CSCI210 Assembly Language
Lab Assignment

Translate the following high-level C functions to assembly.

- You must explicitly create the stack frame and push parameters on the runtime stack. Use local variables if needed. All parameters must be pushed on the stack (for practice)
- Print the results of each procedure along with a descriptive message. Use printf and scanf for any input or output operations.
- For those not familiar with C the ampersand defines reference parameters.

void minMax(int arg[], int size, int& min, int& max) This procedure will set min to the smallest number in arg[ ] and set max to the largest number in arg[ ]

int search(int searchTerm, int args[ ], int size) This procedure will search args[ ] for searchTerm and return its position (zero based) in R0

void bubbleSort(int args[ ], size) This procedure will apply a bubble sort to args[ ].

boolean isDoubloon(char string[ ], int size) This procedure will analyze string[ ] and determine if the data is a doubloon. Set the low order bit of R0 to 1 if the data is a doubloon or 0 if it is not.

A word is said to be a "doubloon" if every letter that appears in the word appears exactly twice. For example, the following are all the doubloons I found in my dictionary. Abba, Anna, appall, appearer, appeases, arraigning, beriberi, bilabial, boob, Caucasus, coco, Dada, deed, Emmett, Hannah, horseshoer, intestines, Isis, mama, Mimi, murmur, noon, Otto, papa, peep, reappear, redder, sees, Shanghaiings, Toto.

Sieve of Eratosthenes (https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)

void sieve(int array[ ], int primes[ ])

Implement the sieve with a limit of 50. The function should fill the **primes** array with the left-over prime numbers. You are not allowed to perform a linear search of the array "looking" for multiples. Be clever with the allocation. Perhaps align the values of the array with the offset of the values location, similar to a hash table, radix sort of bucket sort. **Points will be taken off if this is not followed.**

Print the prime numbers using printf . . . after the function call.

Use the MUL instruction:

Syntax: **MUL{S}{cond} {Rd}, Rn, Rm**

- **cond** is an optional condition code.
- **S** is an optional suffix. If S is specified, the condition flags are updated on the result of the operation.
- **Rd** is the destination register.
- **Rn, Rm** are registers holding the values to be multiplied.

Operation

The MUL instruction multiplies the values from Rn and Rm, and places the least significant 32 bits of the result in Rd.

Have your code generate the list of numbers up to 50

Experiment with:

- The **.skip** directive: **MyArray: .skip 20 * 4**
- This code allocates space for 20 4-byte elements
- The **.set** directive and label subtraction to calculate the "size" of an allocated block.
  string: .asciz "Howdy Globe!"
  after_string:
  .set size_of_string, after_string – string
- **mov r2, #size_of_string** @ move the size of the block into a register

Submission: all source files and a make file. If I must modify anything to build or run, points will be taken off.