8-bit Microprocessor

EE2016

Akshay Anand
EE16B046

SUMMARY

The microprocessor that I designed in XILINX ISE using verilog is a standard 8-bit microprocessor having 8-bit data bus inside, 5-bit address bus (address space of 32 words) and 28 bit instruction size. Each instruction has a 4-bit instruction opcode and three 8-bit operands. The ALU inside the microprocessor has a 3-bit opcode with a total of 6 operations.

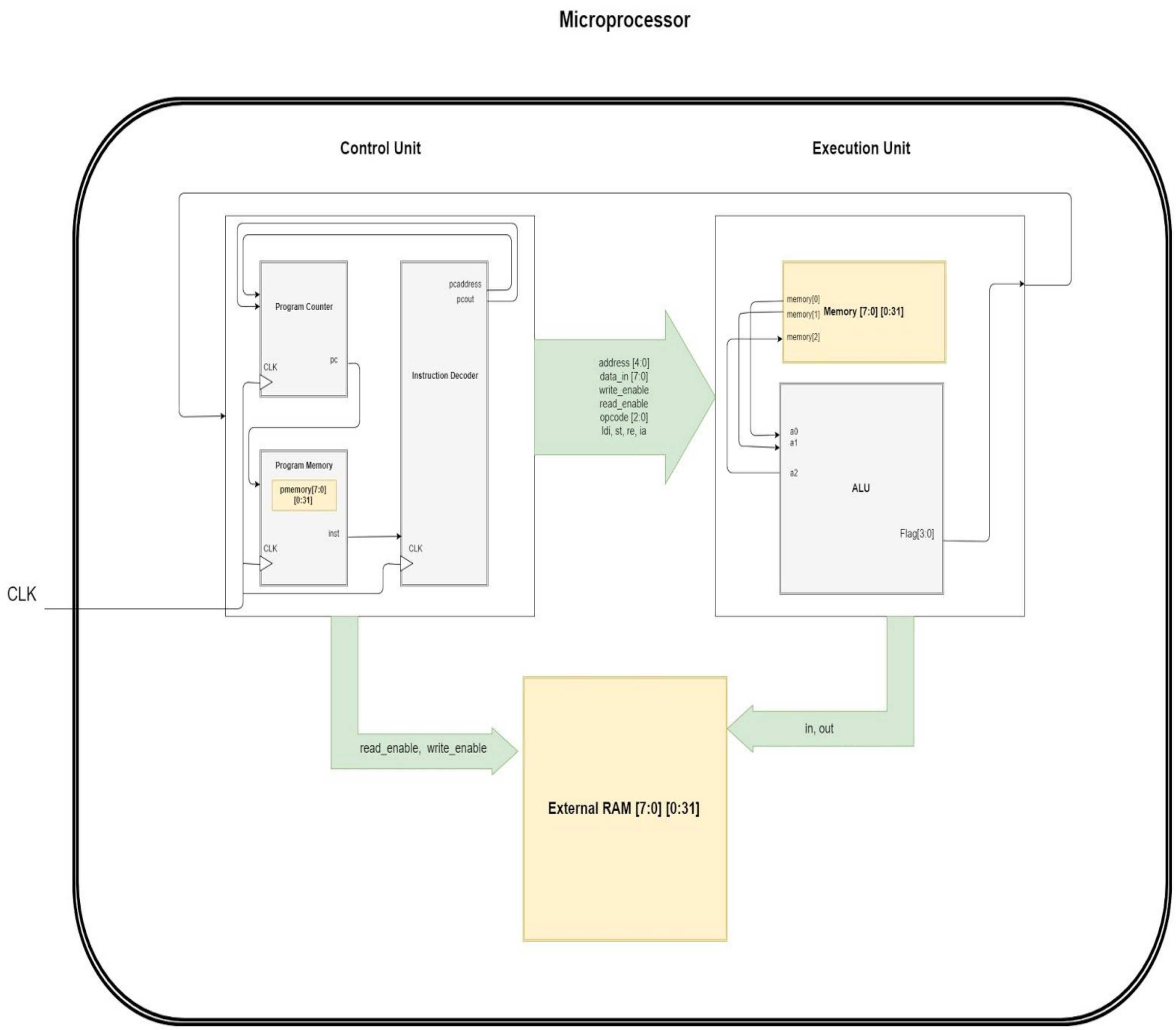
The internal register in Execution Unit has 32 registers with each register having a size of 8-bits. The external memory (RAM) also has the same size. The program memory which holds all the instructions has a 28 bit word length and can store a total of 32 instructions.

Operations performed by microprocessor

1. MOV Ri Rj (instruction opcode 0000) :- Move value of Ri internal register to Rj of internal register.
2. LDI Ri value (instruction opcode 0001) :- Load internal register Ri with a value.
3. ILDI Ri value (instruction opcode 0011) :- Load R [Ri] with a value.
4. LOAD Ri Rj (instruction opcode 1110) :- Read from Ri of external memory and write in Rj of internal memory.
5. ST Ri Rj (instruction opcode 0010) :- Read from Ri of internal memory and write in Rj of external memory.
6. INC Ri :- Increment value in internal memory Ri by 1.
7. DEC Ri :- Decrement value in internal memory Ri by 1.
8. BRNZ {location} (instruction opcode 0110) :- If zero flag not set jump to {location}: region in code, else continue normally.
9. BRNN {location} (instruction opcode 0101) :- If negative flag not set jump to {location}: region in code, else continue normally.
10. BRNO {location} (instruction opcode 0100) :- If overflow/carry flag not set, jump to {location}: region in code, else continue normally.
11. BRNU {location} (instruction opcode 0111) :- If underflow (shift right overflow) flag not set, jump to {location}: region in code, else continue normally.
12. END (instruction opcode 1111) :- End the program
13. ADD Ri Rj Rk (instruction opcode 1000)
14. SUB Ri Rj Rk (instruction opcode 1001)
15. AND Ri Rj Rk (instruction opcode 1010)
16. OR Ri Rj Rk (instruction opcode 1011)
17. SLEFT Ri Rj Rk (instruction opcode 1100)
18. SRIGHT Ri Rj Rk (instruction opcode 1101)

