Lab 08

Lab 08 checkpoint 1 is due on next Thursday in lecture.

Lab 08 checkpoint 2 is due on Friday in two weeks on MySTU.

This lab assignment is challenging and time-consuming! Start early, plan your time accordingly.

Programming in LC-3 assembly language

In this laboratory assignment, you will write a short program in LC-3 assembly code to render a string in a large (9x16-pixel) font to the monitor. The objectives for this assignment are for you to gain some experience with assembly language and to start to understand how data is organized in memory and how such structured data can be used by a program. In this case, the data of interest maps each ASCII character into a picture of the character. By making use of this data to print a string to the monitor, you will begin to learn how to organize data in a way that makes it easy to use in a program.

Download **lab08.tar.gz** from MySTU and extract to your **lab8** directory. You are provided with several files:

- File **lab8.asm** is where you will write your assembly program.
- File **onetest.asm** contains a sample test input.
- **runonetest** is script file to execute your program with the test input.
- **onetestout** contains a correct version of the output for the given sample test.

Problem statement

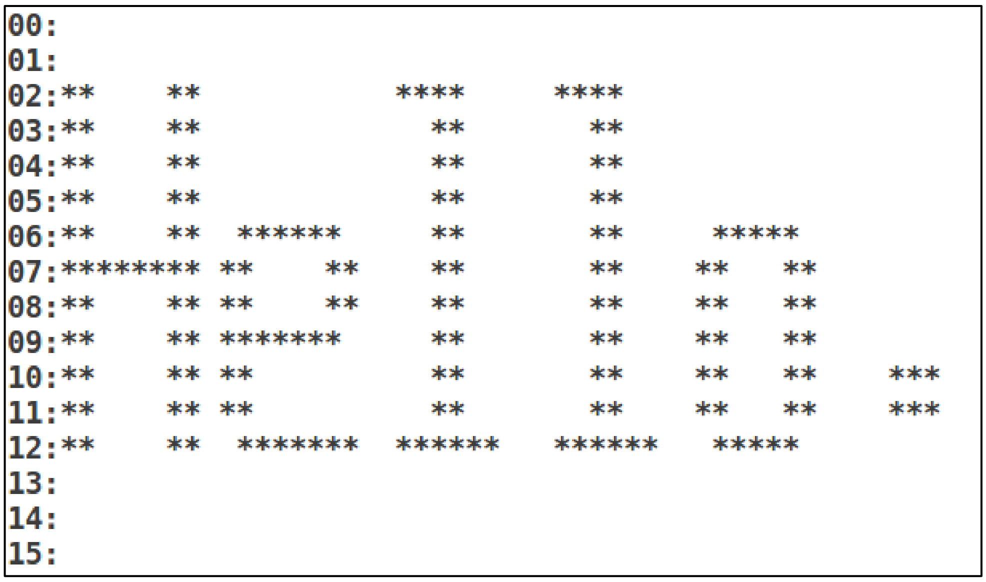
You must write a program in LC-3 assembly language to print a string to the monitor using a large font. We have provided font data that map each 8-bit ASCII character into an 9-bit-wide by 16-bit-high picture of the character. In memory, each such picture is represented using 9 bits in each of 16 contiguous memory locations, and the pictures for each of the 128 possible characters are then simply stored consecutively in memory. The complete set of font data thus occupy a total of 2048 = 16 × 128 LC-3 memory locations.

Additional Note: The FONT_DATA is arranged according to their ASCII encodings. The first 16 addresses (0-15) in the FONT_DATA correspond to ASCII x00, the NUL character. The next 16 addresses (16-31) correspond to ASCII x01, the SOH character. Use the ASCII encoding of the letters you need to print to similarly find the addresses in FONT_DATA.

The font data are provided to you in a file called **lab8.asm**. Note that the file includes only a single label, FONT_DATA, which points to the start of the data. You must add appropriate assembly code and directives to this file to print a given string to the monitor.

Address	Contents	Meaning
x5000	x0048	‘H’
x5001	x0065	‘e’
x5002	x006C	‘l’
x5003	x006C	‘l’
x5004	x006F	‘o’
x5005	x002E	‘.’
x5006	x0000	NUL

Input values to your program are stored in memory locations starting at x5000, as shown in the table above. The string that you must print begins at address x5000 and ends with a NUL (ASCII x00) character. For the example values shown in the table, your program should produce the output shown below. We have added line numbers starting with 0 at the leftmost side of the output to illustrate the inclusion of all spaces.


```
00:
01:
02:**    **           ****      ****
03:**    **             **        **
04:**    **             **        **
05:**    **             **        **
06:**    **  ******     **        **      *****
07:******** **    **    **        **     **   **
08:**    ** **    **    **        **     **   **
09:**    ** *******     **        **     **   **
10:**    ** **          **        **     **   **    ***
11:**    ** **          **        **     **   **    ***
12:**    **  *******  ******    ******    *****
13:
14:
15:
```

The font data for the capital letter 'H' can be seen from the example. The first two memory locations contain x0000. The next five contain xC300 (11000011 followed by eight 0s). The 8th memory location contains xFF00 (11111111 followed by eight 0s). The next five memory locations contain xC300. And the last three memory locations contain x0000.

