

Week 0010

Binary numbers

CCS1310

Computer Systems Architecture

Dr. K.Dimopoulos

OVERVIEW OF THE LECTURE

- Binary numbers
- Binary operations
- Binary forms



BINARY NUMBERS

Binary numbers

- Binary number system was invented by **Gottfried Leibniz**.
- As the word is prefixed with 'Bi' which is a Latin word and means 'two' in English (0 and 1).
- This means that while counting in binary you cannot exceed 1.
- Think about counting in decimal (0 to 9), which means that while counting in binary you cannot exceed 9 :
 - 1,2,3,4,5,6,7,8,9 , then ?
 - 10 (Start from back at **0** again, but carry **1** on the left)
- The same thing is done in Binary
 - 0, 1 then start from back at **0** again, but carry **1** on the left: 10 (which is "three")



Counting in decimal

Decimal	Explanation
0	Start at 0
1	Then 1
2-8	Count 1,2,3,4,5,6,7,8
9	This is the last digit in Decimal
10	Start from back at 0 again, but carry 1 on the left
11-98	Count 11,12,...98
99	When you run out of digits, ...
100	... start from back at 0 again, but carry 1 on the left

Counting in binary

Binary	Explanation
0	Start at 0
1	Then 1
10	Now start back at 0 again, but carry 1 on the left
11	1 more
100	start back at 0 again, and carry one to the number on the left but that number is already at 1 so it also goes back to 0 and 1 is carried to the next position on the left
101	
110	
111	
1000	Start back at 0 again (for all 3 digits), add 1 on the left

Binary to decimal demonstration

- Let's tell you something more about conversion. Conversion from Binary to Decimal is quite a simple task. All you need to do is begin from the right. Follow the steps below:
 - **STEP 1:** Write the decimal value of each digit on top of them respectively. The value which you seek to write is $2^{\text{(place value from right)}}$ beginning from 0 i.e., $2^0, 2^1, 2^2$ continuing up to 2^7 .
 - **STEP 2:** Now, multiply each digit of binary number with its value.
 - **STEP 3:** Add 'em all.
 - **STEP 4:** Result is ready :)

binary	0	1	1	0	0	1	1	1
position	7	6	5	4	3	2	1	0
power	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
result (binary x power)	$+0 \times 2^7$	$+1 \times 2^6$	$+1 \times 2^5$	$+0 \times 2^4$	$+0 \times 2^4$	$+1 \times 2^2$	$+1 \times 2^1$	$+1 \times 2^0$



BINARY OPERATIONS

Binary addition

- Binary addition is similar to Decimal addition.
- As this addition is binary, it implies that you cannot have a number greater than 1 i.e., when you do '1+1' it gives 0 with carry 1 i.e, 10.

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

(0 with 1 carry)

$$\begin{array}{r} 1\ 1\ (3) \\ +\ 1\ 0\ (2) \\ \hline 1\ 0\ 1\ (5) \end{array}$$

Binary Subtraction

- Binary subtraction is also a simple task. You just need to keep in mind i.e., whenever 0 takes borrow, it becomes 10 i.e., 2 in decimal.
- In case number to the left is zero then look for the number more left to that until you find 1. In case nothing is present to borrow then that number becomes negative. Also, the number which gives borrow is reduced by 1.

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$0 - 1 = 1$$

and 1
borrow
from
left

Binary Multiplication

- Binary multiplication requires two concepts i.e., binary addition and decimal multiplication. You can refer to the example next which describes how multiplication occurs in binary. It is similar to decimal multiplication.

- $1 \times 1 = 1$

- $1 \times 0 = 0$

$$\begin{array}{r} 110 \text{ (6)} \\ \times 101 \text{ (5)} \\ \hline \end{array}$$

$$\begin{array}{r} 110 \\ 000x \\ 110xx \\ \hline 11110 \text{ (30)} \end{array}$$

Binary Division

- The division in binary is achieved by binary subtraction. The sole motive of the Binary Division is to subtract the **divisor** from dividends until 0 is obtained or a number that cannot be further subtracted.
- The number of times you subtract is known as a **quotient** (this can be converted to binary) and the number which cannot be reduced or 0 is obtained after some step is known as **Remainder**.
- Another method which can achieve Division in binary is by using Binary Multiplication and subtraction. It is similar to Decimal Division. Multiply the divisor by 1 or 0 wherever needed and reduce the number further.



