EEE-102

19/05/2024

Project Report

PAM-329: Programmable Analytic Machine

Kaan Kırım

Section 4

22301782

Purpose:

The purpose for this project was to design a microprocessor-computer. The name PAM-329 is a reference to a video game character from the game “Fallout 4”, but also an acronym for Programmable Analytic Machine. Specifically, the purpose is to build a machine that can simulate a Turing machine, given enough memory and time. In other words, the purpose is to build a Turing complete computer and implement it on the Basys-3 FPGA board.

Design Specifications:

Although computer and microprocessor are vague definitions, building a Turing Machine is much more straightforward. In essence, this computer should be able to take input, manipulate it according to the program it is running, and output the resulting data in some way. The whole computer mechanism should be constructed around this main idea.

A modern computer has many parts, whereas a physical Turing Machine model is only composed of a tape, and a mechanism that systematically manipulates it. However for this project, a handful of individual components were designed. First, a choice between two conventional architectures of computers had to be made: Harvard or Von Neumann Architecture. The Harvard and Von Neumann architectures differ fundamentally in their memory structures. The Harvard architecture uses separate memory and buses for instructions and data, allowing simultaneous access to both, which enhances performance and efficiency in applications like embedded systems and digital signal processing. This design also offers increased security, as data and instructions are segregated, preventing accidental execution of data as code. Conversely, the Von Neumann architecture utilizes a single memory space for both instructions and data, with a single bus system, which simplifies the design and reduces costs, making it ideal for general-purpose computing. Overall, Von Neumann architecture allowed greater control over whole memory as it can even edit its own program. Hence, I chose the Von Neumann architecture for my project. This single memory, which will store the programs and also the data, will be called the RAM hereafter, as the RAM serves this purpose in computers. Other than RAM, the main parts of this computer were central processing unit, graphical unit and the instruction display.

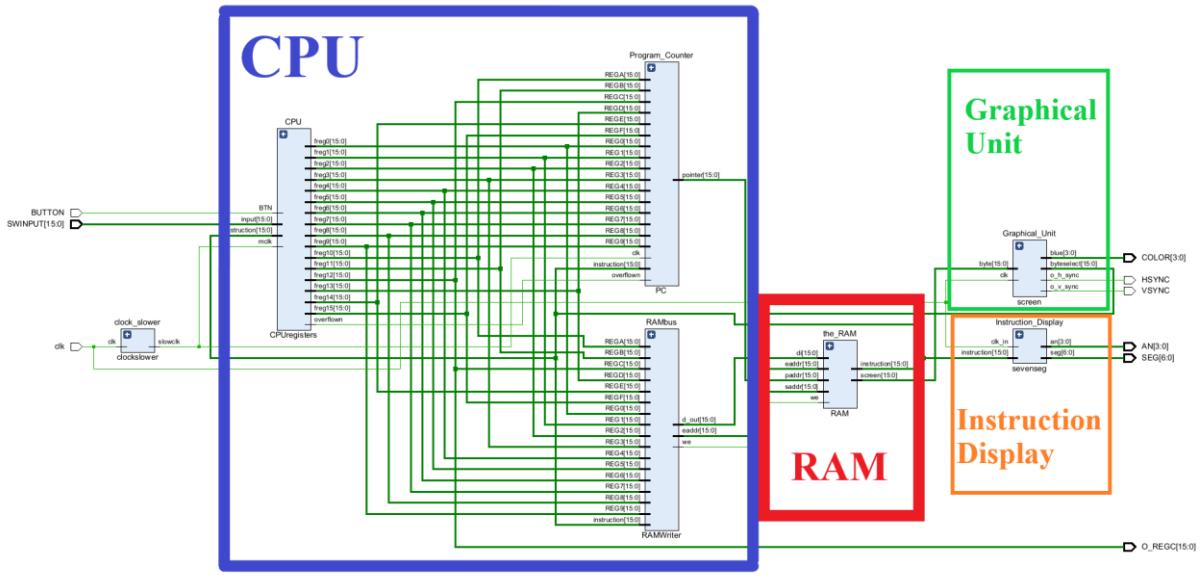

Fig. 1: The Parts of the Computer

1.Central Processing Unit:

Arguably the most important part of the mainstream computers, the CPU executes the program instruction by instruction. Instructions are the basic building blocks programs, and they command the CPU to perform necessary actions on the data. Every processor has an instruction set which it can execute. For that I was designing a processor from scratch, I decided to design my own instruction set as well. In this instruction set, each instruction is a 16-bit number which will be represented by a 4-digit hexadecimal number from now on. Each instruction is composed of two main parts, the operational code and the data bits. The operational code tells the CPU which action to perform, and the data bits tell which data to perform on. In the following table R[x] means the number stored in the xth Register, while values in quotes represent immediate values. Each column except for the explanation and name represents a hexadecimal digit.

Name	OP-CODE	DATA-1	DATA-2	DATA-3	Explanation
ADD	0	Q	P	S	R[Q]<=R[P]+R[S]
SUB	1	Q	P	S	R[Q]<=R[P]-R[S]
AND	2	Q	P	S	R[Q]<=R[P] ∧ R[S]
OR	3	Q	P	S	R[Q]<=R[P] ∨ R[S]
XOR	4	Q	P	S	R[Q]<=R[P] ⊻ R[S]
NEG	5	Q	P	empty	R[Q]<= ¬R[P]
SHR	6	Q	P	empty	R[Q]<=R[P]>>
SHL	7	Q	P	empty	R[Q]<=R[P]<<
WRT	8	Q	P	S	R[Q]<=”PS”
JGE	9	Q	P	S	If R[P] ≥R[S]; PC=R[Q]
JMP	A	Q	empty	empty	PC=R[Q]
SET	B	Q	P	empty	RAM[Q]=R[P]

Table 1: the Instruction Set

To execute these instructions, the CPU needs many components as well. The parts of the CPU can be listed as: arithmetic logic unit, registers, program counter, and the buses.

The Registers:

The registers are parts which hold values in the processor. They can be described as groups of D Flip-Flops. For this specific computer, 16 registers which can hold 16-bit numbers were implemented. Out of all, Register-6 was directly connected to the switches on the Basys-3 take input and could not be manipulated with instructions, whereas Register-C was directly connected to the LEDs on the basys-3 to represent the data it holds. Apart from these, there was not much to the registers.

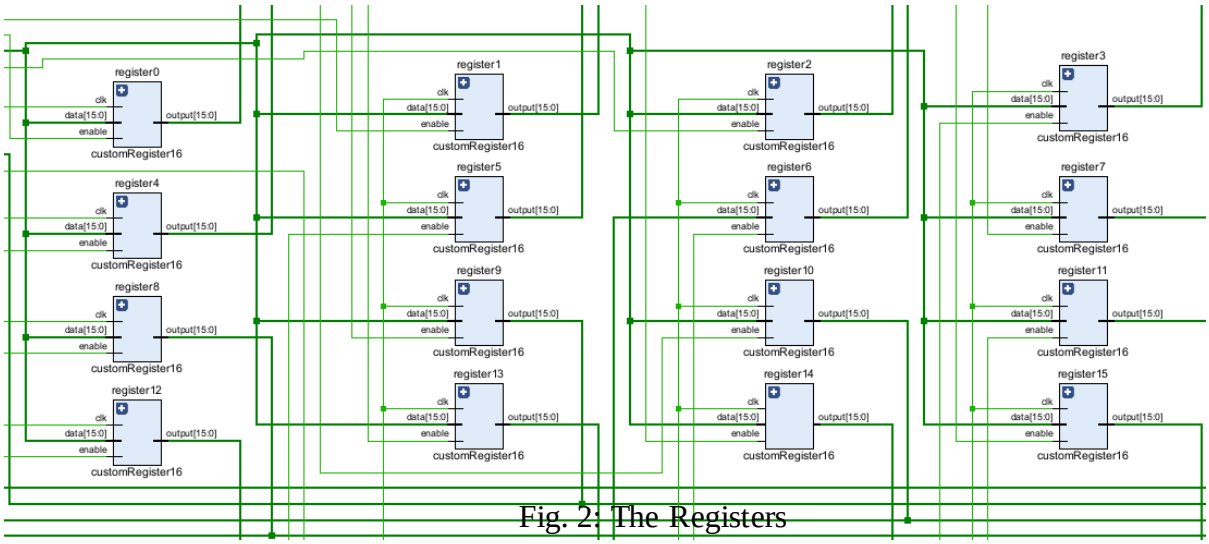

Fig. 2: The Registers

The ALU:

The arithmetic logic unit is the part that executes the instructions in the CPU. It can be described as the brain of the whole CPU. On every rising edge, a new instruction is fed to the CPU, using the data in this instruction, two buses bring two values from the registers via selecting them with a MUX gate. Then all possible arithmetic-logical operations are conducted at the same time. The output of the ALU is determined by the opcode instruction holds. For example, if the opcode is 4 a MUX gate directs the value coming from the XOR gates toward output. Then, on the falling edge of the clock, the register which will store the the output of the ALU is enabled until the next rising edge. Thus, each operation is conducted between rising and falling edge, and the result is saved to a register after the falling edge.

