COMPUTER SCIENCE AND DATA ANALYTICS

Course: **Computer System Architecture**

Term Project

Intel 4004 Calculator

Student: **Shikhaliyev Anar**

Instructor: Dr. Sencer Yeralan

Baku 2022

Table of Contents

1. Abstract

Up until now, we have covered so many topics, did bunch of both practical and research based assignments in this course. As a term project, I chose to go little deeper and build a simple calculator in Intel 4004 architecture. We already did simple assignment with Intel 4004 using emulator and the same emulator is used while building the calculator. Below, I will talk more about algorithmic realization, difficulties and future plans.

2. Introduction

In 1971, Intel Corporation introduced the 4-bit central processing unit (CPU) known as the Intel 4004. It was the first commercially available microprocessor and was sold for US$60. In Intel 4004, 4K 8-bit instruction words of program memory and 5120 bits of data storage RAM can be directly addressed by the CPU.

Intel 4004 designed primarily for use in small business systems like calculators, automated teller machines and cash machines. It had clock speed of 740 kHz and can execute 92.600 instructions per second. Moreover, CPU has 12-bit address space and 4-bit address bus. One of the interesting facts about intel 4004 is that it has separate memory for both data and program.

3. Hardware constraints

Despite knowing that, the creation of this cpu was not so much ago, technology improved rapidly and compared to today's technology intel 4004 is so primitive and weak. While executing the task, I have come across with several difficulties both lack of instructions and design point of view.

To begin with, it is a 4-bit processor, which means it is primarily intended for use with 4-bit binary values. Because the chip only has 16 pins, there isn't much room to begin working with. The CPU is relatively primitive, with just a few ALU operations in its instruction set which is 45 in total.

- operation on 4-bit operands only

And only 2 logic operations:

- complement
- Rotate (left or right)

Moreover, program can not be long, because program counter is 12-bit wide and both conditional or indirect jumps are short and can address only the 8-bit range.

Additionally, since there is no instruction which changes register values, one had to load the value to accumulator, do the calculation and exchange the value with register if needed.

4. Algorithmic Realization

First of all, since I am working on a emulator which also contains some bugs in itself, I have set some standards like initial states, outputs, and basic rules which is applied globally. Overall, I have written 4 algorithm in total for each operation(addition, subtraction, multiplication, division). However, they have common features.

In order to start calculation we need some inputs and since we are in emulator we cannot physically enter the numbers. So, I have decided to store inputs in 0p(R0R1) and 1p(R2R3) registers by fetching immediately in the beginning of the program and use it later on calculation. Additionally, result will be stored in the 4-bit Status nibbles in chip 0, bank 0, Register 0.

The multiplication and division is executed with the help of addition and subtraction algorithms respectively.

4.1. Addition

The logic for the addition is pretty simple and it is executed the natural way that as initially adding the first digits of numbers and save the carry(if there is) for the addition of the next digits. After every addition, we need to normalize the number in the accumulator to be sure that it's in the valid form before writing to the output or doing more calculations.

The project flow is like following:

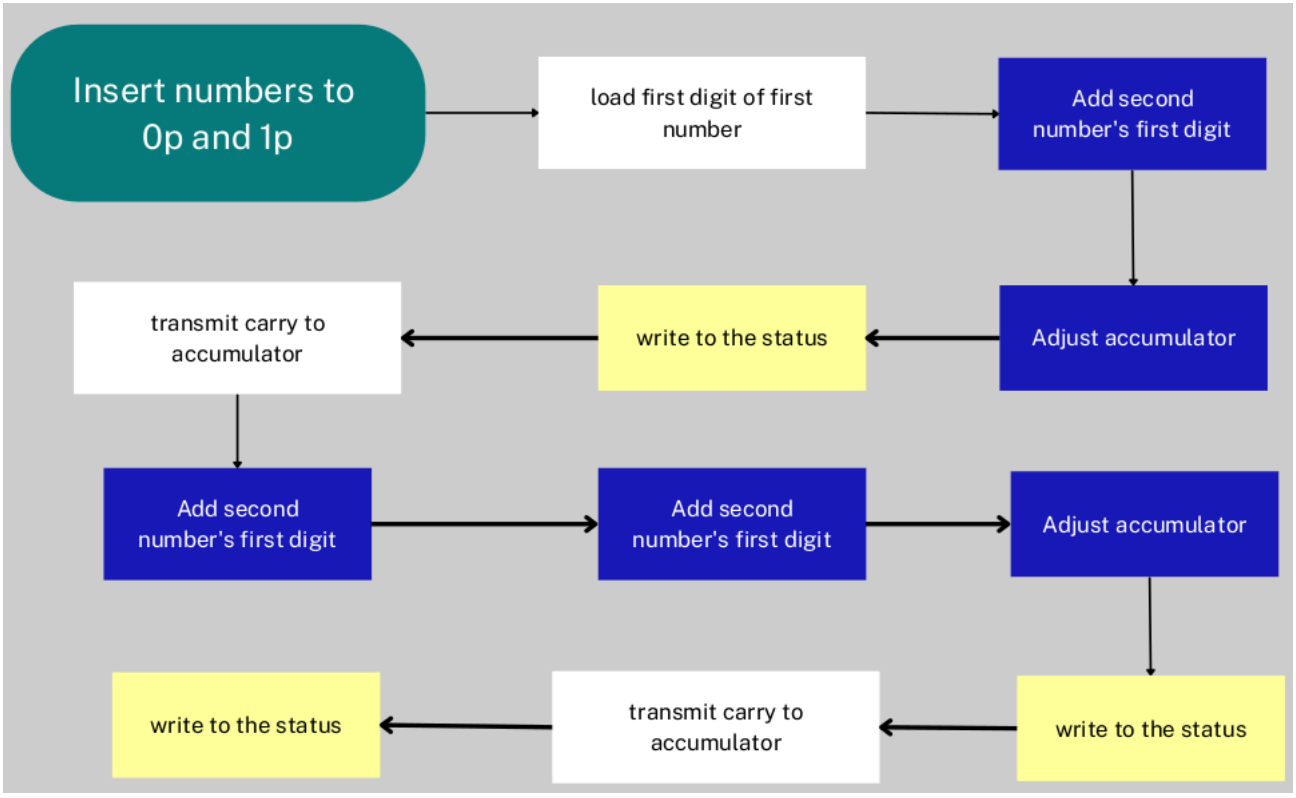

Figure 1.

