Advanced VLSI Design

EEL-6990

Professor: Dr. Subbarao Wunnava

Florida International University

Department of Electrical and Computer Engineering

Project I

4-Bit Slice Microprocessor

Submitted by

Mildred C. Zabawa **Vivek Jayaram**

On 10/28/2003

Table of Contents

1 Objective

Our intentions is to implement a 4-bit slice microprocessor capable of preloading memory with digital words (4 bits wide) as well as perform the indicated instruction. Based on the operation code selected, we can perform the following operations: Add, Subtract, Shift Left, Shift Right, Rotate Left, Rotate Right, AND, OR, Invert, MOVA, MOVB, MOVC, MOVD.

2 Theory of 4-bit slice Microprocessor

The main functional blocks are as follows:

- A 16-word by four bit port RAM.
- Four registers (RegA, RegB, RegC, RegD)
- An ALU selector which select two inputs. The D (Destination Input) will always be rega_data. The S(Source Input) will be either regb_data, regc_data, regd_data based on the selected output from the MUX.
- The MUX module used *instruction[11:8]* as the select input. If select is '0010' then regb_data is set to signal S(Source Input). If select is '0100' then regc_data is set to signal S. If Select is '1000' then regd_data is set to signal S.
- A 4-bit ALU capable of doing arithmetic, logical, and bitwise functions on the selected source and destination words.
- An instruction decoder is used to decide whether to load the ALU output into RegA or whether to load the read data from the RAM. It also load RegB, RegC, and RegD from memory if the MOVB, MOVC, or MOVD operations occurs.

Port	Type	Bit Width	Description
reset	In	1	Reset signal
clk	In	1	Clock signal
instruction	In	12	Instruction word
addr	In	4	Address input to RAM
data	In	4	Data input to RAM
we	In	1	Write/Read Enable for RAM
C	Out	4	Data output from chip
C4	Out	1	Carry output from ALU

The instruction set has a 12-bit instruction, which has three 3-bit fields whose functions are as follows:

Field position	Description
instruction[3:0]	Opcode Select: ‘0000’: ADD ‘0001’: SUB ‘0010’: CMP ‘0011’: SHL ‘0100’: SHR ‘0101’: ROL ‘0110’: ROR ‘0111’: AND ‘1000’: OR ‘1001’: NOT ‘1010’: MOVA ‘1011’: MOVB ‘1100’: MOVC ‘1101’: MOVD
instruction[7:4]	Address
instruction[11:8]	Control Destination Selector for Mux Module to select input source (S) to ALU ‘0001’: Write to Register A ‘0010’: Write to Register B ‘0100’: Write to Register C ‘1000’: Write to Register D

The ALU functions are the following:

Instruction Type	Bit Field instruction[3:0]	ALU Function(Output --> C) CO = 0 where A, B, C, D are the register
ADD	0000	A: A + B
SUB	0001	A: A - B
CMP	0010	Do not update A; A - B, set flag
SHL	0011	A: A shift left 1
SHR	0100	A: A shift right 1
ROL	0101	A: A rotate left 1
ROR	0110	A: A rotate right 1
AND	0111	A: A and B
OR	1000	A: A or B
NOT	1001	A: not A
MOVA	1010	A: mem[addr]
MOVB	1011	B: mem[addr]
MOVC	1100	C: mem[addr]
MOVD	1101	D: mem[addr]

The following is the block diagram of 4-bit bit-slice microprocessor based on the modules described above:

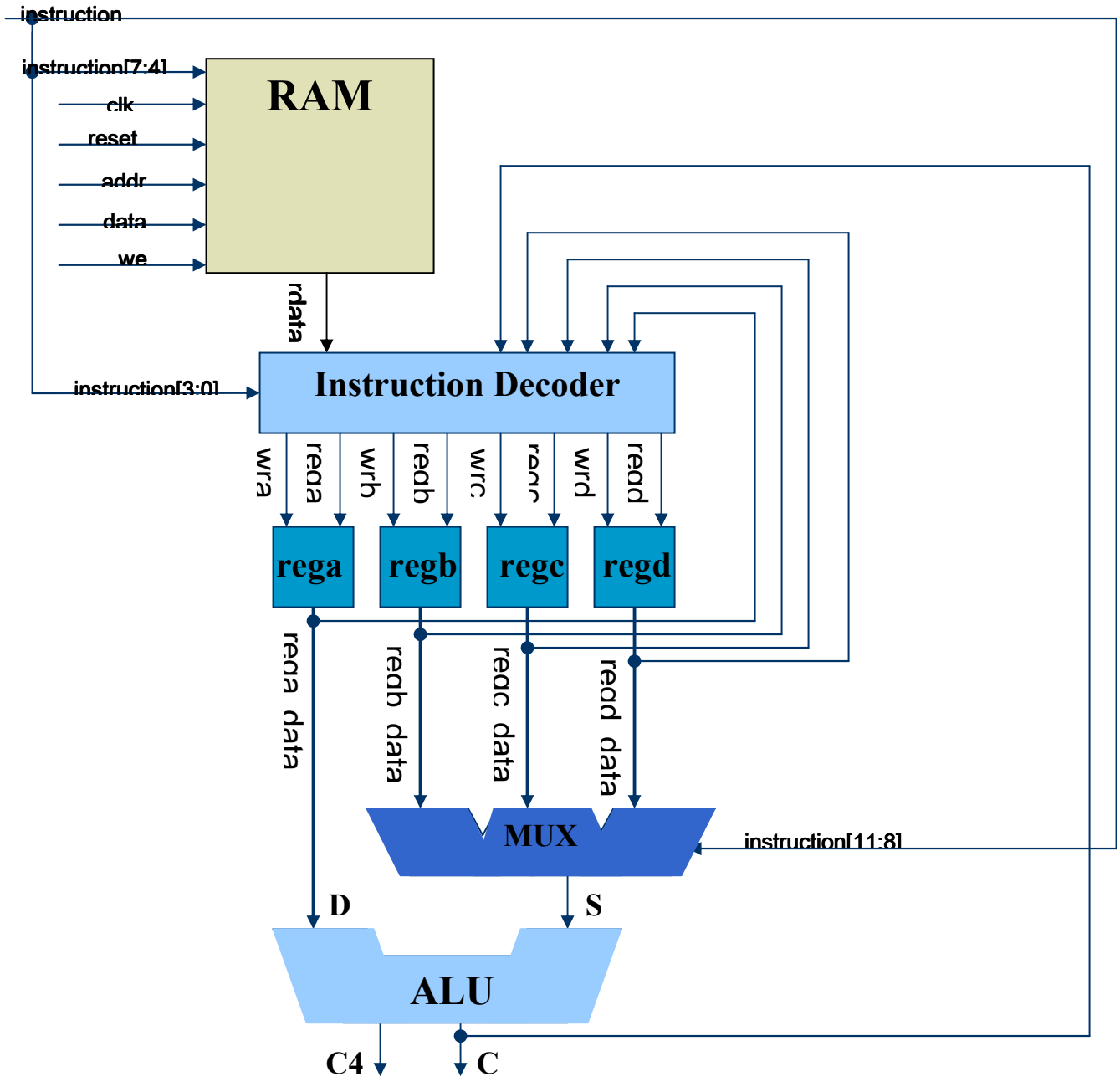


Figure2.1: 4-bit Slice Microprocessor Block Diagram

3 Test Vector Generation Methodology

C++ API Implementation contains method calls used in the test vector C++ source code for stimulus test vector generation.

Test Vector C++ Source Code are written to generate stimulus test vector to test the hardness of the 4-bit slice microprocessor features. The test vector is read by the microprocessor RTL testbench.

Hardware RTL Simulation used Mentor Graphics tools to read, compile, and simulation input/output signals for analysis based on the stimulus test vector.

Mentor Graphics Waveform files where capture based on the select signals under investigation.

