Student name: Nguyen Thi Minh Chau

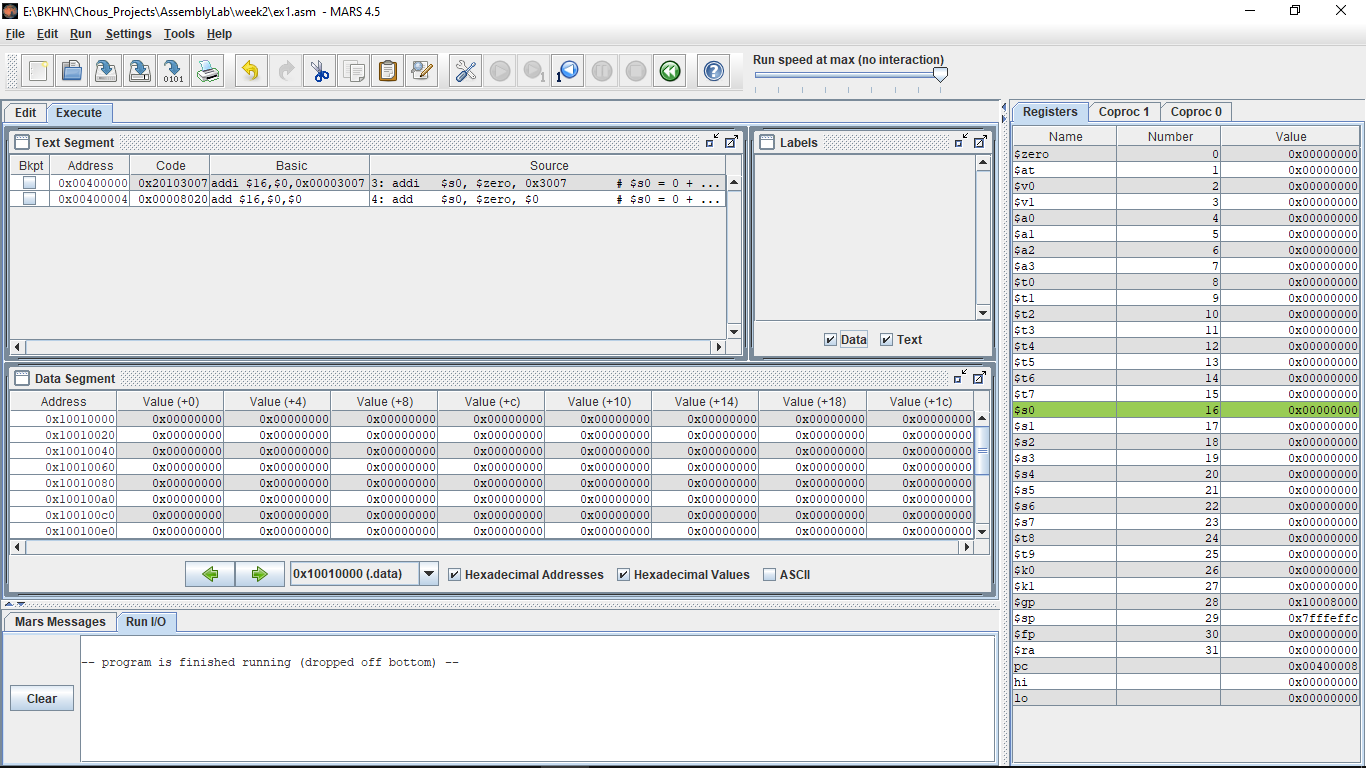
Student ID: 20184238

Class: ICT 01 – K63

**ASSEMBLY LAB 02 REPORT**

**Exercise 1:**

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | $s0 value changes from 0x00000000 to 0x00003007 (according to the operation of adding zero value in register $zero with an immediate value 0x00003007 defined in the instruction and store in $s0)  $pc value changes from 0x00040000 to 0x00400004 (address of the next instruction to be executed, which is add $s0, $zero, $0 ) |
| 2 | $s0 value changes from 0x00003007 to 0x00000000 (according to the operation of adding value in register $0 (which is previously 0) and zero value in register $zero and store in $s0)  $pc value changes from 0x00400004 to 0x00400008 (address of the next instruction to be executed, which is now null) |