In the above 6 ALU operations, Ri and Rj are the 2 operands for the respective ALU and result is stored in Rk.

Block Diagram representing the microprocessor



Code

- Verilog Code

```
`timescale 1ns / 1ps
module microprocessor(clk,ldi,write_enable,read_enable,st);
wire[4:0] address;
wire[7:0] data_in;
wire [2:0] opcode;
wire [3:0] flag;
output wire write_enable,read_enable,ldi;
wire re;
output wire st;
wire ia;
input wire clk;
wire[7:0] out;
reg[7:0] ram[0:31];
initial begin
ram[0] = 5;    //this number is the source for base of exponent in the power program and the number
for the factorial program
ram[1] = 3;           //this is the power for the power program
end
reg[7:0] in;
cu C(clk,address,data_in,write_enable,read_enable,opcode,ldi,st,ia,flag,re);
eu E(address,data_in,write_enable,read_enable,opcode,ldi,out,st,ia,flag,re,in);

always @(*) begin
            if(read_enable&re) begin
                             in = ram[address];
                 end
        if (write_enable&st) begin
            ram[address] = out;
        end
   end

Endmodule

/* Program to find factorial of number stored in external memory(ram[0]) and store result in ram[5]
pmemory[0]=28'b1110_00000000_00000011_00000000;
pmemory[1]=28'b0001_00001000_00000001_00000000;
pmemory[2]=28'b0001_00000111_00000000_00000000;
pmemory[3]=28'b1001_00000011_00001000_00000110;
pmemory[4]=28'b0000_00000110_00000100_00000000;
pmemory[5]=28'b1000_00000111_00000011_00000111;
pmemory[6]=28'b0001_00011111_00000001_00000000;
pmemory[7]=28'b1001_00000100_00011111_00000100;
pmemory[8]=28'b0110_00001001_00000101_00000000;
pmemory[9]=28'b0000_00000111_00000011_00000000;
pmemory[10]=28'b0001_00000111_00000000_00000000;
pmemory[11]=28'b0001_00011111_00000001_00000000;
pmemory[12]=28'b1001_00000110_00011111_00000110;
pmemory[13]=28'b0110_00001110_00000100_00000000;
pmemory[14]=28'b0010_00000011_00000101_00000000;
```

```
pmemory[15]=28'b1111_11111111_11111111_11111111;
*/

/* Program to find ram[0] ^ ram[1] and store in ram[5]
pmemory[0]=28'b1110_00000000_00000100_00000000;
pmemory[1]=28'b1110_00000001_00000101_00000000;
pmemory[2]=28'b0001_00011111_00000001_00000000;
pmemory[3]=28'b1001_00000101_00011111_00000101;
pmemory[4]=28'b0001_00000111_00000000_00000000;
pmemory[5]=28'b0000_00000100_00001000_00000000;
pmemory[6]=28'b0000_00000100_00000110_00000000;
pmemory[7]=28'b1000_00000111_00001000_00000111;
pmemory[8]=28'b0001_00011111_00000001_00000000;
pmemory[9]=28'b1001_00000110_00011111_00000110;
pmemory[10]=28'b0110_00001011_00000111_00000000;
pmemory[11]=28'b0000_00000111_00001000_00000000;
pmemory[12]=28'b0001_00000111_00000000_00000000;
pmemory[13]=28'b0001_00011111_00000001_00000000;
pmemory[14]=28'b1001_00000101_00011111_00000101;
pmemory[15]=28'b0110_00010000_00000110_00000000;
pmemory[16]=28'b0010_00001000_00000101_00000000;
pmemory[17]=28'b1111_11111111_11111111_11111111;
*/


module cu(clk,address,data_in,write_enable,read_enable,opcode,ldi,st,ia,flag,re);
input wire clk;
output wire[4:0] address;
output wire[7:0] data_in;
input wire[3:0] flag;
output [2:0] opcode;
output wire write_enable,read_enable,ldi,st,ia,re;
wire[4:0] pc;
wire [4:0] pcaddr;
wire[27:0] inst;
wire pcout;

pccounter pcc(clk,pcout,pc,pcaddr);
instrdecoder
id(inst,clk,address,data_in,write_enable,read_enable,opcode,ldi,st,ia,pcout,pcaddr,flag,re);
progmemory pm(clk,pc,inst);

endmodule

module pccounter(clk,pcout,pc,pcaddr);
input wire pcout,clk;
input wire[4:0] pcaddr;
output reg[4:0] pc;
initial begin
 pc = 0;
end
 always@(posedge (pcout)) begin
   if(pcaddr == 5'b11111)
        pc = pc + 1;
       else
        pc = pcaddr;
 end
endmodule

 module progmemory(clk, pc, inst);
```

```
       reg[27:0] pmemory[0:31]; output reg[27:0] inst; input wire[4:0]pc; input wire clk; initial
begin
pmemory[0]=28'b1110_00000000_00000100_00000000;
pmemory[1]=28'b1110_00000001_00000101_00000000;
pmemory[2]=28'b0001_00011111_00000001_00000000;
pmemory[3]=28'b1001_00000101_00011111_00000101;
pmemory[4]=28'b0001_00000111_00000000_00000000;
pmemory[5]=28'b0000_00000100_00001000_00000000;
pmemory[6]=28'b0000_00000100_00000110_00000000;
pmemory[7]=28'b1000_00000111_00001000_00000111;
pmemory[8]=28'b0001_00011111_00000001_00000000;
pmemory[9]=28'b1001_00000110_00011111_00000110;
pmemory[10]=28'b0110_00001011_00000111_00000000;
pmemory[11]=28'b0000_00000111_00001000_00000000;
pmemory[12]=28'b0001_00000111_00000000_00000000;
pmemory[13]=28'b0001_00011111_00000001_00000000;
pmemory[14]=28'b1001_00000101_00011111_00000101;
pmemory[15]=28'b0110_00010000_00000110_00000000;
pmemory[16]=28'b0010_00001000_00000101_00000000;
pmemory[17]=28'b1111_11111111_11111111_11111111;
end
              always @(*) begin
              inst = pmemory[pc]; end
 endmodule

module
instrdecoder(inst,clk,address,data_in,write_enable,read_enable,opcode,ldi,st,ia,pcout,pcaddr,flag,re);
input wire clk;
input wire[3:0] flag;
output reg pcout;
output reg [4:0] address,pcaddr;
output reg [7:0] data_in;
output reg [2:0] opcode;
output reg write_enable,read_enable,ldi,st,ia,re;
input wire[27:0] inst;
reg[3:0] s0,s1,s2,s3,s4,s5,s6,s7,state;
initial begin
 pcout = 0;
 s0 = 3'b000;
 s1 = 3'b001;
 s2 = 3'b010;
 s3 = 3'b011;
 s4 = 3'b100;
 s5 = 3'b101;
 s6 = 3'b110;
 s7 = 3'b111;
 state = s0;
 pcaddr = 5'b11111;
end
 always @(posedge(clk))
  begin
    ia = 0;write_enable = 0;read_enable = 0 ; ldi  = 0; st = 0;re=0;
        if(state == s6)
                state = s7;
              if(state == s5)
                state = s6;
              if(state == s4)
                state = s5;
          if(state == s3)
                state = s4;
              if(state == s2)
