Xavier Kuehn

1. Write n-bit shift functions in assembly language.

```
uint64_t LSL64(uint64_t x, int n);
LSL64:     LSL   R1,R1,R2
           SUB   R3,32,R2
           ADD   R1,R1,R0,LSL R3
           LSL   R0,R0,R2
           BX    LR

uint64_t LSR64(uint64_t x, int n);
LSR64:     LSR   R0,R0,R2
           SUB   R3,32,R2
           ADD   R0,R0,R1,LSL R3
           LSR   R1,R1,R2
           BX    LR

int64_t ASR64(int64_t x, int n);
ASR64:     LSR   R0,R0,R2
           SUB   R3,32,R2
           ADD   R0,R0,R1,LSL R3
           ASR   R1,R1,R2
           BX    LR

uint64_t ROR64(uint64_t x, int n);
ASR64:     PUSH  {R4}
           SUB   R3,32,R2
           LSL   R4,R0,R3
           LSR   R0,R0,R2
           ADD   R0,R0,R1,LSL R3
           LSR   R1,R1,R2
           ADD   R1,R1,R4
           POP   {PC}
```

2. Write the assembly language equivalent of the following C function.

```
int32_t Negate(int32_t s32);
Negate:    RSB         R2,R0,R0,LSR 31
           LDR         R1,=1
           AND         R1,R1,R0,LSR 31
           CMP         R1,1
           IT          EQ
```

```
            ADDEQ       R2,R2,1
            MOV         R0,R2
            BX          LR
```

3. Write assembly language equivalents of the following C functions.

```
uint32_t BFC(uint32_t x, uint32_t lsb, uint32_t len);
BFC:  ADD   R2,R1,R2
      LDR   R3,=(1<<R2)-(1<<R1)
      BIC   R0,R0,R3
      BX    LR

uint32_t BFI(uint32_t x, uint32_t y, uint32_t lsb, uint32_t len);
BFI:  PUSH  {R4-R6}
      ADD   R4,R2,R3
      LDR   R5,=(1<<R4)-(1<<R2)
      BIC   R0,R0,R5              // x bits cleared, ready to insert
      AND   R6,R1,R5              // desired bits from y in R6
      ADD   R0,R0,R6              // add x + R6
      POP   {PC}

uint32_t SBFX(uint32_t x, uint32_t lsb, uint32_t len);
SBFX: PUSH  {R4}
      ADD   R4,R1,R2
      LDR   R3,=(1<<R4)-(1<<R1)
      AND   R3,R0,R3              // desired bits in R3
      LSR   R3,R3,R1              // desired bits moved to LSB
      SXTH  R0,R3                 // sign extend
      POP   {PC}

uint32_t UBFX(uint32_t x, uint32_t lsb, uint32_t len);
UBFX: PUSH  {R4}
      ADD   R4,R1,R2
      LDR   R3,=(1<<R4)-(1<<R1)
      AND   R3,R0,R3              // desired bits in R3
      LSR   R3,R3,R1              // desired bits moved to LSB
      UXTH  R0,R3                 // zero extend
      POP   {PC}
```

4. Write the following C function in assembly language.

```
uint32_t Span(uint32_t x);
while (x[ith bit(0>32)] == 0) ++count;
while (x[ith bit(32>0)] == 0) ++count;
Span: LDR        R1,=0           // bit index value
```

```
      LDR       R3,=0           // non-digit counter

L1:   AND       R2,R0,(1<<R1)   // get R0 bit value at R1 (0-31)
      CMP       R1,31           // compare R1 and 31
      BHI       L2              // end loop if R1 > 31
      CMP       R2,1            // compare R2 and 1
      BEQ       L2              // exit loop at first bit=1 in R0
      ADD       R1,R1,1         // else add 1 to R1 to check next bit
      ADD       R3,R3,1         // add 1 to counter
      B         L1              // continue next iteration of loop

L2:   CMP       R3,32           // compare counter and 32
      BEQ       L4              // goto L4 (span calculation)
      MOV       R1,31           // set bit index value to 31

L3:   AND       R2,R0,(1<<R1)   // get R0 bit value at R1 (31-0)
      CMP       R1,0            // compare R1 and 0
      BLO       L4              // end loop if R1 < 0
      CMP       R2,1            // compare R2 and 1
      BEQ       L4              // exit loop at first 1 bit in R0
      SUB       R1,R1,1         // else R1 - 1 to check next bit
      ADD       R3,R3,1         // add 1 to counter
      B         L3              // continue to next iteration of loop

L4:   SUB       R0,32,R3        // span = 32 - counter
      BX        LR              // return span
```

5. Write a program in assembly language equivalent to the C function below.

```
uint32_t REV(uint32_t x);
while (i >= 0) rev_x[31-i] = x[i];
REV:  PUSH  {R4}
      MOV   R1,0                // R1 is reverse copy of x
      MOV   R2,31               // bit index from 31 -> 0

L1:   CMP   R2,0                // compare index and 0
      BLO   L2                  // exit loop if index < 0
      AND   R3,R0,(1<<R2)       // puts R0[R2] bit into R3
      SUB   R4,31,R2            // insert index = 31 - index
      ADD   R1,R1,R3,LSL R4     // R1[31 - index] = R0[index]
      SUB   R2,R2,1             // end of body, --index
      B     L1                  // go to start of loop

L2:   MOV   R0,R1               // move rev_x into R0
      POP   {PC}                // return
```