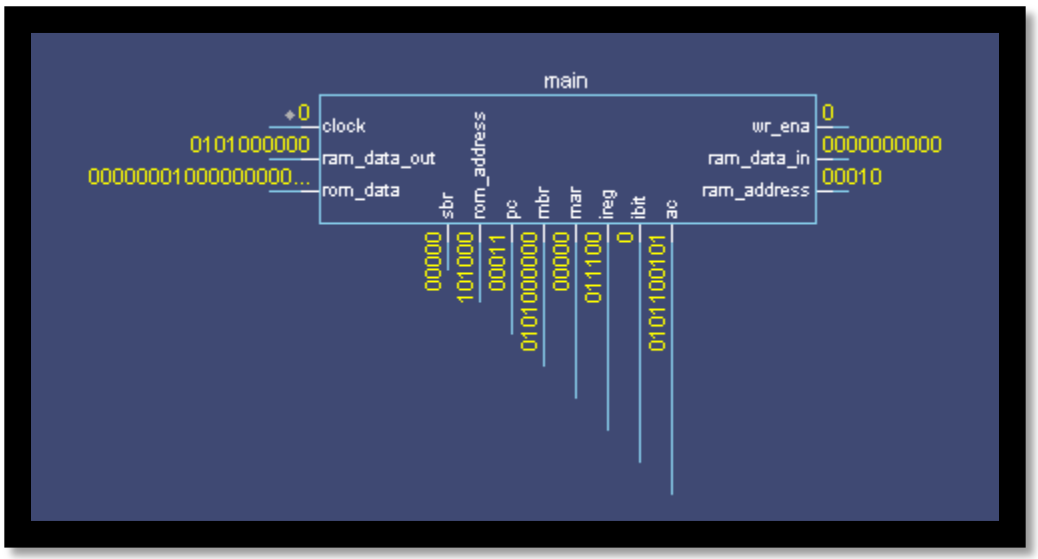
Processor Report

Shubham Jain

327/CO/10

Processor

- The following is a micro-program controlled processor which loads the instructions from RAM one by one and executes them according to the control instruction retrieved from the ROM.
- Along with RAM, ROM it has registers, MBR, MAR, ACC, Ibit and SBR to store the current address.
- It has a clock to synchronize, and RAM and ROM to fetch the instruction and microinstructions.
- Initially it executes the fetch instruction from the ROM which brings the instruction from RAM to MBR and maps it.
- According to the op-code, the microinstructions are executed.
- All the mathematical operations are performed using the accumulator as an output temporary register.



ROM

- Rom receives the address and returns the corresponding micro-instruction into the data signal.
- The ROM made has 18 bit microinstruction while the address in 6 bit address.

F1-4bit	F2-4bit	C-2bit	J-2bit	ADD-6bit

F1-Control instructions1

F2-Control instructions2

C-Condition

J-Jump Type

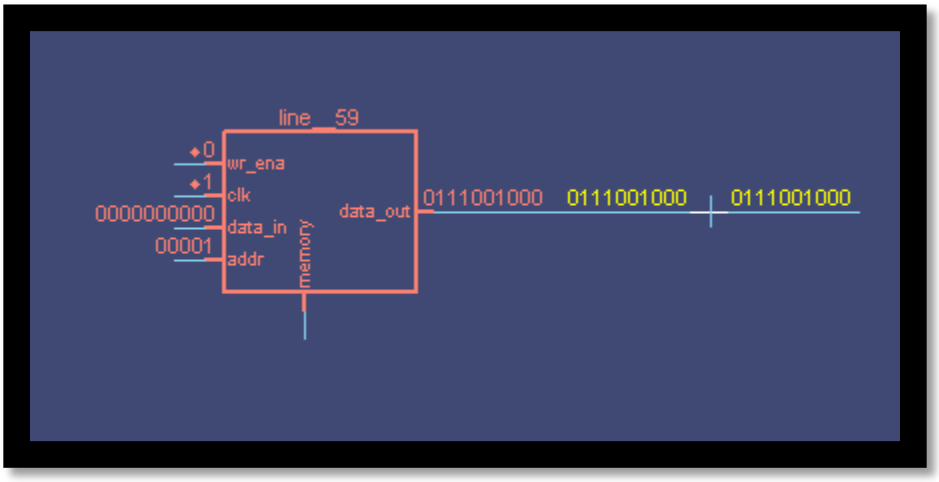
Condition		Jump	
00	Unconditional	00	Jump
01	Indirect	01	Call
10	--	10	Map
11	--	11	Ret

- For each macroinstruction we have 4 microinstructions. Hence an op-code of 4 bits.