The above machine code has the structures as defined:

- addi:

+ I instruction with 4 fields: 6 bits op (addi), 5 bits rs ($zero), 5 bits rt ($s0), 16 bits immediate (0x00003007)

+ 6 bits op: 001000

+ 5 bits rs: 00000

+ 5 bits rt: 10000 = 16 (decimal)

+ 16 bits immediate: 0011000000000111

Final code: 00100000000100000011000000000111 = 0x20103007

- add: R instruction with 6 fields: 6 bits op (addi), 5 bits rs ($zero), 5 bits rt ($0), 5 bits rd ($s0), 5 bits shamt, 6 bits funct

+ 6 bits op: 000000

+ 5 bits rs: 00000

+ 5 bits rt: 00000

+ 5 bits rd: 10000 = 16 (decimal)

+ 5 bits shamt: 00000

+ 6 bits funct: 100000

Final code: 00000000000000001000000000100000 = 0x00008020

When changing the first instruction to addi $s0,$zero,0x2110003d, we see that the basic instruction of add changes from 1 (addi $16, $0, 0x00003007) to 3 (lui $1, 0x0002110, ori $1, $1, 0x0000003d and add $16, $0, $1).

This is because addi in this case is actually a pseudo-instruction. When performing addi on 0x3007 (a 16-bit immediate), normal addi will be selected. However, 0x2110003d this a 32-bit immediate, the instruction will be transformed into 3 smaller instructions

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | Set 16 higher bits of $1 to 16 higher bits of 0x2110003d, and 16 lower bits of $1 to 0 (this result in value of $1 0x21100000) |
| 2 | Set 16 lower bits of $1 to 16 lower bits of 0x2110003d (this result in value of $1 0x2110003d |
| 3 | Add with overflow: set value of $16 to sum of values in $0 and $1 (which is 0x00000000 and 0x2110003d). The final result obtained is 0x2110003d (32-bit value) |

