**COL216 Computer Architecture**
**Quiz 1 Solution** **10.02.2022**

**Q 1.** Three signed integers in 2's complement representation are stored in memory at addresses X, X + 4 and X + 8. Write an ARM assembly program to rearrange these in ascending order of their magnitudes.

**Solution outline**:

Let p, q and r denote the contents of memory locations X, X+4 and X+8, respectively.

We can find a = | p |, b = | q | and c = | r | by using any of the three methods given in slide 24 of Lec05. Then a series of unsigned comparisons shown below will do the rest of the work.

High level code

```
if (a > b) go to L3
   if (b > c) go to L1
      {order is: p, q, r}
   go to End
   L1: if (c > a) go to L2
      {order is: r, p, q}          store r, p, q at addresses X, X+4, X+8
   go to End
   L2:
      {order is: p, r, q}          store r, q at addresses X+4, X+8
go to End
L3:
   if (c > b) go to L4
      {order is: r, q, p}          store r, p at addresses X, X+8
   go to End
   L4: if (c > a) go to L5
      {order is: q, r, p}          store q, r, p at addresses X, X+4, X+8
   go to End
   L5:
      {order is: q, p, r}          store q, p at addresses X, X+4
End:
```

ARM assembly code

```
        ldr r0, =X
        ldr r1, [r0]            get the three numbers into r1, r2 and r3
        ldr r2, [r0, #4]
        ldr r3, [r0, #8]

        cmp r1, #0              get the absolute values into r4, r5 and r6
        movge r4, r1
        rsblt r4, r1, #0

        cmp r2, #0
        movge r5, r2
        rsblt r5, r2, #0

        cmp r3, #0
        movge r6, r3
        rsblt r6, r3, #0

        cmp r4, r5              start comparing absolute values
        bgt L3
        cmp r5, r6
        ble End
L1:     cmp r6, r4
        bgt L2
        str r3, [r0]            store r, p, q at addresses X, X+4, X+8
        str r1, [r0, #4]
        str r2, [r0, #8]
        b End
L2:     str r3, [r0, #4]        store r, q at addresses X+4, X+8
        str r2, [r0, #8]
        b End
L3:     cmp r6, r5
        bgt L4
        str r3, [r0]            store r, p at addresses X, X+8
        str r1, [r0, #8]
        b End
L4:     cmp r6, r4
        bgt L5
        str r2, [r0]            store q, r, p at addresses X, X+4, X+8
        str r3, [r0, #4]
        str r1, [r0, #8]
        b End
L5:     str r2, [r0]            store q, p at addresses X, X+4
        str r1, [r0, #4]
End:
```

Marking scheme:

- 1 mark for loading 3 numbers
- 1x3 marks for finding absolute values of three numbers
- 6 marks for comparing absolute values and arranging numbers
  There are 6 outcomes of this. 1 mark for each outcome.

**Q 2.** What does the following ARM assembly function do? Give justification for your answer.

```
    .global func
    .text
func:
    ldr r2, =consts

    ldr r1, [r2], #4
    and r3, r1, r0
    and r0, r1, r0, LSR #1
    add r0, r0, r3

    ldr r1, [r2], #4
    and r3, r1, r0
    and r0, r1, r0, LSR #2
    add r0, r0, r3

    ldr r1, [r2], #4
    and r3, r1, r0
    and r0, r1, r0, LSR #4
    add r0, r0, r3

    ldr r1, [r2], #4
    and r3, r1, r0
    and r0, r1, r0, LSR #8
    add r0, r0, r3

    ldr r1, [r2], #4
    and r3, r1, r0
    and r0, r1, r0, LSR #16
    add r0, r0, r3

    mov pc,lr

    .data
consts: .word 0x55555555, 0x33333333, 0x0f0f0f0f, 0x00ff00ff, 0x0000ffff
    .end
```

**Solution:**

What does the function do?
This function adds all the 32 bits in the argument supplied in r0. During this process, all 1's get added and 0's have no contribution. The result is a count of 1's in the given argument (in register r0). The result is returned in r0.

Justification
Let the given number be ($b_{31}$ $b_{30}$ . . . . $b_1$ $b_0$).

The function contains 5 similar groups of code, each group containing 4 instructions. The first instruction loads one of the five patterns at address consts onwards. The next two instructions split the contents of r0 into two integer arrays and the 4th instrucion adds these two vectors as explained below.

The first group of instructions adds $b_{i+1}$ to $b_i$, with result in ($b_{i+1}$, $b_i$),
for i = 30, 28, . . . , 2, 0.

The second group of instructions adds ($b_{i+3}$, $b_{i+2}$) to ($b_{i+1}$, $b_i$), with result in ($b_{i+3}$, . . . , $b_i$),
for i = 28, 24, . . . , 4, 0.

The third group of instructions adds ($b_{i+7}$, . . . , $b_{i+4}$) to ($b_{i+3}$, . . . , $b_i$), with result in ($b_{i+7}$, . . . , $b_i$),
for i = 24, 16, 8, 0.

The fourth group of instructions adds ($b_{i+15}$, . . . , $b_{i+8}$) to ($b_{i+7}$, . . . , $b_i$), with result in ($b_{i+15}$, . . . , $b_i$),
for i = 16, 0.

The fifth group of instructions adds ($b_{i+31}$, . . . , $b_{i+16}$) to ($b_{i+15}$, . . . , $b_i$), with result in ($b_{i+31}$, . . . , $b_i$),
for i = 0.

Marking scheme:

4 marks for first part (answering what the given function does).
6 marks for second part (correct and complete justification/explanation).
3 marks for a fair attempt at justification/explanation for whatever answer is given in first part

**Q 3.** Write an ARM assembly function to find the number of times each character appears in a given text, recording these numbers in an array. The text is in the form of a null terminated string of characters (1 byte per character). Each character has a 7-bit code. The 8th bit (MSB) in the byte is parity bit and may be ignored. Address of the text and the result array are passed on to the function in register r0 and r1.

**Solution:**

```
void Freq (const char T [ ], const int H [ ]) {
   int c, i;
   for (c = 1; c < 128; c++) H [c] = 0;                  // Initialize histogram
   i = 0;
   while ((c = (T [i++] & 0x7f)) != 0) H [c]++;          // Update histogram entries
};
```

```
        .global Freq
        .text
Freq:
        mov r2, #1                  // index for frequency array
        mov r3, #0                  // constant 0 to initialize the frequency array
Loop0:
        str r3, [r1, r2, LSL #2]    // initialization
        add r2, r2, #1              // next index
        cmp r2, #127
        blt Loop0
Loop1:
        ldrb r2, [r0], #1           // load byte from string
        ands r2, r2, #0x7f          // ignore parity bit
        beq Done                    // check for null
        ldr r3, [r1, r2, LSL #2]
        add r3, r3, #1              // update frequency
        str r3, [r1, r2, LSL #2]
        b Loop1
Done:   mov pc, lr                  // return
        .end
```

Marking scheme:

10 marks for correct code.

Partial marks - initialization of frequency array (2), correctly loading characters of string sequentially (2), ignoring parity bit (1), checking for string termination (2), correctly indexing frequency array and making updates in it (3).