Computer Architecture CP40079E – Assignment 2

1.

The von Neumann architecture is a computer architecture that shows the function of the basic components of a computer. The basic components are: the Memory or RAM, the central processing unit or CPU, which contains an arithmetic logic unit (ALU) for calculating mathematical sums such as add or subtract, the input and output functions and the BUS.

The memory is where the data and instructions are stored and the central processing unit is where all the instructions and mathematical calculations are carried out. It works with the memory to store instructions and data.

The BUS connects these components to the main circuit board of the computer known as the motherboard. It allows data to be transmitted from these components to other components of the computer which are mainly on the motherboard.

The CPU has the control unit and the arithmetic logic unit (ALU) which used for calculating arithmetical sums if the instruction is a sum such as add or subtract. The control unit has the instruction register which shows the instruction that is being carried out - fetched from memory, decoded and then executed and a program counter which displays the memory address of the next instruction that is going to be carried out. This is known as the fetch-execute cycle:

- Fetch - instructions are taken out of the memory and stored in the instruction register
- Decode – the CPU checks what the instruction has to do before execution
- Execute – the instruction is then carried out and the program counter then shows the location of the next instruction to be executed to begin the cycle again

The input function allows the end user to type in a value such as a number or a letter followed by the output function displaying a value to the user.

Instructions are run in a sequence unless an instruction runs it out of sequence i.e. jump to another part of the program and continue from there. This is called branching.
They are also run out of sequence if the user clicks the reset button to reset the program or the instruction location counter.

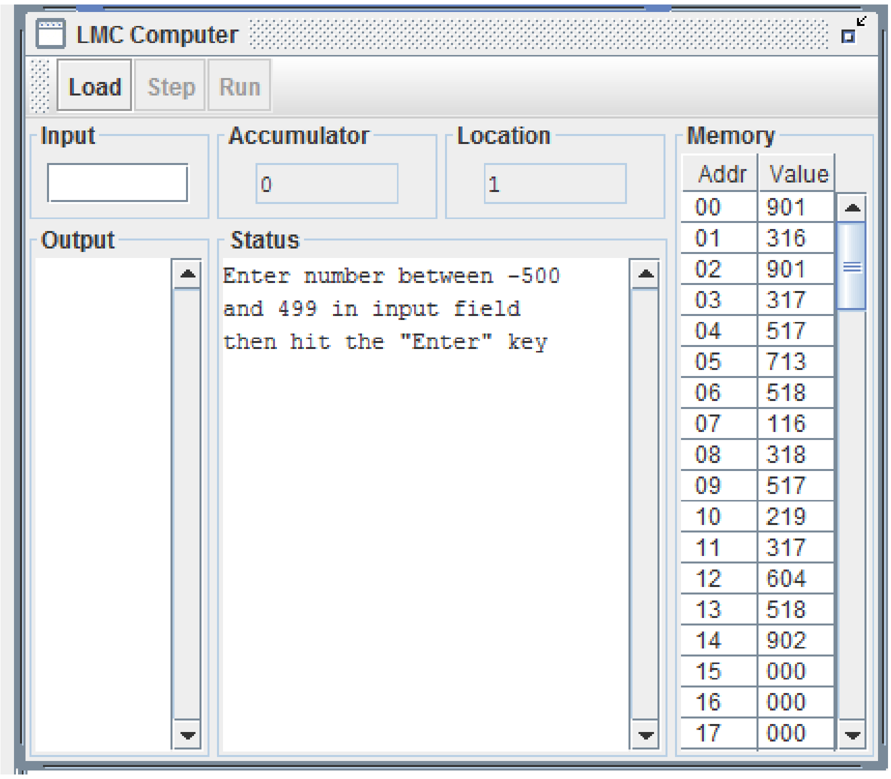
The von Neumann architecture is considered to be the architecture that is followed for all computers and devices that use these basic components of a computer since it was created. Although other computer architectures have been developed, they have not become as successful as this architecture to be followed for computers. The von Neumann is the standard computer architecture to this day.

The guidelines of it are:

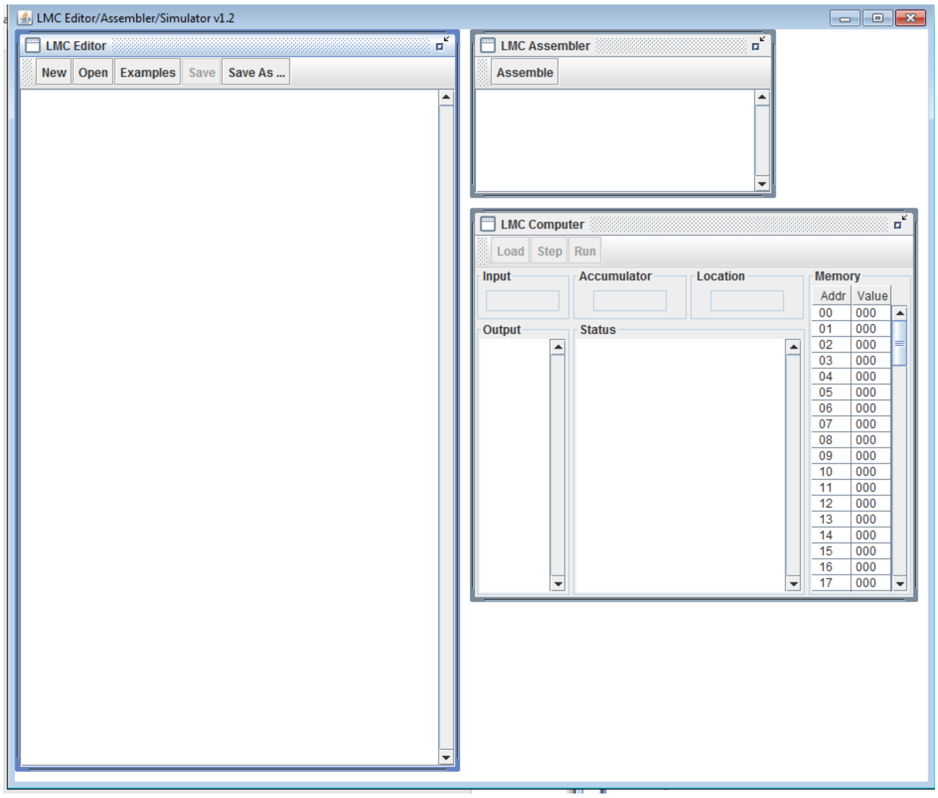
- Data and programs are held in Memory. This is the stored program concept that makes it is easy to change programs.
- The memory has locations. Each and every memory location is addressed in numerical order. The address of each location where they are stored is shown by the program counter as mentioned above. Instructions can be stored in the addresses in this order. This order is a known as a sequence.
- Each location is addressed by a number, whatever the data being held in the location.

A simulator called the Little Man Computer simulates the function of the basic components in a computer shown by the von Neumann architecture. It is a model of the architecture which meets all its guidelines and a useful tool to understand how it works.

2.



The first two programs on the little man computer simulator I'm going to run are multiply and divide. The little man computer simulator shown above shows a calculator (or accumulator) which displays values and answers to sums, a textbox for input of values, a display window for the output of values and the Memory showing a list of addresses going all the way from 00 to 99, with a mailbox next to each address where the instructions are stored.



The simulator allows the user to enter instructions before running the program such as ADD(Add two numbers), LDA (Load the data from the memory) BR or BRA and BRZ (Branch or jump to another instruction) and HLT (End the program). The user runs the program by clicking the run button after entering these instructions which is what I will be doing for the rest of this assignment.

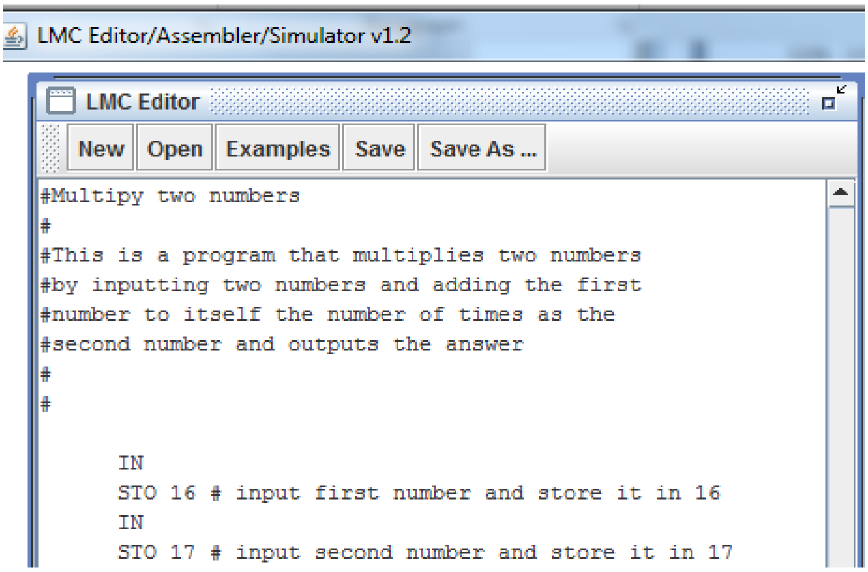
```
IN
STO 16
IN
STO 17
LDA 17
BRZ 13
LDA 18
ADD 16
STO 18
LDA 17
SUB 19
STO 17
BR  04
LDA 18
OUT
HLT
DAT 000
DAT 000
DAT 000
DAT 001
```

In the screenshot above is the available instruction set or instruction set architecture (ISA) that will be used to write the program to multiply two numbers and divide two numbers. These are the instructions that tell the processor what to do with the input command IN and the output command OUT. Instructions in the set such as store, load and branch have memory addresses to go to for e.g. STO 17 stores the inputted number in address 17, LDA 18 loads the value from address 18 onto the accumulator and BR 04 jumps to the instruction at address 4.

Memory

Addr	Value
00	901
01	316
02	901
03	317
04	517
05	713
06	518
07	116
08	318
09	517
10	219
11	317
12	604
13	518
14	902
15	000
16	000
17	000

In the memory are the set of instructions loaded into it in three digits for example or e.g. the instruction 518. This instruction is made of the operation code (OP code) e.g. 5 for load and the operand which is the address where the data is to be loaded is stored e.g. 18.



Multiplication of two numbers starts by first inputting the two numbers to be multiplied.

LMC Editor/Assembler/Simulator v1.2

LMC Editor

New | Open | Examples | Save | Save As ...

```
#Multipy two numbers
#
#This is a program that multiplies two numbers
#by inputting two numbers and adding the first
#number to itself the number of times as the
#second number and outputs the answer
#
#


      IN
      STO 16 # input first number and store it in 16
      IN
      STO 17 # input second number and store it in 17
      LDA 17 # load second number on the accumulator
      BRZ 13 # branch if second number is zero to 13
      LDA 18 # load data stored in 18
      ADD 16 # add  data stored in 18 to first number
```

They are then stored in two different addresses and loaded onto the accumulator while the

Sum is being carried out.

```
      IN
      STO 16 # input first number and store it in 16
      IN
      STO 17 # input second number and store it in 17
      LDA 17 # load second number on the accumulator
      BRZ 13 # branch if second number is zero to 13
      LDA 18 # load data stored in 18
      ADD 16 # add  data stored in 18 to first number
      STO 18 # and store it in 18
      LDA 17
      SUB 19 # subtract 1 from second number
      STO 17
      BR  04 # branch to the top of the loop LDA 17
      LDA 18
      OUT    # output the answer to multiplication
      HLT    # and finally end the program
      DAT 000
      DAT 000 # Data locations 16 - 18 initially zero
      DAT 000
      DAT 001 # one or 1 is a constant in this location
```

The program runs all the loading, adding, subtract instructions and keeps branching to the top of the loop until the second number becomes zero where the program will branch to the end.