After assembling the source code and loading the object code into the emulator, result will be like:

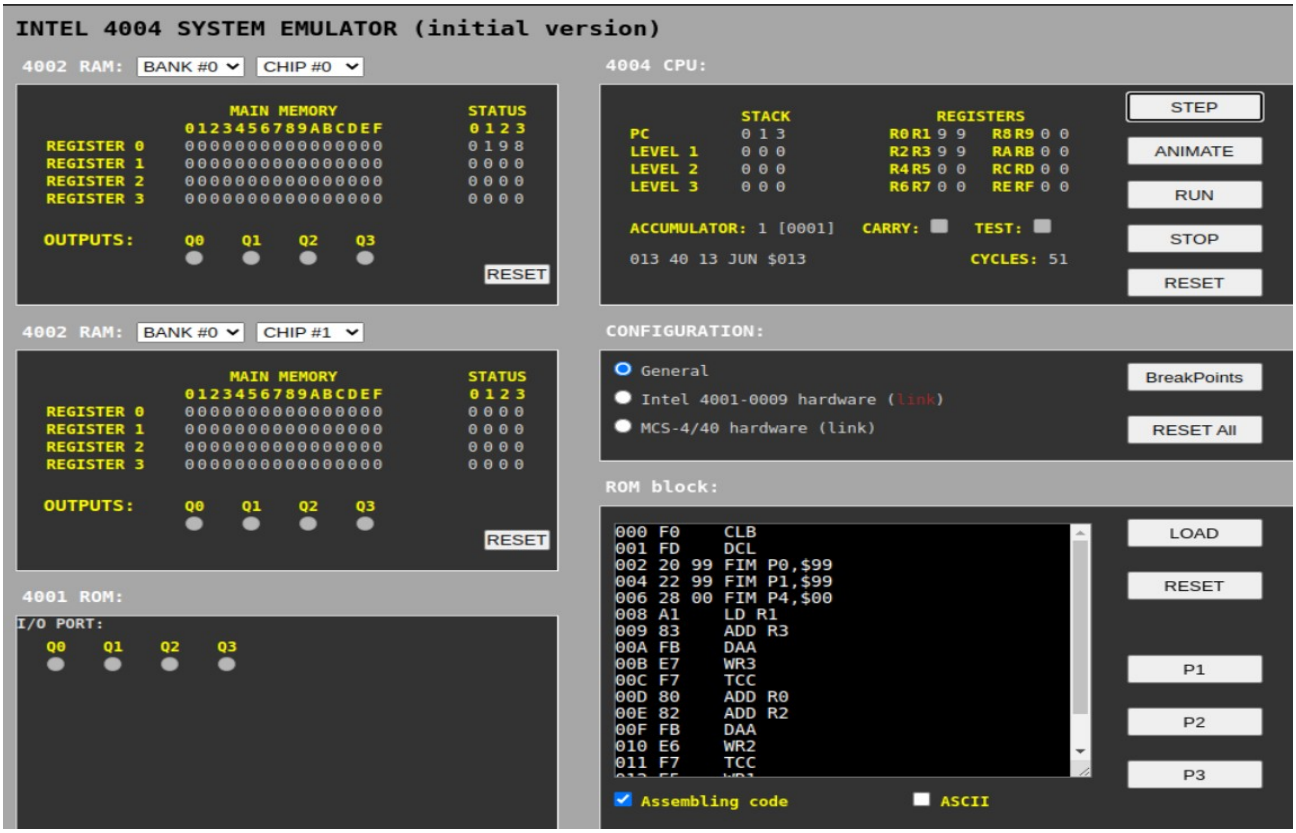

Figure 2. 99+99 = 198

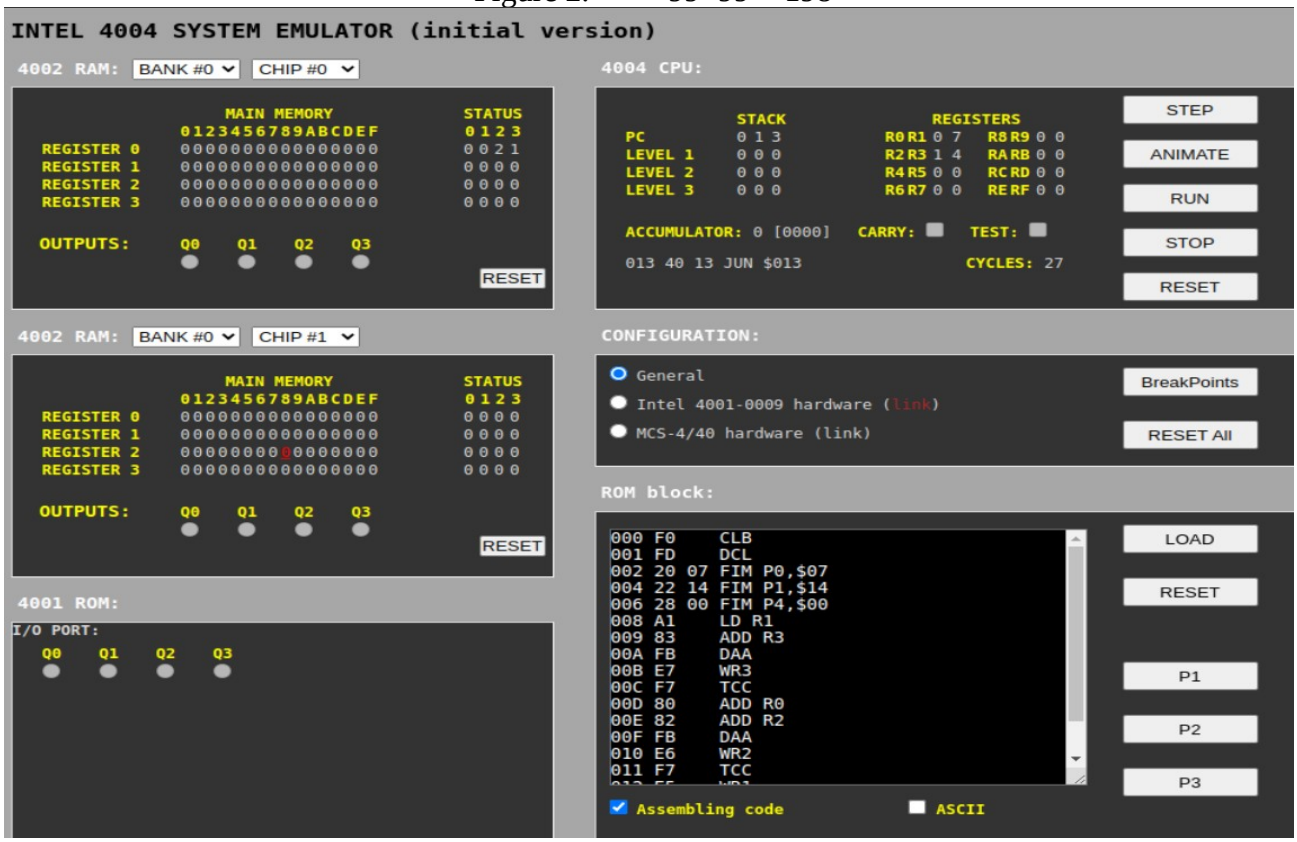

Figure 3. 7+14 =21

4.2. Subtraction

Subtraction is also done the standard way, we subtract