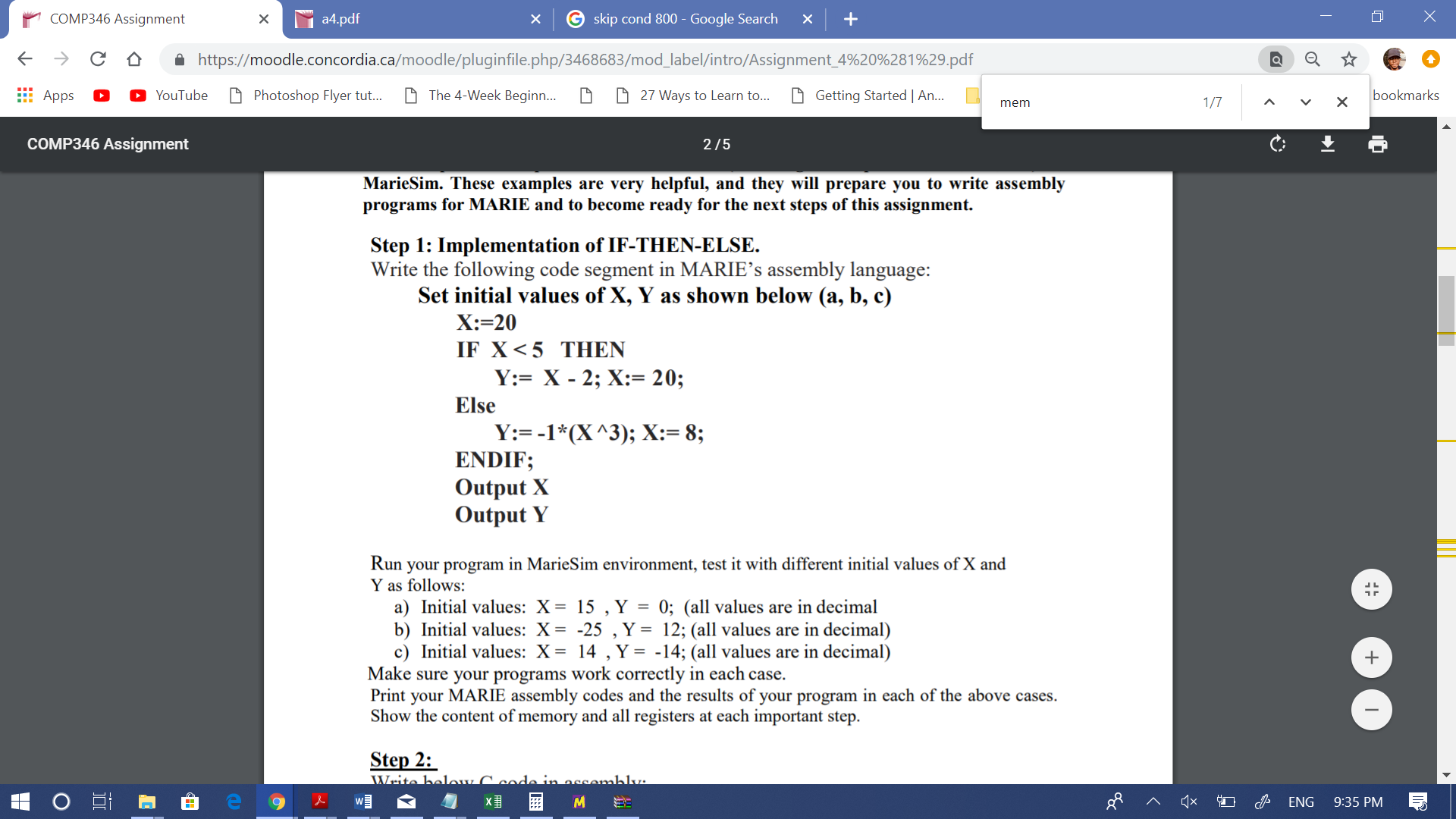
**System Hardware – COMP 228 Assignment 4**

**Team Members: George Mavroeidis & Johanson Felix**

**Step 1: Implementation of IF-THEN-ELSE**



**Program Explanation:**

In short (2-3 lines),

This program initializes a value in an address ‘X’ to 20, then changes the value of X to a new value. If the new value of X is less than 5 then the value in an address ‘Y’ is changed to the value of X subtracted by 2 and the value in address X is changed to 20. Else if X is not less than 5 then then the value in an address ‘Y’ is changed to negated X cubed (done by a loop of addition) and the value of in address X is changed to 8. Both Values of X and Y are then loaded and printed.

In long (detailed),

This program loads the value twenty (20) into the accumulator and stores this value in an address ‘X’. It then loads a new value (newX) to the accumulator and this is stored in address X. This ‘newX’ value (case a, b, c) is additionally stored in an address ‘count’ that will be used as a counter in a subsequent loop.

Conditional branching is then used to determine whether X is greater than 5 by subtracting 5 from the accumulator. If the accumulator is left negative (SKIPCOND 000), meaning X is less than 5, then the line that proceeds is skipped and the address Y is stored as the value of X minus 2. Finally, X is assigned 20 from the accumulator. The program jumps to the end block which loads and outputs X and Y.

If the accumulator is not left negative, meaning X is greater or equal to 5, then Y is stored as the value of X cubed (by simulation of an addition loop), then this value is negated. Finally, X is assigned the value eight from the accumulator. The program jumps to the end block which loads and outputs X and Y.

*Note: The program is tested with 3 case values –*

1. X=20, X=15, Y=0
2. X=20, X=-25, Y=12
3. X=20, X=14, Y=-14

**Source Code:**

ORG 000 //Store program using memory location x000

//Initial value of X is loaded as 20

Load X

//New value for X is loaded into accumulator and stored in X

Load newX

Store X

//Count for subsequent loop takes value of X

Store count

// Subtract five from X, result will be in AC, this will be used to check if X<5

Subt five

//Check to see if AC is negative (implies X<5). If so, skip line Jump Else, and continue from If block

Skipcond 000

//If AC is not negative (X>=5), then execute Jump Else to skip If block

Jump Else

//if AC is negative (implies X<5) then load X back into AC, Subtract two from AC, Store value in AC as Y, Load twentty to AC and Store as X, Jump to end block

If, Load X  
 Subt two   
 Store Y  
 Load twenty  
 Store X  
 Jump End

//If AC is not negative (X>=5), then Jump to Loop1

Else, Jump Loop1

//Simulation of cubing a number by an addition Loop( Loop1 with Count2, Loop2):

//Load X into accumulator, Add value of sum1 (initially 0), Store value of AC in sum1, load count to AC (count initially set to value of X), subtract one from AC, Store AC to count, if AC>0 then skip next line and jump to Loop1 block again, else Jump to Count2 block

Loop1, Load X   
 Add sum1   
 Store sum1  
 Load count   
 Subt one   
 Store count   
 Skipcond 800   
 Jump Count2   
 Jump Loop1

//load X into AC, store value of AC (X) in count and follow code to Loop2

Count2, Load X   
 Store count

//Load sum1 into AC, Add value of sum to AC, Store AC to sum, Load count to AC, subtract one from AC and store AC to count, if if AC>0 then skip next line and jump to Loop2 block again, else Jump to End block

Loop2, Load sum1   
 Add sum   
 Store sum   
 Load count   
 Subt one   
 Store count   
 SkipCond 800  
 Jump Negate   
 Jump Loop2

//Simulation of negating a number by subtracting the number from itself which results in 0 then subtracting the number from 0 which results in the number negated

Negate, Load sum  
 Subt sum   
 Subt sum   
 Store Y   
 Load eight   
 Store X   
 Jump End

//Load X and Y to AC, output X and Y, halt program

End, Load X   
 Output   
 Load Y   
 Output   
 Halt

//Identifier Declarations:

X, DEC 20 /'initial value of X as 20'

newX, DEC 15 /'value of X in each case(a,b,c) 15, -25, 14'

Y, DEC 0 /'value of Y in each case(a,b,c) 0, 12, -14'

one, DEC 1 /'value of 1'

two, DEC 2 /'value of 2'

five, DEC 5 /'value of 5'

eight, DEC 8 /'value of 8'

twenty, DEC 20 /'value of 20'

count, DEC 0 /'count for loops'

sum1, DEC 0 /'sum of Loop1'

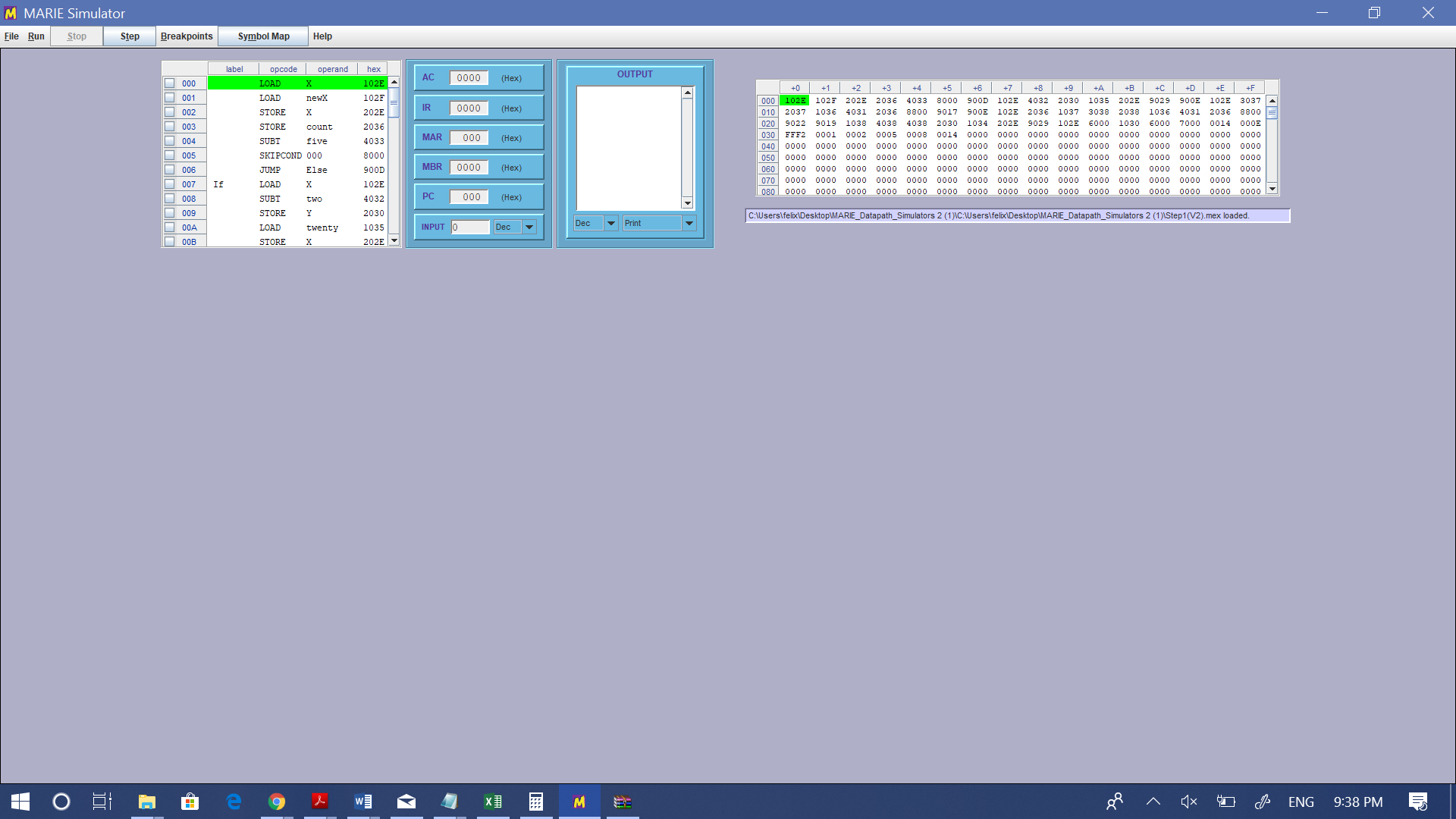
sum, DEC 0 /'Overall sum after looping'

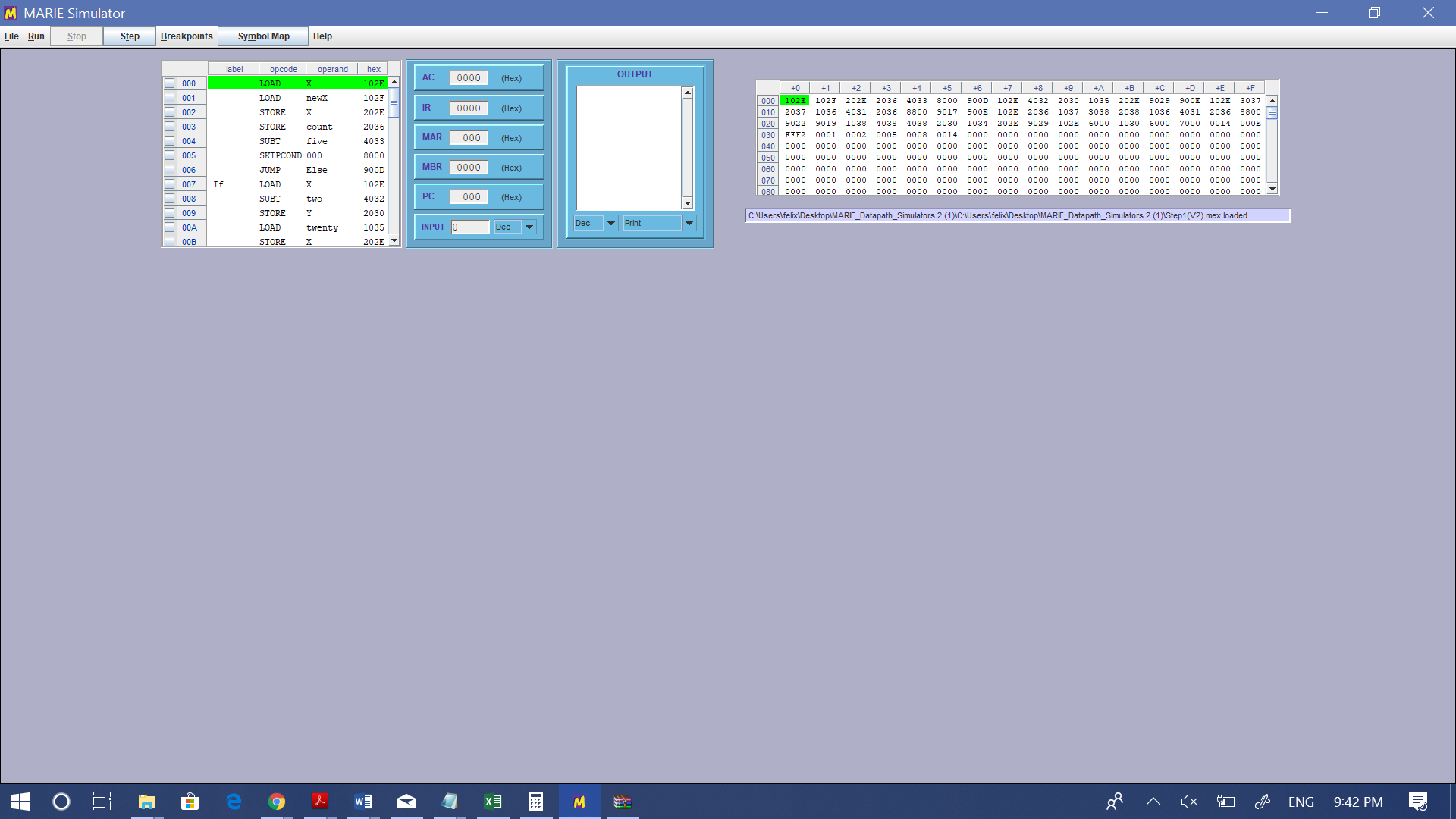
**Results, Contents/Outputs of important variables in Program:**