LMC Editor/Assembler/Simulator v1.2

LMC Editor

New | Open | Examples | Save | Save As ...

```
#Multipy two numbers
#
#This is a program that multiplies two numbers
#by inputting two numbers and adding the first
#number to itself the number of times as the
#second number and outputs the answer
#
#This version uses mnemonics

      IN
      STO 16 # input first number and store it in 16
      IN
      STO 17 # input second number and store it in 17
      LDA 17 # load second number on the accumulator
      BRZ 13 # branch if second number is zero to 13
      LDA 18 # load data stored in 18
      ADD 16 # add  data stored in 18 to first number
      STO 18 # and store it in 18
      LDA 17
      SUB 19 # subtract 1 from second number
      STO 17
      BR  04 # branch to the top of the loop LDA 17
      LDA 18
      OUT    # output the answer to multiplication
      HLT    # and finally end the program
      DAT 000
      DAT 000 # Data locations 16 - 18 initially zero
      DAT 000
      DAT 001 # one or 1 is a constant in this location
```

You can see here all the data storage locations known as DAT which are reserved for

Storage of data.

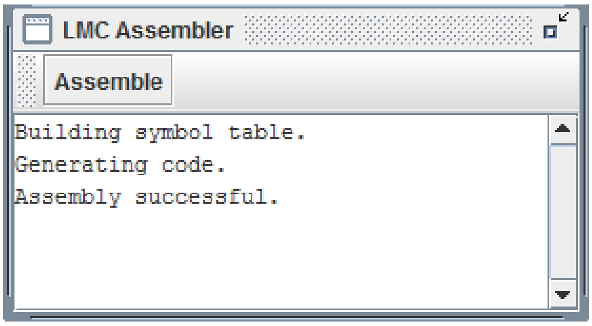
LMC Editor/Assembler/Simulator v1.2

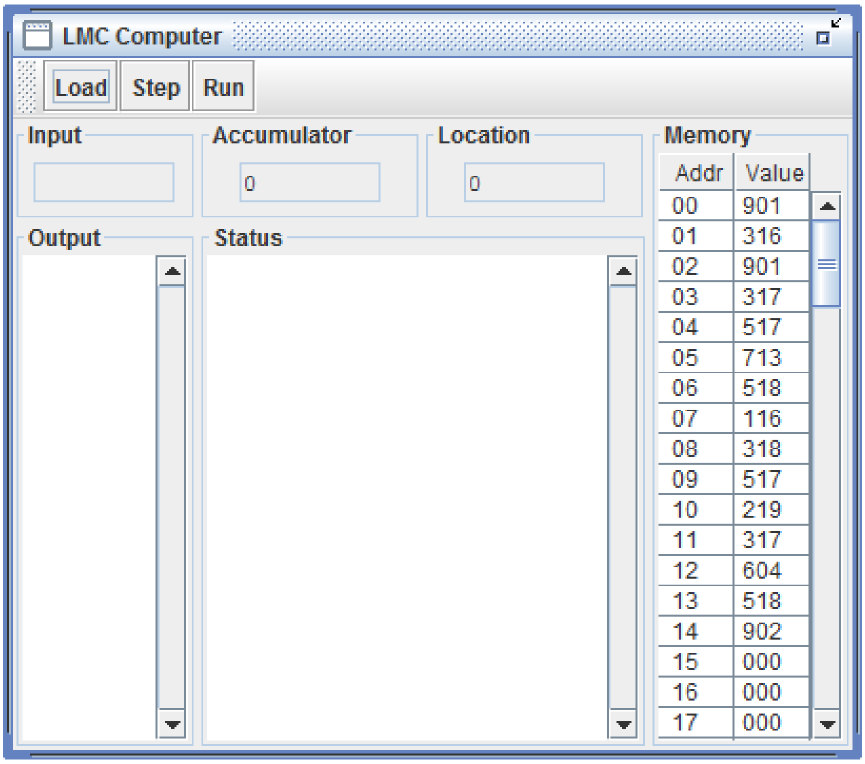
LMC Editor

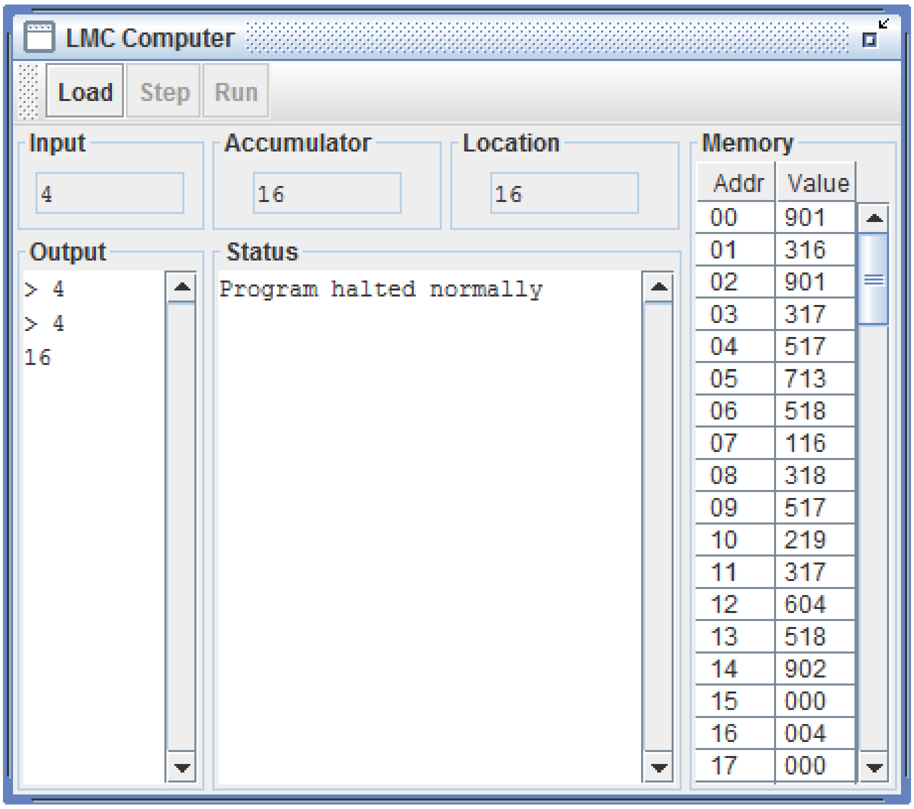
New | Open | Examples | Save | Save As ...

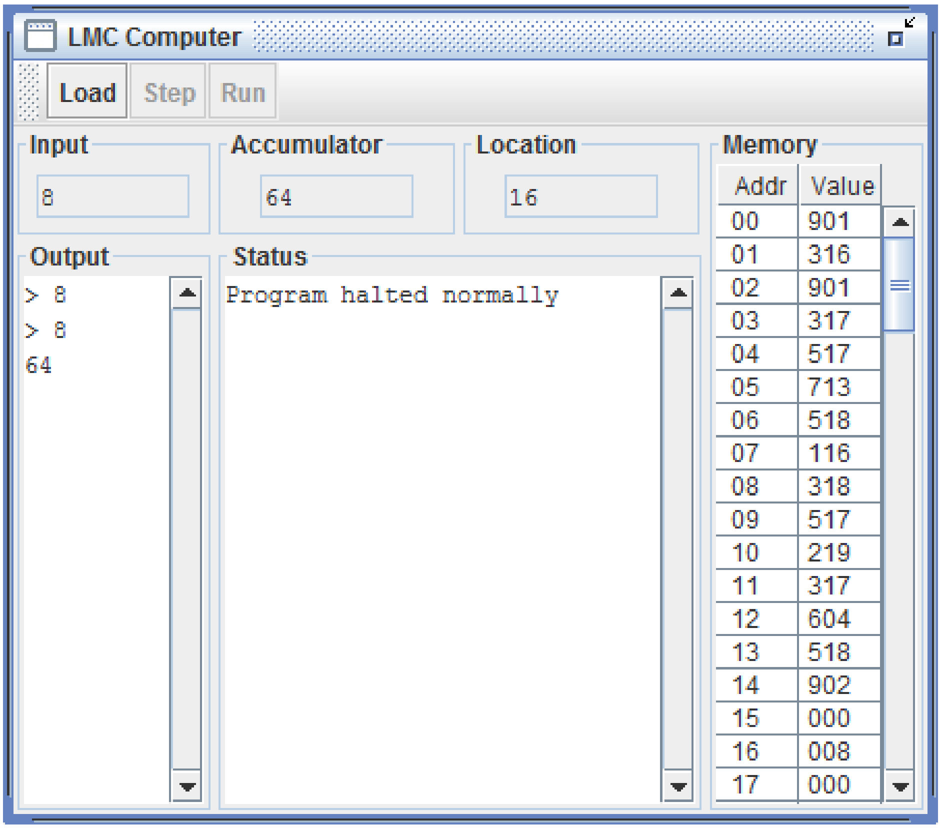
```
#Divide two no.s
#
#This program divides two numbers by subtracting the second
#number drom the first until it reaches zero and the answer
#to the sum is output
#
#
        IN
        STO 15 #input the first number and store it in 15
        IN
        STO 16 #input the second number and store it in 16
        LDA 15 #load the first number onto the accumulator
        BRZ 12 #branch if zero to 12 - LDA 18
        SUB 16
        STO 15
```

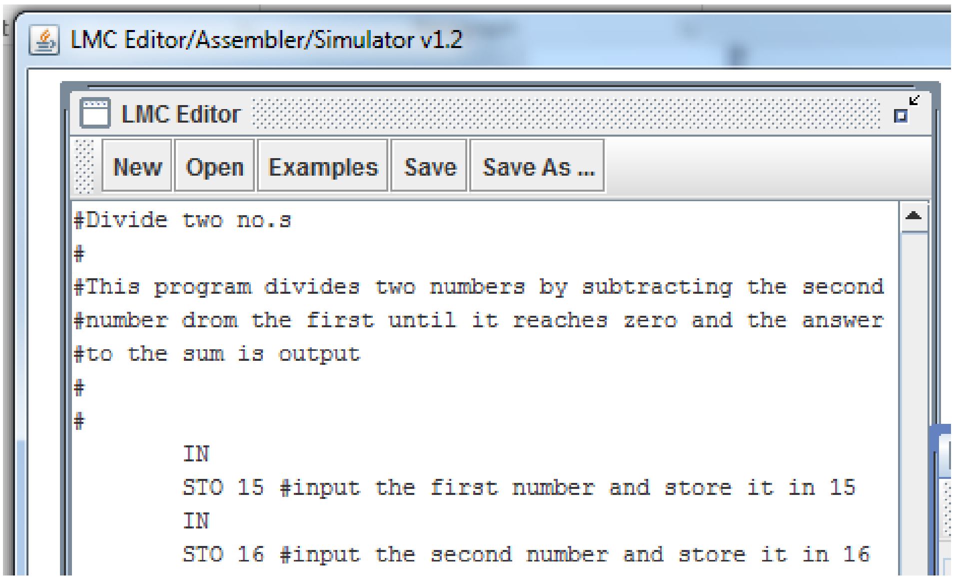
```
        IN
        STO 15 #input the first number and store it in 15
        IN
        STO 16 #input the second number and store it in 16
        LDA 15 #load the first number onto the accumulator
        BRZ 12 #branch if zero to 12 - LDA 18
        SUB 16
        STO 15
        LDA 18
        ADD 17
        STO 18 #add one to and store it in data location 18
        BR  04 #branch to the top of the loop LDA 15
        LDA 18
        OUT    #output the sum
        HLT
        DAT 000
        DAT 000
        DAT 001 # 1 or one as a constant
        DAT 000 # Data location 18 has zero initially
```

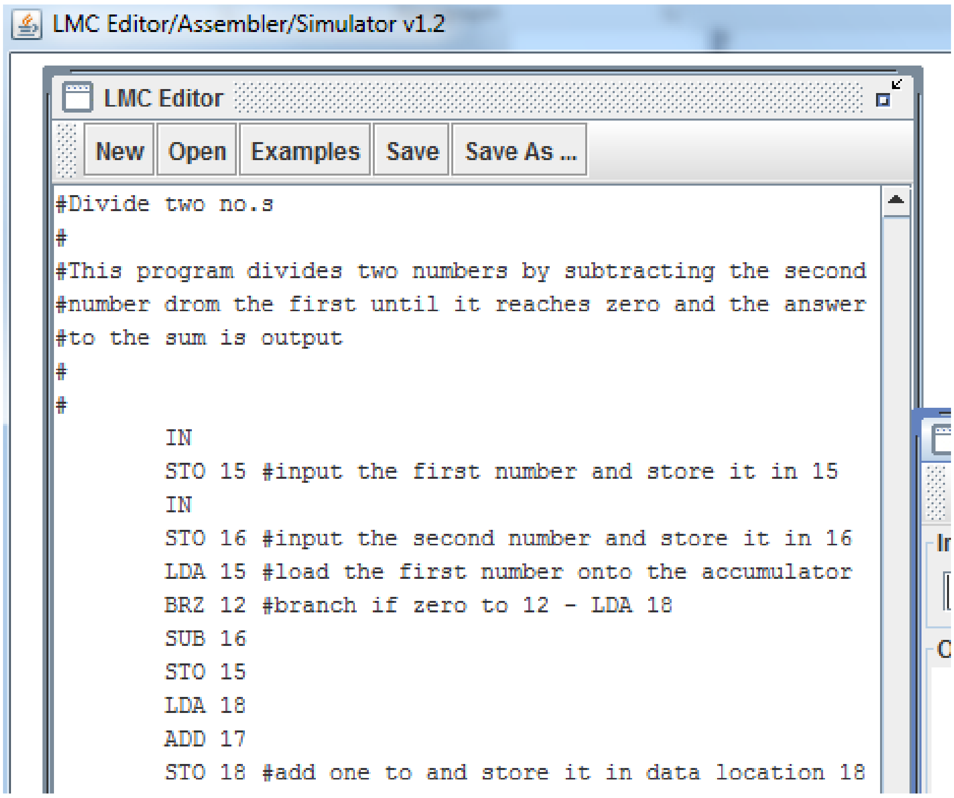

LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.

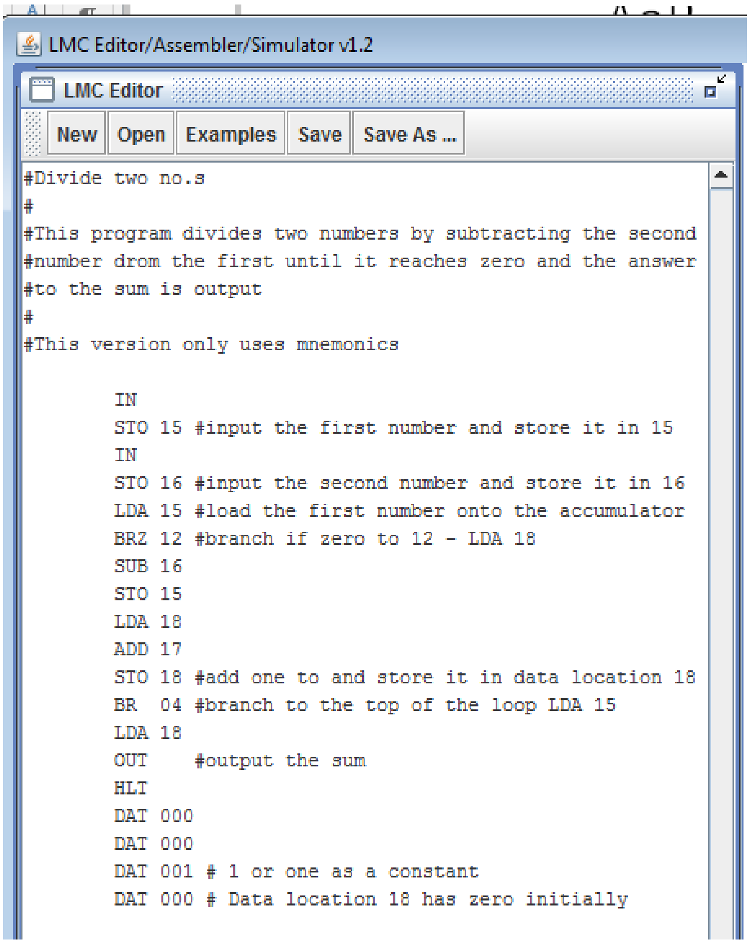

LMC Computer
Load
Step
Run
Input
Accumulator
0
Location
0
Memory
Addr Value
00 901
01 316
02 901
03 317
04 517
05 713
06 518
07 116
08 318
09 517
10 219
11 317
12 604
13 518
14 902
15 000
16 000
17 000
Output
Status

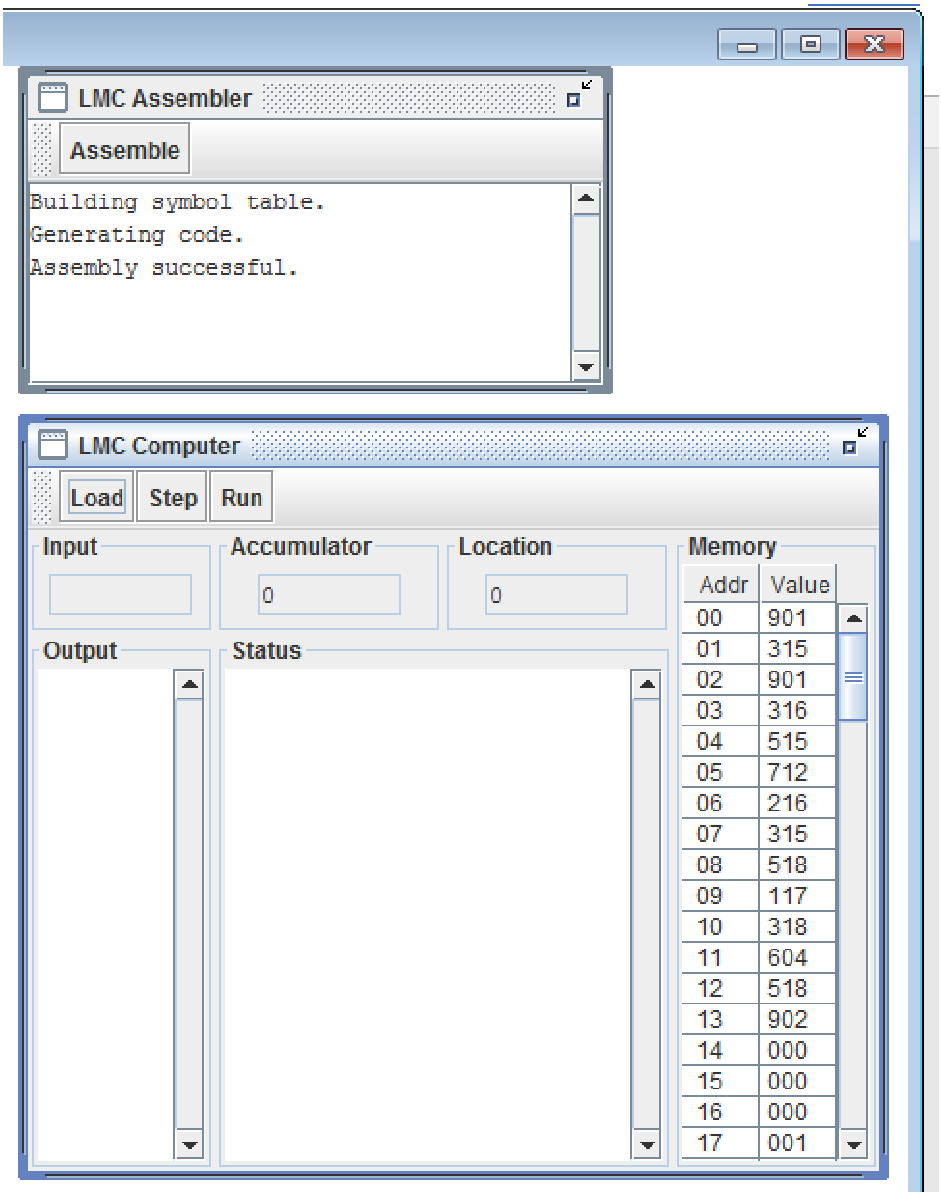

LMC Computer
Load
Step
Run
Input
4
Accumulator
16
Location
16
Memory
Addr Value
00 901
01 316
02 901
03 317
04 517
05 713
06 518
07 116
08 318
09 517
10 219
11 317
12 604
13 518
14 902
15 000
16 004
17 000
Output
> 4
> 4
16
Status
Program halted normally

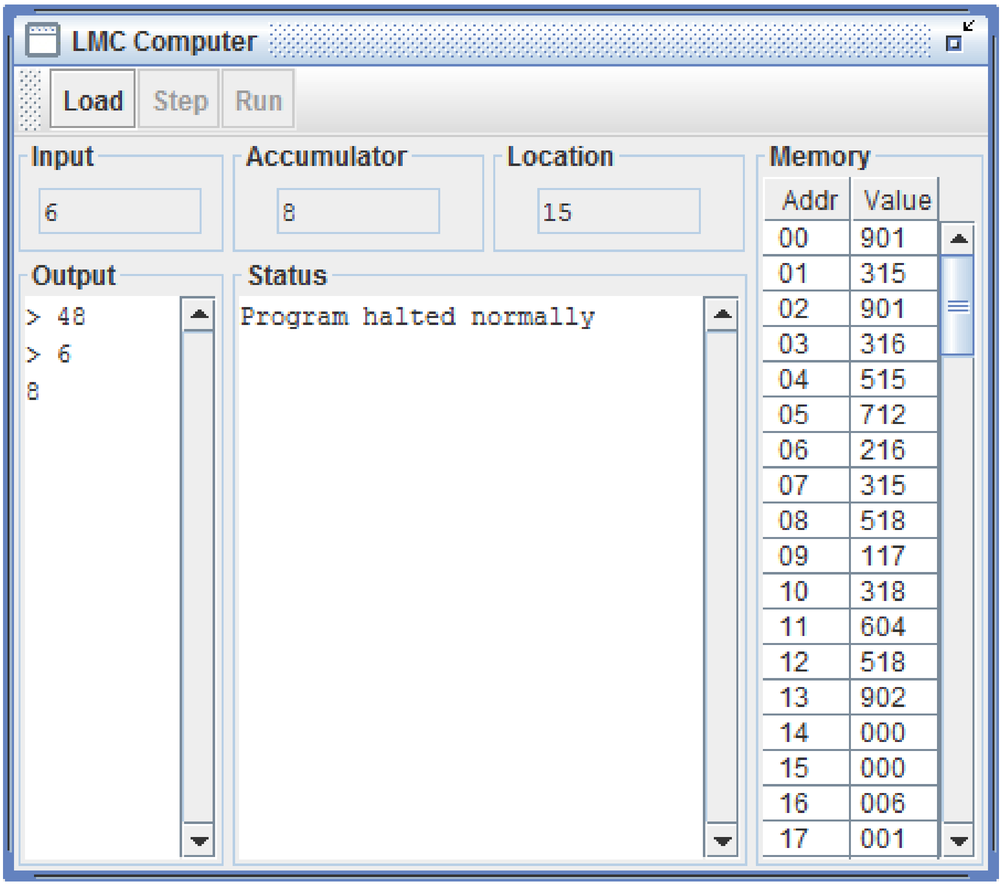

LMC Computer
Load
Step
Run
Input
8
Accumulator
64
Location
16
Memory
Addr Value
00 901
01 316
02 901
03 317
04 517
05 713
06 518
07 116
08 318
09 517
10 219
11 317
12 604
13 518
14 902
15 000
16 008
17 000
Output
> 8
> 8
64
Status
Program halted normally

