To make program run:

1) Open .qar file.

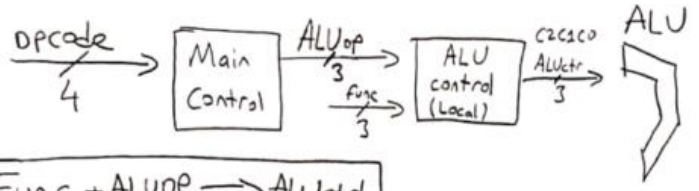
2) Compile project.

3) Open modelsim and compile all the files in the restored folder.

4) simulate MiniMips_w_instruction_tb.v file.

5) Press run button again until the program ends.

Instr	E3 E2 E1 E0 Opcode	F2 F1 F0 Func	P2 P1 P0 ALUop	Desired ALU action	C2 C1 C0 ALUctr	RegDst	ALUsrc	Memto Reg	Reg Wrt	Mem Read	Mem Write	Branch
AND	0000	000	000	AND	110	1	0	0	1	0	0	0
ADD	0000	001	000	ADD	000	1	0	0	1	0	0	0
SUB	0000	010	000	SUB	010	1	0	0	1	0	0	0
XOR	0000	011	000	XOR	001	1	0	0	1	0	0	0
NOR	0000	100	000	NOR	101	1	0	0	1	0	0	0
OR	0000	101	000	OR	111	1	0	0	1	0	0	0
ADDI	0001	XXX	001	ADD	000	0	1	0	1	0	0	0
ANDI	0010	XXX	010	AND	110	0	1	0	1	0	0	0
ORI	0011	XXX	011	OR	111	0	1	0	1	0	0	0
NORI	0100	XXX	100	NOR	101	0	1	0	1	0	0	0
BEQ	0101	XXX	101	SUB	010	x	0	x	0	0	0	1
BNE	0110	XXX	101	SUB	010	x	0	x	0	0	0	1
SLTI	0111	XXX	110	SLT	100	0	1	0	1	0	0	0
LW	1000	XXX	111	ADD	000	0	1	1	1	1	0	0
SW	1001	XXX	111	ADD	000	x	1	X	0	0	1	0



Func + ALUOP $\Longrightarrow$ ALUctrl

$C2 = (\overline{P2}\,\overline{P1}\,\overline{P0})(F2 + \overline{F1}\,\overline{F0}) + (\overline{P2}\,P1 + P2\,\overline{P0})$

$C1 = (\overline{P2}\,\overline{P1}\,\overline{P0})(F2 \odot F0) + (\overline{P2}\,P1 + P2\,\overline{P1}\,P0)$

(XNOR)

$C0 = (\overline{P2}\,\overline{P1}\,\overline{P0})(F2 + F1F0) + (P2\,\overline{P1}\,\overline{P0} + \overline{P2}\,P1P0)$

$P2 = E3 + E2$

$P1 = E3 + E1E0 + \overline{E2}E1 = E3 + E1(E0 + \overline{E2})$

$P0 = E3 + \overline{E1}E0 + \overline{E2}E1E0 + E2E1\overline{E0}$

$= E3 + \overline{E1}E0 + E1(\overline{E2}E0 + E2\overline{E0})$

$= E3 + \overline{E1}E0 + E1(E2 \oplus E0)$ (XOR)

