

**EEE373 – Digital System Design**

Department of Electrical Engineering & Electronics

**Assignment 3\_Report**

-MIPS Processor Design-

|  |  |
| --- | --- |
| **Student Name:** | Yu-Cheng. Lin |
| **Student ID:** | 201447973 |

**Abstract**

Assignment 3 is about MIP processor design, divided into part A and part B. The Part A is to modify the MIPS assembly language program so that the program displays the lowest 8 digits of the student ID on the 7-segment display of the DE2 board. The Part B requires that the design of MIPS be modified, and specified new instructions should be added when a limited number of MIPS instructions can be used normally. The purpose of completing Assignment 3 is to understand the internal operation of the MIPS processor, learn how to use the ASM design method to design a digital system and use Verilog hardware to implement a digital system description language.

**Table of Contents**

[**Abstract**](#_Toc512788800)

[Table of Contents](#_Toc512788801)

[**1** **Part\_A 1**](#_Toc512788802)

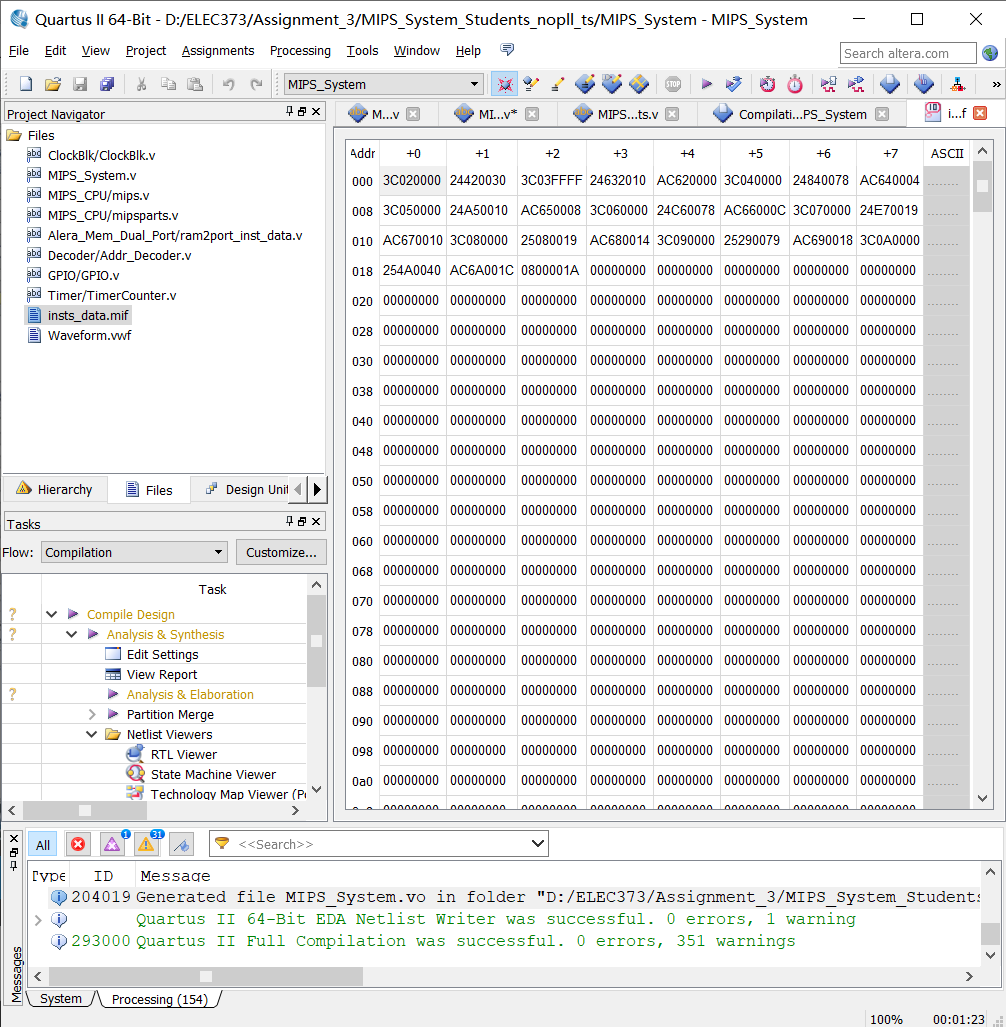
[**2** **Part\_B 3**](#_Toc512788802)

