Project 2 LC3 lil_win

Instructions: Follow lab instructions below and complete the following:

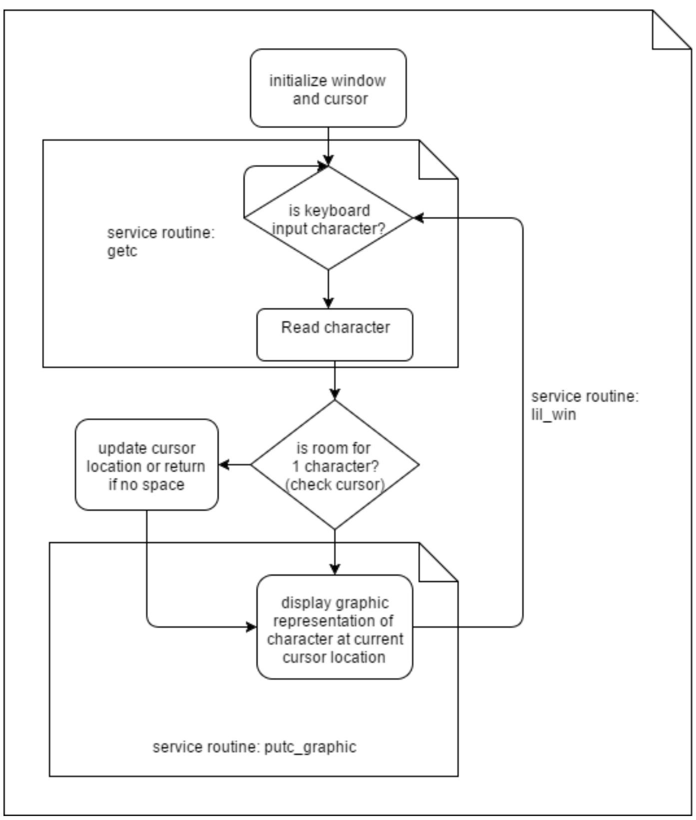
1. **Create a lab report and submit/upload via canvas as a .pdf . See Lab Report Specs document for formatting details.**
2. **Commit code to your personal branch as instructed**

Key

- Methods / Procedures: are enumerated.
- Questions: are italicized and generally ask you to share your observations and conclusions.
- Commit instructions: Code to commit to your branch are underlined.

Objectives: Implement Trap, Implement I/O driver(s).

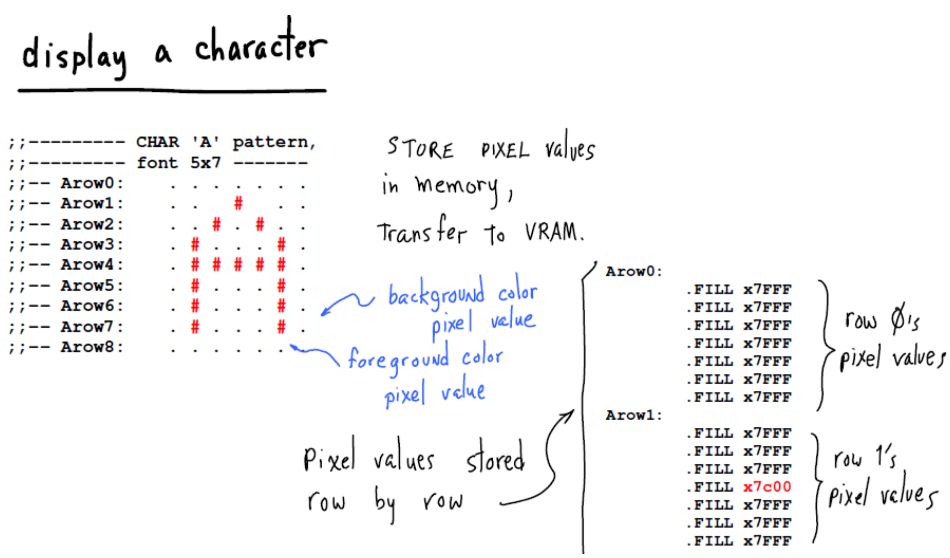
For this project you will be creating a very basic window display program using the memory-mapped, LC3 graphics display. Specifically, your program will initialize and empty window, read input (alphabetic letters) from the keyboard, and display the characters (in order), while “maintaining” a cursor. See high-level flow chart below.



