Essentials of Computer Systems - Exercises #5

Laith Agbaria

Introduction

Unsigned binary numbers

You have previously been taught the binary number system (base-2), including how to convert positive integer values from other number systems into binary. The representation you obtain from those conversions is referred to as the unsigned binary representation, with this representation you can represent 2^n values using n amount of bits.
The range of values that can be represented with an unsigned n-bit binary number is $[0, 2^n - 1]$ or $[0, 2^n)$. The reasoning behind this is simple: Since we always start with zero, it must be one of 2^n values represented. And since the remaining values increment positively, the end point will be $2^n - 1$ as there are 2^n values total.
In the unsigned binary representation, adding a 0 as the new most significant bit does not change the value of the number.

Addition in binary and overflow

Addition in binary is simple:
0+0 = 0+0+0 = 00, (Carry over 0)
0+1 = 1+0+0 = 01, (Carry over 0)
1+1 = 0+1+1 = 10. (Carry over 1)
We start from the least significant bit to the most significant bit similar to decimal addition, we apply the basic addition and add any possible carry over, then we move on to the next bit.
Here are two examples in both binary and decimal:

(a)	0	1	0	1	(b)	0	1	1	1	(a)	0	5	(b)	0	7
+	0	0	0	1	+	0	0	1	1	+	0	1	+	0	3
	0	1	1	0		1	0	1	0		0	6		1	0

In these two examples, the binary values are represented as 4-bit unsigned numbers. Needing to use more bits than the specified amount is called overflow, and it is a common source of program and calculation errors, as we would need to sacrifice some data to store the results. (Look up pigeonhole math)
In terms of number representation systems, overflow is when the result of an operation contained in the output is not accurate due to lack of symbols to represent the correct answer of the operation.
Given n-number of bits in the unsigned binary number representation, you have n-number usable bits. Usable bits are bits you can modify as you wish to represent data while adhering to the rules of the representation, by sacrificing some bits in certain positions the representation method can encode certain information. The encoded information is agreed upon prior to the use of the representation method and therefore can allow more complex ideas to be stored in simpler forms, such as data type, data location, data size, etc. . . .

Signed binary numbers

Unsigned binary number representation is useful for storing positive integers as it maximize the range of absolute values that can be stored in a given number of bits and is also easy to apply addition and multiplication with, however since it is unable to store negative numbers there is greater difficulty with subtraction and more complex functionalities.
Therefore we need a new method of representing values in binary, one that allows us to differentiate between the positive and negative while maintaining our ability to do arithmetic operations. A simple solution would be to reserve and change the most significant bit into a sign bit, with having 0 as the most significant bit meaning a positive value and 1 meaning a negative value, this is called signed binary number representation.
This representation will allow us to represent the values $[-2^{n-1} - 1, 2^{n-1} - 1]$ using n bits, as the most significant bit is used for a sign leaving us with $n - 1$ bits to represent value.
Thinking further on what this representation entails, let us consider the signed number 000 which has a 0 as the most significant bit giving us the value +0. But we can also have 100 which has 1 as the most significant bit, giving us the value -0. In other words $+0 \neq -0$, which shouldn't be true given that they are in fact equal.
Furthermore, while signed numbers do allow us to distinguish between positive and negative values, subtraction is still difficult as except for the most significant bit it is still the same as unsigned representation. It also adds complexity to the arithmetic between two numbers, as you will need to apply the operations such as addition differently based on the sign-bit. Clearly this value representation is not as useful as it appeared and a better representation method is necessary.

Two's complement and subtraction

The better representation is called two's complement, it addresses most of the failings of signed numbers such as double zeros and more complex addition.
In two's complement representation, we again reserve and change the most significant bit into a sign-bit that represents whether the value is positive or negative, however what differentiates it from signed binary number representation is that instead of each value being represented by one complement (combination of bits), each value is represented by two complements (one for positive and one for negative).
That is the technical idea, now to the more intuitive explanation. The purpose of number representations is to simplify the execution of arithmetic operations, starting with addition we want our number representation to be as simple as the unsigned binary number representation. Therefore when the number is positive we set the most significant bit to 0, and the rest of the usable bits should not change from unsigned binary representation.