[**3** **Discussion 6**](#_Toc512788802)

[**Appendix 7**](#_Toc512788802)

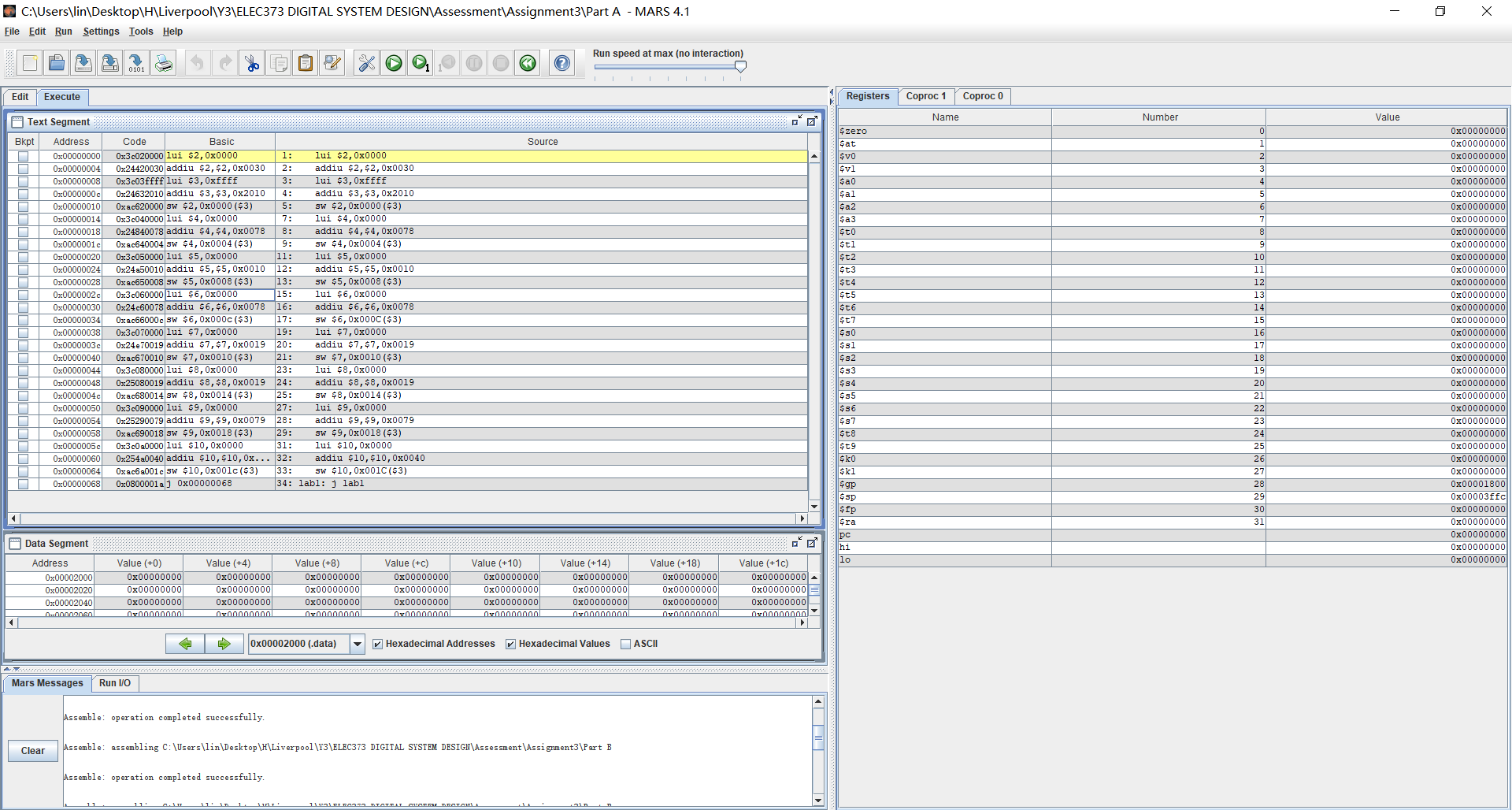
**1. Part\_A**

The ID used in Part A is 201447973 and the lowest 8 digits of it are 01447973. These 8 digits need to be changed into corresponding hexadecimal codes 40, 79, 19, 19, 78, 10, 78, 30 in order to be displayed on the 7-segment display of the DE2 board. Therefore, on the existing MIPS system, the MIPS assembly language program is modified in ‘insts\_data’ as shown in Figure 1.

