Computer Architecture

Programming Assignment #2

2019-11563

자유전공학부 김지원

1. One bit parity. Count the numbers using divide and conquer. XOR the first and and the latter half, and divide them into two parts again and repeat.
2. Check Subtraction. There are two cases of overflow in terms of the numbers signs; (+) – (-) = (-) and (-) – (+) = (+). So first extract signs of x, y, and x-y. And check if x sign and y sign is the same, and if x sign and (x-y) sign is the same. If both are not the same, it’s overflow.
3. Twoscom2SignedVal. If the given number is positive: do nothing, if it’s negative: invert all bits and add 1, after set sign bit to 1. Do this by first getting the sign bit and later the magnitude of the number.
4. Nibble Reverse. Create two masks. First mask = 0x0F0F0F0F, second mask = 0xF0F0F0F0. And swap the nibbles by moving (x & mask) by the necessary amount of bits.
5. Bit Filter. First move the input right and then back by the number of lowbit. This will remove all the lower part to 0. Second, do ‘1<<highbit’ which will be the starting place of the highbit mask. Then, do ‘OR’ operation on the original highbit mask and the right shifted highbit mask by 1, 2, 4, 8, 16. This will make a highbit mask consisting of (highbit+1) number of 1’s. Then, do AND operation with the input.
6. Add and divide by 4. First, calculate sum and store 0 or 1 in ‘int overflow’ according to x, y, and sum’s sign. Do the division of 4 by shifting right by 2. But if it’s a negative number, add an offset 3 before shifting to round towards zero.
7. Num Zeros First. First check the top 16 bits. If top 16 bits are 0, mask = 16, else 0. Then, shift left by mask bits if there were zeros. And then start count with mask (number of bits we’ve handled). Repeat the process to check the top 8, 4, 2, 1 bits. If x is 0, add 1 to 31 zeros and return.
8. Abs Float. First clear the sign bit to get the absolute value. Then, check if the value is NaN by comparing it with 0x7f800000. If so, return the original parameter.
9. Cast float 2 Int. First, extract sign bit (bit 31), extract exponent (bits 30 to 23), and extracts the fraction (bits 22 to 0). Check if it’s a zero value, or overflow. In this case return 0. Then check if underflow, if so, return the minimum possible signed integer. If uf is a normalized floating-point number, so proceed with the conversion. calculate effective exponent by subtracting 127 from the extracted exponent and add implied 1 bit.Then, simulate reversing the normalization shift by shifting the fraction left if exp > 23 or right if exp < 23. Lastly, if the sign bit is 1 (indicating a negative number), it takes the two's complement of the fraction to obtain the negative integer. If the sign bit is 0, it returns the fraction as an unsigned integer.
10. Compare Float. First, extract Sign, Exponent, and Fraction. Bits are extracted from the unsigned integers representing the floating-point numbers to get the sign, exponent, and fraction parts.Second, do NaN check. If either number is NaN (exponent bits are all 1s and fraction bits are non-zero), the function returns 0. Lastly, do zero equality check: both +0 and -0 are considered equal, and the function returns 0 if both numbers represent zero. Finally, do comparison with sign adjustment: if the numbers are negative, two's complement is taken to simplify the comparison. The adjusted numbers are then compared, and the result is returned.