CSE3666

Homework 1 Solutions

1. 10-bit numbers.

Solutions:

a. The left most hex digit specifies 2 bits. The ten bits are 0110101011.

   Unsigned: We can convert either binary or hex to decimal. 1 * 256 + 10 * 16 + 11 = 427.

   2's complement number: Since this is a positive number (the highest bit is 0), the result is the same.

b. The left most hex digit specifies 2 bits. The ten bits are 1011001101.

   Unsigned: We can convert either binary or hex to decimal. 2 * 256 + 12 * 16 + 13 = 717

   2's complement number: Since this is a negative number (the highest bit is 1), the result is – 512 + 12 * 16 + 13 = -307. To check, 717 + 307 = 1024 = 2^{10}. Or we can compute the value by 717 – 2^{10} = 717 – 1024 = – 307. Note that the exponent is 10 because it is a 10-bit 2's complement number.

2. 8-bit two's complement numbers.

Solutions:

a. We can find out the binary representation with division by 2. Although it is not shown here, the steps should be included in your answer.

   We can also find out hexadecimal representation first (using division by 16).

   95 = 0x5F = 0b 0101_1111.

b. Since this is a negative number, we can find binary representation of 100. Then negate it.

   101 = 0x65 = 0b 01100101.

   Flip all bits and add 1: 0b 1001 1010 + 1 = 0b 10011011.

   Two hexadecimal digits are 0x9B.

   Alternative methods. Learn more than one method and use the fastest one for specific numbers. For example, since 155= 256 – 101, the 8-bit representation of unsigned 155 and -101 are the same. We then find the 8 bits that representing 155.

3. Understanding instructions.

The answers are already provided.

Explanation:

We could convert everything to bits (not to a decimal number), find the result, and then convert bits to hex digits, but there are faster ways sometimes.

ADD: Do addition in hexadecimal.

AND: We convert each pair of hex digits to bits, AND, and then convert them back to hex digits. For special cases, we can do it quickly with hexadecimal digits. If a hex digit is 0, we know the resulting digit is 0. If a hex digit is F, we know the result is the other digit.

OR: Similar to AND. There are special cases. Note if a hex digit is F, the result will be F.

XOR: Similar to AND. There are also special cases, for example, when a hexadecimal digit is 0.

ADDI: similar to ADD.

ANDI: Convert -16 to hex first. -16 = (-1 << 4) = 0xFFFF FFF0. We then know the left 7 hex digits do not change. The least significant (right-most) hex digit is cleared.

SLLI: This is a special case. Shift hex digits to left by 3 positions.

SRAI: Pay attention to the sign bit because this is SRAI (not SRLI). This question is a special case. Shift hex digits to right by 2 positions. The highest bit is 1 and the instruction is SRAI. The sign is duplicated so the highest two hex digits are 0xFF.

4. Compute 24*x*.

Solutions:

To get 24, we can do (2 + 1) * 8, (8 + 16), or (32 – 8). The following code does (2 + 1) * 8.

```
        slli  t0, s1, 1       # 2x
        add   t0, t0, s1      # 3x
        slli  s2, t0, 3       # 3x*8 = 24x
```

Note that 32*x* is larger than 24*x*, which may have overflow.

5. Operations on bits.

```
     andi  t0, s1, 0x200         # separate bit 9
     beq   t0, x0, else          # if bit 9 is 0, goto else
     andi  s1, s1, 0x1FF         # set bits 32 to bit 9 to 0
     beq   x0, x0, endif
else:
     ori   s1, s1, 0xFFFFFE00    # note the immediate is sign extended
endif:
```

6. Hamming weight.

Solutions:

a. The loop is controlled by the mask. So the number of iterations of the loop does not depend on the bits in s0. The extracted bit decides if s1 is incremented or not. Therefore, the number of executed instructions depends on the number of 1's in s0, but does not depend on the location of 1's.

   The loop is executed for 32 times. In the loop, addi instruction is executed only if the extracted bit is 1. So the number of executed instruction in the loop is 32 * 4 + 16 = 144. Plus two instructions before the loop, the total number of executed instruction is 146.

b. The code is listed below.

```
      addi   s1, x0, 0          # s1 = 0
      add    t0, x0, s0         # make a copy so s0 is not changed
loop:
      bge    t0, x0, skip       # if the bit is 0, do not increment s1
      addi   s1, s1, 1          # increment the counter
skip:
      slli   t0, t0, 1          # shift bits to left by 1
      bne    t0, x0, loop       # if there is any 1 in t0, continue
```

   The loop terminates when t0 is 0. The number of iterations depends on the location of 1's. If t0 is 0 at the beginning, the loop is executed only once. If t0 is 1, the loop is executed 32 times.

   For s0 = 0xFF00FF00, the loop is executed 24 times. In each iteration, the ADDI instruction (in the loop) is executed only when the highest bit in t0 is 1. So the number of executed instruction in the loop is 24 * 3 + 16 = 88. Plus two instructions before the loop, the total number of executed instruction is 90.

   Can you figure out other ways to compute the Hamming weight?

7. Nested loop.

Solutions:

```
        lui     s5, 0x55AAC         # s5 = the constant
        addi    s5, s5, 0xFFFFFB33

        addi    s2, x0, 1           # i = 1
        beq     x0, x0, condi

loopi:
        #------------ Begin of loop body i
        addi    s3, x0, 0           # j = 0
        beq     x0, x0, condj

loopj:
        #------------ Begin of loop body j
        xor     t0, s3, s5          # ^
        add     s4, s4, t0          # +
        #------------ End of loop body j
        addi    s3, s3, 1           # j ++
condj:
        blt     s3, s2, loopj       # j < i?

        #------------ End of loop body i
        addi    s2, s2, 1           # i ++
condi:
        blt     s2, s1, loopi       # i < a?
```