first digits and even if it is not enough we borrow one from the decimal and decrease the decimal value.

If the result is going to be negative value, we change the values of the pairs (0p → 1p, 1p → 0p).

The project flow is like following:

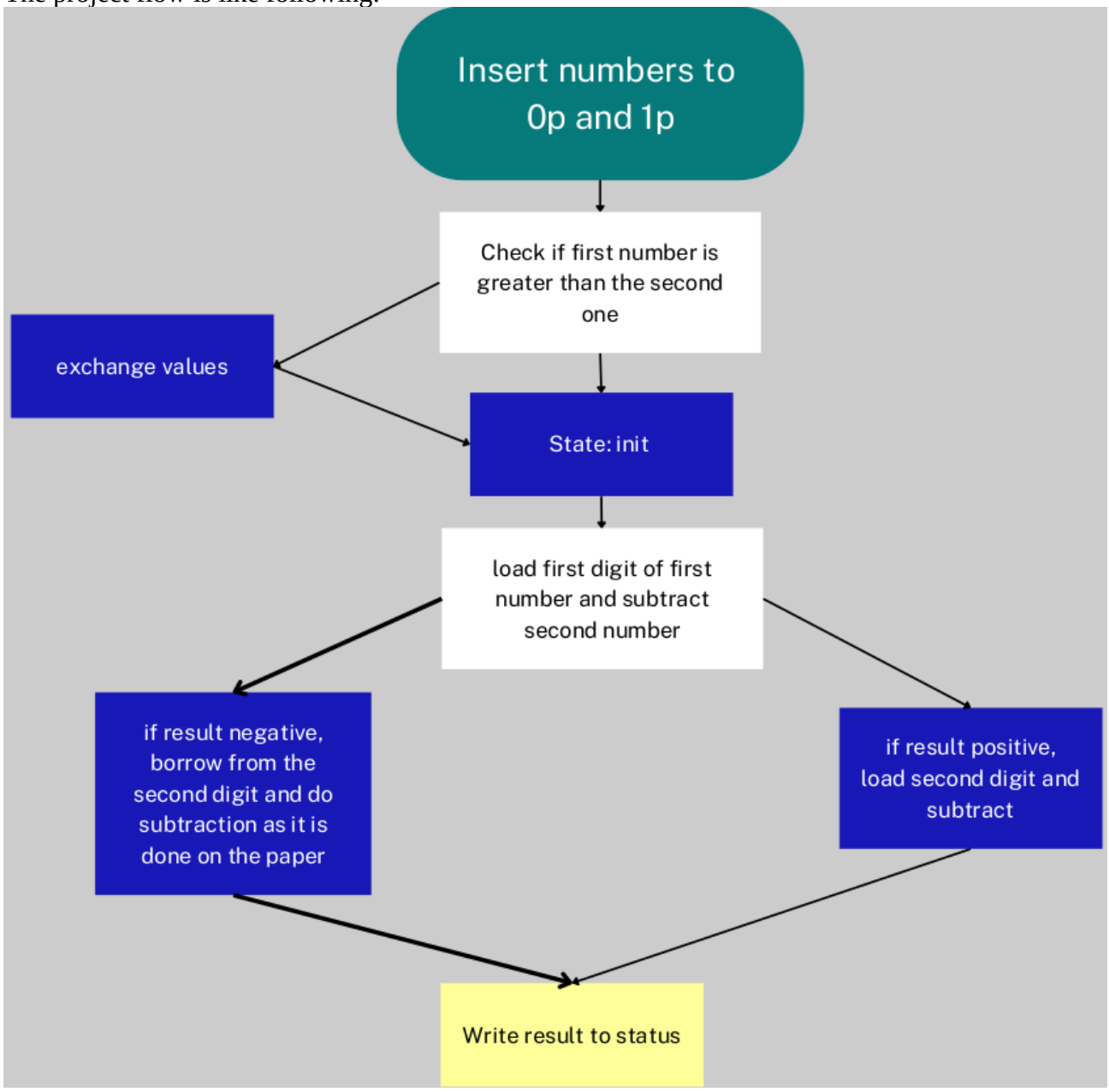

Figure 4.