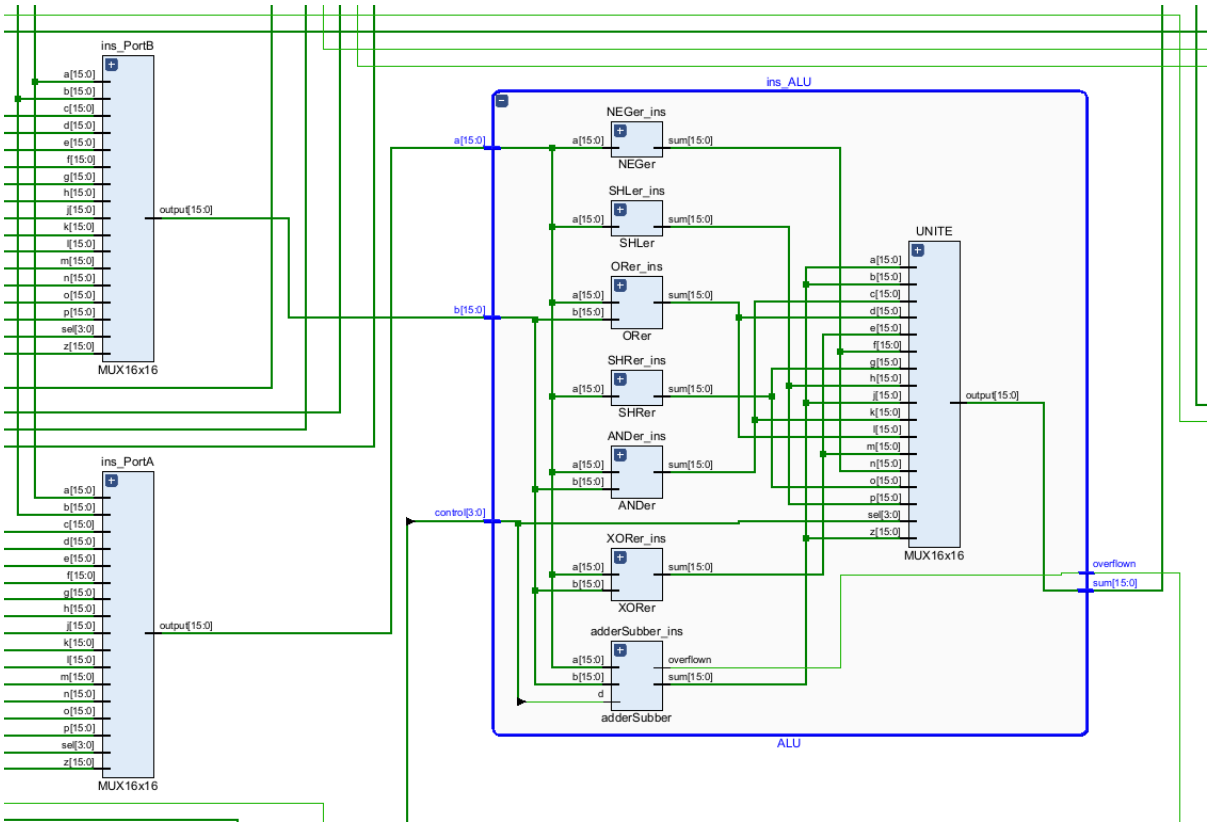

Fig. 3: The ALU

The Program Counter:

If the ALU is the brain of the processor, the program counter is the heart. The program counter holds the address of the current instruction in a register. In every rising edge, the address either increases by one to go to the next instruction, or it changes if the previous instruction involved jumping. The address register is directly connected to an incrementor, while a MUX gate is fed with data from the other registers. A MUX gate is used to select if the address register will be fed with the incremented address, or a whole different value coming from the other registers. If the instruction is JGE, the program counter also looks to the overflow signal coming from the ALU, and decides if it should jump or not.

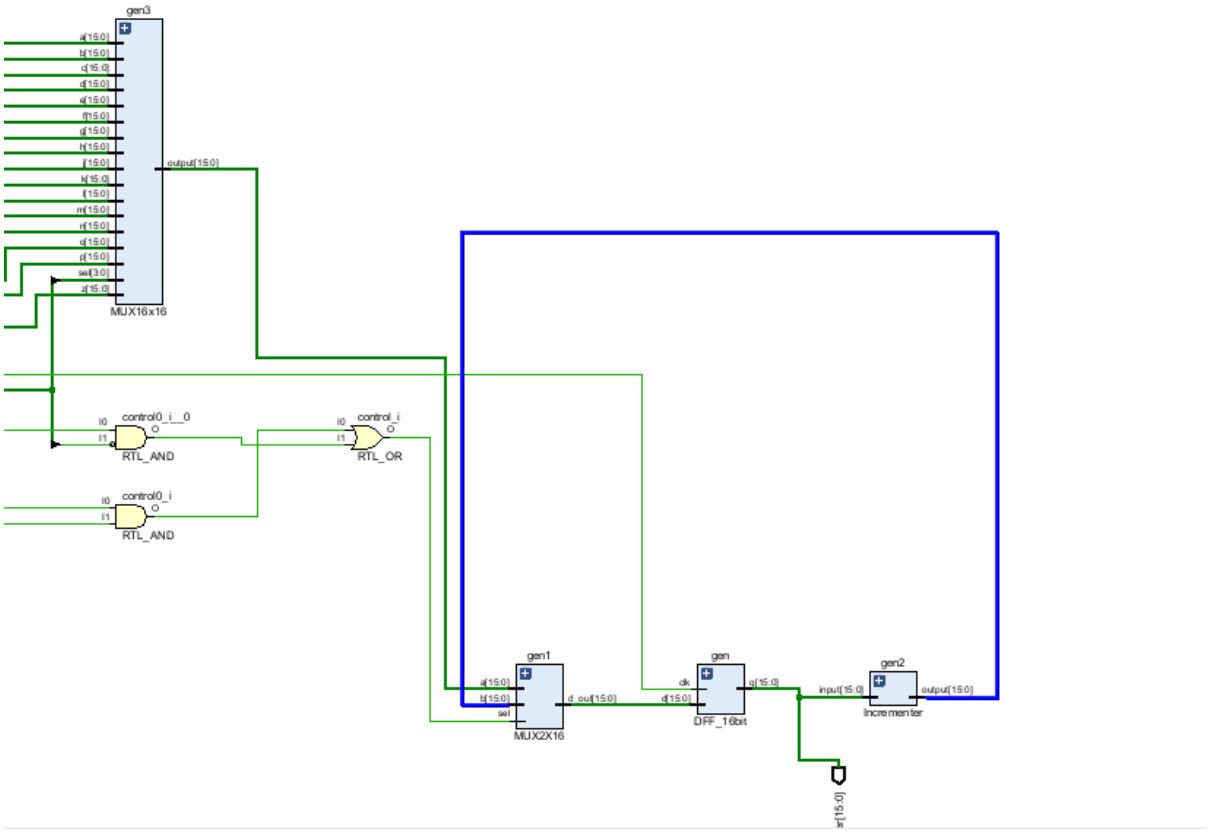

Fig. 4: The Program Counter

Other Parts and Buses:

To handle the WRT instruction, which is the only instruction that involves an immediate value, there is a circuit that sign extends every instruction's last eight bits. Then if the opcode is the one belonging to the WRT instruction, this value is written to the register specified in the instruction. Hence for this instruction the ALU is not even used. Similarly, when the instruction is SET, a MUX gate selects one of the processor registers for value and another for address, then writes the value to the specified address. Lastly, when the opcode is JGE, the ALU still subtracts two values presented in the instruction, then if first value is larger than or equal to the second, it sends the overflow signal. However, the result of this subtraction is not recorded to any register, for that the purpose instruction is only comparison of register values, not real subtraction.

