Binary Arithmetic & Signed Numbers

Computer Science Department

Outline

- Adding Binary Numbers
- Signed Numbers
- Subtracting with Signed Numbers

Adding Binary Numbers

Binary Addition

Binary Addition Rules for Two Numbers

	1)	2)			3)	4)
						1
	0	0		1	1	1
	+ 0	+ 1	OR	+ 0	+ 1	+ 1
Sum	0	1		1	10	11

All values are expressed in binary.

Three 1's will occur during a carry operation.

Binary Addition - Example

01111+00110 =

```
     1   1  1
     0  1 1 1 1      →   (15)10
+    0  0 1 1 0      →   (6)10
   ---------------
      1  0 1 0 1      =     (21)10
```

Binary Addition

11010011+01010110=

```
      1      1      1    1
      1 1 0 1  0  0  1  1      →(211)10
 +    0 1 0 1  0 1   1   0     →( 86)10
    ---------------------------
   1 0 0  1  0 1  0   0  1     = (297)10
```

Adding Binary Fractions

Adding Binary Fractions

1. First, we align the two numbers so that the radix point of each number is located in the same column.

```
  110.01
+   1.011
```

2. Next, we fill in the blank spaces with 0s and add the two numbers together.

```
  110.010
+ 001.011
```

3. The first column adds to 1.

```
  110.010
+ 001.011
        1
```

4. The second column adds to 10_2, so we write a 0 below it and carry a 1 to the next column.

```
      1
  110.010
+ 001.011
       01
```

5. All of the remaining columns add to 1, so we write 1 below them.

```
      1
  110.010
+ 001.011
  111.101
```

6. This gives us a final answer of 111.101_2.

```
      1
  110.010
+ 001.011
  111.101
```

Bits carry across the radix point

Add 10.1b + 10.1b.

```
    1 <--- Carry bit
   10.1b
+  10.1b
---------
  101.0b
```

Verify that 10.1b + 10.1b equals 101.0b.

10.1b = 2.5d

```
   2.5
+  2.5
-------
   5.0 = 101.0b
```

Signed Numbers

Signed Numbers

Until now, we have only considered positive numbers in our study of binary arithmetic

What about negative numbers?

Representing numbers(integers)

Fixed, finite number of bits.

Bits	bytes	C/C++	Intel	Sun
8	1	char	[s]byte	byte
16	2	short	[s]word	half
32	4	int or long	[s]dword	word
64	8	long long	[s]qword	xword

Representing numbers (integers)

Fixed, finite number of bits.

bits	signed	unsigned
8	$-2^7..+2^7-1$	$0..+2^8-1$ ($2^8 = 256$)
16	$-2^{15}..+2^{15}-1$	$0..+2^{16}-1$ ($2^{16} = 65,536$)
32	$-2^{31}..+2^{31}-1$	$0..+2^{32}-1$ ($2^{32} = 4,294,967,296$)
64	$-2^{63}..+2^{63}-1$	$0..+2^{64}-1$ ($2^{64} = 18,446,744,073,709,551,616$)

In general, for k bits, the unsigned range is $[0..+2^k-1]$ and the signed range is $[-2^{k-1}..+2^{k-1}-1]$.

Signed Numbers

Example:

$(5)_{10} = (101)_2$

Positive 5 is 0 1 0 1

Negative 5 is 1 1 0 1

The Problem : We need to specify how many bits in our numbers so we can be certain which bit is representing the sign !!!

Methods for representing signed integers.

1. **Signed Magnitude**
2. **1’s Complement**
3. **2’s Complement**

Signed Numbers

- **Signed Magnitude**

add an extra digit to the front of our binary number to indicate whether the number is positive or negative.

this digit called sign bit.

