

Tutorial 3 [CO1]
CSE 112 Computer Organization

Question 1:

- a. Consider the binary number “1001100” and try to find out the decimal representation of the number for the following cases. Also, find out the max and min number that can be represented using the same number of bits: -
 - i. Assume the number follows the unsigned representation
 - ii. Assume the number follows the signed magnitude representation
 - iii. Assume the number follows the 2’s complement representation
- b. Do all the subparts in the previous part (“a”) for the binary number “0001100”. c. Can you figure out the equivalent decimal number if we don't specify the type of binary representation?

Question 2:

Write the 2’s complement notation for the following using the specified number of bits:

- a. 10 using 5 bits
- b. 10 using 6 bits
- c. -10 using 5 bits
- d. -10 using 6 bits

Do you see a pattern? Can you find out the answer to part b by looking at part a’s answer? Can you find out the answer to part d by looking at part c’s answer?

Question 3:

Convert the following number into unsigned representation, signed magnitude and 2’s complement. Also, report the minimum number of bits required to represent the number, if possible.

- 1) 20
- 2) -20
- 3) 32
- 4) -32

Question 4:

- a. If you are working in a k-ary system (for example, binary has 2 ($k=2$) distinct symbols, 0 and 1; decimal has 10 ($k=10$) distinct symbols 0,1, 2, 3,... 9), and you have a sequence of n characters (a character may take one of k different symbols), then how many numbers can you represent with this sequence?
- b. Given a set of m numbers, what is the minimum length of the sequence of characters required to represent each member of this set uniquely in a k-ary system? c. How does this scale with k? An increase in k will decrease or increase the minimum length of the

sequence?

- d. If you answered “decrease”, then clearly, a larger k can represent more numbers with a shorter sequence. Then why do we use $k = 2$ in most computer systems? Why not a higher k ?

Question 5:

Can you point out some differences between 2's complement and signed magnitude notations?

Question 6:

Suppose we have three registers, say, X , Y and Z . All these registers are 8-bit wide. Suppose X has 01111111, and Y has 00000001. Suppose we perform $Z = X + Y$ using an 8-bit 2's complement ALU.

Would Z have the correct sum of X and Y after the computation? If yes, what will be this sum? If not, can you come up with a way to detect whether the sum of such operations is valid or not?

Question 7:

- For unsigned notation, how many bits are required to represent the sum of d n -bit numbers for proper representation?
- For unsigned notation, how many bits are required to represent the product of an n -bit and an m -bit number for proper representation?

Question 8:

Recall question 6, where we discussed ways to detect overflows. Can you think of some ways of handling/correcting overflows as well? Do you think this would be a useful feature? If yes, why?

Question 9:

Write a program to perform the below-mentioned operation using the R-type instruction set from the RISC-V ISA.

$r3 = r1 + r2 - r4 - r7$

$r2 = r3 - r1 + r3$

$r2 = r2 - r1$

Semantic	Explanation
add rd, rs1, rs2	Add Add the content of the rs1 and rs2 and store the result in rd. $rd = rs1 + rs2$
sub rd, rs1, rs2	Subtract Two's complement subtraction. $rd = rs1 - rs2$.
sll rd, rs1, rs2	Shift left logical $rd = rs1 \ll \text{unsigned}(rs2[4:0])$ Left shift rs1 by the value in lower 5 bits of rs2.
Note: Appending zeros at required places is called logical shift.	
slt rd, rs1, rs2	Set if less than $rd=1$. only if $\text{signed}(rs1) < \text{signed}(rs2)$
Note: Here we treat the register content of rs1, rs2 as signed integers.	
sltu rd, rs1, rs2	Set if less than unsigned $rd=1$. Only if $\text{unsigned}(rs1) < \text{unsigned}(rs2)$
Note: Here we treat the register content of rs1, rs2 as unsigned integers.	
xor rd, rs1, rs2	Bitwise Xor $rd = rs1 \oplus rs2$
srl rd, rs1, rs2	Shift right logical $rd = rs1 \ll \text{unsigned}(rs2[4:0])$ Left shift rs1 by the value in lower 5 bits of rs2.
Note: Here no sign is preserved.	
or rd, rs1, rs2	Bitwise Or $rd = rs1 rs2$
and rd, rs1, rs2	Bitwise And $rd = rs1 \& rs2$