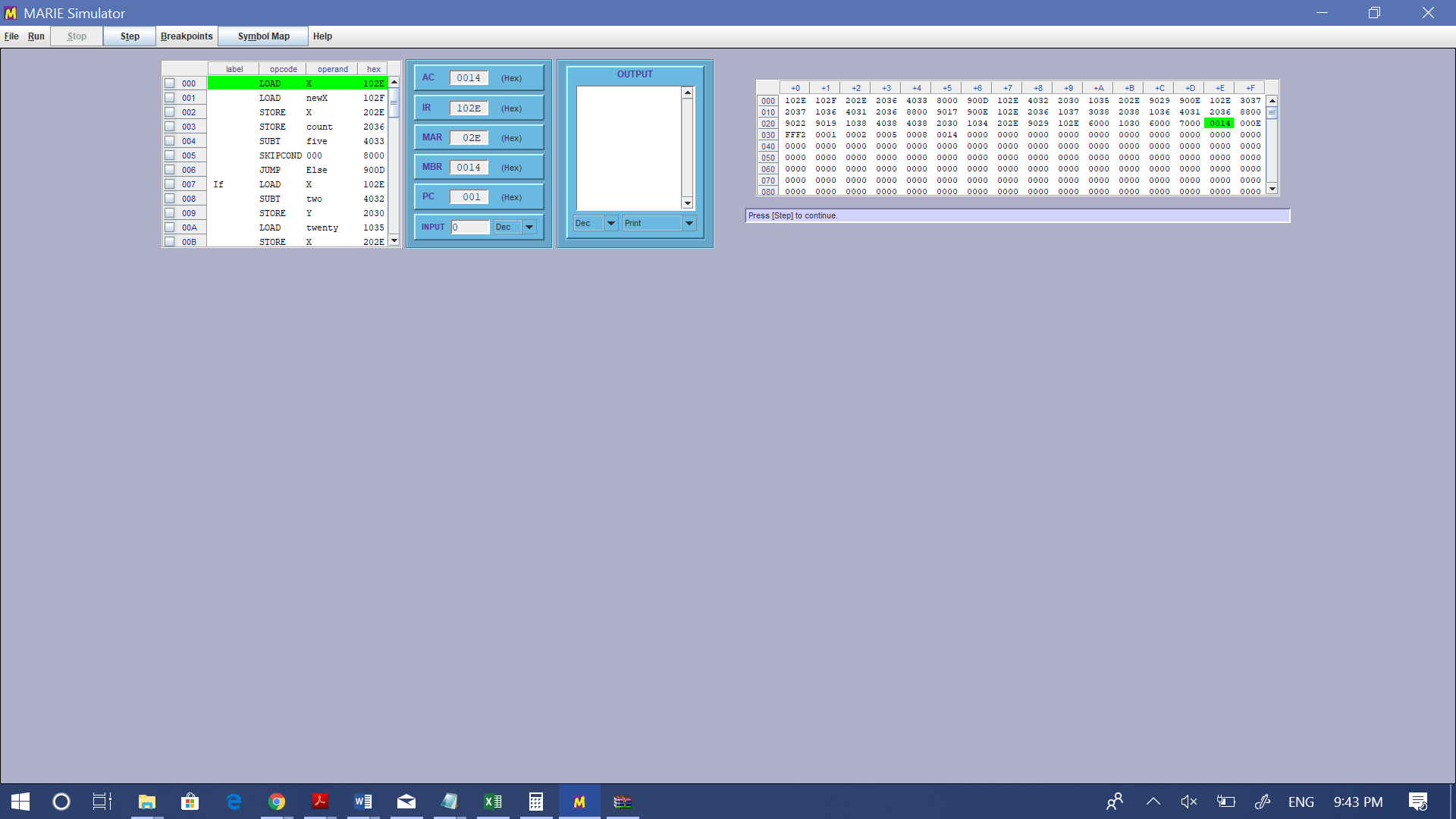
1. X= 8 Y= -3375
2. X=20 Y= -27
3. X=8 Y= -2744