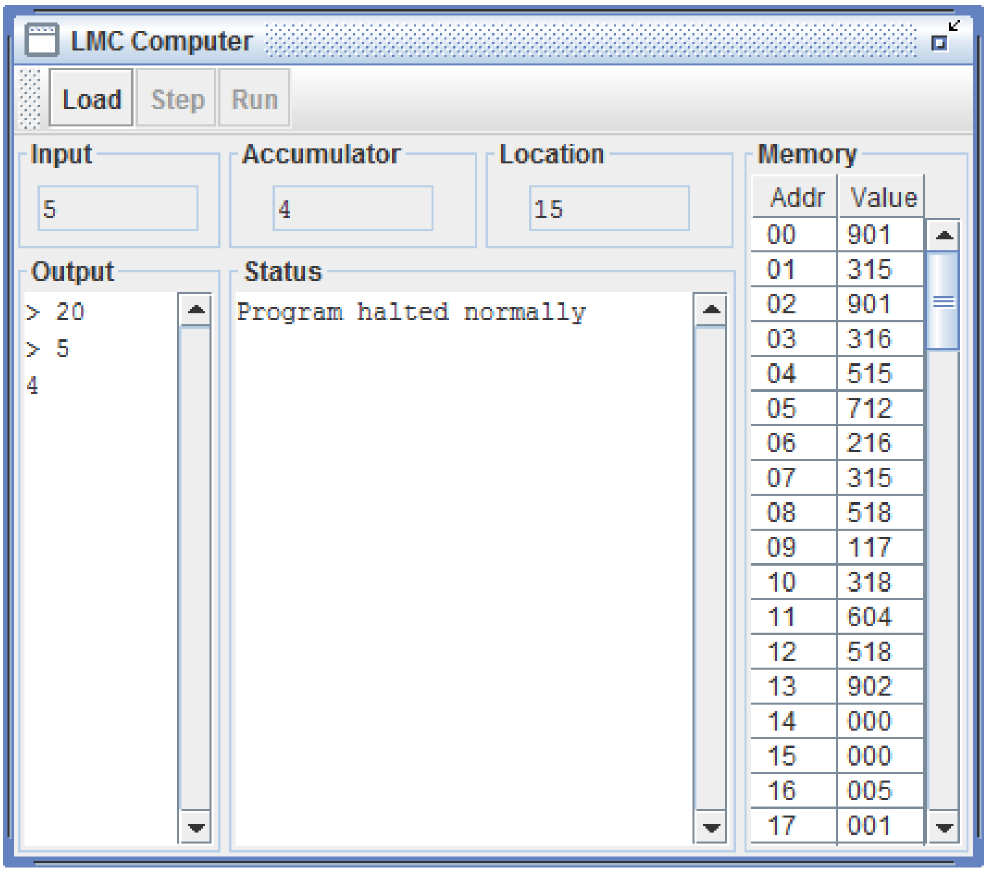

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Divide two no.s
#
#This program divides two numbers by subtracting the second
#number drom the first until it reaches zero and the answer
#to the sum is output
#
#
IN
STO 15 #input the first number and store it in 15
IN
STO 16 #input the second number and store it in 16


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Divide two no.s
#
#This program divides two numbers by subtracting the second
#number drom the first until it reaches zero and the answer
#to the sum is output
#
#
IN
STO 15 #input the first number and store it in 15
IN
STO 16 #input the second number and store it in 16
LDA 15 #load the first number onto the accumulator
BRZ 12 #branch if zero to 12 - LDA 18
SUB 16
STO 15
LDA 18
ADD 17
STO 18 #add one to and store it in data location 18


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Divide two no.s
#
#This program divides two numbers by subtracting the second
#number drom the first until it reaches zero and the answer
#to the sum is output
#
#This version only uses mnemonics
IN
STO 15 #input the first number and store it in 15
IN
STO 16 #input the second number and store it in 16
LDA 15 #load the first number onto the accumulator
BRZ 12 #branch if zero to 12 - LDA 18
SUB 16
STO 15
LDA 18
ADD 17
STO 18 #add one to and store it in data location 18
BR 04 #branch to the top of the loop LDA 15
LDA 18
OUT #output the sum
HLT
DAT 000
DAT 000
DAT 001 # 1 or one as a constant
DAT 000 # Data location 18 has zero initially

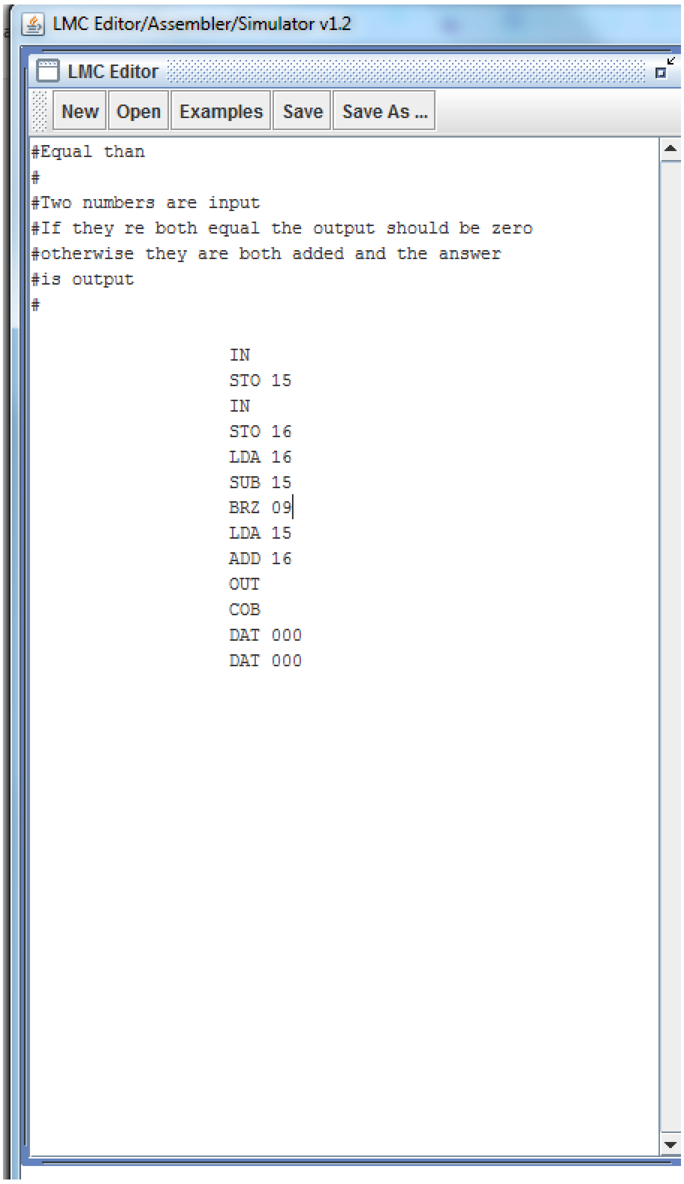

LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.
LMC Computer
Load
Step
Run
Input
Accumulator
0
Location
0
Memory
Addr Value
00 901
01 315
02 901
03 316
04 515
05 712
06 216
07 315
08 518
09 117
10 318
11 604
12 518
13 902
14 000
15 000
16 000
17 001
Output
Status


LMC Computer
Load
Step
Run
Input
6
Accumulator
8
Location
15
Memory
Addr Value
00 901
01 315
02 901
03 316
04 515
05 712
06 216
07 315
08 518
09 117
10 318
11 604
12 518
13 902
14 000
15 000
16 006
17 001
Output
> 48
> 6
8
Status
Program halted normally

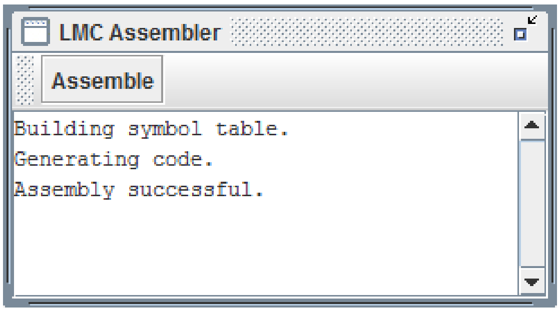

LMC Computer
Load
Step
Run
Input
5
Accumulator
4
Location
15
Memory
Addr Value
00 901
01 315
02 901
03 316
04 515
05 712
06 216
07 315
08 518
09 117
10 318
11 604
12 518
13 902
14 000
15 000
16 005
17 001
Output
> 20
> 5
4
Status
Program halted normally

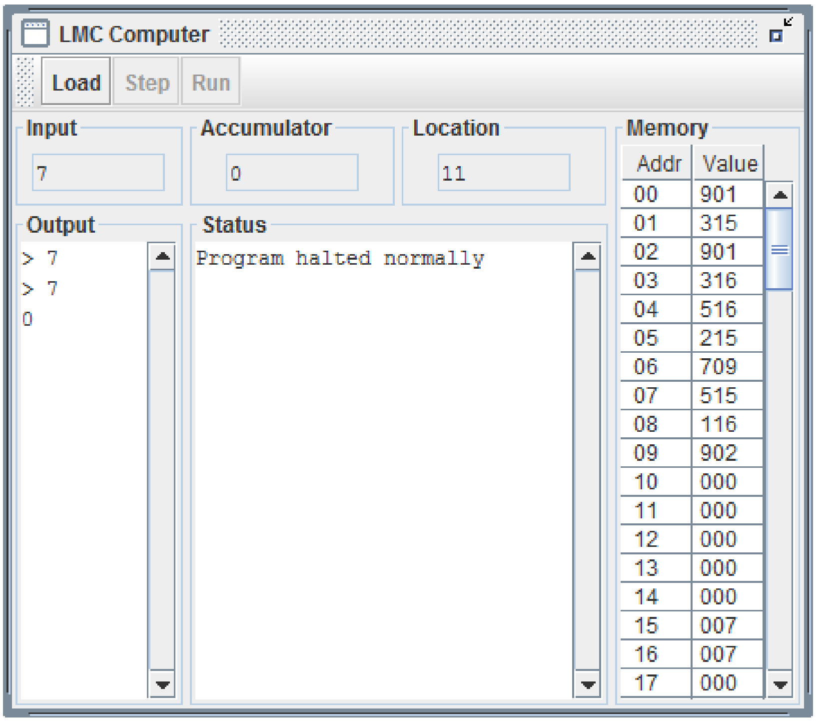
3.

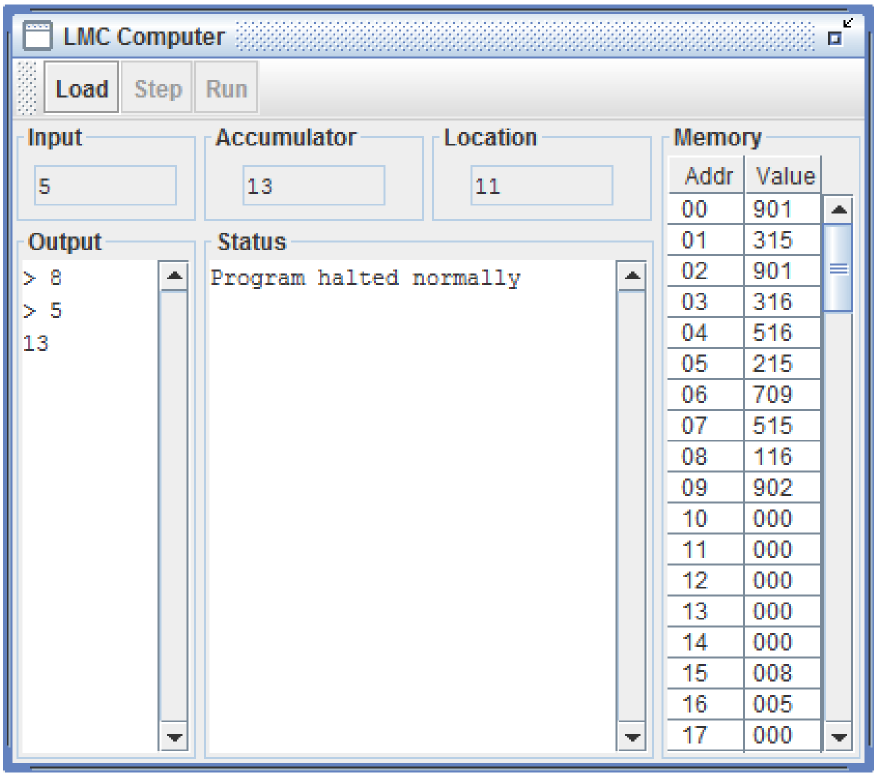
- Equal to


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Equal than
#
#Two numbers are input
#If they re both equal the output should be zero
#otherwise they are both added and the answer
#is output
#
IN
STO 15
IN
STO 16
LDA 16
SUB 15
BRZ 09
LDA 15
ADD 16
OUT
COB
DAT 000
DAT 000