2. RAM

The initial design RAM design I created could store 1048576 (2^20) bits of data. This number is connected to the architecture of the processor registers and length of instructions. As the instructions I had were 16-bits long, I decided to address RAM in 16-bit double bytes also. Hence, the 16-bit registers I had in the processor could uniquely address 2^16 double bytes. Thus this memory could store 65536 instructions. However, I did not use the BRAM on the device correctly, thus the code I wrote built the RAM using latches. Synthesizing this design took more than 9 hours, and to be able to debug my code and work on other things, I had to settle for a smaller RAM. Thus I made a ram of 1024 double bytes or 16384 bits.

This RAM is simultaneously accessed by the program counter, the graphical unit and the RAM bus coming out of the CPU. Thus there are 3 address inputs and 2 according outputs. All programs are directly written to the RAM. When the computer is started, it reads the 0^{th} doublebyte of the RAM and continues onward.

3. Graphical Unit

This module uses the clocking wizard to make a 31.5 Mhz clock, as the VGA communication only allows certain frequencies. I selected for my VGA to have the 640 x 480 resolution, and I designated 192 bytes of the RAM to store data for screen. Hence I displayed 3072 pixels on the screen. For this part I used if statements and numerical calculations. I specifically made a horizontal and vertical index. These are required to adjust H_sync and V_sync, the signals which are required to operate the VGA port. Because I had limited space, my display could only show one color which was green. With horizontal and vertical color increasing each doublebyte for screen data was called from the RAM, then each pixel in the double byte was reflected on the screen.

VGA Signal 640 x 480 @ 73 Hz timing

General timing

Screen refresh rate	73 Hz
Vertical refresh	37.860576923077 kHz
Pixel freq.	31.5 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is negative.

Scanline part	Pixels	Time [µs]
Visible area	640	20.31746031746
Front porch	24	0.76190476190476
Sync pulse	40	1.2698412698413
Back porch	128	4.0634920634921
Whole line	832	26.412698412698

Vertical timing (frame)

Polarity of vertical sync pulse is negative.

Frame part	Lines	Time [ms]
Visible area	480	12.678095238095
Front porch	9	0.23771428571429
Sync pulse	2	0.052825396825397
Back porch	29	0.76596825396825
Whole frame	520	13.734603174603

Fig. 4: Information I used to Design My Graphical Unit

4. Instruction Display

The instruction display was simply a circuit which was also connected to the program counter. It utilized a code written for a previous lab task, which was intending to display numbers on the 7 segment display. As each instruction is a 4 digit hexadecimal number, the 7 segment display on the Basys-3 was perfect for this job. This part was not important as other parts for the computer, it just made it easier to track what the computer is doing.

Results:

In the end, a 16-bit programmable computer emerged. I wrote a few programs, one example may be the Fibonacci calculator. The program takes two inputs first, then adds them, then continues to add the latest two numbers. This program took 12 instructions which can be listed as:

0=>x"8000", WRT, R0,0

1=>x"8101", WRT, R1,1

2=>x"8202", WRT, R2, 2

3=>x"0A60", RA<=R6

4=>x"8909", WRT, R9, 9

5=>x"FFFF", nothing

6=>x"FFFF", nothing

7=>x"FFFF", nothing

8=>x"0B60", RB<=R6

9=>x"0CAB", ADD, RC, RA, RB

10=>x"0AB0", RA<=RB

11=>x"0BC0", RB<=RC

12=>x"A900", JMP, R9

Other programs could also be written, however my purpose for this project was to design the hardware needed to run any program written with these instructions.

Conclusion:

In this project the purpose was to design a computer which was Turing Complete. As for the processor, the design was exactly the way I intended it to be. Building the RAM seemed to be easy but in the end the most difficult and the most performance limiting part was the RAM. If the RAM were capable of storing 65 thousand instructions as I first planned, I was intending to write a compiler in C for this program. Thus any C code would be converted to instructions this computer can operate. Then, I could feed the code for the compiler to this compiler as well and this would allow the computer to compile its own code, effectively programming itself. Limitations on the RAM rendered this impossible, however prewritten programs still work well. One other problem was the constraints file. My constraints file needed to be longer than 1000 lines, because I had to allow combinatorial loops in my design. I think this is due to not using synchronous designs everywhere. Asynchrony introduced the need for allowing combinatorial loops, which is not suggested by Vivado. However, in the end, there was no problem in the function except for some glitch in the VGA port, emerging from two clocks clashing while editing and reading the RAM. Because I have a really modular design, pasting code in this report will be unpractical, for this reason I also uploaded the files on Google Drive with the link:

“https://drive.google.com/file/d/1DRM6pFsT1wSVKK5fePwGtbgP2pvm7Tum/view?usp=sharing”

Appendix:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity PAM329 is
  Port (clk: in STD_LOGIC;
        HSYNC: out STD_LOGIC;
        VSYNC: out STD_LOGIC;
        COLOR: out STD_LOGIC_VECTOR(3 downto 0);
        AN : out STD_LOGIC_VECTOR (3 downto 0);
        SEG : out STD_LOGIC_VECTOR (6 downto 0);
        O_REGC: out STD_LOGIC_VECTOR(15 downto 0);
        BUTTON: in STd_LOGIC;
        SWINPUT: STD_LOGIC_VECTOR(15 DOWNTO 0));
end PAM329;

architecture Behavioral of PAM329 is

component PC is
  Port (clk: in STD_LOGIC;
        instruction: in STD_LOGIC_VECTOR(15 downto 0);
        overflown: in STD_LOGIC;
        REG0: in STD_LOGIC_VECTOR(15 downto 0);
        REG1: in STD_LOGIC_VECTOR(15 downto 0);
        REG2: in STD_LOGIC_VECTOR(15 downto 0);
        REG3: in STD_LOGIC_VECTOR(15 downto 0);
        REG4: in STD_LOGIC_VECTOR(15 downto 0);
        REG5: in STD_LOGIC_VECTOR(15 downto 0);
        REG6: in STD_LOGIC_VECTOR(15 downto 0);
        REG7: in STD_LOGIC_VECTOR(15 downto 0);
        REG8: in STD_LOGIC_VECTOR(15 downto 0);
        REG9: in STD_LOGIC_VECTOR(15 downto 0);
        REGA: in STD_LOGIC_VECTOR(15 downto 0);
        REGB: in STD_LOGIC_VECTOR(15 downto 0);
        REGC: in STD_LOGIC_VECTOR(15 downto 0);
```

```
        REGD: in STD_LOGIC_VECTOR(15 downto 0);
        REGE: in STD_LOGIC_VECTOR(15 downto 0);
        REGF: in STD_LOGIC_VECTOR(15 downto 0);
        pointer: out std_logic_vector(15 downto 0));
end component PC;

component CPUregisters is
  Port (mclk: in STD_LOGIC;
        instruction: in STD_LOGIC_VECTOR(15 downto 0);
        freg0: out STD_LOGIC_VECTOR(15 downto 0);
        freg1: out STD_LOGIC_VECTOR(15 downto 0);
        freg2: out STD_LOGIC_VECTOR(15 downto 0);
        freg3: out STD_LOGIC_VECTOR(15 downto 0);
        freg4: out STD_LOGIC_VECTOR(15 downto 0);
        freg5: out STD_LOGIC_VECTOR(15 downto 0);
        freg6: out STD_LOGIC_VECTOR(15 downto 0);
        freg7: out STD_LOGIC_VECTOR(15 downto 0);
        freg8: out STD_LOGIC_VECTOR(15 downto 0);
        freg9: out STD_LOGIC_VECTOR(15 downto 0);
        freg10: out STD_LOGIC_VECTOR(15 downto 0);
        freg11: out STD_LOGIC_VECTOR(15 downto 0);
        freg12: out STD_LOGIC_VECTOR(15 downto 0);
        freg13: out STD_LOGIC_VECTOR(15 downto 0);
        freg14: out STD_LOGIC_VECTOR(15 downto 0);
        freg15: out STD_LOGIC_VECTOR(15 downto 0);
        overflown: out std_logic;
        BTN: in STD_LOGIC;
        input: in STD_LOGIC_VECTOR(15 downto 0));

end component CPUregisters;

component RAM is

port(
  we   : in  std_logic; --write enable
  paddr : in  std_logic_vector(15 downto 0); --address for program
  eaddr: in std_logic_vector(15 downto 0); --address for editing
  saddr: in std_logic_vector(15 downto 0);--address for screen
  di   : in  std_logic_vector(15 downto 0); --data in
```