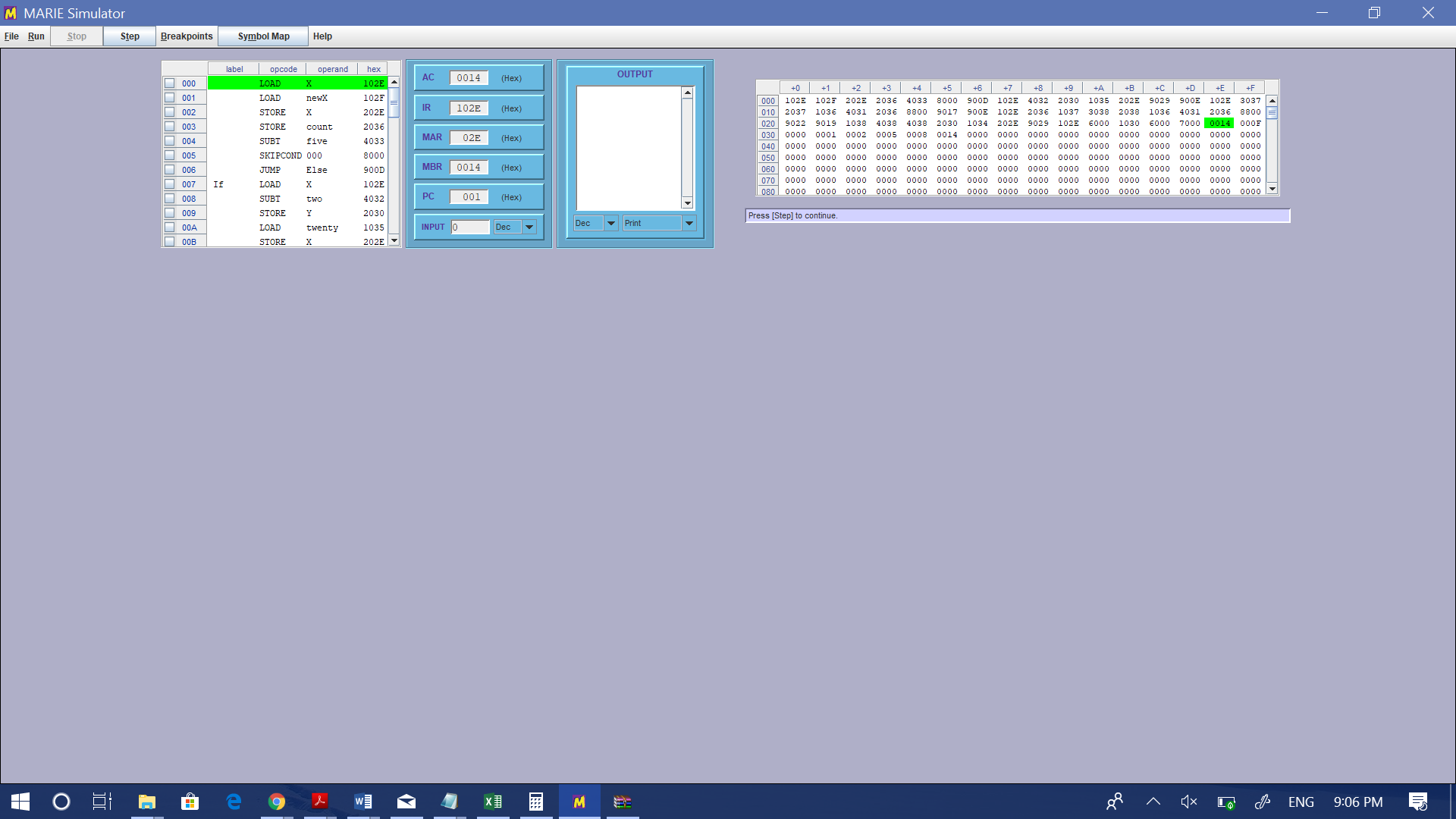
**Snapshots**

*Registers and Contents of Memory before program runs*

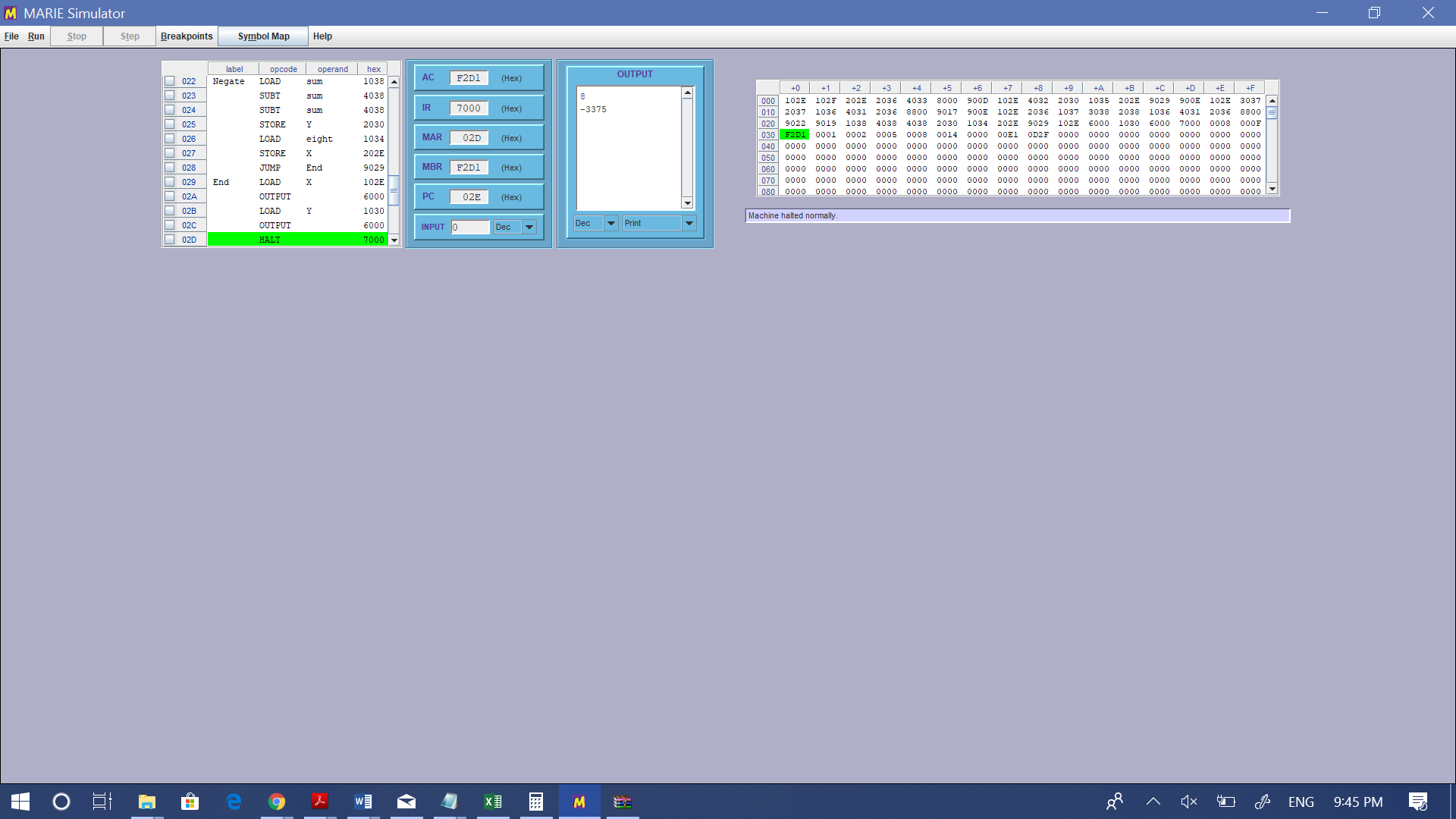


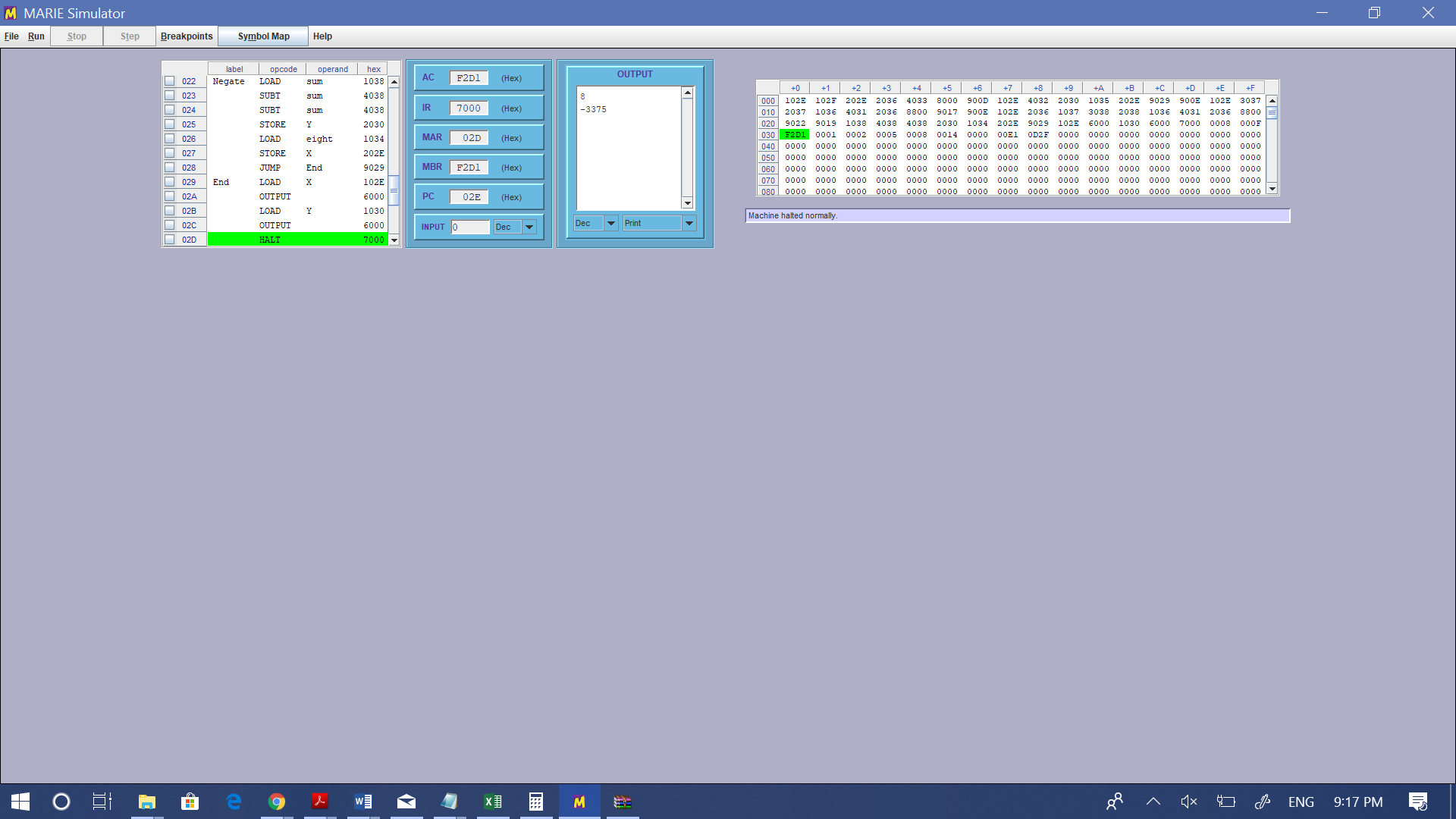


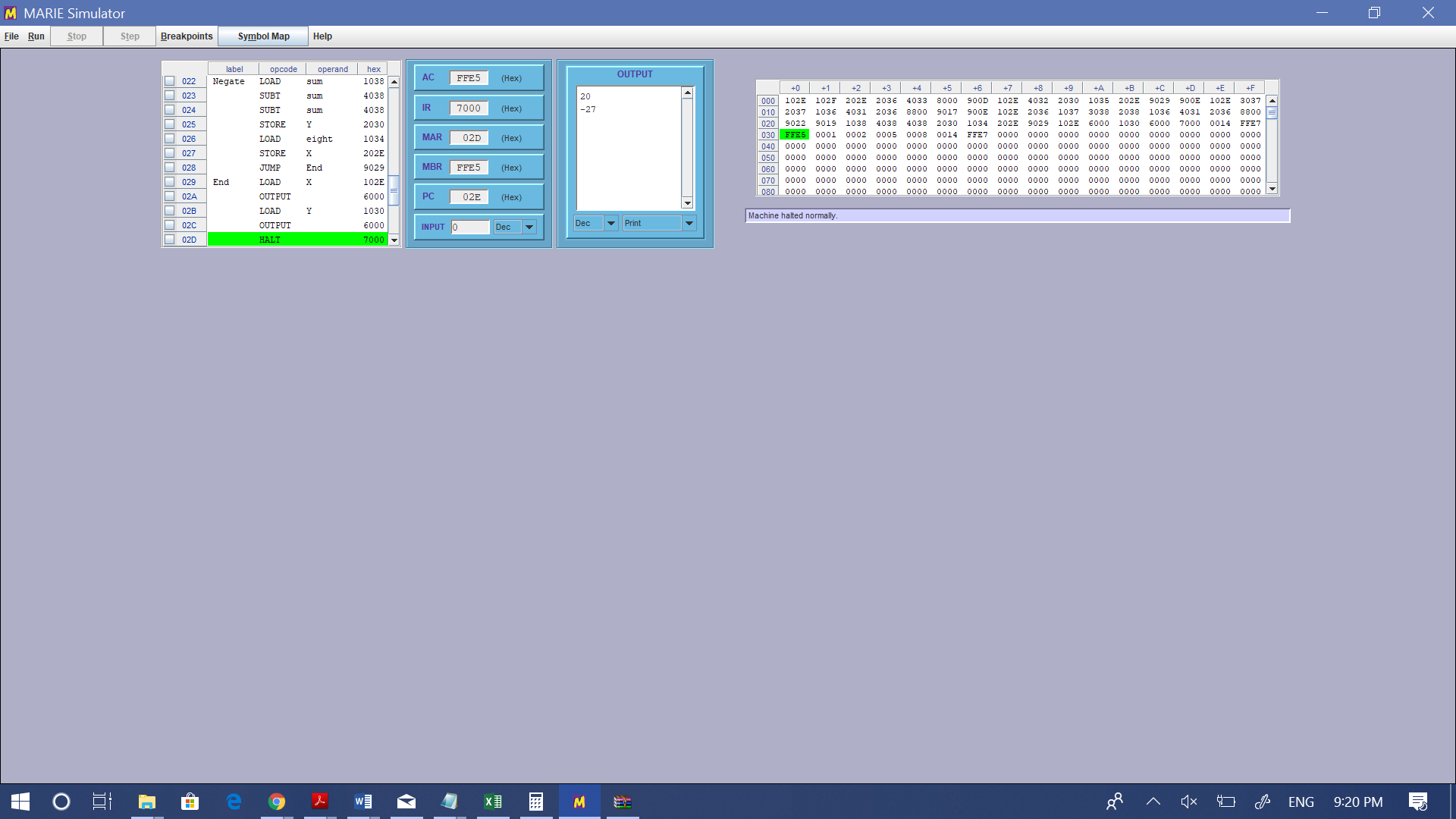
*Registers and Contents of Memory at first Step*