Next we want to consider subtraction, we want a value minus itself to equal 0. In other words given 3-bits for example to represent the value 3 and -3 we want our output to be 000, since we decided that positive values are written the same as unsigned binary we have 3 = 011 and we want our output to be 000, the most significant bit of -3 is 1 by definition of the sign-bit. So -3 = 1XY and adding it to 011 will give us 000, well we know that 1+1=0 with 1 carry-over bit and since Y+1=0 (ignoring carry-over), we get Y = 1 and -3 = 1X1. Using the same logic again on X1+11=00 (ignoring carry-over again) we get X = 0 and -3 = 101, and to check the original goal of 3-3=0 we have 011+101=1000, since we only have 3 bits we remove the carry over and get the result 000 which is correct.
This gives us a value representation that suits our wants, however the process is tedious and time consuming. Thankfully there is a method to directly get the negative representation of a value in binary two's complement representation, it goes as follows:

1. Convert the value to unsigned binary representation.
2. Invert all bits, as in every 1 becomes 0 and every 0 becomes 1.
3. Add 1 to the result of the previous step using binary addition.

Another way to think of this process is that it's counting backwards from 0.
Now that we have a negative number representation that adheres to X-X=0, we can turn any subtraction into addition simply by converting the unsigned binary number into it's negative complement and adding them together.
Another benefit of the two's complement representation is that unlike signed number representation, there is only one 0 representation and replacing the redundant 0 gives us one more value we can represent with the same number bits. Therefore the range of values represented by n-bits in two's complement is $[-2^{n-1}, 2^{n-1} - 1]$ which is $2^n - 1$ different values.
Reverting a negative value represented in two's complement to the unsigned binary representation is simply applying the same three steps in the same order.

(Extra info) Do note that you can also apply binary multiplication using both unsigned and two's complement representations, and that two's complement representation can be combined with other representations that tackle other parts of value such as rational/irrational values, complex values, etc. . . .

Exercises

1 Recap

To refresh what was taught in T1 regarding number systems, convert the following numbers from decimal to binary or vice versa:

1. $[30]_{10}$
2. $[308]_{10}$
3. $\left[2^2\right]_{10}$
4. $\left[2^5\right]_{10}$
5. $[10]_{10}$
6. $[10]_2$
7. $[10101]_2$
8. $[11111]_2$
9. $[1000000000]_2$
10. $[0000000001]_2$
11. $[0000000000]_{10}$
12. $[0000000000]_2$

---

2 Representation conversions

2.1 Two's Complement to Unsigned Binary

Convert the following 9-bit two's complement representation to unsigned 8-bit binary numbers:

1. 000000000
2. 000000001
3. 101010101
4. 110101011
5. 001010100
6. 100010101
7. 110010110
8. 000001111

2.2 Two's Complement to Decimal

Convert the following 10-bit two's complement representation to decimal numbers:

1. 1000000000
2. 0000000001
3. 1111111110
4. 0111111111
5. 1001000101
6. 0110111010
7. 1101001100
8. 0001011101

2.3 Decimal to Two's Complement

Convert the following decimal numbers to their two's complement 10-bit representation:

1. $+30$
2. -30
3. -217
4. $+308$
5. $-\left(2^5\right)$
6. $+\left(2^7+2^3+2^1\right)$
7. $+101$
8. -010

---

3 Addition

Do NOT convert the binary numbers to decimal numbers as that defeats the purpose of the question.