```
  instruction   : out std_logic_vector(15 downto 0); --instruction output
  screen: out std_logic_vector(15 downto 0)); --screen output

end component RAM ;

component screen is
  port (
    clk: in STD_LOGIC;
    byteselect: out std_logic_vector(15 downto 0);
    byte: in std_logic_vector(15 downto 0);
    o_h_sync: out STD_LOGIC;
    o_v_sync: out STD_LOGIC;
    blue: out STD_LOGIC_VECTOR(3 downto 0));
end component screen;

component RAMWriter is
  Port ( REG0: in STD_LOGIC_VECTOR(15 downto 0);
       REG1: in STD_LOGIC_VECTOR(15 downto 0);
       REG2: in STD_LOGIC_VECTOR(15 downto 0);
       REG3: in STD_LOGIC_VECTOR(15 downto 0);
       REG4: in STD_LOGIC_VECTOR(15 downto 0);
       REG5: in STD_LOGIC_VECTOR(15 downto 0);
       REG6: in STD_LOGIC_VECTOR(15 downto 0);
       REG7: in STD_LOGIC_VECTOR(15 downto 0);
       REG8: in STD_LOGIC_VECTOR(15 downto 0);
       REG9: in STD_LOGIC_VECTOR(15 downto 0);
       REGA: in STD_LOGIC_VECTOR(15 downto 0);
       REGB: in STD_LOGIC_VECTOR(15 downto 0);
       REGC: in STD_LOGIC_VECTOR(15 downto 0);
       REGD: in STD_LOGIC_VECTOR(15 downto 0);
       REGE: in STD_LOGIC_VECTOR(15 downto 0);
       REGF: in STD_LOGIC_VECTOR(15 downto 0);
       instruction: in STD_LOGIC_VECTOR(15 downto 0);
       we: out STD_LOGIC;
       eaddr: out STD_LOGIC_VECTOR(15 downto 0);
       d_out: out STD_LOGIC_VECTOR(15 downto 0));

end component RAMWriter;
```

```
component sevenseg is
    Port ( clk_in : in STD_LOGIC;
           instruction: STD_LOGIC_VECTOR(15 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0));
end component sevenseg;

component clockslower is
    Port ( clk : in STD_LOGIC;
           slowclk : out STD_LOGIC);
end component clockslower;



signal s_instruction: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg0: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg1: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg2: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg3: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg4: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg5: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg6: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg7: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg8: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg9: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg10: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg11: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg12: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg13: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg14: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_reg15: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal s_overflown: std_logic:='0';

signal s_pointer: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');



signal we_RAM: std_logic:='0';
signal e_RAM: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal di_RAM: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
```

```
signal s_RAM: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');

signal s_byte: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');


signal slowclk1: STD_LOGIC:='0';




begin


CPU: CPUregisters port
map(slowclk1,s_instruction,s_reg0,s_reg1,s_reg2,s_reg3,s_reg4,s_reg5,s_reg6,s_reg7,s_reg8,s_reg9,s_reg10,s_reg11,s_reg12,s_reg13,s_reg
14,s_reg15,s_overflown,BUTTON,SWINPUT);

Program_Counter: PC port
map(slowclk1,s_instruction,s_overflown,s_reg0,s_reg1,s_reg2,s_reg3,s_reg4,s_reg5,s_reg6,s_reg7,s_reg8,s_reg9,s_reg10,s_reg11,s_reg12,s
_reg13,s_reg14,s_reg15,s_pointer);

the_RAM: RAM port map(we_RAM,s_pointer,e_RAM,s_RAM,di_RAM,s_instruction,s_byte);

Graphical_Unit: screen port map(clk,s_RAM,s_byte,HSYNC,VSYNC,COLOR);

RAMbus: RAMWriter port
map(s_reg0,s_reg1,s_reg2,s_reg3,s_reg4,s_reg5,s_reg6,s_reg7,s_reg8,s_reg9,s_reg10,s_reg11,s_reg12,s_reg13,s_reg14,s_reg15,
s_instruction,we_RAM,e_RAM,di_RAM);

Instruction_Display: sevenseg port map(clk,s_instruction,AN,SEG);

clock_slower: clockslower port map(clk,slowclk1);

O_REGC<=s_reg12;


end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;




entity CPUregisters is


  Port (mclk: in std_logic;

        instruction: in STD_LOGIC_VECTOR(15 downto 0);

        freg0: out STD_LOGIC_VECTOR(15 downto 0);

        freg1: out STD_LOGIC_VECTOR(15 downto 0);

        freg2: out STD_LOGIC_VECTOR(15 downto 0);

        freg3: out STD_LOGIC_VECTOR(15 downto 0);

        freg4: out STD_LOGIC_VECTOR(15 downto 0);

        freg5: out STD_LOGIC_VECTOR(15 downto 0);

        freg6: out STD_LOGIC_VECTOR(15 downto 0);

        freg7: out STD_LOGIC_VECTOR(15 downto 0);

        freg8: out STD_LOGIC_VECTOR(15 downto 0);
```

```
        freg9: out STD_LOGIC_VECTOR(15 downto 0);
        freg10: out STD_LOGIC_VECTOR(15 downto 0);
        freg11: out STD_LOGIC_VECTOR(15 downto 0);
        freg12: out STD_LOGIC_VECTOR(15 downto 0);
        freg13: out STD_LOGIC_VECTOR(15 downto 0);
        freg14: out STD_LOGIC_VECTOR(15 downto 0);
        freg15: out STD_LOGIC_VECTOR(15 downto 0);
        overflown: out std_logic;
        BTN: in STD_LOGIC;
        input: in STD_LOGIC_VECTOR(15 downto 0));

end CPUregisters;

architecture Behavioral of CPUregisters is

component customRegister16 is
  Port (data: in STD_LOGIC_VECTOR(15 downto 0);
      enable: in STD_LOGIC;
      output: out STD_LOGIC_VECTOR(15 downto 0);
      clk: in STD_LOGIC);
end component customRegister16;



component MUX16x16 is
  Port (sel:in STD_LOGIC_VECTOR(3 downto 0);
                          a  :in STD_LOGIC_VECTOR(15 downto 0);
                          b  :in STD_LOGIC_VECTOR(15 downto 0);
                          c  :in STD_LOGIC_VECTOR(15 downto 0);
                          d  :in STD_LOGIC_VECTOR(15 downto 0);
                          e  :in STD_LOGIC_VECTOR(15 downto 0);
                          f  :in STD_LOGIC_VECTOR(15 downto 0);
                          g  :in STD_LOGIC_VECTOR(15 downto 0);
                          h  :in STD_LOGIC_VECTOR(15 downto 0);
                          z  :in STD_LOGIC_VECTOR(15 downto 0);
                          j  :in STD_LOGIC_VECTOR(15 downto 0);
                          k  :in STD_LOGIC_VECTOR(15 downto 0);
                          l  :in STD_LOGIC_VECTOR(15 downto 0);
                          m  :in STD_LOGIC_VECTOR(15 downto 0);
                          n  :in STD_LOGIC_VECTOR(15 downto 0);
```

```
                o  :in STD_LOGIC_VECTOR(15 downto 0);
                p  :in STD_LOGIC_VECTOR(15 downto 0);
                output: out STD_LOGIC_VECTOR(15 downto 0));
end component MUX16x16;

component ALU is
  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        control: in std_logic_vector(3 downto 0);
        sum: out std_logic_vector(15 downto 0);
        overflown: out std_logic);
end component ALU;

component SgnExtnd8to16 is
  Port ( i_data: in STD_LOGIC_VECTOR(7 downto 0);
       o_data: out STD_LOGIC_VECTOR(15 downto 0));
end component SgnExtnd8to16;

component MUX2x8 is
  Port (a: in STD_LOGIC_VECTOR(15 downto 0);
      b: in STD_LOGIC_VECTOR(15 downto 0);
      sel: in STD_LOGIC;
      output: out STD_LOGIC_VECTOR(15 downto 0));
end component MUX2x8;

component Decoder4b is
  Port (number: in STD_LOGIC_VECTOR(3 downto 0);
      opcode : in std_logic_vector(3 downto 0);
      a: out STD_LOGIC;
      b: out STD_LOGIC;
      c: out STD_LOGIC;
      d: out STD_LOGIC;
      e: out STD_LOGIC;
      f: out STD_LOGIC;
      g: out STD_LOGIC;
      h: out STD_LOGIC;
      z: out STD_LOGIC;
      j: out STD_LOGIC;
      k: out STD_LOGIC;
```