After assembling the source code and loading the object code into the emulator, result will be like:

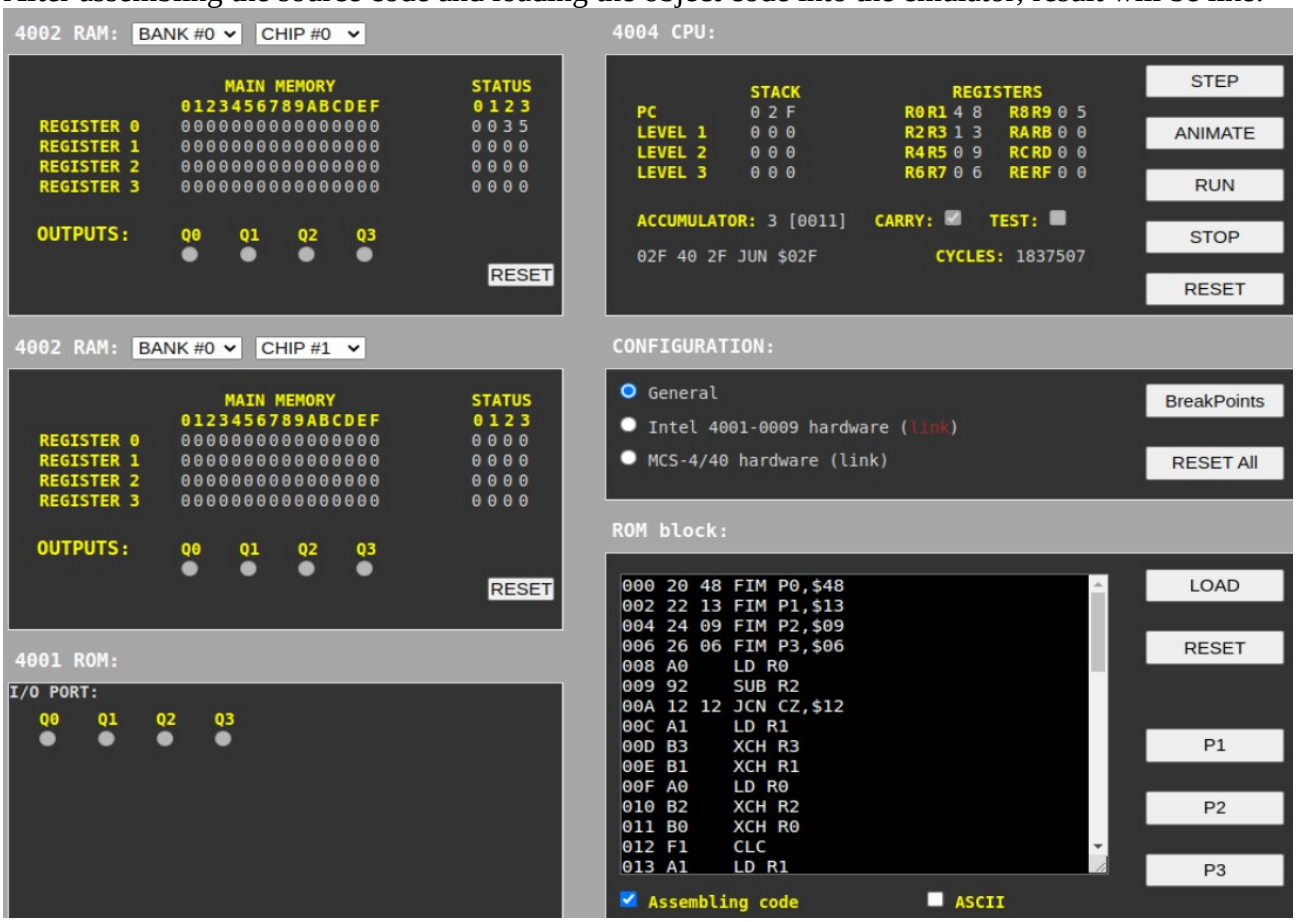

Figure 5. 48-13 = 35

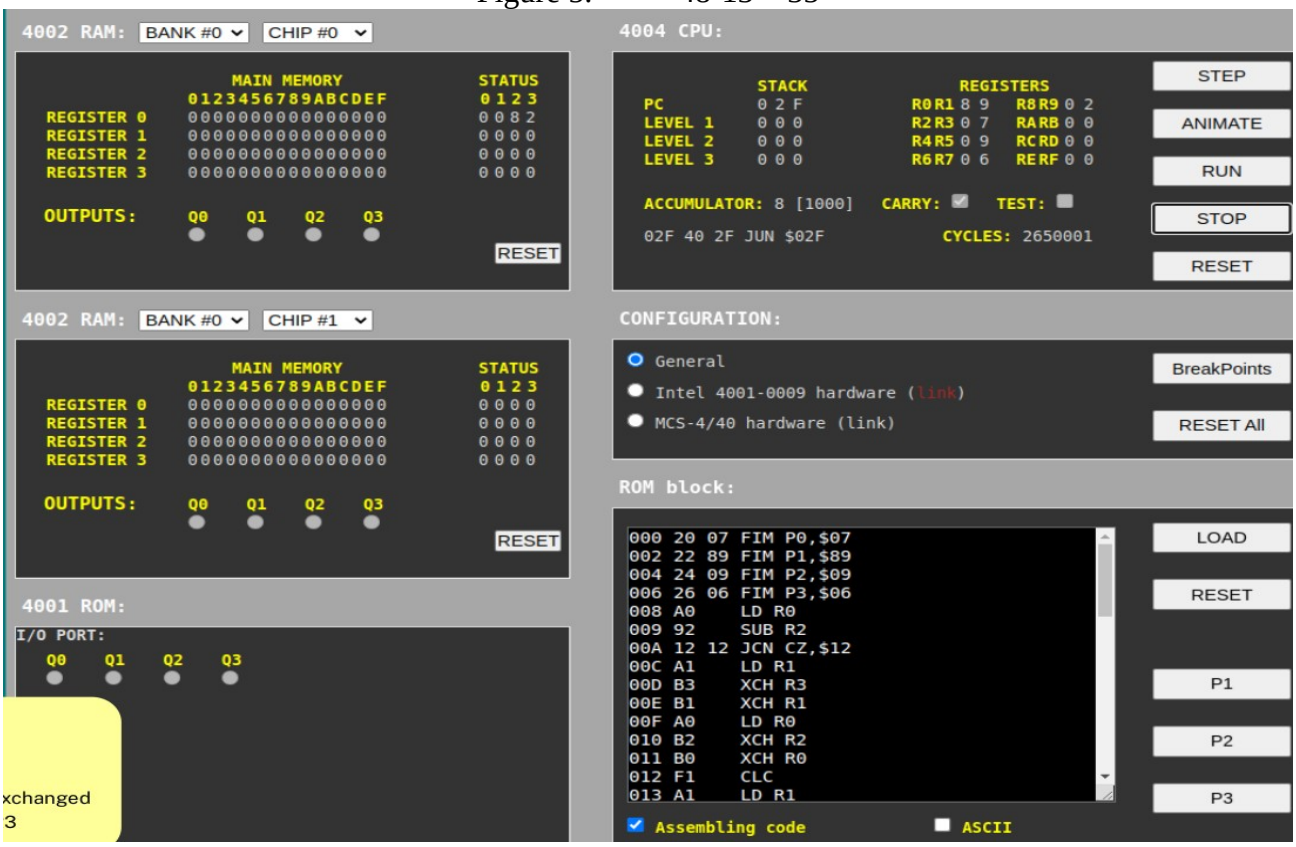

Figure 6. 89-7 = 82

4.3. Multiplication

For multiplication, we re-use the multiplication by addition method with implementing loops and using complement of the second input as a loop counter.