                state = s3;
              if(state == s1)
                state = s2;
              if(state == s0) begin
              pcaddr = 5'b11111;
                pcout = 0;
                state = s1;
                end

    case (inst[27:24])
      4'b0000  :     //MOV R1 R2 => move from r1 to r2
               begin
                 if (state == s1) begin
                 address = inst[23:16];
                     read_enable = 1;
                     write_enable = 0;
                     end
                     if(state == s2) begin
                     address = inst[15:8];
                     read_enable = 0;
                     write_enable = 1;
                     state = s0;
                     pcout = 1;
                     end
                     /*if(state == s3) begin
                     write_enable = 0;
                     pcout = 1;
                     state = s0;
                     end*/
               end

               4'b0001  :    //Load value to internal register
               begin
                 if(state == s1) begin
                 address = inst[23:16];
                     data_in = inst[15:8];
                     read_enable = 0;
                     write_enable= 0;
                     ldi = 1;
                     st = 0;
                     end
                     if(state == s2) begin
                     ldi = 0;
                     write_enable = 1;
                     state = s0;
                     pcout = 1;
                     end
                     /*if(state == s3) begin
                     write_enable= 0;
                     pcout = 1;
                     state = s0;
                     end*/
               end

               4'b0010  :    //Store value in external RAM
               begin
                 if(state == s1) begin
                 address = inst[23:16];
                     read_enable = 1;
                write_enable = 0;
                ldi = 0;
                st = 1;
                end
                if(state == s2) begin
                st = 1;
                address = inst[15:8];
                write_enable = 1;
                read_enable = 0;
                pcout = 1;
                state = s0;
                end
                /*if(state == s3) begin
                write_enable = 0;
                st = 0;
                pcout = 1;
                state = s0;
                end*/
          end

          4'b0011  :    //Indirect Addressing LOAD R(Rx) with a value
          begin
                if(state == s1) begin
            address = inst[23:16];
                read_enable = 1;
                write_enable = 0;
                end
                if(state == s2) begin
                read_enable = 0;
                ia = 1;
                end
                if(state == s3) begin
                data_in = inst[15:8];
                ia = 1;
                ldi = 1;
                end
                if(state == s4) begin
                ldi = 0;
                ia = 1;
                write_enable = 1;
                end
                if(state == s5) begin
                ia = 0;
                write_enable = 0;
                pcout = 1;
                state = s0;
                end
                /*if(state == s5) begin
                ia = 0;
                write_enable = 0;
                pcout = 1;
                state = s0;
                end*/
          end

         endcase

          if (inst[27:26] == 2'b01)    //program memory jump subject to flag value
           begin
            if(state == s1) begin
```

```
                    if(flag[inst[25:24]] == 1'b1)
                         begin
                               pcaddr = inst[23:16];
                             end
                        else begin
                          pcaddr = inst[15:8];
                        end
                      end
                      if(state == s2) begin
                      pcout = 1;
                      state = s0;
                      end

                 end

                if (inst[27])
                 begin
                  if(inst[27:24] == 4'b1111) begin   //end program
                        state = s7;
                        pcout = 0;
                      end
                      else if(inst[27:24] == 4'b1110) begin   // Load value from external ram
                         if(state == s1) begin
                             address = inst[23:16];
                             write_enable = 0;
                             read_enable = 1;
                             re = 1;
                             end
                             if(state == s2) begin
                             address = inst[15:8];
                             write_enable = 1;
                             re = 1;
                             end
                             if(state == s3) begin
                             pcout = 1;
                             state = s0;
                             end
                      end
                      else begin     //Do ALU operations
                  if(state == s1) begin
                  write_enable = 0;
                  read_enable = 1;
                      address = inst[23:16];
                      end
                      if(state == s2) begin
                      read_enable = 0; write_enable = 1; address = 4'b0000;
                      end
                      if(state == s3) begin
                      read_enable = 1; write_enable = 0; address = inst[15:8];
                      end