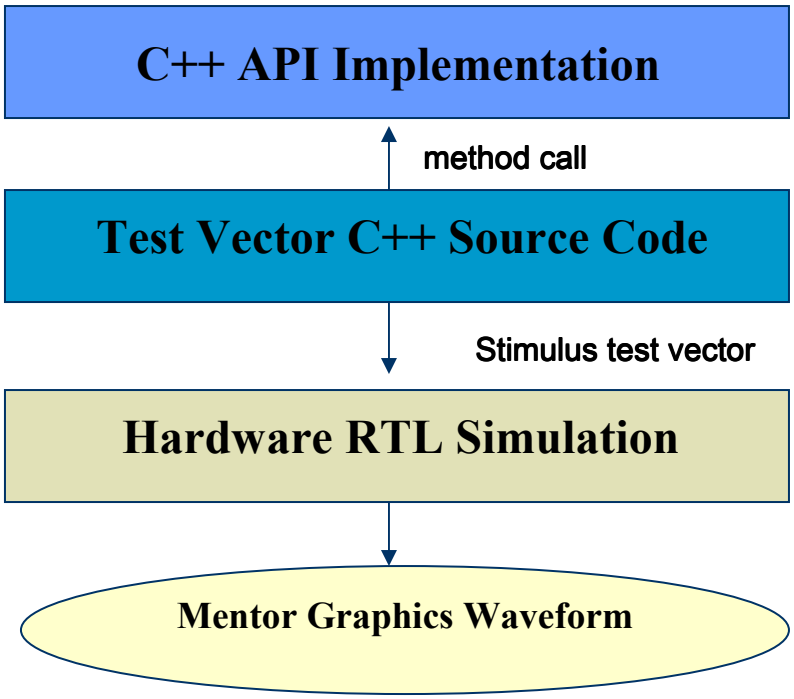


Figure 3.1: Hardware Verification Methodology

3.1 C++ API (Instruction Set)

The following Instruction Set for the 4-bit slice Microprocessor was designed to generate test vectors (stimulus file) to test the hardness of the hardware design. Using this C++ API, we can write multiple test vectors to test the various corner cases. The following function description are described below:

```
/*****************************************************************************/
/* Title          :  C++ style                                               */
/* Filename       :  instruction_api.h                                       */
/* Authors        :  Mildred C. Zabawa and Vivek Jayaram
*/
/* Description    :  This gives the description of the instruction_api class  */
/*****************************************************************************/
#include <iostream.h>
#include <fstream.h>
#include <cstdlib>

class instruction_api{

        void openISS_File(char *filename);
        void closeISS_File();
        void openRef_File(char *filename);
        ofstream iss_file;
        ofstream ref_file;

        enum opcode {ADD, SUB, CMP, SHL, SHR, ROL, ROR, AND, OR, NOT,
                MOVA,  MOVB, MOVC, MOVD};

        void writeBinToFile(int data, int size);

        int addr_add;
        int addr_sub;
        int addr_cmp;
        int addr_shl;
        int addr_shr;
        int addr_rol;
        int addr_ror;
        int addr_and;
        int addr_or;
        int addr_not;
        void LoadMemory(int addr, int X, int Y);
        public:
```

```
        void Execute(int opcode);
        void setADD(int addr, int AX, int BX);
        void setSUB(int addr, int AX, int BX);
        void setCMP(int addr, int AX, int BX);
        void setSHL(int addr, int AX, int CX);
        void setSHR(int addr, int AX, int CX);
        void setROL(int addr, int AX, int CX);
        void setROR(int addr, int AX, int CX);
        void setAND(int addr, int AX, int BX);
        void setOR(int addr, int AX, int BX);
        void setNOT(int addr, int AX);
        void setMOVA(int addr);
        void setMOVB(int addr);
        void setMOVC(int addr);
        void setMOVD(int addr);
        int getaddrADD();
        int getaddrSUB();
        int getaddrCMP();
        int getaddrSHL();
        int getaddrSHR();
        int getaddrROL();
        int getaddrROR();
        int getaddrAND();
        int getaddrOR();
        int getaddrNOT();

        instruction_api(char *filename);
        ~instruction_api();

};
```

```
/*****************************************************************************/
/* Title              :  C++ style                                           */
/* Filename           :  instruction_api.c                                   */
/* Authors            :  Mildred C. Zabawa and Vivek Jayaram
*/
/* Description        :  This gives the functions of the instruction_api class*/
/*****************************************************************************/
#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include "test_api.h"

/*****************************************************************************/
/* Function           :  instruction_api                                     */
/* Description        :  This is the constructor function which is used to initialize and open a test */
/*                       vector stimulus file                                */
/*****************************************************************************/
instruction_api::instruction_api(char *filename)
{
  openISS_File(filename);
}

/*****************************************************************************/
/* Function           :  ~instruction_api                                    */
/* Description        :  This is the destructor function which is used to close a test */
/*                       vector stimulus file                                */
/*****************************************************************************/
instruction_api::~instruction_api()
{
  closeISS_File();
}

/*****************************************************************************/
/* Function           :  closeISS_File                                       */
/* Description        :  This will close the  test  vector stimulus file     */
/*****************************************************************************/
void instruction_api::closeISS_File()
{
  iss_file.close();
}
```

```
/*****************************************************************************/
/* Function              :  openISS_File                                     */
/* Description           :  This will open the  test  vector stimulus file   */
/*****************************************************************************/
void instruction_api::openISS_File(char *filename)
{
        iss_file.open(filename,  ios::out);
        if(!iss_file){
                cout << "Cannot open " << filename << endl;
                exit(1);
        }
}
/*****************************************************************************/
/* Function              :  writeBinToFile                                   */
/* Description           :  This will convert a integer data into a binary format based on the unsigned */
/*                          logic size                                       */
/*****************************************************************************/
void instruction_api::writeBinToFile(int data, int size)
{
        int mask_value = 0x1;
        for (int i = (size - 1); i >= 0; i--)
        {
                iss_file.width(1);
                iss_file << ((data >> i)&0x1);
        }
}
/*****************************************************************************/
/* Function              :  LoadMemory                                       */
/* Description           :  This will write the memory address, X, and Y data in a specific format */
/*                          to be interpreted by the testbench vhdl code.    */
/*                          Format:   <w> <addr> <dataX>                     */
/*                                    <w > <addr + 1> < dataY>               */
/*****************************************************************************/
void instruction_api::LoadMemory(int addr, int X, int Y)
{
  iss_file << "w ";
  writeBinToFile(addr, 4);
  iss_file << " ";
  writeBinToFile(X, 4);
  iss_file << endl;
  iss_file << "w ";
  writeBinToFile(addr + 1, 4);
  iss_file << " ";
  writeBinToFile(Y, 4);
  iss_file << endl;
}
```

```
/*****************************************************************************/
/* Function          :  setADD                                               */
/* Description       :  This will load the AX, BX data to a specific memory address. Also, it will */
/*                      save the address of the AX of the ADD function. The address of BX is the */
/*                      address AX + 1.                                      */
/*****************************************************************************/
void instruction_api::setADD(int addr, int AX, int BX)
{
  LoadMemory(addr, AX, BX);
  addr_add = addr;
}
/*****************************************************************************/
/* Function          :  setSUB                                               */
/* Description       :  This will load the AX, BX data to a specific memory address. Also, it will */
/*                      save the address of the AX of the SUB function. The address of BX is the */
/*                      address AX + 1.                                      */
/*****************************************************************************/
void instruction_api::setSUB(int addr, int AX, int BX)
{
  LoadMemory(addr, AX, BX);
  addr_sub = addr;
}
/*****************************************************************************/
/* Function          :  setCMP                                               */
/* Description       :  This will load the AX, BX data to a specific memory address. Also, it will */
/*                      save the address of the AX of the CMP function. The address of BX is the */
/*                      address AX + 1.                                      */
/*****************************************************************************/
void instruction_api::setCMP(int addr, int AX, int BX)
{
  LoadMemory(addr, AX, BX);
  addr_cmp = addr;
}
/*****************************************************************************/
/* Function          :  setSHL                                               */
/* Description       :  This will load the AX, CX data to a specific memory address. Also, it will */
/*                      save the address of the AX of the SHL function. The address of CX is the */
/*                      address AX + 1.                                      */
/*****************************************************************************/
void instruction_api::setSHL(int addr, int AX, int CX)
{
  LoadMemory(addr, AX, CX);
  addr_shl = addr;
}
```

```
/*****************************************************************************/
/* Function          : setSHR                                                */
/* Description       : This will load the AX, CX data to a specific memory address. Also, it will */
/*                     save the address of the AX of the SHR function. The address of CX is the */
/*                     address AX + 1.                                       */
/*****************************************************************************/

void instruction_api::setSHR(int addr, int AX, int CX)
{
  LoadMemory(addr, AX, CX);
  addr_shr = addr;
}
/*****************************************************************************/
/* Function          : setROL                                                */
/* Description       : This will load the AX, CX data to a specific memory address. Also, it will */
/*                     save the address of the AX of the ROL function. The address of CX is the */
/*                     address AX + 1.                                       */
/*****************************************************************************/
void instruction_api::setROL(int addr, int AX, int CX)
{
  LoadMemory(addr, AX, CX);
  addr_rol = addr;
}
/*****************************************************************************/
/* Function          : setROR                                                */
/* Description       : This will load the AX, CX data to a specific memory address. Also, it will */
/*                     save the address of the AX of the ROR function. The address of CX is the */
/*                     address AX + 1.                                       */
/*****************************************************************************/
void instruction_api::setROR(int addr, int AX, int CX)
{
  LoadMemory(addr, AX, CX);
  addr_ror = addr;
}
/*****************************************************************************/
/* Function          : setAND                                                */
/* Description       : This will load the AX, BX data to a specific memory address. Also, it will */
/*                     save the address of the AX of the AND function. The address of BX is the */
/*                     address AX + 1.                                       */
/*****************************************************************************/
void instruction_api::setAND(int addr, int AX, int BX)
{
  LoadMemory(addr, AX, BX);
  addr_and = addr;
}
```

```
/*****************************************************************************/
/* Function          : setOR                                                 */
/* Description       : This will load the AX, BX data to a specific memory address. Also, it will  */
/*                     save the address of the AX of the OR function. The address of BX is the  */
/*                     address AX + 1.                                         */
/*****************************************************************************/
void instruction_api::setOR(int addr, int AX, int BX)
{
  LoadMemory(addr, AX, BX);
  addr_and = addr;
}

/*****************************************************************************/
/* Function          : setNOT                                                */
/* Description       : This will load the AX data to a specific memory address. Also, it will  */
/*                     save the address of the AX of the NOT function.         */
/*****************************************************************************/
void instruction_api::setNOT(int addr, int AX)
{
  LoadMemory(addr, AX, 0);
  addr_not = addr;
}

/*****************************************************************************/
/* Function          : getaddrADD                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the ADD operation                                   */
/*****************************************************************************/
int instruction_api::getaddrADD()
{
  return(addr_add);
}

/*****************************************************************************/
/* Function          : getaddrSUB                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the SUB operation                                   */
/*****************************************************************************/
int instruction_api::getaddrSUB()
{
  return(addr_sub);
}
```

```
/*****************************************************************************/
/* Function          : getaddrCMP                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the CMP operation                                 */
/*****************************************************************************/
int instruction_api::getaddrCMP()
{
  return(addr_cmp);
}

/*****************************************************************************/
/* Function          : getaddrSHL                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the SHL operation                                 */
/*****************************************************************************/
int instruction_api::getaddrSHL()
{
        return(addr_shl);
}

/*****************************************************************************/
/* Function          : getaddrSHR                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the SHR operation                                 */
/*****************************************************************************/
int instruction_api::getaddrSHR()
{
  return(addr_shr);
}

/*****************************************************************************/
/* Function          : getaddrROL                                            */
/* Description       : This will get the address location of the first element data stored in memory */
/*                     for the ROL operation                                 */
/*****************************************************************************/
int instruction_api::getaddrROL()
{
        return(addr_rol);
}
```

```
/*****************************************************************************/
/* Function        : getaddrROR                                              */
/* Description     : This will get the address location of the first element data stored in memory */
/*                   for the ROR operation                                   */
/*****************************************************************************/
int instruction_api::getaddrROR()
{
        return(addr_ror);
}

/*****************************************************************************/
/* Function        : getaddrAND                                              */
/* Description     : This will get the address location of the first element data stored in memory */
/*                   for the AND operation                                   */
/*****************************************************************************/
int instruction_api::getaddrAND()
{
  return(addr_and);
}

/*****************************************************************************/
/* Function        : getaddrOR                                               */
/* Description     : This will get the address location of the first element data stored in memory */
/*                   for the OR operation                                    */
/*****************************************************************************/
int instruction_api::getaddrOR()
{
  return(addr_and);
}

/*****************************************************************************/
/* Function        : getaddrNOT                                              */
/* Description     : This will get the address location of the first element data stored in memory */
/*                   for the NOT operation                                   */
/*****************************************************************************/
int instruction_api::getaddrNOT()
{
  return(addr_not);
}
```

```
/*****************************************************************************/
/* Function              :  setMOVA                                          */
/* Description           :  This will write the address of memory location to Register A */
/*****************************************************************************/
void instruction_api::setMOVA(int addr)
{
  iss_file << "i ";
  writeBinToFile(1,4);
  writeBinToFile(addr, 4);
  writeBinToFile(MOVA, 4);
  iss_file << endl;
}
/*****************************************************************************/
/* Function              :  setMOVB                                          */
/* Description           :  This will write the address of memory location to Register B */
/*****************************************************************************/
void instruction_api::setMOVB(int addr)
{
  iss_file << "i ";
  writeBinToFile(2,4);
  writeBinToFile(addr, 4);
  writeBinToFile(MOVB, 4);
  iss_file << endl;
}
/*****************************************************************************/
/* Function              :  setMOVC                                          */
/* Description           :  This will write the address of memory location to Register C */
/*****************************************************************************/
void instruction_api::setMOVC(int addr)
{

  iss_file << "i ";

  writeBinToFile(4,4);
  writeBinToFile(addr, 4);
  writeBinToFile(MOVC, 4);
  iss_file << endl;

}
```

```
/*****************************************************************************/
/* Function              :  setMOVD                                          */
/* Description           :  This will write the address of memory location to Register D */
/*****************************************************************************/
void instruction_api::setMOVD(int addr)
{

  iss_file << "i ";

  writeBinToFile(8, 4);
  writeBinToFile(addr, 4);
  writeBinToFile(MOVD, 4);
  iss_file << endl;

}
/*****************************************************************************/
/* Function              :  Execute                                          */
/* Description           :  This will write the opcode to be perform         */
/*                          Format:  i  <addr><opcode>                       */
/*****************************************************************************/
void instruction_api::Execute(int opcode)
{
  iss_file << "i ";

  writeBinToFile(0, 8);
  writeBinToFile(opcode, 4);
  iss_file << endl;
}
```

3.2 C++ (Test Vector Source Code)

The following is an example of a Test Vector source code written in C++. This source code will generate a test vector stimulus file which will be read in by the microprocessor test bench. We can load memory into the RAM, set the A, B, C, D Registers using the MOV commands, and write the instruction and operation command. Based on this information we will generate a test vector.

```
/*****************************************************************************/
/* Title              :  C++ style                                           */
/* Filename           :  instruction_api.h                                   */
/* Authors            :  Mildred C. Zabawa and Vivek Jayaram
*/
/* Description        :  This gives the description of the instruction_api class */
/*****************************************************************************/
#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include "test_api.h"

int main ()
{
        enum opcode {ADD, SUB, CMP, SHL, SHR, ROL, ROR, AND, OR, NOT, MOVA,
                        MOVB, MOVC, MOVD};

        class instruction_api iss_api("test_opcode.in");

        //load memory
        iss_api.setADD(0x0, 0x5,0x7);
        iss_api.setSUB(0x2, 0x3,0x7);
        iss_api.setCMP(0x4, 0x3,0x2);
        iss_api.setSHL(0x6, 0xA,0x1);
        iss_api.setSHR(0x8, 0xE,0x2);
        iss_api.setROL(0xA, 0xB,0x3);
        iss_api.setROR(0xC, 0xD,0x2);
        iss_api.setAND(0xE, 0x3,0x4);

        //do instructions
        iss_api.setMOVA(iss_api.getaddrADD());          // A = 5
        iss_api.setMOVB(iss_api.getaddrADD() + 1);      // B = 7
        iss_api.Execute(ADD);                           // A = A + B = 12
        iss_api.Execute(SUB);                           // A = A - B = 5
        iss_api.setMOVA(iss_api.getaddrSUB());          // A = 3
```

```
        iss_api.setMOVB(iss_api.getaddrSUB() + 1);          // B = 7
        iss_api.Execute(SUB);
        iss_api.setMOVA(iss_api.getaddrCMP());
        iss_api.setMOVB(iss_api.getaddrCMP() + 1);
        iss_api.Execute(CMP);
        iss_api.setMOVA(iss_api.getaddrSHL());
        iss_api.setMOVC(iss_api.getaddrSHL() + 1);
        iss_api.Execute(SHL);
        iss_api.setMOVA(iss_api.getaddrSHR());
        iss_api.setMOVC(iss_api.getaddrSHR() + 1);
        iss_api.Execute(SHR);
        iss_api.setMOVA(iss_api.getaddrROL());
        iss_api.setMOVC(iss_api.getaddrROL() + 1);
        iss_api.Execute(ROL);
        iss_api.setMOVA(iss_api.getaddrROR());
        iss_api.setMOVC(iss_api.getaddrROR() + 1);
        iss_api.Execute(ROR);
        iss_api.setMOVA(iss_api.getaddrAND());
        iss_api.setMOVD(iss_api.getaddrAND() + 1);
        iss_api.Execute(AND);

        iss_api.setOR( 0x0, 0x3,0x5);
        iss_api.setNOT(0x2, 0xA);

        iss_api.setMOVA(iss_api.getaddrOR());
        iss_api.setMOVD(iss_api.getaddrOR() + 1);
        iss_api.Execute(OR);
        iss_api.setMOVA(iss_api.getaddrNOT());
        iss_api.Execute(NOT);

  return 0;
}
```

3.3 C++ Test Vector (Stimulus File)

A test vector stimulus file is generated to be read by the microprocessor test bench.

