CS1021 Tutorial #8
Bit-Wise Operations

1 Basic Bit Manipulation

Show how you would perform each of the following bit manipulation operations in an ARM Assembly Language program:

(i) Clear all bits in the least significant byte of the word in `R0`

(ii) Clear bits 4, 7 and 12 of the word in `R4`

(iii) Invert the most significant bit of the word value in `R2`

(iv) Invert the middle two bytes of the word in `R3`.

(v) Set bits 2, 3 and 4 of the word in `R5`

(vi) Swap the most and least significant bytes of the word in `R3`

(vii) Replace the 2^{nd} least significant bytes of the word in `R4` with the value 0x44.

2 Shift-and-Add Multiplication by a Constant

In ARM Assembly Language, we can shift a binary value left or right by a specified number of bits by combining a shift operation with another instruction, such as move. For example, to shift the value in R1 left by two bits:

```
MOV R1, R1, LSL #2
```

It is always the last operand that the shift operation is applied to. For example, the following instruction adds R1 to R2 after shifting the value in R2 right by one bit, before storing the result in R0:

```
ADD R0, R1, R2, LSR #1
```

We can express multiplication by any value as the sum of the results of multiplying the value by different powers of 2. For example:

$$a \times 12 = a \times (8 + 4) = a \times (2^3 + 2^2) = (a \times 2^3) + (a \times 2^2)$$

Multiplication of a value by 2^n can be implemented efficiently by shifting the value left by n bits. For example:

$$a \times 12 = (a \ll 3) + (a \ll 2)$$

Design and write ARM Assembly Language programs that will multiply the value in `R1` by:

(a) 10

(b) 15

(c) 17

(d) 25

(e) 100

The result of each multiplication operation should be stored in `R0`.

3 64-bit Shift

Design and write an ARM Assembly Language program that will perform a logical shift operation on a 64-bit value stored in registers `R0` and `R1`. Assume that the least-significant 32-bits are stored in `R0` and the most-significant 32-bits are stored in `R1`. The count of the number of bits to shift is stored in `R2`. If the count in `R2` is negative, your program should shift left. Conversely, if the count is positive, your program should shift right.

The following example shows how your program should logically shift left the 64-bit value stored in `R0` and `R1` when `R2` contains -2 (`0xFFFFFFFD` using the 2s Complement representation of negative values.)