MIPS

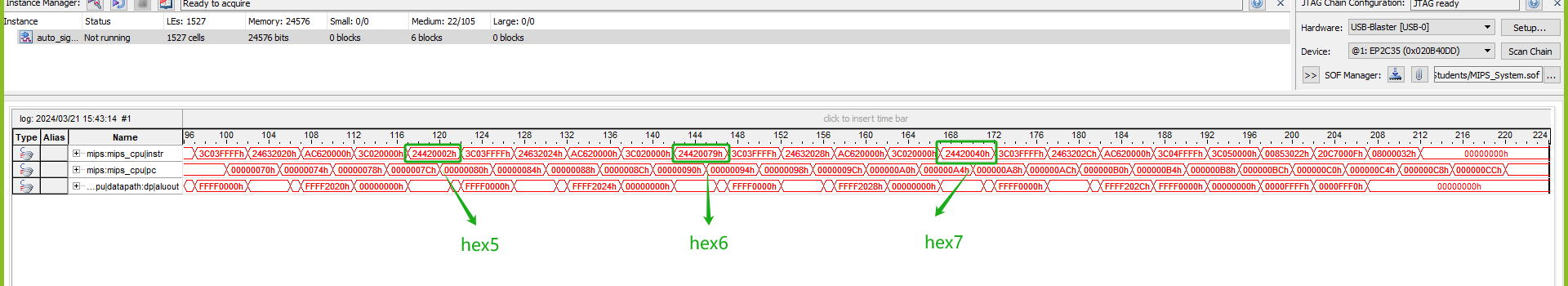
# Part A

1. Modify the MIPS assembly language program so that the program displays the lowest 8 digits of your ID on the DE2 board 7 segment display.

|  |  |
| --- | --- |
| **assemble code** | **machine code** |
| hex0 displays 9 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x10 | 24420010 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2010 | 24632010 |
| sw $v0,0x0000($v1) | ac620000 |
| hex1 dispalys 5 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x12 | 24420012 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2014 | 24632014 |
| sw $v0,0x0000($v1) | ac620000 |
| hex2 displays 3 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x30 | 24420030 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2018 | 24632018 |
| sw $v0,0x0000($v1) | ac620000 |
| hex3 displays 7 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x78 | 24420078 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x201C | 2463201c |
| sw $v0,0x0000($v1) | ac620000 |
| hex4 displays 7 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x78 | 24420078 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2020 | 24632020 |
| sw $v0,0x0000($v1) | ac620000 |
| hex5 dsiplays 6 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x02 | 24420002 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2024 | 24632024 |
| sw $v0,0x0000($v1) | ac620000 |
| hex6 displays 1 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x79 | 24420079 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x2028 | 24632028 |
| sw $v0,0x0000($v1) | ac620000 |
| hex7 displays 0 | |
| lui $v0,0 | 3c020000 |
| addiu,$v0,$v0,0x40 | 24420040 |
| lui,$v1,0xFFFF | 3c03ffff |
| addiu,$v1,$v1,0x202C | 2463202c |
| sw $v0,0x0000($v1) | ac620000 |

1. Show that your program functions correctly by taking a screen shot(s) of the SignalTap Logic analyser showing your program executing your modified programme. Make sure that the instruction and programme counter can be read.

* The highlighted assembly codes within the green rectangles in the figure execute the storage of the contents of hex5, hex6, and hex7 into memory.



1. In your report you should include your assembly language code and a screen dump of the SignalTap Logic analyser. Also include a photograph of the 7 segment displays showing your ID.

* Include your photograph here.

# Part B

The MIPS design presented in MIPS\_System only implements a limited number of the MIPS instructions. For the R-Type instructions ADD, ADDU, SUB, SUBU, AND, OR and SLT are implemented. Your task is to modify the MIPS design so that it implements the additional instructions shown in Table 1 whilst still ensuring the existing instructions work correctly. Once you have modified your design you need to write programs to demonstrate that your hardware correctly implements the instructions. Your results should include print outs of the SignalTap logic analyser showing your program operating. Annotate the print out to explain what is happening. You should submit an electronic copy of your design and assembly language programs onto CANVAS. Your written report should explain what modifications you have made to the Verilog code and include the Verilog code you have developed. There is no need to include the Verilog code for the modules you haven’t modified. You should also include ASM/ASMD charts for your modified code. For your report on instruction 3 you should include a block diagram showing the extra data pathways you have added.

1. NOR

* As depicted in the figure below, three instructions are carried out to validate the NOR instruction in MARS. This process results in ~(0xFFFF000 | 0x00000000) yielding 0x0000FFFF.