                      if(state == s4) begin
                      read_enable = 0; write_enable = 1; address = 4'b0001;
                      end
                      if(state == s5) begin
                      write_enable = 0;
                      opcode = inst[26:24];
                      end
                      if(state == s6) begin
                      read_enable = 1;address = 4'b0010;
                      end
                      if(state == s7) begin
                      read_enable = 0; write_enable = 1; address = inst[7:0];
                      pcout = 1;
                      state = s0;
                      end
                      end
                 end

  end

endmodule


module eu(address,data_in,write_enable,read_enable,opcode,ldi,out,st,ia,flag,re,in);
input wire[4:0] address;
input wire[7:0] data_in,in;
input [2:0] opcode;
input write_enable,read_enable,ldi,st,ia,re;
output reg [7:0] out;
output reg [3:0] flag;
wire[7:0] a0,a1;
wire [7:0] a2;
wire [3:0] fl;
reg [7:0] memory [0:31];
reg [7:0] bus;
reg [4:0] addressbus;

assign a0 = memory[0];
assign a1 = memory[1];

alu a(a0,a1,a2,opcode,fl);


    always @(*) begin
          if(!ia) begin
                      addressbus = address;
              end
              if(ldi&!st) begin
                             bus = data_in;
                      end
                      if(write_enable&!st&re) begin
                             memory[address] = in;
                      end
        if (write_enable&!st&!re) begin
            memory[addressbus] = bus;
        end
                 if (read_enable&!st&!re) begin
                    bus = memory[addressbus];
                 end
                 if (read_enable&st&!re) begin
                    out = memory[addressbus];
                 end
                 if (ia&!ldi&!read_enable&!write_enable) begin
                        addressbus = bus;
                 end
                  memory[2] = a2;
                      flag <= fl;
    end
//always @(*) begin
```

```
//end

endmodule



module alu(a0,a1,a2,opcode,fl);
   input wire[7:0] a0, a1;
        input wire [2:0] opcode;
   output wire [7:0] a2;
        output wire [3:0] fl;
        reg[3:0] flag;
        reg[7:0] res,ctr;
   always @(*)
  begin
    case (opcode)
      3'b000  : begin
                                                  res = a0 + a1;
                                                  if(res < a0 | res < a1) begin
                                                         flag = 4'b0001;
                                                  end
                                                  else if(res == 0) begin
                                                         flag = 4'b0100;
                                                  end
                                                  else begin
                                                         flag = 4'b0000;
                                                  end
                                            end
      3'b001  : begin
                                                  res = a0 - a1;
                                                  if(a1 > a0) begin
                                                         flag = 4'b0010;
                                                  end
                                                  else if(res == 0) begin
                                                     flag = 4'b0100;
                                                  end
                                                  else begin
                                                         flag = 4'b0000;
                                                  end
                                            end
               3'b010  : begin
                                                  res = a0 & a1;
                                                  if(res == 0) begin
                                                         flag = 4'b0100;
                                                  end
                                                  else begin
                                                         flag = 4'b0000;
                                                  end
                                            end
               3'b011  : begin
                                                  res = a0 | a1;
                                                  if(res == 0) begin
                                                         flag = 4'b0100;
                                                  end
                                                  else begin
                                                         flag = 4'b0000;
                                                  end
                                            end
               3'b100  : begin
                                                  res = a0 << a1;
```

```
                                             if(res < (a0 << (a1-1) ) ) begin
                                                    flag = 4'b0001;
                                             end
                                       end
               3'b101  :  begin
                                             ctr = a0 >> (a1-1);
                                             res = a0 >> a1;
                                             if(ctr[0] == 1'b1 ) begin
                                                    flag = 4'b1000;
                                             end
                                       end
               default : res = a2;
     endcase
   end
        assign a2 = res;
        assign fl = flag;
endmodule
```