3.1 Addition in Unsigned Binary

Given the following 8-bit unsigned binary representation number additions, calculate the result and determine whether there is overflow or not:

1. $00000000 + 11111111$
2. $00000001 + 01111111$
3. $00000001 + 11111111$
4. $10111111 + 00011010$
5. $11001010 + 00101010$
6. $10101010 + 00101101$
7. $10101010 + 01010101$
8. $01100101 + 11101011$

3.2 Addition in Two's Complement

Given the following 8-bit unsigned binary representation number additions and subtractions, convert to two's complement representation and calculate the result and determine whether there is overflow or not:

1. $00000000 + 11111111$
2. $00000001 + 01111111$
3. $00000001 - 00000001$
4. $10111111 + 00011010$
5. $01001010 - 00101010$
6. $00101010 + 00101101$
7. $00101010 - 01010101$
8. $0 - 01100101 - 01101011$

4 Further understanding questions

The purpose of the following questions is to more deeply ingrain the ideas and intuition behind the two's complement representation. If you face difficulty answering these questions, try writing different possible solutions out on a piece of paper and contemplate the idea that the question wants to reinforce

1. How many values can be represented with an unsigned binary number using 8 bits?
2. How many values can be represented with a number in two's complement representation using 8 bits?
3. What is the range of values that can be represented through an unsigned binary number using 5 bits?
4. What is the range of values that can be represented with two's complement using 5 bits?
5. Given a number in two's complement representation, how is the sign of the value determined?
6. Is the value 0 positive or negative or both in two's complement representation?
7. What is the reason for adding 1 after inverting the bits in two's complement?
8. How do you increase the number of bits for a number in two's complement representation without changing its value?
9. Can overflow occur when adding two positive values in two's complement representation? What about two negative values, or one positive and the other negative? Why?
10. Create an 8-bit-input 9-bit-output addition circuit in Digital using AND and XOR gates.
11. Create your own exercises to practice representation conversions and binary additions.

Solution key

Recap

1. $[30]_{10} = [11110]_2$
2. $[308]_{10} = [100110100]_2$
3. $\left[2^2\right]_{10} = [100]_2$
4. $\left[2^5\right]_{10} = [100000]_2$
5. $[10]_{10} = [1010]_2$
6. $[10]_2 = [2]_{10}$
7. $[10101]_2 = [21]_{10}$
8. $[11111]_2 = [31]_{10}$
9. $[1000000000]_2 = [512]_{10}$
10. $[0000000001]_2 = [1]_{10}$
11. $[0000000000]_{10} = [0]_2$
12. $[0000000000]_2 = [0]_{10}$

Representation conversions

Two's Complement to Unsigned Binary

1. 00000000
2. 00000001
3. 10101011
4. 01010101
5. 01010100
6. 11101011
7. 01101010
8. 00001111

Two's Complement to Decimal

1. −512
2. +1
3. −2
4. +511
5. −443
6. +364
7. −177
8. +93

Decimal to Two's Complement

1. 0000011110
2. 1111100001
3. 1100100111
4. 0100110100
5. 1111100000
6. 0010001010
7. 0001100101
8. 1111110101

Addition

Addition in Unsigned Binary

1. 11111111
2. 10000000
3. 1 00000000, overflow
4. 11011001
5. 11110100
6. 11010111
7. 11111111
8. 1 01010000, overflow

Addition in Two's Complement

1. 11111111, overflow (−1)
2. 10000000, overflow (−128)
3. 1 00000000, NO overflow (0)
4. 11011001, overflow (−39)
5. 1 00100000, NO overflow (+32)
6. 01010111, NO overflow (+87)
7. 11010101, NO overflow (−43)
8. 1 00110000, overflow (+48)

Further understanding questions

1. An 8-bit unsigned binary number can represent 2^8 or 256 unique values.
2. An 8-bit number in two's complement representation can represent 2^8 or 256 unique values.
3. The range of values that can be represented with a 5-bit unsigned binary number is $[0, 31]$.
4. The range of values that can be represented with a 5-bit number in two's complement representation is $[-16, 15]$.
5. Depending on the sign of the most significant bit, a 0 represents a positive value and a 1 represents a negative value.
6. The value 0 is positive in two's complement representation.
7. Just flipping the bits will give each value an exact complement, meaning there will be a positive and a negative for every value, including 0. Having a positive and negative 0 is sub-optimal for many reasons, simplest being that by "shifting" the bits up by adding 1, we can increase the range of represented values by one. This also preserves the even or odd of the value.
8. By adding another 0 as the most significant bit for positive numbers, and a 1 as the most significant bit for negative numbers.