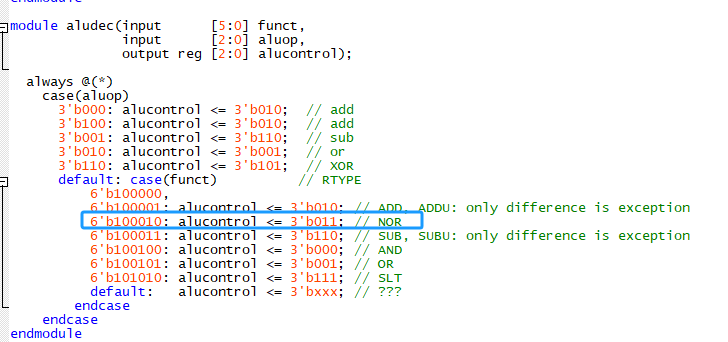


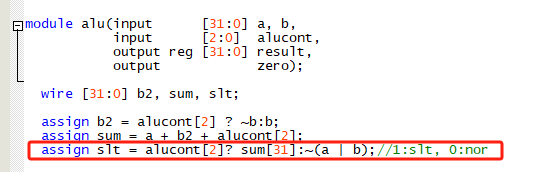


The validation process is also captured in the SignalTap logic analyser, shown in figure below.

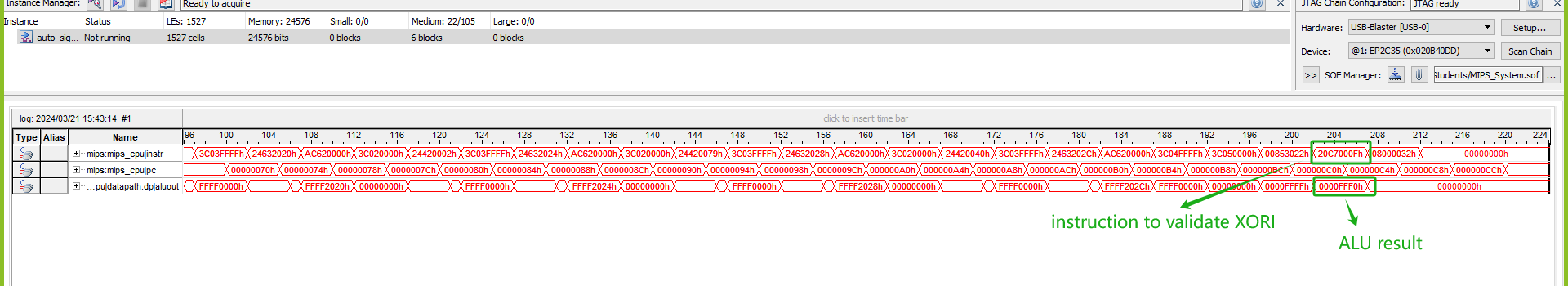


* Below are the modifications made in the Verilog code to achieve the NOR function.





1. XORI

The instruction chosen for verifying XORI is 0x20C7000F, representing "xori $7, $6, 0x000F". Following the execution of the NOR instruction, the content stored in register 6 becomes 0xFFFF. Consequently, the result of 0xFFFF ^ 0x000F is 0xFFF0.

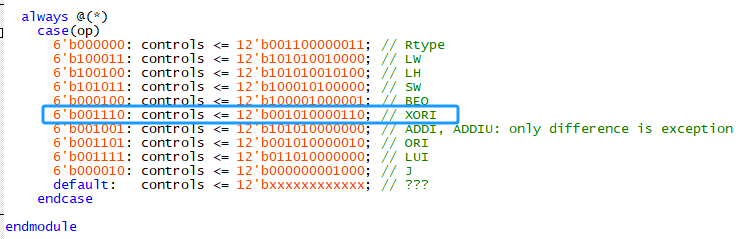
However, in MARS, the machine code for "xori $7, $6, 0x000F" is 0x38C7000F, as can be seen in the figure below. This mismatch arose because I mistakenly selected the wrong opcode for XORI, which should have been 6'b001110 instead of 6'b001000.

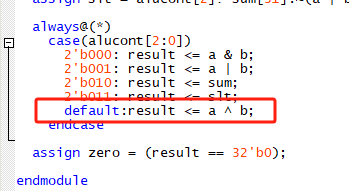


After rectifying the error in the opcode selection, the waveform depicted below demonstrates the accurate execution of "xori $7, $6, 0x000F".



Below is the revised code, where the signal **control** and **alucont** have been extended by one bit to accommodate the XORI instruction.