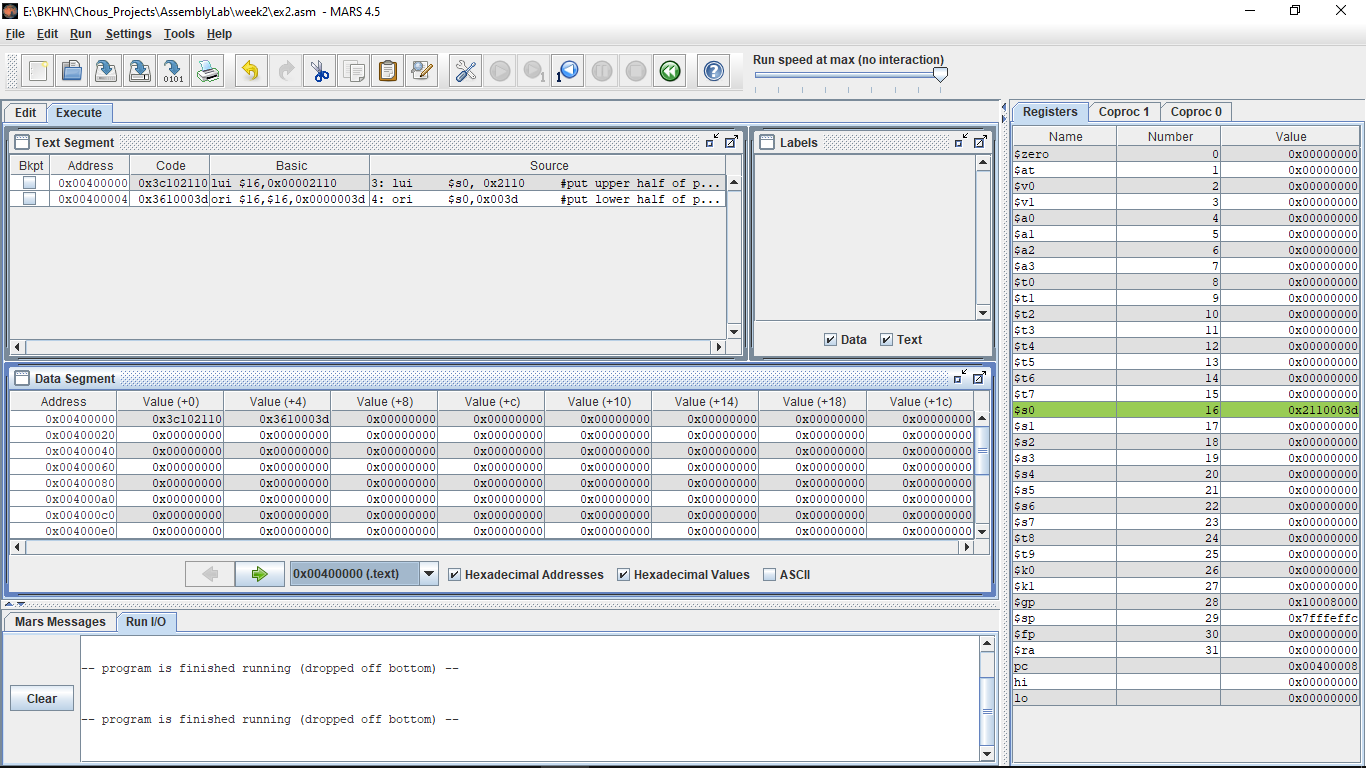
**Exercise 2:**

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | Set 16 higher bits of $s0 to 16 higher bits of 0x2110, and 16 lower bits of $s0 to 0 (this result in value of $s0 0x21100000)  $pc value changes from 0x00004000 to 0x00400004 (address of the next instruction to be executed, which is ori $s0, 0x003d) |
| 2 | Set 16 lower bits of $s0 to 16 lower bits of 0x0003d (this result in value of $s0 0x2110003d)  $pc value changes from 0x00040004 to 0x00400008 (address of the next instruction to be executed, which is now null) |

The first bytes of .text segments belongs to Code column in Text segment table, which indicate the instructions to be executed:

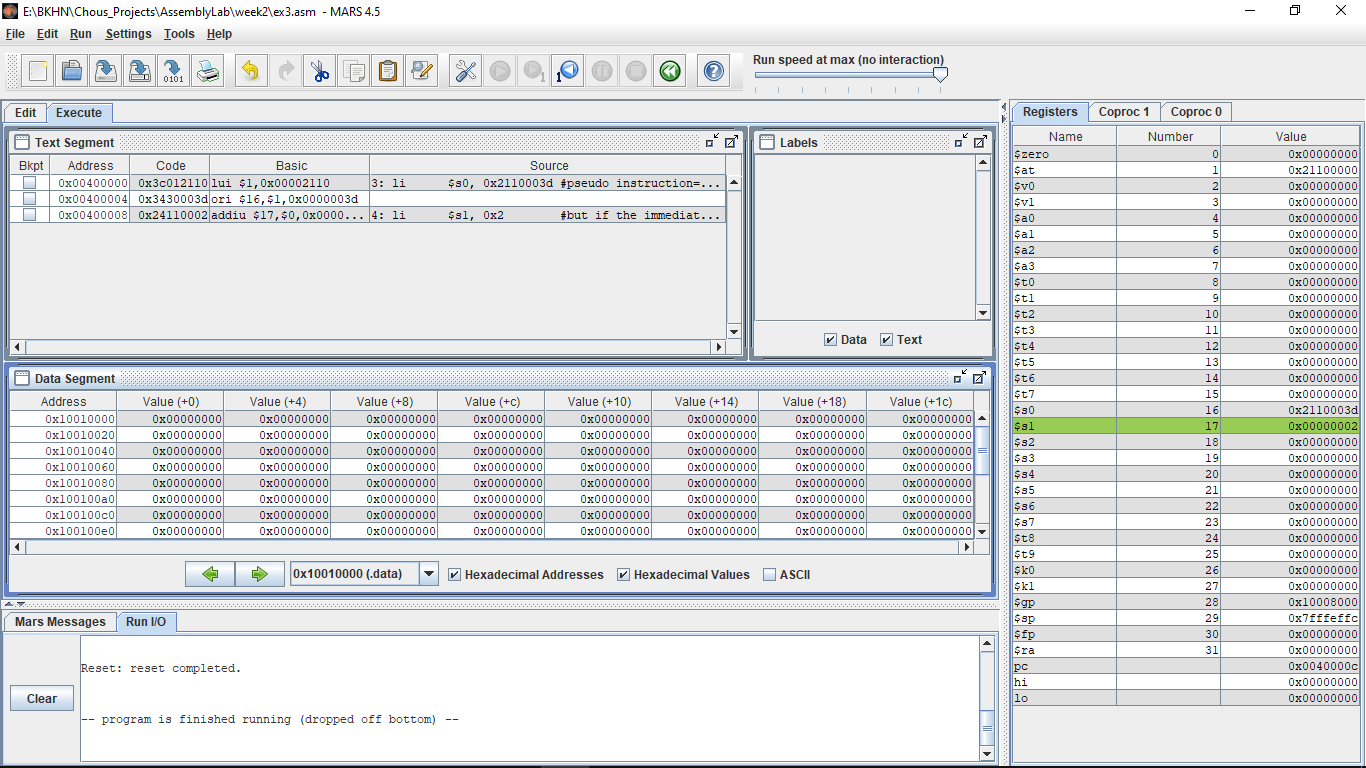
- At 0x00400000: Value 0x3c102110 belongs to instruction lui $s0, 0x2110

- At 0x00400004: Value 0x3610003d belongs to instruction ori $s0,0x003d



**Exercise 3:**

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | li is a pseudo-instruction. When performing on 32-bit immediate (which is now 0x2110003d), it will be decomposed into 2 smaller instructions:  - lui: set 16 higher bits of $1 to 16 higher bits of 0x2110003d, and 16 lower bits of $1 to 0 (this result in value of $1 0x21100000)  - ori: Set value of $16 to bitwise OR of $1 and 0x0000003d (bitwise or of 0x2110 and 0x003d is 0x2110003d) |
| 2 | However, if the immediate value is small and positive (16-bits immediate, 0x2 in this case), the instruction will become an addition with unsigned 16-bit immediate value for optimization  Here, addiu will assign sum of $0 and $0x2 to $s1 (which result in value of $s1 be 0x00000002) |



**Exercise 4:**

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | $t1 value changes from 0 to 0x00000005 (5 in signed decimal) |
| 2 | $t2 value changes from 0 to 0xffffffff (-1 in signed decimal) |
| 3 | $s0 value changes from 0 to 0x0000000a (10 = 5+5 in decimal) |
| 4 | $s0 value changes from 0x0000000a to 0x00000009 (9 = 10 + (-1) in decimal) |

Consider addi instructions: The machine code has the same structure as assembly code (in this case, type I)

- addi $t1, $zero, 5: I instruction with 4 fields: 6 bits op (addi), 5 bits rs ($zero), 5 bits rt ($t1), 16 bits immediate (0x00000005)

+ 6 bits op: 001000

+ 5 bits rs: 00000

+ 5 bits rt: 01001 (9 in decimal)

+ 16 bit immediate: 0000000000000101 (0x00000005 in hexadecimal)