```
      l: out STD_LOGIC;
      m: out STD_LOGIC;
      n: out STD_LOGIC;
      o: out STD_LOGIC;
      p: out STD_LOGIC);
end component Decoder4b;


--register things
constant reg0 : std_logic_vector(3 downto 0) := "0000";
constant reg1 : std_logic_vector(3 downto 0) := "0001";
constant reg2 : std_logic_vector(3 downto 0) := "0010";
constant reg3 : std_logic_vector(3 downto 0) := "0011";
constant reg4 : std_logic_vector(3 downto 0) := "0100";
constant reg5 : std_logic_vector(3 downto 0) := "0101";
constant reg6 : std_logic_vector(3 downto 0) := "0110";
constant reg7 : std_logic_vector(3 downto 0) := "0111";
constant reg8 : std_logic_vector(3 downto 0) := "1000";
constant reg9 : std_logic_vector(3 downto 0) := "1001";
constant reg10: std_logic_vector(3 downto 0) := "1010";
constant reg11: std_logic_vector(3 downto 0) := "1011";
constant reg12: std_logic_vector(3 downto 0) := "1100";
constant reg13: std_logic_vector(3 downto 0) := "1101";
constant reg14: std_logic_vector(3 downto 0) := "1110";
constant reg15: std_logic_vector(3 downto 0) := "1111";

signal en0:STD_LOGIC:='1';
signal en1:STD_LOGIC:='1';
signal en2:STD_LOGIC:='1';
signal en3:STD_LOGIC:='1';
signal en4:STD_LOGIC:='1';
signal en5:STD_LOGIC:='1';
signal en6:STD_LOGIC:='1';
signal en7:STD_LOGIC:='1';
signal en8:STD_LOGIC:='1';
signal en9:STD_LOGIC:='1';
signal en10:STD_LOGIC:='1';
signal en11:STD_LOGIC:='1';
signal en12:STD_LOGIC:='1';
```

```
signal en13:STD_LOGIC:='1';
signal en14:STD_LOGIC:='1';
signal en15:STD_LOGIC:='1';

signal o_Reg0 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg1 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg2 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg3 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg4 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg5 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg6 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg7 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg8 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg9 : STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg10: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg11: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg12: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg13: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg14: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal o_Reg15: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
--end of register things



--ALU things
signal i_PortA: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');
signal i_portB: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');

signal o_ALU: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');

signal data: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
--end of ALU things


signal extended_signal: STD_LOGIC_VECTOR(15 downto 0):=(others =>'0');

signal iswrt: std_logic:='0';
```

```
begin

iswrt <= instruction(15) and (not instruction(14)) and (not instruction(13)) and (not instruction(12));

register0 : customRegister16 port map(data,en0,o_Reg0,mclk);
register1 : customRegister16 port map(data,en1,o_Reg1,mclk);
register2 : customRegister16 port map(data,en2,o_Reg2,mclk);
register3 : customRegister16 port map(data,en3,o_Reg3,mclk);
register4 : customRegister16 port map(data,en4,o_Reg4,mclk);
register5 : customRegister16 port map(data,en5,o_Reg5,mclk);
register6 : customRegister16 port map(input,BTN,o_Reg6,mclk);
register7 : customRegister16 port map(data,en7,o_Reg7,mclk);
register8 : customRegister16 port map(data,en8,o_Reg8,mclk);
register9 : customRegister16 port map(data,en9,o_Reg9,mclk);
register10: customRegister16 port map(data,en10,o_Reg10,mclk);
register11: customRegister16 port map(data,en11,o_Reg11,mclk);
register12: customRegister16 port map(data,en12,o_Reg12,mclk);
register13: customRegister16 port map(data,en13,o_Reg13,mclk);
register14: customRegister16 port map(data,en14,o_Reg14,mclk);
register15: customRegister16 port map(data,en15,o_Reg15,mclk);

ins_PortA: MUX16x16 port map(instruction(7 downto
4),o_Reg0,o_Reg1,o_Reg2,o_Reg3,o_Reg4,o_Reg5,o_Reg6,o_Reg7,o_Reg8,o_Reg9,o_Reg10,o_Reg11,o_Reg12,o_Reg13,o_Reg14,o_Reg15, i_PortA);

ins_PortB: MUX16x16 port map(instruction(3 downto
0),o_Reg0,o_Reg1,o_Reg2,o_Reg3,o_Reg4,o_Reg5,o_Reg6,o_Reg7,o_Reg8,o_Reg9,o_Reg10,o_Reg11,o_Reg12,o_Reg13,o_Reg14,o_Reg15, i_PortB);

ins_ALU:ALU port map(i_portA,i_portB,instruction(15 downto 12),o_ALU,overflown);

ins_SignExtender: SgnExtnd8to16 port map(instruction(7 downto 0),extended_signal);

ChooseALUorWRT: MUX2x8 port map(o_ALU,extended_signal,iswrt,data);

AddrDecoder: Decoder4b port map(instruction(11 downto 8),instruction(15 downto
12),en0,en1,en2,en3,en4,en5,en6,en7,en8,en9,en10,en11,en12,en13,en14,en15);

freg0 <= o_Reg0;
freg1 <= o_Reg1;
```

```
freg2 <= o_Reg2;
freg3 <= o_Reg3;
freg4 <= o_Reg4;
freg5 <= o_Reg5;
freg6 <= o_Reg6;
freg7 <= o_Reg7;
freg8 <= o_Reg8;
freg9 <= o_Reg9;
freg10<= o_Reg10;
freg11<= o_Reg11;
freg12<= o_Reg12;
freg13<= o_Reg13;
freg14<= o_Reg14;
freg15<= o_Reg15;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity customRegister16 is
  Port (data: in STD_LOGIC_VECTOR(15 downto 0);
                        enable: in STD_LOGIC;
                        output: out STD_LOGIC_VECTOR(15 downto 0);
      clk: in STD_LOGIC);
end customRegister16;




architecture Behavioral of customRegister16 is
signal s_out: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');



begin
process(enable,data,clk)
```

```
begin
if falling_edge(clk) then
  if enable='1'then
    s_out<=data;
  end if;
end if;
end process;
output<=s_out;


end Behavioral;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity MUX16x16 is
  Port (sel:in STD_LOGIC_VECTOR(3 downto 0);
                        a  :in STD_LOGIC_VECTOR(15 downto 0);
                        b  :in STD_LOGIC_VECTOR(15 downto 0);
                        c  :in STD_LOGIC_VECTOR(15 downto 0);
                        d  :in STD_LOGIC_VECTOR(15 downto 0);
                        e  :in STD_LOGIC_VECTOR(15 downto 0);
                        f  :in STD_LOGIC_VECTOR(15 downto 0);
                        g  :in STD_LOGIC_VECTOR(15 downto 0);
                        h  :in STD_LOGIC_VECTOR(15 downto 0);
                        z  :in STD_LOGIC_VECTOR(15 downto 0);
                        j  :in STD_LOGIC_VECTOR(15 downto 0);
                        k  :in STD_LOGIC_VECTOR(15 downto 0);
                        l  :in STD_LOGIC_VECTOR(15 downto 0);
                        m  :in STD_LOGIC_VECTOR(15 downto 0);
                        n  :in STD_LOGIC_VECTOR(15 downto 0);
                        o  :in STD_LOGIC_VECTOR(15 downto 0);
                        p  :in STD_LOGIC_VECTOR(15 downto 0);
                        output: out STD_LOGIC_VECTOR(15 downto 0));
end MUX16x16;


architecture Behavioral of MUX16x16 is
```

```
begin
process(sel,a,b,c,d,e,f,g,h,z,j,k,l,m,n,o,p)
begin
case sel is
when x"0"=> output <= a;
when x"1"=> output <= b;
when x"2"=> output <= c;
when x"3"=> output <= d;
when x"4"=> output <= e;
when x"5"=> output <= f;
when x"6"=> output <= g;
when x"7"=> output <= h;
when x"8"=> output <= z;
when x"9"=> output <= j;
when x"A"=> output <= k;
when x"B"=> output <= l;
when x"C"=> output <= m;
when x"D"=> output <= n;
when x"E"=> output <= o;
when x"F"=> output <= p;
when others=> output<=x"FFFF";
end case;
end process;




end Behavioral;
-- most sincere thanks to prof. atalar for RAM
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity RAM is


