

Final LAB

Chue Zhang

Csc343 Fall 2021

Professor Gertner

Due Date : December 15th,2021 by 12:00pm

Contents

Introduction:	3
Objective:	4
Specifications:	5
Code:	6
Simulations:.....	13
Conclusion:.....	20

Introduction:

This is the final take home test to be done with one computing device by Chue Zhang in the FALL semester of 2021 with professor Izidor Garter belt. My goal for this final Lab is to extend upon the ALU that I have previously created to run multiple instructions that are stored inside a data memory which we will then feed into the instruction memory to process.

Objective:

The objective of this final lab is to show that my program can perform multi-step instructions, for example a data memory that stores 3 different instructions and runs them all. Ooh, black and yellow! Let's shake it up a little. We will not be using any bitwise operations and only using store word, load word and add operation that we have previously created. Barry! Breakfast is ready! Coming! Hang on a second. Hello.

Specifications:

What I will be doing is I will be loading an instruction from the data memory, incrementing the program counter by 4 so that we know what to read next in the data memory. With the current data instruction, I then feed it into the instruction memory and get a n instruction, then feed it into the instruction register to get my instructions. Later I will process this instruction and get my output. This process is repeated however many times are there are instructions as you will see below

Code:

Quartus Prime Lite Edition - C:/Users/czhan/Desktop/Schoolwork/Gertner/Zhang_CS343_FA21/ZHANG_NOVEMBER10_ALU/ZHANG_ALU - ZHANG_ALU

File Edit View Project Assignments Processing Tools Window Help

Search altera.com Close

Project Navigator Files

Files

- ./././././ZHANG_dataMem_12142020.vhd
- ./././././ZHANG_InstructionMemory_12142020.vhd
- ./././././ZHANG_InstructMem_12142020.vhd
- ./././././ZHANG_PCAdd4_12142020.vhd
- ./././././ZHANG_OCTOBER13_INTEGRATION_PROJECT/ZHAN
- ./././././ZHANG_OCTOBER13_INTEGRATION_PROJECT/zhan
- ./././././ZHANG_SEPTEMBER19_ADDERS/ZHANG_SEPTEMBE
- ZHANG_OR.vhd
- ZHANG_AND.vhd
- ZHANG_NOR.vhd
- ZHANG_sign_ext.vhd
- ZHANG_ALU.vhd
- ZHANG_alu_package.vhd
- ZHANG_miniCPU.vhd
- ZHANG_sl.vhd
- ZHANG_sr.vhd
- ZHANG_sra.vhd
- ZHANG_Main.vhd
- ZHANG_instruction_IR.vhd
- ZHANG_ext.vhd

Tasks Compilation

Task

- Compile Design
- Analysis & Synthesis
- Fitter (Place & Route)
- Assembler (Generate programming files)
- Timing Analysis
- EDA Netlist Writer

ZHANG_dataMem_12142020.vhd

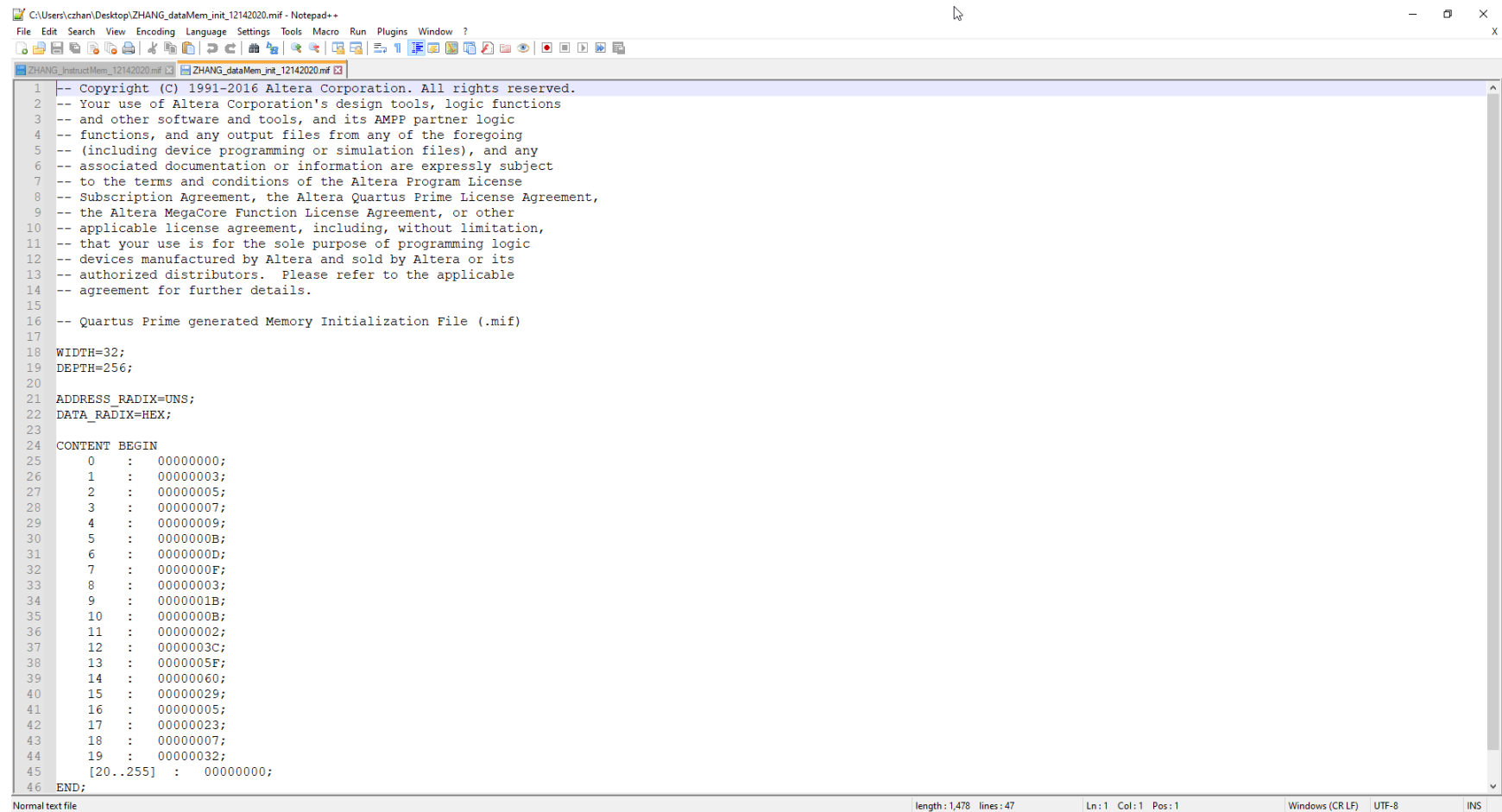
```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  LIBRARY altera_mf;
5  USE altera_mf.altera_mf_components.all;
6
7  ENTITY Chen_dataMem_12142020 IS
8  PORT
9  (
10     address      : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
11     clock         : IN STD_LOGIC := '1';
12     data         : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
13     wren         : IN STD_LOGIC;
14     q            : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
15 );
16 END Chen_dataMem_12142020;
17
18 ARCHITECTURE SYN OF Chen_dataMem_12142020 IS
19     SIGNAL sub_wire0 : STD_LOGIC_VECTOR (31 DOWNTO 0);
20
21 BEGIN
22     q <= sub_wire0(31 DOWNTO 0);
23
24     altsyncram_component : altsyncram
25     GENERIC MAP (
26         clock_enable_input_a => "BYPASS",
27         clock_enable_output_a => "BYPASS",
28         init_file => "Chen_dataMem_init_12142020.mif",
29         intended_device_family => "Cyclone V",
30         lpm_hint => "ENABLE_RUNTIME_MOD=NO",
31         lpm_type => "altsyncram",
32         numwords_a => 256,
33         operation_mode => "SINGLE_PORT",
34         outdata_aclr_a => "NONE",
35         outdata_reg_a => "UNREGISTERED",
36         power_up_uninitialized => "FALSE",
37         read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
38         widthad_a => 8,
39         width_a => 32,
40         width_byteena_a => 1
41     )
42     PORT MAP (
43         address_a => address,
44         clock0 => clock,
45         data_a => data,
46         wren_a => wren,
47         q_a => sub_wire0
48     );
49
50 END SYN;