BINARY FORMS

Unsigned Binary

- Direct binary equivalent for any decimal number
- Range of integers that can be stored is determined by the available bits
 - 8-bit storage location → 0-255
 - 16-bit storage location → 0-65535
- Expand range of integers: provide more bits
- Multiple storage locations
 - E.g. Four consecutive 1-byte storage locations

Binary Coded Decimal (BCD)

- Number stored as a digit-by-digit binary representation of the original decimal integer
- Digits are individually converted to binary (4 bits per digit)
 - How many integers can be represented by an 8-bit BCD storage location?
- Calculations are hard: a number has to be broken into the 4 bit binary groupings that correspond to individual digits
- Was used when decimal precision was important (Cobol)

What about negative numbers?

- Currently, we have just looked at **unsigned** numbers - they can only be positive, as there is no sign.
- We can easily make a number **signed**, by using the leftmost bit for the sign: 0 for positive 1 for negative. Then the leftmost bit is known as the **sign bit**:
 - 10000011 would be 131 if the number is **unsigned**, but if the number is **signed**, the actual representation would be -3

Representing Integer Data

- Binary code depends on whether it is always positive or also negative
- Unsigned binary representation is for integers that are always positive
- For negative numbers?
 - Signed binary representation
 - One's complement
 - Two's complement

Signed and Magnitude

- We use + or – to represent signed integers
- Choose leftmost bit to represent the sign:
 - 0 for plus, 1 for minus
 - e.g. 0100101=+37 and 1100101=-37
- Range size remains the same, redistributed to represent numbers both positive and negative,...
- ...but magnitude is half as large

Question:

**We have 8 bits available for storage manipulation.
How many numbers can we represent?
What is the range?**

Signed and Magnitude

- Disadvantages:

- Two representations for zero
- Arithmetic rules don't work simply: Addition algorithm depends on the sign bits

- Try $5 + (-5) = 0$

0 00101

1 00101

1 01010 (Oops! Not zero!)

Let's try: $4 + 2 = 6$, $4 - 2 = 2$, $2 - 4 = -2$

Overflows

- It is possible to have a combination of numbers that adds up to a result outside of the range
 - overflow
- e.g. if we have 2 places of range, Max is 99
 - $64+65=129$ Out of Range
- Extra number indicates the overflows

Example: Overflow

- Detection of Overflows:

- if both inputs are of the same sign and the sign of the result is different overflow has occurred

- Example:

$$0\ 1000000 = 64$$

$$0\ 1000001 = 65$$

$$1\ 0000001 = -1 \text{ (in signed and magnitude)}$$

ONE'S Complement



- Uses most significant bit as the sign
 - 0 for +, 1 for -
- Magnitude of negative numbers is represented by taking the 1's complement of each bit: replacing 0 with 1 and 1 with 0 on a bit-by-bit basis

- Try $5 + (-5) = 0$

00 0101

11 1010

11 1111 (negative representation of zero)

NOTE: If after the addition you have a carry bit you add it again

Let's try: $5 + (-3) = 2$ and $3 + (-5) = -2$

- Might need an extra addition for the carry bit

TWO'S Complement



- Uses most significant bit as the sign
 - 0 for +, 1 for -
- Taking the negative of a number is a two step process:
 - Step 1: Take one's complement
 - Step 2: Add 1 to the result (Ignore any carry bits)
- QUESTIONS:
 - Find the 2's complement of 3 (-3) and then backwards
 - How about the range of 2's complement? Assume a 4 bit cell

Example: TWO'S Complement

$$0\ 00101 = 5$$

$$\begin{array}{r} 1\ 11110 = -2 \text{ (in two's complement)} \\ \hline \end{array}$$

$$0\ 00011 = 3$$

$$0\ 00011 = 3$$




$$\begin{array}{r} 1\ 11011 = -5 \text{ (in two's complement)} \\ \hline \end{array}$$

$$1\ 11110 = -2 \text{ (in two's complement)}$$




Trade OFFs



■ 1's complement

- makes it easier to change the sign of the number 
- addition requires an extra end-around carry step 
- algorithm must check for and convert -0 to 0 at the end of each operation 

■ 2's complement

- simplifies the addition operation 
- ...but has an additional add operation any time the sign change operation is required 
- only one representation for 0 

Is $0 = -0$ in TWO's Complement?



- $0 = 00000000$
- Bitwise not 11111111
- Add 1 to LSB $+1$
- Result $1\ 00000000$
- Overflow is ignored, so:
- $-0 = 0$

All Together ...