$R = \overline{E3}\,\overline{E2}\,\overline{E1}\,\overline{E0}$

$I = \overline{R}(\overline{P2}\,\overline{P0})$

$B = E2\,(E1 \oplus E0) \Rightarrow 101 + 110$

$LW = \overline{E3}\,\overline{E0}$

$SW = E3\,E0$

$RegDst = R$

$ALUsrc = I + LW + SW$

$MemtoReg = LW$

$RegWrt = R + I + LW$

$MemRead = LW$

$MemWrite = SW$

$Branch = B$

Main control testbench:

```
initial begin
opCode[3:0] = 4'b0000;
func[2:0] = 3'b000;
#`DELAY;
opCode[3:0] = 4'b0000;
func[2:0] = 3'b001;
#`DELAY;
opCode[3:0] = 4'b0000;
func[2:0] = 3'b010;
#`DELAY;
opCode[3:0] = 4'b0000;
func[2:0] = 3'b011;
#`DELAY;
opCode[3:0] = 4'b0000;
func[2:0] = 3'b100;
#`DELAY;
opCode[3:0] = 4'b0000;
func[2:0] = 3'b101;
#`DELAY;

opCode[3:0] = 4'b0001;
#`DELAY;
opCode[3:0] = 4'b0010;
#`DELAY;
opCode[3:0] = 4'b0011;
#`DELAY;
opCode[3:0] = 4'b0100;
#`DELAY;
opCode[3:0] = 4'b0101;
#`DELAY;
opCode[3:0] = 4'b0110;
#`DELAY;
opCode[3:0] = 4'b0111;
#`DELAY;
opCode[3:0] = 4'b1000;
#`DELAY;
opCode[3:0] = 4'b1001;
#`DELAY;
```

```
VSIM 647> run
# time =   0, a =0, b=0 , aluCtrl= 110,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  10, a =0, b=1 , aluCtrl= 0,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  20, a =0, b=10 , aluCtrl= 10,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  30, a =0, b=11 , aluCtrl= 1,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  40, a =0, b=100 , aluCtrl= 101,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  50, a =0, b=101 , aluCtrl= 111,RegDst= 1,ALUSrc= 0,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 0
# time =  60, a =1, b=101 , aluCtrl= 0,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 1
# time =  70, a =10, b=101 , aluCtrl= 110,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 1
# time =  80, a =11, b=101 , aluCtrl= 111,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 1
# time =  90, a =100, b=101 , aluCtrl= 101,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 1
VSIM 648> run
# time = 100, a =101, b=101 , aluCtrl= 10,RegDst= 0,ALUSrc= 0,MemtoReg= 0,RegWrt= 0,MemRead= 0,MemWrite= 0,Branch= 1,I TYPE : 0
# time = 110, a =110, b=101 , aluCtrl= 10,RegDst= 0,ALUSrc= 0,MemtoReg= 0,RegWrt= 0,MemRead= 0,MemWrite= 0,Branch= 1,I TYPE : 0
# time = 120, a =111, b=101 , aluCtrl= 100,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 1,MemRead= 0,MemWrite= 0,Branch= 0,I TYPE : 1
# time = 130, a =1000, b=101 , aluCtrl= 0,RegDst= 0,ALUSrc= 1,MemtoReg= 1,RegWrt= 1,MemRead= 1,MemWrite= 0,Branch= 0,I TYPE : 0
# time = 140, a =1001, b=101 , aluCtrl= 0,RegDst= 0,ALUSrc= 1,MemtoReg= 0,RegWrt= 0,MemRead= 0,MemWrite= 1,Branch= 0,I TYPE : 0

VSIM 648>
```

Mux2x1ThreeBit test:

```
mux2x1ThreeBits mux(write_reg,read_reg2,write_regRegDst1,RegDst);

initial begin
read_reg2[2:0] = 3'b110; write_regRegDst1[2:0] = 3'b001; RegDst = 1'b0;
#`DELAY;
read_reg2[2:0] = 3'b100; write_regRegDst1[2:0] = 3'b011;RegDst = 1'b1;
#`DELAY;
read_reg2[2:0] = 3'b110; write_regRegDst1[2:0] = 3'b111;RegDst = 1'b0;
#`DELAY;
read_reg2[2:0] = 3'b010; write_regRegDst1[2:0] = 3'b101;RegDst = 1'b1;
end
```

```
# time = 0, read_reg2 =110, b=1, sum=110,regdst = 0
# time = a, read_reg2 =100, b=11, sum=11,regdst = 1
# time = 14, read_reg2 =110, b=111, sum=110,regdst = 0
# time = 1e, read_reg2 =10, b=101, sum=101,regdst = 1

VSIM 651>
```

