COS10004 Computer Systems
Assignment 2 Part B

Daniel Coady (102084174) – 12:30 Wednesday

26/10/2019

Summary

This report covers the custom application that I have created as part of part B in assignment 2. The application is an implementation of fizzbuzz in ARMv7 assembly that runs under an operating system, which in this case was tested on Raspbian. Fizzbuzz is a children's game where you count from 1 to n with the following rules:

- If the number is divisible by 3, you must say fizz
- If the number is divisible by 5, you must say buzz
- If the number is divisible by 3 and 5, you must say fizzbuzz

For the sake of simplicity since this would be out of scope otherwise, I have also decided to simply print out "number" instead of the number itself.

Example Output

The progam when run will have an output similar to this:

```
number
number
fizz
number
buzz
```

Implementation

I have approached this task with the following ideas:

- We will loop from 1 to 30
- We will store our "state" in r1
- The states will range from 0-3
  - 0 is a number
  - 1 is fizz
  - 2 is buzz
  - 3 is fizzbuzz
- States will dictate what it is that we print to screen

With this in mind, we perform the following actions on every loop through:

Checking State

First we check if the number is divisible by the given number. To achieve this I have created the following function:

```
// parameters:
// r0 = number
// r1 = divisor
//
// return values:
// r3 = true (1) false (0)
isdivisible:
  push {r0, r2}
  udiv r2, r0, r1 // store division result in r2
  mul r2, r1      // multiply result by divisor
  mov r3, #0      // store a false return value
  cmp r2, r0      // check if r2 == r0
  addeq r3, #1    // add one to our return value if they are equal
  pop {r0, r2}
  bx lr
```

This function is essentially just an implementation of the modulus operator in other languages. The basic idea of it is:

- Get the number and the divisor
- Divide the number by the divisor
- Multiply the result by the divisor
- Compare the number against the result
  - If equal, it is divisible
  - If not equal, it is not divisible

This function forms the backbone of the application, being used in both functions checkfizz and checkbuzz. Both checkfizz and checkbuzz do as their names imply.

```
// parameters:
// r0 = number
// r1 = current result
//
// return values:
// r1 = result
checkfizz:
  push {lr}
  push {r0, r1}
  mov r1, #3
  bl isdivisible // check if r0 is divisible by 3
  pop {r0, r1}
  cmp r3, #1
  addeq r1, #1   // add 1 to result if is divisible
  pop {lr}
  bx lr
```

Notice that if it does end up passing the test that we add 1 to r1. This is the aforementioned state that we store. The state starts off at 0 and with each passing test we add more to it. So if we pass the fizz test then we add 1 to it and if we pass the buzz test we add 2. With this system we are able to accurately track what we should be printing to the screen and even make easier changes to it in the future if we want to.

Printing State

Just like when checking state, when printing state we have a single function that forms the backbone of this functionality:

```
print:
  mov r7, #4  // sys_write
  mov r0, #1  // stdout
  swi 0
  bx lr
```

This is a significantly simpler function, simply setting the appropriate registers to the correct values so that we are able to write to stdout. This function does not work on it's own however and must be called by one of the other functions up one level of abstraction from it:

```
sayfizz:
  push {lr}
  ldr r1, =fizz
  mov r2, #5 // length of word
  bl print
  pop {lr}
  bx lr
```

Here we set the word specific things such as the ASCII array to print out and the length of the ASCII array. From here we can call the print function which will then print our desired word to the shell.