```

0% 00:00:00

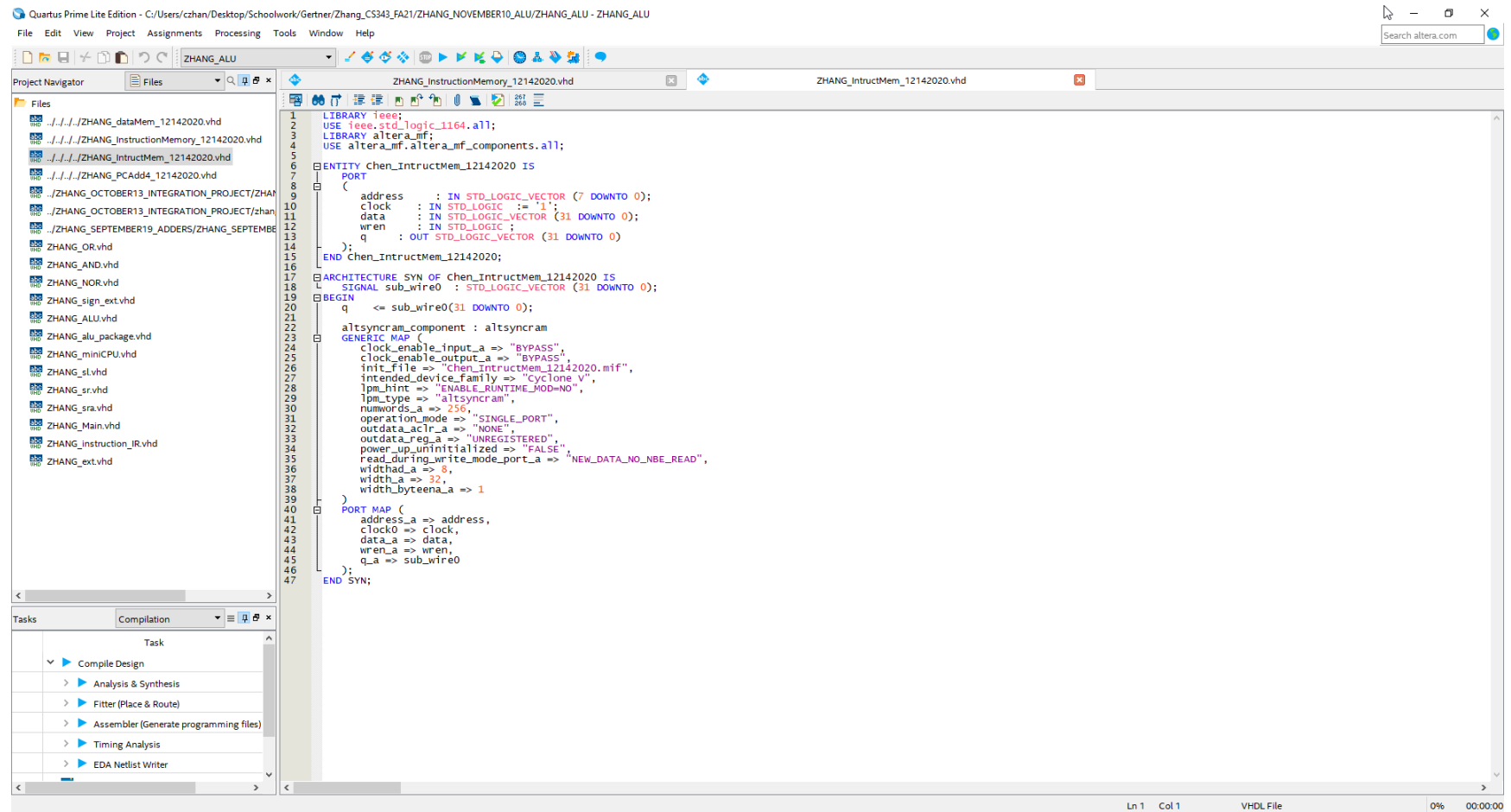
Data memory ram



```
1  -- Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, the Altera Quartus Prime License Agreement,
9  -- the Altera MegaCore Function License Agreement, or other
10 -- applicable license agreement, including, without limitation,
11 -- that your use is for the sole purpose of programming logic
12 -- devices manufactured by Altera and sold by Altera or its
13 -- authorized distributors. Please refer to the applicable
14 -- agreement for further details.
15
16 -- Quartus Prime generated Memory Initialization File (.mif)
17
18 WIDTH=32;
19 DEPTH=256;
20
21 ADDRESS_RADIX=UNS;
22 DATA_RADIX=HEX;
23
24 CONTENT BEGIN
25 0 : 00000000;
26 1 : 00000003;
27 2 : 00000005;
28 3 : 00000007;
29 4 : 00000009;
30 5 : 0000000B;
31 6 : 0000000D;
32 7 : 0000000F;
33 8 : 00000003;
34 9 : 0000001B;
35 10 : 0000000B;
36 11 : 00000002;
37 12 : 0000003C;
38 13 : 0000005F;
39 14 : 00000060;
40 15 : 00000029;
41 16 : 00000005;
42 17 : 00000023;
43 18 : 00000007;
44 19 : 00000032;
45 [20..255] : 00000000;
46 END;
```

