# Brief Solutions to Tutorial 2

## 2.1  ARM Addressing Modes, their Characteristics and Applications

(1) and (2) Solutions:

| No. | Mnemonics | Source addressing mode | Content in R0 |
|-----|-----------|------------------------|---------------|
| 1. | MOV R0,R2 | Register Direct | 0x00000004 |
| 2. | MOV R0,#0x0100 | Immediate | 0x00000100 |
| 3. | LDR R0,[R1] | Register Indirect | 0x00000102 |
| 4. | LDR R0,[R1,#4] | Base plus Offset (Pre-index) | 0xFFFEAABB |
| 5. | LDR R0,[R1,R2] | Base plus Index Register (Pre-index) | 0xFFFEAABB |
| 6. | LDR R0,[R1,#4]! | Offset with Autoindexing (Pre-index) | 0xFFFEAABB |
| 7. | LDR R0,[R1],#4 | Offset with Autoindexing (Post-index) | 0x00000102 |
| 8. | LDR R0,[R1,R2]! | Index Register with Autoindexing (Pre-index) | 0xFFFEAABB |
| 9. | LDR R0,[R1],R2 | Index Register with Autoindexing (Post-index) | 0x00000102 |

**Table 1_Answers – The different ARM addressing modes and its effects on R0 after execution**

(3) Solutions:

    (a) Register direct -    `MOV R0,R3`

    (b) Immediate -       `MOV R0,#0x100`
                                     `ADD R0,R0,#2`

    (c) Register indirect -  `MOV R0,[R1]`

(4) Solutions:

    (a) Register indirect.

    (b) Register direct.

    (c) Immediate.

(5) 
```
LDR R0,[R1]
STR R0,[R1,#4]
```

(6) Method 1 (less efficient)
```
        ADD R3,R1,#0x1C
        SUB R2,R3,#4
        :                      ; setup loop structure
Loop    LDR R0,[R2]
        STR R0,[R3]
        SUB R3,R3,#4
        SUB R2,R2,#4
        :                      ; loop back until 7 loops done
```

# Brief Solutions to Tutorial 2

Method 2 (more efficient)
```
        ADD R3,R1,#0x1C
        :                           ; setup loop structure
 Loop   LDR R0,[R3,#-4]
        STR R0,[R3],#-4
        :                           ; loop back until 7 loops done
```

(7) Swap registers
```
     MOV R4,R2
     MOV R2,R3
     MOV R3,R4
```

Swap memory locations
```
     LDR R2,[R1]
     LDR R3,[R1,#4]
     STR R2,[R1,#4]
     STR R3,[R1]
```

(8) Full Descending stack

Push R0 into a FD stack -         `STR R0,[SP,#-4]!`

Pop word from FD stack into R1 -  `LDR R1,[SP],#4`

## 2.2 Arithmetic and Logical Instructions

(1) Possible solutions:
```
     a. MOV R0,#0
     b. SUB R0,R0,R0
     c. EOR R0,R0,R0
     d. AND R0,R0,#0
```

(2) Possible solution:
```
             MOV R0,#0x11
             ADD R0,R0,#0x1100
             ADD R0,R0,#0x11000000
             ORR R2,R2,R0
```

(3) Possible solution:
```
             EOR R2,R2,R3
             EOR R3,R2,R3
             EOR R2,R2,R3
```

(4) Possible solutions:
```
        ADD R0,R0,R0,LSL #3
```

(i) `0x7FFFFFFF/9 = 0xE38E38E` **(238,609,294)**.

(ii) `RSB R0,R0,R0,LSL #3`

# Brief Solutions to Tutorial 2

---

**2.3  ARM Assembly Program Analysis – Comparison and Counting**

---

(1) Solutions:

> **R0** = 0x00000007
> **R1** = 0x00000109
> **R2** = 0x00000003
> **R3** = 0x00000080
> **R4** = 0x00000101
> **HotDays(0x100)** = 0x03
> **DaySum (0x101)** = 0x07

(2) The program computes the total number of days (**DaySum**) by counting number of byte-sized values retrieved before the value -128 (**0x80**) is encountered. It also counts the number of values that are larger than or equal to the decimal value 36 (i.e. **0x24**) and leave the result in memory variable **HotDays**.

(3) Possible solution:

```
Start     MOV    R4,#0x100      ; initialise R4 to point to start of memory data 0x100
          ADD    R1,R4,#1       ; initialise array pointer R1 to point to 0x101
          MOV    R0,#0          ; clear DaySum counter register R0
          MOV    R2,R0          ; clear HotDays counter register R2
Loop      LDRB   R3,[R1,#1]!    ; use index with pre-increment to access next element
          CMP    R3,#0x80       ; compare array element in R3 with terminator 0x80
          BEQ    Done           ; branch to Done if equal to terminator
          ADD    R0,R0,#1       ; increment DaySum counter register R0
          CMP    R3,#36         ; compare array element in R3 with threshold 0x24
          BHS    HotFound       ; branch to HotFound if >= 0x24
          B      Loop           ; branch back to Lopp
HotFound  ADD    R2,R2,#1       ; increment HotDays counter register R2
          B      Loop           ; branch back to Lopp
Done      STRB   R2,[R4]        ; store R2 into HotDays memory variable at 0x100
          STRB   R0,[R4,#1]!    ; store R0 into DaySum mem var at 0x101 and inc R4
          END
```

Note: Proposed optimized code executes in 93 clock cycles (savings of 34 clock cycles).

(4) **BHS HotFound** is a conditional branch for unsigned values. It needs to be changes to **BGE HotFound**. However, before the 32-bit comparison can be done, the byte read in from memory (which is zero-extended to 32 bits) must be sign-extended using the code segment below. This code segment must be inserted just before **CMP R3,#36**.

```
MOV   R5,#0
SUB   R7,R5,R3,LSR #7
ORR   R3,R3,R7,LSL #8
```