This format of this file is the following:

<w> <addr> <data> -- load memory to RAM
<i> <reg><addr> <opcode> -- instruction of opcode to be performed

The following is an example of a Test Vector Stimulus File:

w 0000 0101
w 0001 0111
w 0010 0011
w 0011 0111
w 0100 0011
w 0101 0010
w 0110 1010
w 0111 0011
w 1000 1110
w 1001 0010
w 1010 1011
w 1011 0011
w 1100 1101
w 1101 0010
w 1110 0011
w 1111 0100
i 000100001010
i 001000011011
i 000000000000
i 000000000001
i 000100101010
i 001000111011
i 000000000001
i 000101001010
i 001001011011
i 000000000010
i 000101101010
i 010001111100
i 000000000011
i 000000000011
i 000000000011
i 000110001010
i 010010011100
i 000000000100
i 000000000100
i 000110101010

i 010010111100
i 000000000101
i 000000000101
i 000000000101
i 000111001010
i 010011011100
i 000000000110
i 000000000110
i 000111101010
i 100011111101
i 000000000111
w 0000 0011
w 0001 0101
w 0010 1010
w 0011 0000
i 000100001010
i 100000011101
i 000000001000
i 000100101010
i 000000001001

4 Block diagrams, VHDL Programs, Simulated waveforms and Synthesized Schematics of 4-bit slice Microprocessor:

The following are the

4.1 TestBench for a 4-bit Slice Microprocessor

4.1.1 VHDL Program (4-bit slice microprocessor TestBench)

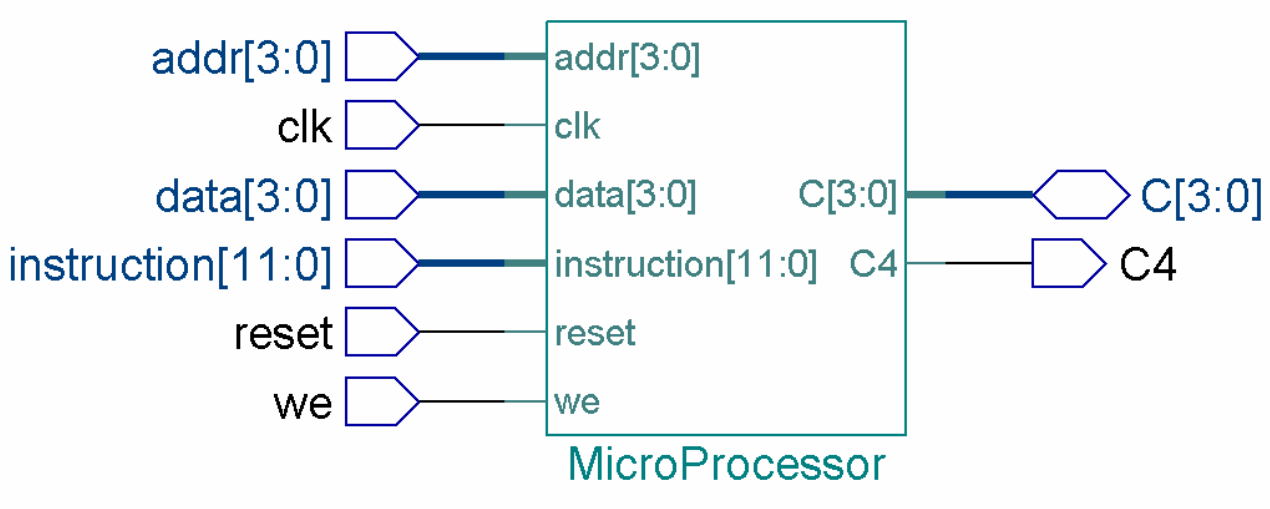
```
------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : MicroProcessor_TB.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   : This Program describes the behaviour of test bench
--                 for the Microprocessor.
------------------------------------------------------------------
--
------------------------------------------------------------------
library IEEE;
use IEEE.STD_Logic_1164.all;
use IEEE.Numeric_STD.all;


entity MicroProcessor_TB is
end MicroProcessor_TB;

architecture MicroPro_TB of MicroProcessor_TB is

component MicroProcessor
  port (reset      : in  std_logic;
        clk        : in  std_logic;
        instruction: in  unsigned(11 downto 0);
        addr       : in  unsigned(3 downto 0);
        data       : in  unsigned(3 downto 0);
        we         : in  std_logic;
       C           : inout unsigned(3 downto 0);
       C4         : out std_logic);
end component;

function bv_to_natural(bv: in bit_vector) return natural is
  variable result: natural:= 0;
begin
  for index in bv'range loop
     result:= result*2 + bit'pos(bv(index));
  end loop;
  return result;
end bv_to_natural;

signal C: unsigned(3 downto 0);
signal instruction: unsigned(11 downto 0);
```

```
signal we : std_logic;
signal data : unsigned (3 downto 0);
signal addr : unsigned(3 downto 0);
signal clk : std_logic := '1';
signal reset: std_logic;

type state_cycle is (memop, mova, movb, movc, movd, add, sub, cmp, shl,
shr, rl, rr, c_and,
c_or, c_not, nop);
signal state: state_cycle;
signal opcode: unsigned(3 downto 0);
constant cADD: unsigned(3 downto 0)  := "0000";
constant cSUB: unsigned(3 downto 0)  := "0001";
constant cCMP: unsigned(3 downto 0)  := "0010";
constant cSHL: unsigned(3 downto 0)  := "0011";
constant cSHR: unsigned(3 downto 0)  := "0100";
constant cROL: unsigned(3 downto 0)  := "0101";
constant cROR: unsigned(3 downto 0)  := "0110";
constant cAND: unsigned(3 downto 0)  := "0111";
constant cOR : unsigned(3 downto 0)  := "1000";
constant cNOT: unsigned(3 downto 0)  := "1001";
constant cMOVA: unsigned(3 downto 0) := "1010";
constant cMOVB: unsigned(3 downto 0) := "1011";
constant cMOVC : unsigned(3 downto 0):= "1100";
constant cMOVD: unsigned(3 downto 0) := "1101";



begin

MicroPro_CPU: MicroProcessor
  port map(reset => reset,
           clk         => clk,
           instruction => instruction,
           addr => addr,
           data => data,
           we => we,
           C           => C);


clk <= not clk after 10 ns;

reset_input:process
begin
  reset <= '0';
  wait for 10 ns;
  reset <= '0';
  wait for 9310 ns;
end process;


stimulus_interpreter: process
```

```
  use std.textio.all;
  file testcase:text is in "test_opcode.in";
  variable command: line;
  variable read_ok: boolean;
  variable next_time: time;
  variable whitespace: character;
  variable signal_id: character;
  variable addr_value: bit_vector(3 downto 0);
  variable data_value: bit_vector(3 downto 0);
  variable instr_value: bit_vector(11 downto 0);
  variable vinstruction: unsigned(11 downto 0);
begin

   test_loop: while not endfile(testcase) loop
   wait until clk = '1';

   readline(testcase, command);
   read(command, signal_id, read_ok);
If not read_ok  then
   assert read_ok
      report "error reading signal_id must be 'w' or 'i':"
      severity warning;
      next test_loop;
end if;
--   end if;

   case signal_id is

      when 'w' =>
           read (command,addr_value,read_ok);
           If  not read_ok  then
           assert read_ok
                 report "error reading addr_value from line:" &command.all
             severity warning;
                 next test_loop;
           end if;

           addr <= to_unsigned(bv_to_natural(addr_value),4);
           read (command,data_value,read_ok);

      if not read_ok  then
       assert read_ok
                     report "error reading data_value from line:" & command.all
             severity warning;
             next test_loop;
      end if;

                data <= to_unsigned(bv_to_natural(data_value),4);
                state <= memop;
           we  <= '1';

           wait for 20 ns;
```

```
        when 'i' =>
           read (command,instr_value,read_ok);

       if (not read_ok )then
            assert read_ok
                      report "error reading instr_value from line:" & command.all
              severity warning;
              next test_loop;
      end if;

                  instruction <= to_unsigned(bv_to_natural(instr_value),12);
                  vinstruction := to_unsigned(bv_to_natural(instr_value),12);

                  case(vinstruction(3 downto 0)) is
                    when cMOVA => state <= mova;
                    when cMOVB => state <= movb;
                    when cMOVC => state <= movc;
                    when cMOVD => state <= movd;
                    when cADD => state <= add;
                    when cSUB => state <= sub;
                    when cCMP => state <= cmp;
                    when cSHL => state <= shl;
                     when cSHR => state <= shr;
                     when cROL => state <= rl;
                    when cROR => state <= rr;
                    when cAND => state <= c_and;
                    when cOR  => state <= c_or;
                    when cNOT => state <= c_not;
                    when others => state <= nop;
                  end case;
                   we <= '0';
       when others =>
       assert false
            report "invalid signal id in line: " & signal_id
            severity warning;
            next test_loop;
       end case;

    end loop test_loop;

    wait;

end process stimulus_interpreter;

end MicroPro_TB;
```

4.2 4-bit Bit-Slice Microprocessor (Top View)



