CS1021 Tutorial #9
Bit Manipulation

1 Basic Bit Manipulation

Show how you would perform each of the following bit manipulation operations in an ARM Assembly Language program. (Note: the syntax **j:i** is sometimes used to denote the **end:start** (inclusive) of a sequence of bits (a *bit field*) in a larger value.)

(i) Clear bits 7:4 of `R0`

(ii) Clear the first and last bytes of `R1`

(iii) Invert the most significant bit of `R2`

(iv) Assume `R3`, contains an alphabetic character (A–Z, a–z); invert the case (i.e. if it upper case, change it to lower case and if it is lower case, change it to upper case)

(v) Set bits 4:2 of the word in `R4`

(vi) Swap the most and least significant bytes of the word in `R5`

(vii) Replace bits 15:8 in `R6` with the value 0x44.

2 Shift-and-Add Multiplication by a Constant

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:

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

(Hint: You can quickly see the powers of two that are needed by inspecting the (binary) multiplier!)

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 Shift-And-Add Multiplication by a Variable

Design and write an ARM Assembly Language Program that will use shift-and-add multiplication to multiply the value in R1 by the value in R2, storing the result in R0. You may ignore any overflow problems.

4 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.)