Q1

```
    mov r3, #0 ; a=0
    str r3, a

    ldr r3, y ; if (y != 0) {
    cmp r3, #0
    beq end

    bge skip ; if (y < 0) {
    rsb r3, r3, #0 ; y = -y
    str r3, y

    ldr r2, x
    rsb r2, r2, #0 ; x = -x
    str r2, x

    skip ; }
    ldr r3, x ; if (x >= 0)
    cmp r3, #0
    blt Loop2

    Loop1
    ldr r3, x ; while (x >= y) {
    ldr r2, y
    cmp r3, r2
    blt end

    ldr r3, y ; x = x - y
    ldr r2, x
    sub r2, r2, r3
    str r2, x

    ldr r3, a ; a = a + 1
    add r2, r3, #1
    str r2, a

    b Loop1 ; }

    Loop2 ; else
    ldr r2, x ; while (-x >= y) {
    rsb r3, r2, #0
    ldr r2, y
    cmp r3, r2
    blt end ; ble for "while (-x > y)"

    ldr r3, y ; x = x + y
    ldr r2, x
    add r2, r2, r3
    str r2, x

    ldr r3, a ; a = a - 1
    sub r2, r3, #1
    str r2, a

    b Loop2 ; }

    end ; }
```

Some optimizations that you could do:

- Use conditional instructions to save some branches
- Only LDR the variables at the beginning and STR them at the end
Q2: Write ARM code which, given the address of a string in R1, uses a loop to calculate in R0 the arithmetic total of the bytes of the string using:

a) indirect addressing

```
MOV R0, #0 ; R0 = total
again LDRB R2, [R1] ; R2 = next byte
CMP R2, #0
ADDNE R1, R1, #1
ADDNE R0, R0, R2
BNE again
```

5 instructions obeyed each time around loop + 1 initialisation

b) offset (indexed) addressing

```
MOV R3, #0 ; R3 = index
MOV R0, #0 ; R0 = total
again LDRB R2, [R1,R3] ; R2 = next byte
CMP R2, #0
ADDNE R3, R3, #1
ADDNE R0, R0, R2
BNE again
```

5 instructions obeyed each time around loop + 2 initialisation

(b) = (a) + initialising and using R3

c) pre-indexed (pre-increment) addressing

```
MOV R0, #0 ; R0 = total
SUB R1, R1, #1
again LDRB R2, [R1, #1]! ; R2 = next byte
CMP R2, #0
ADDNE R0, R0, R2
BNE again
```

4 instructions obeyed each time around loop + 2 initialisation

(c) = (a) - ADD #1 instruction in loop body

d) post-indexed (post-increment) addressing

```
MOV R0, #0 ; R0 = total
again LDRB R2, [R1],#1 ; R2 = next byte
CMP R2, #0
ADDNE R0, R0, R2
BNE again
```

4 instructions obeyed each time around loop + 1 initialisation

(d) = (c) - initial SUB #1 on address in R1

In each case, you should use any extra facilities of the ARM instruction set (e.g. conditional instructions) to make your code faster.

Actually, using e.g. ADDNE instead of ADD in any of the loop bodies is unnecessary. If students use it, please ask them why, and get them to reason about why it makes no difference to the answer in R0.

e) Which of your 4 answers (a) to (d) is the fastest

(d) then (c) then (a) then (b) (but see note for c)

and why?

See notes after each answer above.