1. Assembly
    a. LSL64:

```
CMP     R1, 64
MOVHI   R0, 0
MOVS    R0, R0, LSL R1
BX      LR
```

    b. LSR64:

```
CMP     R1, 64
MOVHI   R0, 0
MOVS    R0, R0, LSR R1
BX      LR
```

    c. ASR64:

```
CMP     R1, 64
MOVHI   R0, 0
MOVS    R0, R0, ASR R1
BX      LR
```

    d. ROR64:

```
ANDS    R1, R1, 63
LSRS    R2, R0, R1
LSL     R1, #32
SUBS    R1, #32, R1
ORRS    R0, R2, R0, LSR R1
BX      LR
```

C Program

```
#include <stdio.h>
#include <stdint.h>

extern uint64_t LSL64(uint64_t x, int n);
extern uint64_t LSR64(uint64_t x, int n);
extern int64_t ASR64(uint64_t x, int n);
extern uint64_t ROR64(uint64_t x, int n);

int main() {
   uint64_t x = 0x123456789abcdef0;
   int n = 4;

   uint64_t result_lsl64 = LSL64(x, n);
   uint64_t result_lsr64 = LSR64(x, n);
   int64_t result_asr64 = ASR64(x, n);
   uint64_t result_ror64 = ROR64(x, n);
```

```
    printf("LSL64: 0x%016llX\n", result_lsl64);
    printf("LSR64: 0x%016llX\n", result_lsr64);
    printf("ASR64: 0x%016llX\n", result_asr64);
    printf("ROR64: 0x%016llX\n", result_ror64);

    return 0;
}
```

2. Assembly

```
.global Negate
.type Negate, %function

Negate:
        mvn     r0, r0
        adds    r0, r0, 1
        bx      lr
```

C Program

```
#include <stdio.h>
#include <stdint.h>

extern int32_t Negate(int32_t s32);

int main() {
    int32_t num = 42;
    int32_t neg = Negate(num);

    printf("Original: %d\n", num);
    printf("Negate: %d\n", neg);

    return 0;
}
```

3. Assembly
    a. BFC:

```
        LDR     R3, [SP, 8]
        LDR     R2, [SP, 12]
        LDR     R1, [SP, 16]
        MVN     R0, 0
        LSL     R0, R0, R1
```

```
LSRS    R0, R0, R2
BIC     R3, R3, R0
STR     R3, [SP, 8]
BX      LR
```

b. BFI:

```
LDR     R3, [SP, 8]
LDR     R2, [SP, 12]
LDR     R1, [SP, 16]
LSLS    R2, R2, R1
LDR     R0, [SP, 20]
LSL     R0, R0, R2
MVN     R2, 0
LSL     R2, R2, R1
LSRS    R2, R2, 31
ORR     R3, R3, R2
STR     R3, [SP, 8]
BX      LR
```

c. SBFX:

```
LDR     R3, [SP, 8]
LDR     R2, [SP, 12]
LDR     R1, [SP, 16]
ADDS    R2, R2, R1
LSLS    R3, R3, 32
ASRS    R3, R3, 32
LSLS    R3, R3, R2
ASRS    R3, R3, R2
LSL     R3, R3, 32
ASR     R3, R3, 32
BX      LR
```

d. UBFX:

```
LDR     R3, [SP, 8]
LDR     R2, [SP, 12]
LDR     R1, [SP, 16]
ADDS    R2, R2, R1
LSLS    R3, R3, R2
LSRS    R3, R3, R2
BX      LR
```

C Program

```
#include <stdio.h>
```

```
#include <stdint.h>

extern void BFC(uint32_t *value, int bitPos, int bitWidth);
extern void BFI(uint32_t *value, int bitPos, int bitWidth, uint32_t srcValue);
extern int32_t SBFX(int32_t value, int bitPos, int bitWidth);
extern uint32_t UBFX(uint32_t value, int bitPos, int bitWidth);

int main() {
   uint32_t value = 0xABCDEF12;
   uint32_t srcValue = 0x12345678;
   int bitPos = 8;
   int bitWidth = 12;

   printf("Original value: 0x%08X\n", value);

   BFC(&value, bitPos, bitWidth);
   printf("After BFC: 0x%08X\n", value);

   BFI(&value, bitPos, bitWidth, srcValue);
   printf("After BFI: 0x%08X\n", value);

   int32_t sbfxResult = SBFX((int32_t)value, bitPos, bitWidth);
   printf("SBFX Result: 0x%08X\n", sbfxResult);

   uint32_t ubfxResult = UBFX(value, bitPos, bitWidth);
   printf("UBFX Result: 0x%08X\n", ubfxResult);

   return 0;
}
```

5. Assembly

```
REV:
        LDR     R1, [SP, 4]
        MOV     R0, 0
        MOV     R2, 0

        LDRB    R3, [R1, R2]
        LSL     R3, R3, 24
        ORR     R0, R0, R3

        LDRB    R3, [R1, R2, LSL 8]
```

```
        LSL     R3, R3, 16
        ORR     R0, R0, R3

        LDRB    R3, [R1, R2, LSL 16]
        LSL     R3, R3, 8
        ORR     R0, R0, R3

        LDRB    R3, [R1, R2, LSL 24]
        ORR     R0, R0, R3

        BX      LR
```

C Program

```
#include <stdio.h>
#include <stdint.h>

extern uint32_t REV(uint32_t x);

int main() {
   uint32_t x = 0x12345678;
   uint32_t reversed = REV(x);

   printf("Original value: 0x%08X\n", x);
   printf("Reversed value: 0x%08X\n", reversed);

   return 0;
}
```