- C++ code for Assembler

```
// Assembler.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include<iostream>
#include<conio.h>
#include<fstream>
#include<string>
#include<vector>
using namespace std;
/*
  MOV Ri(from)  Rj(to)
  LDI Ri(address of internal register) value
  ILDI R(Rx) value
  LOAD Ri(address of external ram from which to read) Xj(Address of internal register to which
to write)
  ST Ri(read address of internal register) Rj(write address of register in external ram)
  INC Ri (increment value in internal register)
  DEC Ri (increment value in internal register)
  BRNZ location (Branch if not zero)
  BRNN location (Branch if not negative)
  BRNO location (Branch if not overflow)
  BRNU location (Branch if not underflow (shift right underflow) )
  ( Following alu operations: read from Ri and Rj and store result in Rk)
  ADD Ri Rj Rk
  SUB Ri Rj Rk
  AND Ri Rj Rk
  OR Ri Rj Rk
  SLEFT Ri Rj Rk
  SRIGHT Ri Rj Rk
  END (end the program)
*/
fstream fout, fin;
```

```
void load(string &res) {
	string op1;
	int op2;
	int i, ctr;
	res.append("0001");
	fin >> op1;
	res.push_back('_');
	if (op1.size() == 2)
		ctr = op1[1] - '0';
	else
		ctr = (op1[1] - '0') * 10 + op1[2] - '0';
	i = 7;
	while (i >= 0)
	{
		if (ctr - pow(2, i) >= 0)
		{
			ctr -= pow(2, i);
			res.push_back('1');
		}
		else
			res.push_back('0');
		i--;
	}
	res.push_back('_');
	fin >> op2;
	i = 7;
	ctr = op2;
	while(i>=0)
	{
		if (ctr - pow(2, i) >= 0)
		{
			ctr -= pow(2, i);
			res.push_back('1');
		}
		else
			res.push_back('0');
		i--;
	}
}

void move(string &res) {
	string op;
	int i, ctr;
	res.append("0000");
	fin >> op;
	res.push_back('_');
	if (op.size() == 2)
		ctr = op[1] - '0';
	else
		ctr = (op[1] - '0') * 10 + op[2] - '0';
	i = 7;
	while (i >= 0)
	{
		if (ctr - pow(2, i) >= 0)
		{
			ctr -= pow(2, i);
			res.push_back('1');
		}
		else
			res.push_back('0');
```

```
                i--;
        }
        fin >> op;
        res.push_back('_');
        if (op.size() == 2)
                ctr = op[1] - '0';
        else
                ctr = (op[1] - '0') * 10 + op[2] - '0';
        i = 7;
        while (i >= 0)
        {
                if (ctr - pow(2, i) >= 0)
                {
                        ctr -= pow(2, i);
                        res.push_back('1');
                }
                else
                        res.push_back('0');
                i--;
        }
}

void alu(string &res)
{
        string op;
        int i, ctr;
        fin >> op;
        res.push_back('_');
        if (op.size() == 2)
                ctr = op[1] - '0';
        else
                ctr = (op[1] - '0') * 10 + op[2] - '0';
        i = 7;
        while (i >= 0)
        {
                if (ctr - pow(2, i) >= 0)
                {
                        ctr -= pow(2, i);
                        res.push_back('1');
                }
                else
                        res.push_back('0');
                i--;
        }
        fin >> op;
        res.push_back('_');
        if (op.size() == 2)
                ctr = op[1] - '0';
        else
                ctr = (op[1] - '0') * 10 + op[2] - '0';
        i = 7;
        while (i >= 0)
        {
                if (ctr - pow(2, i) >= 0)
                {
                        ctr -= pow(2, i);
                        res.push_back('1');
                }
                else
                        res.push_back('0');
```

```
			i--;
		}
		fin >> op;
		res.push_back('_');
		if (op.size() == 2)
			ctr = op[1] - '0';
		else
			ctr = (op[1] - '0') * 10 + op[2] - '0';
		i = 7;
		while (i >= 0)
		{
			if (ctr - pow(2, i) >= 0)
			{
				ctr -= pow(2, i);
				res.push_back('1');
			}
			else
				res.push_back('0');
			i--;
		}
}

void inc_dec(string &res)
{
		string op,buf;
		int i, ctr;
		fin >> op;
		res.push_back('_');
		if (op.size() == 2)
			ctr = op[1] - '0';
		else
			ctr = (op[1] - '0') * 10 + op[2] - '0';
		i = 7;
		while (i >= 0)
		{
			if (ctr - pow(2, i) >= 0)
			{
				ctr -= pow(2, i);
				buf.push_back('1');
			}
			else
				buf.push_back('0');
			i--;
		}
		res.append(buf);
		res.push_back('_');
		res.append("00011111");
		res.push_back('_');
		res.append(buf);
}


void main()
{
		string s,op1;
		int num,ctr,ctr1,k,j;
		fout.open("../../progmemory.v", fstream::in | fstream::out | fstream::trunc);
		fin.open("../../input.txt", fstream::in | fstream::out | fstream::app);
		int i = 0,n;
		bool cmd = false;
```

```
        vector<string> res;
        vector<pair<string, int> > stack;
        while (fin >> s)
        {
                string ctr;
                if (s.compare("LDI") == 0)
                {
                        load(ctr);
                        ctr.append("_00000000");
                        ctr.push_back(';');
                }
                else if (s.compare("ILDI") == 0)
                {
                        load(ctr);
                        ctr[2] = '1';
                        ctr.append("_00000000");
                        ctr.push_back(';');
                }
                else if (s.compare("INC") == 0)
                {
                        ctr.append("0001_00011111_00000001_00000000;");
                        res.push_back(ctr);
                        i++;
                        ctr = "";
                        ctr.append("1000");
                        inc_dec(ctr);
                        ctr.push_back(';');