All the additional rules which is implemented in the addition is also valid for multiplication.

The project flow is like following:

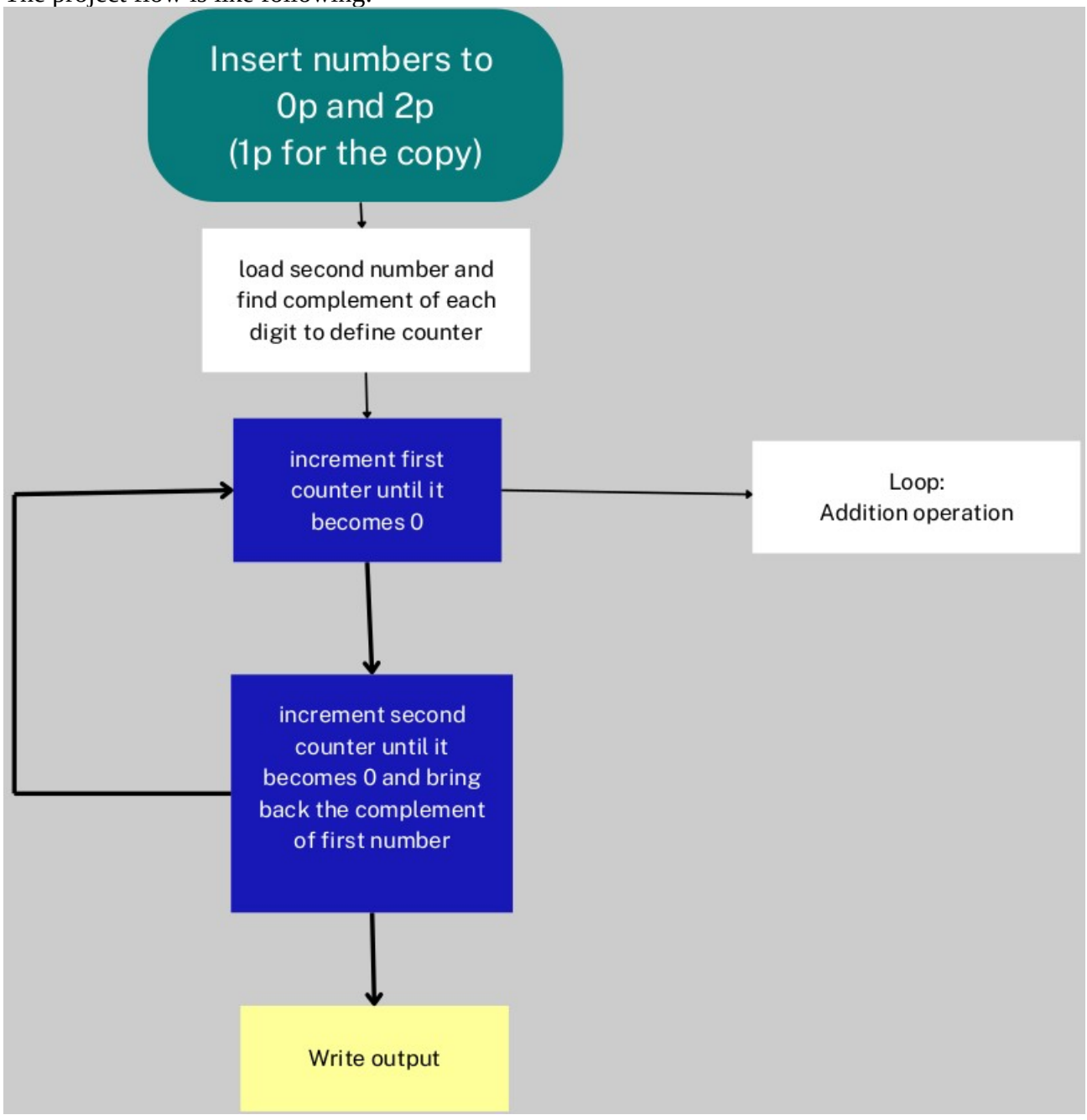

Figure 7.

After assembling the source code and loading the object code into the emulator, result will be like:

