Assembly language programming with I/O

LC-3 TRAP routines for Input/Output (I/O)

* TRAPs for input:

1) IN: print to screen "Input a character>",
   store ASCII value in RO

2) GETC: don't print anything on screen,
   store ASCII value in RO
* TRAPs for output:
  1) OUT:
     Print to screen ASCII value in RO
  2) PUTS:
     Print to screen null-terminated string stored in memory, location of first character stored in RO
  3) PUTSP:
     Similar to PUTS but get two characters per memory location
Programming example

* Problem statement:
  Print prompt in screen asking user to type a character, then write on screen

    "The character you pressed was:"

  and print on screen the character entered by the user
* Flowchart:
* Program:

Start

Initialization

Get character

Save character

Print string

Print character

Stop

Register usage:

RO: TRAP interface

RA: return from TRAP

R1: character saved
trapIO.asm

; TRAP I/O example
;
; Print prompt in screen asking the user to type a character,
; then write on screen "The character you pressed was: "
; and print on screen the character entered by the user
;
; Register usage:
; R0: TRAP interface
; R1: character saved
; R7: return from TRAP
;
; Start
.ORIG x3000

; Get character
IN

; Save character
ADD R1, R0, #0

; Print string
LEA R0, TEXT
PUTS

; Print character
ADD R0, R1, #0
OUT

; Stop
HALT

; String to print
TEXT .STRINGZ "The character you pressed was: "

.END

trapIO.sym

// Symbol table
// Scope level 0:
// Symbol Name Page Address
// -------------------------
// TEXT  3007
Example: sprite

Sprite is a two-dimensional figure that is embedded in a larger scene.

Sprite: 

Scene: 

© 2017-2018 Juan Jose Jaramillo. All rights reserved.
Example:

```
* * * * * * * *
*   ^   ^   *
* * * * * * * *
* * * * * * * *
* * * * * * * *
```

8 rows

8 columns

Question: how do we store a two-dimensional figure in a one-dimensional memory?
Answer: row after row, as if it were a one-dimensional array

```assembly
; row 0
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 .FILL x2A ; *
 ; row 1
 .FILL x2A ; *
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x20 ; " "
 .FILL x2A ; *
 ; row 2
 ...
```
* Algorithm:

```
Start
  R2: sprite starting address
  R3: row counter
  R4: column counter

  R3 > 0
    Yes
       R4 = R4 + 1
       R3 = R3 - 1
    No
       Output NL

  R3 = 0
    Yes
       Stop
    No

Valid range:
R0: MEM[R3]
R1: R2
R3: R4
```

Register usage:
R2: memory location
R3: row counter
R4: column counter
.ORIG x3000

; load output string's address
LEA R2, SPRITE

; row counter set to 8
AND R3, R3, #0
ADD R3, R3, #8

NEXT_ROW
ADD R3, R3, #0
BRz DONE

; column counter set to 8
AND R4, R4, #0
ADD R4, R4, #8

NEXT_COLUMN
ADD R4, R4, #0
BRz DONE_ROW

; print next char
LDR R0, R2, #0 ; read next char
OUT
ADD R2, R2, #1 ; move to next char

; decrement column counter and move to next
ADD R4, R4, #1
BRnzp NEXT_COLUMN

; print new line char
DONE_ROW
LD R0, ASCII_NL ; load new line ASCII value
OUT

; move to next row
ADD R3, R3, #1
BRnzp NEXT_ROW

DONE
HALT

ASCII_NL .FILL xA

; row 0
SPRITE .FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
.FILL x2A ; *
; row 1
.FILL x2A ; *
.FILL x20 ; " "
.FILL x20
.FILL x20
.FILL x20
.FILL x20
.FILL x20
.FILL x20
.FILL x20
; row 2
.FILL x2A ; *
.FILL x2A ; *
.FILL x20
.FILL x5E ; ^