CS413 Lab 2 ARM Stack Program
Simple Binary Logic

The purpose of this program is to get the student to use the stack to pass parameters to and from their own defined subroutines/functions in ARM Assembly.

Implement a simple four function binary logic calculator (bitwise AND, ORR, EOR, BIC) that meet the following requirements:

1. Each function must be accessed as a subroutine/function (ARM instruction BL).
2. The operands are passed to the function via the stack. Use the ARM pseudo instructions PUSH and POP. To ensure you are properly implementing the stack the registers used in your main routine to hold the operands must be different than those used in the subroutines. For example, your calling routine will push r4 and r5 on the stack but the called routine will pop into registers r10 and r11.
3. The function returns the results on the top of the stack.
4. Your program is to print a welcome/instruction messages to the user.
5. The user is prompted to enter **a 32-bit hexadecimal digit** (%x)* and the operation to be performed.
6. The program shall verify the user input is valid and report any entry errors.
7. The program shall perform the calculation.
8. Display the results of the operation in hexadecimal (%x). Ask the user if they want to continue with another calculation or quit the program.

The TAs will run your programs to ensure correct functioning and will also review your code to ensure you have properly implemented the functions and stacks.

As with all programs your program:
1. Checks user inputs for errors and prints appropriate error messages.
2. All outputs are clearly labeled.
3. Contains generous code documentation.

* Students will need to do some error checking when using the %x as the input string. If the user input is considered a valid hex number the value returned in r0 will be 1. If it is NOT valid r0 will contain a 0. See the grading rubric for invalid input examples. Also note that the %x will not check for an overflow condition (i.e., more than 8 hex digits are entered). If that happens the input value will be all F's (FFFFFFFF). Students will not need to check for an overflow input condition since FFFFFFFF is a valid hex input. When there is an invalid input the input buffer will have to be cleared out before trying another read operation. See code example in student_inputC.s (or latest version) on how to clear out the input buffer.

This assignment was given in:

Fall 2019
Spring 2021

CS413 Lab 2 ARM Stack Program
Simple Binary Logic

Grading Rubric

Feature	Points off
Code Comments/Documentation • Class, Term, Author, Date. • Purpose of software • Documentation for the start of each loop and function (subroutine) • Documentation for error checking • In-line comments that explain why the following code exists.	
Welcome and instruction messages are displayed and clear.	
User is prompted to enter hex digits no more than 32-bits or 8 hex digits.	
User is prompted to enter the operation (AND, OR, XOR, BIC).	
Perform the following valid inputs and calculations	
Perform the following AND and verify the shown result: F0F0F0F0 AND AFAFAFAF = A0A0A0A0	
Perform the following OR and verify the shown result: F0F0F0F0 OR 0A0A0A0A = FAFAFAFA	
Perform the following Exclusive OR and verify the shown result: F XOR A = 5	
Perform the following Bit clear and verify the shown result: A BIC F = 0 F BIC A = 5	
Perform the following invalid inputs	
Invalid input for operand 1: VAX	
Valid input for operand 1: 10f0 Invalid input for operand 2: UNIX	

CS413 Lab 2 ARM Stack Program
Simple Binary Logic

Select an invalid operation to perform. Note: the method for user input to select a function is left to the student. Typically this is either select the number from a menu (1 for AND, 2 for OR, etc.) or enter the character function (*, +, - , /). Students are allowed to use other prompt methods.	
Code Review - Review the code and verify the following required implementation.	
The code for each of the four functions is accessed via a BL call.	
The values to use by the function are pushed onto the stack prior to the BL call.	
The calculated value(s) is/are returned via the stack.	
Different registers used for the push (source) and pop (destination).	