1. Copy boot.asm, user.asm, and OS.asm files to a local directory, then open and read over them using your favorite editor. You can find these files in …/projects2/121-2017/LC3-OS/putc

You will need to add a few subroutines and graphics information in OS.asm. Below you can choose standard Traps so you can use pseudo-ops if you wish. Otherwise, you can directly call Trap as appropriate.

2. In OS.asm create the following service routines (rename p2OS.asm)
    a. _getc (Trap x20 aka GETC)
        i. poll: when KBSR == 1, R0 ← KBDR
    b. _putc_graphic (Trap x21 aka OUT)
        i. R0: inputChar
        ii. R1: cursorLocation
        iii. Store graphic Representation of inputChar at corresponding cursor location. Choose color.
            1. NOTE: You will need to store (in memory) an encoding of each letter to be displayed, ie ascii to pixel representation. For example, for the letter A. See LC3-GraphicsMode-ascii-2-pixels.pdf . NOTE: You need only produce graphics for Capital Alphabetic characters (26) and a space. Use 5x7 , 7x9 representation for each char. See example below.



    c. _lil_win (Trap x23 aka IN)
        i. initialize window
            1. windowSize: 40x84
            2. windowLocation (upperleftHandCorner): row = 30, col = 20
            3. Choose a color.
            4. Initialize cursor location.
        ii. Repeatedly call getc and putc_graphic.
            1. Maintain cursor location (where the next char should appear in graphic display. Note: cursor should wrap around to the next line when the end of the window is reached. If the final line in the window is filled, the subroutine should return to the caller.

3. In OS.asm create subroutines to initialize each of the newly created service routines. The initializers should be called when the OS is first executed.
   a. _init_getc
   b. _init_putc_graphic
   c. _init_lil_win

4. Before implementing these methods, create a design plan.

Q.1 Create a flow diagram (along with a brief description) ***for each*** *of the subroutines produced, except for the initializers (probably 3 to 5 depending on your implementation). Provide more detail than the high-level diagram included in the instructions. Feel free to use your favorite application that creates flow diagrams, or feel free to draw this by hand and digitize, or try draw.io, or … .*

Note: Most code updates will be made to OS.asm (please rename as p2OS.asm). You may need to make minor updates to boot.asm and user.asm. Note: user.asm should simply call subroutine lil_win. And boot.asm should simply call the main in p2OS.asm.

Other Requirements:

- Use TRAP, JSSR and RET to facilitate subroutine calls, appropriately.
- Adhere to "callee-save" protocol, when creating subroutines
- Provide comments generously

Testing: To simulate keyboard input, we will use PennSim's "input" command. (Before testing the final product, I encourage incremental testing of each of the subroutines. The simulator allows for this easily – simply step through each subroutine and provide inputs as needed.)

5. Create a txt file named "input.txt". Within this text file, type a message such as "HELLO WORLD FROM <YOUR NAME HERE>"
6. Assemble p2OS.asm, p2user.asm, and p2boot.asm and then load their corresponding .obj files.
7. Begin simulation by stepping through the code. You will reach a polling loop (in getc) which repeats since KBSR is 0. At this point, you can simulate keyboard input by typing "input input.txt" at the PennSim command line.

At this point the KBSR should go high and the characters contained in input.txt should be sequentially placed into KBDR by the simulator.

8. Continue to step through the code.

The lil_win subroutine should graphically display the contents of input.txt into the graphics window area.

Q.2 Provide a screen shot of PennSim with a message successfully displayed in the graphics area.

Q.3 As always, include all code in the Appendix of your report.

<u>C.1 Save new files as p2OS.asm, p2boot.asm and p2user.asm and Commit to your branch.</u>

ASCII Table. (You need only concern yourself with capital alphabetic letters and space. Feel free to represent all other (unrecognized) input chars as a space.)

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL