2. Given the C declaration statements,

```
uint8_t    u8  ;
uint16_t   u16 ;
uint32_t   u32 ;
uint64_t   u64 ;
```

Translate each of the following C assignment statements into ARM Cortex-M4 assembly:

(a) u8 = 0 ;

```
LDR    R0,=0
STRB   R0,u8
```

(b) u16 = 0 ;

```
LDR    R0,=0
STRH   R0,u16
```

(c) u32 = 0 ;

```
LDR    R0,=0
STR    R0,u32
```

(d) u64 = 0 ;

```
LDR    R0,=0
STRD   R0,R0,u64
```

The same register can be used to provide both the least and most-significant halves of the 64-bit value if those bit patterns are identical

(e) u16 = u8 ;

```
LDRB   R0,u8
STRH   R0,u16
```

(f) u32 = u8 ;

```
LDRB   R0,u8
STR    R0,u32
```

(g) u32 = u16 ;

```
LDRH   R0,u16
STR    R0,u32
```

(h) u64 = u32 ;

```
LDR    R0,u32
LDR    R1,=0
STRD   R0,R1,u64
```

(i) u8 = u32 ;

```
LDR    R0,u32
STRB   R0,u8
```

(j) u8 = u16 ;

```
LDRH   R0,u16
STRB   R0,u8
```

(k) u16 = u32 ;

```
LDR    R0,u32
STRH   R0,u16
```

3. Given the C declaration statements,

```
int8_t    s8 ;
int16_t   s16 ;
int32_t   s32 ;
int64_t   s64 ;
```

Translate each of the following C assignment statements into ARM Cortex-M4 assembly:

(a) s8 = -1 ;

```
LDR    R0,=-1
STRB   R0,s8
```

(b) s16 = -1 ;

```
LDR    R0,=-1
STRH   R0,s16
```

(c) s32 = -1 ;

```
LDR    R0,=-1
STR    R0,s32
```

(d) s64 = -1 ;

```
LDR    R0,=-1
STRD   R0,R0,s64
```

The same register can be used to provide both the least and most-significant halves of the 64-bit value if those bit patterns are identical

(e) s16 = s8 ;

```
LDRSB  R0,s8
STRH   R0,s16
```

(f) s32 = s8 ;

```
LDRSB  R0,s8
STR    R0,s32
```

(g) s32 = s16 ;

```
LDRSH  R0,s16
STR    R0,s32
```

(h) s64 = s32 ;

```
LDR    R0,s32
ASR    R1,R0,31
STRD   R0,R1,s64
```

ASR shifts R0 right 31 bit positions. Each 1-bit shift preserves the value of the most-significant bit. When done, R1 will be all 1's if R0 < 0, else all 0's.

(i) s8 = s32 ;

```
LDR    R0,s32
STRB   R0,s8
```

(j) s8 = s16 ;

```
LDRSH  R0,s16
STRB   R0,s8
```

(k) s16 = s32 ;

```
LDR    R0,s32
STRH   R0,s16
```

4. Given the C declaration statements,

```
int8_t   s8, *ps8 ;
int32_t s32, *ps32 ;
```

Translate each of the following C assignment statements into ARM Cortex-M4 assembly:

(a) ps32 = &s32 ;

```
ADR   R0,s32
STR   R0,ps32
```

(b) ps8 = &s8 ;

```
ADR   R0,s8
STR   R0,ps8
```

(c) ps8++ ;

```
LDR   R0,ps8
ADD   R0,R0,1
STR   R0,ps8
```

ps8 is a pointer to an 8-bit byte. Adding 1 to such a pointer requires adding 1 to the address it holds.

(d) ps32++ ;

```
LDR   R0,ps32
ADD   R0,R0,4
STR   R0,ps32
```

ps32 is a pointer to a 32-bit word of 4 bytes. Adding 1 to such a pointer requires adding 4 to the address it holds.

(e) *ps32 = 0 ;

```
LDR   R0,=0
LDR   R1,ps32
STR   R0,[R1]
```

(f) *(ps8 + 1) = 0 ;

```
LDR   R0,=0
LDR   R1,ps8
STRB  R0,[R1,1]
```

(g) *(ps32 + 1) = 0 ;

```
LDR   R0,=0
LDR   R1,ps32
STR   R0,[R1,4]
```

(h) *(ps32 + s32) = 0 ;

```
LDR   R0,=0
LDR   R1,ps32
LDR   R2,s32
STR   R0,[R1,R2,LSL 2]
```

The address of the operand is computed as R1 + R2 << 2, which is equivalent t o R1 + 4*R2

(i) ((int8_t *) &s32)[1] = 0 ;

```
LDR   R0,=0
ADR   R1,s32
STRB  R0,[R1,1]
```

(int8_t *) is a cast operator. It causes "&s32" to be interpreted as the address of an 8-bit signed byte. Thus the subscript causes 0 to be stored in the next byte rather than the next 32-bit word.

7. `void Swap(int32_t *p1, int32_t *p2) ;`

```
Swap: // R0 = p1, R1 = p2
      LDR   R2,[R0]    // R2 = *p1
      LDR   R3,[R1]    // R3 = *p2
      STR   R2,[R1]    // *p2 = R2
      STR   R3,[R0]    // *p1 = R3
      BX    LR
```