Final code: 00100000000010010000000000000101 = 0x20090005

- addi $t2, $zero, -1: I instruction with 4 fields: 6 bits op (addi), 5 bits rs ($zero), 5 bits rt ($t2), 16 bits immediate (0xffffffff)

+ 6 bits op: 001000

+ 5 bits rs: 00000

+ 5 bits rt: 01010 (9 in decimal)

+ 16 bit immediate: 1111111111111111 (0xffffffff in hexadecimal)

Final code: 00100000000010101111111111111111 = 0x200affff

Consider add instructions: The machine code has the same structure as assembly code (in this case, type R)

- add $s0, $s1, $t1: 6 bits op (addi), 5 bits rs ($s1), 5 bits rt ($t1), 5 bits rd ($s0), 5 bits shamt, 6 bits funct

+ 6 bits op: 0000000

+ 5 bits rs: 01001

+ 5 bits rt: 01001

+ 5 bits rd: 10000= 16 (decimal)

+ 5 bits shamt: 00000

+ 6 bits funct: 100000

Final code: 0000001001010011000000000100000 = 0x01298020

- add $s0, $s0, $t2: 6 bits op (addi), 5 bits rs ($s0), 5 bits rt ($t2), 5 bits rd ($s0), 5 bits shamt, 6 bits funct