Getting Started

Your first step should be to systematically decompose the problem to the level of LC-3 instructions. **We require that you develop your flow chart first**, and we strongly advise you not to try to write the code by simply sitting down at the computer and starting to write without having the flowchart. **We will not help you if you do not show us your systematic decomposition flowchart first!**

We suggest that you start by coming up with an algebraic expression for the address that holds the bits needed for a particular line of a particular character. You will find the character to be printed by walking over the string (once for each of the 16 lines to be printed). You need to keep track of the line numbers yourself. You might choose to use one of the LC-3 registers as a loop counter with the line numbers shown in the example; then use the loop counter when calculating the address of the font data for a character. For your convenience, the 9 bits of interest are stored in the upper 9 bits of the word in memory. If we had used the lower 9 bits, your code would have to shift each word up after it was loaded in order to use the LC-3 condition code to check the value of each bit. Note that you can use an immediate mode ADD operation with second operand equal to 0 to set the condition codes based on the value in a given register.

Next, think about how you will structure your solution in terms of iterations, conditionals, and sequences. Note that the grading rubric gives a few hints as to what you will need to include, if you're having trouble starting.

Specifics

Your program must be called lab8.asm — we will NOT grade files with any other name.

- Your code must begin at memory location x3000.
- The last instruction executed by your program must be a HALT (TRAP x25).
- The string to be printed starts at x5000 and ends with a NUL (x00) character.
- Use a single line feed (x0A) character to end each line printed to the monitor. Your program must use an iteration for each line in the output.
- Your program must use an iteration for each character in the string to be printed. Remember that a string ends with a NUL (x00) character, which should not be printed.
- Your program must use an iteration for each bit to be printed for a given character in the string. You may not make assumptions about the initial contents of any register.
- You may assume that the string is composed of valid ASCII characters (x0000 to x007F).
- You may use any registers except R7. The reason why we recommend that you avoid using R7 will be explained in later lectures.

Note that you must print all leading and trailing characters, even if they are not visible on the monitor (as with spaces). Do not try to optimize your output by eliminating trailing spaces, as you will make your output different from ours (and will thus lose points).

Tools

Use a text editor on a Linux machine (**gedit**, for example) to write your program for this lab. Use the LC-3 simulator in order to execute and test the program. Your code must work on the server machine to receive credit.

Testing

You should test your program thoroughly before handing in your solution. Remember, when testing your program, you need to set the relevant memory contents appropriately in the simulator. You may want to use a separate ASM file to specify test inputs, as discussed below. When we grade your lab, we will initialize the memory for you. Developing a good testing methodology is essential for being a good programmer. For this assignment, you should run your program multiple times for different functions and inputs and double check the output by hand.

We have also given you a sample test input, **onetest.asm** (you will need to assemble this file), a script file to execute your program with the test input, **runonetest**, and a correct version of the output, **onetestout**. Look at the script: load the sample input, then load your program. The "reset" command/button will not work, since you are using more than one program. Set the PC by hand if necessary (for example, r pc 3000), or re-load both files (input and your program) whenever you need to restart a debug effort.

First you must debug---don't assume that you can simply run the script to debug your

code. Once you think that your code is working, you can execute the script by typing

the following:

```
lc3sim -s runonetest > myout
```

This command executes the LC-3 simulator on the script file and saves the output to the file myout. If the command does not return, your program is stuck in an infinite loop (press CTRL-C). Otherwise, you can compare your program's output with our program's by typing

```
tkdiff myout onetestout
```

Note that **your final register values need not match those of the output that we provide**, but all other outputs must match exactly.

What to Submit

- Checkpoint 1 submission should consist of a systematic decomposition of your entire algorithm. It is to be drawn on paper and turned in Lecture on Thursday.
- Checkpoint 2 submission is to be done via MySTU before the deadline for the assignment. You should double check that your file was uploaded properly. Demonstrate your code at lab in front of a teaching assistant, answer questions, and sign your name.

Grading Rubric for checkpoint 2

- Functionality (50%)
  - 50% - program prints string stored starting at x5000 correctly
- Style (35%)
  - 10% - program uses a single iterative construct to print every line of output
  - 10% - program uses a single iterative construct to print every character in string
  - 10% - program uses a single iterative construct to print every bit in a character
  - 5% - program uses a single conditional construct to select character for printing
- Comments, clarity, and write-up (15%)
  - 5% - introductory paragraph clearly explaining program's purpose and approach used
  - 5% - code includes table of registers that explains their meaning and contents as used by the code
  - 5% - code is clear and well-commented

Table E.2 The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F