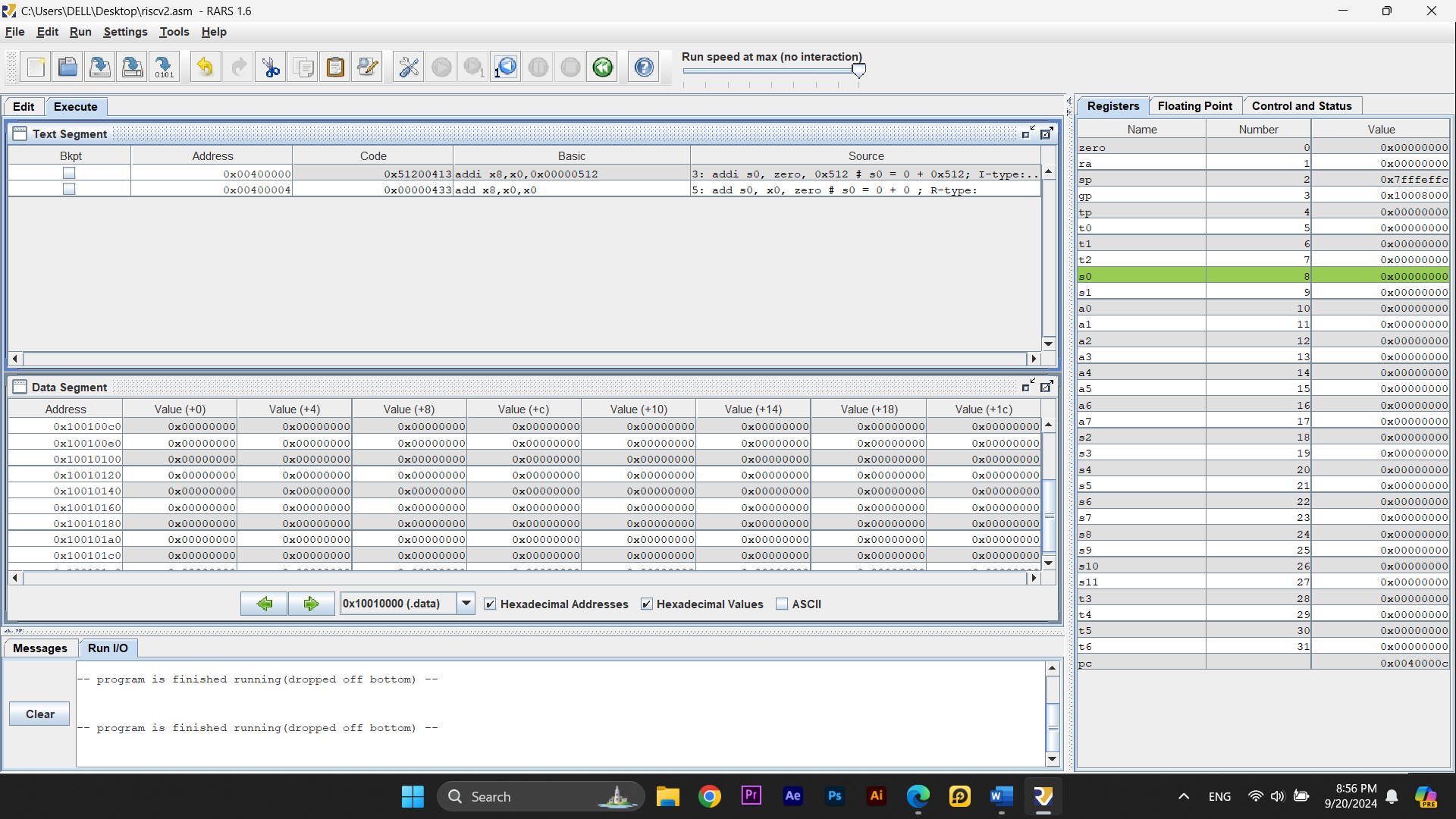
**LAB REPORT**

**IT3280E– 152049– Assembly Language and Computer Architecture Lab**

**Lab 02: Instruction Set, Basic Instructions, Compiler Directives**

# **Assignment 1: Assign 12-bit integer numbers/small integer:**



* Instruction <**lb>, <sb>:**

+ **lb** is used to load a byte from memory into a register.

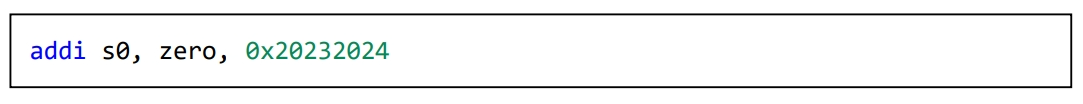
+ **sb** is used to store a byte from a register to memory.

* Instruction Formats:

+ **addi**: I-type format.

+ **add**: R-type format.

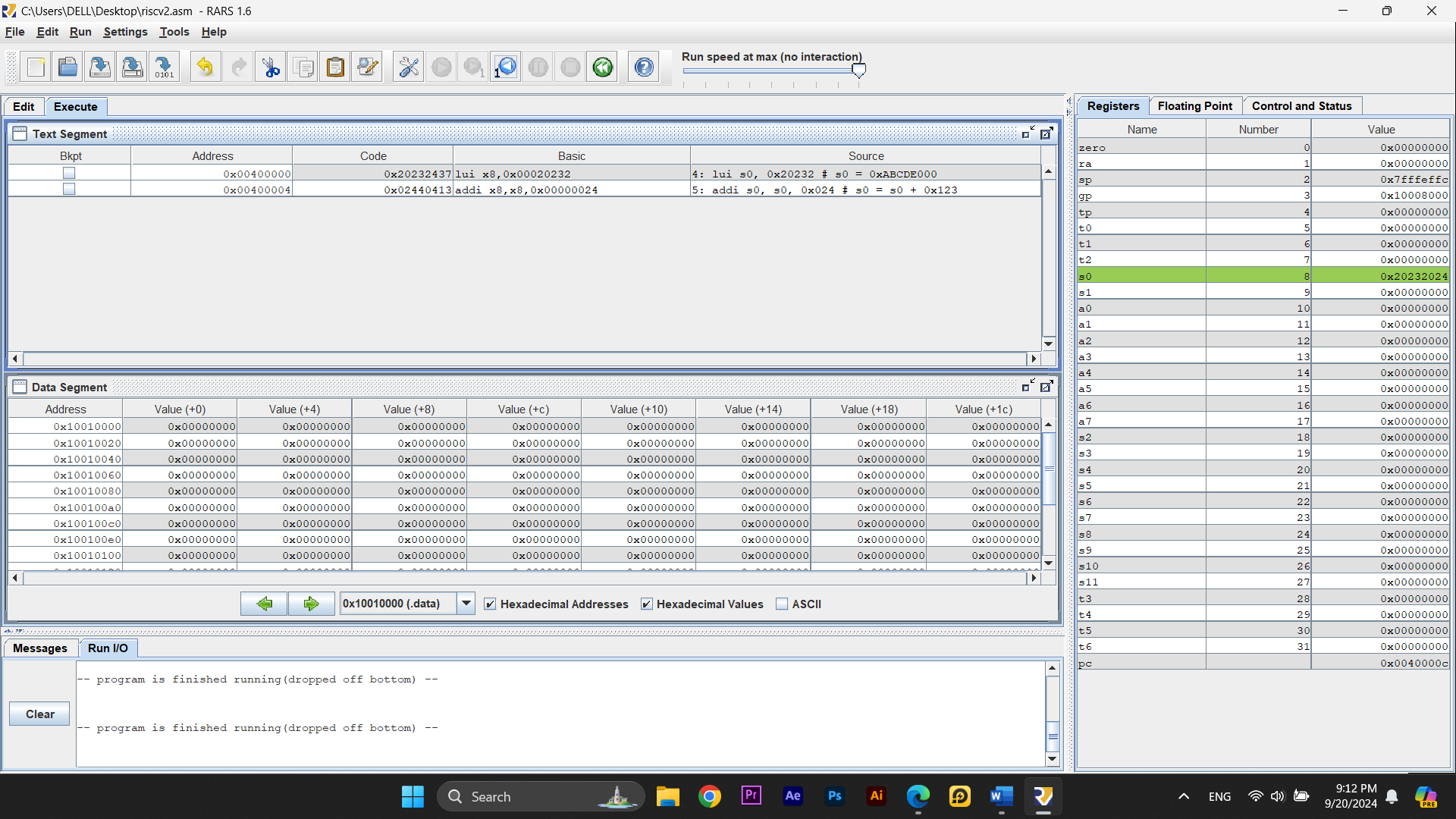
* When we modify the instruction addi as below:



will gives the result of:



**Assignment 2: Assign 32-bit integer:**



* Monitor Register Values:

+ ***lui s0, 0x20232***: This instruction loads the upper 20 bits of *0x20232* into the s0 register, making *s0* = *0x20232000*.