+ 6 bits op: 0000000

+ 5 bits rs: 10000

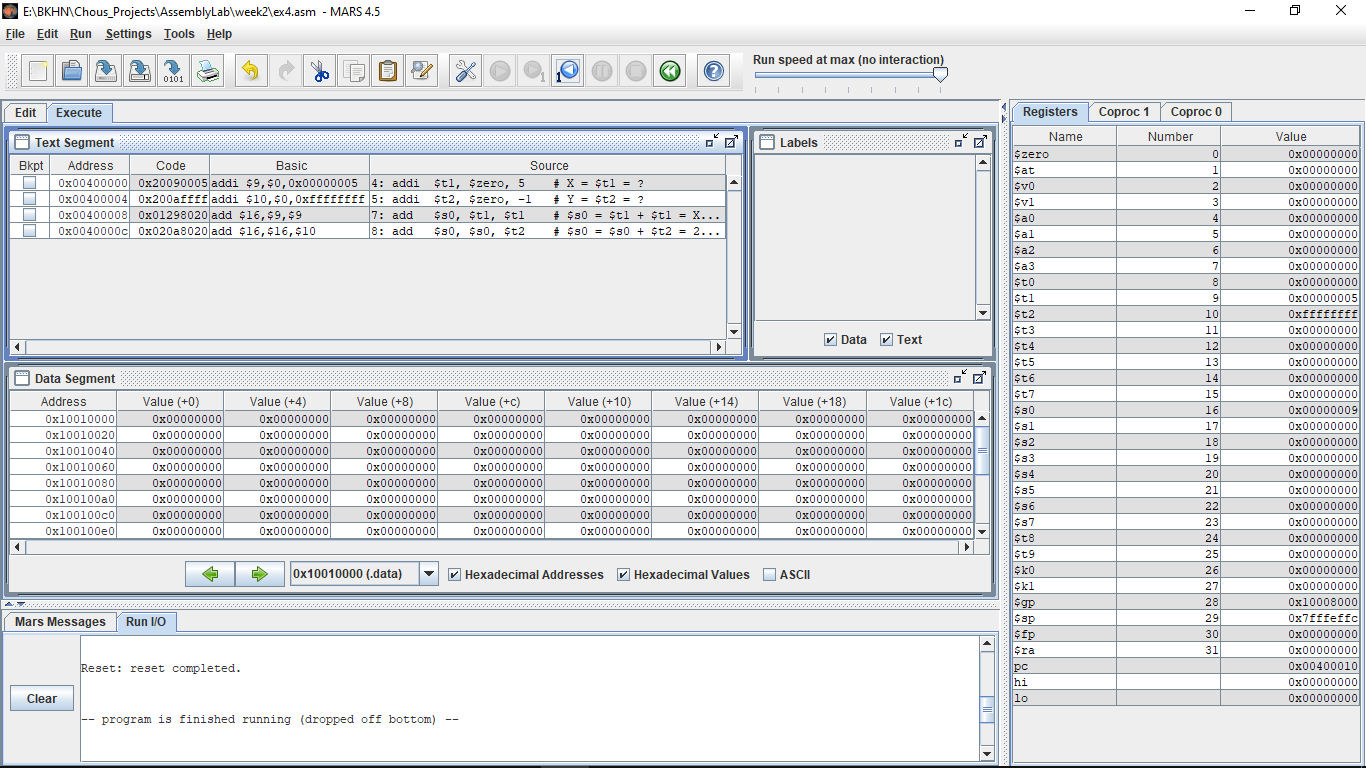
+ 5 bits rt: 01010

+ 5 bits rd: 10000= 16 (decimal)