                }
                else if (s.compare("DEC") == 0)
                {
                        ctr.append("0001_00011111_00000001_00000000;");
                        res.push_back(ctr);
                        i++;
                        ctr = "";
                        ctr.append("1001");
                        inc_dec(ctr);
                        ctr.push_back(';');
                }
                else if (s.compare("MOV") == 0)
                {
                        move(ctr);
                        ctr.append("_00000000");
                        ctr.push_back(';');
                }
                else if (s.compare("ST") == 0)
                {
                        move(ctr);
                        ctr[0] = ctr[1] = ctr[3] = '0';
                        ctr[2] = '1';
                        ctr.append("_00000000");
                        ctr.push_back(';');
                }
                else if (s[0] == 'B')
                {
                        ctr.append("01");
                        if (s[3] == 'Z')
                                ctr.append("10");
                        else if (s[3] == 'N')
                                ctr.append("01");
                        else if (s[3] == 'O')
                ctr.append("00");
        else if (s[3] == 'U')
                ctr.append("11");
        fin >> op1;
        ctr.push_back('_');
        k = 7;
        j = i + 1;
        while (k >= 0)
        {
                if (j - pow(2, k) >= 0)
                {
                        j -= pow(2, k);
                        ctr.push_back('1');
                }
                else
                        ctr.push_back('0');
                k--;
        }
        bool ans = false;
        for (j = 0; j < stack.size(); j++) {
                if (stack[j].first.compare(op1) == 0) {
                        ctr.push_back('_');
                        k = 7;
                        ans = true;
                        while (k >= 0)
                        {
                                if (stack[j].second - pow(2, k) >= 0)
                                {
                                        stack[j].second -= pow(2, k);
                                        ctr.push_back('1');
                                }
                                else
                                        ctr.push_back('0');
                                k--;
                        }
                }
        }
        if (!ans)
        {
                cout << "Tag doesn't exist in the program" << endl;
                exit(0);
        }
        ctr.append("_00000000");
        ctr.push_back(';');
}
else if (s.compare("END") == 0)
{
        ctr.append("1111_11111111_11111111_11111111;");
        res.push_back(ctr);
        break;
}
else if (s.compare("ADD") == 0)
        {
                ctr.append("1000");
                alu(ctr);
                ctr.push_back(';');
        }
        else if (s.compare("SUB") == 0)
        {
                ctr.append("1001");
```

```
                                alu(ctr);
                                ctr.push_back(';');
                        }
                        else if (s.compare("AND") == 0)
                        {
                                ctr.append("1010");
                                alu(ctr);
                                ctr.push_back(';');
                        }
                        else if (s.compare("OR") == 0)
                        {
                                ctr.append("1011");
                                alu(ctr);
                                ctr.push_back(';');
                        }
                        else if (s.compare("SLEFT") == 0)
                        {
                                ctr.append("1100");
                                alu(ctr);
                                ctr.push_back(';');
                        }
                        else if (s.compare("SRIGHT") == 0)
                        {
                                ctr.append("1101");
                                alu(ctr);
                                ctr.push_back(';');
                        }
                        else if (s.compare("LOAD") == 0)
                        {
                                move(ctr);
                                ctr[0] = '1'; ctr[1] = '1'; ctr[2] = '1'; ctr[3] = '0';
                                ctr.append("_00000000");
                                ctr.push_back(';');
                        }
                        else {
                                s.pop_back();
                                stack.push_back(make_pair(s, i));
                                i--;
                        }
                        i++;

                        if(!ctr.empty())
                        res.push_back(ctr);
        }
        fout << "`timescale 1ns / 1ps \n module progmemory(clk, pc, inst); \n    reg[27:0]
pmemory[0:31]; output reg[27:0] inst; input wire[4:0]pc; input wire clk; initial begin " <<
endl;
        for (i = 0; i < res.size(); i++)
        {
                fout<<"pmemory["<<i<<"]=28'b"<<res[i]<<endl;
        }
        fout << "end \n              always @(*) begin \n         inst = pmemory[pc]; end \n
endmodule " << endl;
        fout.close();
        fin.close();
        _getch;
}
```

