CS1021 Tutorial #8 Solution
Bit-Wise Operations

1 Basic Bit Manipulation

(i) Use AND or BIC.

```
    LDR R9, =0xFFFFFF00      ; Mask with 0s in the bits we want to clear
    AND R0, R0, R9
```

or

```
    BIC R0, R0, #0x000000FF  ; Mask with 1s in the bits we want to clear
```

(ii) Use AND. (Mask cannot be represented as an immediate operand so there is no advantage in using BIC.)

```
    LDR R9, =0xFFFFEF6F      ; Mask with 0s in bits 4, 7 and 12
    AND R4, R4, R9
```

(iii) Use EOR.

```
    LDR R9, =0x80000000      ; Mask with a 1 in the MSB and 0s elsewhere
    EOR R2, R2, R9
```

Alternatively, we can specify the mask as an immediate value.

```
    EOR R2, R2, #0x80000000
```

(iv) Use EOR.

```
    LDR R9, =0x00FFFF00      ; Load mask into R9
    EOR R3, R3, R9           ; Apply mask using EOR to invert
```

(v) Use OR.

```
    ORR R5, R5, #0x0000001C  ; Could have used LDR ... ORR but for masks
                             ;   that the assembler can store as an
                             ;   immediate operand, this is a little more
                             ;   efficient
```

(vi) Extract bytes, swap them and merge them into new location.

```
    AND R4, R3, #0x000000FF ; Isolate LS-byte
    AND R5, R3, #0xFF000000 ; Isolate MS-byte
    LDR R9, =0x00FFFF00     ; Clear old LS- and MS-bytes
    AND R3, R3, R9          ;
    MOV R4, R4, LSL #24     ; Move old LS-byte to new MS position
    MOV R5, R5, LSR #24     ; Move old MS-byte to new LS position
    ORR R3, R3, R4          ; Combine new MS byte with middle two bytes
    ORR R3, R3, R5          ; Combine new LS byte to finish
```

or

```
    MOV R4, R3, LSL #24     ; Isolate LS-byte and shift to new position
    MOV R5, R3, LSR #24     ; Isolate MS-byte and shift to new position
    LDR R9, =0x00FFFF00     ; Clear old LS- and MS-bytes
    AND R3, R3, R9          ;
    ORR R4, R4, R5          ; Combine new LS- and MS-bytes
    ORR R4, R3, R4          ; Combine with old value
                            ;   (i.e. middle two bytes)
```

(vii) Clear the 2nd least significant byte and then merge in the new value.

```
    BIC R4, R4, #0x0000FF00 ; Clear 2nd byte
    LDR R5, =0x44           ; Load new value
    ORR R4, R4, R5, LSL #8  ; Combine (using OR), while first shifting new
                            ;   value into correct position (2nd byte)
```

2 Shift-and-Add Multiplication by a Constant

(i) 10

```
    MOV R0, R1, LSL #3      ; a*8
    ADD R0, R0, R1, LSL #1  ; + a*2 = a*10
```

(ii) 15

```
    MOV R0, R1, LSL #3      ; a*8
    ADD R0, R0, R1, LSL #2  ; + a*4 = a*12
    ADD R0, R0, R1, LSL #1  ; + a*2 = a*14
    ADD R0, R0, R1          ; + a = a*15
```

or

```
    RSB R0, R1, R1, LSL #4  ; a*16 - a = a*15
```

(iii) 17

```
    MOV R0, R1, LSL #4      ; a*16
    ADD R0, R0, R1          ; + a = a*17
```

or

```
    ADD R0, R1, R1, LSL #4  ; a + a*16 = a*17
```

(iv) 25 (this could be shortened by one instruction!)

```
    MOV R0, R1, LSL #4      ; a*16
    ADD R0, R0, R1, LSL #3  ; + a*8 = a*24
    ADD R0, R0, R1          ; +a = a*25
```

(v) 100

```
    MOV R0, R1, LSL #6      ; a*64
    ADD R0, R0, R1, LSL #5  ; + a*32 = a*96
    ADD R0, R0, R1, LSL #2  ; + a*4 = a*100
```

3 64-bit Shift

When shifting by *n* bits, *n* bits will need to be moved from one end of R0/R1 to the other end of R1/R0. The direction of the transfer will depend on the direction of the shift.

```
        CMP     R2, #0
        BEQ     shiftEnd
        BLT     shiftLeftN

        ; shift right
        RSB     R4, R2, #32             ; oppShift = 32-n
        MOV     R3, R1, LSL R4          ; tmp = upr << oppShift
        MOV     R0, R0, LSR R2          ; lwr = lwr >> n
        ORR     R0, R0, R3              ; lwr = lwr | tmp
        MOV     R1, R1, LSR R2          ; upr = upr >> n
        B       shiftEnd
shiftLeftN
        ; shift left
        RSB     R2, R2, #0              ; n = -n
        RSB     R4, R2, #32             ; oppShift = 32-n
        MOV     R3, R0, LSR R4          ; tmp = lwr >> oppShift
        MOV     R1, R1, LSL R2          ; upr = upr << n
        ORR     R1, R1, R3              ; upr = upr | tmp
        MOV     R0, R0, LSL R2          ; lwr = lwr << n
shiftEnd
```