F1		F2	
0000	No-op	0000	No-op
0001	Read1	0001	inc A
1111	Read2	0010	AC2MBR
0010	Add	0011	Write
0011	And	0100	MBR2AC
0100	Or	0101	PC2SBR
0101	Nor	110	SBR2PC
0110	Nand	0111	PC2MAR
0111	Xor	1000	INCPC
1000	Xnor	1001	MBR2PC
1001	Cla	1010	MAR2PC

RAM

- Ram has four ports:-Data-in,Data-out,Wr-ena,Add
- When write enable is 0(off) then ram reads the address from the address port and passes the equivalent data in data-out port.
- When write enable in 1(on) then ram reads the data from the data-in port and write it in the address read from the address port.



- This RAM has memory of 5 bit i.e. 32 locations. And 10 bit instruction
- The instruction set consists of:-

Indirect bit-1bit	Opcode-4bit	Address-5bit

The macroinstructions sets available are:-

S.No	Bit	Instruction
1.	0000	--
2.	0001	--
3.	0010	ADD
4.	0011	AND
5.	0100	OR
6.	0101	NOR
7.	0110	NAND
8.	0111	XOR
9.	1000	XNOR
10.	1001	CLA
11.	1010	INCA
12.	1011	STA
13.	1100	LDA
14.	1101	BUN
15.	1110	BSA
16.	1111	RET

Code:

A small program is written in RAM in order to illustrate all the operations this processor can perform.

PROCESSOR

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use STD.textio.all;
use IEEE.std_logic_textio.all;

entity processor is
  generic(
        rom_a:integer:=6;
        rom_d:integer:=18;
        ram_a:integer:=5;
        ram_d:integer:=10);
end entity;

architecture processor_arch of processor is

  component rom is
          generic (words: integer :=64;
                  bits: integer :=18);
          port(
                  addr: in std_logic_vector(5 downto 0);
                  data: out std_logic_vector(bits-1 downto 0) );
  end component;

  component ram is
          generic (bits:integer:=1+4+5;
                          words: integer:=5);
          port(clk, wr_ena: in std_logic;
                  addr:in std_logic_vector(words-1 downto 0);
```

```
                data_in:in std_logic_vector(bits-1 downto 0);
                data_out:out std_logic_vector(bits-1 downto 0));
end component;

signal clock: std_logic:='1';
signal mar: std_logic_vector(ram_a-1 downto 0):=(others=>'0');
signal pc: std_logic_vector(ram_a-1 downto 0):=(others=>'0');
signal ireg: std_logic_vector(rom_a-1 downto 0):=(others=>'0');
signal sbr: std_logic_vector(ram_a-1 downto 0):=(others=>'0');
signal mbr: std_logic_vector(ram_d-1 downto 0):=(others=>'0');
signal ac: std_logic_vector(ram_d-1 downto 0):=(others=>'0');
signal ibit: std_logic:='0';
signal rom_address: std_logic_vector(rom_a-1 downto 0):=(others=>'0');
signal rom_data: std_logic_vector(rom_d-1 downto 0):=(others=>'0');
signal ram_address: std_logic_vector(ram_a-1 downto 0);
signal ram_data_in: std_logic_vector(ram_d-1 downto 0):=(others=>'0');
signal ram_data_out: std_logic_vector(ram_d-1 downto 0):=(others=>'0');
signal wr_ena: std_logic:='0';

begin
 p1: ram port map(clock,wr_ena,ram_address,ram_data_in,ram_data_out);
 p2: rom port map(rom_address, rom_data);

 clk1: process
 begin
  clock<=not clock after 20 ns;
  wait for 20 ns;
 end process;

 main:process(clock)
 begin
 if(clock' event and clock = '1') then
 --F1
 case rom_data(17 downto 14) is
  when "0001" =>
   wr_ena<='0';
   ram_address<=mar;
  when "0010" =>
   ac<=mbr+ac;
  when "0011" =>
   ac<= mbr AND ac;
  when "0100" =>
   ac<= mbr OR ac;
  when "0101" =>
```