Instruction Memory test:

```
1  `define DELAY 10
2  module instruction_memory_tb();
3  reg clk;
4  wire [15:0] instruction_set;
5  wire [31:0] newPc;
6
7   reg [31:0] readAddress;
8   instruction_memory uut(instruction_set,newPc,readAddress,clk);
9   initial
10    begin
11       clk = 1'b1;
12       readAddress <=0;
13    end
14  always
15    begin
16       #1 clk = ~clk;
17    end
18  always
19    begin
20       #`DELAY;
21       if(readAddress>29)
22       begin
23          $stop;
24       end
25       else
26          begin
27             readAddress<=newPc;
28          end
29    end
30 initial begin
31  $readmemb("instructions.txt", uut.instructions);
32  end
33
34  initial
35 begin
36  $monitor("time : %1d,instruction: %1b, address = %1d,newPc=%1d",$time,instruction_set,readAddress,newPc);
37  end
38  endmodule
```

```
VSIM 706> run
# time : 0,instruction: x, address = 0,newPc=x
# time : 2,instruction: 1010011000, address = 0,newPc=1
# time : 10,instruction: 1010011000, address = 1,newPc=1
# time : 12,instruction: 1001001011000110, address = 1,newPc=2
# time : 20,instruction: 1001001011000110, address = 2,newPc=2
# time : 22,instruction: 1000001101000110, address = 2,newPc=3
# time : 30,instruction: 1000001101000110, address = 3,newPc=3
# time : 32,instruction: 101011101000001, address = 3,newPc=4
# time : 40,instruction: 101011101000001, address = 4,newPc=4
# time : 42,instruction: 1101111010001, address = 4,newPc=5
# time : 50,instruction: 1101111010001, address = 5,newPc=5
# time : 52,instruction: 1101001010001, address = 5,newPc=6
# time : 60,instruction: 1101001010001, address = 6,newPc=6
# time : 62,instruction: 110011100000001, address = 6,newPc=7
# time : 70,instruction: 110011100000001, address = 7,newPc=7
# time : 72,instruction: 1101001011101, address = 7,newPc=8
# time : 80,instruction: 1101001011101, address = 8,newPc=8
# time : 82,instruction: 110011101000001, address = 8,newPc=9
# time : 90,instruction: 110011101000001, address = 9,newPc=9
# time : 92,instruction: 1101110000001, address = 9,newPc=10
VSIM 707> run
# time : 100,instruction: 1101110000001, address = 10,newPc=10
# time : 102,instruction: 110010100000, address = 10,newPc=11
# time : 110,instruction: 110010100000, address = 11,newPc=11
# time : 112,instruction: 100010011001, address = 11,newPc=12
# Break in Module instruction_memory_tb at D:/ORG_HW4/DraftForHW4_restored/instruction_memory_tb.v line 23
```

Branch unit test:

```
initial begin
beqCommand = 1'b1;
noBranchNewPc = 32'hFB;
write_data = 32'h0;
imm = 32'h4;
Branch = 1'b1;
#`DELAY; // branch == successful

beqCommand = 1'b1;
noBranchNewPc = 32'hFB;
write_data = 32'h0;
imm = 32'h4;
Branch = 1'b0;
#`DELAY;  // branch != successful

beqCommand = 1'b1;
noBranchNewPc = 32'hFB;
write_data = 32'hF;
imm = 32'h4;
Branch = 1'b1;
#`DELAY; // branch != successful

beqCommand = 1'b0;
noBranchNewPc = 32'hFB;
write_data = 32'hF;
imm = 32'h4;
Branch = 1'b1;
#`DELAY; // branch == successful

beqCommand = 1'b0;
noBranchNewPc = 32'hFB;
write_data = 32'h0;
imm = 32'h4;
Branch = 1'b1;
#`DELAY; // branch != successful

beqCommand = 1'b0;
noBranchNewPc = 32'hFB;
write_data = 32'hF;
imm = 32'h4;
Branch = 1'b0;
#`DELAY; // branch != successful
end
```

```
# time = 0, beqCommand =1, noBranchNewPc=fb, write_data=0,imm=4,Branch=1, newPc = ff
# time = a, beqCommand =1, noBranchNewPc=fb, write_data=0,imm=4,Branch=0, newPc = fb
# time = 14, beqCommand =1, noBranchNewPc=fb, write_data=f,imm=4,Branch=1, newPc = fb
# time = 1e, beqCommand =0, noBranchNewPc=fb, write_data=f,imm=4,Branch=1, newPc = ff
# time = 28, beqCommand =0, noBranchNewPc=fb, write_data=0,imm=4,Branch=1, newPc = fb
# time = 32, beqCommand =0, noBranchNewPc=fb, write_data=f,imm=4,Branch=0, newPc = fb
```

