CSCI210 Assembly Language and Computer Systems
Lab Assignment
Comparisons and Branching

Before reading a value from the keyboard you must load appropriate values into registers. This is analogous to passing arguments to a function. The Linux “read” function needs to know

- What is the source of the read?
- How much data will be read?
- What is the destination of the read?

Register	Value	Meaning
R7	3	Sys call for “read”
R0	0	Where are you reading from? 0 => Keyboard
R2	number	How many characters are to be read?
R1	address	Address of buffer. What is the destination of the read?

Use Linux SYS calls to write a value to the monitor. Before writing a value to the monitor you must load appropriate values into registers. The Linux “write” function needs to know

- What is the destination of the write?
- How much data will be written?
- What is the source of the write?

Register	Value	Meaning
R7	4	Sys call for “write”
R0	1	Where are you writing to? 1 => Monitor
R2	number	How many characters are to be written?
R1	address	Address of buffer. What is the source of the write?

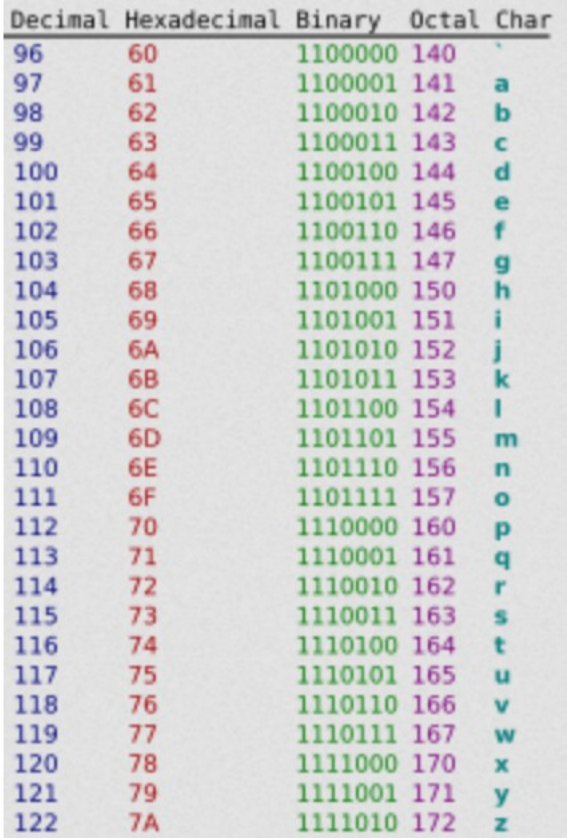
LAB TASKS: Throughout this lab you must comment each line of your source with a descriptive message

Task One

Write an ARM assembly program that will toggle the case of multiple characters in an ASCII string. The string will be entered by the user using Linux SYS calls to read a value from the keyboard. The modified string will then be written to the console window.

In this task you will implement code to toggle the case of an entered ASCII string. Do not worry about trying to make this code flexible; just chose a manageable character limit like 10. You will be converting lower case text to upper case text using an AND operation. This operation will be applied to each character in the entered string. Be sure to enter lower case text when you run this program.

Only apply the toggle to actual ASCII characters. You must skip everything else and leave it unchanged.



Decimal	Hexadecimal	Binary	Octal	Char
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z

- Implement a compound Boolean expression for the range of lower-case ASCII characters and only apply the toggle for this range.
- You must implement short circuiting.

1. Create a source file called: ***toggle_case.s***

2. **Remember:** ASCII data is a single byte per character. You will need to access a single byte at a time.

    1. You can accomplish this using the **LDRB** instruction. **L**oa**DR**egister**B**yte

3. **Algorithm:**
    1. load a byte from the ASCII buffer into a register
    2. Test it for the lower-case range
    3. Apply the AND operation when appropriate
    4. Put the modified byte back
    5. Continue until done

4. Write the manipulated ASCII string to the console using the Linux **write** syscall. At this point you will have to copy and paste the **write** code whenever you need it. We will be writing procedures next week to eliminate this need. Feel free to investigate that now if you'd like. It is not required.

Task Two:

Write an ARM assembly program to convert the contents of a register to an ASCII representation of hexadecimal and display it to the console.

Create a source file called: **print_reg.s**

If the register contains:	3731165867
Display it as:	0xDE6512AB

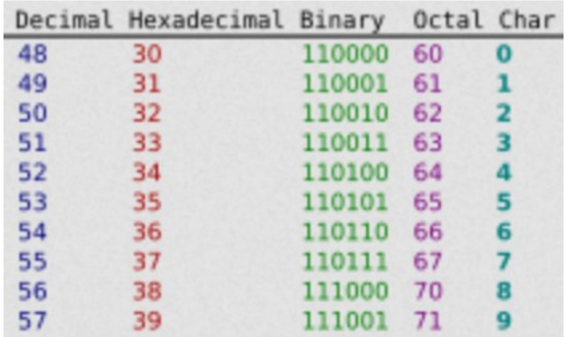
This conversion is rather easy if you consider the convenient relationship between binary and hex.

- 4 bits (nybble) => Hex Digit

3731165867 in binary => 1101-1110_0110-0101_0001-0010_1010-1011

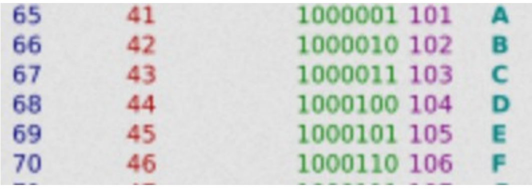
1101	1110	0110	0101	0001	0010	1010	1011
D	E	6	5	1	2	A	B

Also keep in mind the convenient ASCII mapping of the digits {0 – 9} to their associated characters. You can convert a digit to its ASCII representation by setting bits 5 and 6. You would want to do this if the nybble is less than 10.



Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9

If the nybble is larger than 9 you will need to convert the value to the characters {A – B}. Is there a convenient binary operation you could generalize to perform this conversion without individually comparing each A, B, C . . . etc? Consider the numerical value of 0xA and the decimal value of ‘A’.



65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F

Task Three:

Write an ARM assembly program to find the range (difference between highest and lowest value) of values in an array. The array must include both signed and unsigned values.

Create a new source code file called: **array_range.s**

Define a 10-element word array. Print the range with a descriptive message using printf

Task Four:

Write an ARM assembly program to count the number of odd and even values in an array.

Create a new source code file called: **odd_even.s**

Define a 10-element word array. Put the count of evens in R0, the count of odds in R1

Print the results with a descriptive message using printf

Submit: All relevant files and make file.