+ 5 bits shamt: 00000

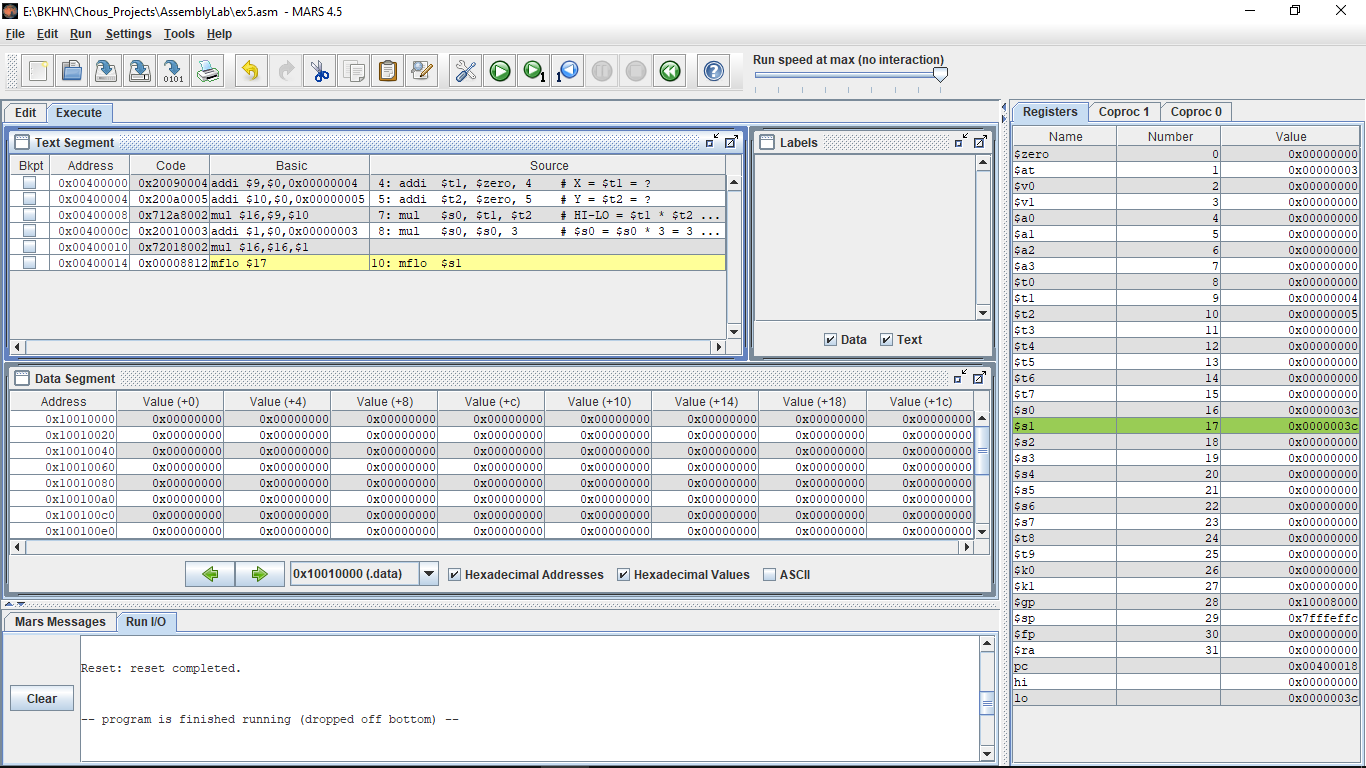
+ 6 bits funct: 100000

Final code: 00000010000010101000000000100000 = 0x020a8020



**Exercise 5:**

|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | $t1 value changes from 0 to 0x00000004 (4 in signed decimal) |
| 2 | $t2 value changes from 0 to 0x00000005 (5 in signed decimal) |
| 3 | Multiply value in $t1 and $t2 (4x5)  - $s0 value changes from 0 to 0x00000014 (20 in signed decimal)  - LO value changes from 0 to 0x00000014 (20 in signed decimal; since the operation produce a 16-bit result, the whole value is stored in LO register, HI register remains 0x0) |
| 4 | Triple the value in $s0  - Since this is the multiplication with an immediate, addi $1, $0, 0x00000003 is performed to move value 0x3 to register $1  - Then, the multiplication between $1 and $s0 is performed and the result is stored in $s0  + The value of $s0 changes from 0x00000014 to 0x0000003c (60 in signed decimal)  + The value of LO changes from 0x00000014 to 0x0000003c (60 in signed decimal, since the operation produce a 16-bit result, the whole value is stored in LO register, HI register remains 0x0) |
| 5 | mflo moves the value in LO register to $s1 register, which results in value of $s1 0x0000003c (60 in signed decimal) |