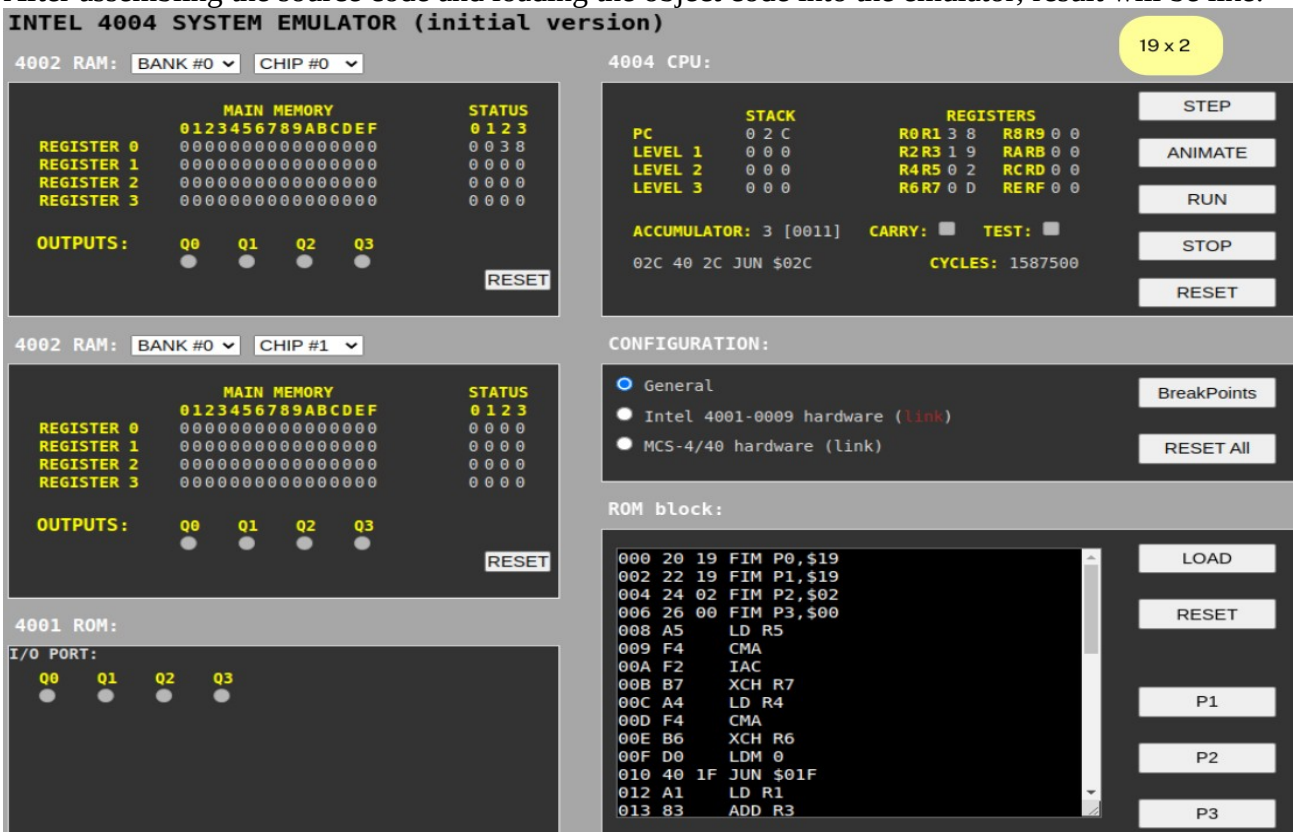

Figure 8. 19*2=38

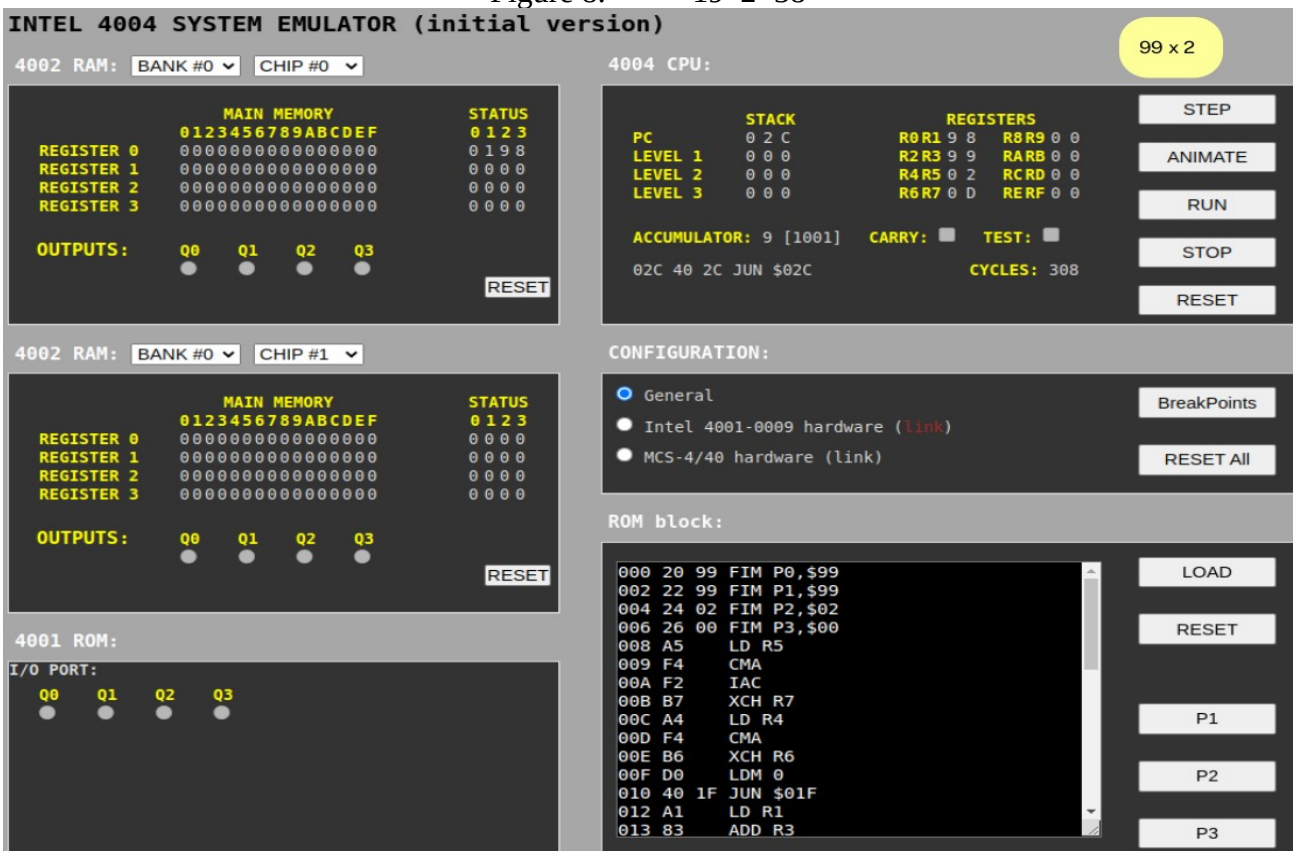

Figure 9. 99*2 =198

4.4. Division

Division is by far was the most sophisticated and longest one. We also re-used the division by subtraction method with implementing loops and using multiple conditional jumps. Since, it is integer division, we will have a remainder and quotient as a result which is separated with a letter C in status.

