Chapter 7 problems

1. Write functions in ARM Cortex-M4 assembly language that implement the following 64-bit shifts. Write a C program to test your function. The function prototypes are:

```
(d)   uint64_t ROR64(uint64_t u64) ;   // See Listing 7-1, page 118:

ROR64:  // R1.R0 = u64
        LSRS  R1,R1,1              // R1 ← MSHalf(u64) >> 1, C ← lsb
        ORR   R1,R1,R0,LSL 31      // R1[31] ← R0[0]
        RRX   R0,R0                // R0[31] ← C, and ...
        BX    LR                   // R0[30..0] ← (LSHalf(u64) >> 1)
```

3. Write a function in ARM Cortex-M4 assembly language similar to what the Bit-Field Clear (BFC) instruction does. The function returns its first parameter, but with 0's inserted starting at a bit position given by the second parameter and a field width in bits specified by the third parameter. Write a C program to test your function. The function prototype is:

```
(b)     uint32_t BFI(uint32_t x, uint32_t y, uint32_t lsb, uint32_t len) ;

BFI:    // R0 = x, R1 = y, R2 = lsb, R3 = len
        LDR   R12,=1
        LSL   R3,R12,R3   // R3 = 0…010…0  (R3 = 1 << len)
        SUB   R3,R3,1     // R3 = 0…001…1   (R3 = len 1's)
        LSL   R3,R3,R2    // R3 now has 1's in bitfield
        BIC   R0,R0,R3    // zero's into bitfield of x
        AND   R1,R1,R3    // zero's into other bits of y
        ORR   R0,R0,R1    // insert bits of y into x
        BX    LR          // and return with result in R0

(c)     int32_t SBFX(uint32_t x, uint32_t lsb, uint32_t len) ;

SBFX:   // R0 = x, R1 = lsb, R2 = len
        RSB   R3,R2,32    // R3 = 32 - lsb
        SUB   R1,R3,R2    // R1 = 32 – (lsb + len)
        LSL   R0,R0,R1    // shift bitfield into MS bits
        ASR   R0,R0,R3    // ASR bitfield to LS bits
        BX    LR          // and return with result in R0
```

Chapter 8 problems:

1. You can multiply register R0 by the binary constant 01011110 using 5 shifts and 4 additions. However, you can reduce the total number of operations if you also use subtractions. Give a minimal length sequence of ARM Cortex-M4 instructions to do this. *(Note: This can be done in 3 instructions.)*

```
// Note: 01011110₂ = 94₁₀ = 64 + 32 - 2

LSL   R1,R0,6          // R1 ← 64*R0
ADD   R1,R1,R0,LSL 5   // R1 ← 64*R0 + 32*R0
SUB   R0,R1,R0,LSL 1   // R0 ← 64*R0 + 32*R0 – 2*R0
```

3. Suppose you need to divide an unsigned 8-bit integer variable X by 9, but there is no divide instruction. If you use reciprocal multiplication, what constant should you multiply times X?

$2^8/9 = 256/9 = 28.444444 \rightarrow$ use 28

7. Without using a divide instruction, write an assembly language function to compute the <u>modulus</u> of its first parameter (which may be positive or negative) with respect to 2^k, where k is specified by the second parameter. ~~Write a C program to test your function~~. The function prototype is:

```
uint32_t Modulus(int32_t s32, uint32_t k) ;
```

```
Modulus:  LDR   R2,=1       // R2 = 1
          LSL   R2,R2,R1    // R2 = 10…0
          SUB   R2,R2,1     // R2 = 01…1
          AND   R0,R0,R2
          BX    LR
```

Note: Modulus != Remainder when N<0:

N	N % 4 (Remainder)	N mod 4 (N & (4-1))
5	1	1
4	0	0
3	3	3
2	2	2
1	1	1
0	0	0
-1	-1	3
-2	-2	2
-3	-3	1
-4	0	0
-5	-1	3