0 for positive
1 for negative

Signed Magnitude

Ex. 4-bit signed magnitude

1 bit for sign
3 bits for magnitude

	$+N$	$-N$
0	0000	1000
1	0001	1001
2	0010	1010
3	0011	1011
4	0100	1100
5	0101	1101
6	0110	1110
7	0111	1111

Sign

Signed Magnitude

Ex. 4-bit signed magnitude

1 bit for sign
3 bits for magnitude

	$+N$	$-N$
0	0000	1000
1	0001	1001
2	0010	1010
3	0011	1011
4	0100	1100
5	0101	1101
6	0110	1110
7	0111	1111

magnitude

Signed Numbers

1 1 0 1 is 13 or -5

- **One's Complement**

Representing a signed number with 1's

Complement is done by changing all the bits that are 1 to 0 and all bits that are 0 to 1.

Signed Numbers - Examples

- Represent **-5** in 1's complement by using 4-bit arithmetic?

  0101 → 1010

- Represent **-1** in 1's complement ?

  0001 → 1110

1’s complement (Alternative def.)

Let x be a non-negative number.
Then –x is represented by

$b^D-1+(-x)$, where

b = base
D = (total) # of bits (including the sign bit)

Example: Let b=2 and D=4.

Then -1 is represented by $2^4-1-1 = 14_{10}$ or 1110_2.
-5 is represented by $2^4-1-5 = 10_{10}$ or 1010_2.

4-bit binary numbers in **1's complement notation**

- All of the negative values begin with a 1
- Here MSB always tells us the sign of
- the number
- 2 ways of representing the number zero.

- **Rule**(we already know):

If *x* is positive, simply convert *x* to binary.
If *x* is negative, write the positive value of *x* in binary
Reverse each bit.

Binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-0
1110	-1
1101	-2
1100	-3
1011	-4
1010	-5
1001	-6
1000	-7

Signed Numbers

- **Two's Complement**

2's comp= (1'comp) + 1

- Represent -5 in 2's complement by using 4-bit arithmetic?

(0101)1's→ 1010

2's + 1

----

1 0 1 1 =(-5)

4-bit binary numbers in Two's Complement Notation

- Most significant bit to represent the sign.
- We only have one way to represent 0 in 2's complement.

Rule:

If *x* is positive, simply convert *x* to binary.
If *x* is negative, write the positive value of *x* in binary
Reverse each bit.
Add 1 to the complemented number.

Binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Two's Complement

Problems with sign-magnitude and 1's complement

1. two representations of zero (+0 and –0)
2. arithmetic circuits are complex

   How to add two sign-magnitude numbers?

   e.g., try 2 + (-3)

   How to add two one's complement numbers?

   e.g., try 4 + (-3)

Two's complement **representation developed to make circuits easy for arithmetic.**

for each positive number (X), assign value to its negative (-X),
such that X + (-X) = 0 with "normal" addition, ignoring carry out

```
  00101  (5)
+ 11011  (-5)
  00000  (0)
```

2's Complement Addition

- **Easy**
- **No special rules**
- **Just add**

Subtraction with Signed Numbers

What is -5 plus +5?

Zero, of course, but let's see

Sign-magnitude

```
-5:    10000101
+5:   +00000101
       --------
       10001010
```

Twos-complement

```
      11111111
-5:    11111011
+5:   +00000101
       --------
       00000000
```

Subtracting with Signed Numbers

- Convert our subtraction problems to addition
- Example: **subtracting 1_{10} from 7_{10}.**
- Solution:
  1. Convert 1_{10} to -1_{10} with either 1's or 2's complementation.
  2. Add -1_{10} to 7_{10}.
  3. **Adjust our answer**:

- If sum in step 2 exceeds the number of bits in our representation, then we have **overflow**
- We handle the extra bit **differently** in 1's and 2's complement.
- In 1's complement, we *add the overflow bit to our sum* to obtain the final answer.
- In 2's complement, we simply *discard the extra bit* to obtain the final answer.

Subtraction with One's Complement with Overflow

Let's consider how we would solve our problem of subtracting 1_{10} from 7_{10} using 1's complement.

