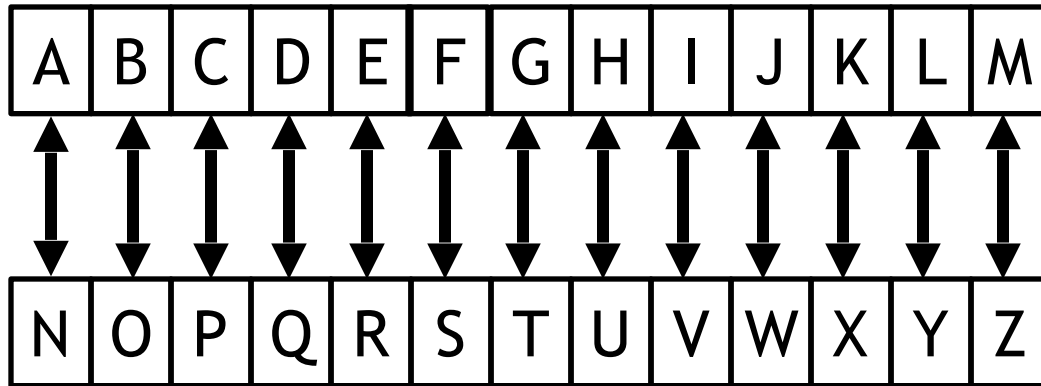


Project 2 Overview

ECE 5484

ROT13



FUZZY → SHMML

SHMML → FUZZY

Program Operation

- Input phase
 - Input a character ('A'-'Z' or '.' only)
 - Apply the Rotate-13 (ROT13) transformation to the character if 'A'-'Z'
 - Store the transformed character in memory
 - Repeat until the input is a period ('.')
- Output phase
 - Output the stored characters (but not the '.')
 - The program halts (instruction "Halt")

Character Input

A screenshot of a debugger's register window. The window displays several registers: AC (0204), IR (5000), MAR (102), MBR (0204), PC (103), and INPUT (Y). The PC and INPUT fields are circled in yellow. The INPUT field also has a dropdown menu set to ASCII. The background shows a memory dump with hexadecimal values.

Register	Value	Format
AC	0204	(Hex)
IR	5000	(Hex)
MAR	102	(Hex)
MBR	0204	(Hex)
PC	103	(Hex)
INPUT	Y	ASCII

Memory Dump (Hex):

Address	Value
412E	9129
1131	C11E
0041	005A
0021	

Character Output

The image shows a computer simulation interface with two main panels. The left panel contains five registers: AC (0000), IR (7000), MAR (11D), MBR (002E), and PC (11E), all in hexadecimal. The bottom register, INPUT, contains a period '.' and is circled in yellow. The right panel, titled 'OUTPUT', shows the characters 'S', 'H', 'M', 'M', and 'L' stacked vertically and is circled in green. Below the output window are two dropdown menus labeled 'ASCII' and 'Control'.

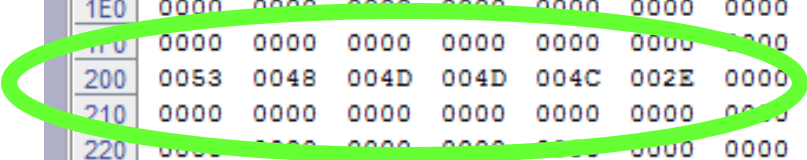
AC	0000	(Hex)
IR	7000	(Hex)
MAR	11D	(Hex)
MBR	002E	(Hex)
PC	11E	(Hex)
INPUT	.	ASCII ▼