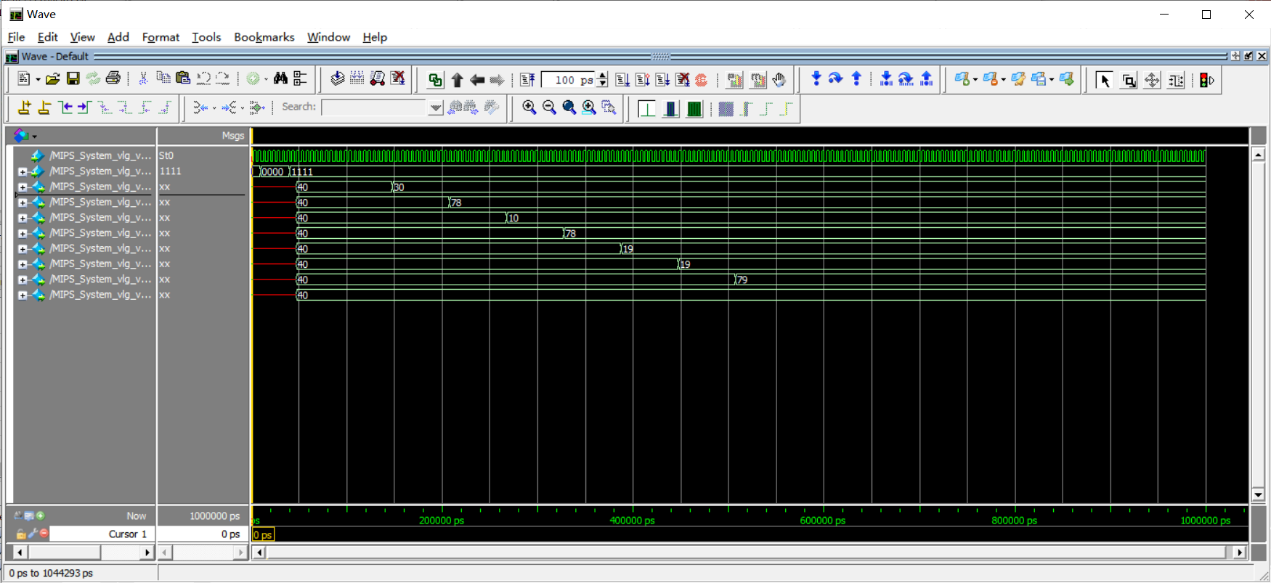


**Figure 1:** Encoding used in Part A (Code)

The encoding of Figure 1 is obtained by decoding the MIPS assembly code of Figure 2. Since the **lui** instruction can only generate the first 16 bits, use the **addi** instruction to add the lowest 16 bits. The **$3** register is used to determine the HEX0 to HEX7 of the DE2 board. Finally, the **sw** instruction is used to store the hexadecimal code to be displayed into the **$3** register. Note that the **$3** register changes the stored HEX through +4, where 0xFFFF\_2010 is HEX0.



**Figure 2:** MIPS Assembly Code (Source)

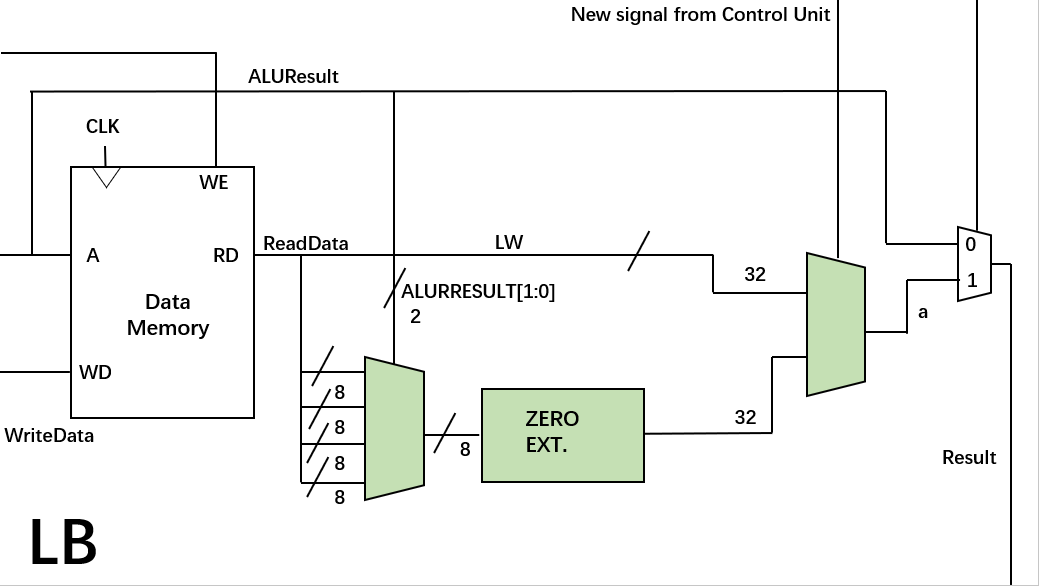
Figure 3 is the result of the simulation of the MIPS system through ModelSim.