port(
```

```
  we   : in  std_logic; --write enable
  paddr : in  std_logic_vector(15 downto 0); --address for program
  eaddr: in std_logic_vector(15 downto 0); --address for editing
  saddr: in std_logic_vector(15 downto 0);--address for screen
  di   : in  std_logic_vector(15 downto 0); --data in
  instruction   : out std_logic_vector(15 downto 0); --instruction output
  screen: out std_logic_vector(15 downto 0)); --screen output


end RAM ;
architecture behavioral of RAM is
type ram_type is array (1023 downto 0) of std_logic_vector(15 downto 0);
signal RAM : ram_type := (
    0=>x"8001",
    1=>x"8101",
    2=>x"8202",
    3=>x"0A60",
    4=>x"8909",
    5=>x"FFFF",
    6=>x"FFFF",
    7=>x"FFFF",
    8=>x"0B60",
    9=>x"0CAB",
    10=>x"0AB0",
    11=>x"0BC0",
    12=>x"A900",



    others => (others => '0')
);


begin
    process (paddr, saddr, we, eaddr, di)
    begin
        instruction <= RAM(conv_integer(paddr));
        screen <= RAM(conv_integer(saddr));


        if we = '1' then
```

```
            RAM(conv_integer(eaddr)) <= di;
        end if;
    end process;
end behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;


entity screen is
  port (
    clk: in STD_LOGIC;
    byteselect: out std_logic_vector(15 downto 0);
    byte: in std_logic_vector(15 downto 0);
    o_h_sync: out STD_LOGIC;
    o_v_sync: out STD_LOGIC;
    blue: out STD_LOGIC_VECTOR(3 downto 0));
end screen;


architecture Behavioral of screen is


  component clk_wiz_0 is
    port (
      clk_in1: in STD_LOGIC;
      clk_out1: out STD_LOGIC
    );
  end component;


  signal hi: integer:=0;
  signal vi: integer:=0;
  signal clk315: STD_LOGIC:='0';
  signal h_sync, v_sync: STD_LOGIC:='1';
```

```
begin
  o_h_sync <= h_sync;
  o_v_sync <= v_sync;
  byteselect<=std_logic_vector(TO_UNSIGNED((((hi/10)+(vi/10)*64)/16)+828,16));
  inst_clk_wiz: clk_wiz_0 port map(clk,clk315);


  process(clk315)
  begin
    if rising_edge(clk315) then
      --h_sync control
      if (hi>663) and (hi<704) then
        h_sync <= '0';
      else
        h_sync <= '1';
      end if;


      if (hi>=0) and (hi<640) then
      --DRAWING FUNCTION HERE
        if byte(((hi/10)+(vi/10)*64) mod 16)='1' then
            blue<="1111";
        else
            blue<="0000";
        end if;
      --DRAWING FUNCTION ENDS HERE
      else
        blue <= "0000";
      end if;







      --v_sync control
      if (vi=489) or (vi=490) then
        v_sync <= '0';
      else
        v_sync <= '1';
```

```
    end if;

    -- Update output signals
    o_h_sync <= h_sync;
    o_v_sync <= v_sync;

    -- Update counters
    if hi<832 then
      hi <= hi+1;
    else
      hi <= 0;
      if vi<520 then
        vi <= vi+1;
      else
        vi <= 0;
      end if;
    end if;
   end if;
  end process;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity RAMWriter is
  Port ( REG0: in STD_LOGIC_VECTOR(15 downto 0);
                        REG1: in STD_LOGIC_VECTOR(15 downto 0);
                        REG2: in STD_LOGIC_VECTOR(15 downto 0);
                        REG3: in STD_LOGIC_VECTOR(15 downto 0);
                        REG4: in STD_LOGIC_VECTOR(15 downto 0);
                        REG5: in STD_LOGIC_VECTOR(15 downto 0);
                        REG6: in STD_LOGIC_VECTOR(15 downto 0);
                        REG7: in STD_LOGIC_VECTOR(15 downto 0);
                        REG8: in STD_LOGIC_VECTOR(15 downto 0);
                        REG9: in STD_LOGIC_VECTOR(15 downto 0);
```

```
                REGA: in STD_LOGIC_VECTOR(15 downto 0);
                REGB: in STD_LOGIC_VECTOR(15 downto 0);
                REGC: in STD_LOGIC_VECTOR(15 downto 0);
                REGD: in STD_LOGIC_VECTOR(15 downto 0);
                REGE: in STD_LOGIC_VECTOR(15 downto 0);
                REGF: in STD_LOGIC_VECTOR(15 downto 0);
                instruction: in STD_LOGIC_VECTOR(15 downto 0);
                we: out STD_LOGIC;
                eaddr: out STD_LOGIC_VECTOR(15 downto 0);
                d_out: out STD_LOGIC_VECTOR(15 downto 0));

end RAMWriter;

architecture Behavioral of RAMWriter is




begin
PROCESS(instruction,REG0,REG1,REG2,REG3,REG4,REG5,REG6,REG7,REG8,REG9,REGA,REGB,REGC,REGD,REGE,REGF)
begin
if instruction(15 downto 12)="1011" then
we<='1';
case instruction(11 downto 8) is
when x"0"=> eaddr <= REG0;
when x"1"=> eaddr <= REG1;
when x"2"=> eaddr <= REG2;
when x"3"=> eaddr <= REG3;
when x"4"=> eaddr <= REG4;
when x"5"=> eaddr <= REG5;
when x"6"=> eaddr <= REG6;
when x"7"=> eaddr <= REG7;
when x"8"=> eaddr <= REG8;
when x"9"=> eaddr <= REG9;
when x"A"=> eaddr <= REGA;
when x"B"=> eaddr <= REGB;
when x"C"=> eaddr <= REGC;
when x"D"=> eaddr <= REGD;
```

```
when x"E"=> eaddr <= REGE;
when x"F"=> eaddr <= REGF;
end case;
case instruction(7 downto 4) is
when x"0"=> d_out <= REG0;
when x"1"=> d_out <= REG1;
when x"2"=> d_out <= REG2;
when x"3"=> d_out <= REG3;
when x"4"=> d_out <= REG4;
when x"5"=> d_out <= REG5;
when x"6"=> d_out <= REG6;
when x"7"=> d_out <= REG7;
when x"8"=> d_out <= REG8;
when x"9"=> d_out <= REG9;
when x"A"=> d_out <= REGA;
when x"B"=> d_out <= REGB;
when x"C"=> d_out <= REGC;
when x"D"=> d_out <= REGD;
when x"E"=> d_out <= REGE;
when x"F"=> d_out <= REGF;
end case;
else
we<='0';
end if;
end process;
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sevenseg is
    Port ( clk_in : in STD_LOGIC;
           instruction: STD_LOGIC_VECTOR(15 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0));
end sevenseg;

architecture Behavioral of sevenseg is
signal one_second_enable: std_logic;
```

```
signal displayed_number: STD_LOGIC_VECTOR (15 downto 0);

signal led_binary_coded_decimal: STD_LOGIC_VECTOR (3 downto 0);
signal counter: STD_LOGIC_VECTOR (19 downto 0);

signal top_2_of_counter: std_logic_vector(1 downto 0);

begin

displayed_number <= instruction;

process(led_binary_coded_decimal)
begin
   case led_binary_coded_decimal is ---   gfedcba
            when "0000" => seg <= "1000000";-- 1
            when "0001" => seg <= "1111001";-- 2
            when "0010" => seg <= "0100100";-- 3
            when "0011" => seg <= "0110000";-- 4
            when "0100" => seg <= "0011001";-- 5
            when "0101" => seg <= "0010010";-- 6
            when "0110" => seg <= "0000010";-- 7
            when "0111" => seg <= "1111000";-- 8
            when "1000" => seg <= "0000000";-- 9
            when "1001" => seg <= "0010000";-- a
            when "1010" => seg <= "0001000";-- b
            when "1011" => seg <= "0000011";-- c
            when "1100" => seg <= "1000110";-- d
            when "1101" => seg <= "0100001";-- e
            when "1110" => seg <= "0000110";-- f
            when "1111" => seg <= "0001110";-- g
            when others => seg <= "1111111";
   end case;
end process;

process(clk_in)
begin
```

```
if(rising_edge(clk_in)) then
        counter <= counter + 1;
    end if;
end process;
top_2_of_counter <= counter(19 downto 18);


process(top_2_of_counter, displayed_number)
begin
    case top_2_of_counter is
    when "00" =>
        an <= "0111";


        led_binary_coded_decimal <= displayed_number(15 downto 12);


    when "01" =>
        an <= "1011";


        led_binary_coded_decimal <= displayed_number(11 downto 8);


    when "10" =>
        an <= "1101";


        led_binary_coded_decimal <= displayed_number(7 downto 4);




    when "11" =>
        an <= "1110";


        led_binary_coded_decimal <= displayed_number(3 downto 0);


    when others =>
        an <= "1111";


        led_binary_coded_decimal <= displayed_number(3 downto 0);
```

```
    end case;
end process;


end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity clockslower is
    Port ( clk : in STD_LOGIC;
           slowclk : out STD_LOGIC);
end clockslower;


architecture Behavioral of clockslower is
    signal counter_reg : STD_LOGIC_VECTOR (25 downto 0) := (others => '0');
begin
    process(clk)
    begin
        if rising_edge(clk) then
            counter_reg <= counter_reg + 1;
        end if;
    end process;


    slowclk <= counter_reg(25);
end Behavioral;



