom[106] = 32'b10101100000001010000000001100000; // sw mem[r0+56] <= r10      e0(add 56)              -             mem[56]= d18fa600

// branch forward not taken
rom[107] = 32'b00010000110010100000000000000001; // beq r6,r10,#1             be705a00           not taken
rom[108] = 32'b00000000001001000110000000100000; // add r12,r1,r4             56344002           r12= 56344002              -
rom[109] = 32'b00000000001000111001100000100000; // add r19,r1,r3             6a3143a0           r19= 6a3143a0              -
rom[110] = 32'b10101100000011000000000011100100; // sw mem[r0+57] <= r12      e4(add 57)              -             mem[56]= 56344002

// branch backward taken
rom[111] = 32'b00100001101011010000000000000001; // addi r13,r13,#1           00000003           r13 = 00000003             -
rom[112] = 32'b00010001101011111111111111111110; // beq r13,r15,#-2           00000000           branch to instruction rom[111]
// here if the content of r13 = 00000003 then the branch works. The first time branch is taken the second time is not taken
rom[113] = 32'b10101100000011010000000011101000; // sw mem[r0+58] <= r13      e8(add 58)              -             mem[58]= 00000004

// branch backward not taken
rom[114] = 32'b00100001110100000000000000000011; // addi r16,r14,#3           00000008           r16 = 00000008             -
rom[115] = 32'b00000010000000011000000000100000; // add r16,r16,r1            0000000d           r16 = 0000000d             -
rom[116] = 32'b00010000001100001111111111111110; // beq r1,r16,#-2            fffffff8           not taken
rom[117] = 32'b10101100000010000000000011101100; // sw mem[r0+59] <= r16      ec(add 59)              -             mem[59]= 0000000d

// branch forward taken
rom[118] = 32'b00010000001011100000000000000001; // beq r1,r14,#1             00000000           branch to instruction rom[120]
rom[119] = 32'b00000000001000100111100000100000; // add r15,r1,r2             0fdf6e96           r15= 0fdf6e96         doesnt run
rom[120] = 32'b00000000001000111001100000100000; // add r19,r1,r3             6a3143a0           r19= 6a3143a0              -
rom[121] = 32'b10101100000011110000000011110000; // sw mem[r0+60] <= r15      f0(add 60)              -             mem[60]= 00000003



//jump forward
rom[122] = 32'b00001000000000000000000001111100; // j #7c                     jump to instruction rom[124]
rom[123] = 32'b00000000001000100101000000100000; // add r10,r1,r2             0fdf6e96           r10= 0fdf6e96         doesnt run
rom[124] = 32'b10101100000010100000000011110100; // sw mem[r0+61] <= r10      f4(add 61)              -             mem[61]= d18fa600


//jump forward
rom[122] = 32'b00001000000000000000000001111100; // j #7c                     jump to instruction rom[124]
rom[123] = 32'b00000000001000100101000000100000; // add r10,r1,r2             0fdf6e96           r10= 0fdf6e96         doesnt run
rom[124] = 32'b10101100000010100000000011110100; // sw mem[r0+61] <= r10      f4(add 61)              -             mem[61]= d18fa600

//jump forward
rom[125] = 32'b00001000000000000000000010000000; // j #80                    jump to instruction rom[128]
rom[126] = 32'b00000000001000100100100000100000; // add r9,r1,r2              0fdf6e96           r9 = 0fdf6e96         doesnt run
rom[127] = 32'b00000000001000100111100000100000; // add r15,r1,r2             0fdf6e96           r15= 0fdf6e96         doesnt run
rom[128] = 32'b10101100000010010000000011111000; // sw mem[r0+62] <= r9       f8(add 62)              -             mem[62]= b55bd831

//jump forward
rom[129] = 32'b00001000000000000000000010000100; // j #84                    jump to instruction rom[132]
rom[130] = 32'b00000000001000100100000000100000; // add r8,r1,r2              0fdf6e96           r8 = 0fdf6e96         doesnt run
rom[131] = 32'b00000000001000100111100000100000; // add r15,r1,r2             0fdf6e96           r15= 0fdf6e96         doesnt run
rom[132] = 32'b10101100000010000000000011111100; // sw mem[r0+63] <= r8       fc(add 63)              -             mem[63]= bcd11247

//jump forward
rom[133] = 32'b00001000000000000000000010001000; // j #88                    jump to instruction rom[136]
rom[134] = 32'b00000000001000100011100000100000; // add r7,r1,r2              0fdf6e96           r7 = 0fdf6e96         doesnt run
rom[135] = 32'b00000000001000100111100000100000; // add r15,r1,r2             0fdf6e96           r15= 0fdf6e96         doesnt run
rom[136] = 32'b10101100000001110000000100000000; // sw mem[r0+64] <= r7       100(add 64)             -             mem[64]= 9134fd75

//jump forward
rom[137] = 32'b00001000000000000000000010001100; // j #8c                    jump to instruction rom[40]
rom[138] = 32'b00000000001000100011000000100000; // add r6,r1,r2              0fdf6e96           r6 = 0fdf6e96         doesnt run
rom[139] = 32'b00000000001000100111100000100000; // add r15,r1,r2             0fdf6e96           r15= 0fdf6e96         doesnt run
rom[140] = 32'b10101100000001100000000100000100; // sw mem[r0+65] <= r6       104(add 65)             -             mem[65]= 90000000
```

ANDI 1 success!

NOR 1 success!

SLT 1 success!

SLL 1 success!

SRL 1 success!

SRA 1 success!

XOR 1 success!

MULT 1 success!

DIV 1 success!

ANDI 2 success!

NOR 2 success!

SLT 2 success!

SLL 2 success!

SRL 2 success!

SRA 2 success!

XOR 2 success!

MULT 2 success!

DIV 2 success!

ANDI 3 success!

NOR 3 success!

SLT 3 success!

SLL 3 success!

SRL 3 success!

SRA 3 success!

XOR 3 success!

MULT 3 success!

DIV 3 success!

ANDI 4 success!

NOR 4 success!

SLT 4 success!

SLL 4 success!

SRL 4 success!

SRA 4 success!

XOR 4 success!

MULT 4 success!

DIV 4 success!

ANDI 5 success!

NOR 5 success!

SLT 5 success!

SLL 5 success!

SRL 5 success!

SRA 5	success!
XOR 5	success!
MULT 5	success!
DIV 5	success!
BEQ 1	failed!
BEQ 2	success!
BEQ 3	failed!
BEQ 4	success!
BEQ 5	success!
j 1	failed!
j 2	success!
j 3	failed!
j 4	failed!
j 5	failed!

points : 55