4.2.1 Synthesized blocks (4-bit bit-slice microprocessor)

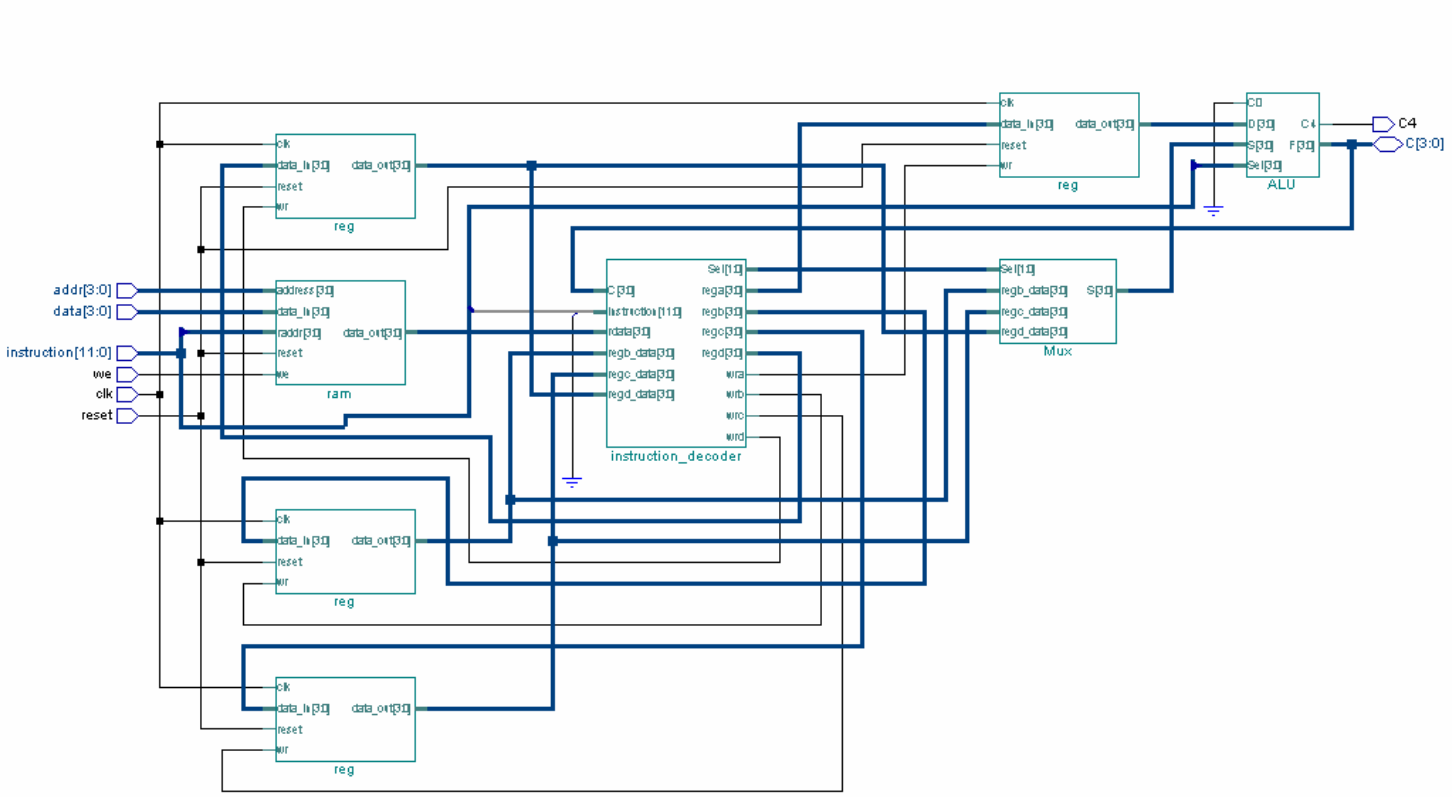


Figure 4.2: Synthesized blocks of a 4-Bit Slice Microprocessor

4.2.2 VHDL Program (4-bit bit-slice microprocessor)

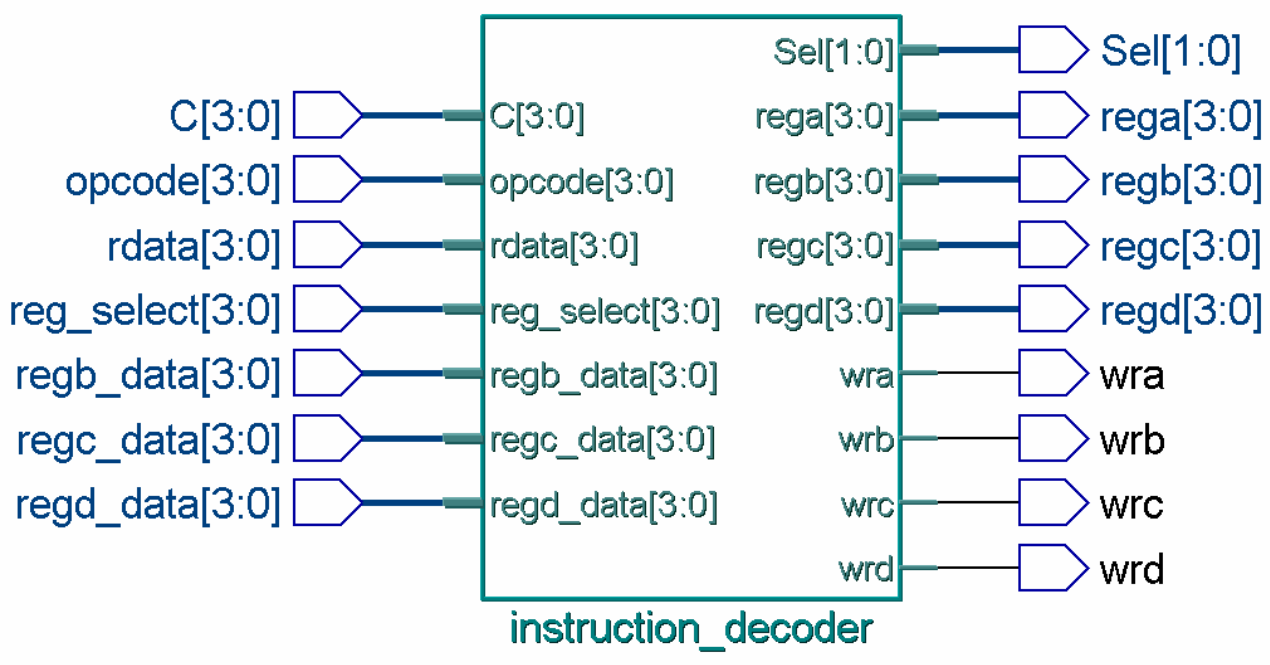
```
------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : MicroProcessor.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   : This Program describes the top level entity of the
--                 Microprocessor.
------------------------------------------------------------------
--
------------------------------------------------------------------
library IEEE;
use IEEE.STD_Logic_1164.all;
use IEEE.numeric_std.all;


entity MicroProcessor is
  port(reset      : in std_logic;
       clk        : in std_logic;
       instruction: in unsigned(11 downto 0);
       addr       : in unsigned(3 downto 0);
       data       : in unsigned(3 downto 0);
       we         : in std_logic;
       C          : inout unsigned(3 downto 0);
       C4         : out std_logic);
end MicroProcessor;


architecture MicroPro of MicroProcessor is

component instruction_decoder
  port(instruction: in unsigned(11 downto 0);
       rdata      : in unsigned(3 downto 0);
       regb_data  : in unsigned(3 downto 0);
       regc_data  : in unsigned(3 downto 0);
       regd_data  : in unsigned(3 downto 0);
       C          : in unsigned(3 downto 0);
       rega       : out unsigned(3 downto 0);
       regb       : out unsigned(3 downto 0);
       regc       : out unsigned(3 downto 0);
       regd       : out unsigned(3 downto 0);
       wra        : out std_logic;
       wrb        : out std_logic;
       wrc        : out std_logic;
       wrd        : out std_logic;
       Sel        : out unsigned(1 downto 0)
       );
end component;
```

```
component ALU
  port (Sel : in  unsigned(3 downto 0);
      D   : in  unsigned(3 downto 0);
      S   : in  unsigned(3 downto 0);
      C0  : in  std_logic;
      C4  : out std_logic;
      F   : out unsigned(3 downto 0));
end component;

component ram
port (
  reset    : in  std_logic;
  address  : in  unsigned(3 downto 0);
  data_in  : in  unsigned(3 downto 0);
  raddr    : in  unsigned(3 downto 0);
  data_out : out unsigned(3 downto 0);
  we       : in  std_logic

);
end component;

component Mux
  port(Sel        : in unsigned(1 downto 0);
       regb_data  : in unsigned(3 downto 0);
       regc_data  : in unsigned(3 downto 0);
       regd_data  : in unsigned(3 downto 0);
       S          : out unsigned(3 downto 0));
end component;


component reg
port (reset    : in std_logic;
      clk      : in std_logic;
      wr       : in std_logic;
      data_in  : in  unsigned(3 downto 0);
      data_out : out unsigned(3 downto 0));
end component;


signal X: unsigned(3 downto 0);
signal Y: unsigned(3 downto 0);
signal Sel: unsigned(1 downto 0);
signal data_out: unsigned(3 downto 0);
signal data_x: unsigned (3 downto 0);
signal data_y: unsigned (3 downto 0);
signal raddr: unsigned(3 downto 0);
signal rega_data, regb_data, regc_data, regd_data: unsigned(3 downto 0);
signal C0: std_logic := '0';

signal S: unsigned(3 downto 0);
signal D: unsigned(3 downto 0);
signal F: unsigned(3 downto 0);
```

```
signal rdata: unsigned( 3 downto 0);
signal rega: unsigned(3 downto 0);
signal regb: unsigned(3 downto 0);
signal regc: unsigned(3 downto 0);
signal regd: unsigned(3 downto 0);
signal wra, wrb, wrc, wrd: std_logic;



begin

RRAM:ram
port map(
  reset => reset,
  address  => addr,
  data_in  => data,
  raddr    => instruction(7 downto 4),
  data_out => rdata,
  we       => we);


dec:instruction_decoder
port map(instruction => instruction,
       rdata      => rdata,
       regb_data  => regb_data,
       regc_data  => regc_data,
       regd_data  => regd_data,
       C          => C,
       rega       => rega,
       regb       => regb,
       regc       => regc,
       regd       => regd,
       wra        => wra,
       wrb        => wrb,
       wrc        => wrc,
       wrd        => wrd,
       Sel        => Sel
             );



ALU_Inst: ALU
  port map

     ( Sel          => instruction(3 downto 0),
       D           => D,
        S           => S,
        C0          => C0,
        C4          => C4,
         F           => C);
```

```
Mux_Inst:Mux
  port map(Sel    => Sel,
       regb_data  => regb_data,
       regc_data  => regc_data,
       regd_data  => regd_data,
       S          => S);


A_Reg:reg
port map (reset    => reset,
          clk      => clk,
          wr       => wra,
          data_in  => rega,
          data_out => D);


B_Reg:reg
port map (reset    => reset,
          clk      => clk,
          wr       => wrb,
          data_in  => regb,
          data_out => regb_data);


C_Reg:reg
port map (reset    => reset,
          clk      => clk,
          wr       => wrc,
          data_in  => regc,
          data_out => regc_data);


D_Reg:reg
port map (reset    => reset,
          clk      => clk,
          wr       => wrd,
          data_in  => regd,
          data_out => regd_data);


end MicroPro;
```

4.3 Instruction Decoder



4.3.1 Synthesized blocks (Instruction Decoder)

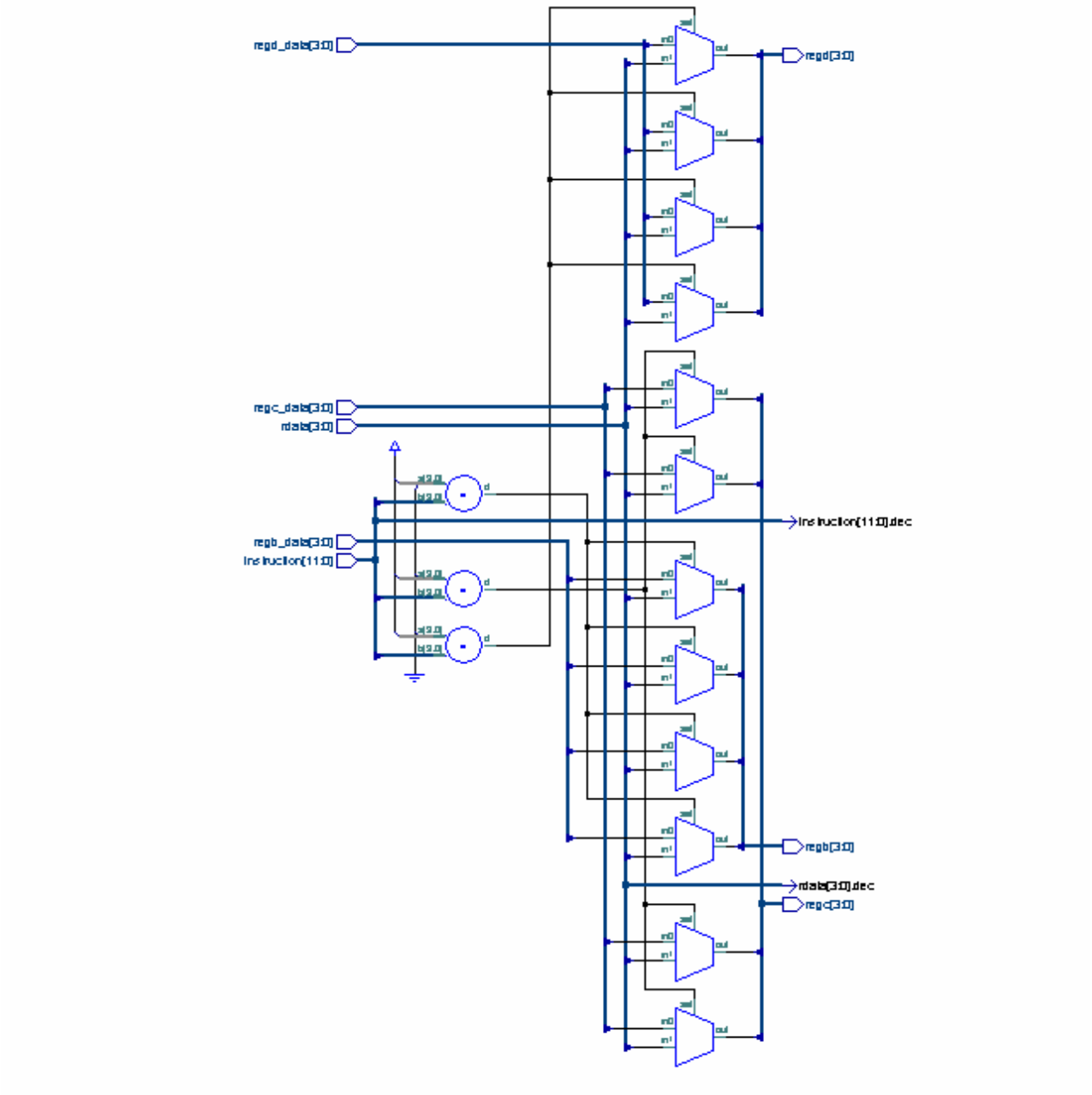


Figure 4.3: Block Diagram of a Instruction Decoder (part 1)

