Tuan Nguen

Question 1:

a) The function that maps unsigned 32-bit number to the corresponding integer is:

$$map1(a) = \sum_{i=0}^{31} a_i . 2^i$$

where a_i is the i-th least significant bit.

The function, which maps a 32-bit signed number in twos complement to its integer is:

$$map2(a) = \sum_{i=0}^{30} a_i . 2^i - a_{31} . 2^{31}$$

where again a_i is the i-th least significant bit.

b) The add function for two 32-bit unsigned numbers is going to be:

$$map1(a \oplus b) = map1(a) + map1(b)\ (mod\ 2^{32})$$

We can only do the mod of this binary addition, because of the limitations of the output of map1. Each 32-bit number is uniquely specified modulus 2^32, so this addition function is uniquely specified. This adder can be used for signed numbers as well, because map2(a) = map1(a) modulus 2^32. So:

$$map2(a \oplus b) = map2(a) + map2(b)\ (mod\ 2^{32})$$

c) To negate a 32-bit signed number, we can flip every bit and then add 1. So if we wanted a-b, then we can add a and (-b). Therefore, to subtract the two numbers, we can add a and c, then add 1, where c is the 32-bit number, where the bits correspond to those in b, but are flipped.

d) Suppose we have to compare 3(0000000000000011) and -72(1111111110111000). If we use either blt or blo, the compiler would subtract them and look at the flags to see which one is bigger. If we use blt, in order not to overflow we will need a 33rd bit and then subtract them. Then the result would be positive, so 3 > -72. If we use blo, the compiler would think we are comparing 3 and 65464. So the result would be 3 < -72.

e) When we want to compare two signed number, we subtract them and look at the result. If the result of a-b is negative, then it could be either that b is larger than a, or that a is positive, b is negative and the subtraction overflowed. Suppose a = 100 and b = -100(in 8-bit twos complement), the sign bit is going to be negative, which is not the desired result. To avoid this, we can look at the V flag to see if the subtraction overflowed. To implement blt, we have to check if the N flag is different from the V flag.

f) Because blo treats both numbers as unsigned, we can just look at the C flag, and see whether it is 0 or 1.

Question 2:

a) The calling convention is that the argument for a function is in r0, and the result is put in r0. If we want to use r4-r7, then we need to save the values in those registers, and then restore them back. We need to save lr as well.

b) The push instruction saves the values of r4, r5 and lr. When a pop instruction is executed, those registers will have the old values. This way we can use r4, r5 and lr without corrupting caller-saved registers.

c) instruction 1: blt foo_ret1
instruction 2: subs r0, r0, #3
instruction 3: adds r0, r0, r5

d) blank 1: x <= 2
blank 2: return x – foo(x+1)