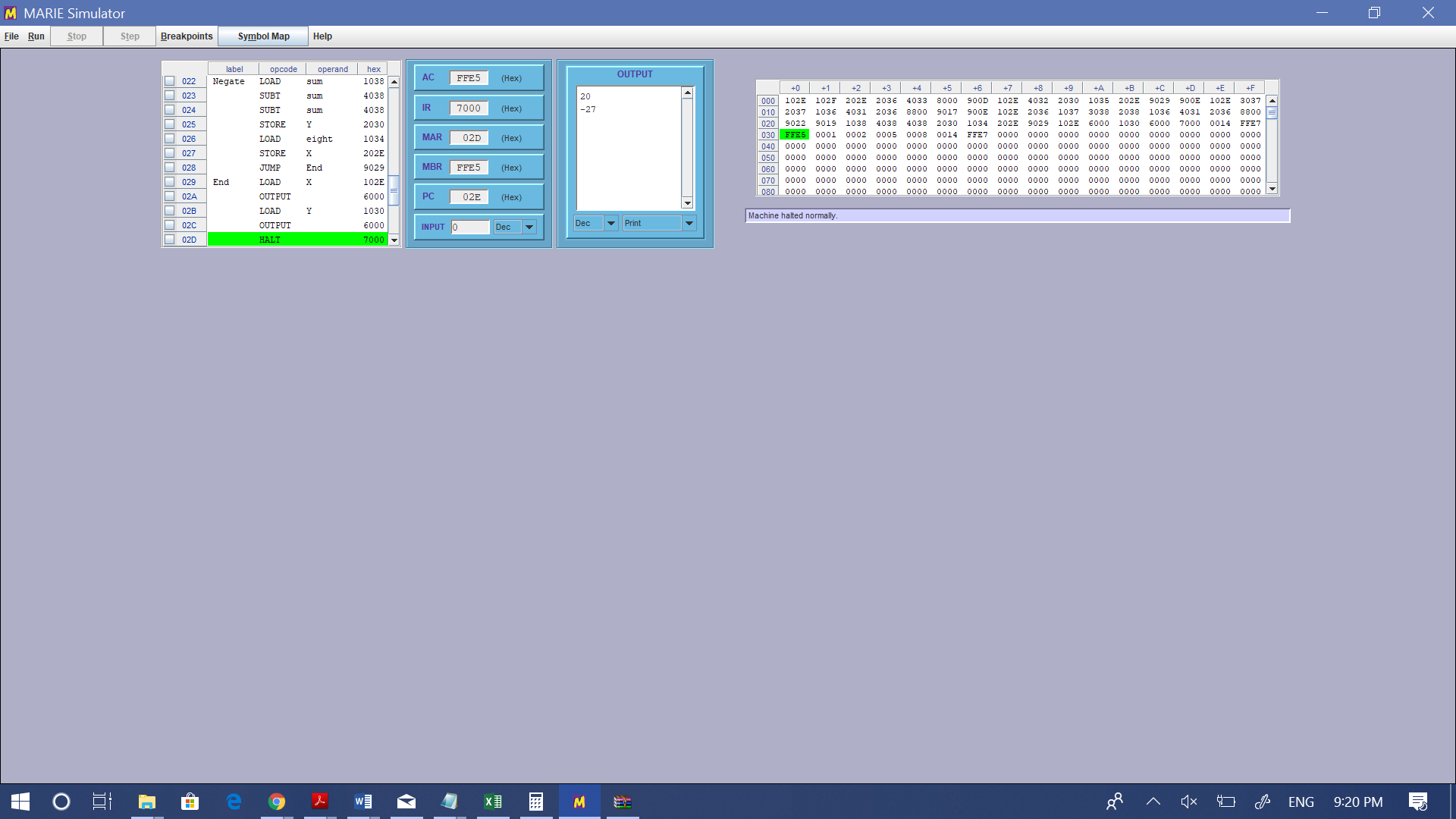


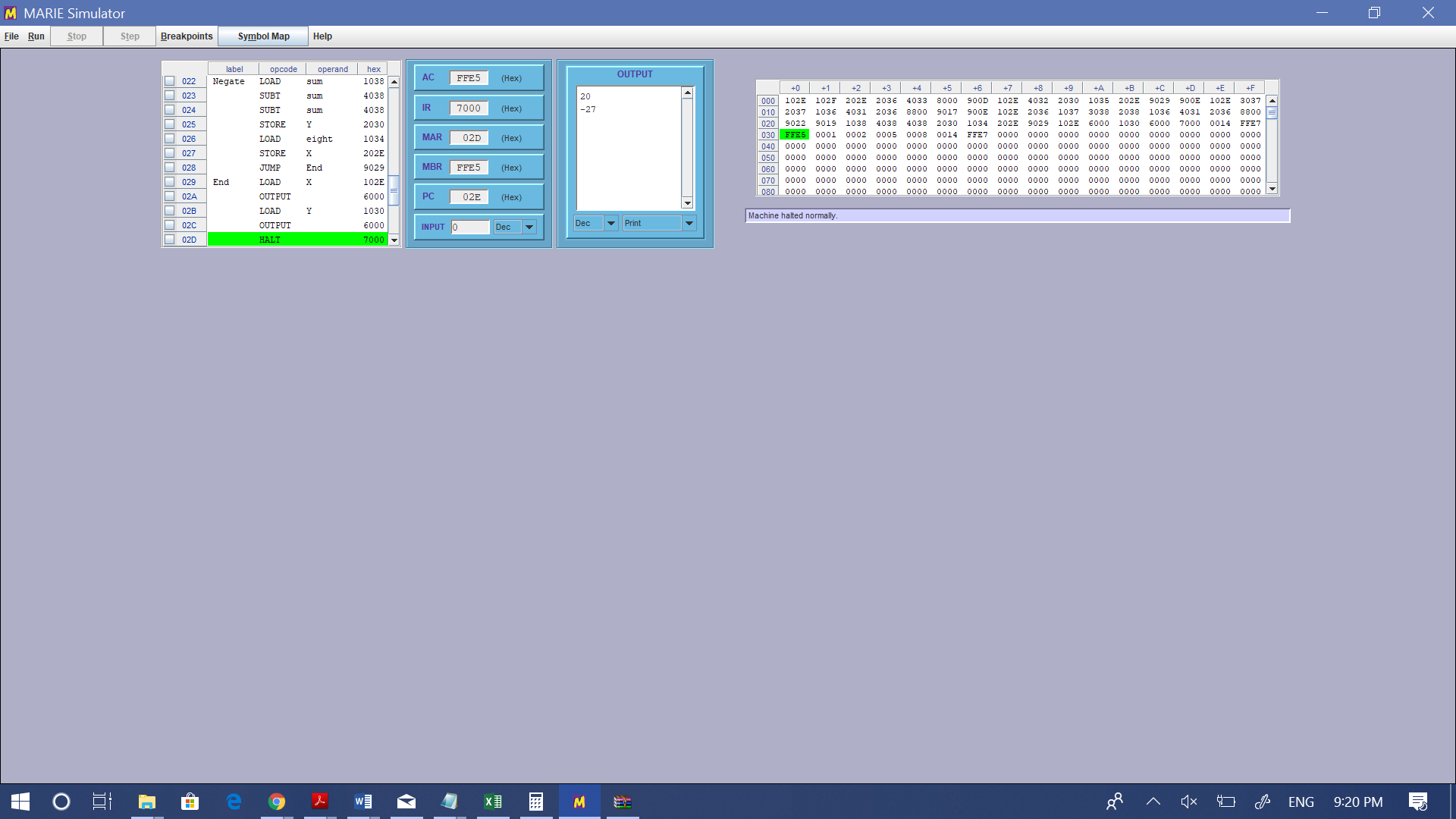
*CASE A*

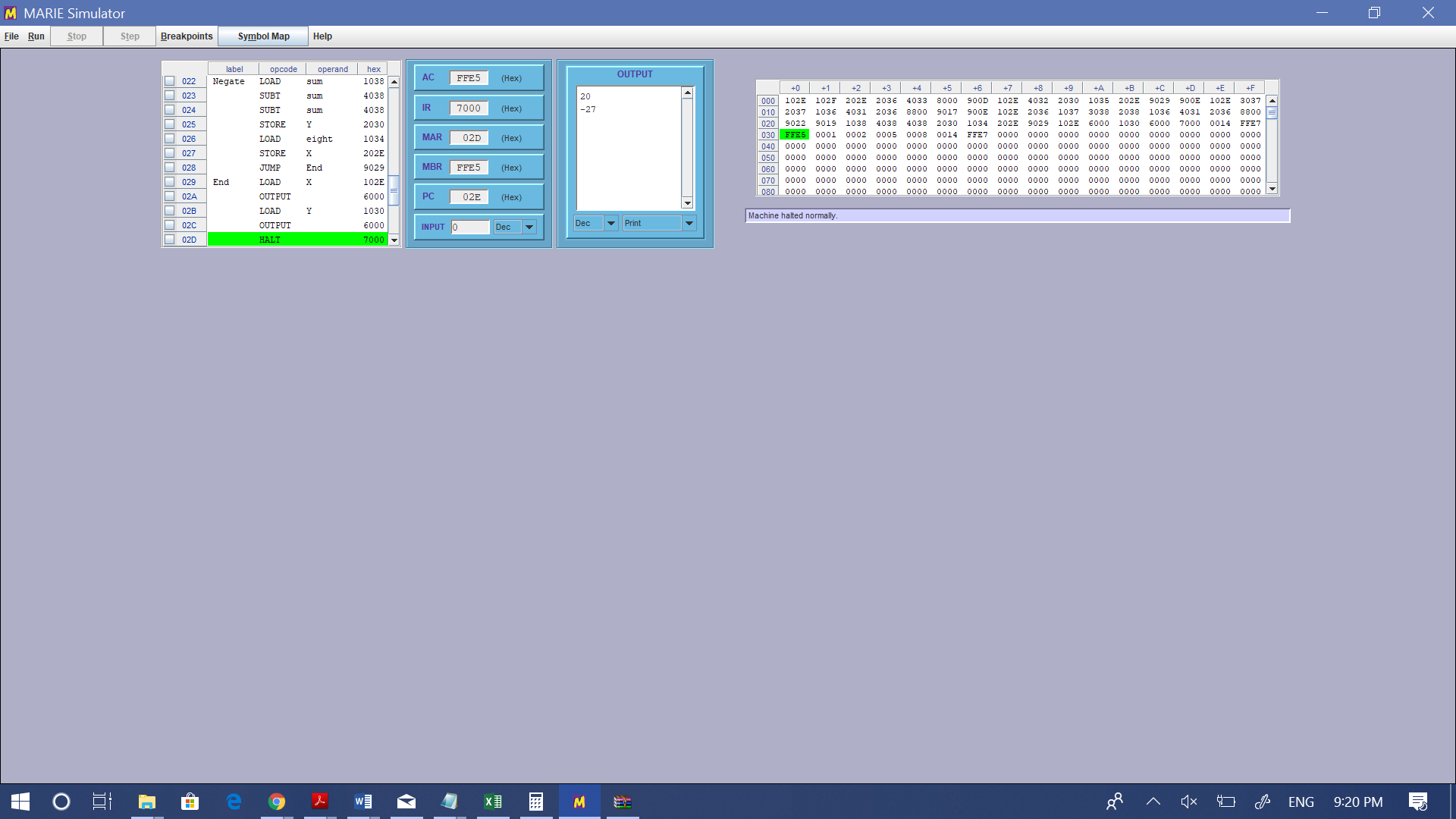




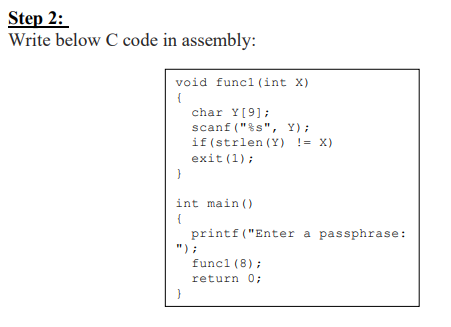
*CASE B*



*CASE C*



**Step 2: C code to Assembly translation**



**Program Explanation:**

In short (2-3 lines),

The program starts by prompting the user to enter the length of the word first, before entering the actual word. Then the line “Enter a passphrase:” is displayed via a loop and the user is prompted to enter the characters for the word. The characters are entered via a loop and the loop iterates until the length of the word entered initially. If the length of the word is not equal to ‘X=8’, then the program exists and outputs 1. If the condition is not met, the program exists and outputs 0.

In long (detailed),

The program starts by providing a user-defined decimal input to be stored in the address ‘length’. The contents of ‘length’ are loaded into the accumulator and stored as ‘Count2’, a counter for a subsequent loop.

A loop labeled ‘Loop’ loads indirectly the address of the array ‘A’ contents, which is a series of characters. The character within the specific address are printed. The address is loaded into the accumulator and one (1) is added in order to move to next address within the array. The address ‘Count’ is reduced by one (1) and it is checked under the condition ‘Skipcond 400’, where ‘Count’ is checked whether it equals to 0 or not. If it does, the loop iterates again and if not, the step moves forward outside of the loop. The end result is displaying the phrase: “Enter a Passphrase:”.

A second loop labeled ‘Loop2’ accepts ASCII input from user to be stored in array ‘A2’. The loop starts with an input. This input is stored indirectly on address ‘Add2’ which is the reference for the array index of ‘A2’. A counter ‘Count2’ is the array’s index, which is decremented by the address ‘newOne’ which contains a value of one (1). A condition ‘Skipcond 400’ checks whether the counter reaches 0. If it does not, the next step jumps at the beginning of the loop again.

A third loop labeled ‘Loop3’ outputs the characters of the array ‘A2’, which are the characters entered by the user in the above loop, ‘Loop2’. The address of the given array index is loaded indirectly in the accumulator. Then, the contents are displayed by the output operation. The address is incremented by ‘newOne’ and ‘Count2’ which is the loop counter is subtracted by ‘newOne’. Both are stored back to their address respectively. A condition ‘Skipcond 400’ checks whether the counter reaches 0. If it does not, the next step jumps at the beginning of the loop again.

An if-else labeled statement(s) is included which compares the length of the array, basically, the length of the word provided by the user. The address of length is loaded into the accumulator. The condition checks whether the length is equal to eight (8). To check this, eight is subtracted from the length and by implementing ‘Skipcond 400’, it checks whether the difference is equal to 0 or not. If the difference was indeed 0, the length of the word was equal to eight (8), therefore, the next step jumps to label ‘Else’, loads ‘zero’, which contains 0 and the program halts. If the difference was not 0, the length of the word was not equal to eight (8). This instead, sets the next step to jump to label ‘Exit1’, which loads ‘newOne’, which contains 1 and the program halts.

**Source Code:**

/ Store our program starting from memory location x000. Not necessary to include this. Default is 000.

ORG 000

/User Inputs length of passphrase. Passphrase length is stored and also stored as a counter for a subsequent loop

Input

Store length

Load length

Store Count2

/Loop to print string "Enter a Passphrase:"

Loop, LoadI Addr

Output

Load Addr

Add One

Store Addr

Load Count

Subt One

Store Count

SkipCond 400

Jump Loop

/Loop to accept input of passphrase of size length

Loop2, Input

StoreI Addr2

Load Addr2

Add newOne

Store Addr2

Load Count2

Subt newOne

Store Count2

SkipCond 400

Jump Loop2

Load Addr2

Subt length

Store Addr2

Load length

Store Count2

Loop3, LoadI Addr2

Output

Load Addr2

Add newOne

Store Addr2

Load Count2

Subt newOne

Store Count2

SkipCond 400

Jump Loop3

/If passphrase length is not equal to eight then jump to Exit1, else jump to Else

If, Load length

Subt eight

Skipcond 400

Jump Exit1

Jump Else

/If passphrase is equal to 8 then Load zero into the AC then halt

Else, Load zero

Halt

/If passphrase is not equal to 8 then Load one into the AC then halt

Exit1, Load newOne

Halt

length, DEC 000 /length of passphrase

zero, DEC 000 /'value of zero'

eight, DEC 008 /value of 'eight'

Addr2, HEX 036 /Holds the address of A2

Count2, DEC 000 /counter that allows the loop to run for the length of the passphrase

newOne, DEC 001 /value of 'one'

A2, HEX 000 /Characters of the passphrase input are stored starting from this location

Addr, HEX 03A /Holds the address of A

Count, DEC 19 /Counter for the loop for printing the string "Enter a Passphrase:password"

One, DEC 001 /value of 'one'