Example Assembler codes

- Factorial of a number stored in memory

Assembler Code

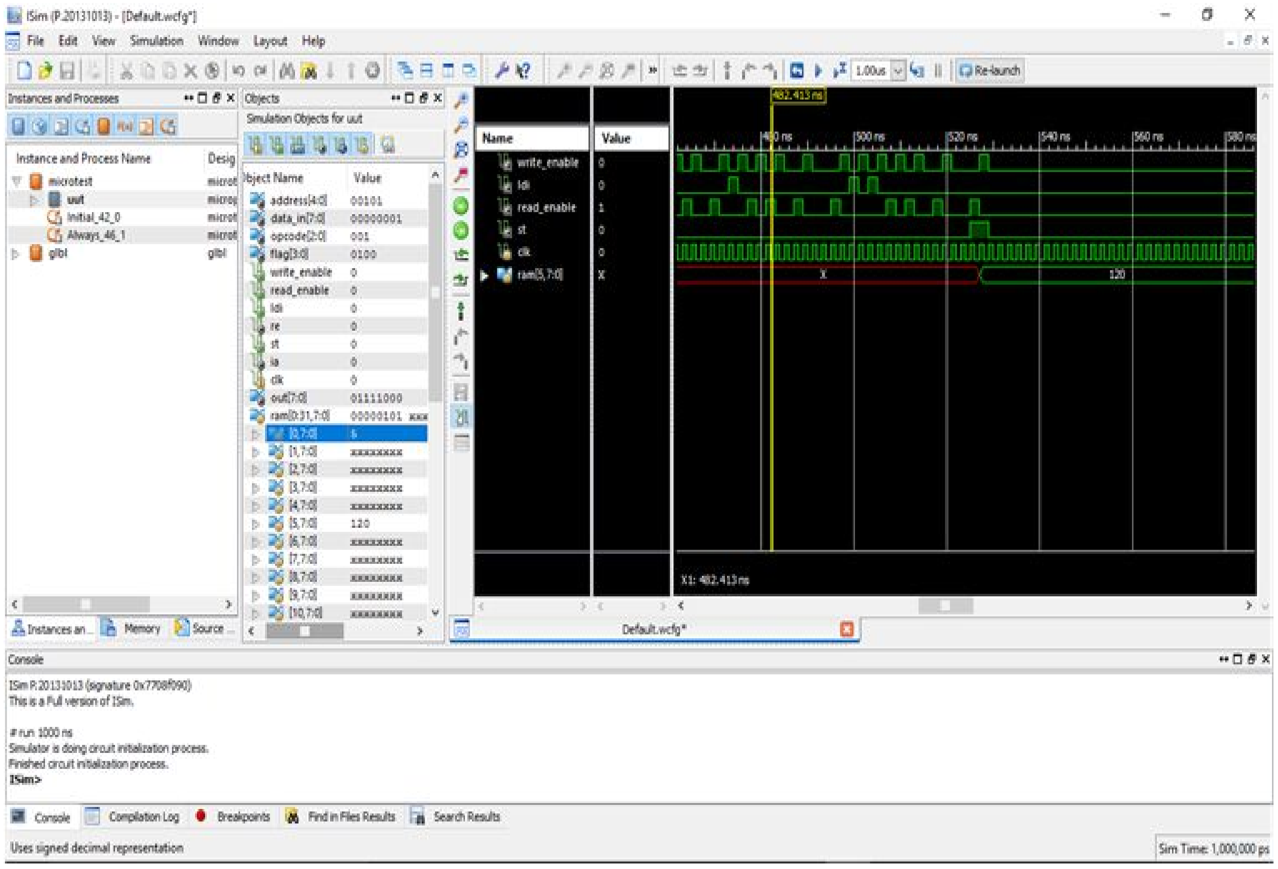
```
LOAD R0 R3
LDI R8 1
LDI R7 0
SUB R3 R8 R6
flag:
MOV R6 R4
main:
ADD R7 R3 R7
DEC R4
BRNZ main
MOV R7 R3
LDI R7 0
DEC R6
BRNZ flag
ST R3 R5
END
```

Output of Assembler

```
pmemory[0]=28'b1110_00000000_00000011_00000000;
pmemory[1]=28'b0001_00001000_00000001_00000000;
pmemory[2]=28'b0001_00000111_00000000_00000000;
pmemory[3]=28'b1001_00000011_00001000_00000110;
pmemory[4]=28'b0000_00000110_00000100_00000000;
pmemory[5]=28'b1000_00000111_00000011_00000111;
pmemory[6]=28'b0001_00011111_00000001_00000000;
pmemory[7]=28'b1001_00000100_00011111_00000100;
```

```
pmemory[8]=28'b0110_00001001_00000101_00000000;
pmemory[9]=28'b0000_00000111_00000011_00000000;
pmemory[10]=28'b0001_00000111_00000000_00000000;
pmemory[11]=28'b0001_00011111_00000001_00000000;
pmemory[12]=28'b1001_00000110_00011111_00000110;
pmemory[13]=28'b0110_00001110_00000100_00000000;
pmemory[14]=28'b0010_00000011_00000101_00000000;
pmemory[15]=28'b1111_11111111_11111111_11111111;
```

