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:

(a) `uint64_t LSL64(uint64_t u64) ;`

```
LSL64:  // R1.R0 = u64
        LSLS  R0,R0,1             // R0 ← LSHalf(u64) << 1, C ← msb
        ADC   R1,R1,R1            // R1 ← (MSHalf(u64) << 1) + C
        BX    LR
```

(b) `uint64_t LSR64(uint64_t u64) ;`

```
LSR64:  // R1.R0 = u64
       LSRS  R1,R1,1              // R1 ← MSHalf(u64) >> 1, C ← lsb
       RRX   R0,R0                // R0[31] ← C, and ...
       BX    LR                   // R0[30..0] ← (LSHalf(u64) >> 1)
```

(c) `int64_t ASR64(int64_t u64) ;   // corrected relative to text`

```
ASR64:  // R1.R0 = s64
        ASRS  R1,R1,1             // Similar to LSR64 (above)
        RRX   R0,R0
        BX    LR
```

(d) `uint64_t ROR64(uint64_t u64) ;`

```
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)
```

2. Write a function in ARM Cortex-M4 assembly language that returns the negative of its argument without using any subtract, negate, multiply or divide instructions. Write a C program to test your function. The function prototype is:

`int32_t Negate(int32_t s32) ;`

```
Negate: // R0 = s32
        MVN   R0,R0               // R0 ← ~s32
        ADD   R0,R0,1             // R0 ← ~s32 + 1
        BX    LR
```

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:

   (a) `uint32_t BFC(uint32_t x, uint32_t lsb, uint32_t len) ;`

```
BFC:    // R0 = x, R1 = lsb, R2 = len
        LDR    R3,=1                  // R3 = 1
        LSL    R1,R3,R1               // R1 = 1 << lsb
        LSL    R2,R1,R2               // R2 = 1 << (lsb + width)
        SUB    R1,R2,R1               // R1 = 1's in bitfield
        BIC    R0,R0,R1               // clear bitfield
        BX     LR                     // return result in R0
```

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

   This is part of a lab assignment.

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

   This is similar to part of a lab assignment.

   (d) `uint32_t UBFX(uint32_t x, uint32_t lsb, uint32_t len) ;`

   This is part of a lab assignment.

4. Write a function in ARM Cortex-M4 assembly language that returns the number of bits within it parameter that are surrounded with leading and trailing 0's. For example, if the parameter was $006203F0_{16}$, the function would return the value 13. Write a C program to test your function. The function prototype is:

   `uint32_t Span(uint32_t x) ;`

```
Span: // R0 = x
      CLZ   R1,R0       // Count the number of leading zeroes
      RBIT  R0,R0       // Reverse all the bits in the register
      CLZ   R0,R0       // Count the number of trailing zeroes
      ADD   R0,R0,R1    // Add the leading and trailing zeroes
      RSB   R0,R0,32    // subtract the total from 32
      BX    LR
```

5. Write a function in ARM Cortex-M4 assembly language similar to what the Reverse Byte Order (etc) instruction does, but without using that instruction. Write a C program to test your function. The function prototype is:

```
uint32_t REV(uint32_t x) ;
```

```
      REV:  // R0 = x
            UBFX  R1,R0,0,8          // get the least significant byte (#0)
            BFI   R2,R1,24,8         // insert into most sig position (#3)
            UBFX  R1,R0,8,8          // get byte #1
            BFI   R2,R1,16,8         // insert into position #2
            UBFX  R1,R0,16,8         // get byte #2
            BFI   R2,R1,8,8          // insert into position #1
            UBFX  R1,R0,24,8         // get byte #3
            BFI   R2,R1,0,8          // insert into position #0
            MOV   R0,R2              // make a copy of parameter x
            BX    LR
```

6. Write a function in ARM Cortex-M4 assembly language that given an address of a byte in the bit-band region of memory, returns the address of the word containing bit 0 of that byte in the bit-band alias. Write a C program to test your function. The function prototype is:

```
uint32_t *BitBandAlias(uint8_t *pByte) ;
```

```
BitBandAlias:     // R0 = pByte
            SUB   R0,R0,0x20000000  // compute bit band offset of byte
            LSL   R0,R0,5           // position offset in bit band alias address
            ADD   R0,R0,0x22000000  // add starting adrs of bit band alias region
            BX    LR
```