All the additional rules which is implemented in the subtraction is also valid for division.

The project flow is like following:

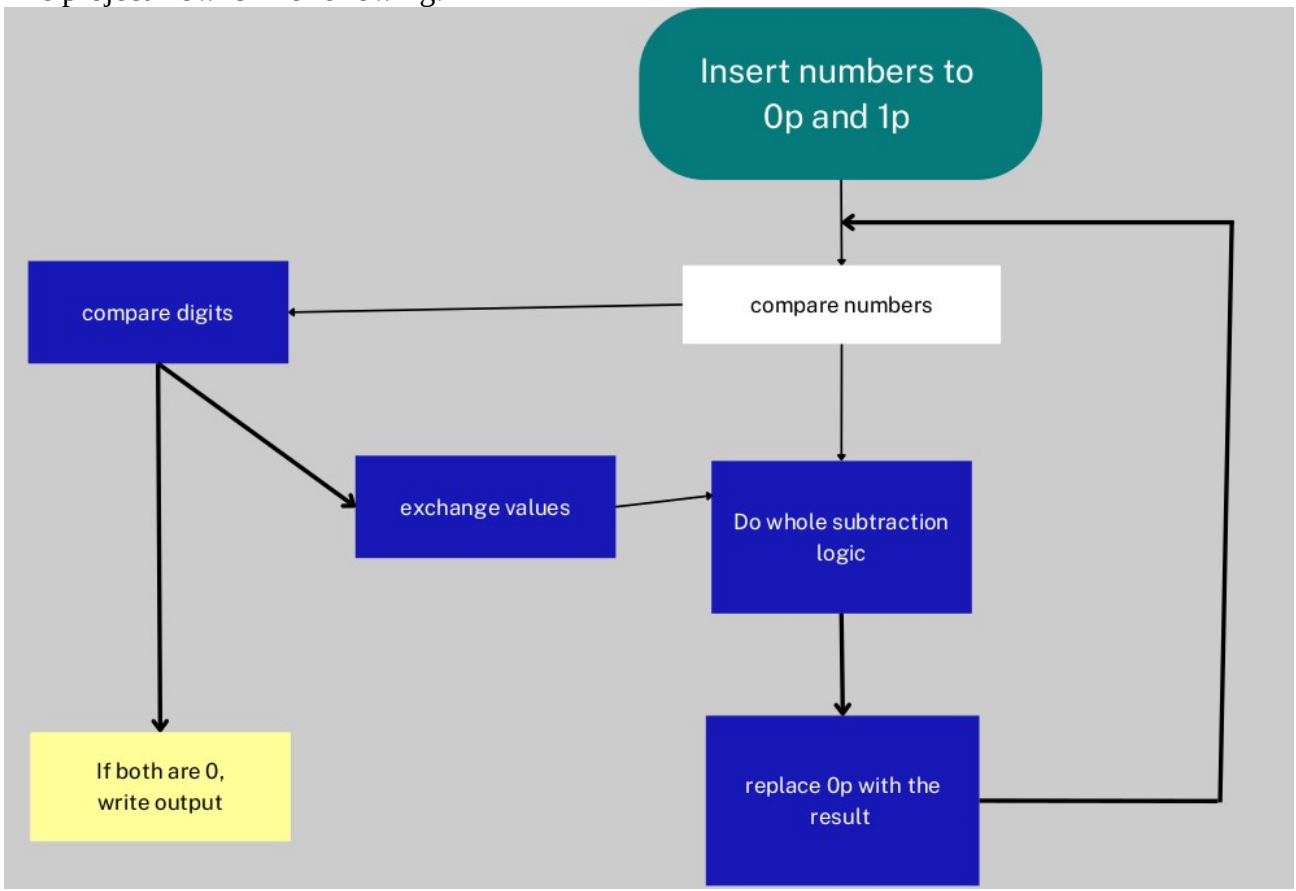

Figure 10.

After assembling the source code and loading the object code into the emulator, result will be like:

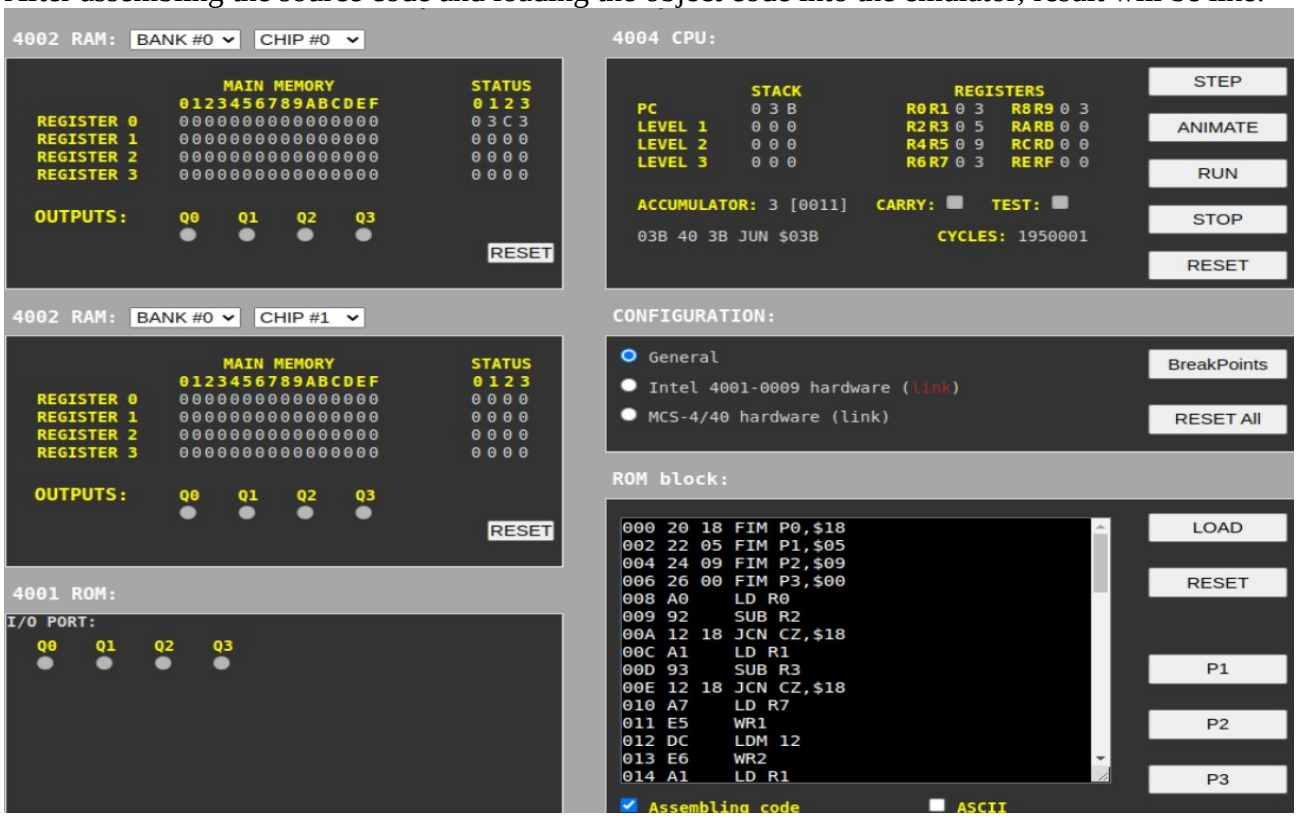

Figure 11. 18/5 = 3 (3)

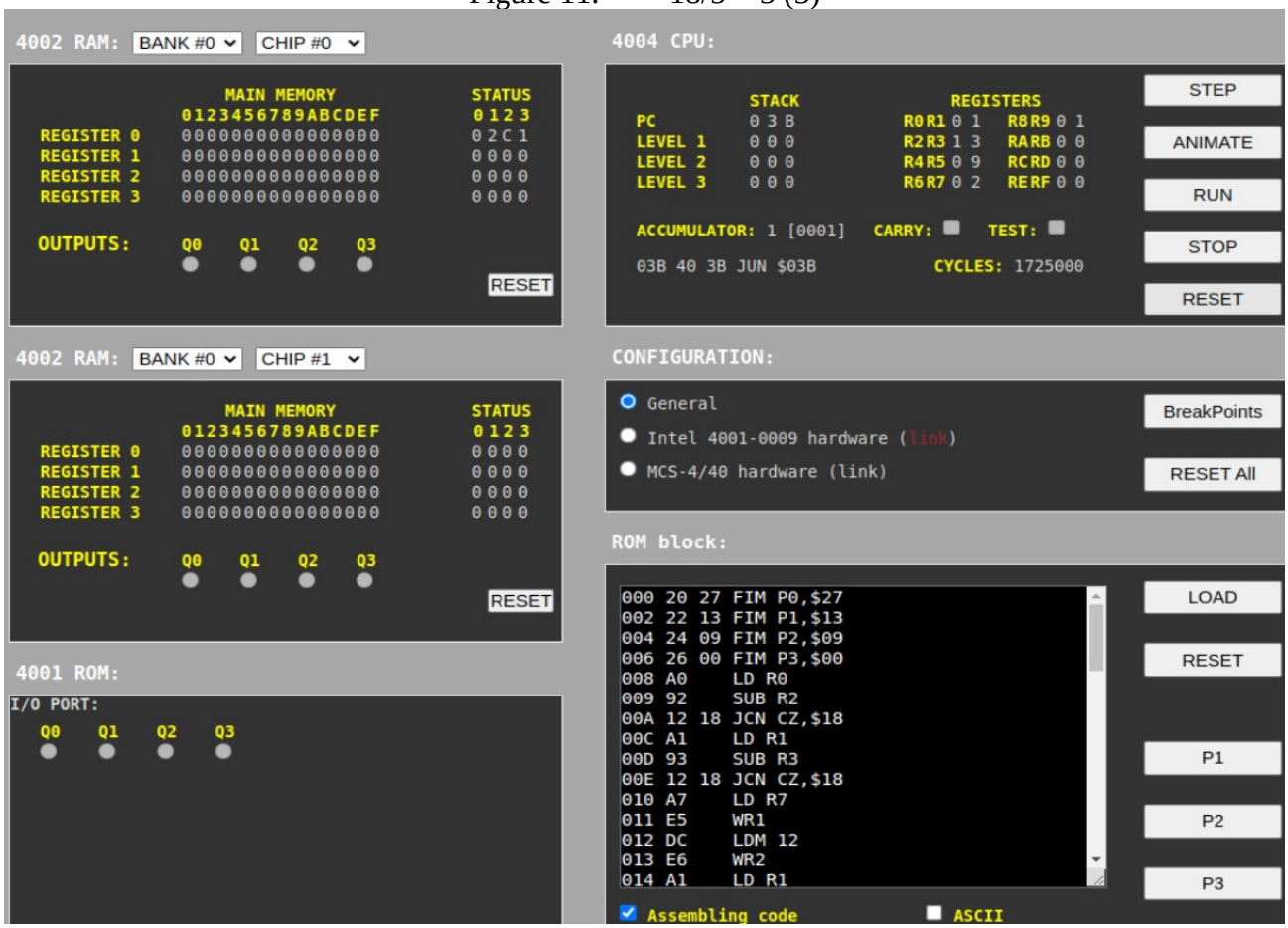

Figure 12. 27/13=2 (1)

5. Future improvements

For what it is worth, the assembly code is ready for the use and can be tested in the emulator. However, there is always room to grow and there are 2 things that I want to improve about the project.

5.1. More operations

Since the intel 4004 is designed to be used as a calculator in small business. Adding more operations will be good implementation and help the code get richer. Operations like finding percentage, power or square root can be implemented with some modifications on already written code. Surely, as the operations get complex, code will get more complex and detailed.

5.2. User Friendly

Second improvement can be making code more user friendly and easy to use. Currently, I have got 4 separate source code and for each calculation I have to change the values by hand. I am planning to write a C code which asks the user for an operation and values which will automatically generate the assembly source code to be ready for loading to emulator.

6. Conclusion

Overall, while doing this task, I have examined the architecture of Intel 4004 and its instruction set which made me appreciate the work had been done in 70s. There have been so many difficulties and limitations during the coding process which I believe knowing these limitations were the main reason that tackled the future designs.

7. Source Code