A, HEX 045 / E

HEX 06E / n

HEX 074 / t

HEX 065 / e

HEX 072 / r

HEX 020 / space

HEX 061 / a

HEX 020 / space

HEX 0050 / P

HEX 061 / a

HEX 073 / s

HEX 073 / s

HEX 070 /p

HEX 068 /h

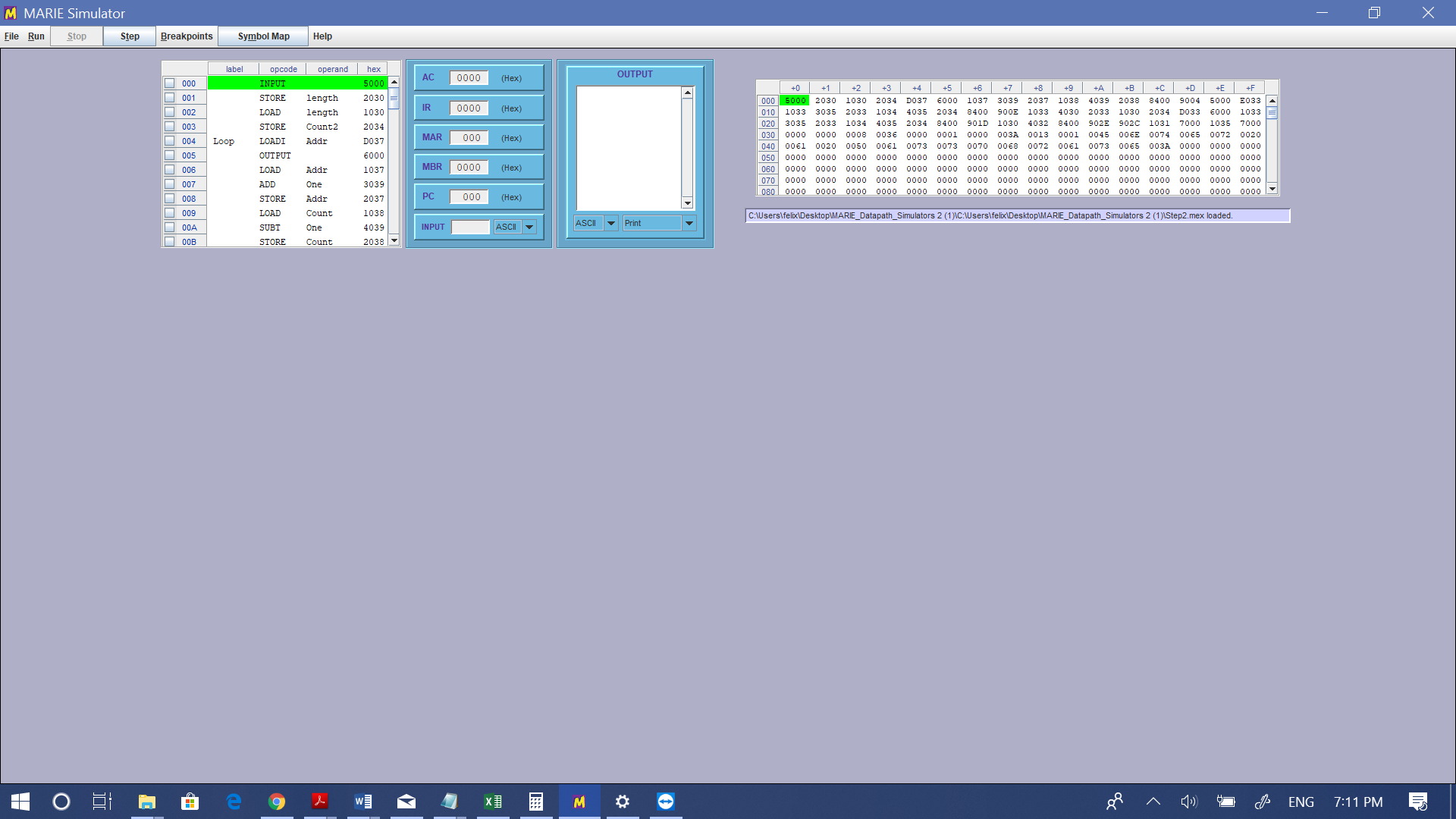
HEX 072 /r

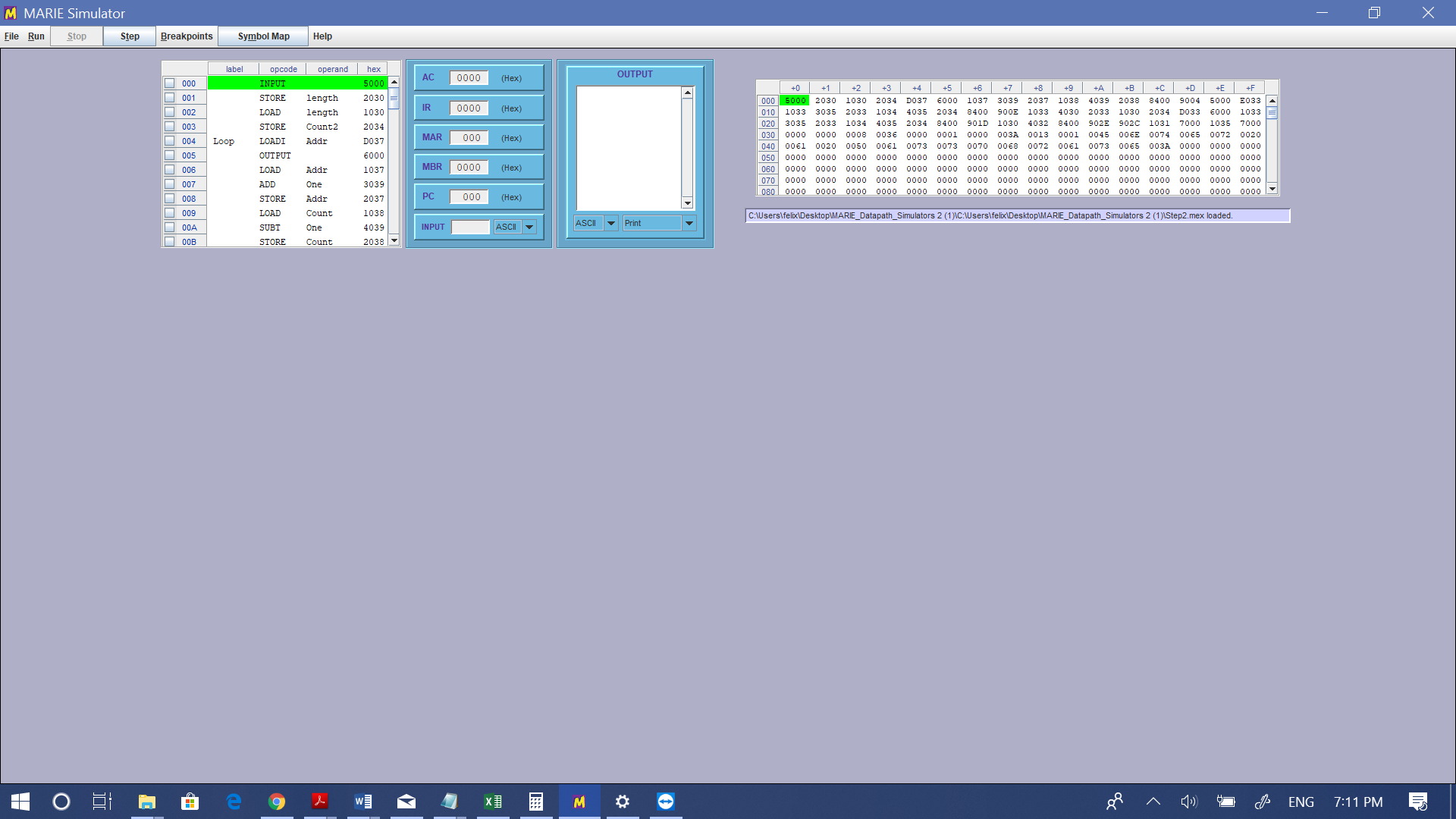
HEX 061 /a

HEX 073 /s

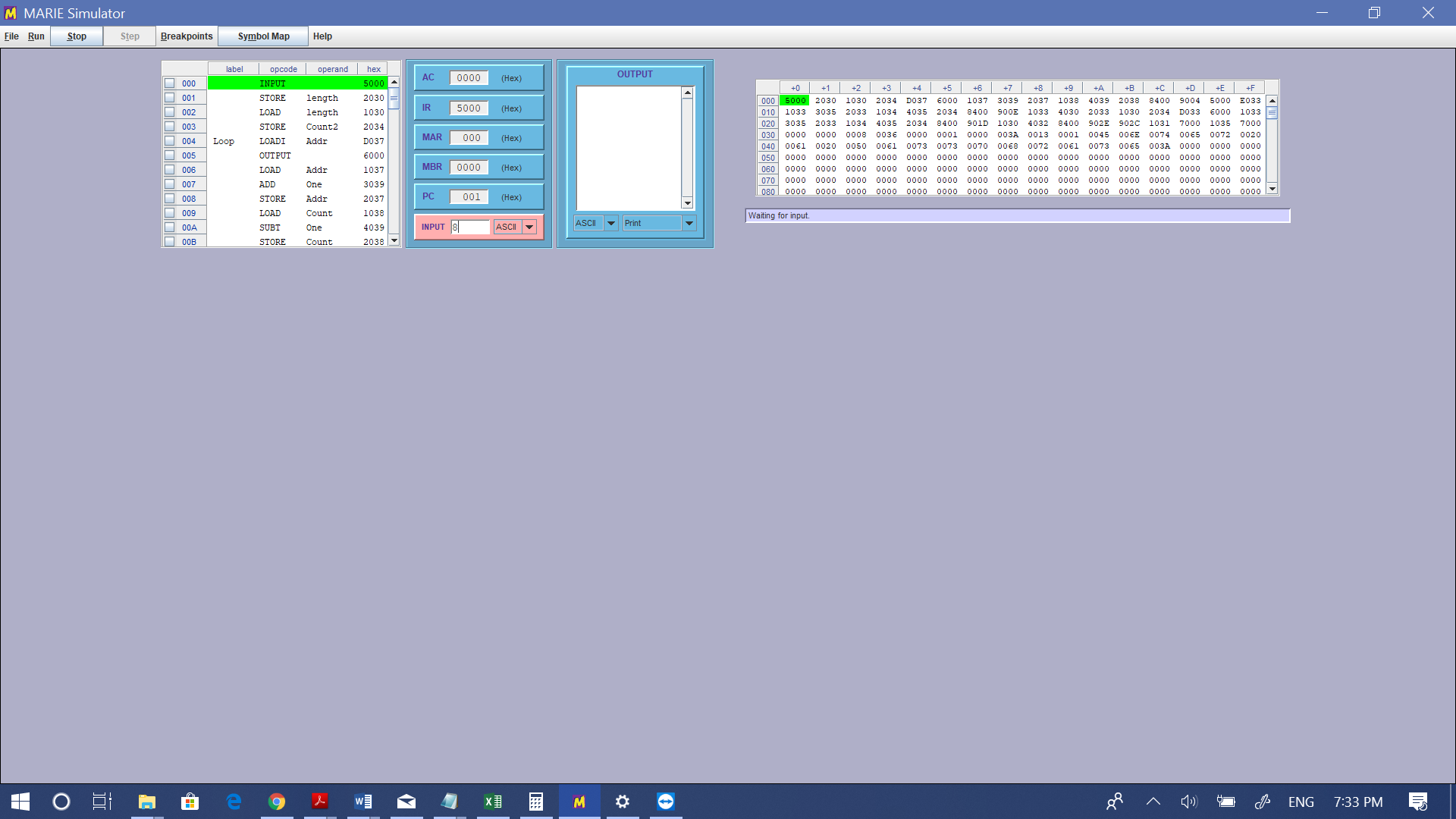
HEX 065 /e

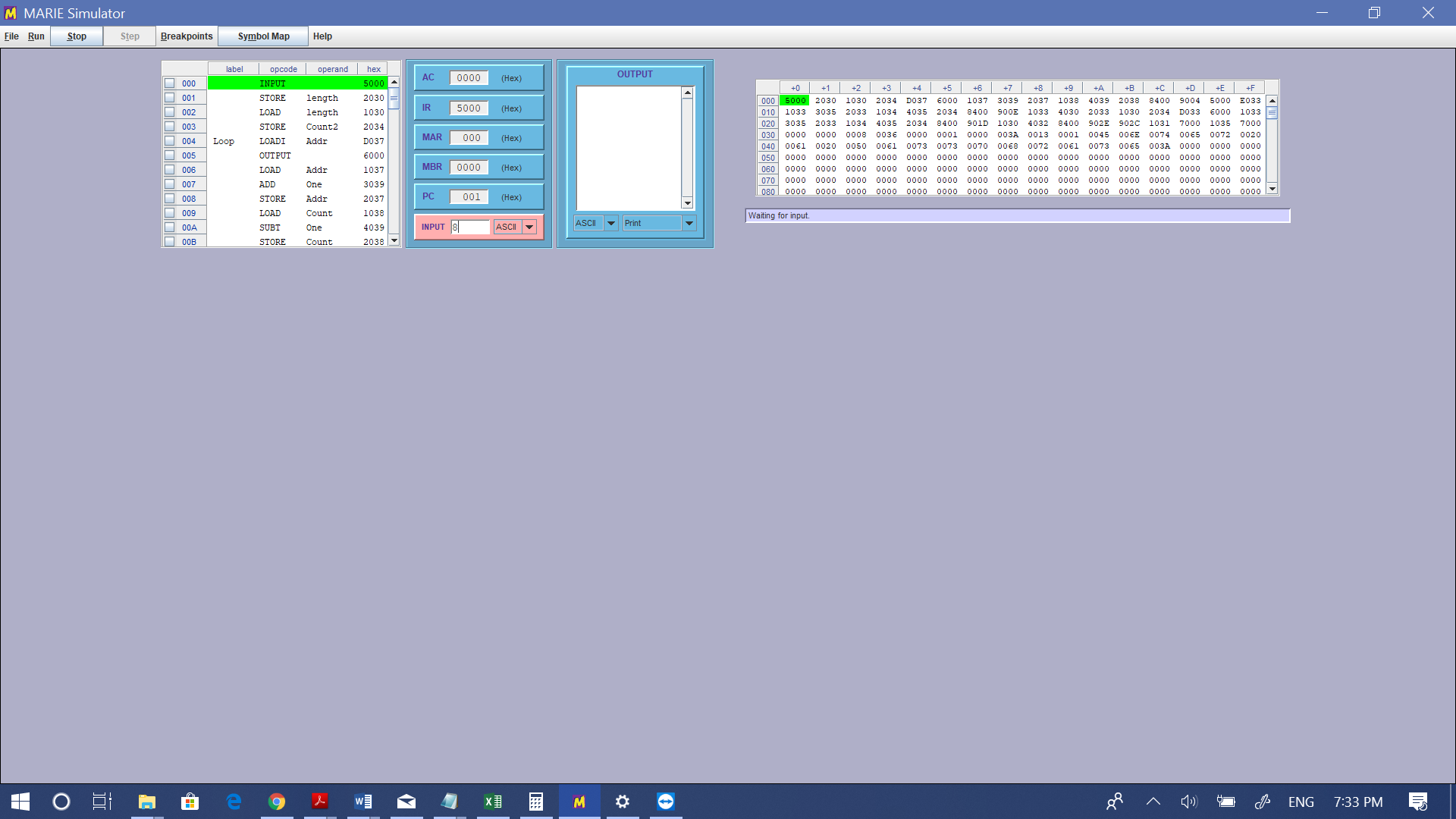
HEX 03A /:



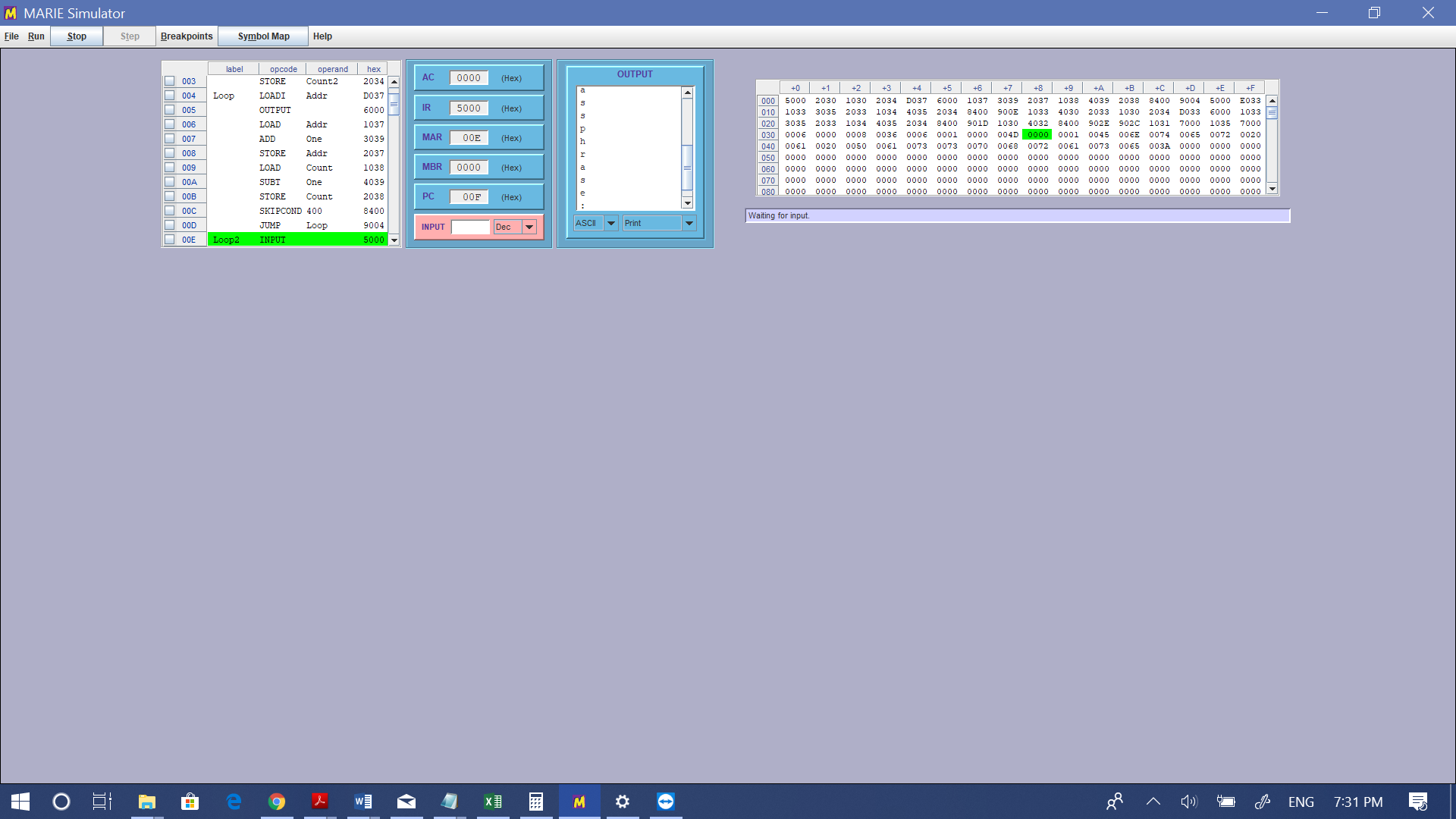


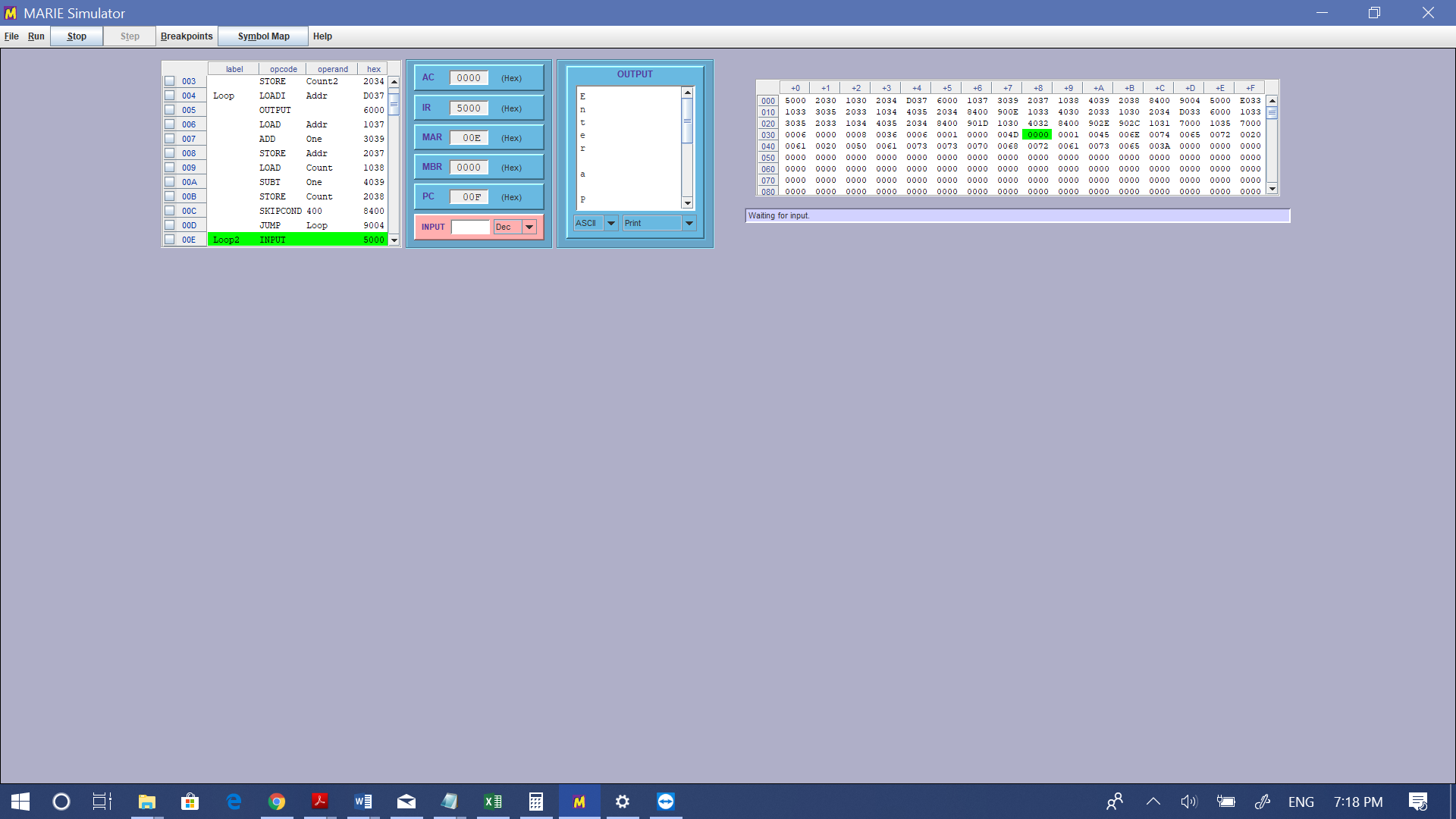
1. *State of the code before the program is excecuted*



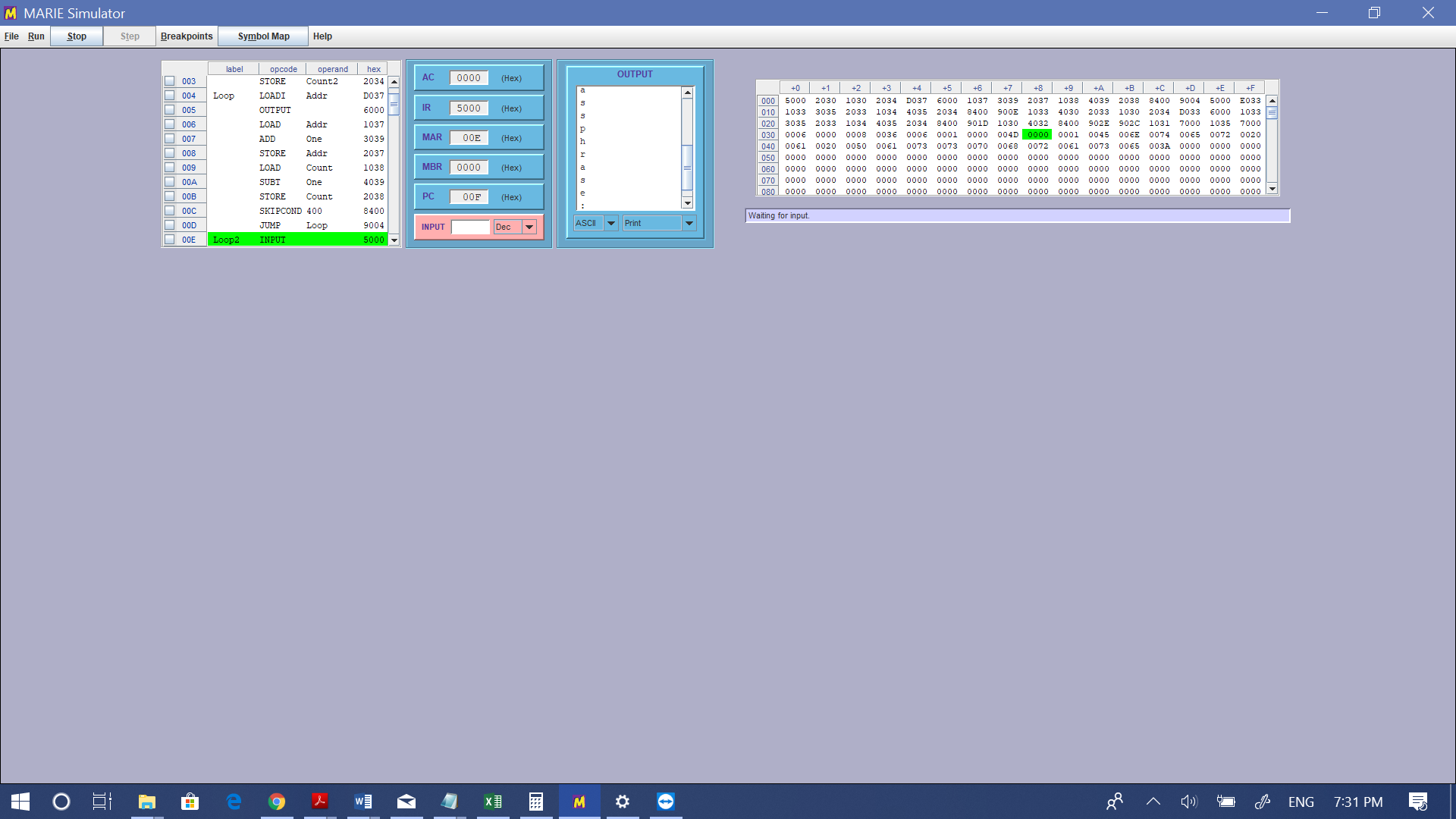


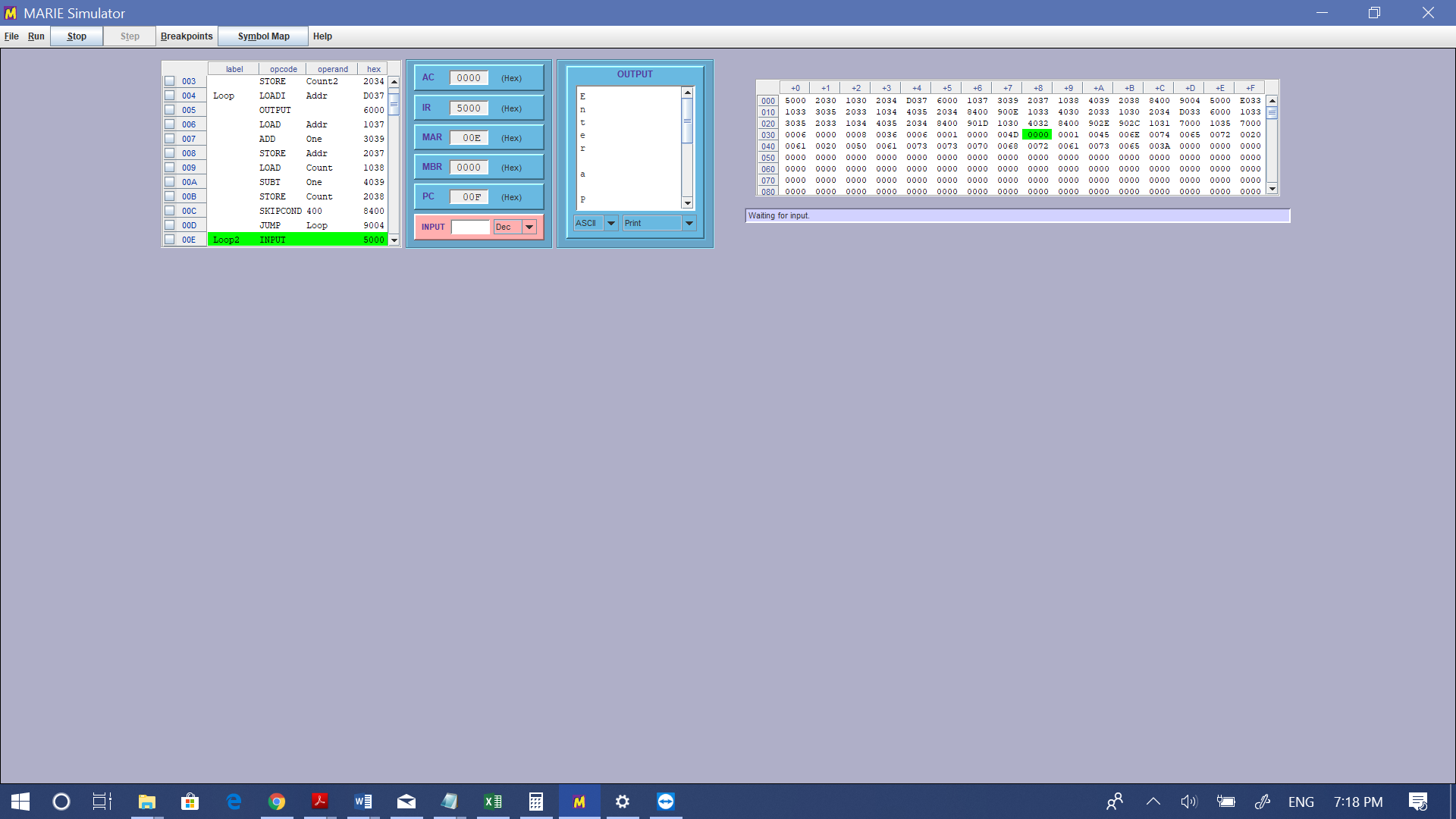
1. *Run #1: A word of length 8 is being attempted as a demonstration for returning 0*