**Figure 3:** The simulation results of Part A

When KEY is ‘low’, the system is activated. The first one is that the output of HEX0 is 30 in hexadecimal and 3 is displayed on the DE2 board.

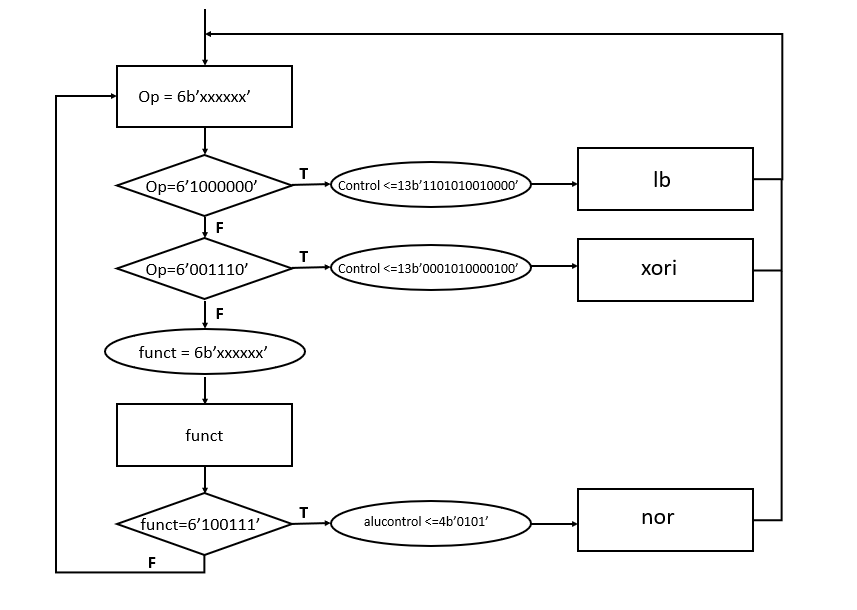
**2. Part\_B**

In Part B, the design of MIPS is modified to add **nor**, **xori** and **lb** instructions. Mainly designed according to the block diagram in Figure 4.