Data memory test:

```
 data_memory dm(read_data,write_data, address, mem_write, mem_read,clk );
 initial
   begin
      clk = 1'b1;
   end
always
   begin
      #1 clk = ~clk;
   end
initial
   begin
   write_data = 32'hFF;
   address = 32'h5;
   mem_write = 1'b1;
   mem_read = 1'b0;
   #`DELAY;

   address = 32'h5;
   mem_write = 1'b0;
   mem_read = 1'b1;
   #`DELAY;

   write_data = 32'hFFFF_ABCD;
   address = 32'h5;
   mem_write = 1'b1;
   mem_read = 1'b0;
   #`DELAY;

   address = 32'h5;
   mem_write = 1'b0;
   mem_read = 1'b1;
   #`DELAY;

   end
initial begin
$readmemb("data.txt", dm.memory);
end
initial
begin
$monitor("time : %1d,writeCommand: %1b, address = %1h,written=%1h, readCommand = %1b,read: %1h",$time,mem_write,address,write_data,mem_read,read_data);
```

```
# time : 0,writeCommand: 1, address = 5,written=ff, readCommand = 0,read: x
# time : 10,writeCommand: 0, address = 5,written=ff, readCommand = 1,read: ff
# time : 20,writeCommand: 1, address = 5,written=ffffabcd, readCommand = 0,read: ff
# time : 30,writeCommand: 0, address = 5,written=ffffabcd, readCommand = 1,read: ffffabcd
```

ExtendImm test:

```
1   `define DELAY 10
2   module extendImm_tb();
3   reg[5:0] imm;
4   wire [31:0] extended;
5
6   extendImm extend(extended,imm);
7
8
9   initial begin
10  imm[5:0] = 6'hF;
11  #`DELAY;
12  imm[5:0] = 6'b111111;
13  #`DELAY;
14  imm[5:0] = 6'b001011;
15  #`DELAY;
16  end
17
18
19  initial
20  begin
21  $monitor("time = %1h, imm =%1b extend=> extended=%1b ", $time, imm, extended);
22  end
23  endmodule
24
```

```
VSIM 50> run
# time = 0, imm =1111 extend=> extended=1111
# time = a, imm =111111 extend=> extended=111111
# time = 14, imm =1011 extend=> extended=1011
```

Mips registers test:

```
   mips_registers mr( read_data_1, read_data_2, write_data, read_reg_1, read_reg_2, write_reg, signal_reg_write, clk );
   initial
     begin
        clk = 1'b1;
     end

  always
     begin
        #1 clk = ~clk;
     end
   initial
     begin
     write_data = 32'hFFFF_AAAA;
     write_reg = 3'h1;
     signal_reg_write = 1'b1;
     #`DELAY;

     read_reg_1 = 3'h1;
     signal_reg_write = 1'b0;
     #`DELAY;

     write_data = 32'hFFFF_AAAA; // Zero register's content can not be changed
     write_reg = 3'h0;
     signal_reg_write = 1'b1;
     #`DELAY;

     read_reg_1 = 3'h0;
     signal_reg_write = 1'b0;
     #`DELAY;
     end
initial begin
 $readmemb("registers.mem", mr.registers);
```

```
# time : 0,read_data_1 = x, read_data_2= x, write_data= ffffaaaa, read_reg_1= x, read_reg_2= x, write_reg= 1, signal_reg_write= 1
# time : 10,read_data_1 = ffffaaaa, read_data_2= x, write_data= ffffaaaa, read_reg_1= 1, read_reg_2= x, write_reg= 1, signal_reg_write= 0
# time : 20,read_data_1 = ffffaaaa, read_data_2= x, write_data= ffffaaaa, read_reg_1= 1, read_reg_2= x, write_reg= 0, signal_reg_write= 1
# time : 30,read_data_1 = 0, read_data_2= x, write_data= ffffaaaa, read_reg_1= 0, read_reg_2= x, write_reg= 0, signal_reg_write= 0
```

Mux2x1 32 Bit test:

```
`define DELAY 10
module mux2x1_32Bits_tb();

reg[31:0] input1,input2;
wire[31:0] out;
reg select;

 mux2x1_32Bits mux(out,input1,input2,select);
initial begin
input1 = 32'b110; input2 = 32'b001; select = 1'b0;
#`DELAY;
input1 = 32'b110; input2 = 32'b001; select = 1'b1;
#`DELAY;

end


initial
begin
$monitor("time = %1h, input1 = %1b,input2 = %1b,select = %1b,output = %1b", $time, input1,input2,select,out);
end
endmodule
```

```
# time = 0, input1 = 110,input2 = 1,select = 0,output = 110
# time = a, input1 = 110,input2 = 1,select = 1,output = 1
```