1. First, we need to convert 0001_2 to its negative equivalent in 1's complement.	0111 - 0001	(7) - (1)
2. To do this we change all the 1's to 0's and 0's to 1's. Notice that the most-significant digit is now 1 since the number is negative.	0001 -> 1110	
3. Next, we add the negative value we computed to 0111_2. This gives us a result of 10101_2.	0111 + 1110 10101	(7) +(-1) (?)
4. Notice that our addition caused an overflow bit. Whenever we have an overflow bit in 1's complement, we add this bit to our sum to get the correct answer. If there is no overflow bit, then we leave the sum as it is.	0101 + 1 0110	 (6)
5. This gives us a final answer of 0110_2 (or 6_{10}).	0111 - 0001 0110	(7) - (1) (6)

Subtraction with One's Complement without Overflow

Subtract 7_{10} from 1_{10} using 1's complement.

1. First, we state our problem in binary.

```
  0001     (1)
- 0111   - (7)
```

2. Next, we convert 0111_2 to its negative equivalent and add this to 0001_2.

```
  0001     (1)
+ 1000   +(-7)
  1001     (?)
```

3. This time our results does not cause an overflow, so we do not need to adjust the sum. Notice that our final answer is a negative number since it begins with a 1. Remember that our answer is in 1's complement notation so the correct decimal value for our answer is -6_{10} and not 9_{10}.

```
  0001     (1)
+ 1000   +(-7)
  1001    (-6)
```

2's Complement Subtraction

- **Easy**
- **No special rules**
- **Just subtract, well … actually … just add!**

$$A - B = A + (-B)$$

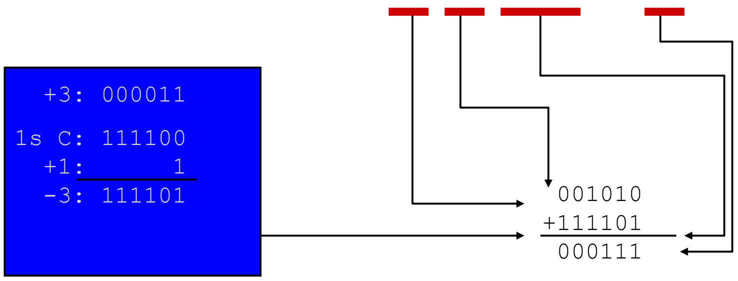
add

2's complement of B

What is 10 subtract 3?

7, of course, but…
Let's do it (we'll use 6-bit values)

10 – 3 = 10 + (-3) = 7

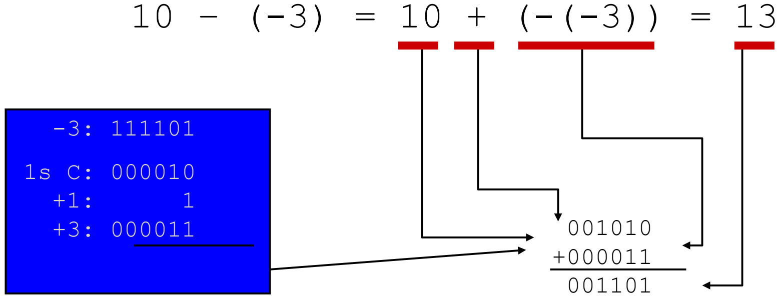


What is 10 subtract -3?

(-(-3)) = 3

13, of course, but…
Let's do it (we'll use 6-bit values)

10 – (-3) = 10 + (-(-3)) = 13



What is subtracting 1_{10} from 7_{10} ?

Now let's consider how we would solve our problem of subtracting 1_{10} from 7_{10} using 2's complement.

1. First, we need to convert 0001_2 to its negative equivalent in 2's complement.

```
  0111     (7)
- 0001   - (1)
```

2. To do this we change all the 1's to 0's and 0's to 1's and add one to the number. Notice that the most-significant digit is now 1 since the number is negative.

```
0001 -> 1110
           1
        1111
```

3. Next, we add the negative value we computed to 0111_2. This gives us a result of 10110_2.

```
  0111     (7)
+ 1111   +(-1)
 10110     (?)
```

4. Notice that our addition caused an overflow bit. Whenever we have an overflow bit in 2's complement, we discard the extra bit. This gives us a final answer of 0110_2 (or 6_{10}).

```
  0111     (7)
- 0001   - (1)
  0110     (6)
```

H.W

Lab 1 . P8,9

Q.1,2,3,4,8,10