Normal text file | length : 1,478 | lines : 47 | Ln : 1 | Col : 1 | Pos : 1 | Windows (CR LF) | UTF-8 | INS

Data memory mif in increments of 4



Instruction memory ram

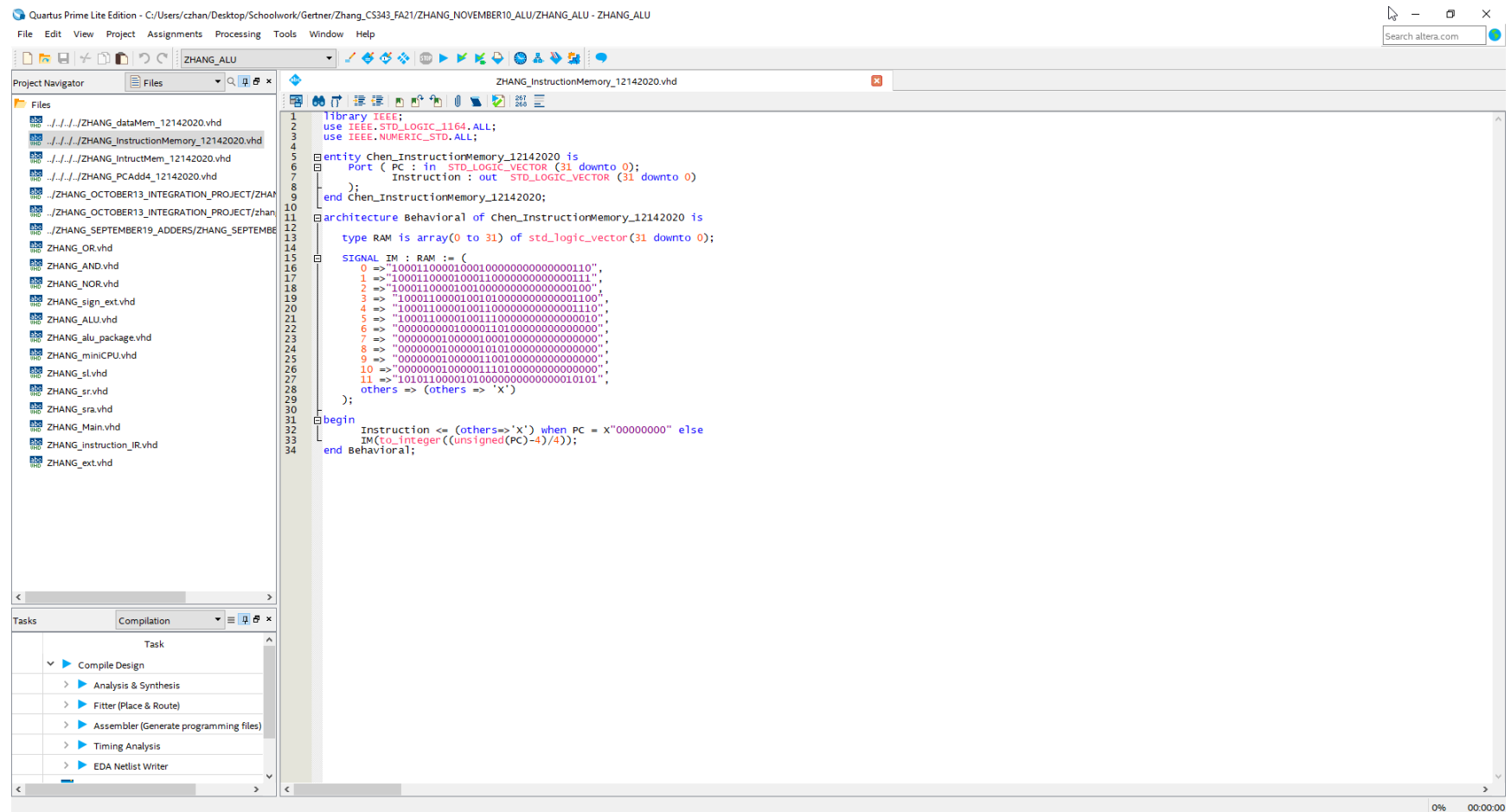

```

1  -- Copyright (C) 1991-2016 Altera Corporation. All rights reserved.
2  -- Your use of Altera Corporation's design tools, logic functions
3  -- and other software and tools, and its AMPP partner logic
4  -- functions, and any output files from any of the foregoing
5  -- (including device programming or simulation files), and any
6  -- associated documentation or information are expressly subject
7  -- to the terms and conditions of the Altera Program License
8  -- Subscription Agreement, the Altera Quartus Prime License Agreement,
9  -- the Altera MegaCore Function License Agreement, or other
10 -- applicable license agreement, including, without limitation,
11 -- that your use is for the sole purpose of programming logic
12 -- devices manufactured by Altera and sold by Altera or its
13 -- authorized distributors. Please refer to the applicable
14 -- agreement for further details.
15
16 -- Quartus Prime generated Memory Initialization File (.mif)
17
18 WIDTH=32;
19 DEPTH=32;
20
21 ADDRESS_RADIX=UNS;
22 DATA_RADIX=BIN;
23
24 CONTENT BEGIN
25   -- Load Instruction load six value from data memory
26   -- in reg[1] value is intialized to 4
