DIGITAL SYSTEMS

SAMPLE EXAM QUESTIONS

TT 2019

GABRIEL MOISE

1. [2011/2]

(a)

(i)

```
        cmp     r0, #0
        beq     skip
```

(ii)

```
        cmp     r0, #0
        bne     skip
```

(iii)

```
        movs    r3, r0      @ we are told that r3 can be overwritten
        lsrs    r3, #1
        bcs     skip
```

(iv)

```
        movs    r3, r0
        lsrs    r3, #1
        bcc     skip
```

(b)

```
acount(a,n)      // I suppose that it is numbered from 0
  total = 0
  for (i = 0 until n)
     total = total + count(a[i])
  return total
```

(c)

```
        @ the address of the array is in r2, and the length is in r3, r1 builds up the result
        …
        movs    r4, #0          @ the index where we currently are
        cmp     r3, #0
        beq     done            @nothing to do if the array is empty
acount :
        lsls    r5, r4, #2      @getting the address of the current element of the array
        ldr     r0, [r2, r5]    @ loading the value from the address in r0, to use count on it
        bl      count           @ the result comes back in r0
        adds    r1, r0, r1      @ we add it to r1
        cmp     r4, r3          @ checking if we finished or not
        blt     acount          @ if not, repeating
done :
        …
```

(d)

0x 358	= 0000 0011 0101 1000		=> 5 bits set
0x 357	= 0000 0011 0101 0111		=> 7 bits set
0x 358 & 0x 357	= 0000 0011 0101 0000	= 0x 350	=> 4 bits set

By subtracting 1 from x, we basically add 1111 1111 1111 1111 to it, as the negation of 1 is 1111 1111 1111 1110 and we also need to add a 1. Therefore, (x-1) will basically be opposite to x starting from the most significant bit, until it gets to a 1, where (x-1) will have a 0, and then the rest stays the same. Therefore we will have all the bits equal until that position (starting from bit 15 until bit k let's say), then at bit k x will have a 1 and (x-1) will have a 0 and from there, we have 0 for x and 1 for (x-1). Therefore, the "and" will produce the same bits from 15 until k as in x, then from k to 0 there will be 0, because the bits are opposite in x and (x-1). So, the number of set bits resulting will be equal to the number of set bits from x, – 1.

count(x) = count(x & (x-1)) + 1 , for x > 0

(e)

```
 count (x)
   result = 0
   while (x > 0)
      result = result + 1
```

```
    x = x &(x-1)
  return result


count :
        movs    r1, #0          @ the result is built here
loop :
        adds    r1, r1, #1
        subs    r2, r0, #1
        ands    r0, r0, r2
        cmp     r0, #0
        bgt     loop
done :
        movs    r0, r1
```

(f)

How should I make the code “simpler”? What does it mean to be simpler? Less instructions? Clearer instructions? Improve running time? Because I feel like I cannot reduce either of those anymore…

2. [part of 2008/3]

(a)

```
        mov     r0, #0          @ i = 0
        mov     r1, #0          @ m= 0
loop :
        ldr     r2, =a          @ the base address of the array
        lsls    r3, r0, #2      @ 4*i in r3
        ldr     r3, [r2, r3]    @ the value of a[i] in r3
        adds    r0, r0, #1      @ we increment i ; in r3 we have a[i-1] now
        cmp     r3, r1          @ compare a[i-1] with m
        ble     test            @ if a[i-1] <= m we skip updating m
        movs    r1, r3          @ otherwise we put a[i-1] in m
```

```
test :
        cmp     r0, #N          @ compare i with N
        blt     loop            @ if i < N loop again
```

(b)

```
        mov     r0, #0          @ i = 0
        mov     r1, #0          @ m= 0
        ldr     r2, =a          @ the base address of the array (we don’t load it each time)
loop :
        ldr     r3, [r2]        @ the value of a[i] in r3
        adds    r0, r0, #1      @ we increment i ; in r3 we have a[i-1] now
        adds    r2, r2, #4      @ we go to the address of the next element in a
        cmp     r3, r1          @ compare a[i-1] with m
        ble     test            @ if a[i-1] <= m we skip updating m
        movs    r1, r3          @ otherwise we put a[i-1] in m
test :
        cmp     r0, #N          @ compare i with N
        blt     loop            @ if i < N loop again
```

This version is faster because we have one less load instruction (and we have an adds instruction instead of a lsls, but that doesn’t have an impact on the speed). The idea is that we do not need to calculate the address of the current element each time, instead we can keep the current address and add 4 after each iteration.