The following simulation screenshot has 5 stored in R0 of external memory and 5! Stored in R5 of external memory.



```
Object Name      Value
address[4:0]     00101
data_in[7:0]     00000001
opcode[2:0]      001
flag[3:0]        0100
write_enable     0
read_enable      0
ldi              0
re               0
st               0
ia               0
clk              0
out[7:0]         01111000
ram[0:31,7:0]    00000101  xxx
[0,7:0]          5
[1,7:0]          xxxxxxxx
[2,7:0]          xxxxxxxx
[3,7:0]          xxxxxxxx
[4,7:0]          xxxxxxxx
[5,7:0]          120
[6,7:0]          xxxxxxxx
[7,7:0]          xxxxxxxx
[8,7:0]          xxxxxxxx
[9,7:0]          xxxxxxxx
[10,7:0]         xxxxxxxx
```

```
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Uses signed decimal representation

- Power of a number raised to another number, both stored in memory

Assembly Code

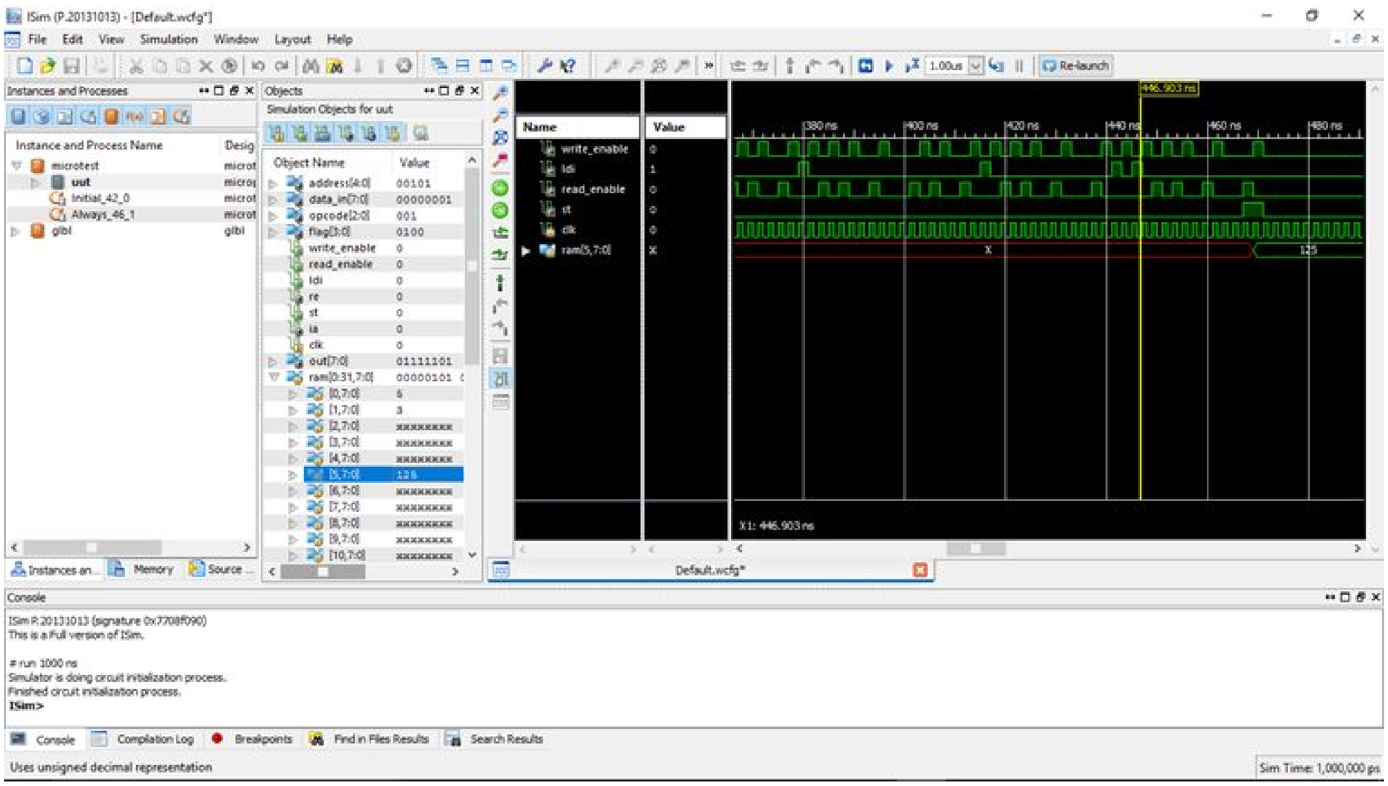
```
LOAD R0 R4
LOAD R1 R5
DEC R5
LDI R7 0
MOV R4 R8
flag:
MOV R4 R6
repeat:
ADD R7 R8 R7
DEC R6
BRNZ repeat
MOV R7 R8
LDI R7 0
DEC R5
BRNZ flag
ST R8 R5
END
```

Assembler Output

```
pmemory[0]=28'b1110_00000000_00000100_00000000;
pmemory[1]=28'b1110_00000001_00000101_00000000;
pmemory[2]=28'b0001_00011111_00000001_00000000;
pmemory[3]=28'b1001_00000101_00011111_00000101;
pmemory[4]=28'b0001_00000111_00000000_00000000;
pmemory[5]=28'b0000_00000100_00001000_00000000;
pmemory[6]=28'b0000_00000100_00000110_00000000;
pmemory[7]=28'b1000_00000111_00001000_00000111;
pmemory[8]=28'b0001_00011111_00000001_00000000;
pmemory[9]=28'b1001_00000110_00011111_00000110;
pmemory[10]=28'b0110_00001011_00000111_00000000;
pmemory[11]=28'b0000_00000111_00001000_00000000;
```

```
pmemory[12]=28'b0001_00000111_00000000_00000000;
pmemory[13]=28'b0001_00011111_00000001_00000000;
pmemory[14]=28'b1001_00000101_00011111_00000101;
pmemory[15]=28'b0110_00010000_00000110_00000000;
pmemory[16]=28'b0010_00001000_00000101_00000000;
pmemory[17]=28'b1111_11111111_11111111_11111111;
```

In the following screenshot, 5 is stored in R0 of external memory and 3 in R1 of external memory and result 5^3 is stored in R5 of external memory.



Object Name	Value
address[4:0]	00101
data_in[7:0]	00000001
opcode[2:0]	001
flag[3:0]	0100
write_enable	0
read_enable	0
ldi	0
re	0
st	0
ia	0
clk	0
out[7:0]	01111101
ram[0:31,7:0]	00000101
[0,7:0]	5
[1,7:0]	3
[2,7:0]	xxxxxxxx
[3,7:0]	xxxxxxxx
[4,7:0]	xxxxxxxx
[5,7:0]	125
[6,7:0]	xxxxxxxx
[7,7:0]	xxxxxxxx
[8,7:0]	xxxxxxxx
[9,7:0]	xxxxxxxx
[10,7:0]	xxxxxxxx

Name	Value
write_enable	0
ldi	1
read_enable	0
st	0
clk	0
ram[5,7:0]	X

```
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.

# run 1000 ns
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Uses unsigned decimal representation

Sim Time: 1,000,000 ps