27   0 : 100011000010001000000000000000110; --by adding to 6 to reg[1], data load from address 10 and store in to reg[2]
28   1 : 100011000010001100000000000000111; --by adding to 7 to reg[1], data load from address 11 and store in to reg[3]
29   2 : 100011000010010000000000000000100; --by adding to 4 to reg[1], data load from address 8 and store in to reg[4]
30   3 : 100011000010010100000000000000100; --by adding to 12 to reg[1], data load from address 16 and store in to reg[5]
31   4 : 100011000010011000000000000000110; --by adding to 14 to reg[1], data load from address 18 and store in to reg[6]
32   5 : 100011000010011100000000000000010; --by adding to 2 to reg[1], data load from address 6 and store in to reg[6]
33
34   -- Addition Instruction summing six
35   6 : 000000000100001101000000000000000; --Add reg 2 to reg 3 and store into reg 8
36   7 : 000000010000010001000000000000000; --Add reg 8 to reg 4 and store into reg 8
37   8 : 000000010000010101000000000000000; --Add reg 8 to reg 5 and store into reg 8
38   9 : 000000010000011001000000000000000; --Add reg 8 to reg 6 and store into reg 8
39   10 : 000000010000011101000000000000000; --Add reg 8 to reg 7 and store into reg 8
40
41   --Store Instruction
42   11 : 101011000011000000000000000010101;--store reg 8 into address 4+21
43   [12..31] : 0;
44 END;
45

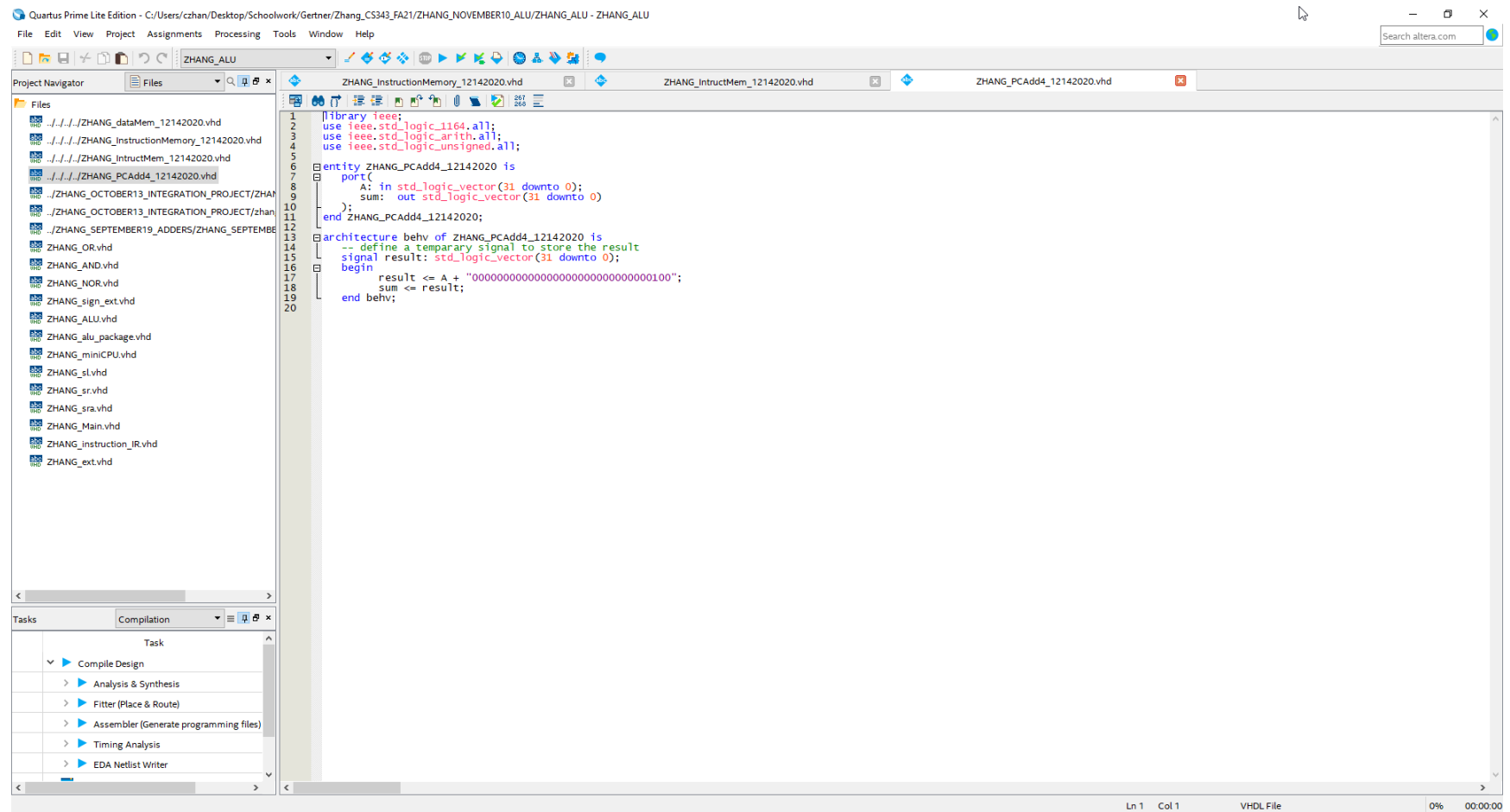
```

