# CS1021 Tutorial 5

## Logic and Shift Instructions

Q1 Calculate, in hexadecimal, the results of the following 8 bit expressions

| | | |
|---|---|---|
| (i) | 0x96 & 0xF0 | 0x90 |
| (ii) | 0x96 \| 0x0F | 0x9F |
| (iii) | 0xAA ^ 0xF0 | 0x5A |
| (iv) | ~0xA5 | 0x5A |
| (v) | 0x96 >> 2 | 0x25 |

and 32-bit expressions

| | | |
|---|---|---|
| (vi) | 0x0123 << 2 | 0x048c |
| (vii) | 0x12345678 >> 24 | 0x12 |
| (viii) | 0x12345678 >> 16 | 0x1234 |
| (ix) | (0x12345678 >> 16) & 0xFF | 0x34 |
| (x) | (0x12345678 & ~0xFF00) \| 0x4400 | 0x12344478 |

Q2 Write ARM Assembly Language instructions to perform the following operations (assume the LSB of a register is bit 0).

(i) clear bits 4 to 7 of R0

```
LDR     R1, =0xFFFFFF0F     ; mask
AND     R0, R0, R1          ; and

or

BIC     R0, R0, #0xF0       ; bit clear
```

(ii) clear the first and last bytes of R0

```
LDR     R1, =0xFF0000FF     ; mask
BIC     R0, R0, R1          ; bit clear
```

(iii) invert the most significant bit of R0

```
EOR     R0, R0, #0x80000000 ; invert MS bit
```

(iv) set bits 2 to 4 of R0

```
ORR     R0, R0, #0x1C       ; or
```

(v) swap the most and least significant bytes of R0

```
AND     R1, R0, #0xFF           ; extract LS byte
AND     R2, R0, #0xFF000000     ; extract MS byte
BIC     R0, R0, #0xFF           ; clear LS byte
BIC     R0, R0, #0xFF000000     ; clear MS byte
ORR     R0, R0, R1, LSL #24     ; insert extracted and shifted LS byte into MS byte
ORR     R0, R0, R2, LSR #24     ; insert extracted and shifted MS byte into LS byte
```

(vi) replace bits 8 to 15 in R0 with the value 0x44

```
BIC     R0, R0, #0xFF00         ; clear bits
ORR     R0, R0, #0x4400         ; insert 0x44 in correct position
```

(vii) R0 = R1*10

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

(viii) R0 = R1*100

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

(ix) R0 = R1/256

```
MOV     R0, R1, LSR #8          ; R0 = R0 / 256
```

(x) R0 = R1 % 256 (mod operator - remainder on division)

```
AND     R0, R1, #0xFF           ; R0 = R1 % 256
```

Q3 Write an ARM assembly language program to calculate, in R0, the (sum % 256) of the 4 bytes in R1. For example, if R1 = 0x12345678, R0 = (0x12 + 0x34 + 0x56 + 0x78) % 256 = 0x14

```
AND     R0, R1, #0xFF           ; R0 = LS byte
ADD     R0, R0, R1, LSR #8      ; add next byte (ignore carry/over flow from LS byte)
ADD     R0, R0, R1, LSR #16     ; add next byte (ignore carry/over flow from LS byte)
ADD     R0, R0, R1, LSR #24     ; add next byte (ignore carry/over flow from LS byte)
AND     R0, R0, #0xFF           ; R0 = R0 % 256 (mod operator)
```

Q4 Write and ARM assembly language program to calculate, in R0, the number of one bits in R1. For example, if R1 = 0x12345678, R0 = 13.

```
        LDR     R1, =0x12345678     ; R1 = 0x12345678 (13 bits set)
        MOV     R0, #0              ; R0 = 0
L       CMP     R1, #0              ; if R1 == 0?
        BEQ     L0                  ; finished
        MOVS    R1, R1, LSR #1      ; shift LS bit into CARRY flag
        ADC     R0, R0, #0          ; add CARRY to R0
        B       L                   ; next bit

L0      B       L0
```