Homework #6 Computer Organization Due: 4/12/13 (F) at 5PM

(You have your choice of doing either Booth's algorithm OR the bit string set-of-letters assignment)

Booth's Algorithm Option:

The goal of this assignment is to provide you with experience in using the "Logical and Shift Instructions". For this assignment, you are to implement Booth's multiplication algorithm on two 32-bit signed integers and get a 64-bit integer result.

I want you to write a function called "Multiply" that is passed two signed integers (in a1 and a2) and returns their 64-bit product across registers a1 and a2. Register a1 should return the most-significant 32-bits of the product and a2 should contain the least-significant 32-bits of the product. Be sure that you DO NOT use any form of the multiply (i.e., MUL) assembly language instruction, and be sure to follow the ARM register convensions when implementing "Multiply."

Your "main" program should:
1) Read two 32-bit integer variables MULTIPLICAND and MULTIPLIER from memory. Enter the value -30 for the MULTIPLICAND and 983 for the MULTIPLIER.
2) call your Multiply function with these values
3) Store the 64-bit product back to memory in a variable called PRODUCT

The data AREA of your program should look something like:

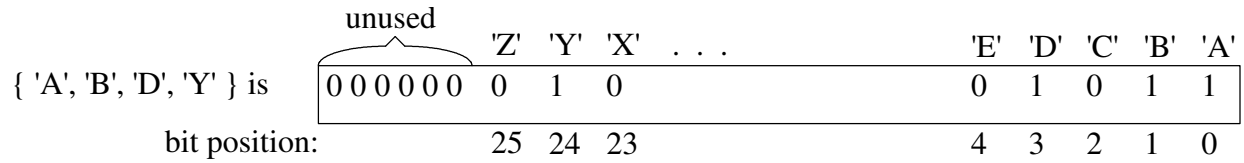
```
                AREA BOOTHS_PROGRAM, DATA, READWRITE
                ALIGN
MULTIPLICAND    DCD -30
MULTIPLIER      DCD 983
PRODUCT         DCD 0,0

STACK_END       SPACE 0x000000FF
                ALIGN
STACK_START     DCD 0
                END
```

Turn in the following:

1) The ARM assembly language programs (main and Multiply) for performing Booth's algorithm.
2) A snapshot of the simulator AFTER RUNNING YOUR PROGRAM, and showing the portion of memory containing the data area above.

Bit string set-of-letters Option:

You are to build an abstract-data type (ADT) for the set of letters using a bit string. The bit string representation for the set of letters can use a 32-bit word with the least-significant bit associated with the letter 'A', etc.



The set of letters ADT should have the following operations (subprograms):

Subprogram Name	**Parameters**	**Description**
bitString	▪pass in a pointer to a null terminated ASCII string ▪returns a 32-bit word containing the set of letters as a bitString	Returns a bit string corresponding to the set of letters in the null terminated string. Non-letter characters are ignored, and both upper and lower-case letters should be represented as the upper-case letter.
union	▪passed two set bitStrings ▪returns the set union of the two sets	The resulting set should contain the elements that are in one or both of the input sets.
intersection	▪passed two set bitStrings ▪returns the set intersection of the two sets	The resulting set should contain the elements that are in both of the input sets.
difference	▪passed two set bitStrings ▪returns the set difference of the first set - second set	The resulting set should contain the elements that are in the first set, but not also in the second set.
contains	▪passed an ASCII character and a set bitString ▪returns a Boolean (0 for false or 1 for true)	Returns 1 (true) if the `ASCII character is in the bitString set; otherwise return 0 (false).
writeString	▪passed a set bitString and a pointer to a character buffer (80 bytes)	Writes the bitString to memory in the form of a null terminated ASCII character string. The string should be written in the conventional format, i.e., "{ E, G, T, Y }"

Additionally, you should have a main program that

1) constructs two bitString sets from two null terminated character strings in memory,
2) writes the set of letters contained in the first string to a character buffer in memory,
3) determines and stores to memory the union, intersection, and difference of two bitString sets,
4) checks to see if the first bitString set contains the letter: 'F'. The result of this check should be stored in a byte of memory as 0 (false) or 1 (true).

The data AREA of your program should look something like:

```
                AREA BITSTRING_PROGRAM, DATA, READWRITE
STR_1           DCB "The first string!",0
STR_2           DCB "Second string?",0

BUFFER          SPACE 80
CONTAINS_F      DCB 0xFF
                ALIGN
HERE            DCD 0x12345678
UNION           DCD 0
INTERSECTION    DCD 0
DIFFERENCE      DCD 0
STACK_END       SPACE 0x000000FF
                ALIGN
STACK_START     DCD 0
                END
```

Turn in the following:

1) The ARM assembly language programs (main and bitString functions)
2) A snapshot of the simulator AFTER RUNNING YOUR PROGRAM, and showing the portion of memory containing the data area above.