Normal text file length: 2,431 lines: 45 Ln: 1 Col: 1 Pos: 1 Windows (CR LF) UTF-8 INS

Instruction memory mif



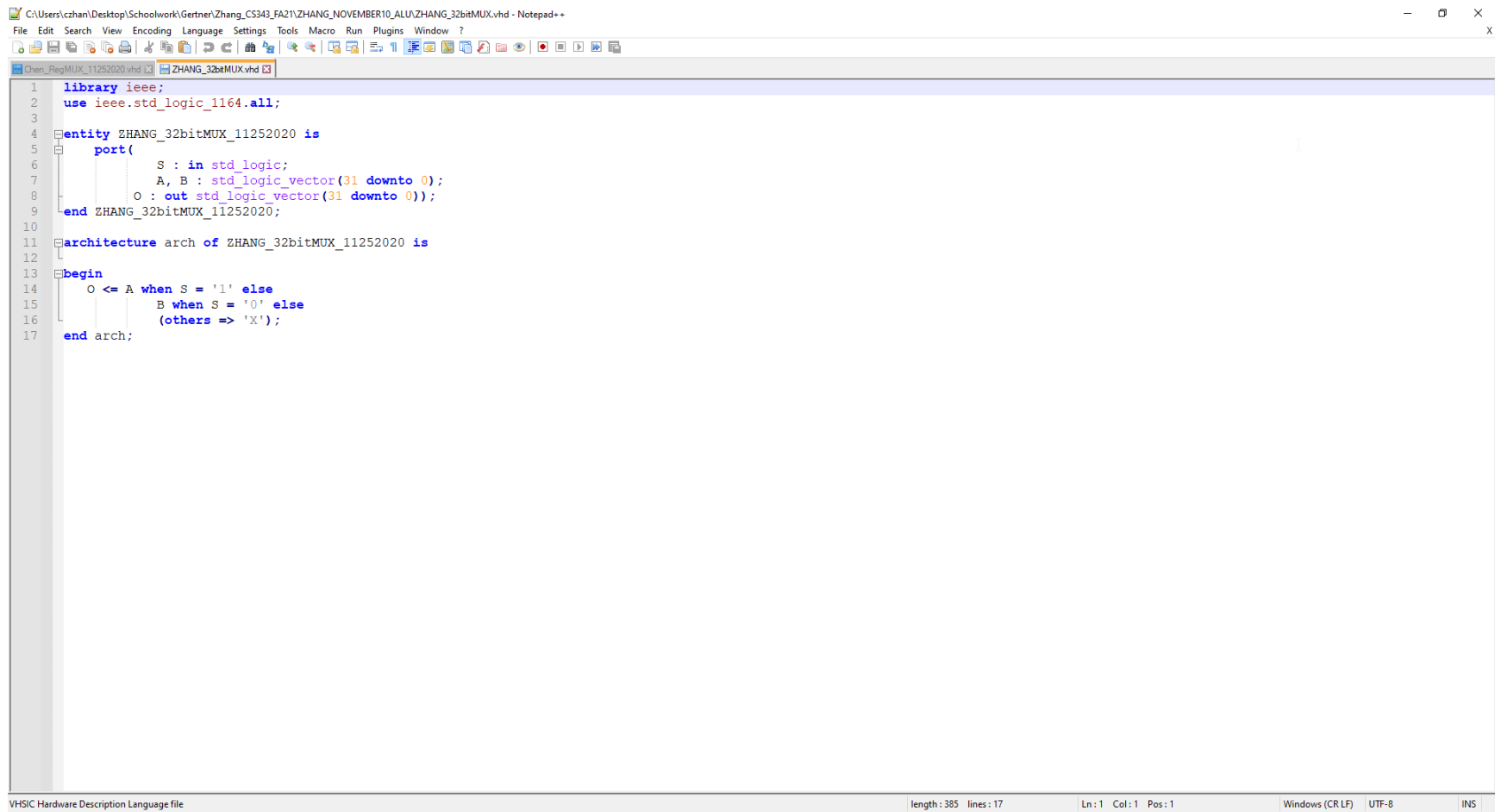
RAM with basic instructions



The screenshot displays the Quartus Prime Lite Edition interface. The top menu bar includes File, Edit, View, Project, Assignments, Processing, Tools, Window, and Help. The toolbar contains various icons for file operations and simulation. The Project Navigator on the left lists several VHDL files, including ZHANG_PCAdd4_12142020.vhd, which is currently selected. The main editor window shows the VHDL code for ZHANG_PCAdd4_12142020.vhd. The code defines a 32-bit adder entity with two 32-bit inputs, A and B, and a 32-bit output, sum. The architecture, named 'behv', uses a temporary signal 'result' to store the sum of A and B. The code is as follows:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.std_logic_unsigned.all;
5
6 entity ZHANG_PCAdd4_12142020 is
7   port(
8     A: in std_logic_vector(31 downto 0);
9     B: in std_logic_vector(31 downto 0);
10    sum: out std_logic_vector(31 downto 0)
11  );
12 end ZHANG_PCAdd4_12142020;
13
14 architecture behv of ZHANG_PCAdd4_12142020 is
15   -- define a temporary signal to store the result
16   signal result: std_logic_vector(31 downto 0);
17 begin
18   result <= A + B;
19   sum <= result;
20 end behv;
```

The bottom status bar shows the current position as Ln 1, Col 1, and the file type as VHDL File. The bottom right corner displays 0% and 00:00:00.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity ZHANG_32bitMUX_11252020 is
5     port(
6         S : in std_logic;
7         A, B : std_logic_vector(31 downto 0);
8         O : out std_logic_vector(31 downto 0));
9 end ZHANG_32bitMUX_11252020;
10
11 architecture arch of ZHANG_32bitMUX_11252020 is
12
13 begin
14     O <= A when S = '1' else
15         B when S = '0' else
16         (others => 'X');
17 end arch;
```