OUTPUT

S
H
M
M
L

ASCII ▼ Control ▼

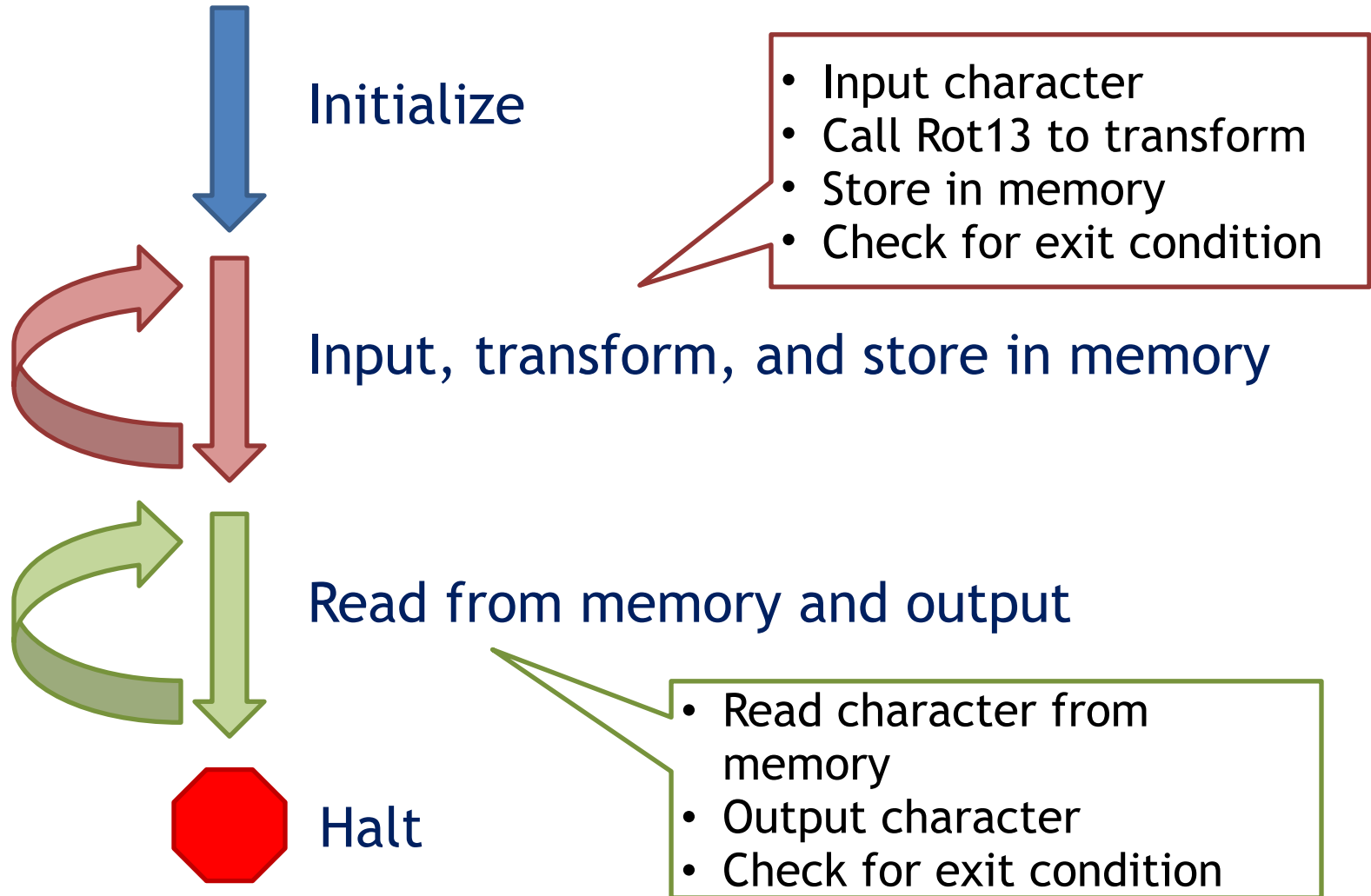
Characters in Memory



	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
1B0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1C0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1D0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1E0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1F0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
200	0053	0048	004D	004D	004C	002E	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
210	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
220	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
230	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000

Machine halted normally.

Program Design



Additional Requirements (1)

- a) The first instruction of the program must be placed at location (address) 0x100 (100 hexadecimal) in MARIE's memory.
- b) Constant data values should not be changed by the program.
- c) Transformed input characters must be stored in successive memory locations beginning at location 0x200 (200 hexadecimal). The program should store all transformed input characters before any characters are output.

Additional Requirements (2)

- d) The program should always initialize the values for Ptr in the working data memory and not rely on the values for these locations that are defined in the assembly source file.
- e) The program should work for any inputs 'A' through 'Z' and '.' (a period terminates input). The program does not need to validate inputs.
- f) When transformed characters are stored and when transformed characters are output, the program must use a loop and indirect addressing to access the values in the array of words.

You *may* define a Count variable to count the number of characters, but there are also correct designs that do not require a Count variable.

Two Test Cases

- Test 1: Input the eight-character sequence “VIRGINIA” followed by a ‘.’ to terminate the input. The ROT13 value of each character (“IVETVAVN”) should be displayed after the ‘.’ character is input.
- Test 2: Reload the program in MarieSim, without reassembling, input the four-character sequence “GRPU” followed by a ‘.’ to terminate the input.

Starting Code: Initialization

ORG 100

/Initialize Ptr (pointer).

Load Start

Store Ptr

/ ** Add code to accomplish the input and output phases.

/ Here's an example of how subroutine ROT13 is called.

/ We'll just transform 'A' in this example then halt.

Load ChA

Store InVal

Jns ROT13

/ Halt

Halt

Starting Code: ROT13 Subroutine

ROT13, HEX	0
Load	InVal
Add	Val13
Store	Hold
Subt	ChZ
Skipcond	800
Jump	NoAdj
Add	ChA
Jump	Done
NoAdj, Load	Hold
Done, JumpI	ROT13



Starting Code: Constants, Data Area

```
/ -----  
/ Constants (the program should not write to these  
locations)  
/ -----  
ChA,    HEX          0041      / Constant value 'A'  
ChZ,    HEX          005A      / Constant value 'Z'  
ChPe,   HEX          2E        / Constant period char  
Val13,  DEC          13        / Constant rot value 13  
One,    HEX          1         / Constant value 1  
Start,  HEX          200       / Constant address  
  
/ -----  
/ Data area (these locations are for reading and writing)  
/ -----  
InVal,  HEX          0         / Subroutine input value  
Hold,   HEX          0         / Temporary variable  
Ptr,    HEX          0         / Character pointer
```

Project 2: Report Contents

- Name, email, title at the top of page 1
- Body of the Report
 - Section 1 - Objectives
 - Section 2 - Design Description
 - Section 3 - Testing and Results
 - Section 4 - Conclusions
- Submit three files
 - Report (*.pdf)
 - Source (*.mas)
 - Listing (*.list)
 - Machine language (*.mex)

Hint: Learn to use MarieSim's Features

- Editing and assembling code
- Executing code
 - Run
 - Step
 - Breakpoints
- Inspecting registers and memories
 - Registers
 - Memory

Hint: Develop in Stages

- Input character until ‘.’ then Halt
- Input character, do ROT13 until ‘.’ then Halt
- Input character, do ROT13, store character until ‘.’ then Halt
- Input characters, do ROT13, store character until ‘.’ then output characters

This is just one approach. The key is to start simple, test, fix, and expand slowly until fully functional.

Hint: Look at Examples

- Starting code provided with Project 2
 - Some initialization
 - Most of the subroutine
 - Example of call to the subroutine
- Ex4_2.mas (downloaded with the MARIE software files)
 - Loop to read characters from memory and output
 - “Null” (00H) is the terminating character
- Ex4_4.mas (downloaded with the MARIE software files)
 - Subroutine call