CS1021 Tutorial 4

Condition Code Flags

Q1 Translate following pseudo-code statement into a sequence of ARM assembly language instructions. Assume that x and y are signed integers in R1 and R2 respectively.

```
if ((x == 0x30) || (x == 0x31) || (x >= 0x40)) {
    y = 1;          // s0
} else {
    y = 0;          // s1
}
```

```
        CMP  R1, #0x30          ; x == 0x30
        BEQ  L0                 ; if TRUE execute s0
        CMP  R1, #0x31          ; x == 0x31
        BEQ  L0                 ; if TRUE execute s0
        CMP  R1, #0x40          ; x >= 0x40
        BLT  L1                 ; if FALSE execute s1 else s0
L0      MOV R2, #01             ; y = 1 (s0)
        B    L2                 ; skip s1
L1      MOV R2, #0              ; y = 0 (s1)
L2
```

```
if (((x >= 0x30) && (x <= 0x39)) || ((x >= 0x41) && (x <= 0x5a))) {
    y = 1;          // s0
} else {
    y = 0;          // s1
}
```

```
        CMP  R1, #0x30          ; x >= 0x30
        BLT  L0                 ; if FALSE check ((x >= 0x41) && (x <= 0x5a))
        CMP  R1, #0x39          ; x <= 0x39
        BLE  L1                 ; if TRUE execute s0
L0      CMP  R1, #0x41          ; x >= 0x41
        BLT  L2                 ; if FALSE execute s1
        CMP  R1, #0x5a          ; x <= 0x5a
        BGT  L2                 ; if FALSE execute s1 else s0
L1      MOV R2, #01             ; y = 1 (s0)
        B    L3                 ; skip s1
L2      MOV R2, #0              ; y = 0 (s1)
L3
```

Q2 For each ARM Assembly Language code segment below, determine the value stored in R0 and the state of the N (Negative), Z (Zero), C (Carry) and V (oVerflow) flags after the instructions have been executed

(i)
```
LDR     R0, =0x00000000
LDR     R1, =0x00000001
ADDS    R0, R0, R1          ; 0x00000001 N=0, Z=0, C=0, V=0 (0)
```

(ii)
```
LDR     R0, =0x00000001
LDR     R1, =0x00000000
SUBS    R0, R0, R1          ; 0x00000001 N=0, Z=0, C=1, V=0 (2)
```

(iii)
```
LDR     R0, =0x80000000
LDR     R1, =0x80000001
ADDS    R0, R0, R1          ; 0x00000001 N=0, Z=0, C=1, V=1 (3)
```

(iv)
```
LDR     R0, =0x00000000
LDR     R1, =0x00000000
SUBS    R0, R0, R1          ; 0x00000000 N=0, Z=1, C=1, V=0 (6)
```

(v)
```
LDR     R0, =0x00000000
LDR     R1, =0x00000000
ADDS    R0, R0, R1          ; 0x00000000 N=0, Z=1, C=0, V=0 (4)
```

(vi)
```
LDR     R0, =0x80000000
LDR     R1, =0x80000000
SUBS    R0, R0, R1          ; 0x00000000 N=0, Z=1, C=1, V=0 (6)
```

(vii)
```
LDR     R0, =0x80000000
LDR     R1, =0x80000000
ADDS    R0, R0, R1          ; 0x00000000 N=0, Z=1, C=1, V=1 (7)
```

(viii)
```
LDR     R0, =0x80000000
LDR     R1, =0x00000000
SUBS    R0, R0, R1          ; 0x80000000 N=1, Z=0, C=1, V=0 (10)
```

Q3 If x and y are signed 64-bit integers in R0:R1 and R2:R3 respectively and z is an integer in R4, translate the following pseudo-code statements into a sequence of ARM assembly language instructions

```
if (x == y) {     //
    z = 1;        // s0
} else {          //
    z =0;         // s1
}
```

```
;
; compare MS 32-bits of x and y
; if different, now know x != y so execute s1
; if equal, need to compare LS 32 bits to determine if x != y
;
        CMP  R0, R2             ; compare MS 32-bits
        BNE  L2                 ; if R0 != R2 execute s1
        CMP  R1, R3             ; MS 32-bits equal, compare LS 32-bits
        BNE  L2                 ; if R1 != R3 execute s1
L1      MOV  R4, #01            ; z = 1 (s0)
        B    L3                 ; skip s1
L2      MOV  R4, #0             ; z = 0 (s1)
L3
```

```
if (x < y) {      //
    z = 1;        // s0
} else {          //
    z =0;         // s1
}
```

```
;
; compare MS 32-bits of x and y
; if different, now know if x < y or x > y
; if equal, need to compare LS 32 bits to determine if x < y
;

        CMP  R0, R2             ; compare MS 32-bits
        BLT  L1                 ; if R0 < R1 execute s0 (signed branch)
        BGT  L2                 ; if R0 > R1 execute s1 (signed branch)
        CMP  R1, R3             ; MS 32-bits equal, compare LS 32-bits
        BHS  L2                 ; if R1 >= R3 execute s1 (unsigned branch)
L1      MOV  R4, #01            ; z = 1 (s0)
        B    L3                 ; skip s1
L2      MOV  R4, #0             ; z = 0 (s1)
L3
```