VHDL Hardware Description Language file | length : 385 | lines : 17 | Ln : 1 | Col : 1 | Pos : 1 | Windows (CR LF) | UTF-8 | INS

Simulations:

I will be performing a series of add operations of $1 + 1$ of instructions that were stored in the data memory and instruction memory ram vdhI

files. Every time an instruction is performed, the program counter is incremented by 4 as you will be able to see below in the simulations section

PC:

ModelSim - INTEL FPGA STARTER EDITION 2020.1

File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help

100 ps

Layout Simulate

ColumnLayout [Default]

sim - Default

Design

- zhang_pcadd4_12...
- line__17
- zhang
- standard
- textio
- std_logic_1164
- std_logic_arith
- std_logic_unsigned

Objects

Name	Value
A	00000000000000000000000000000000
sum	00000000000000000000000000000000
result	00000000000000000000000000000000

Processes (Active)

Name	Type (filtered)	State
------	-----------------	-------

Wave - Default

Msgs	00000000	00000004	00000008	0000000C
/zhang_pcadd4_12...	00000000	00000004	00000008	0000000C

Now 300 ps

Cursor 1 0 ps

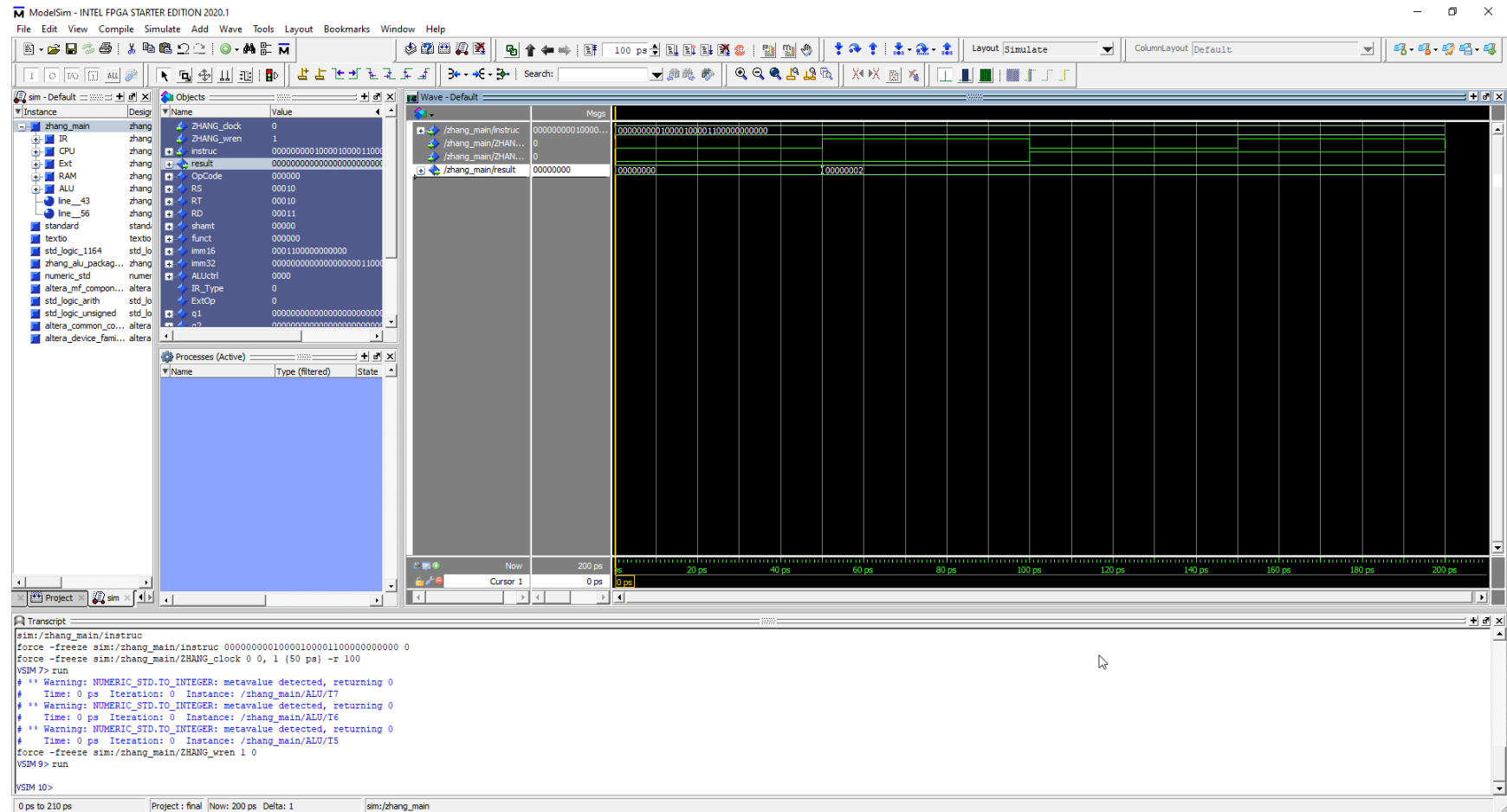
0 ps to 315 ps

Project: final Now: 300 ps Delta: 0 sim:/zhang_pcadd4_12142020

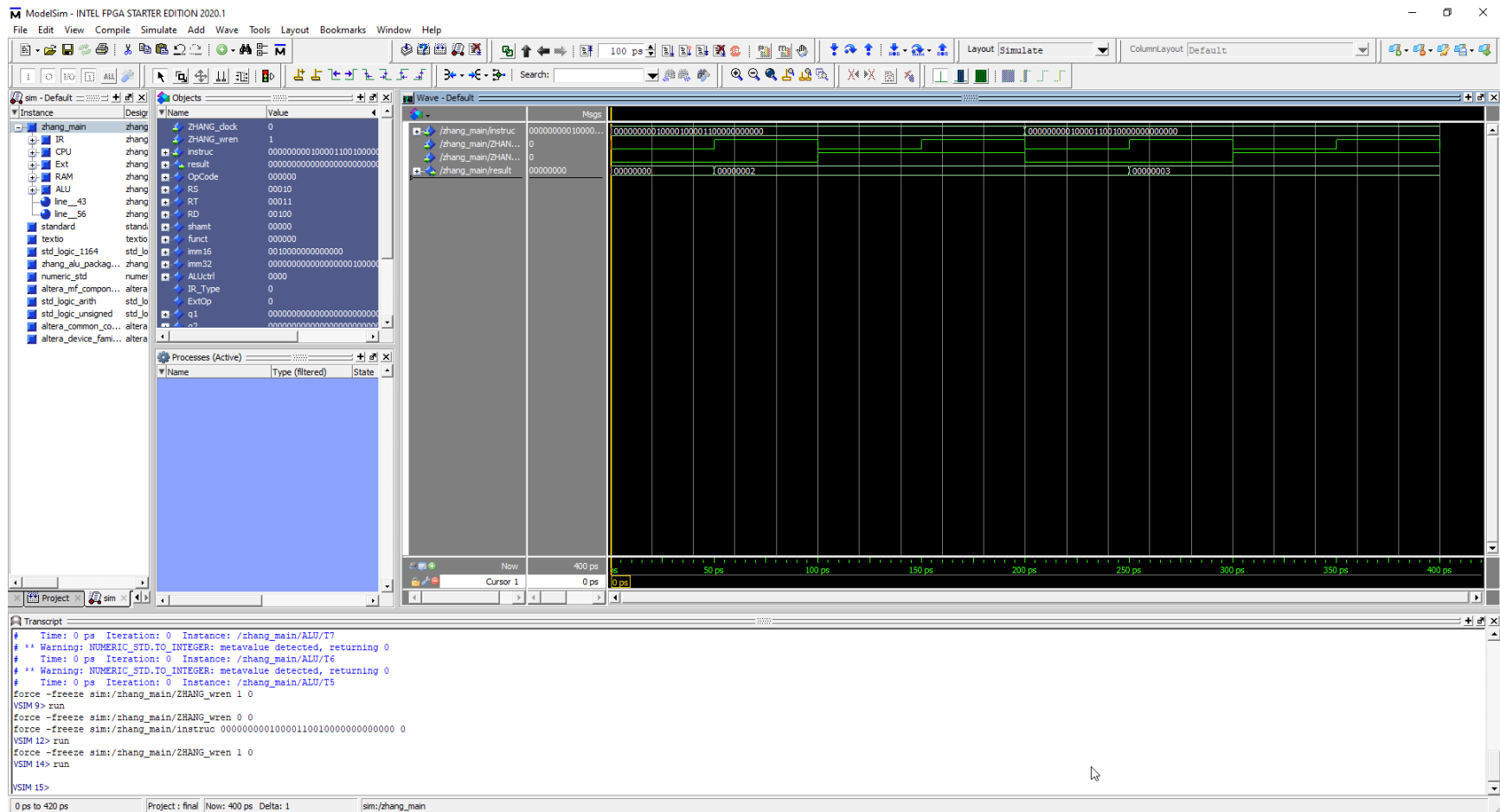
Transcript

```

force -freeze sim:/zhang_pcadd4_12142020/A X"00000004" 0
VSIM8> run
VSIM9> run -continue
VSIM10> run
VSIM11> restart
# ** Note: (vsim-12125) Error and warning message counts have been reset to '0' because of 'restart'.
force -freeze sim:/zhang_pcadd4_12142020/A X"00000000" 0
VSIM13> run
force -freeze sim:/zhang_pcadd4_12142020/A X"00000004" 0
VSIM15> run
force -freeze sim:/zhang_pcadd4_12142020/A X"00000008" 0
VSIM17> run
VSIM18>
  
```

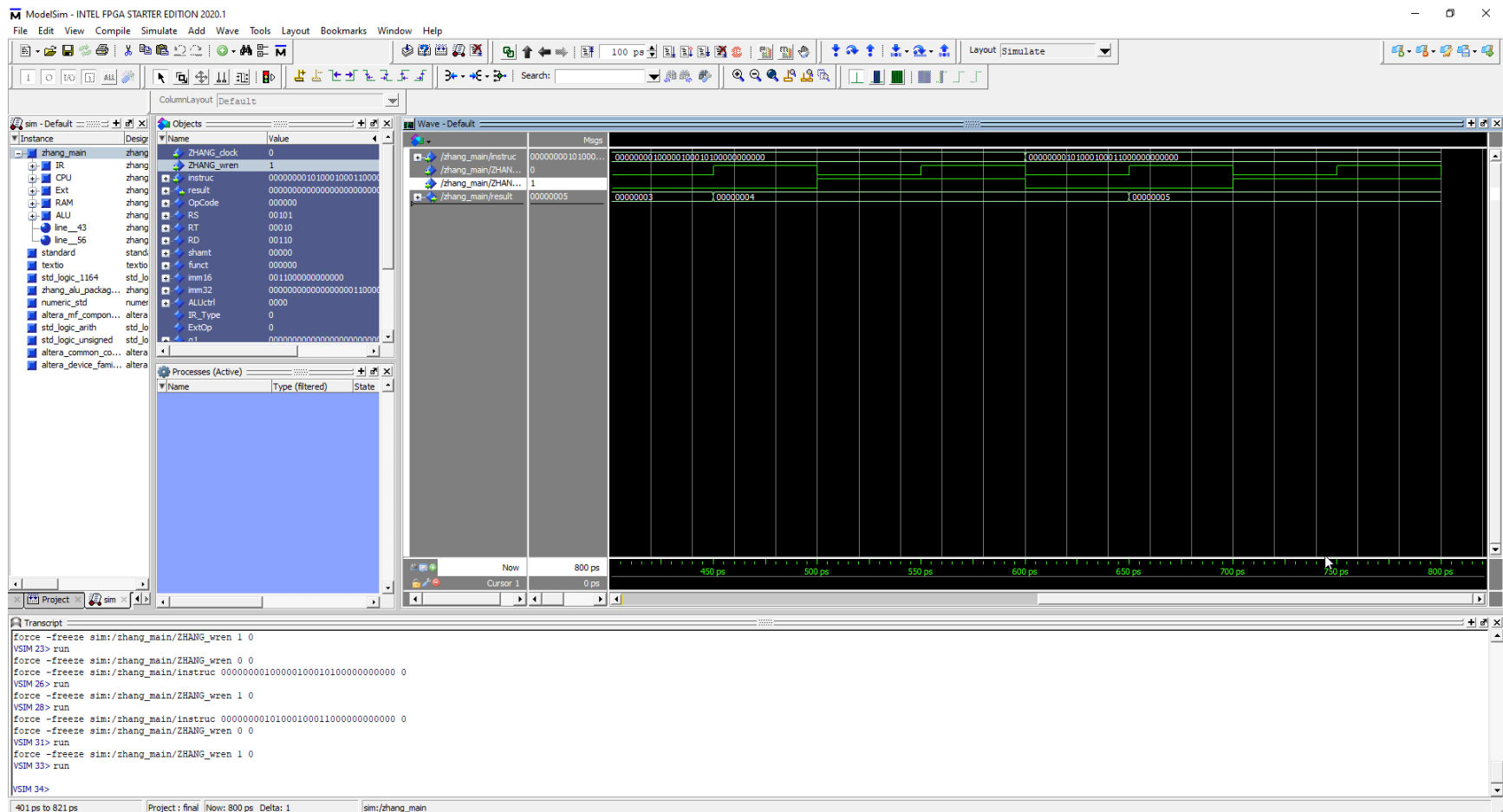
Instructions(1st – 5th):

