CS1021 Tutorial #9 Solution
Bit-Wise Operations

1 Basic Bit Manipulation

(i) Could use AND or BIC here (modifying the mask as appropriate)

```
1    LDR R10, =0xFFFFFF0F    ; Mask with 0s in the bits we want to clear
2    AND R0, R0, R10         ;   (because we're using AND)
```

(ii) Could use AND or BIC here (modifying the mask as appropriate)

```
1    LDR R10, =0xFF0000FF    ; Mask with 1s in the bits we want to clear
2    BIC R1, R1, R10         ;   (because we're using BIC)
```

(iii) EOR with 1s in the positions we want to invert

```
1    LDR R10, =0x00001090    ; Mask with 1s in bits 4, 7 and 12
2    EOR R2, R2, R10
```

(iv) The case of an ASCII alphabetic character is determined only by bit 5

```
1    LDR R10, =0x00000020    ; Mask with a 1 in bit 5
2    EOR R3, R3, R10
```

(v) Use ORR

```
1    LDR R10, =0x0000001C    ; mask with 1s in bits 4:2
2    ORR R4, R4, R10
```

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

```
1    AND R0, R5, #0x000000FF ; Isolate LS-byte
2    AND R1, R5, #0xFF000000 ; Isolate MS-byte
3    LDR R10, =0x00FFFF00    ; Clear old LS- and MS-bytes
4    AND R5, R5, R10         ;
5    MOV R0, R0, LSL #24     ; Move old LS-byte to new MS position
6    MOV R1, R1, LSR #24     ; Move old MS-byte to new LS position
7    ORR R5, R5, R0          ; Combine new MS byte with middle two bytes
8    ORR R5, R5, R1          ; Combine new LS byte to finish
```

or, using LSL/LSR with ORR instead of MOV for fewer instructions

```
1    AND R0, R5, #0x000000FF ; Isolate LS-byte
2    AND R1, R5, #0xFF000000 ; Isolate MS-byte
3    LDR R10, =0x00FFFF00    ; Clear old LS- and MS-bytes
4    AND R5, R5, R10         ;
5    ORR R5, R5, R0, LSL #24 ; Combine new MS byte with middle two bytes
6    ORR R5, R5, R1, LSR #24 ; Combine new LS byte to finish
```

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

```
1    BIC R6, R6, #0x0000FF00 ; Clear 2nd byte
2    LDR R0, =0x44           ; Load new value
3    ORR R6, R6, R0, LSL #8  ; Combine (using OR), while first shifting new
4                            ;   value into correct position (2nd byte)
```

2 Shift-and-Add Multiplication by a Constant

(i) 10

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

(ii) 15

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

or

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

(iii) 17

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

or

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

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

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

(v) 100

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

3 Shift-and-Add Multiplication by a Variable

```
1         MOV     R0, #0                  ; result = 0
2         MOV     R4, R2                  ; tmp = b
3         MOV     R5, #0                  ; count = 0
4         CMP     R4, #0                  ; while (tmp != 0)
5         BEQ     ewhMul                  ; {
6         MOVS    R4, R4, LSR #1          ;   tmp = tmp >> 1 (updating status)
7         BCC     elf1                    ;   if (shifted-out bit was 1) {
8         ADD     R0, R0, R1, LSL R5      ;     result = tesult + (a << count)
9 elf1                                    ;   }
10        ADD     R5, R5, #1              ;   count++
11        B       whMul                   ; }
12 ewhMul                                 ;
```

4 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.

```
1         CMP     R2, #0
2         BEQ     shiftEnd
3         BLT     shiftLeftN
4
5         ; shift right
6         RSB     R4, R2, #32             ; oppShift = 32-n
7         MOV     R3, R1, LSL R4          ; tmp = upr << oppShift
8         MOV     R0, R0, LSR R2          ; lwr = lwr >> n
9         ORR     R0, R0, R3              ; lwr = lwr | tmp
10        MOV     R1, R1, LSR R2          ; upr = upr >> n
11        B       shiftEnd
12 shiftLeftN
13        ; shift left
14        RSB     R2, R2, #0              ; n = -n
15        RSB     R4, R2, #32             ; oppShift = 32-n
16        MOV     R3, R0, LSR R4          ; tmp = lwr >> oppShift
17        MOV     R1, R1, LSL R2          ; upr = upr << n
18        ORR     R1, R1, R3              ; upr = upr | tmp
19        MOV     R0, R0, LSL R2          ; lwr = lwr << n
20 shiftEnd
```