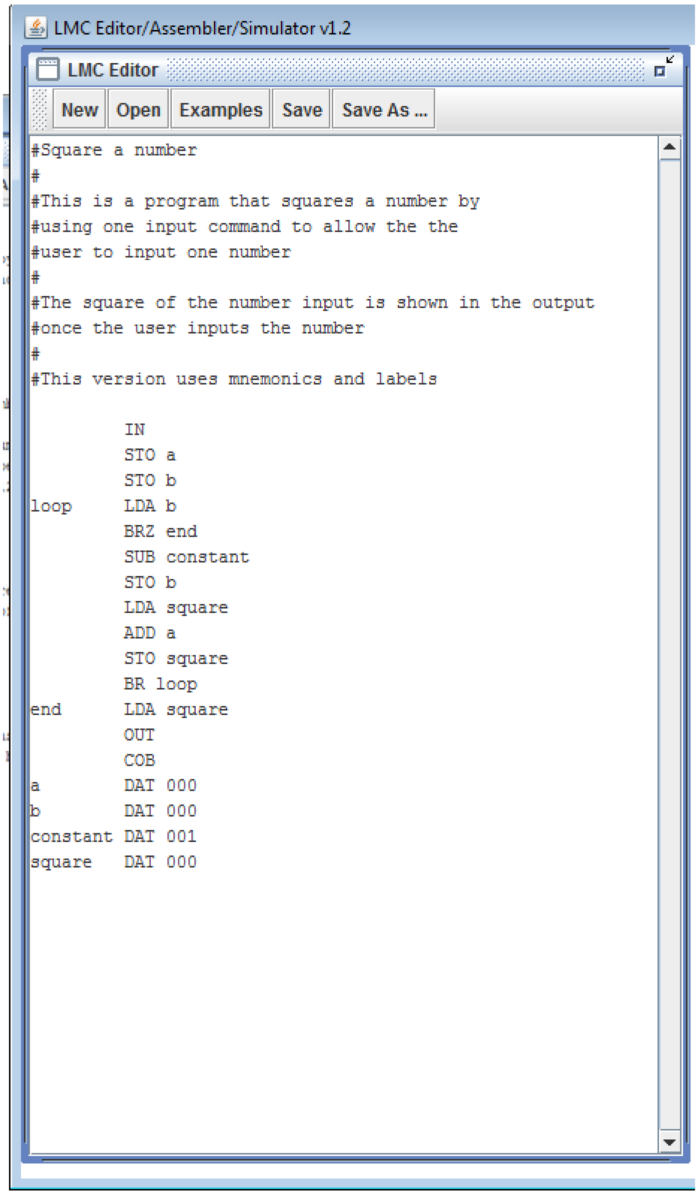
Student ID 21359035 [Type here] Date 03/19

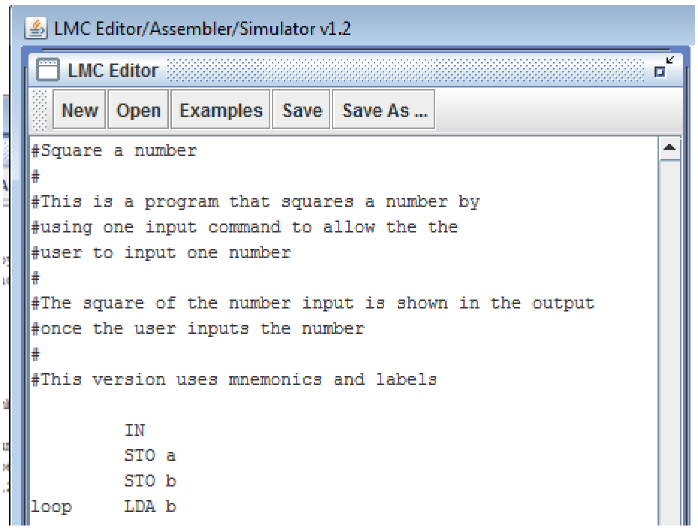


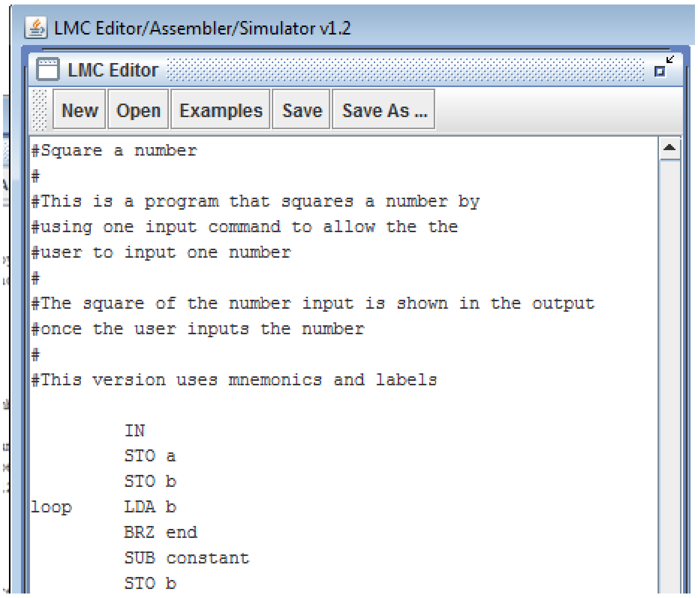


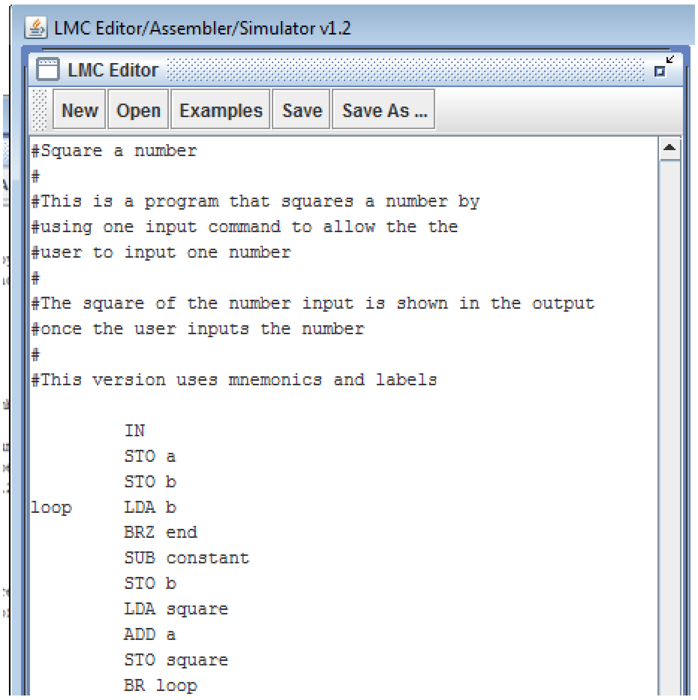


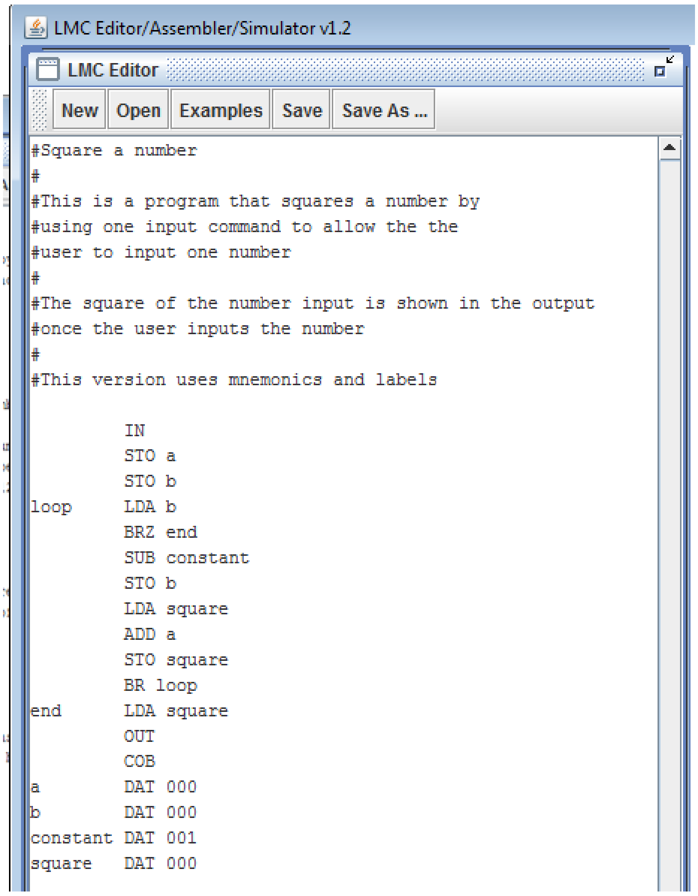
- Square


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New Open Examples Save Save As ...
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses mnemonics and labels
IN
STO a
STO b
loop LDA b
BRZ end
SUB constant
STO b
LDA square
ADD a
STO square
BR loop
end LDA square
OUT
COB
a DAT 000
b DAT 000
constant DAT 001
square DAT 000

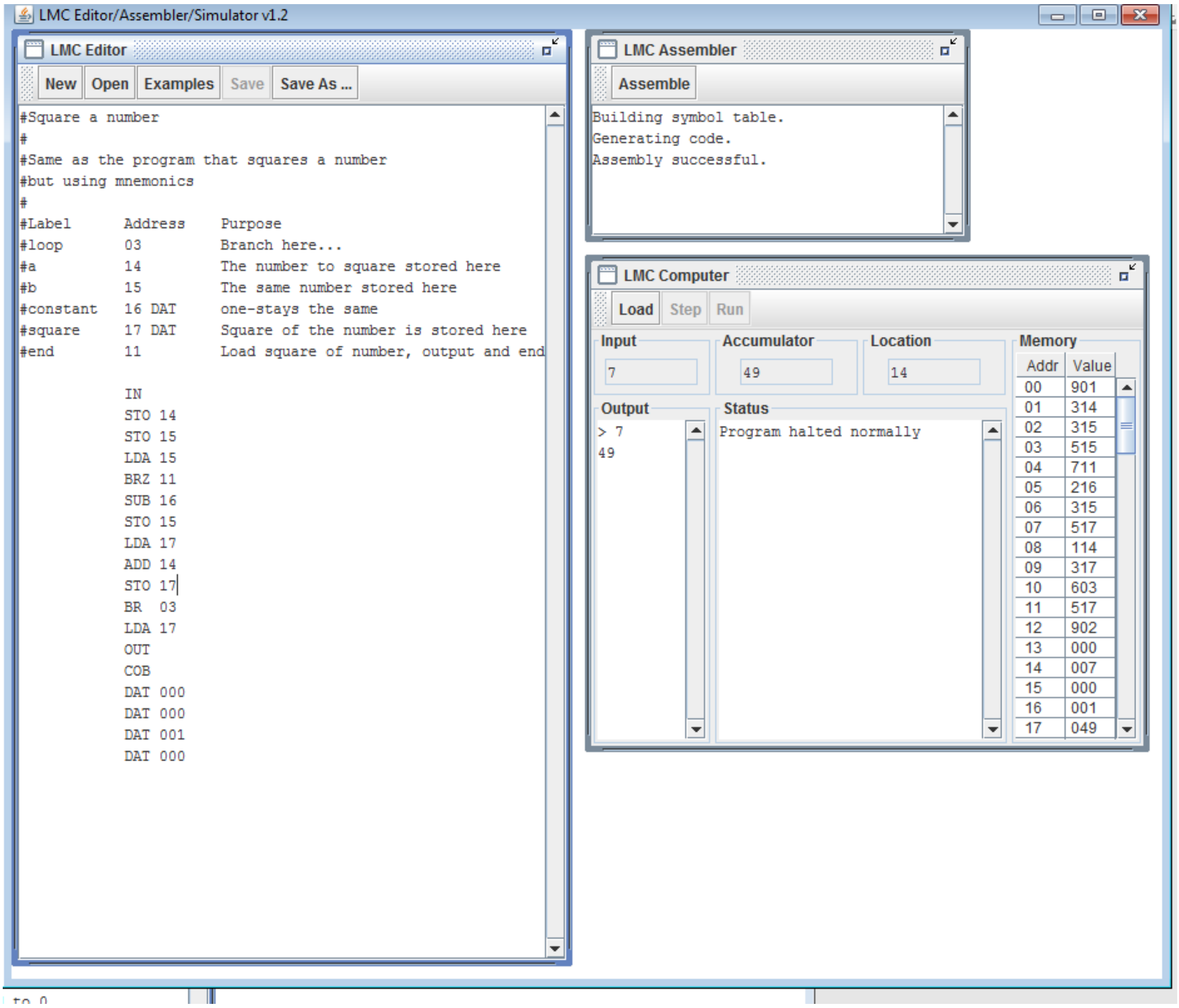

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New Open Examples Save Save As ...
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses mnemonics and labels
IN
STO a
STO b
loop LDA b