This is the first instruction of $1 + 1$, if you look at the result, 2 is outputted then stored into the 2nd register slot

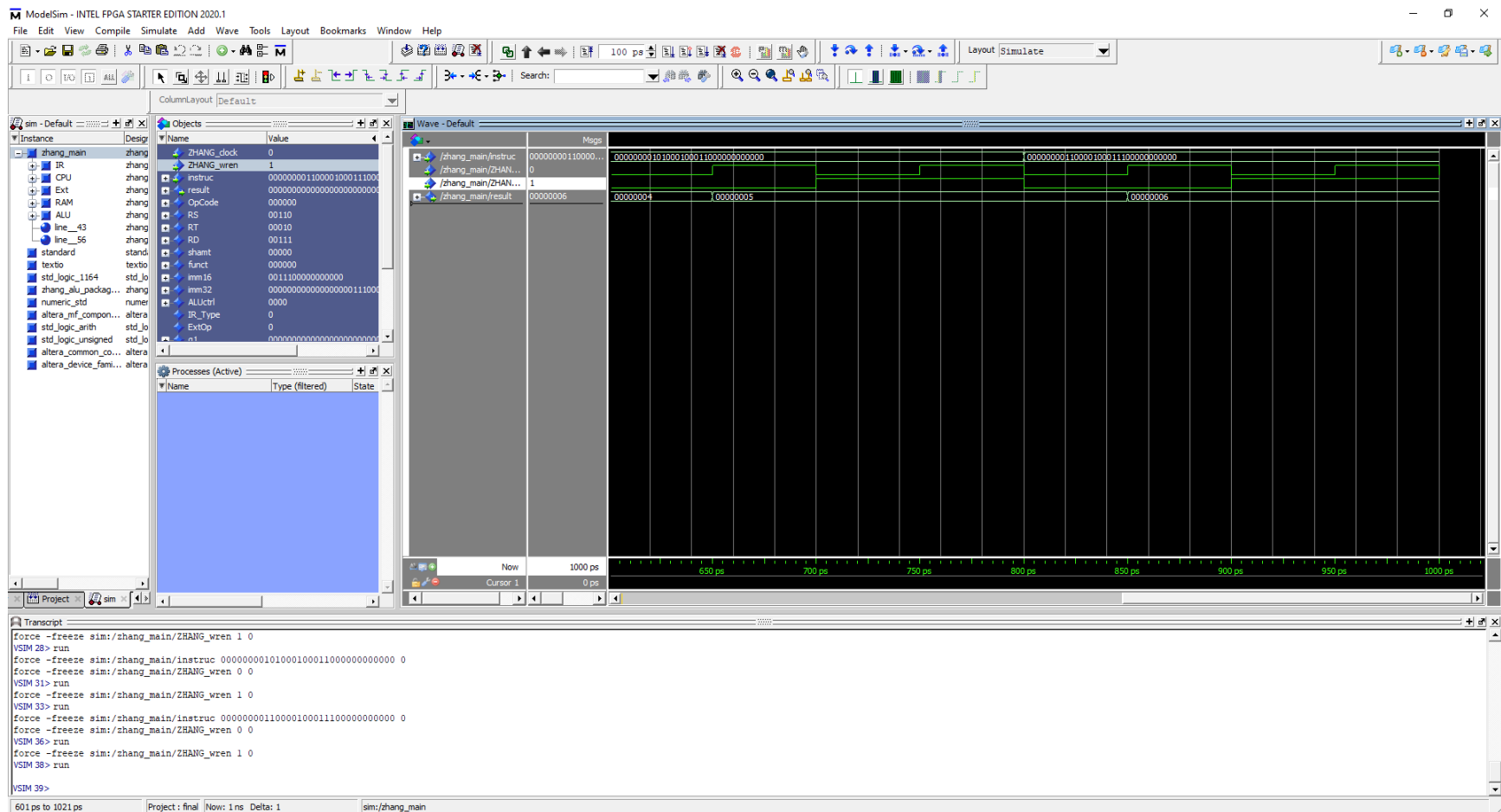


This is the second instruction which adds the 2nd register slot with 1 then stores the result into the 3rd register slot





This is the 4th instruction where we perform 4th register slot + 1 and then we store it in the 5th register slot



Lastly, for the last instruction, we perform 5th register slot + 1 and then store it in the 6th register slot

Conclusion:

In conclusion, I have learned about using program counters, instruction memory, data memory and how to store and load those data. This is a very interesting process, and I am grateful for have learnt more about this.