ASSESSED **INDIVIDUAL WORK**

COMP1006
LAB EXERCISE #3

Steven R. Bagley

Submission Deadline:
03/11/2021 1200

Version: 1.10

These exercises are designed to be straightforward and are primarily to get you used to using Komodo on the school's Linux system and writing ARM assembly language. These exercises are all built around implementing conditional statements as seen in the online lecture engagement on the 25th October, and expect you to be familiar with both loading and 'compiling' (actually, the **Compile** button should read assemble) source code in Komodo. If you are not yet familiar with Komodo I suggest you refer to the video on Moodle outlining how to use Komodo.

Getting Started

You will need to fork and clone the relevant project via `git` in the usual fashion. In this case, the project to fork can be found at:

`https://projects.cs.nott.ac.uk/2021-COMP1006/2021-COMP1006-LabExercise03`

Once cloned, you will find skeleton `.s` files for each exercise. It is **strongly recommended** that you start from these skeletons otherwise your programs may not work against the marking test scripts.

Note: For all these programs you may well find it useful to start by generating a C version (or any other programming language you are familiar with) first to ensure the logic of the program is sorted before you start transliterating it into ARM assembler.

There are two exercises this week, one which is a variation on the classic 'fizz-buzz' problem, and another which implements a very basic piece of text formatting. Full detail of the exercises can be found overleaf.

Assessment Notes

The exercises are starting to get harder this week, and the marking will reflect this. Therefore, you should not necessarily expect to obtain full marks for each exercise — remember the University marking scheme is such that any mark over 40% is a pass mark, and marks over 70% are considered first class. As an example, for the second exercise this week, some of the marks are only available if you complete the advanced variant of the exercise.

Exercise 1 — 'Four Buzz'

In the file `01-fourbuzz.s`, you will find a simple program that prints out all the integers from 1 to `n`, where `n` is read in from memory using `LDR`. For this exercise, you are to modify the program so that *if* the number to be printed (using `SWI 4`) is wholly divisible by four (i.e. the remainder after dividing by four is zero) then you should print out the string '`Buzz`' instead (the string is defined for you in the skeleton provided). Secondly, every fourth time you print out '`Buzz`' your program should actually print out '`Fourth Buzz`' instead.

An example expected output for the program is shown in Appendix A.

Some hints for completing this exercise:

- You will need to implement the equivalent of an `if` statement in this exercise, so it might help to refer back to the lecture engagement from 25/10/2021. You'll probably need to use nested `if` statements so make sure you use unique labels…
- You will need to calculate the remainder of dividing by four. ARM Assembler has no divide (or modulo) instruction, but fortunately, because four is a power of two, we can use the `AND` instruction to simulate it. The following instruction will calculate the remainder of dividing `R1` by three (you can change the registers to suit your program:
  ```
  AND R2,R1, #3
  ```
- You will need a counter to count the number of times your print 'Buzz'

Exercise 2 — Text printing

The file 02-print.s contains a simple program that repeatedly reads in a character (using `SWI 1`) and then prints it out (using `SWI 0`) until the hash character ('`#`') is typed, at which point the program quits.

Currently, the program just continually prints the characters one after another. You are to modify this program so that it enforces a specific line length specified in the variable, `width`. In other words, if `width` is 12, then after 12 characters are printed your program should print a new line character (ASCII code 10) *before* the thirteenth character is printed to force the text onto a new line. The program should then continue outputting characters on that line until 12 characters have been output and so on… Appendix B contains sample input and output for this exercise.

Some hints for completing this exercise:

- You will need to read the value of `width` using `LDR`. Your program should work for any positive integer value of `width` greater than zero.
- You will need to keep count of the number of characters that have been input, and *if* twelve characters have already been printed then you will need to print a newline character.
- You can print a newline character using `SWI 0` (set `R0` to have 10 in it), but remember you also need to print the character that was read in with `SWI 1` (`SWI 1` also puts the character read into `R0`).

- As shown in the example, spaces count as characters.
- The user may press the RETURN key when typing, this should still cause the text to start again *from the beginning* of a new line — remember you can have up to `width` characters on a line. If the RETURN key is pressed, then `SWI 1` will return 10 in `R0`.

Advanced Variation

As it stands, the program will break words in half when it reaches width characters, you might want to try developing an advanced variant of the program which waits until the next space character (ASCII code 32) is typed after the `width` characters have printed before moving on to the next line, unless the last character is a space in which case it should print a newline immediately. This extra space character should not be printed. This will inevitably lead to some lines that are longer than `width` characters but will give an easier to read output.

Consult Appendix C for the output of this advanced variant.

APPENDIX A

Four Buzz output:

```
1
2
3
Buzz
5
6
7
Buzz
9
10
11
Buzz
13
14
15
Fourth Buzz
17
18
19
Buzz
21
22
23
Buzz
25
…
```

APPENDIX B

Example output for the Text printer program, given the typed input:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ultricies porttitor tellus congue semper.

The program should print (note the space at the end of the first line):

```
Lorem ipsum 
dolor sit am
et, consecte
tur adipisci
ng elit. Aen
ean ultricie
s porttitor 
tellus congu
e semper.
```

APPENDIX C

Example output for the advanced variant Text printer program, given the typed input:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean ultricies porttitor tellus congue semper.

The program should print (note the space at the end of the first line):

```
Lorem ipsum 
dolor sit amet,
consectetur
adipiscing elit.
Aenean ultricies
porttitor tellus
congue semper.
```