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity SgnExtnd8to16 is
  Port ( i_data: in STD_LOGIC_VECTOR(7 downto 0);
                        o_data: out STD_LOGIC_VECTOR(15 downto 0));
```

```
end SgnExtnd8to16;

architecture Behavioral of SgnExtnd8to16 is

begin

gen_large:for i in 15 downto 8 generate
o_data(i)<=i_data(7);
end generate gen_large;

gen_same: for i in 7 downto 0 generate
o_data(i)<=i_data(i);
end generate gen_same;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity MUX2x8 is
  Port (a: in STD_LOGIC_VECTOR(15 downto 0);
                b: in STD_LOGIC_VECTOR(15 downto 0);
                sel: in STD_LOGIC;
                output: out STD_LOGIC_VECTOR(15 downto 0));
end MUX2x8;

architecture Behavioral of MUX2x8 is

begin

gen_mux: for i in 15 downto 0 generate
output(i)<= ((not sel) and a(i)) or (sel and b(i));
end generate gen_mux;
```

```
end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;




entity decoder4b is

  Port (number: in STD_LOGIC_VECTOR(3 downto 0);

        opcode: in STD_LOGIC_VECTOR(3 downto 0);

                        a: out STD_LOGIC;

                        b: out STD_LOGIC;

                        c: out STD_LOGIC;

                        d: out STD_LOGIC;

                        e: out STD_LOGIC;

                        f: out STD_LOGIC;

                        g: out STD_LOGIC;

                        h: out STD_LOGIC;

                        z: out STD_LOGIC;

                        j: out STD_LOGIC;

                        k: out STD_LOGIC;

                        l: out STD_LOGIC;

                        m: out STD_LOGIC;

                        n: out STD_LOGIC;

                        o: out STD_LOGIC;

                        p: out STD_LOGIC);

end decoder4b;


architecture Behavioral of decoder4b is


begin




a <= (not number(3)) and (not number(2)) and (not number(1)) and (not number(0)) and (not opcode(3) or (opcode(3) and (not op code(2)) and (not opcode(1)) and (not opcode(0))));

b <= (not number(3)) and (not number(2)) and (not number(1)) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2 )) and (not opcode(1)) and (not opcode(0))));
```

```
c <= (not number(3)) and (not number(2)) and number(1) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

d <= (not number(3)) and (not number(2)) and number(1) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

e <= (not number(3)) and number(2) and (not number(1)) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

f <= (not number(3)) and number(2) and (not number(1)) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

g <= (not number(3)) and number(2) and number(1) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

h <= (not number(3)) and number(2) and number(1) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

z <= number(3) and (not number(2)) and (not number(1)) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

j <= number(3) and (not number(2)) and (not number(1)) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

k <= number(3) and (not number(2)) and number(1) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

l <= number(3) and (not number(2)) and number(1) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

m <= number(3) and number(2) and (not number(1)) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

n <= number(3) and number(2) and (not number(1)) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

o <= number(3) and number(2) and number(1) and (not number(0)) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));

p <= number(3) and number(2) and number(1) and number(0) and (not opcode(3) or (opcode(3) and (not opcode(2)) and (not opcode(1)) and (not opcode(0))));



end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;




entity PC is

  Port (clk: in STD_LOGIC;

        instruction: in STD_LOGIC_VECTOR(15 downto 0);

        overflown: in STD_LOGIC;

        REG0: in STD_LOGIC_VECTOR(15 downto 0);

        REG1: in STD_LOGIC_VECTOR(15 downto 0);

        REG2: in STD_LOGIC_VECTOR(15 downto 0);

        REG3: in STD_LOGIC_VECTOR(15 downto 0);

        REG4: in STD_LOGIC_VECTOR(15 downto 0);
```

```
        REG5: in STD_LOGIC_VECTOR(15 downto 0);
        REG6: in STD_LOGIC_VECTOR(15 downto 0);
        REG7: in STD_LOGIC_VECTOR(15 downto 0);
        REG8: in STD_LOGIC_VECTOR(15 downto 0);
        REG9: in STD_LOGIC_VECTOR(15 downto 0);
        REGA: in STD_LOGIC_VECTOR(15 downto 0);
        REGB: in STD_LOGIC_VECTOR(15 downto 0);
        REGC: in STD_LOGIC_VECTOR(15 downto 0);
        REGD: in STD_LOGIC_VECTOR(15 downto 0);
        REGE: in STD_LOGIC_VECTOR(15 downto 0);
        REGF: in STD_LOGIC_VECTOR(15 downto 0);
        pointer: out std_logic_vector(15 downto 0));
end PC;

architecture Behavioral of PC is

component DFF_16bit is
    Port ( clk : in STD_LOGIC;
           d : in STD_LOGIC_VECTOR(15 downto 0);
           q : out STD_LOGIC_VECTOR(15 downto 0));
end component DFF_16bit;

component Incrementer is
    port (
        input: in std_logic_vector(15 downto 0);
        output: out std_logic_vector(15 downto 0)
    );
end component Incrementer;

component MUX2X16 is
  Port (a: in STD_LOGIC_VECTOR(15 downto 0);
        b: in STD_LOGIC_VECTOR(15 downto 0);
        sel: in STD_LOGIC:='0';
        d_out: out STD_LOGIC_VECTOR(15 downto 0));
end component MUX2X16;

component MUX16x16 is
```

```
  Port (sel:in STD_LOGIC_VECTOR(3 downto 0);
      a  :in STD_LOGIC_VECTOR(15 downto 0);
      b  :in STD_LOGIC_VECTOR(15 downto 0);
      c  :in STD_LOGIC_VECTOR(15 downto 0);
      d  :in STD_LOGIC_VECTOR(15 downto 0);
      e  :in STD_LOGIC_VECTOR(15 downto 0);
      f  :in STD_LOGIC_VECTOR(15 downto 0);
      g  :in STD_LOGIC_VECTOR(15 downto 0);
      h  :in STD_LOGIC_VECTOR(15 downto 0);
      z  :in STD_LOGIC_VECTOR(15 downto 0);
      j  :in STD_LOGIC_VECTOR(15 downto 0);
      k  :in STD_LOGIC_VECTOR(15 downto 0);
      l  :in STD_LOGIC_VECTOR(15 downto 0);
      m  :in STD_LOGIC_VECTOR(15 downto 0);
      n  :in STD_LOGIC_VECTOR(15 downto 0);
      o  :in STD_LOGIC_VECTOR(15 downto 0);
      p  :in STD_LOGIC_VECTOR(15 downto 0);
      output: out STD_LOGIC_VECTOR(15 downto 0));
end component MUX16x16;




signal oinc: std_logic_vector(15 downto 0):=(others=>'0');
signal odff: std_logic_vector(15 downto 0):=(others=>'0');
signal omux: std_logic_vector(15 downto 0):=(others=>'0');
signal control: STD_LOGIC:='0';
signal registers:std_logic_vector(15 downto 0):=(others=>'0');




begin
control<= ((instruction(15) and (not instruction(14)) and (not instruction(13)) and instruction(12) and (overflown)) or (instruction(15) and
(not instruction(14)) and (instruction(13)) and (not instruction(12))));




gen: DFF_16bit port map(clk,omux,odff);
gen1: MUX2X16 port map(registers,oinc,control,omux);
```

```
gen2: Incrementer port map(odff,oinc);

gen3: MUX16x16 port map(instruction(11 downto
8),REG0,REG1,REG2,REG3,REG4,REG5,REG6,REG7,REG8,REG9,REGA,REGB,REGC,REGD,REGE,REGF,registers);



pointer<=odff;



end Behavioral;

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity DFF_16bit is

    Port ( clk : in STD_LOGIC;

           d : in STD_LOGIC_VECTOR(15 downto 0);

           q : out STD_LOGIC_VECTOR(15 downto 0));

end DFF_16bit;



architecture Behavioral of DFF_16bit is



    signal q_int : STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');



begin



    process(clk)

    begin

        if rising_edge(clk) then

              q_int <= d;

        end if;

    end process;



    q <= q_int;



end Behavioral;



library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;




entity MUX2X16 is
  Port (a: in STD_LOGIC_VECTOR(15 downto 0);
                        b: in STD_LOGIC_VECTOR(15 downto 0);
                        sel: in STD_LOGIC;
                        d_out: out STD_LOGIC_VECTOR(15 downto 0));
end MUX2X16;

architecture Behavioral of MUX2X16 is

signal gated_a: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');
signal gated_b: STD_LOGIC_VECTOR(15 downto 0):=(others=>'0');



begin

gen0: for i in 15 downto 0 generate
gated_a(i)<=a(i) and sel;
end generate gen0;

gen1: for i in 15 downto 0 generate
gated_b(i)<=b(i) and (not sel);
end generate gen1;

gen2: for i in 15 downto 0 generate
d_out(i)<= gated_a(i) or gated_b(i);
end generate gen2;

end Behavioral;
library ieee;
use ieee.std_logic_1164.all;
```

```
entity Incrementer is
    port (
        input: in std_logic_vector(15 downto 0);
        output: out std_logic_vector(15 downto 0)
    );
end entity Incrementer;

architecture behavior of Incrementer is

component half_adder is
Port (a: in std_logic;
  b: in std_logic;
  sum: out std_logic;
  cout: out std_logic);
end component half_adder;

signal one: std_logic_vector(15 downto 0):=(others=>'0');
signal cout: std_logic_vector(15 downto 0):=(others=>'0');
signal sum: std_logic_vector(15 downto 0):=(others=>'0');



begin
one<=x"0001";



hmm1: half_adder port map(input(0),one(0),sum(0),cout(0));
gen: for i in 1 to 15 generate
hmm: half_adder port map(input(i),cout(i-1),sum(i),cout(i));
end generate;

output<=sum;
end architecture behavior;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity half_adder is
Port (a: in std_logic;
  b: in std_logic;
  sum: out std_logic;
  cout: out std_logic);
end half_adder;

architecture Behavioral of half_adder is

begin
sum<= a xor b;
cout<= a and b;



end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity ALU is
  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        control: in std_logic_vector(3 downto 0);
        sum: out std_logic_vector(15 downto 0);
        overflown: out std_logic);
end ALU;

architecture Behavioral of ALU is
```

```
component adderSubber
    Port (a: in std_logic_vector(15 downto 0);
          b: in std_logic_vector(15 downto 0);
          d: in std_logic;
          sum: out std_logic_vector(15 downto 0);
          overflown: out std_logic);
end component;



component SHLer
  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );
end component;

component SHRer
  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );
end component;







component ANDer

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );


end component;
```

```
component ORer

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end component;

component XORer

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end component;

component NEGer

  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );
```

```
end component;


component MUX16x16 is
  Port (sel:in STD_LOGIC_VECTOR(3 downto 0);
      a  :in STD_LOGIC_VECTOR(15 downto 0);
      b  :in STD_LOGIC_VECTOR(15 downto 0);
      c  :in STD_LOGIC_VECTOR(15 downto 0);
      d  :in STD_LOGIC_VECTOR(15 downto 0);
      e  :in STD_LOGIC_VECTOR(15 downto 0);
      f  :in STD_LOGIC_VECTOR(15 downto 0);
      g  :in STD_LOGIC_VECTOR(15 downto 0);
      h  :in STD_LOGIC_VECTOR(15 downto 0);
      z  :in STD_LOGIC_VECTOR(15 downto 0);
      j  :in STD_LOGIC_VECTOR(15 downto 0);
      k  :in STD_LOGIC_VECTOR(15 downto 0);
      l  :in STD_LOGIC_VECTOR(15 downto 0);
      m  :in STD_LOGIC_VECTOR(15 downto 0);
      n  :in STD_LOGIC_VECTOR(15 downto 0);
      o  :in STD_LOGIC_VECTOR(15 downto 0);
      p  :in STD_LOGIC_VECTOR(15 downto 0);
      output: out STD_LOGIC_VECTOR(15 downto 0));
end component MUX16x16;
```

```
    signal o_adderSubber: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_ANDer: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_ORer: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_XORer: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_NEGer: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_SHRer: std_logic_vector(15 downto 0):=(others=>'0');
    signal o_SHLer: std_logic_vector(15 downto 0):=(others=>'0');



    signal zeroes: std_logic_vector(15 downto 0):="0000000000000000";




begin
zeroes<="0000000000000000";



adderSubber_ins: adderSubber port map(a,b,control(0),o_adderSubber, overflown);
ANDer_ins: ANDer port map(a,b,o_ANDer);
ORer_ins: ORer port map(a,b,o_ORer);
XORer_ins: XORer port map(a,b,o_XORer);
NEGer_ins: NEGer port map(a,o_NEGer);
SHRer_ins: SHRer port map(a,o_SHRer);
SHLer_ins: SHLer port map(a,o_SHLer);

UNITE: MUX16x16 port
map(control,o_adderSubber,o_adderSubber,o_ANDer,o_ORer,o_XORer,o_NEGer,o_SHRer,o_SHLer,o_adderSubber,o_adderSubber,o_A
NDer,o_ORer,o_XORer,o_NEGer,o_SHRer,o_SHLer,sum);

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity adderSubber is
    Port (a: in std_logic_vector(15 downto 0);
          b: in std_logic_vector(15 downto 0);
          d: in std_logic;
          sum: out std_logic_vector(15 downto 0);
          overflown: out std_logic);
end adderSubber;


architecture Behavioral of adderSubber is
    component fullAdder
        Port (
            a : in std_logic;
            b : in std_logic;
            cin : in std_logic;
            sum : out std_logic;
            cout : out std_logic
        );
    end component;


    signal carry: std_logic_vector(16 downto 0) :=(others=>'0');
    signal mb: std_logic_vector(15 downto 0) :=(others=>'0');
begin
    carry(0) <= d;
     gen_fa: for i in 0 to 15 generate
        muxedB: mb(i) <= ((not d) and  b(i)) or (d and (not b(i)));
        FA: fullAdder port map(mb(i), a(i), carry(i), sum(i), carry(i+1));
    end generate gen_fa;


    overflown <= carry(16);



end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity fullAdder is
  Port ( a: in std_logic;
         b: in std_logic;
         cin: in std_logic;
         sum: out std_logic;
         cout: out std_logic);
end fullAdder;

architecture Behavioral of fullAdder is



begin

sum <= a xor b xor cin;
cout <= (a and b) or (b and cin) or (a and cin);



end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity ANDer is

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end ANDer;

architecture Behavioral of ANDer is
```

```
begin

    gen_AND: for i in 0 to 15 generate
        sum(i)<=a(i) and b(i);
    end generate gen_AND;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity ORer is

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end ORer;

architecture Behavioral of ORer is

begin

    gen_OR: for i in 0 to 15 generate
        sum(i)<=a(i) or b(i);
    end generate gen_OR;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity XORer is

  Port (a: in std_logic_vector(15 downto 0);
        b: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end XORer;

architecture Behavioral of XORer is

begin

    gen_AND: for i in 0 to 15 generate
        sum(i)<=a(i) xor b(i);
    end generate gen_AND;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity NEGer is

  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end NEGer;

architecture Behavioral of NEGer is
```

```
begin

    gen_NEG: for i in 0 to 15 generate
        sum(i)<= not a(i);
    end generate gen_NEG;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;




entity SHRer is

  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end SHRer;

architecture Behavioral of SHRer is

begin
    sum(15)<='0';
    gen_SHR: for i in 0 to 14 generate
        sum(i)<= a(i+1);
    end generate gen_SHR;

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity SHLer is

  Port (a: in std_logic_vector(15 downto 0);
        sum: out std_logic_vector(15 downto 0)
        );

end SHLer;

architecture Behavioral of SHLer is

begin
    sum(0)<='0';
    gen_SHL: for i in 0 to 14 generate
        sum(i+1)<= a(i);
    end generate gen_SHL;

end Behavioral;
```