

1. Translate each of the following C function calls into a sequence of ARM Cortex-M4 Instructions. Assume that all constants and variables are of type `int32_t`.

(a) `f1(a) ;`

LDR R0,a // R0 <-- a
BL f1

(b) `f2(&a) ;`

ADR R0,a // R0 <-- &a
BL f2

(c) `f3(a, b) ;`

LDR R0,a
LDR R1,b
BL f3

(d) `b = f4() ;`

BL f4
STR R0,b // R0 --> b

2. Translate each of the following C function calls into a sequence of ARM Cortex-M4 Instructions. Assume that all constants and variables are of type `uint64_t`.

(a) `g1(a) ;`

LDRD R0,R1,a // R1.R0 <-- a
BL g1

(b) `g2(&a) ;`

ADR R0,a // Although a is 64 bits, &a is 32 bits
BL g2

(c) `g3(a, b) ;`

LDRD R0,R1,a
LDRD R2,R3,b
BL g3

(d) `b = g4() ;`

```
BL      g4
STRD   R0,R1,b
```

3. Translate each of the following C function calls into a sequence of ARM Cortex-M4 Instructions. Assume that all constants and variables are of type `int8_t`.

(a) `h1(a) ;`

```
LDRSB  R0,a          // LDRSB = Load Register with Signed Byte
BL     h1
```

(b) `h2(&a) ;`

```
ADR    R0,a          // Although a is 8 bits, &a is 32 bits
BL     h2
```

(c) `h3(a, b) ;`

```
LDRSB  R0,a
LDRSB  R1,b
BL     h3
```

(d) `b = h4() ;`

```
BL     h4
STRB  R0,b          // STRB = Store Register to Byte
```

4. Translate each of the following functions into ARM Cortex-M4 assembly language:

(a) `uint64_t f4(uint32_t u32)`
`{`
 `return (uint64_t) u32 ;`
`}`

```
f4:   LDR   R1,=0        // R1 = 32-bit extension to R0
      BX    LR
```

```
(b) int32_t f5(int32_t a, int32_t b)
{
    // Prototype declaration
    int32_t f6(int32_t) ;

    return f6(a) + f6(b) ;
}

f5:    PUSH   {R4,R5,LR}
        MOV    R4,R1          // R4 is a copy of b
        BL     f6             // Call f6(a).  R0 = a
        MOV    R5,R0          // Preserve f6(a) in R5
        MOV    R0,R4          // R0 = b
        BL     f6             // Call f6(b)
        ADD    R0,R0,R5        // R0 = f6(a) + f6(b)
        POP    {R4,R5,PC}
```



```
(c) uint32_t f7(uint32_t a, uint32_t b)
{
    // Prototype declaration
    uint32_t f8(uint32_t, uint32_t) ;

    return f8(b, a) ;
}

f7:    MOV    R2,R0          // Swap R0 and R1
        MOV    R0,R1          // using R2 as a
        MOV    R1,R2          // temporary
        B     f8             // R0 <- f8(b,a)
```



```
(d) int32_t f9(int32_t a)
{
    // Prototype declaration
    int8_t f10(int8_t) ;

    return a + (int32_t) f10(0) ;
}

f9:    PUSH   {R4,LR}
        MOV    R4,R0          // Preserve a in R4
        LDR    R0,=0          // Prepare parameter for f10
        BL     f10            // R0 <- f10(0)
        ADD    R0,R0,R4        // R0 = f10(0) + a
        POP    {R4,PC}
```

```
(e) uint64_t f11(uint32_t a)
{
    // Prototype declaration
    uint32_t f12(uint32_t) ;

    return (uint64_t) f12(a + 10) ;
}

f11:  PUSH   {LR}
        ADD    R0,R0,10      // R0 <-- a + 10
        BL     f12           // R0 <-- f12(a + 10)
        LDR    R1,=0
        POP    {PC}
```