Assembly Language and Computer Architecture Lab

Week 2

Assignment 1: Assigning a 16-bit number to a register

Source code: **addi $s0, $zero, 0x2110003d** is to calculate sum of 2 values **0x2110003d** and **0**, then store the final value to register **$s0**.

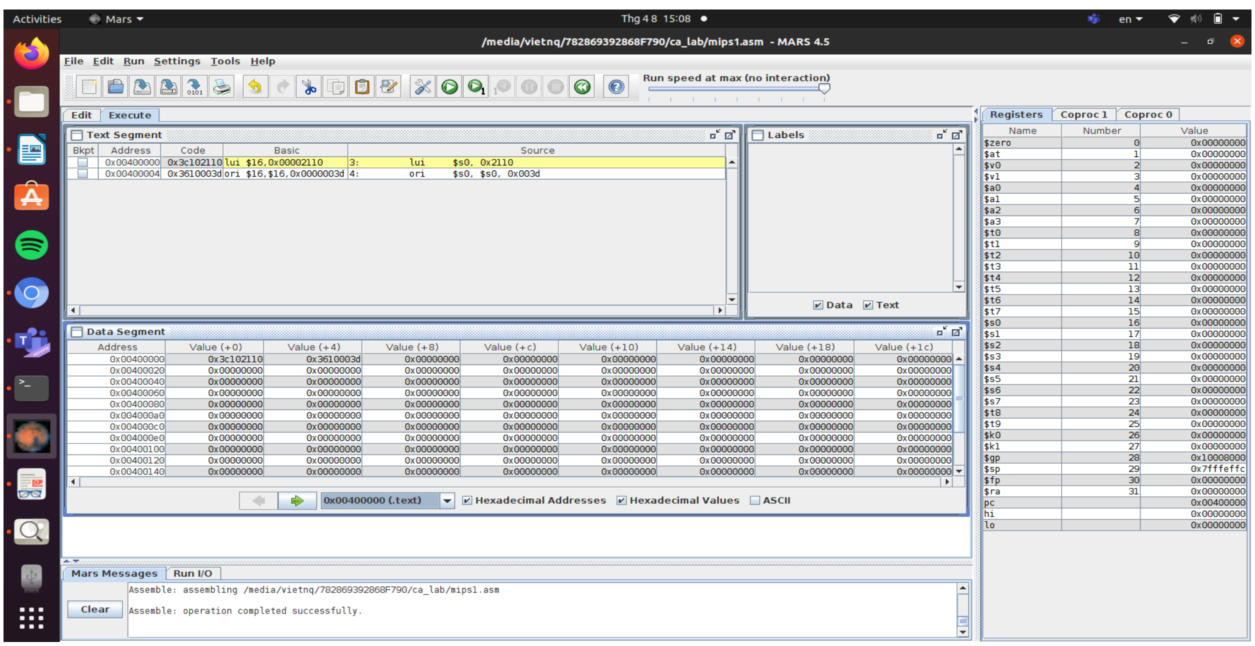
MIPS implemented 3 instructions:

- Set high-order 16 bits of **$at** to **0x2110** and low-order 16 bits to 0

- Use bitwise OR immediate to set **$at** to bitwise OR of itself and **0x003d** (value of **0x2110003d** is now stored in **$at**).

- Add **$zero** to **$at**, then store final result to **$s0**.

Assignment 2: Assigning a 32-bit number to a register



The location (address) of each instruction as we can see in the first row of the Data Segment table, they are at the first 2 cells:

lui $s0, 0x2110 => 0x00400000

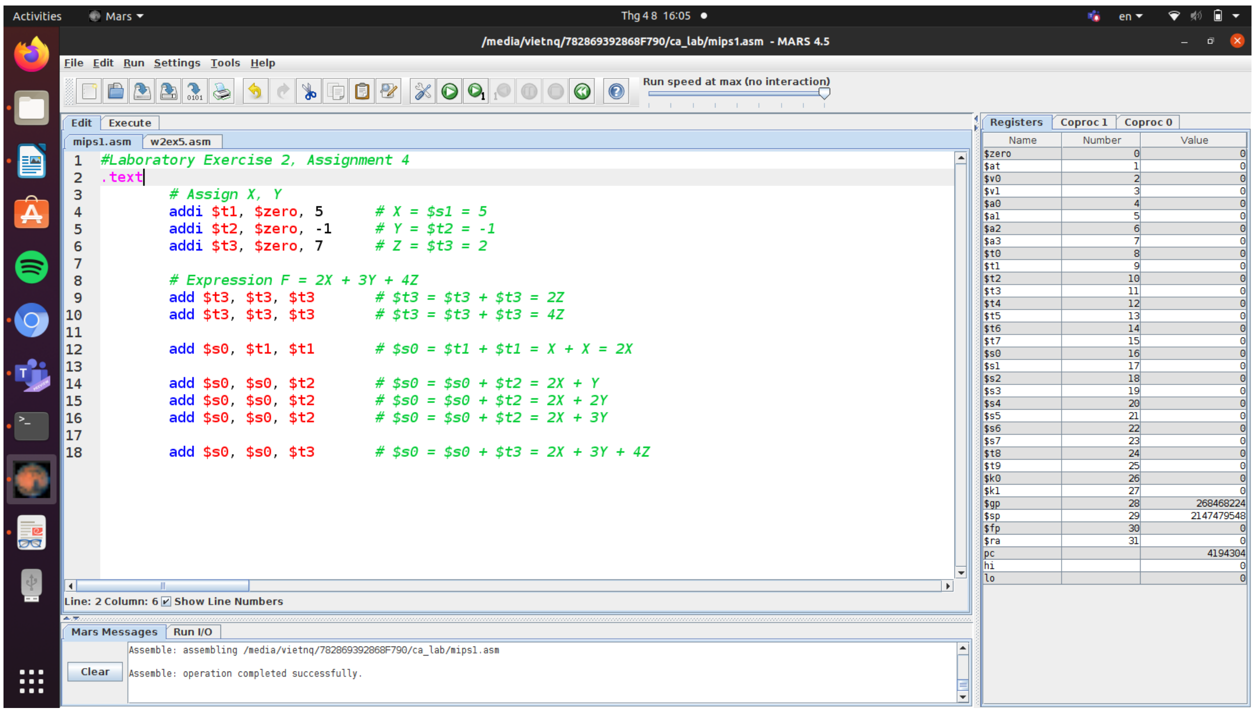
ori $s0, $s0, 0x003d => 0x00400004

Assignment 3: *li* psuedo instruction

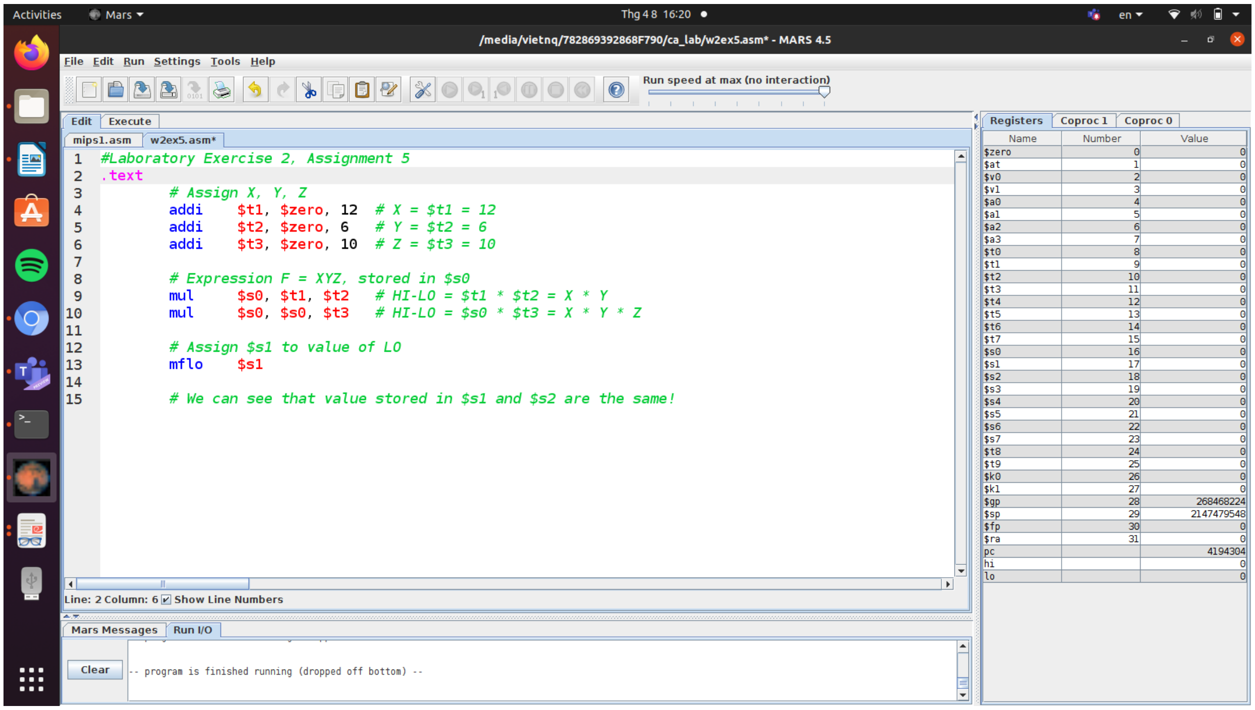
The value to be loaded by **li** is **0x2110003d** ***(in 32-bit range)***, while the operand is limited in range of 16 bits for the I-type instruction, so MIPS implemented 2 basic instructions for each half of the value instead of 1 instruction compared to the smaller value (**0x2**).

The ways these 2 basic instructions implemented are similar to the first 2 instruction in the case of ***Assignment 1***.

Assignment 4: Calculating simple math expression 2X+Y



Assignment 5: Multiplication



Assignment 6: Declaring and accessing variables

- The **la** instruction is implemented by 2 basic instructions **lui** and **ori**. For example, the machine code of these 2 instructions for variable X are: **0x3c011001** and **0x34380000**, where the last 4 numbers are 2 parts of the address of variable X (**10010000**).

- **lw**: load value of a variable in memory by its address, then store in the register.

- **sw**: assign a value for a variable in memory.