1. LH

Instructions to verify LH command are listed below:

1. lui $8, 0xEE

0x00EE0000 stored in register 8

1. lui $9, 0x0000

0x00000000 stored in register 9

1. addiu $9, $9, 0x40

0x00000040 stored in register 9

1. sw $8, 0x0000($9)

0x00EE0000 stored in memory[0x40]

1. lui $11,0x0000

0x00000000 stored in register 11

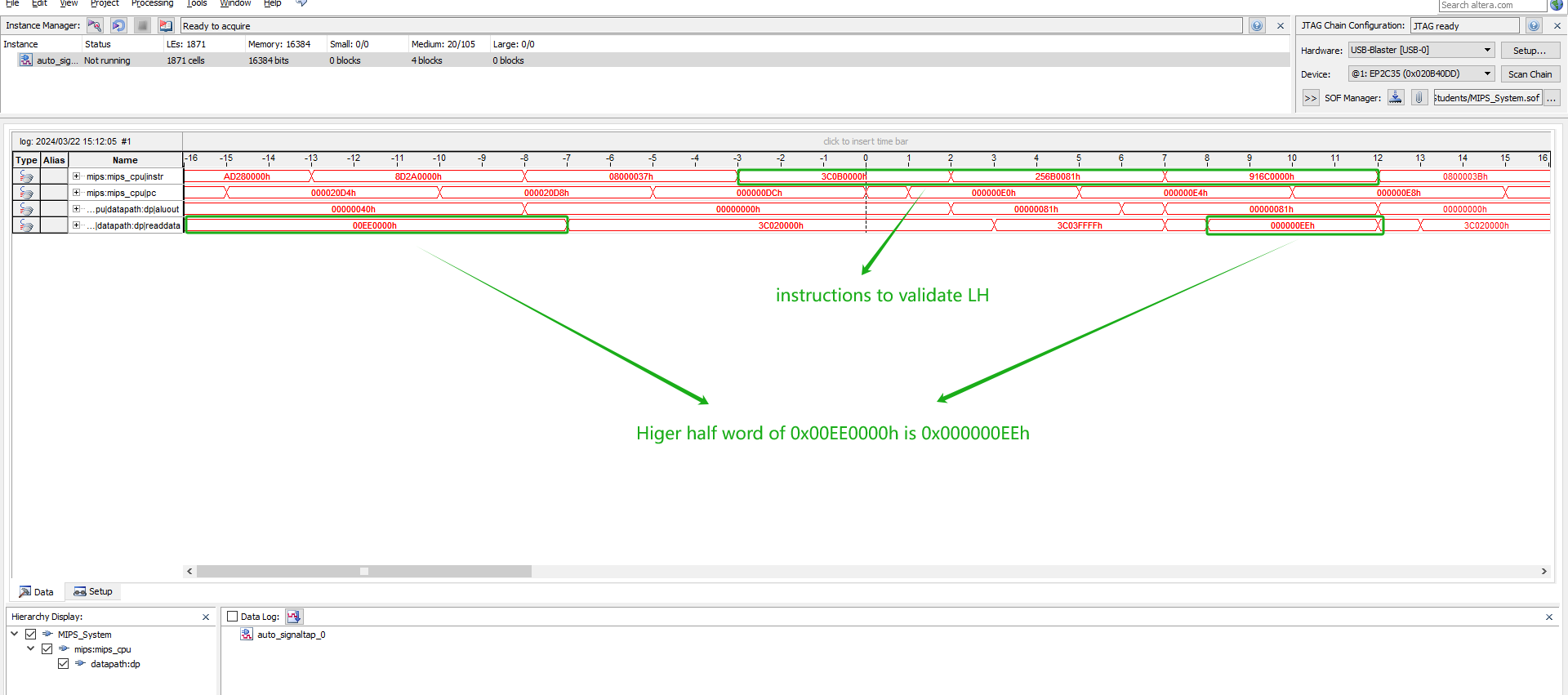
1. addiu $11, $11, 0x81

0x00000081 stored in register 11

1. lh $12,0x0000($11)

To load the higher half word from memory address 0x40, the last bit of 0x81, which is 1, indicates this operation. Consequently, 0x81 is right-shifted by 1 bit to obtain the memory address 0x40. Ultimately, register 12 will contain the value 0x0000EE00.

The part of validation process can be seen in the figure below:



Once more, the opcode for LH as depicted in the figure above is 0x916C0000, which deviates from the one in MARS (0x856C0000). By rectifying the opcode, the LH instruction can continue to function accurately.



And the modified code are depicted in the figure below.