Signed

<i>Binary</i>	<i>Signed Decimal</i>
00000000	+0
00000001	+1
00000010	+2
...	...
01111101	+125
01111110	+126
01111111	+127
10000000	-0
10000001	-1
10000010	-2
...	...
11111101	-125
11111110	-126
11111111	-127

1's Complement

<i>Binary</i>	<i>Signed Decimal</i>
00000000	+0
00000001	+1
00000010	+2
...	...
01111101	+125
01111110	+126
01111111	+127
10000000	-127
10000001	-126
10000010	-125
...	...
11111101	-2
11111110	-1
11111111	-0

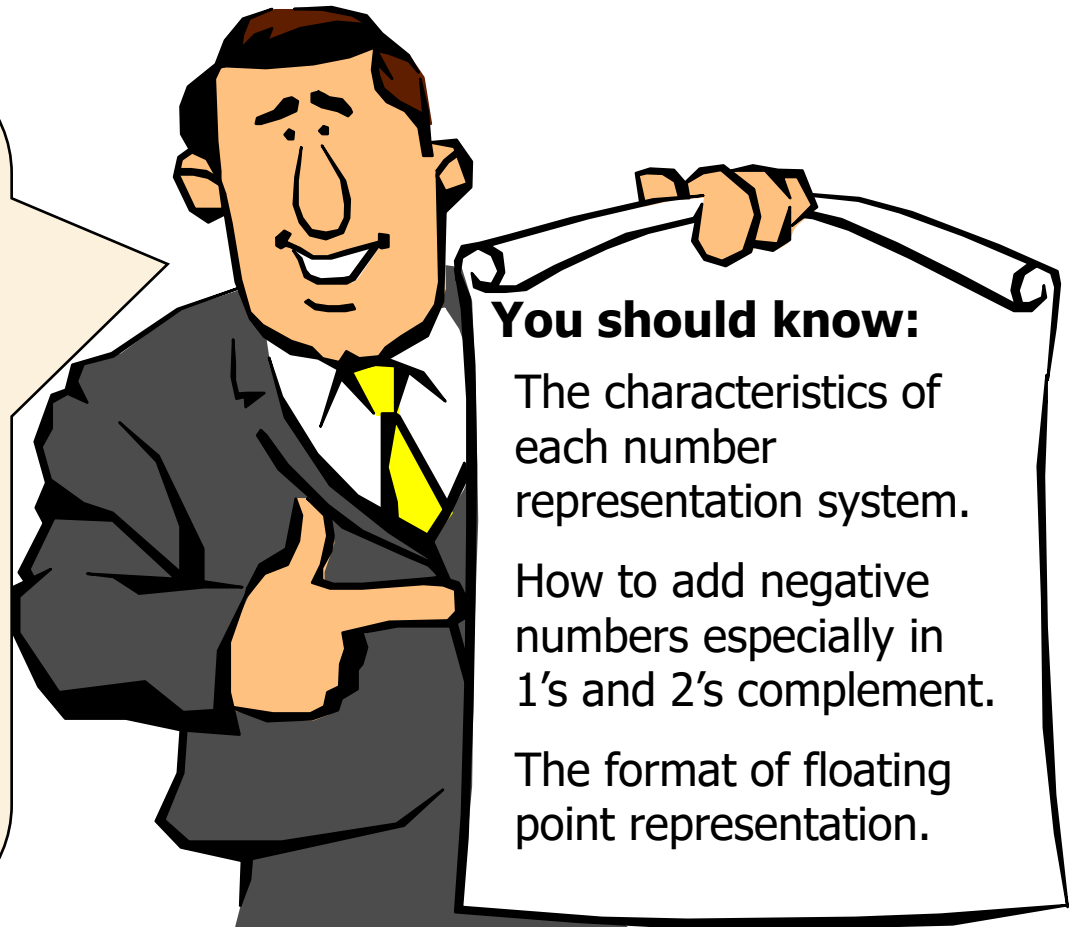
2's Complement

<i>Binary</i>	<i>Signed Decimal</i>
00000000	+0
00000001	+1
00000010	+2
...	...
01111101	+125
01111110	+126
01111111	+127
10000000	-128
10000001	-127
10000010	-126
...	...
11111101	-3
11111110	-2
11111111	-1

Summary

Binary addition is a computer's basic operation. All other operations can be done just with addition.

There are many different ways that numbers can be represented. Unsigned, sign and magnitude, BCD, 1's and 2's complement are the most important.



You should know:

The characteristics of each number representation system.

How to add negative numbers especially in 1's and 2's complement.

The format of floating point representation.