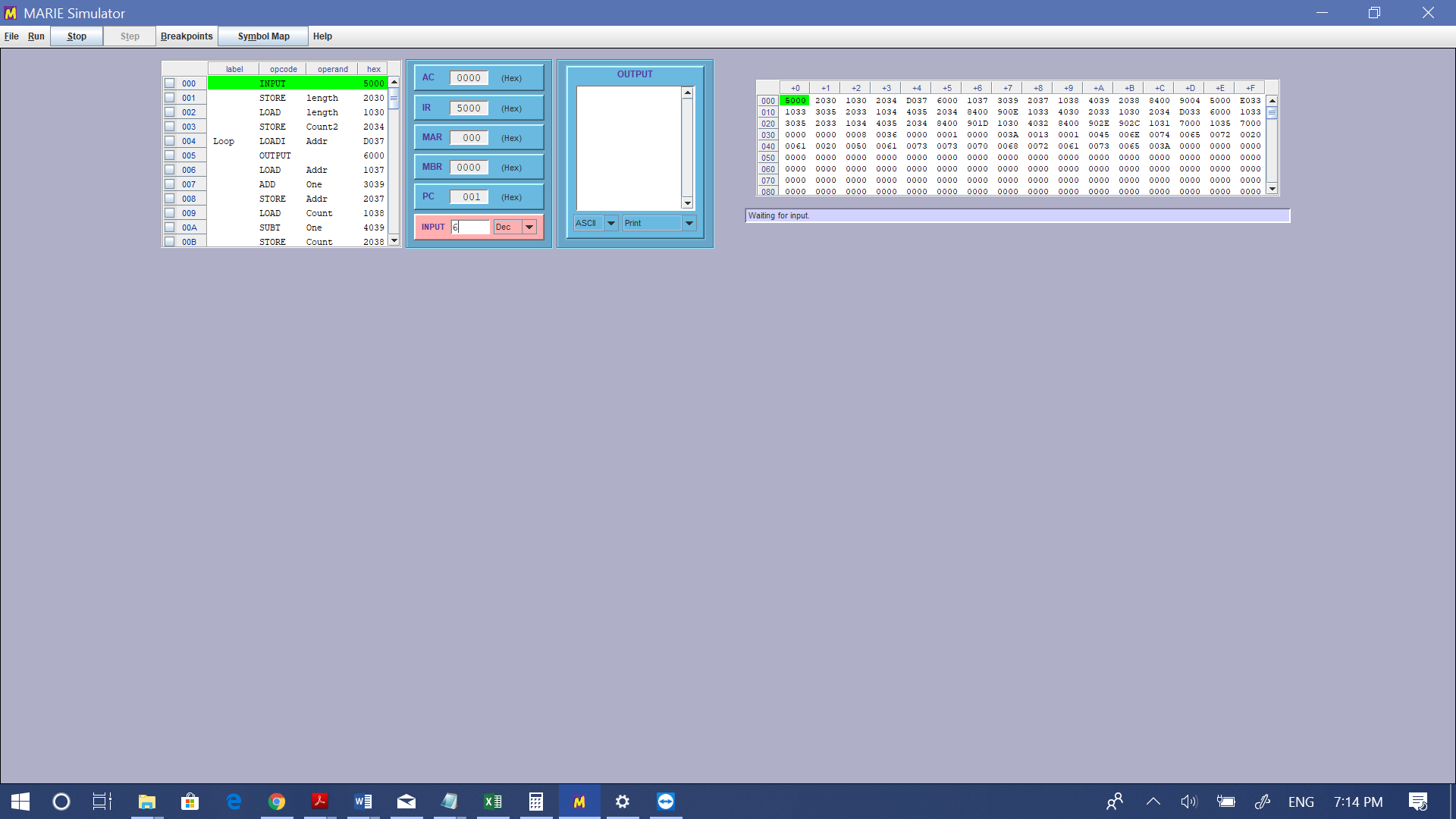


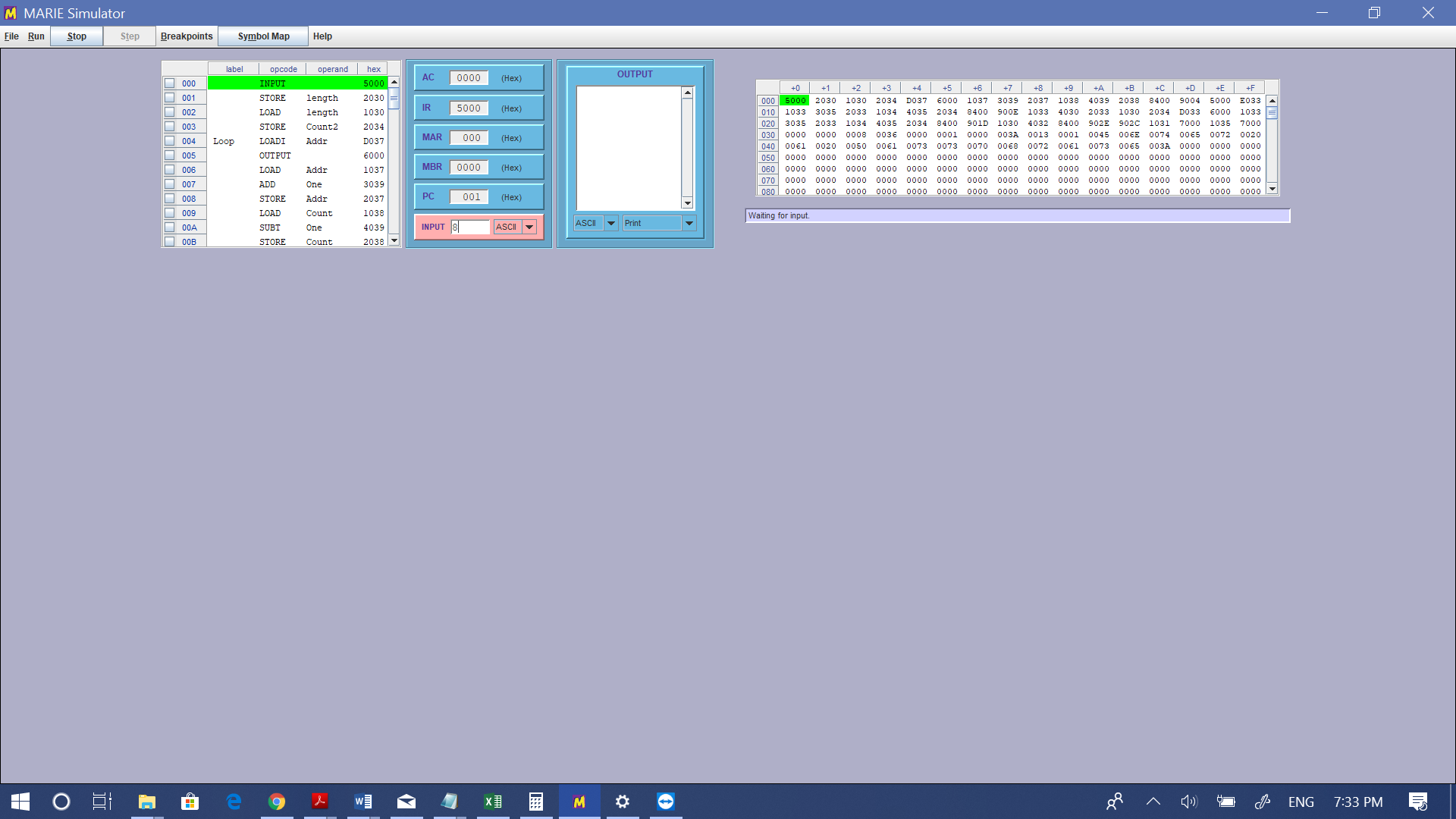
*Run #1: The phrase “Enter a Passphrase:” is printed*



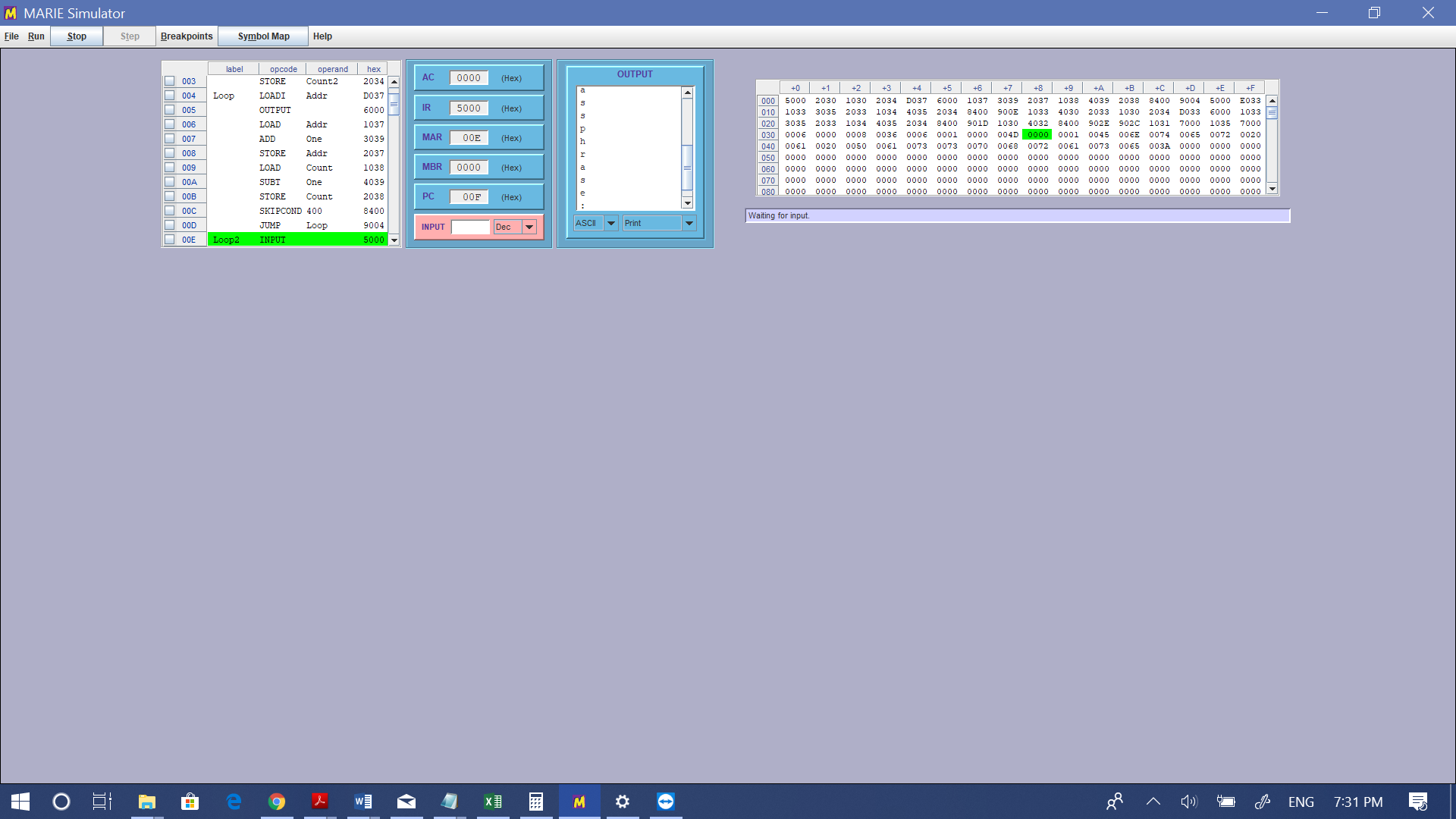


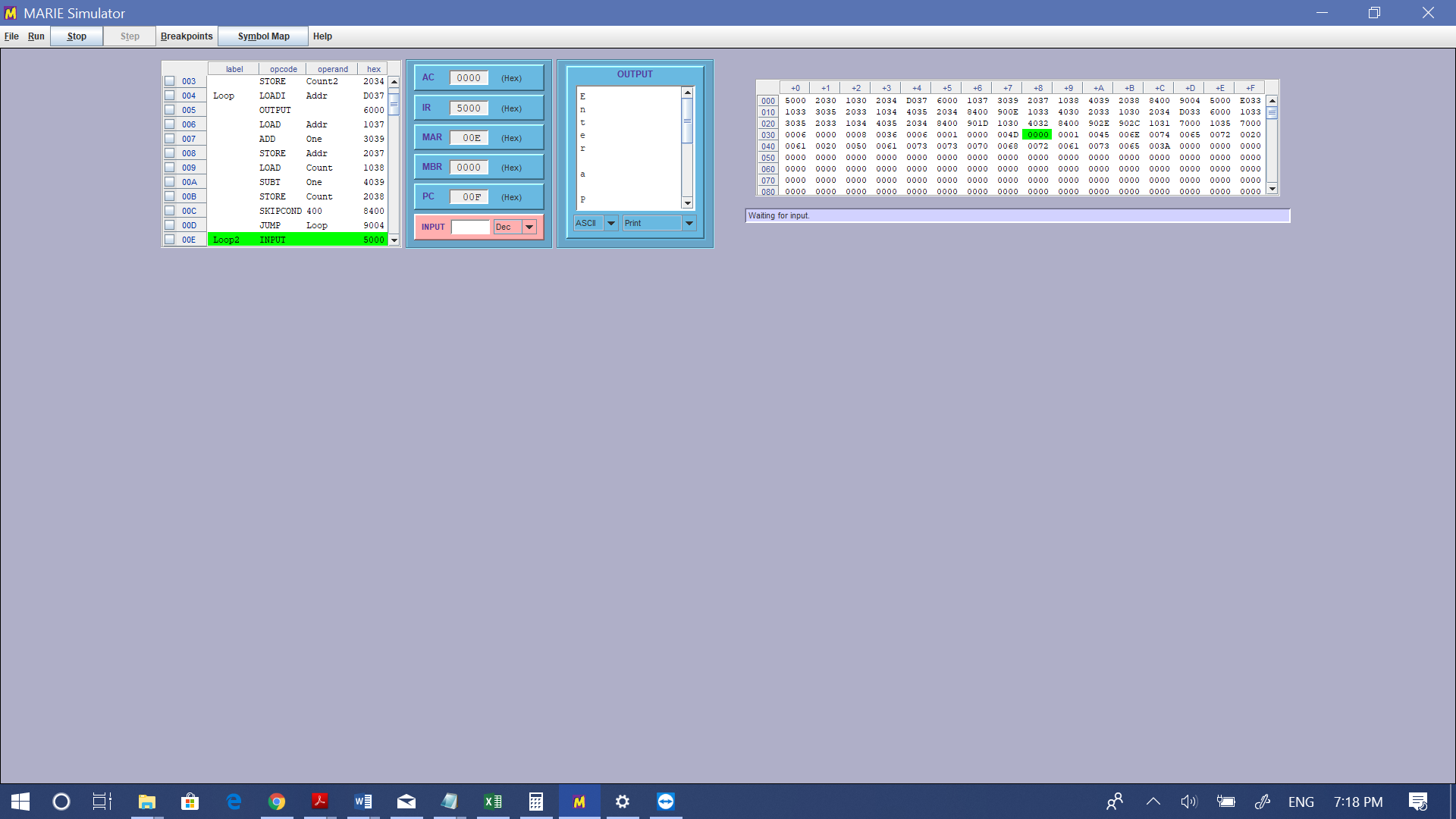
*Run #1: The word “password” has a length of 8 characters. It equals to X, therefore, the program will return 0 at the respective address.*