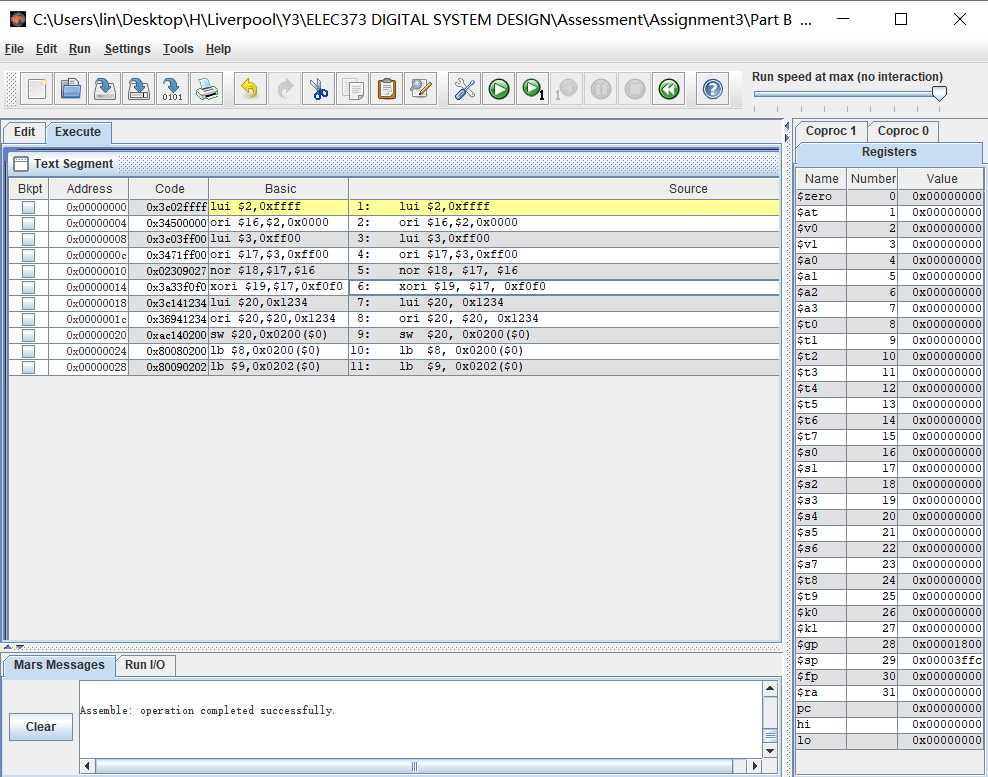
**Figure 4:** The Block Diagram of Part B

The code is in the appendix. The ASM chart is drawn through the code as shown in Figure 5. Each instruction has its own code. According to the input code, the MIPS system can determine the instruction used. However, **lb**, **xori** and **nor** are different types of instructions, thereby **op** and **funct** are used to distinguish between them. The **lb** and **xori** are I-type instructions. The **nor** is R-type instruction.

