LC3 Programming Assignment – Hexadecimal Input and Multiply

In this assignment you will need to create three subroutines. Create a file named assignHexMult.asm and put the following subroutines inside. For this assignment, you will not need any helper subroutines. Only create the three subroutines shown above.

The three subroutines are summarized here:

> GETHD – Get a single hex digit character from the user and put the integer version of that digit in R0.
> MULT – Multiply two numbers stored in R1 and R2 **efficiently** and return the result in R0.
> GETH4 – Gets 4 hex digits from the user and converts those digits to a single UNSIGNED 16-bit integer stored in R0. You will need to use MULT for this one.

NOTE: There are subroutines in the textbook (and online) to do conversions similar to the above, but they use SIGNED NUMBERS and the algorithms will be very different.

GETHD – Get Hex Digit

1. You may modify GETNUM from the previous assignment.
2. DO NOT print any sort of prompt.
3. Return value must be stored in R0.
4. Get a single character from the user.
    a. If that character is a good hex digit (0-9, A-F, a-f).
        i. Print it. DO NOT PRINT A NEWLINE.
        ii. Convert it from ASCII to its decimal equivalent (for letters a=10, b=11, c=12, etc...)
        iii. Store the decimal value in R0.
    b. If that character is bad
        i. Get another character.
5. This method should not return until the user has typed in a good character.
6. Make sure to check boundary characters. That is check the character before 0 in the ASCII chart, the character after 9, the character before A, the character after F, the character before a and the character after f. Your program should ONLY allow hex digits. I believe the previous GETNUM had a boundary issue.
7. This portion of the program should use around 50 instructions steps. The longest test will use around 120 steps. The limit for you is 300. You can see the number of instructions use in the Java based simulator.

MULT – Multiply efficiently – The problem with the MULT shown below is that a multiplication like 4096 * 3 could be done efficiently or inefficiently depending on the order of the multiplicands. In one of the multiplications, you would do 3 add operations but in the other you would do 4096 add operations.

1. Create a function that multiplies by repeated addition. Multiplicands will be stored in R1 and R2. You can simply add R1 to R0 repeatedly R2 number of times. Here is a simple algorithm.

```
        CLEAR R0
TOP     ADD R1 to R0
        Decrement R2
        If R2 != 0 goto TOP
```

2. Make it so that the multiplicands (R1 and R2) are used such that the largest is always the value added and the smallest is always the number of times to add.

   For example, R1=500, R2= 2, you should add 500 to R0 2 times.
   But if R1=2, R2=500, you would still want to add 500 to R0 2 times.
   You would not want to add 2 to R0 500 times because the loop is inefficient. You want to loop the fewest times possible.

   You can have two different sets of code. You could have one block for adding R1 repeatedly and one for adding R2 repeatedly and then branch to the appropriate one. Or you can detect which is bigger R1 or R2 and swap them when needed.

3. Return the result in R0. Note that MULT does not PRINT anything and does not use GETC. MULT simply takes what is in R1 and multiplies it efficiently by what is in R2 and puts the value into R0.
4. This portion of the program should use around 50 instructions steps for the tests. The limit for you is 200. You can see the number of instructions use in the Java based simulator.

NOTE that you do not have to worry about negative numbers. Treat this as unsigned multiplication. So, if R1 is 0x5000 and R2 is 2, the result in R0 should be 0xA000, but since this is unsigned 0xA000 is not negative. It is a very large positive. If you hover over the value with a mouse, you will see a negative number but it is all in the context.

GETH4 – Get Hex – Get a 4-digit hex number and return it in R0.

1. Do not print any prompts.
2. Use GETHD to get a single hex digit into R0 and print the digit. GETHD should take care of any bad characters. GETH4 should not in any way use GETC or IN. Use JSR GETHD to get each hex number.
3. DON'T MODIFY GETHD IN ANY WAY. It should still print the characters as they are typed.
4. Multiply R0 by the appropriate column value.
5. Add R0 to a sum for this number.
6. Repeat steps 2, 3, and 4 three more times. I would not use a loop, just copy, and paste.
7. The input will ALWAYS be 4 digits. So simply read a digit, process it, and repeat 4 more times.
8. **Print a newline after you have processed the last number. You should see four hex digits on the screen on their own line after every call to GETH4.**
9. Move the sum to R0.