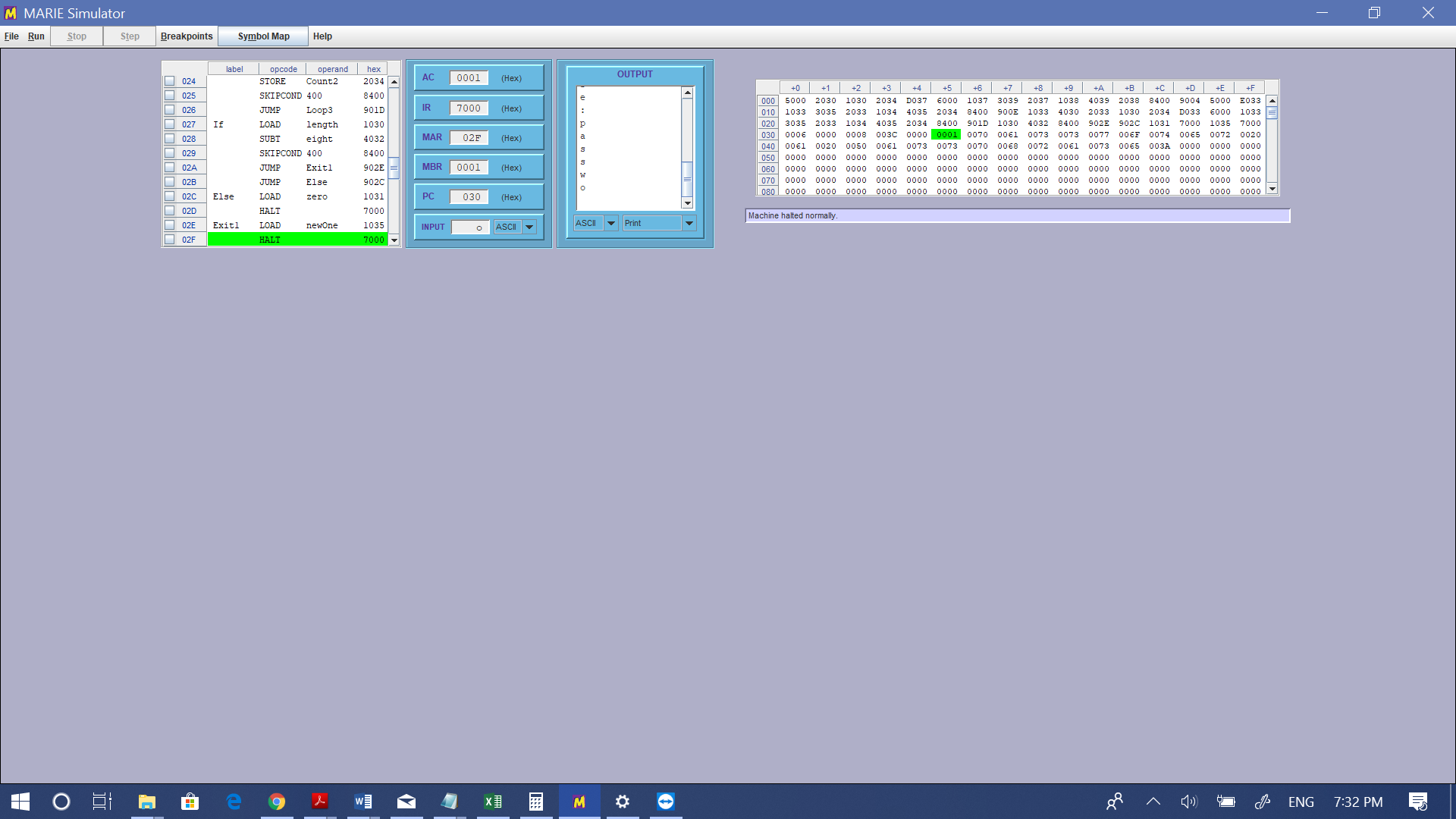


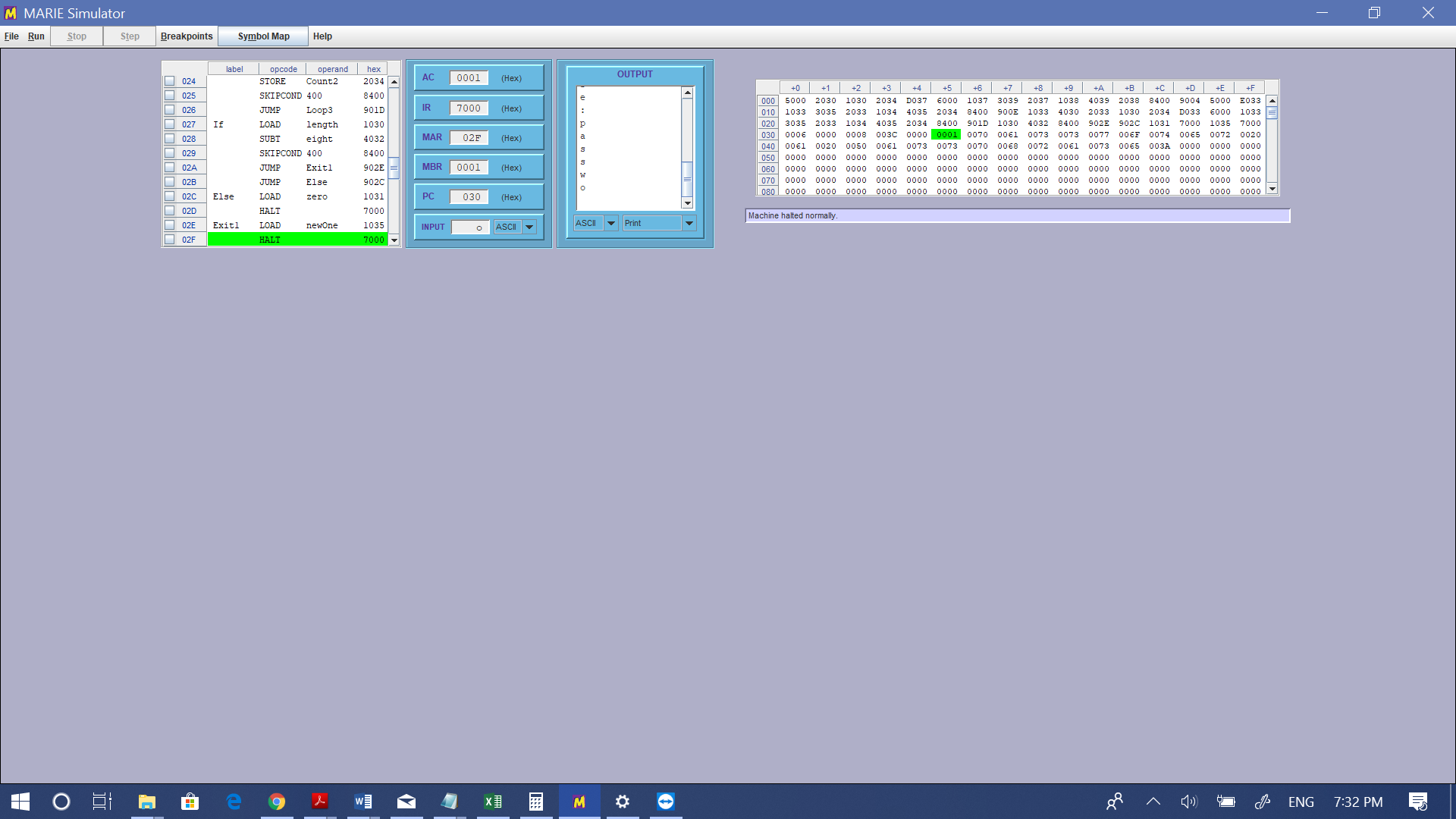
1. *Run #2: A word of length 6 is being attempted as a demonstration for returning 1*





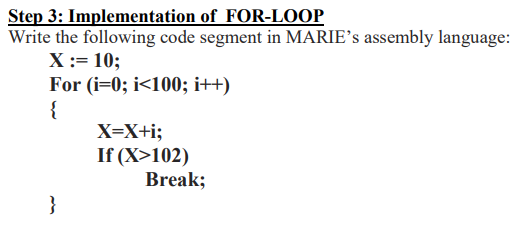
*Run #2: The phrase “Enter a Passphrase:” is printed*





*Run #2: The word “passwo” has a length of 6 characters. It does not equal to X, therefore, the program will return 1 at the respective address.*

**Step 3: Implementation of FOR-LOOP**



**Program Explanation:**

In short (2-3 lines),

In this program, a variable X is declared and iterated within a loop. The statement within the loop increments X by the index of each loop iteration. In addition, a condition checks whether X is bigger than 102. If that condition is met, the loop breaks. In this case, the loop never goes through all its iterations and is instead broken by the condition mentioned above.

In long (detailed),

This program starts with the loop at address 000. First, it loads the value ten (10) into the accumulator and stores this value in an address ‘X’. Then, the contents of address ‘Summer’ are added to X before ‘X’ being stored back into the accumulator. This process fulfills the requirements of the ‘X = X + i’ operation, where ‘i’ is the incrementation of the loop in high-level programming languages. The ‘Summer’ is loaded into the accumulator and ‘One’ is added to it, which is a constant of one (1), and the new contents of ‘Summer’ are stored back. This process occurs again for the address of ‘Count’ but ‘One’ is subtracting instead of adding. The address ‘Count’ is the iterator for the loop.

Within the loop, there is a condition that checks whether the contents of ‘X’ have reached a certain condition. Once this condition is met, the loop breaks and the program jumps at the label ‘End’, where the program is halted. This condition checks whether the contents of address ‘X’ have exceeded 102 or not. To check this, address ‘Cond’ (102) is subtracted from address ‘X’. Then ‘Skipcond 000’ is implemented so that it checks whether contents of ‘X’ are now negative (less than 0). If negative, it indicates that ‘X’ was not greater than 102 in the particular iteration of the loop. If positive, it indicates that ‘X’ was greater than 102 in the particular iteration of the loop. If positive, the step jumps to the ‘End’ label and program is halted. If negative, the step continues to the next condition.

The other condition controls the loop. It checks whether count has reached to zero (0). If it has not, the next step jumps at the beginning of the loop again. If it has reached zero (0), the loop ends and the program is halted.

**Source Code:**

/ Program made by George Mavroeidis and Johanson Felix

/ Use marie.js.org, make sure ‘Output Mode’ is set to ‘DEC’.

/ For loop that sums index and variable until a condition has been met

/ Respective columns (left to right):

/ Label | Opcode or Directive | Operand or value |

/ Store our program starting from memory location x000. Not necessary to include this. Default is 000.

ORG 000

/ Load variable X and add index from every iteration

Loop, Load X

Add Summer

Store X

/ Load the summer and increment it by one for every iteration

/ This simulates the for loop incrementation

Load Summer

Add One

Store Summer

/ Substract counter by 1, which is the actual loop incrementation

Load Count

Subt One

Store Count

/ This is the if statement, where it checks whether X is bigger than 120

/ To check this, we substract 120 from X to check whether it is positive or negative

Load X

Subt Cond

/ If the substraction is negative, it means X is smaller

/ So if X is not smaller than 102, the step jumps to the 'End" label

SkipCond 000

Jump End

/ If the 'Count' reaches 0, it means loop is finished iterating

/ If not, the step iterates the loop once again

/ Note: The loop will never finish but it will break until the above condition is met

Skipcond 400

Jump Loop

/ This was a testing case to check if the program runs correctly

/ By setting the End label here, we check the last result calculated for X,

/ which is 115. By simulating this in C language, X results indeed at 115

/ Load X

/ Output

End, Halt

/ X variable initialized to 10

X, DEC 10

/ Counter initialized to 100, which is the end of the loop at C language

Count, DEC 100

/ Variable that is added at X and incremented at every loop

Summer, DEC 0

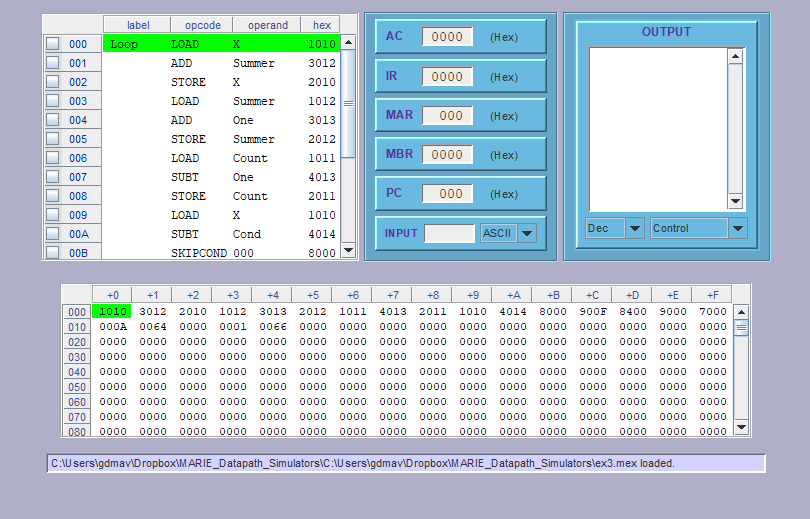
/ Incrementation by 1

One, DEC 1

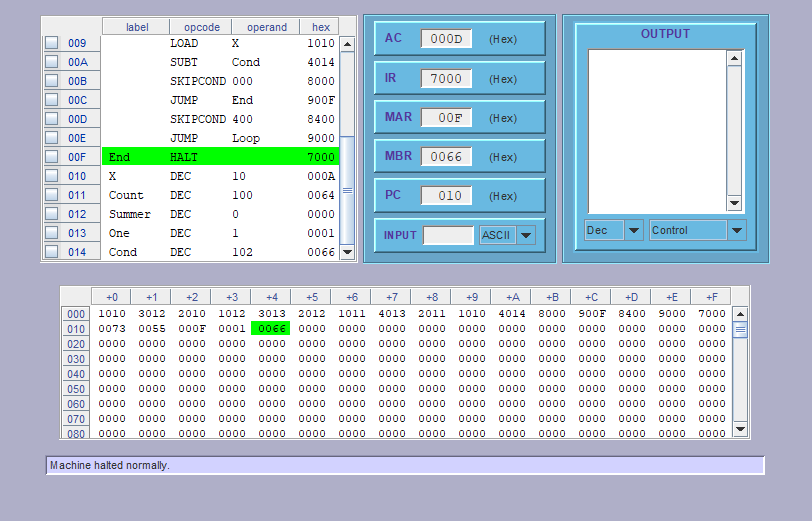
/ Variable that is used to check whether X is bigger or smaller than 102 in its current state

Cond, DEC 102

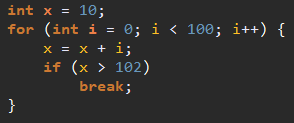
**Snapshots**



1. *Registers and Contents of Memory before program runs*



1. *After the program ends, the final step being halt, The IR has 7000 (halt) and the AC contains the last loaded variable, being X. X finishes with a variable of 115 in DECIMAL*



1. *This is the pseudocode of the program in Java programming language. The iteration has a limit of 100, which is the initial value of variable count in Assembly. In addition, index “i” serves as the index of the loop (variables count and summer in Assembly) and the if statement checks whether X is bigger than 102 (variable cond in Assembly).*

# **References**

|  |  |
| --- | --- |
| [1] | L. Null and J. Lobur, The Essentials of Computer Organization and Architecture, -: Jones and Bartlett Publishers, 2014. |
| [2] | L. Null and J. Lobur, " MarieSim: The MARIE computer," pp. 2, 3, 2003. |