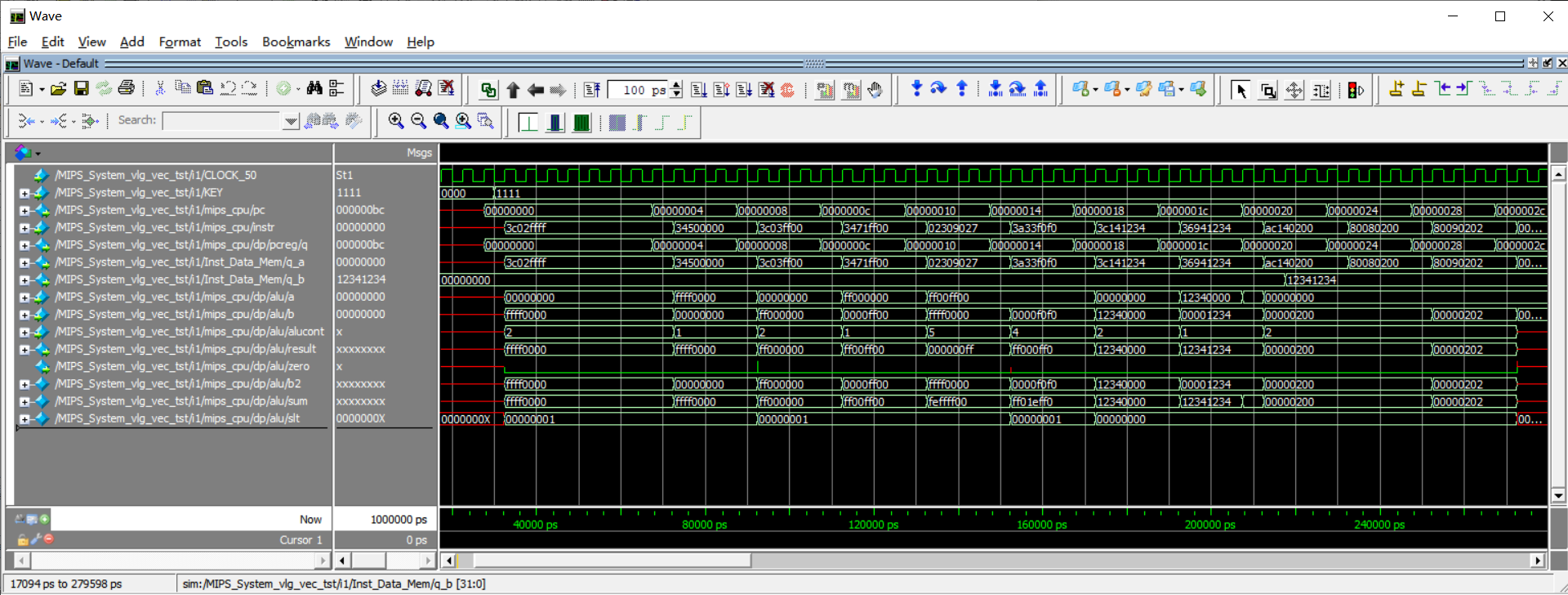


**Figure 5:** The ASM of Part B

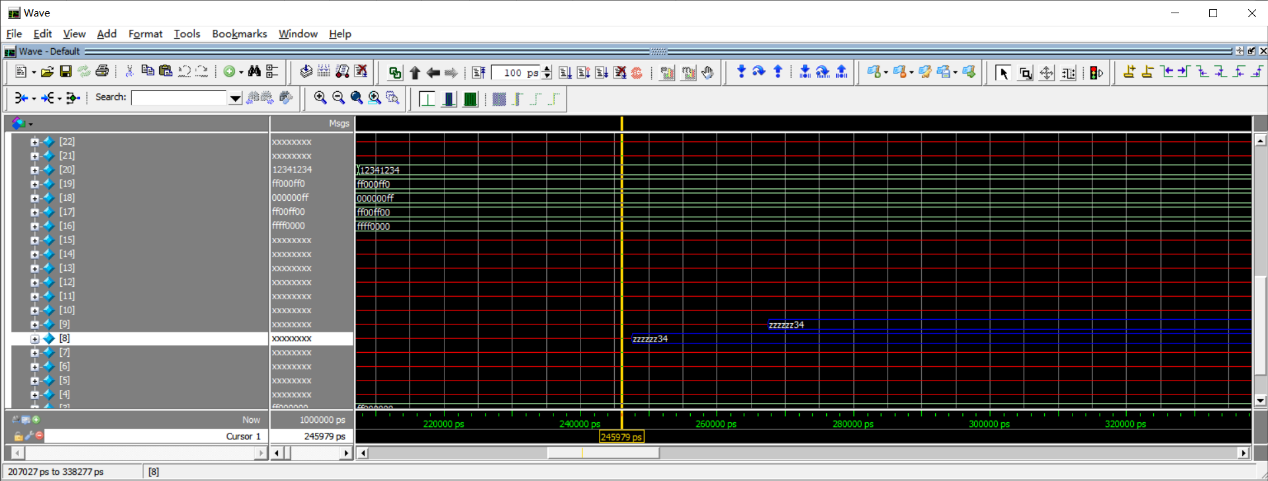
In order to test the performance of the instruction, the code in Figure 6 was entered into the ‘insts\_data’.



**Figure 6:** The encoding of Part B

The results of the test are shown in Figure 7. The result of nor instruction will be ‘1’ only when a=’0’ and b=’0’. The result of the **xori** instruction is ‘1’ only when a=’1’ and b=’0’ or a=’0’ and b=’1’. The **lb** instruction only accesses the lowest 8 bits to the register, and the remaining 24 bits are expanded according to the 7-bit sign

**Figure 7:** The simulation results of Part B

It can be seen that by using the **nor** instruction on 0xff00\_ff00 and 0xffff\_0000, the result is 0x0000\_00ff. Similarly, by using the **xori** instruction for 0xff00\_ff00 and 0x0000\_f0f0, the result is 0xff00\_0ff0. However, the figure does not see the result of the **lb** instruction on 0x1234\_1234. Therefore, rt is added for simulation, as shown in Figure 8.

**Figure 8:** The simulation results of rt

According to the code, the obtained result of **lb** instruction is placed in register 8 and register 9. The results are 0xzzzz\_zz34.

**3. Discussion**

The tasks are all completed, the only problem should be the **lb** command. The tasks are all completed, the only problem should be the **lb** command. The code to extend the 7-bit sign is missing. The normal **lb** instruction will load the lowest 8-bit code, and the remaining 24 bits are determined according to the eighth bit. If it is ‘1’, the others are all ‘1’, and vice versa, if it is ‘0’, the others are all ‘0’. In the test, the lowest 8 bits are ‘0011\_0100’, and the result should be 0x0000\_0034.

**Appendix**