```
    ac<= mbr NOR ac;
  when "0110" =>
    ac<= mbr NAND ac;
  when "0111" =>
    ac<= mbr XOR ac;
  when "1000" =>
    ac<= mbr XNOR ac;
  when "1001" =>
    ac<=(others =>'0');
  when "1111" =>
    mbr<=ram_data_out;
  when others =>
end case;


case rom_data(13 downto 10) is
  when "0000" =>

  when "0001" =>
    ac<=ac+'1';
  when "0010" =>
    mbr<=ac;
  when "0011" =>
    wr_ena<='1';
    ram_address<=mar;
    ram_data_in<=mbr;
  when "0100"=>
    ac<=mbr;
  when "0101"=>
    sbr<=pc;
  when "0110"=>
    pc<=sbr+'1';
  when "0111"=>
    mar<=pc;
  when "1000"=>
    pc<=pc+'1';
  when "1001"=>
    mar<=mbr(4 downto 0);
  when "1010"=>
    pc<=mar;
  when others =>
end case;

case rom_data(7 downto 6) is
```

```
    when "00" =>
      rom_address<=rom_data(5 downto 0);
   --indirect specific
    when "01" =>
      if (ibit = '1') then
        ireg<=rom_address;
        rom_address<=rom_data(5 downto 0);
      else
        rom_address<=rom_address+'1';
      end if;
    when "10" =>
      ibit<=mbr(9);
      rom_address<=(others =>'0');
      rom_address(5 downto 2)<=mbr(8 downto 5);
      --write the map instruction acc to the instruction set....
    when "11" =>
      rom_address<=ireg+'1';
    when others =>
   end case;

  end if;
   end process;

  end architecture;
```

ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity rom is
        generic (words: integer :=64;
                bits: integer :=18);
        port(
                addr: in std_logic_vector(5 downto 0);
                data: out std_logic_vector(bits-1 downto 0) );
end rom;
```

```
architecture rom_arch of rom is
	type vector_array is array(0 to words-1) of std_logic_vector(bits-1 downto 0);
	constant memory: vector_array:=(
		 --fetch
		 "000001110000000001",
		 "000110000000000010",
		"111100000000000011",
		"000010010010000000",

		--indirect
		"000100000000000101",
		"111100000000000110",
		"000010010011000000",
		"000000000000000000",
		--add
		"000000000101000100",
		“000100000000001010",
		"111100000000001011",
		"001000000000000000",
		--and
		"000000000101000100",
		"000100000000001110",
		"111100000000001111",
		"001100000000000000",
		--or
		"000000000101000100",
		"000100000000010010",
		"111100000000010011",
		"010000000000000000",
		--nor
		"000000000101000100",
		"000100000000010110",
		"111100000000010111",
		"010100000000000000",
		--nand
		"000000000101000100",
		"000100000000011010",
		"111100000000011011",
		"011000000000000000",
		--xor
		"000000000101000100",
		"000100000000011110",
```

"111100000000011111",
"011100000000000000",
--xnor
"000000000101000100",
"000100000000100010",
"111100000000100011",
"100000000000000000",
--cla
"100100000000000000",
"000000000000000000",
"000000000000000000",
"000000000000000000",
--inca
"000000010000000000",
"000000000000000000",
"000000000000000000",
"000000000000000000",
--sta
"000000000101000100",
"000000100000101110",
"000000110000000000",
“000000000000000000",
--lda
"000000000101000100",
"000100000000110010",
"111100000000110011",
"000001000000000000",
--bun
"000000000101000100",
"000010100000000000",
"000000000000000000",
"000000000000000000",
--bsa
"000001010000111001",
"000000000101000100",
"000010100000000000",
"000000000000000000",
-ret
"000001100000000000",
"000000000000000000",
"000000000000000000",
"000000000000000000");

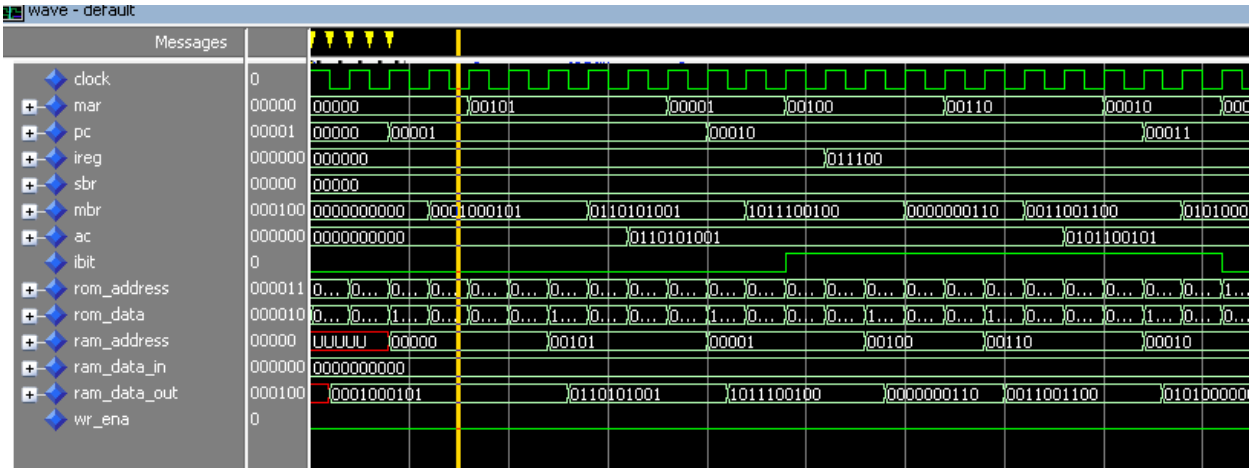
```
begin
	data<=memory(conv_integer(addr));
end architecture;
```

RAM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ram is
	generic (bits:integer:=1+4+5;
				words: integer:=5);
	port(clk, wr_ena: in std_logic;
		addr:in std_logic_vector(words-1 downto 0);
		data_in:in std_logic_vector(bits-1 downto 0);
		data_out:out std_logic_vector(bits-1 downto 0));
end entity;

architecture ram_arch of ram is
	type vector_array is array (0 to 32-1) of std_logic_vector(bits-1 downto 0);
	signal addd:integer;
	signal memory: vector_array:=( "0001000101",--ADD 00101
				"1011100100",--I AND 00100
				"0101000000",--INC A
				"0111001000",--BSA

				"0000000110",--ADRESS DATA 2
				"0110101001",--DATA1
				"0011001100",--DATA2
				"0000000000",--LOC where acc is stored

				"0101100111",--STA at 00111
				"0100100000",--CLA
				"0110000111",--LDA 00111
				"0000000000",
```

```
                        "0000000000",
                        "0000000000",
                        "0000000000",
                        "0000000000",

                        "0000000000",
                        "0000000000",
                        "0000000000",
                        "0000000000",

                        "0000000000",
                        "0000000000",
                        "0000000000",
                        "0000000000",

                        "0000000000",
                        "0000000000",
                        "0000000000",
                        "0000000000",

                        "0000000000",
                        "0000000000",
                        "0000000000",
                        "0000000000");
        begin
         process(wr_ena,clk)
  begin
         if(wr_ena='0' and clk'event) then
                  data_out <=memory(conv_integer(addr));
         elsif(wr_ena='1') then
                   if (clk'event) then
                     memory(conv_integer(addr)) <= data_in;
                   end if;
        end if;
        end process;
end architecture;
```

OUTPUT


wave - default
Messages
clock
mar
pc
ireg
sbr
mbr
ac
ibit
rom_address
rom_data
ram_address
ram_data_in
ram_data_out
wr_ena