NOTE THAT THIS PORTION WILL AMPLIFY ANY INEFFICIENCIES IN GETHD AND MULT. IF YOU ARE GETTING AN OVERALL TIMEOUT WHERE YOU DON'T PASS ANYTHING, THEN THIS IS ALMOST DEFINITELY THE CASE.

Do not load constants in loops. Load them before the loop. Do not load positive constants and then make them negative. Limit the number of constants. You do not need -48 and -57. Add -48, save the result, then subtract 9 using immediate add. Reuse registers where possible. Use as many registers as you need but try to design your program so you reuse registers for intermediate results to so you don't have so many saves and restores. Try to use registers in a way to prevent unnecessary data moving. Do not save intermediate results to memory when you can use a register.

Submitting

You will be submitting to **THREE** different assignments on Web-CAT; one submission for each part.

You should have all these subroutines in the same assignHexMult.asm file but you MUST change the portion of code that calls the subroutines for each submission. I call that part 'main' because it acts like main in a Java program because that is where the execution starts.

You will lose 5 points per day late per section. You will lose 1 point per section for submissions over 5. Submissions will not be accepted more than 2 days late.

I HIGHLY SUGGEST you do these in order. Get the first subroutine running and passing Web-CAT before moving on to the second. You will need MULT in GETH4 anyway.

See submitting below.

Submitting GETHD

1. Use the following code as main.
   ```
   .orig x3000
   JSR  GETHD
   JSR  GETHD
   HALT
   ```
2. Save the file as assignHexMult.asm and assemble.
3. Upload the asm file to Web-CAT for the appropriate assignment.

Submitting MULT

1. Use the following code as main.
   ```
   .orig x3000
   JSR  MULT
   JSR  MULT
   HALT
   ```
2. Save the file as assignHexMult.asm and assemble.
3. Upload the asm file to Web-CAT for the appropriate assignment.

Submitting GETH4

1. Use the following code as main.
   ```
   .orig x3000
   JSR  GETH4
   JSR  GETH4
   HALT
   ```
2. Save the file as assignHexMult.asm and assemble.

Upload the asm file to Web-CAT for the appropriate assignment.

TROUBLESHOOTING TIPS:

If your program prints a square it is because OUT requires a printable character. So, if you are trying to print 5 you must have the ASCII code for 5 in R0, not the number 5. The only value under 32 that you should be printing is the newline, also known as linefeed, which is ASCII 10.

Run your code until it is complete. If you think everything is running correctly use Continue to test. Using the main functions as given above, run your code before uploading. Make sure it prints everything as described if needed and make sure that R0 contains the correct values.

EACH function should have its very own .fills for register saves and variables. DON'T TRY TO USE ONE BIG GLOBAL VARIABLE LIST AT THE BOTTOM OF THE CODE. IT WILL LEAD TO ISSUES.

I suggest using a naming system for your labels that identifies the label to the function. For example intead of SR71, SR72, SR73, use a prefix like GD_SR7 for GETHD, ML_SR7 for MULT, and G4_SR7 for GETH4. Using the wrong label is easy to do if you don't have good names.

Here is an ASCII chart for reference.

Dec	Hex	Char
0	00	Null
1	01	Start of heading
2	02	Start of text
3	03	End of text
4	04	End of transmission
5	05	Enquiry
6	06	Acknowledge
7	07	Audible bell
8	08	Backspace
9	09	Horizontal tab
10	0A	Line feed
11	0B	Vertical tab
12	0C	Form feed
13	0D	Carriage return
14	0E	Shift out
15	0F	Shift in
16	10	Data link escape
17	11	Device control 1
18	12	Device control 2
19	13	Device control 3
20	14	Device control 4
21	15	Neg. acknowledge
22	16	Synchronous idle
23	17	End tran. Block
24	18	Cancel
25	19	End of medium
26	1A	Substitution
27	1B	Escape
28	1C	File separator
29	1D	Group separator
30	1E	Record separator
31	1F	Unit separator

Dec	Hex	Char
32	20	Space
33	21	!
34	22	“
35	23	#
36	24	$
37	25	%
38	26	&
39	27	‘
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2c	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	K
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	Delete