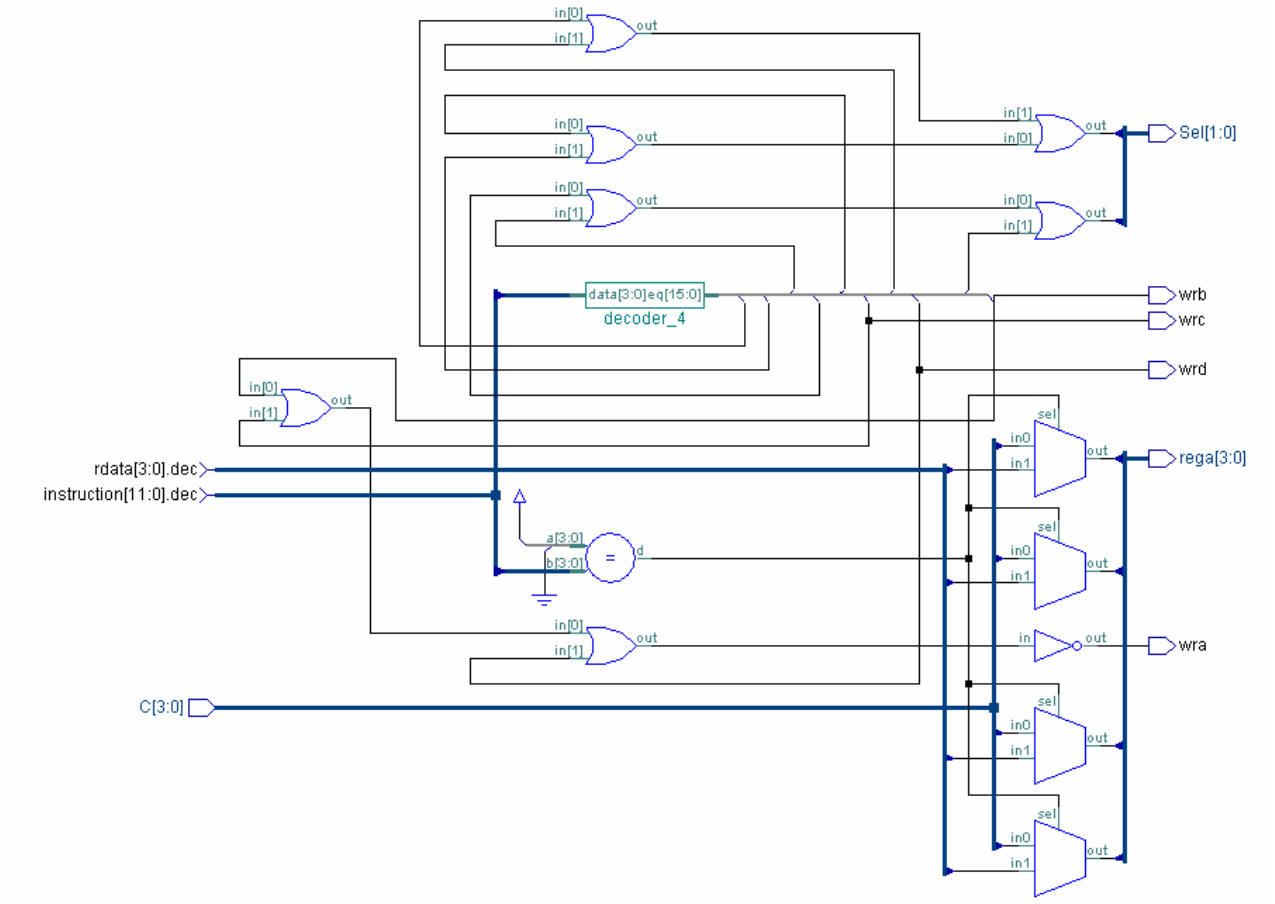


Figure 4.3: Block Diagram of a Instruction Decoder (part II)

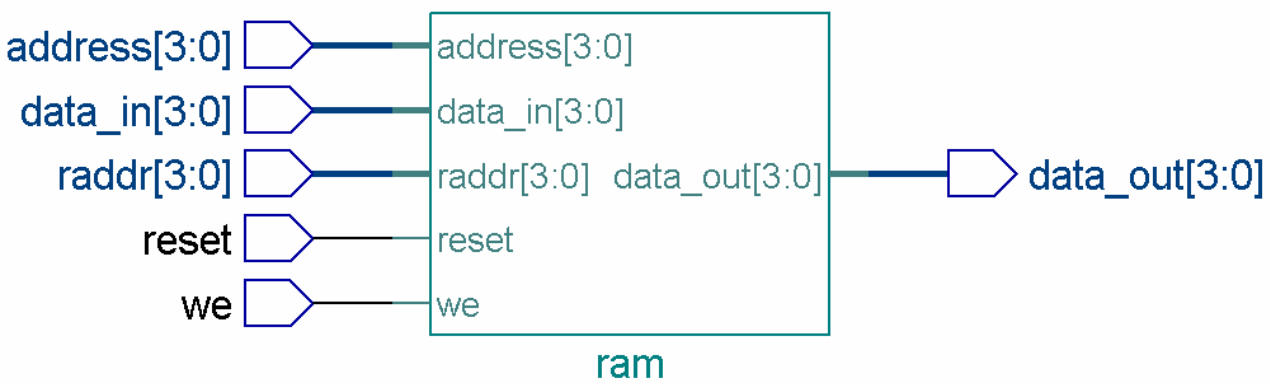
4.3.2 VHDL Program for Instruction Decoder:

```
-------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : Instruction_Decoder.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   : This Program describes the functional behaviour of
--                 Instruction Decoder.
-------------------------------------------------------------------
--
-------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_STD.all;


entity instruction_decoder is
  port(instruction: in unsigned(11 downto 0);
       rdata      : in unsigned(3 downto 0);
       regb_data  : in unsigned(3 downto 0);
       regc_data  : in unsigned(3 downto 0);
       regd_data  : in unsigned(3 downto 0);
       C          : in unsigned(3 downto 0);
       rega       : out unsigned(3 downto 0);
       regb       : out unsigned(3 downto 0);
       regc       : out unsigned(3 downto 0);
       regd       : out unsigned(3 downto 0);
       wra        : out std_logic;
       wrb        : out std_logic;
       wrc        : out std_logic;
       wrd        : out std_logic;
       Sel        : out unsigned(1 downto 0)
       );
end instruction_decoder;


architecture control_hdl of instruction_decoder is

constant cADD: unsigned(3 downto 0)  := "0000";
constant cSUB: unsigned(3 downto 0)  := "0001";
constant cCMP: unsigned(3 downto 0)  := "0010";
constant cSHL: unsigned(3 downto 0)  := "0011";
constant cSHR: unsigned(3 downto 0)  := "0100";
constant cROL: unsigned(3 downto 0)  := "0101";
constant cROR: unsigned(3 downto 0)  := "0110";
constant cAND: unsigned(3 downto 0)  := "0111";
constant cOR : unsigned(3 downto 0)  := "1000";
constant cNOT: unsigned(3 downto 0)  := "1001";
constant cMOVA: unsigned(3 downto 0) := "1010";
constant cMOVB: unsigned(3 downto 0) := "1011";
```

```
constant cMOVC : unsigned(3 downto 0):= "1100";
constant cMOVD: unsigned(3 downto 0) := "1101";
constant SelB: unsigned(1 downto 0) := "00";
constant SelC: unsigned(1 downto 0) := "01";
constant SelD: unsigned(1 downto 0) := "10";
begin



process(instruction(11 downto 8), rdata, C)
begin
  case (instruction(11 downto 8)) is
     when "0001" => rega <=  rdata;
     when others  => rega <= C;
  end case;
end process;



process(instruction(3 downto 0))
begin
  case (instruction(3 downto 0)) is
                  when cADD  => Sel <= SelB;
             when cSUB  => Sel <= SelB;
             when cCMP  => Sel <= SelB;
                  when cSHL  => Sel <= SelC;
               when cSHR  => Sel <= SelC;
               when cROL  => Sel <= SelC;
                  when cROR  => Sel <= SelC;
                  when cAND  => Sel <= SelD;
                  when cOR   => Sel <= SelD;
                  when cNOT  => Sel <= SelD;
                  when others => Sel <= SelB;
                end case;
end process;


with instruction(11 downto 8) select
     regb <= rdata when "0010",
      regb_data when others;


with instruction(11 downto 8) select
     regc <= rdata when "0100",
      regc_data when others;


with instruction(11 downto 8) select
     regd <= rdata when "1000",
      regd_data when others;

with instruction(3 downto 0) select
```

```
     wra <= '0' when "1011" | "1100" | "1101",
    '1' when others;

with instruction(3 downto 0) select
     wrb <= '1' when "1011",
    '0' when others;

with instruction(3 downto 0) select
     wrc <= '1' when "1100",
    '0' when others;

with instruction(3 downto 0) select
     wrd <= '1' when "1101",
    '0' when others;


end control_hdl;
```

4.4 RAM



4.4.1 Block Diagram (RAM)

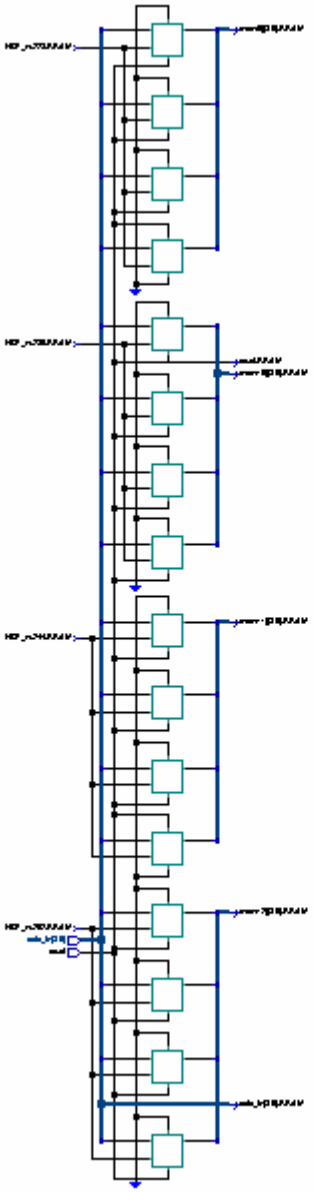


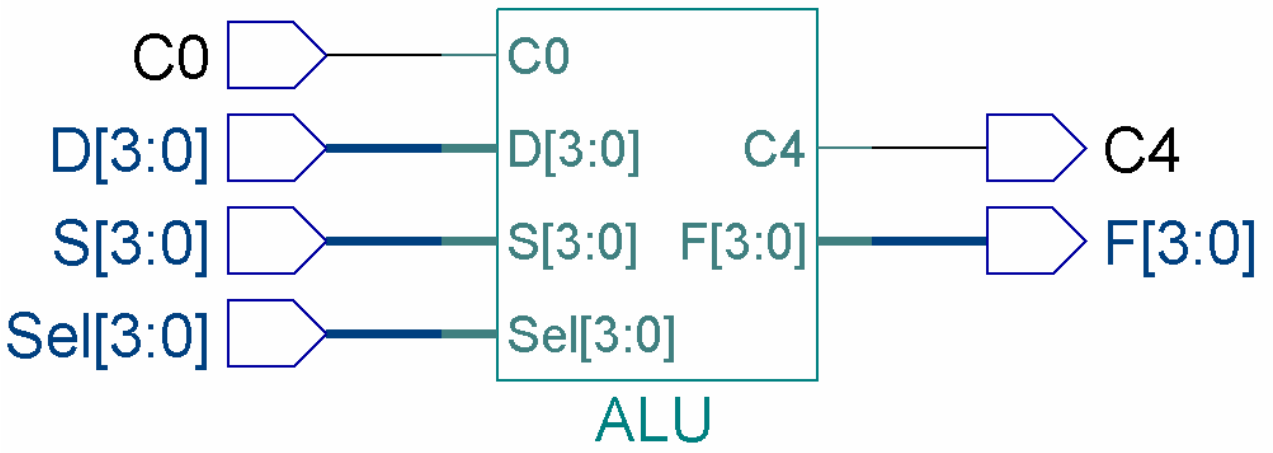
Figure 4.4: Block Diagram of a RAM

4.4.2 VHDL Program (RAM)

```
-------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : ram.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   : This Program describes the behaviour of 16 X 4 RAM
-------------------------------------------------------------------
--
-------------------------------------------------------------------
Library IEEE ;

use IEEE.std_logic_1164.all ;
use IEEE.Numeric_STD.all;

entity ram is
port (
  reset    : in  std_logic;
  address  : in  unsigned(3 downto 0);
  data_in  : in  unsigned(3 downto 0);
  raddr    : in  unsigned(3 downto 0);
  data_out : out unsigned(3 downto 0);
  we       : in  std_logic

);

end ram ;



architecture rtl of ram is

type mem_type is array (0 to 15) of unsigned(3 downto 0);

signal mem : mem_type ;

begin


process (reset, we,raddr, address)
begin
   if reset = '1' then
     mem(0)  <= "0000";
     mem(1)  <= "0001";
     mem(2)  <= "0000";
     mem(3)  <= "0001";
     mem(4)  <= "0000";
     mem(5)  <= "0001";
     mem(6)  <= "0000";
     mem(7)  <= "0001";
     mem(8)  <= "0000";
```

```
     mem(9)  <= "0001";
     mem(10) <= "0000";
     mem(11) <= "0001";
     mem(12) <= "0000";
     mem(13) <= "0001";
     mem(14) <= "0000";
     mem(15) <= "1001";
    else

          if we = '1' then
             mem(to_integer(address)) <= data_in;
          end if;
             data_out <= mem(to_integer(raddr));
    end if;
end process;

end rtl ;
```

4.5 ALU



4.5.1 Block Diagram (ALU)

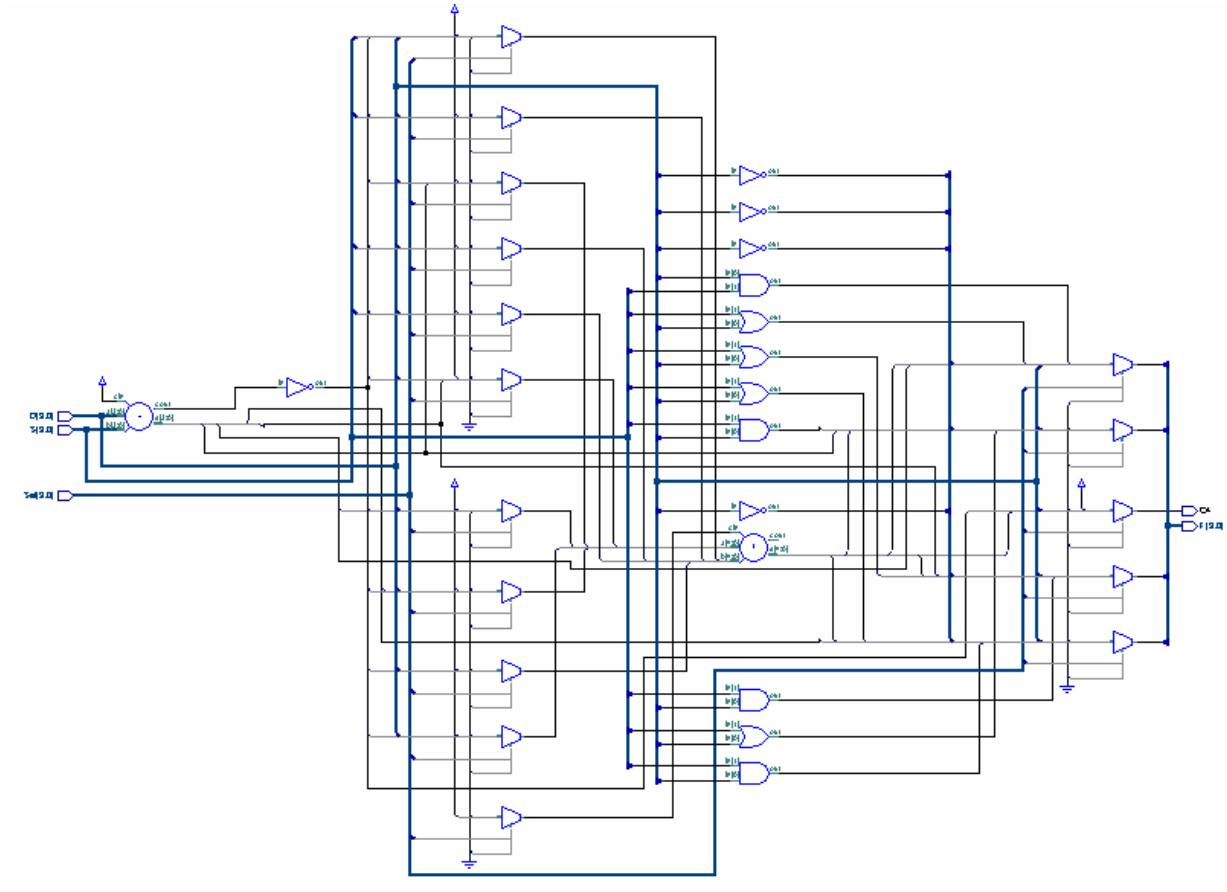


Figure 4.5: Block Diagram of a ALU

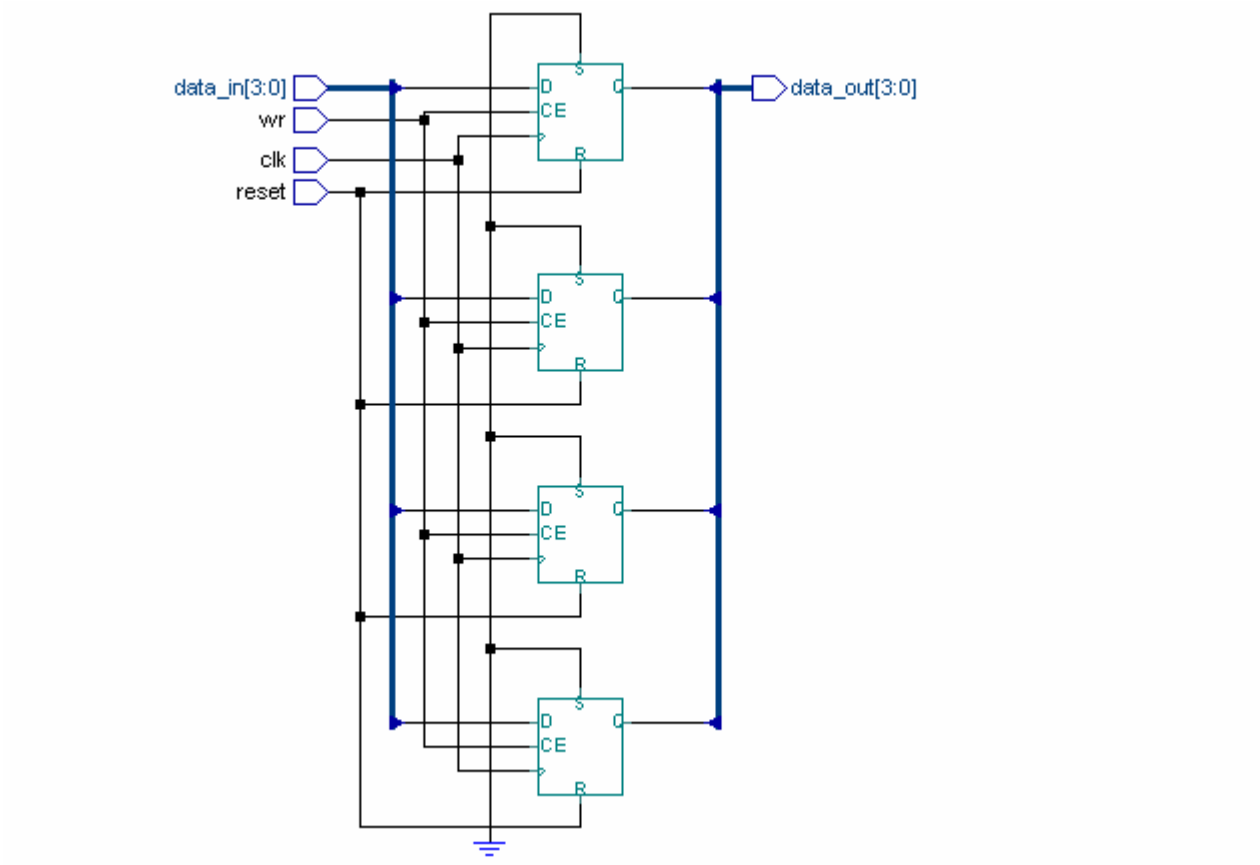
4.5.2 VHDL Program (ALU)

```
-------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : alu.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   :
-------------------------------------------------------------------
--
-------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.NUMERIC_STD.all;
entity ALU is
  port (Sel : in  unsigned(3 downto 0);
      D   : in  unsigned(3 downto 0);
      S   : in  unsigned(3 downto 0);
      C0  : in  std_logic;
      C4  : out std_logic;
      F   : out unsigned(3 downto 0)
      );
end ALU;


architecture alu_block of ALU is

constant cADD: unsigned(3 downto 0) := "0000";
constant cSUB: unsigned(3 downto 0) := "0001";
constant cCMP: unsigned(3 downto 0) := "0010";
constant cSHL: unsigned(3 downto 0) := "0011";
constant cSHR: unsigned(3 downto 0) := "0100";
constant cROL: unsigned(3 downto 0) := "0101";
constant cROR: unsigned(3 downto 0) := "0110";
constant cAND: unsigned(3 downto 0) := "0111";
constant cOR : unsigned(3 downto 0) := "1000";
constant cNOT: unsigned(3 downto 0) := "1001";

signal extD, extS, carry: unsigned(4 downto 0);
```

```
signal result: unsigned(4 downto 0);
begin


  extS <= '0' & S;
  extD <= '0' & D;
  carry <= "0000" & C0;

  with Sel select
     result <= extD + extS + carry when cADD,
               extD - extS - carry when cSUB,
                  extD - extS when cCMP,
               shift_left(extD,1) when cSHL,
               shift_right(extD,1) when cSHR,
               rotate_left(extD,1) when cROL,
               rotate_right(extD,1) when cROR,
               extD and extS when cAND,
               extD or extS when cOR,
               not(extD) when cNOT,
               extD when others;



  F   <= result(3 downto 0);
  C4  <= result(4);


end alu_block;
```

4.6 Register (RegA, RegB, RegC, RegD)

4.6.1 Block Diagram (Register)



4.6.2 VHDL Program (Reg)

```
-------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : reg.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   :
-------------------------------------------------------------------
--
-------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_STD.all;

entity reg is
port (reset    : in std_logic;
      clk      : in std_logic;
      wr       : in std_logic;
      data_in  : in  unsigned(3 downto 0);
      data_out : out unsigned(3 downto 0));
end reg;

architecture behav of reg is
signal data: unsigned(3 downto 0);
begin

  data_out <= data;

  reg_data:process(clk, reset)
  begin
        if reset = '1' then
            data <= "0000";
     elsif clk'event and clk = '1' then
            if wr = '1' then
           data <= data_in;
         end if;
        end if;
  end process;
end behav;



-------------------------------------------------------------------
-- Title         : VHDL style
-- File name     : reg.vhd
-- Authors       : Mildred C. Zabawa and Vivek Jayaram
-- Description   :
-------------------------------------------------------------------
--
-------------------------------------------------------------------
```

```
library IEEE;
use IEEE.STD_Logic_1164.all;
use IEEE.numeric_std.all;


entity Mux is
  port(Sel        : in unsigned(1 downto 0);
       regb_data  : in unsigned(3 downto 0);
       regc_data  : in unsigned(3 downto 0);
       regd_data  : in unsigned(3 downto 0);
       S          : out unsigned(3 downto 0));
end entity Mux;

architecture mux_control of mux is

constant SelB: unsigned(1 downto 0) := "00";
constant SelC: unsigned(1 downto 0) := "01";
constant SelD: unsigned(1 downto 0) := "10";

begin

process(Sel, regb_data, regc_data, regd_data)
begin
case Sel is
  when SelB => S <= regb_data;
  when SelC => S <= regc_data;
  when SelD => S <= regd_data;
  when others => S <=regb_data;
end case;
end process;

end mux_control;
```

4.7 Simulated Results

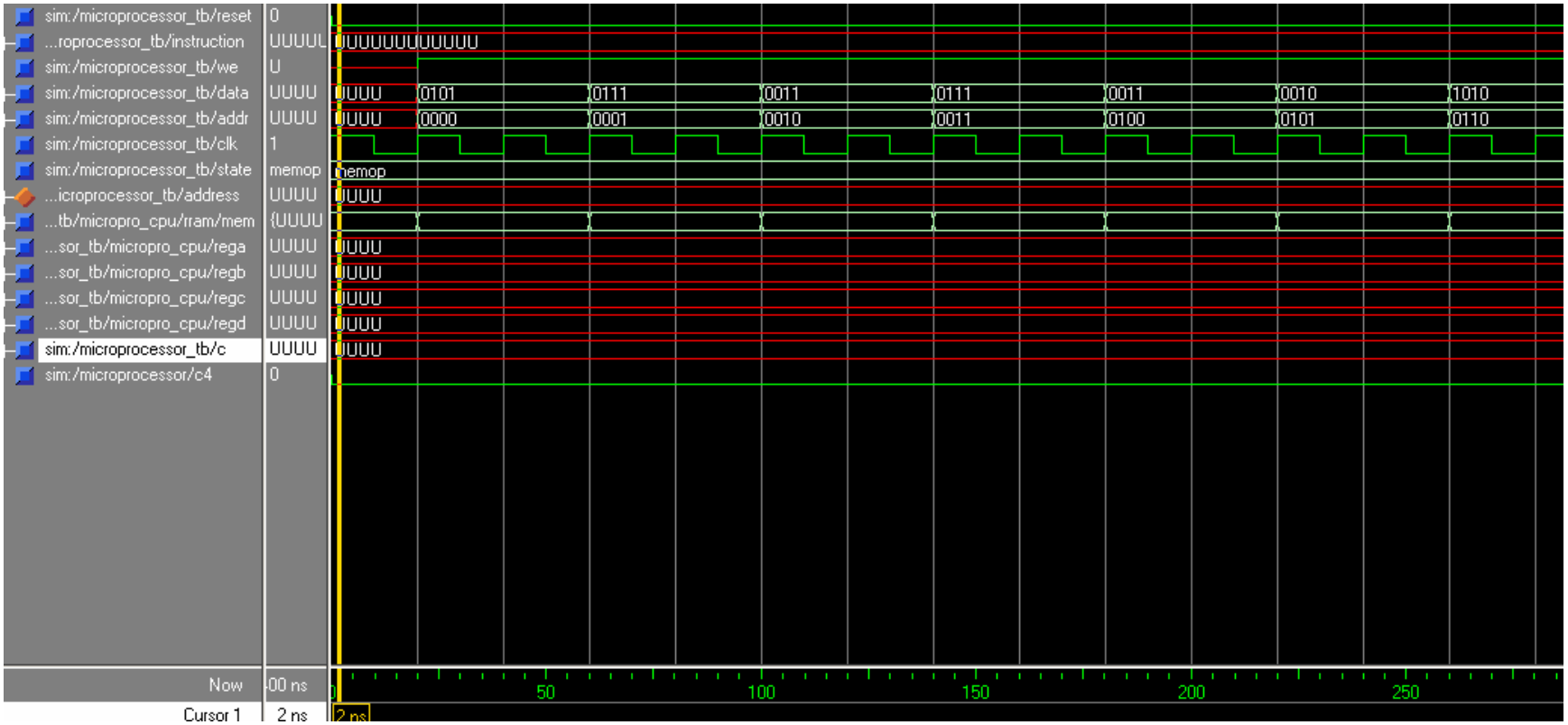


Figure 4.7.1 Timing Diagram of 4-bit microprocessor (preload memory phase)

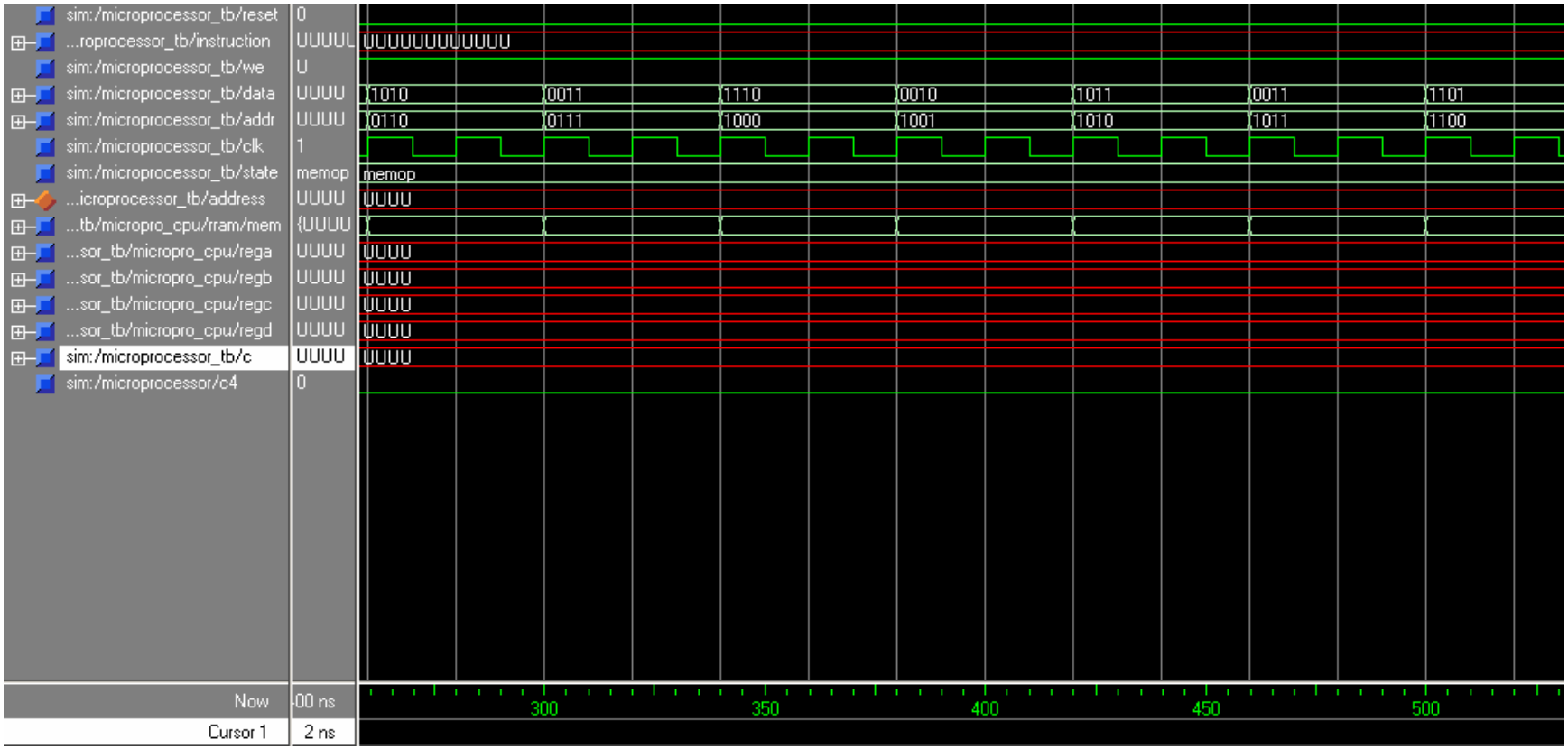


Figure 4.7.2 Timing Diagram of 4-bit microprocessor (preload memory phase)

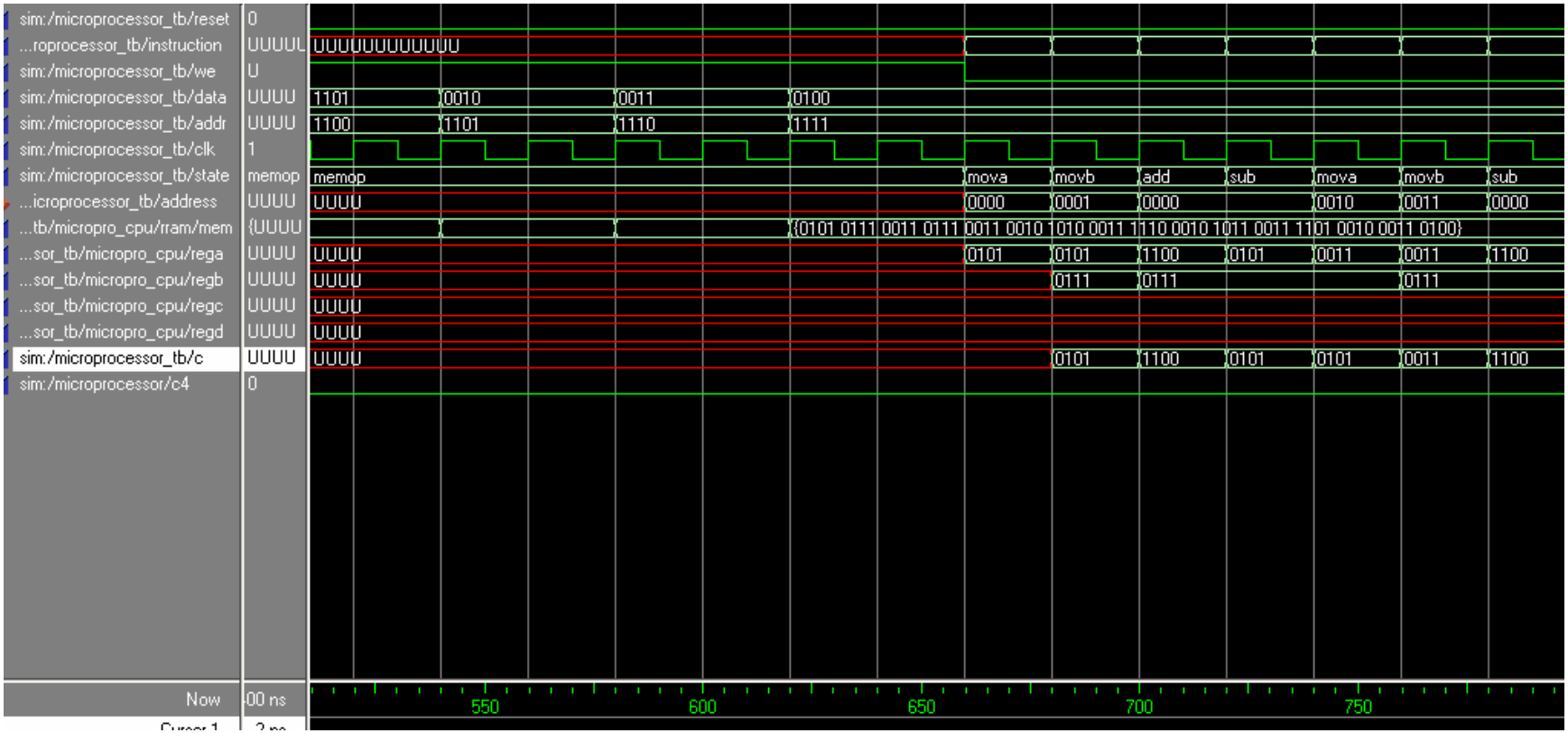


Figure 4.7.3 Timing Diagram of 4-bit microprocessor (instruction phase: ADD, SUB)