MiniMips test:

```
MiniMips_w_instruction uut(pc,clk,instruction_set, newPc);
 initial
   begin
      clk = 1'b0;
      pc <= 0;
   end
always
   begin
      #1 clk = ~clk;
   end
always
   begin
      #`DELAY;
      if(pc>35)
      begin
         $writememb("registers_outp.mem", uut.mipsregisters.registers);
         $writememb("data_memory_outp.mem", uut.DataMemory.memory);
         $stop;
      end
      else
         begin
            pc<=newPc;
            $writememb("registers_outp.mem", uut.mipsregisters.registers);
            $writememb("data_memory_outp.mem", uut.DataMemory.memory);
            // to be able to see result of instructions one by one, stop command must be activated
            //$stop;
         end
   end
initial begin
$readmemb("registers.mem", uut.mipsregisters.registers);
$readmemb("data.txt", uut.DataMemory.memory);
$readmemb("instructions.txt", uut.instructionMemory.instructions);
end
```

I do not know if the usage of ‘if’ to make program end is forbidden, but I could not find any other way to do it.

Instructions:

instructions.txt

D: > ORG_HW4 > DraftForHW4_restored > s

```
0000001010011000
1001001011000110
1000001101000110
0101011101000001
0001_101_111_010001
0001101001010001
0110011100000001
0001_101_001_011101
0110011101000001
0001_101_110_000001
0000_110_010_100_000
0000_100_010_011_001
0000_100_011_001_001
0000_101_100_010_010
0000_111_100_011_010
0000_111_011_010_011
0000_111_001_110_011
0000_111_011_100_100
0000_111_001_101_100
0000_001_110_010_101
0000_001_011_111_101
0010_101_001_111111
0010_001_011_110111
0011_101_001_111000
0011_001_011_101010
0100_101_001_111000
0100_001_011_100010
0111_001_100_111000
0111_100_101_100010
1001_101_111_000111
1000_101_001_000111
0011_011_001_111111
0011_011_000_111111
```

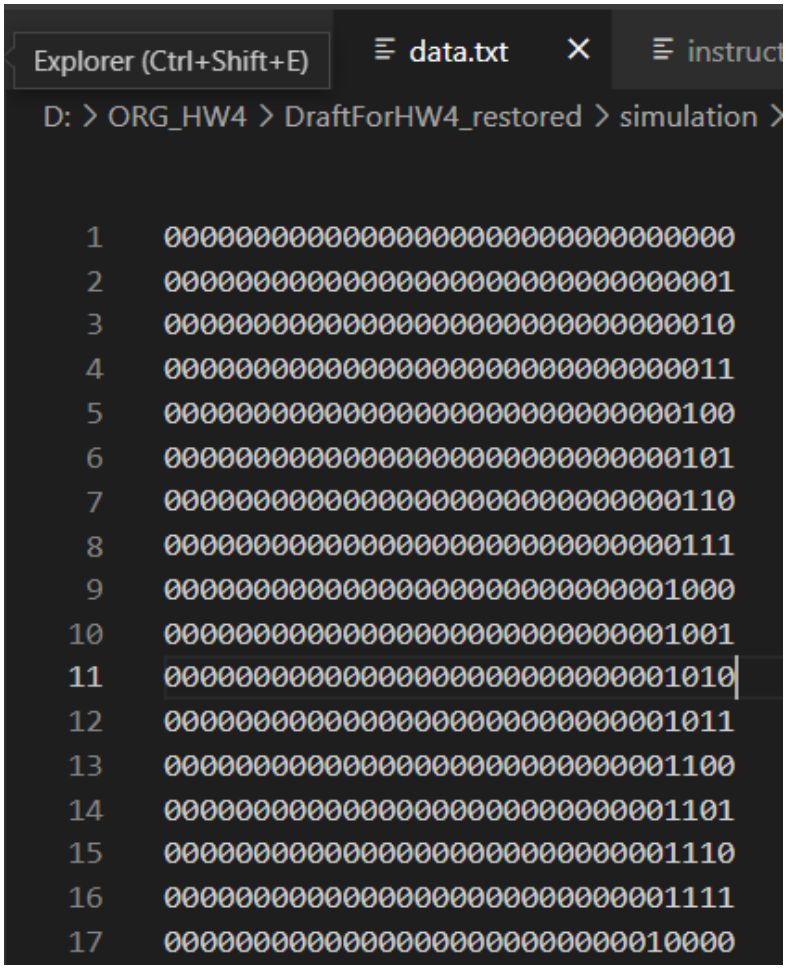
Registers at start:

registers.mem | data.txt | instructionsWithExplanation.txt

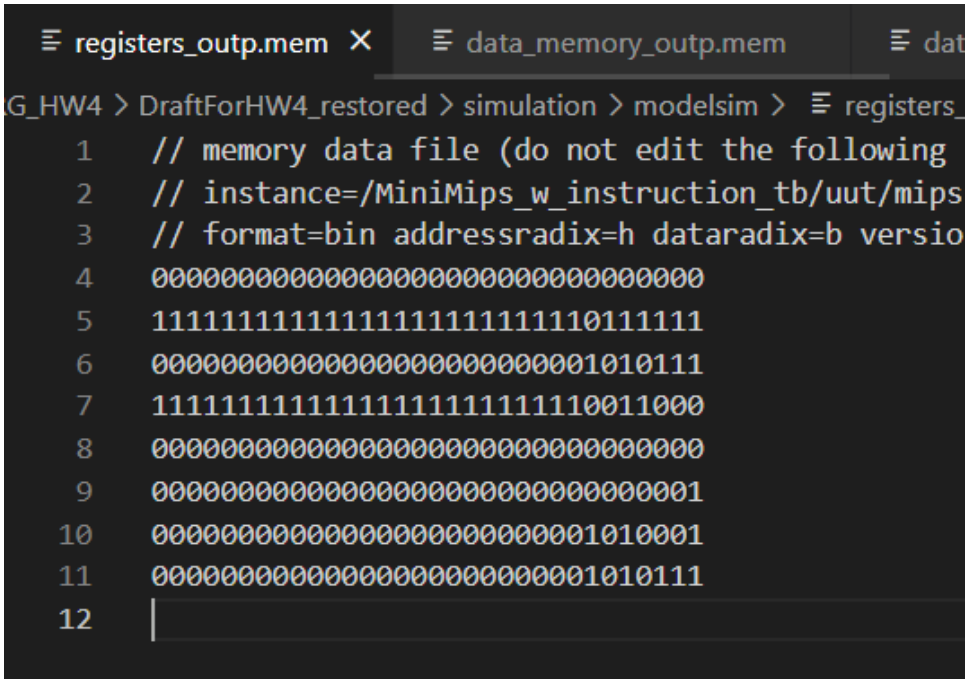
D: > ORG_HW4 > DraftForHW4_restored > simulation > modelsim > registers.mem

```
00000000000000000000000000000000
00000000000000000000000000001010
00000000000000000000000000010010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000100101
00000000000000001100000011111101
00000000000000000000000001000111
```

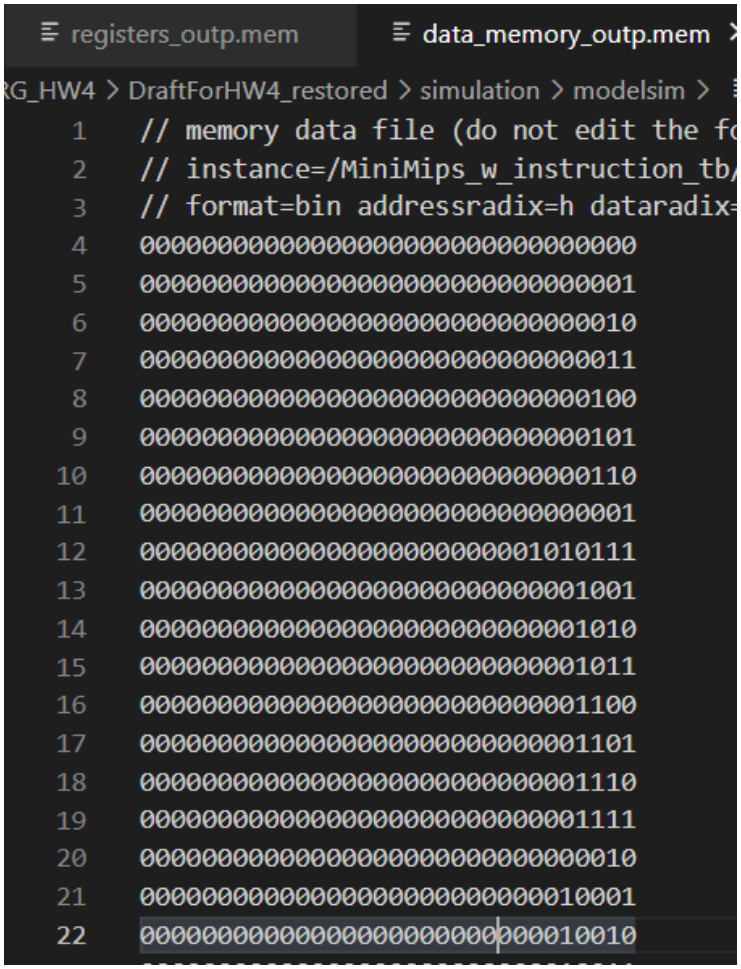
Memory at the start:



Registers at the end:



Data Memory at the end:



Instructions and their explanation, I also added this part as txt file, whose name is instructionsWithExplanation.txt:

0000001010011000 R3 = R1 & R2, R1 = 1010, R2 = 10010, R3 will become 10

1001001011000110 sw R3,R1(6), Mem[16] = 32'b10

1000001101000110 lw R5,R1(6), R5 = 32'b10

0101011101000001 // beq r3,r5,1, next command will not work

0001101111010001 addi R7,R5,10001(in binary) r7 will become 10 + 10001 = 10011

0101011100000001 // beq r3,r4,1, next command will work r3!=r4

0001101001010001 addi R1,R5,10001(in binary) r1 will become 10 + 10001 = 10011

0110011100000001 //bneq r3, r4,1 next command will not work r3!=r4

0001_101_001_011101 addi R1,R5,11101(in binary) r1 will not become 10 + 11101 = 11111, it will stay same

0110011101000001 //bneq r3, r5,1 next command will work r3==r5

0001_101_110_000001 addi R6,R5,11101(in binary) r6 will become 10 + 000001 = 11,

/*

registers:

0: 00000000000000000000000000000000

1: 00000000000000000000000000010011

2: 00000000000000000000000000010010

3: 00000000000000000000000000000010

4: 00000000000000000000000000000100

5: 00000000000000000000000000000010

6: 00000000000000000000000000000011

7: 00000000000000000000000001000111

*/

0000_110_010_100_000 R4 = R6 & R2, R6 = 11, R2 = 10010, R4 will become 10

0000_100_010_011_001 R3 = r4 + r2, r4 = 10, r2 =10010, r3 will become 10100

0000_100_011_001_001 R1 = r4 + r3, r4 = 10, r3 =10100, r1 will become 10110

/*

registers:

0: 00000000000000000000000000000000

1: 00000000000000000000000000010110

2: 00000000000000000000000000010010

3: 00000000000000000000000000010100

4: 00000000000000000000000000000010

5: 00000000000000000000000000000010

6: 00000000000000000000000000000011

7: 00000000000000000000000001000111

```
*/
sub:
0000_101_100_010_010 => R2 = R5 - R4, R2 will become 0
0000_111_100_011_010 => R3 = R7 - R4, R3 will become 1000111 - 10 = 1000101
/*
registers:
   00000000000000000000000000000000
   00000000000000000000000000010110
R2: 00000000000000000000000000000000
R3: 00000000000000000000000001000101
R4: 00000000000000000000000000000010
R5: 00000000000000000000000000000010
   00000000000000000000000000000011
R7: 00000000000000000000000001000111
*/
xor:
0000_111_011_010_011 // r2 = r7 xor r3 => r2 will be 10
0000_111_001_110_011 // r6 = r7 xor r1 => r6 will be 1010001
/*
registers:
   00000000000000000000000000000000
   00000000000000000000000000010110
R2: 00000000000000000000000000000010
   00000000000000000000000001000101
   00000000000000000000000000000010
   00000000000000000000000000000010
R6: 00000000000000000000000001010001
   00000000000000000000000001000111
*/
//nor
```

0000_111_011_100_100 // r4 = r7 (1000111)nor r3(1000101) => r4 will be 11111111111111111111111110111000

0000_111_001_101_100 // r5 = r7 (1000111)nor r1(0010110)=> r6 will be 11111111111111111111111110101000

/*

registers:

00000000000000000000000000000000

00000000000000000000000000010110

00000000000000000000000000000010

00000000000000000000000001000101

r4: 11111111111111111111111110111000

r5: 11111111111111111111111110101000

00000000000000000000000001010001

00000000000000000000000001000111

*/

0000_001_110_010_101 // r2 = r1 (10110)nor r6(1010001) => r2 will be 1010111

0000_001_011_111_101 // r7 = r1 (10110)nor r3(1000101)=> r7 will be 1010111

/*

registers:

00000000000000000000000000000000

00000000000000000000000000010110

r2: 00000000000000000000000001010111

00000000000000000000000001000101

11111111111111111111111110111000

11111111111111111111111110101000

00000000000000000000000001010001

r7:00000000000000000000000001010111

*/

0010_101_001_111111 andi R1,R5,111111(in binary) r1 will become 101000

0010_001_011_110111 andi R3,R1,110111(in binary) r3 will become 100000

/*

registers:

00000000000000000000000000000000

r1: 00000000000000000000000000101000

00000000000000000000000001010111

r3: 00000000000000000000000000100000

11111111111111111111111110111000

11111111111111111111111110101000

00000000000000000000000001010001

00000000000000000000000001010111

*/

0011_101_001_111000 ori R1,R5,111000(in binary) r1 will become
11111111111111111111111110111000

0011_001_011_101010 ori R3,R1,101010(in binary) r3 will become
11111111111111111111111110111010

/*

registers:

00000000000000000000000000000000

r1: 11111111111111111111111110111000

00000000000000000000000001010111

r3: 11111111111111111111111110111010

11111111111111111111111110111000

11111111111111111111111110101000

00000000000000000000000001010001

00000000000000000000000001010111

*/

0100_101_001_111000 nori R1,R5(11111111111111111111111110101000),111000(in binary) r1 will become 1000111

0100_001_011_100010 nori R3,R1(1000111),100010(in binary) r3 will become
11111111111111111111111110011000

/*

registers:

00000000000000000000000000000000

00000000000000000000000001000111

00000000000000000000000001010111

11111111111111111111111110011000

11111111111111111111111110111000

11111111111111111111111110101000

00000000000000000000000001010001

00000000000000000000000001010111

*/

0111_001_100_111000 slti R4,R1(1000111),111000(in binary) r4 will become 0

0111_100_101_100010 slti R5,R4(0),100010(in binary) r5 will become 1

/*

registers:

00000000000000000000000000000000

00000000000000000000000001000111

00000000000000000000000001010111

11111111111111111111111110011000

r4: 00000000000000000000000000000000

r5: 00000000000000000000000000000001

00000000000000000000000001010001

00000000000000000000000001010111

*/

1001_101_111_000111 sw R7,R5(7), Mem[7 + 1(r5 = 5)] = Mem[8] = 1010111

/*

Memory:

[0] = 00000000000000000000000000000000

[1] = 00000000000000000000000000000001

[2] = 00000000000000000000000000000010

[3] = 00000000000000000000000000000011

[4] = 00000000000000000000000000000100

[5] = 00000000000000000000000000000101

[6] = 00000000000000000000000000000110

[7] = 00000000000000000000000000000001

[8] = 00000000000000000000000001010111

*/

1000_101_001_000111 lw R1,R5(7), R1 = 1010111

/*

registers:

00000000000000000000000000000000

R1: 00000000000000000000000001010111

00000000000000000000000001010111

11111111111111111111111110011000

00000000000000000000000000000000

00000000000000000000000000000001

00000000000000000000000001010001

00000000000000000000000001010111

*/

//ORI

0011_011_001_111111 // ORI R1, R3, 111111 in binary

/*

R3 = 11111111111111111111111110011000

Imm= 00000000000000000000000000111111

ORI ----------

R1 = 11111111111111111111111110111111

*/

//ORI but trying to change R0

0011_011_000_111111 // ORI R0, R3, 111111 in binary

/*

R3 = 11111111111111111111111110011000

Imm= 00000000000000000000000000111111

ORI ----------

R0 != 11111111111111111111111110111111, R0 will stay as 0 because, R0 is always zero.

*/

/*

Registers:

R0: 00000000000000000000000000000000

R1: 11111111111111111111111110111111

R2: 00000000000000000000000001010111

R3: 11111111111111111111111110011000

R4: 00000000000000000000000000000000

R5: 00000000000000000000000000000001

R6: 00000000000000000000000001010001

R7: 00000000000000000000000001010111

*/