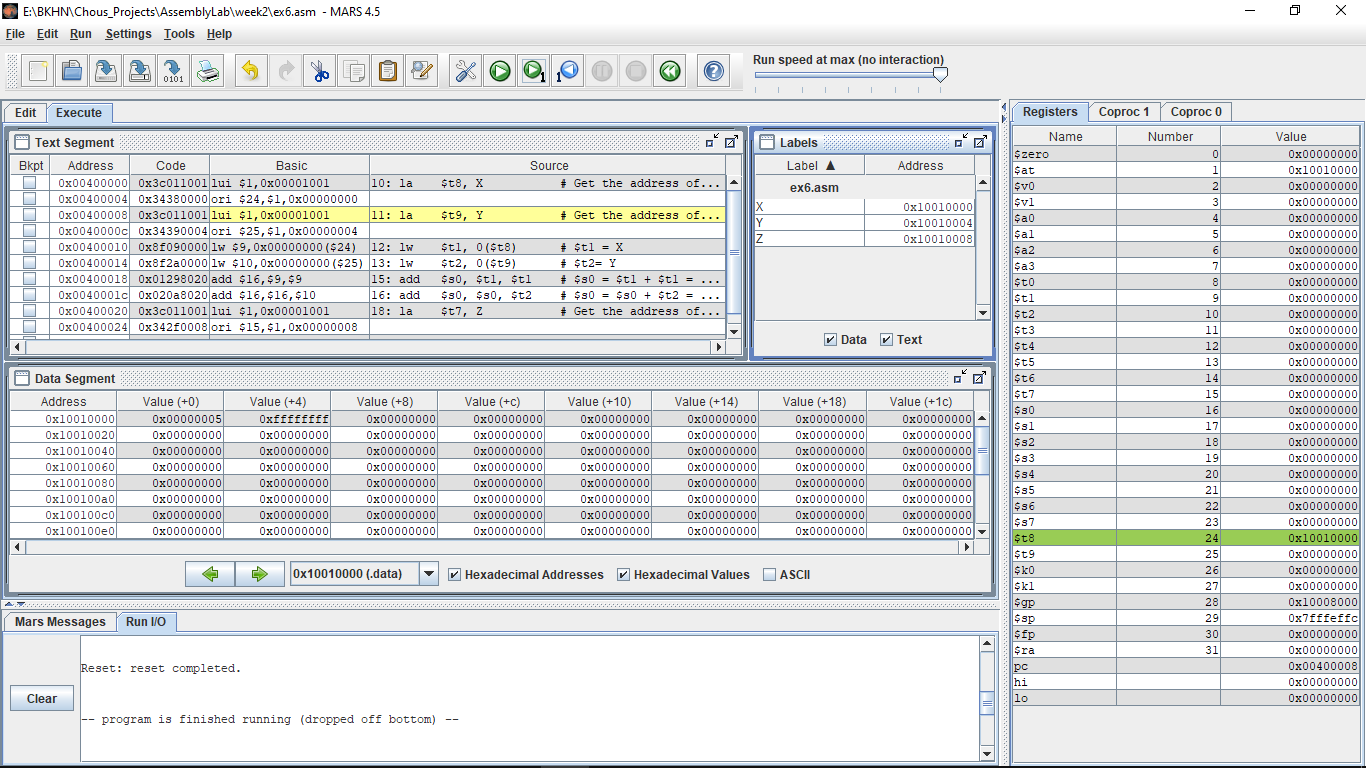
**Exercise 6:**

When complied:

- la instruction is destructed into 2 smaller instructions: lui and ori

- At the address of X (0x10010000): value obtained after la is 0x00000005 (5 in signed decimal, as defined in .data section)

- At the address of Y (0x10010004): value obtained after la is 0xffffffff (-1 in signed decimal, as defined in .data section)



|  |  |
| --- | --- |
| Step No. | Description & Explanation |
| 1 | - lui moves 16 higher order bits in address of X to 16 higher order bits of $1 register. Value of $1 is 0x10010000  - ori moves set value of $24 to result of bitwise OR value in $1 and 16 lower bits in address of X (0x10010000 and 0x00000000, which results in value of $24 0x10010000) |
| 2 | - lui moves 16 higher order bits in address of Y to 16 higher order bits of $1 register. Value of $1 is 0x10010000  - ori moves set value of $25 to result of bitwise OR value in $1 and 16 lower bits in address of Y (0x10010000 and 0x00000004, which results in value of $25 0x10010004) |
| 3 | lw store the value in the address stored at offset 0 of $24 to $t1 (in this case, address stored at offset 0 of $24 is 0x10010000, this address has the value of X, which is 5. Therefore, the value of $t1 is 0x00000005) |
| 4 | lw store the value in the address stored at offset 0 of $25 to $t2 (in this case, address stored at offset 0 of $25 is 0x10010004, this address has the value of Y, which is -1. Therefore, the value of $t2 is 0xffffffff) |
| 5 | add adds 2 times the value of $t1 and store it in $s0, which results in value of $s0 0x0000000a (10 = 5 + 5 in signed decimal) |
| 6 | add adds the value of $s0 and $t2 and store it in $s0, which results in value of $s0 0x00000009 (9 = 10 + (-1) in signed decimal) |
| 7 | - lui moves 16 higher order bits in address of Z to 16 higher order bits of $1 register. Value of $1 is 0x10010000  - ori moves set value of $15 to result of bitwise OR value in $1 and 16 lower bits in address of Z (0x10010000 and 0x00000008, which results in value of $24 0x10010008) |
| 8 | sw takes the values in $16 and store it at offset 0 of address stored in $15 (which is the address of Z), which makes value at address 0x10010008 becomes 0x00000009 |

- lb instruction: lb $t1 1($t2): set $t1 to sign-extended 8 bit value from effective memory byte address (at offset 1 of value in $t2)

- sb instruction: sb $t1 1($t2): store the low-order 8 bits of $t1 into effective memory byte address (at offset 1 of value in $t2)