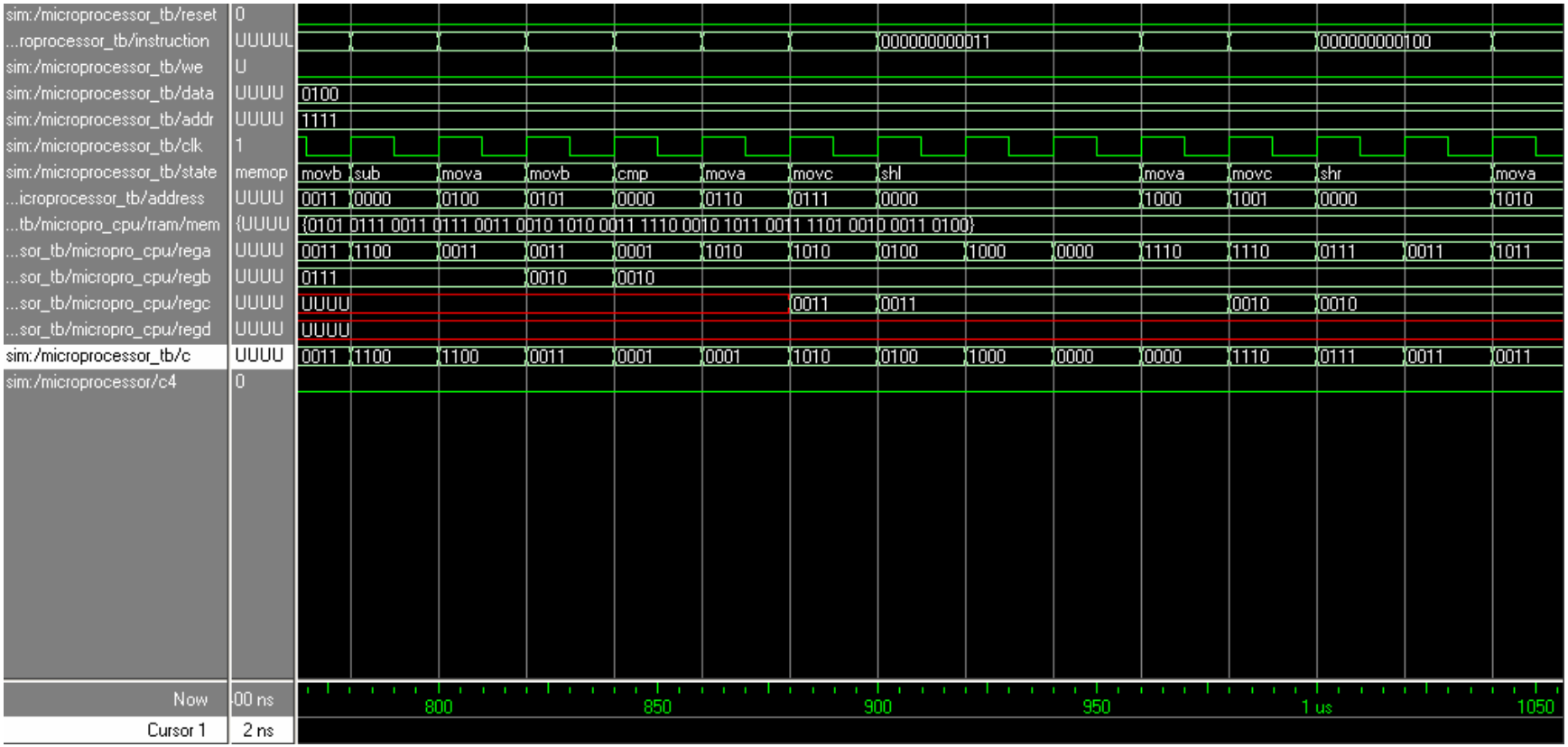


Figure 4.7.4 Timing Diagram of 4-bit microprocessor (instruction phase: CMP, SHL, SHR)

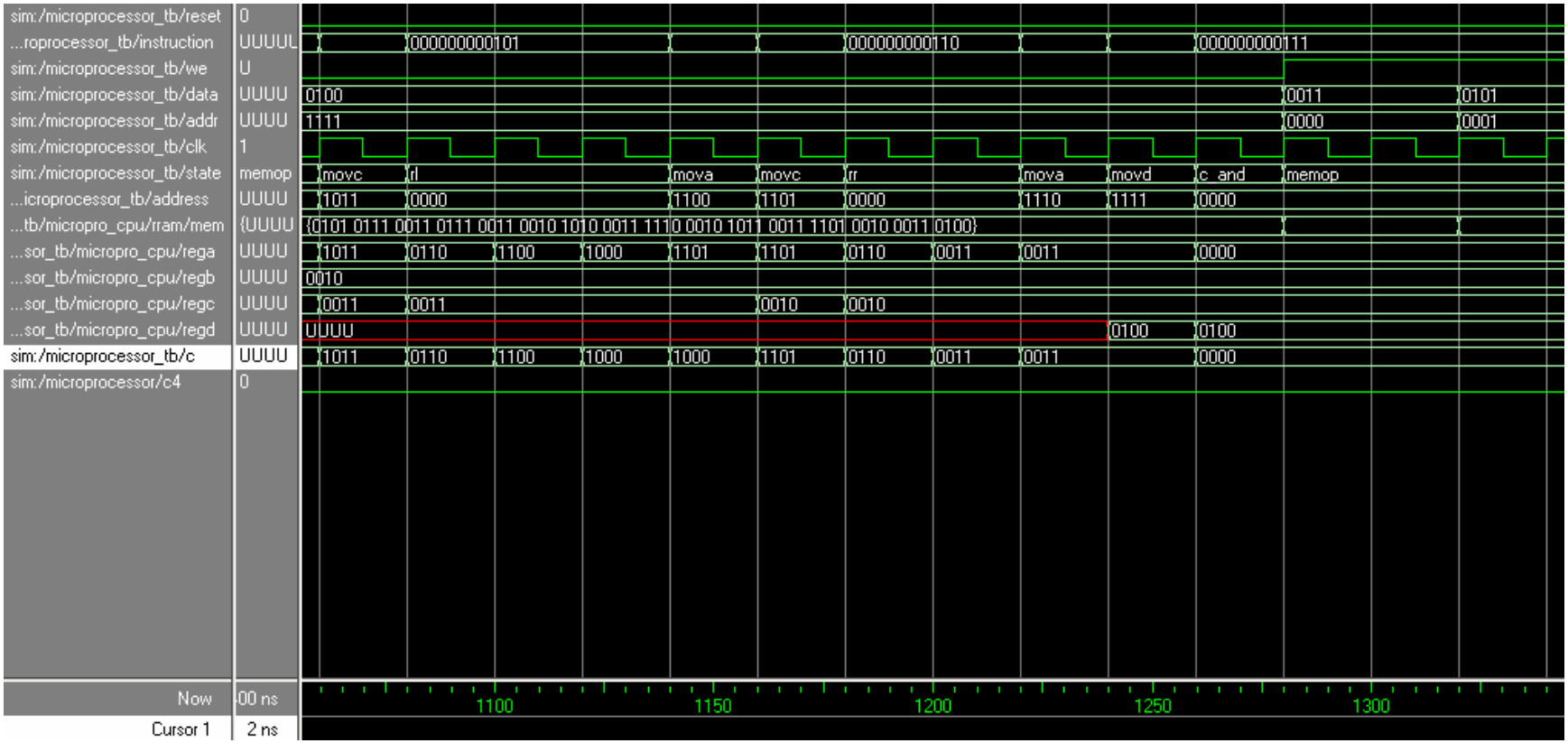


Figure 4.7.5 Timing Diagram of 4-bit microprocessor (instruction phase: RL, RR, AND)

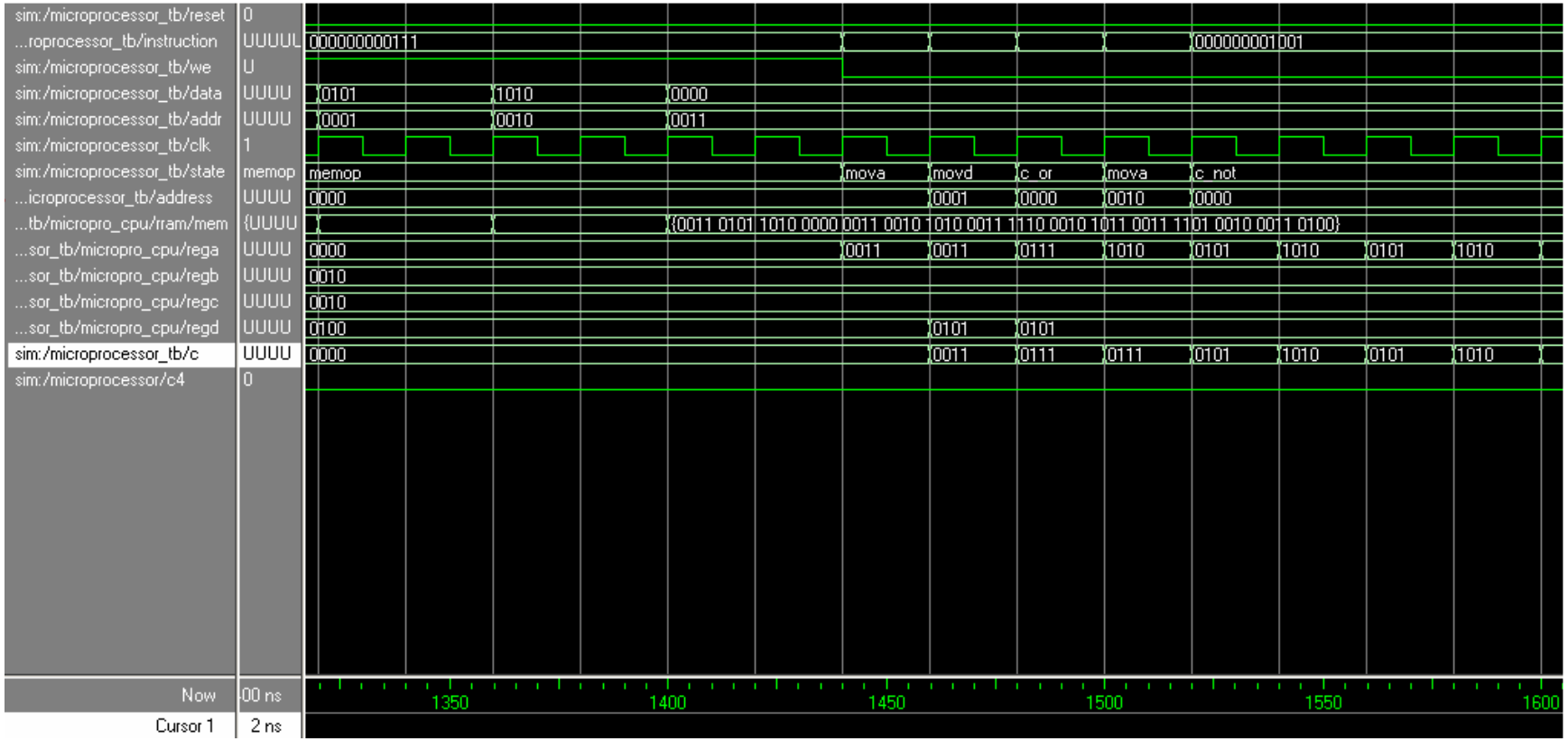


Figure 4.7.6 Timing Diagram of 4-bit microprocessor (instruction phase: OR, NOT)

5 Conclusion

a. **4 Bit-slice microprocessor**

b. **The module has the following pins:**
   i. **Address (4 bit)**
   ii. **Data (4 bit)**
   iii. **Instruction (12 bit)**
   iv. **Write Enable**
   v. **Clock**
   vi. **Reset**
   vii. **Carry out**
   viii. **Output (4 bit)**