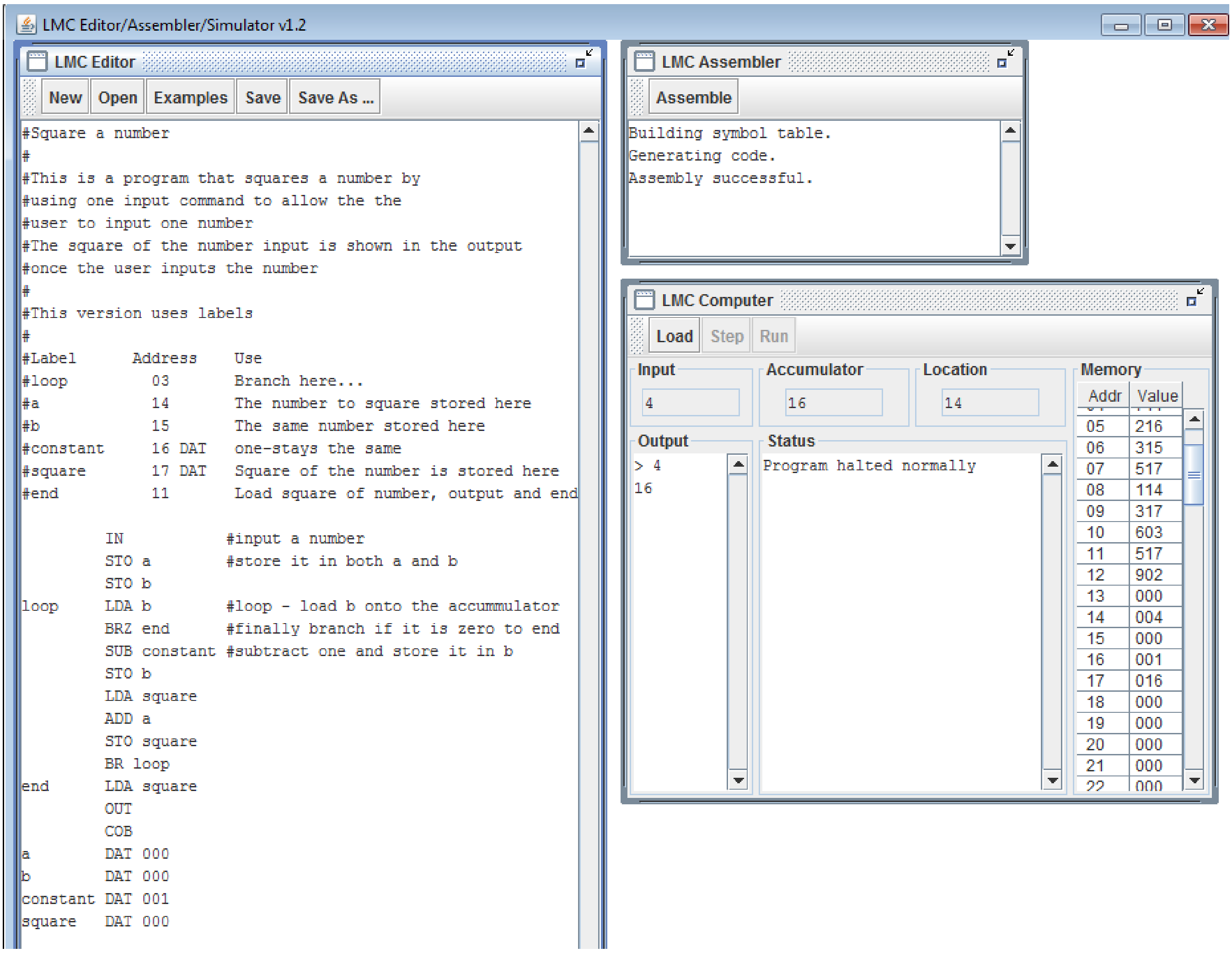

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses mnemonics and labels
IN
STO a
STO b
loop LDA b
BRZ end
SUB constant
STO b

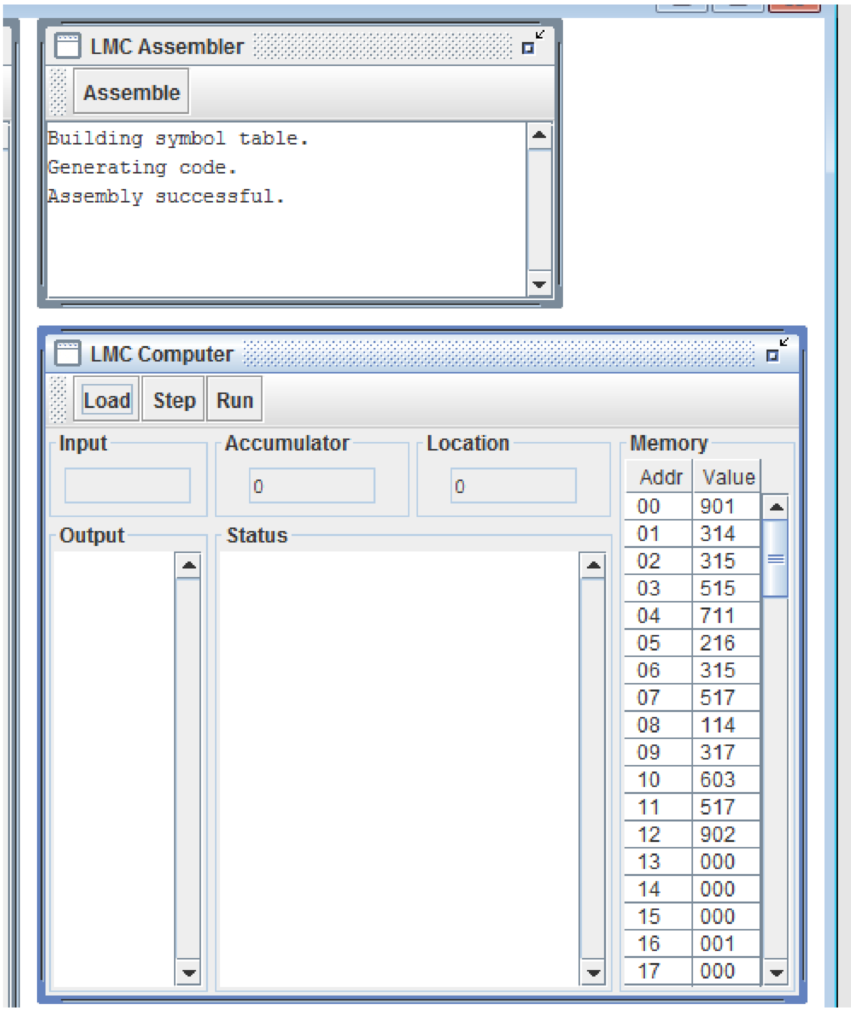

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses mnemonics and labels
IN
STO a
STO b
loop LDA b
BRZ end
SUB constant
STO b
LDA square
ADD a
STO square
BR loop

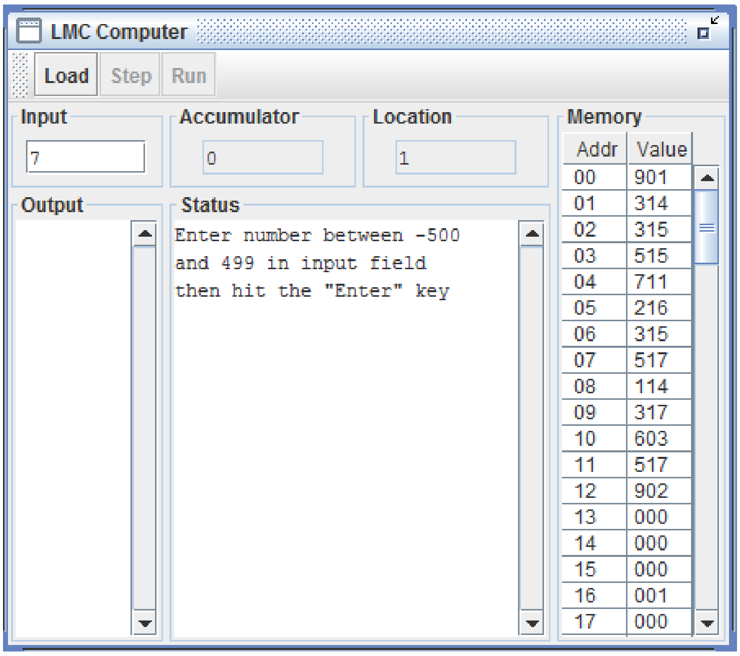


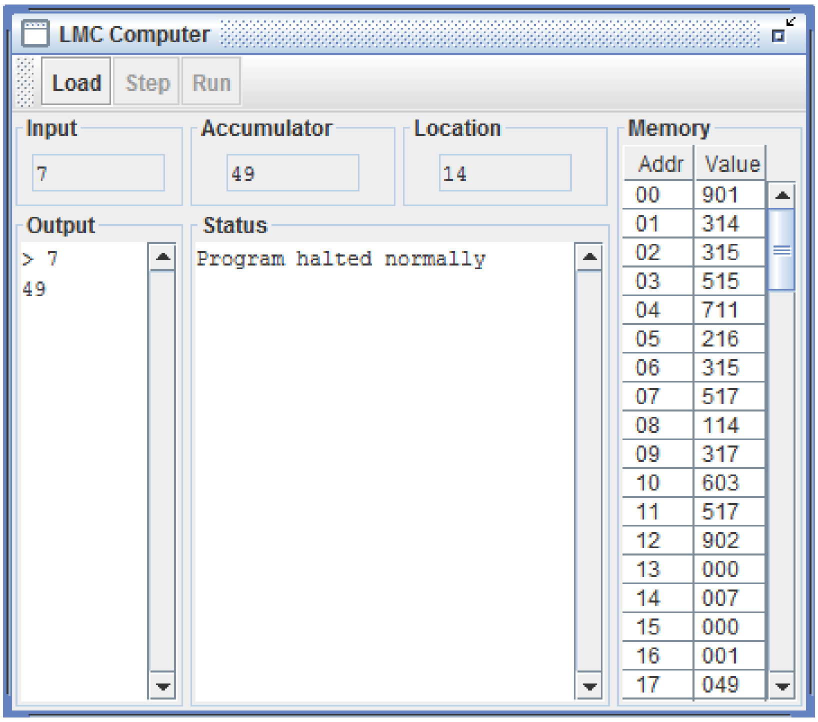

```
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses mnemonics and labels

         IN
         STO a
         STO b
loop     LDA b
         BRZ end
         SUB constant
         STO b
         LDA square
         ADD a
         STO square
         BR loop
end      LDA square
         OUT
         COB
a        DAT 000
b        DAT 000
constant DAT 001
square   DAT 000
```

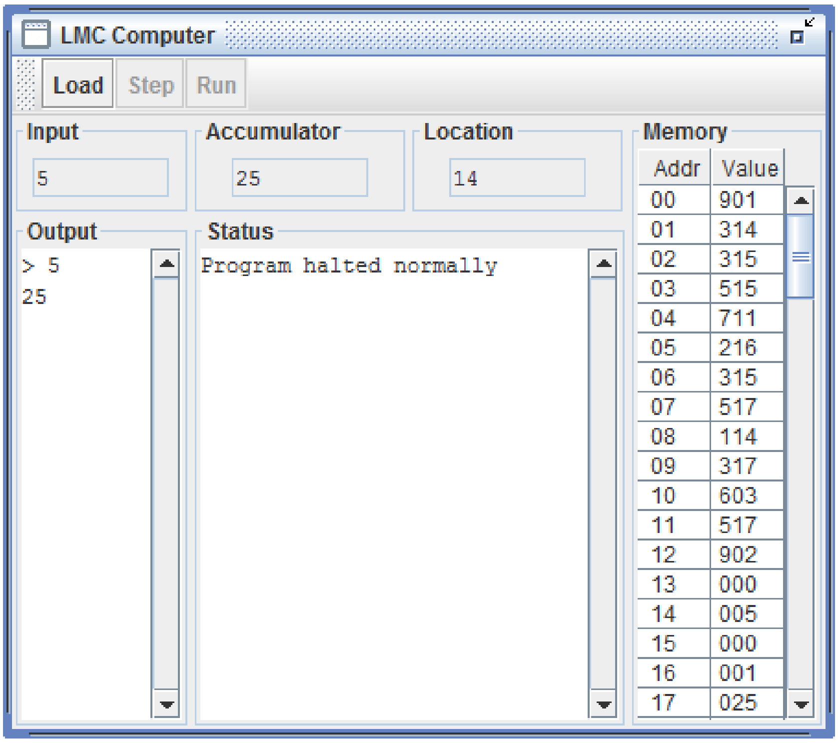

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Square a number
#
#Same as the program that squares a number
#but using mnemonics
#
#Label Address Purpose
#loop 03 Branch here...
#a 14 The number to square stored here
#b 15 The same number stored here
#constant 16 DAT one-stays the same
#square 17 DAT Square of the number is stored here
#end 11 Load square of number, output and end
IN
STO 14
STO 15
LDA 15
BRZ 11
SUB 16
STO 15
LDA 17
ADD 14
STO 17
BR 03
LDA 17
OUT
COB
DAT 000
DAT 000
DAT 001
DAT 000
LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.
LMC Computer
Load
Step
Run
Input
7
Accumulator
49
Location
14
Memory
Addr Value
00 901
01 314
02 315
03 515
04 711
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 007
15 000
16 001
17 049
Output
> 7
49
Status
Program halted normally


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New Open Examples Save Save As ...
#Square a number
#
#This is a program that squares a number by
#using one input command to allow the the
#user to input one number
#The square of the number input is shown in the output
#once the user inputs the number
#
#This version uses labels
#
#Label Address Use
#loop 03 Branch here...
#a 14 The number to square stored here
#b 15 The same number stored here
#constant 16 DAT one-stays the same
#square 17 DAT Square of the number is stored here
#end 11 Load square of number, output and end
IN #input a number
STO a #store it in both a and b
STO b
loop LDA b #loop - load b onto the accummulator
BRZ end #finally branch if it is zero to end
SUB constant #subtract one and store it in b
STO b
LDA square
ADD a
STO square
BR loop
end LDA square
OUT
COB
a DAT 000
b DAT 000
constant DAT 001
square DAT 000
LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.
LMC Computer
Load Step Run
Input
4
Accumulator
16
Location
14
Memory
Addr Value
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 004
15 000
16 001
17 016
18 000
19 000
20 000
21 000
Output
> 4
16
Status
Program halted normally

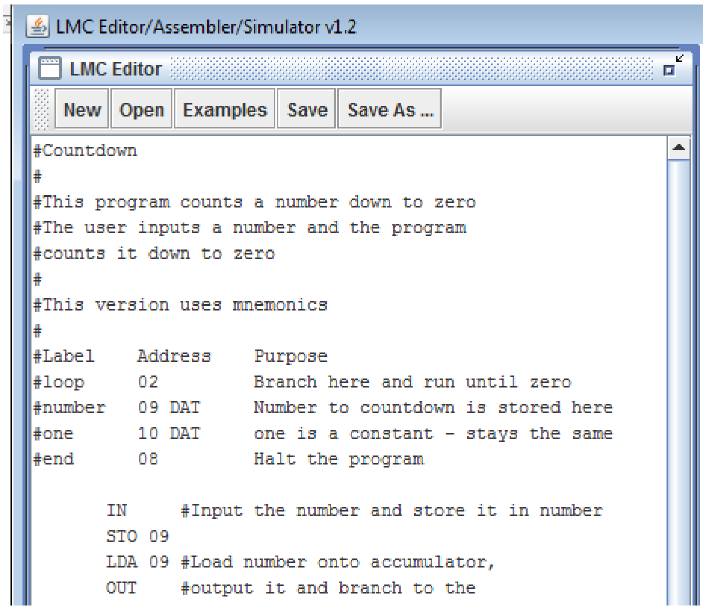

LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.
LMC Computer
Load
Step
Run
Input
Accumulator
0
Location
0
Memory
Addr
Value
00 901
01 314
02 315
03 515
04 711
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 000
15 000
16 001
17 000
Output
Status

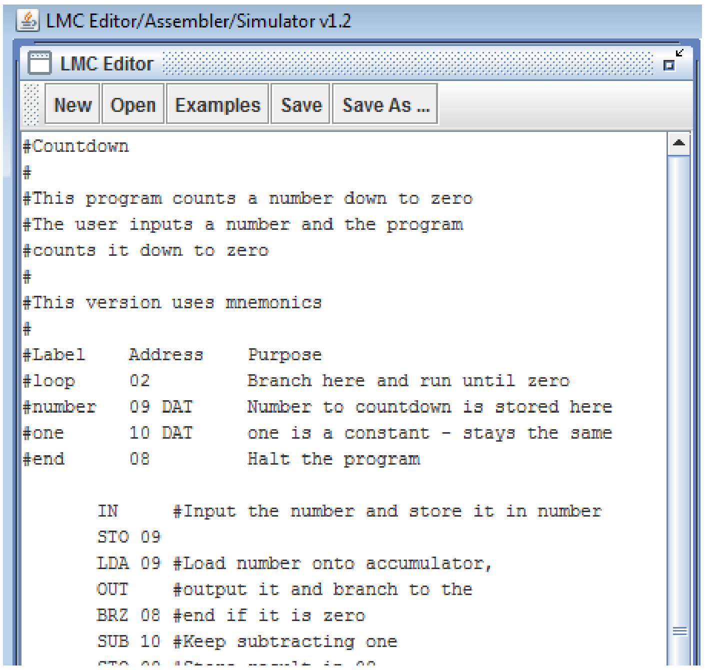

LMC Computer
Load
Step
Run
Input
7
Accumulator
0
Location
1
Memory
Addr
Value
00 901
01 314
02 315
03 515
04 711
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 000
15 000
16 001
17 000
Output
Status
Enter number between -500
and 499 in input field
then hit the "Enter" key

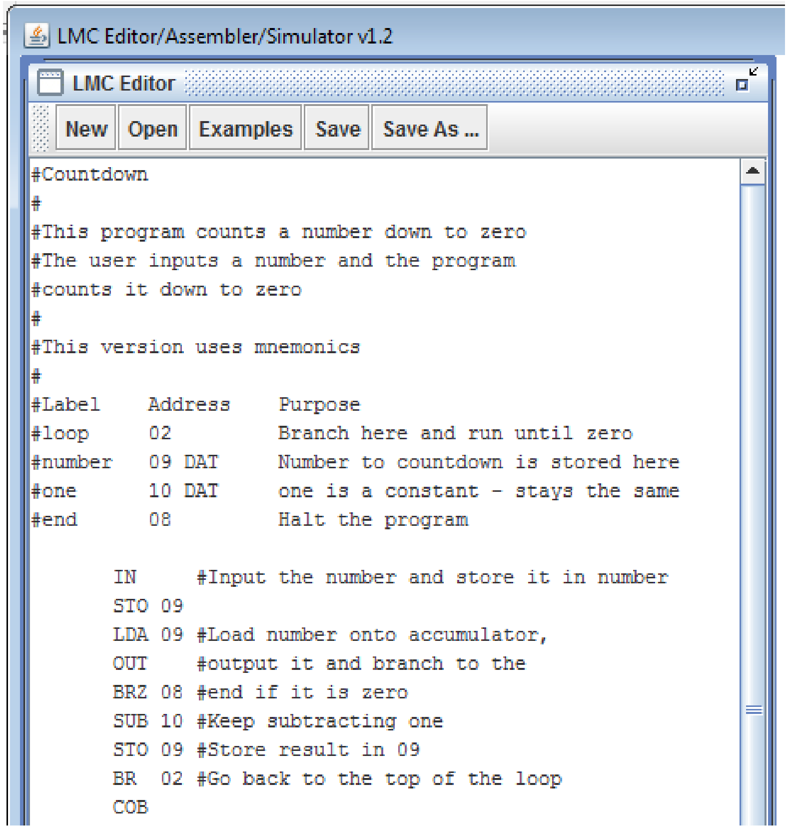

LMC Computer
Load Step Run
Input
7
Accumulator
49
Location
14
Memory
Addr Value
00 901
01 314
02 315
03 515
04 711
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 007
15 000
16 001
17 049
Output
> 7
49
Status
Program halted normally


LMC Computer
Load Step Run
Input
5
Accumulator
25
Location
14
Memory
Addr Value
00 901
01 314
02 315
03 515
04 711
05 216
06 315
07 517
08 114
09 317
10 603
11 517
12 902
13 000
14 005
15 000
16 001
17 025
Output
> 5
25
Status
Program halted normally

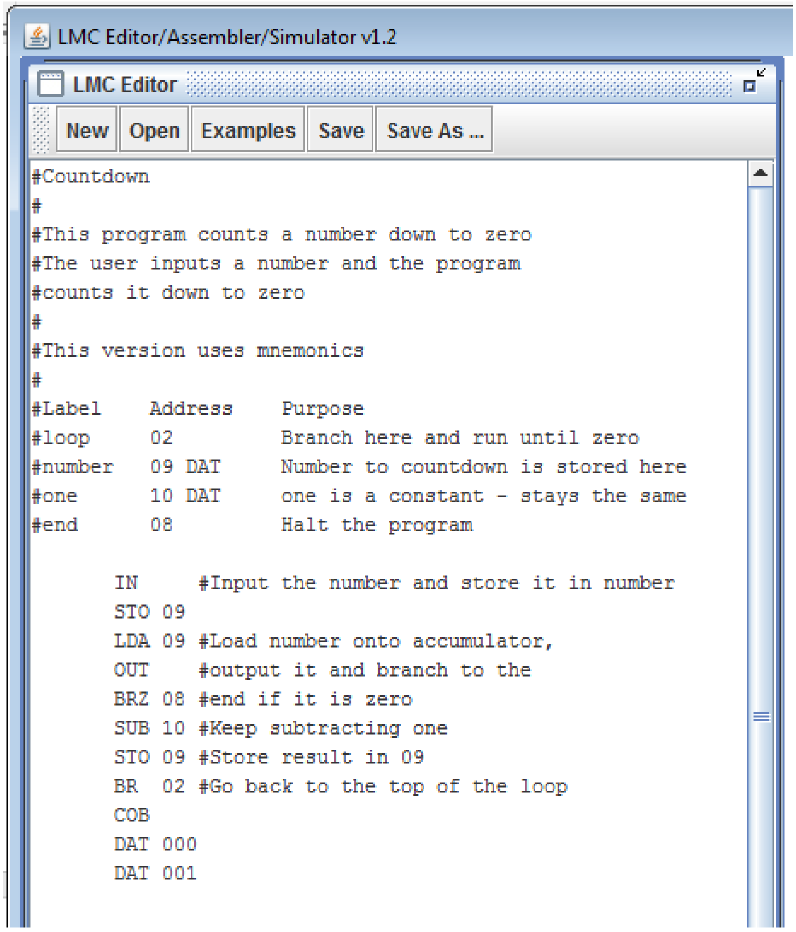
- Countdown


LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Countdown
#
#This program counts a number down to zero
#The user inputs a number and the program
#counts it down to zero
#
#This version uses mnemonics
#
#Label Address Purpose
#loop 02 Branch here and run until zero
#number 09 DAT Number to countdown is stored here
#one 10 DAT one is a constant - stays the same
#end 08 Halt the program
IN #Input the number and store it in number
STO 09
LDA 09 #Load number onto accumulator,
OUT #output it and branch to the

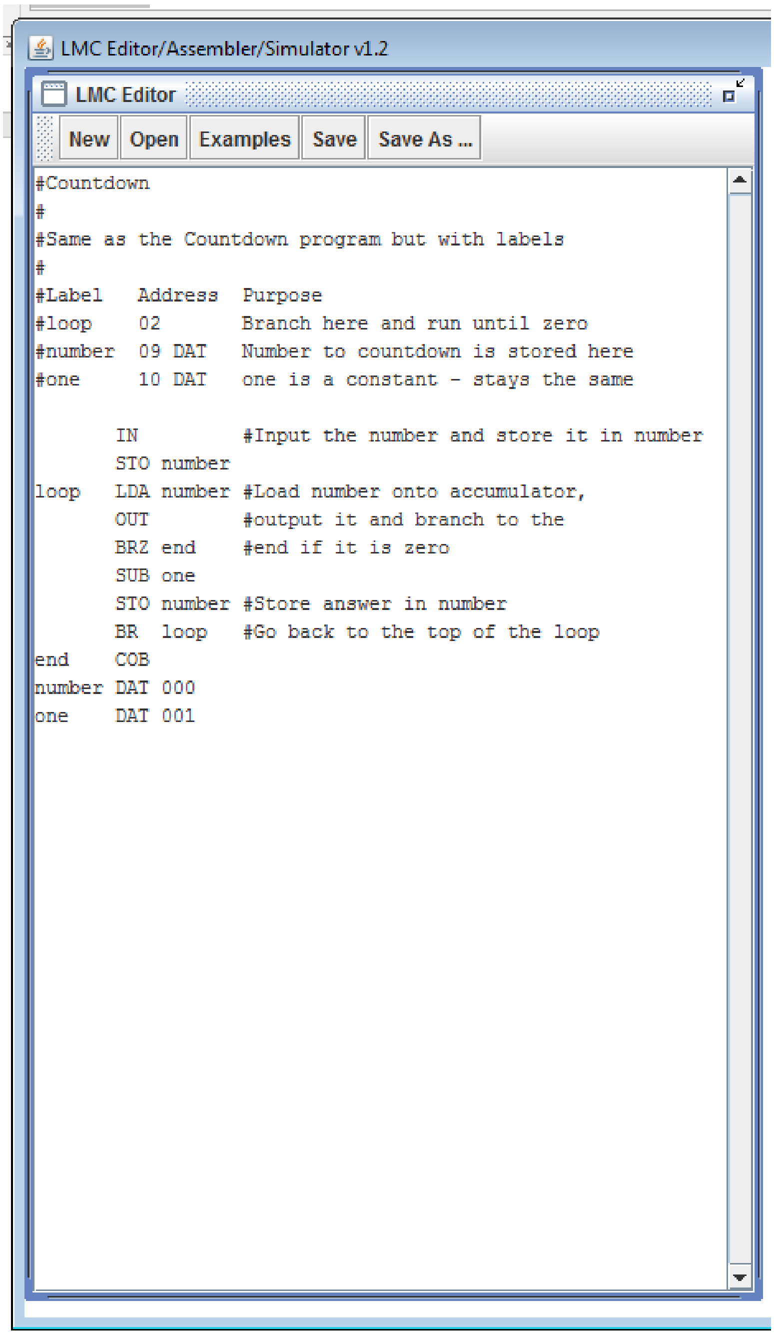

LMC Editor/Assembler/Simulator v1.2
LMC Editor
New
Open
Examples
Save
Save As ...
#Countdown
#
#This program counts a number down to zero
#The user inputs a number and the program
#counts it down to zero
#
#This version uses mnemonics
#
#Label Address Purpose
#loop 02 Branch here and run until zero
#number 09 DAT Number to countdown is stored here
#one 10 DAT one is a constant - stays the same
#end 08 Halt the program
IN #Input the number and store it in number
STO 09
LDA 09 #Load number onto accumulator,
OUT #output it and branch to the
BRZ 08 #end if it is zero
SUB 10 #Keep subtracting one



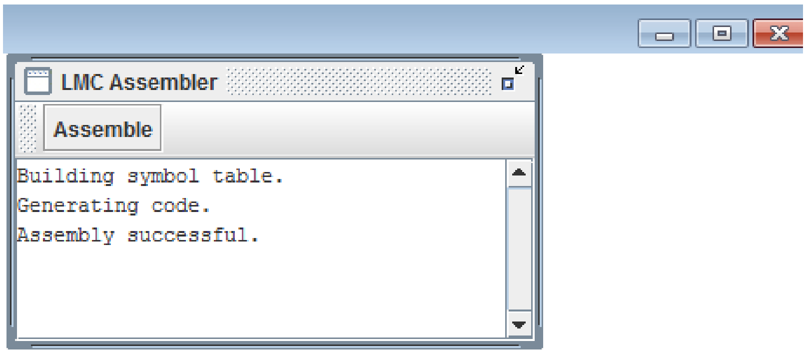

```
#Countdown
#
#This program counts a number down to zero
#The user inputs a number and the program
#counts it down to zero
#
#This version uses mnemonics
#
#Label    Address    Purpose
#loop     02         Branch here and run until zero
#number   09 DAT     Number to countdown is stored here
#one      10 DAT     one is a constant - stays the same
#end      08         Halt the program

       IN     #Input the number and store it in number
       STO 09
       LDA 09 #Load number onto accumulator,
       OUT    #output it and branch to the
       BRZ 08 #end if it is zero
       SUB 10 #Keep subtracting one
       STO 09 #Store result in 09
       BR  02 #Go back to the top of the loop
       COB
```

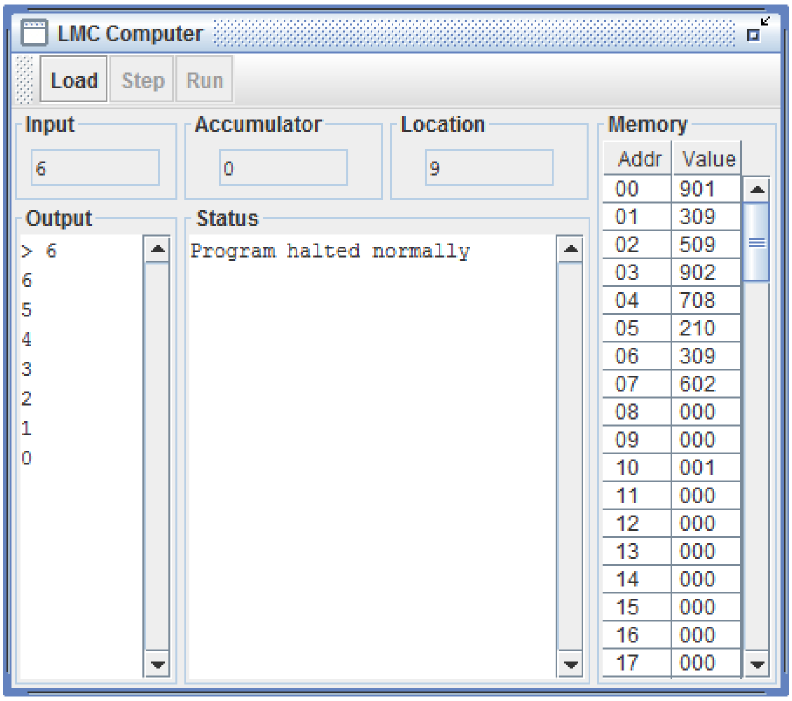


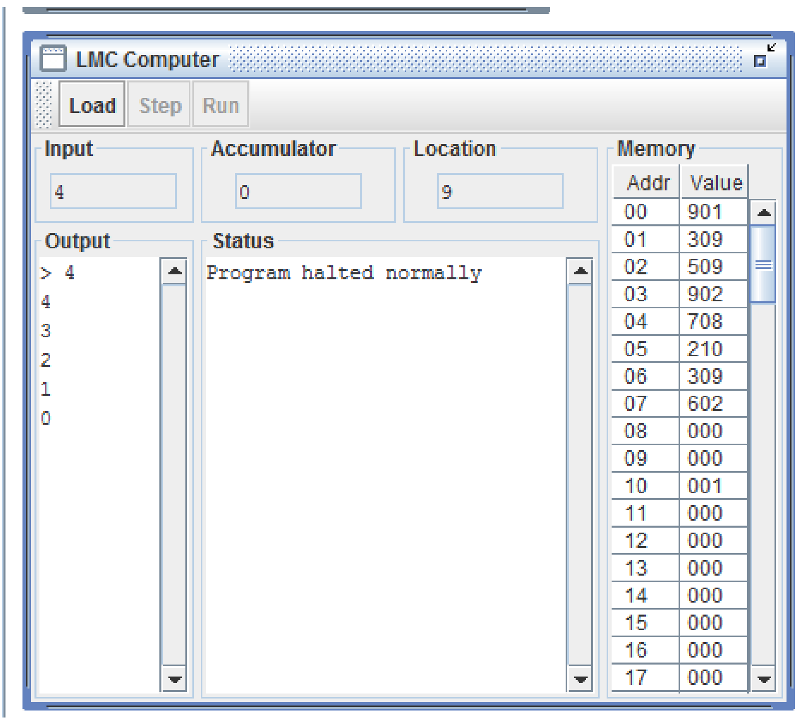

```
#Countdown
#
#This program counts a number down to zero
#The user inputs a number and the program
#counts it down to zero
#
#This version uses mnemonics
#
#Label    Address    Purpose
#loop     02         Branch here and run until zero
#number   09 DAT     Number to countdown is stored here
#one      10 DAT     one is a constant - stays the same
#end      08         Halt the program

       IN     #Input the number and store it in number
       STO 09
       LDA 09 #Load number onto accumulator,
       OUT    #output it and branch to the
       BRZ 08 #end if it is zero
       SUB 10 #Keep subtracting one
       STO 09 #Store result in 09
       BR  02 #Go back to the top of the loop
       COB
       DAT 000
       DAT 001
```




```
#Countdown
#
#Same as the Countdown program but with labels
#
#Label   Address  Purpose
#loop    02       Branch here and run until zero
#number  09 DAT   Number to countdown is stored here
#one     10 DAT   one is a constant - stays the same

       IN         #Input the number and store it in number
       STO number
loop   LDA number #Load number onto accumulator,
       OUT        #output it and branch to the
       BRZ end    #end if it is zero
       SUB one
       STO number #Store answer in number
       BR  loop   #Go back to the top of the loop
end    COB
number DAT 000
one    DAT 001
```


LMC Assembler
Assemble
Building symbol table.
Generating code.
Assembly successful.


LMC Computer
Load
Step
Run
Input
6
Accumulator
0
Location
9
Memory
Addr Value
00 901
01 309
02 509
03 902
04 708
05 210
06 309
07 602
08 000
09 000
10 001
11 000
12 000
13 000
14 000
15 000
16 000
17 000
Output
> 6
6
5
4
3
2
1
0
Status
Program halted normally


LMC Computer
Load
Step
Run
Input
4
Accumulator
0
Location
9
Memory
Addr Value
00 901
01 309
02 509
03 902
04 708
05 210
06 309
07 602
08 000
09 000
10 001
11 000
12 000
13 000
14 000
15 000
16 000
17 000
Output
> 4
4
3
2
1
0
Status
Program halted normally

- Fibonacci Sequence(Not complete)