Tuan Nguen

Question 1:
Instructions that set register `r0` to zero: `MOV r0, #0`(sets the condition codes) `SUB r0, r0, r0`(sets the condition codes)
Instructions that copy register `r1` to `r0`: `LSL r0, r1, #0`(sets the condition codes on the result), `LSR r0, r1, #0`(sets the condition codes on the result), `ASR r0, r1, #0`(sets the condition codes on the result), `STR r1, [r0, #0]`, `LDR r0 [r1, #0]`

Question 2:
`movs r2, #0` – sets r2 to 0
`cmp r0, #0` – compares r0 to 0
`beq cc <done>` - sets the pc to cc, which is to exit the loop
`subs r0, #1` – subtracts 1 from r0
`adds r2, r2, r1` – adds r1 to r2 and stores the result in r2
`b c2 <loop>` - sets the pc to c2, which is to continue the loop
`movs r0, r2` – movs r2 to r0
`bx lr` – exits the program

Question 3:
I will be keeping x, y and z in registers `r0`, `r1`, `r2`.

```
foo:
movs r2, #0

loop1:
movs cx r1
subs r0, #1
adds r2, r2, r1
loop loop1
```

Question 5:
The `blt` is the command for signed branches, which branches if the result is less than zero. So if we compared 80 and 02(hexadecimal), then 02 would be considered to be larger, because for signed numbers 80 is -128 in decimal and 02 is 2 in decimal.
The `blo` is the command for the unsigned branches, which branches if the result is higher or equal. If we compared 80 and 02(hexadecimal), then 80 would be considered to be larger, because for unsigned numbers 80 is 128 in decimal and 02 is 2 in decimal.

Question 6:
```
/** Division of two integers
def slowDiv(a : Int, b : Int) : Int = {
  var m = a
  //Invariant : m >= q * b ^ 0 <= q ^ 0 <= m <= a
  //Variant : m
  while(m != 0){
    q += 1
    m -= b
  }
  //m < 0, so q = a / b
  q
}

foo:
movs r2, r0
movs r3, #0
b test

again:
subs r2, r1
adds r3, #1

test:
cmp r2, #0
bne again
movs r0, r3
bx lr
```

Question 7:

```
/** Function for fast division */
def fastDiv(a : Int, b : Int) : Int = {
  var r = 0; var q = 0; var i = 31
  //q – quotient, r – remainder, i – counting the bits
  //Invariant : r(0) = a(i) ^ q(i) = 1 ^ 0 <= i <= 31
  //Variant : i
  while(i >= 0){
    r = r << 1
    r = ((a & (1 << i)) >> i) | r
    if(r >= b){
      r -= b
      q = q | (1 << i)
    }
    i -= 1
  }
  //i = 0, so all the 32 bits of the quotient are calculated
```

```
    q
}
```