+ ***addi s0, s0, 0x024***: adds *0x24* to *s0*, resulting in *s0* = *0x20232024*.

* Instruction Formats:

+ **lui**: U-type instruction.

+ **addi**: I-type instruction.

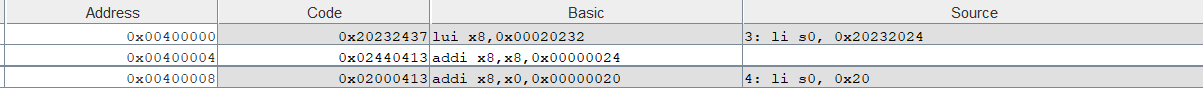
* Compare data in Data Segment and machines code in Text Segment:





* The data in the Data Segment matches the machine code from the Text Segment, confirming that the binary representations of the instructions are correctly loaded into memory.

# **Assignment 3: New assignment instructions:**



* Compare:

+ <Source> column: you get the exact same code as you typed.

+ <Basic> column: actual instruction that RARS translates into. (“li” is a pseudoinstruction)

* Explain:

+ ***li s0, 0x20232024****:*

* This large immediate value will be split into multiple instructions.
* Likely, it will be broken down into a **lui** instruction and an addi (Add Immediate) instruction:
  + **lui** *s0, 0x20232* to load the upper part of the value.
  + **addi** *s0, s0, 0x24* to add the lower part (24).
* These instructions together load the full *0x20232024* into *s0*.

+ ***li s0, 0x20:***

* Since *0x20* is a small value that fits within 12 bits, the pseudoinstruction li *s0*, *0x20* will be directly translated into a single addi instruction:

***addi s0, zero, 0x20***

* For small values, it gets translated to a single ***addi*** instruction. For large values, it gets translated into multiple intrusctions.

# **Assignment 4: Calculate the expression 2x + y = ?:**

A screenshot of a computer

Description automatically generated

* The value of *s0* after the final instruction is 9 => which is correct
* Machine codes:

*+ 0x00500313*: addi

*+ 0xfff00393*: addi

*+ 0x00630433:* add

*+ 0x00740433:* add

* Compile to machine code manually:

+ ***addi t1, zero, 5:***

imm[11:0] | rs1 | funct3 | rd | opcode

000000000101 | 00000 | 000 | 00110 | 0010011

* Machine code: *0x00500313*

***+ addi t2, zero, -1:***

imm[11:0] | rs1 | funct3 | rd | opcode

111111111111 | 00000 | 000 | 00111 | 0010011

* Machine code: *0xFFF00393*

***+ add s0, t1, t1:***

funct7 | rs2 | rs1 | funct3 | rd | opcode

0000000 | 00110 | 00110 | 000 | 01000 | 0110011

* Machine code: *0x00630333*

***+ add s0, s0, t2:***

funct7 | rs2 | rs1 | funct3 | rd | opcode

0000000 | 00111 | 01000 | 000 | 01000 | 0110011

* Machine code: *0x00740333*
* Same as those displayed in the <Text Segment> window.

# **Assignment 5: Multiplication:**

A screenshot of a computer

Description automatically generated

* *t1* hold the values of X = 4.
* *t2* hold the values of Y = 5.
* *s1* store the results of the multiplication (which is 20 or 0x014 in hex code).
* Division instructions:

***div s2, t1, t2***

***A screenshot of a computer

Description automatically generated***

* *s2* will have a value of 0 because the integer division of 4/5 results in 0.

# **Assignment 6: Declare and access variables:**

A screenshot of a computer

Description automatically generated

* Compare values:

+ X = 5, Y = -1, Z = 0 => match their intial values in the code

* Instruction **la**:

+ ***la*** loads the address of a label (X, Y, or Z) into a register.

+ It’s a pseudo-instruction, so it gets converted into two machine instructions: a ***lui*** and an ***addi***.

* Monitor **lw** and **sw**:

+ ***lw*** is used to load the values of X and Y from memory into registers *t1* and *t2*.

+ ***sw*** stores the result of the calculation (Z) back into memory from register *s0*.

* Instructions **lb** and **sb**:

+ ***lb*** loads a single byte of data into a register.

+ ***sb*** stores a single byte from a register into memory.

# **Assignment 7: Declare variables or instructions at specified addresses:**

**A screenshot of a computer

Description automatically generated**