

# *10*

## *Number Systems*

ENGR 3410 - Computer Architecture

Mark L. Chang

Fall 2007

## Decimal (Base 10) Numbers

---

- Positional system - each digit position has a value

$$2534 = 2*1000 + 5*100 + 3*10 + 4*1$$

- Alternate view: Digit position  $i$  from the right = Digit \*  $10^i$   
(rightmost is position 0)

$$2534 = 2*10^3 + 5*10^2 + 3*10^1 + 4*10^0$$

## Base R Numbers

---

- Each digit in range 0..(R-1)

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F ...

A = 10

B = 11

C = 12

D = 13

E = 14

F = 15

- Digit position  $i = \text{Digit} * R^i$

$D_3 D_2 D_1 D_0 \text{ (base R)} = D_3 * R^3 + D_2 * R^2 + D_1 * R^1 + D_0 * R^0$

## Conversion to Decimal

---

- Binary:  $(101110)_2$
- Octal:  $(325)_8$
- Hexadecimal:  $(E32)_{16}$

## Conversion Decimal

---

- Binary:  $(110101)_2$
- Octal:  $(524)_8$
- Hexadecimal:  $(A6)_{16}$

## Conversion of Decimal to Binary (Method 1)

---

- For positive, unsigned numbers
  - Successively subtract the greatest power of two less than the number from the value. Put a 1 in the corresponding digit position
- 
- $2^0=1$     $2^4=16$     $2^8=256$     $2^{12}=4096$  (4K)
  - $2^1=2$     $2^5=32$     $2^9=512$     $2^{13}=8192$  (8K)
  - $2^2=4$     $2^6=64$     $2^{10}=1024$  (1K)
  - $2^3=8$     $2^7=128$     $2^{11}=2048$  (2K)

## Decimal to Binary Method 1

---

- Convert  $(2578)_{10}$  to binary
  
- Convert  $(289)_{10}$  to binary

## Conversion of Decimal to Binary (Method 2)

---

- For positive, unsigned numbers
- Repeatedly divide number by 2. Remainder becomes the binary digits (right to left)
- Explanation:



## Decimal to Binary Method 2

---

- Convert  $(289)_{10}$  to binary

## Decimal to Binary Method 2

---

- Convert  $(85)_{10}$  to binary

## Converting Binary to Hexadecimal

---

- 1 hex digit = 4 binary digits
- Convert  $(11100011010111010011)_2$  to hex
  
- Convert  $(A3FF2A)_{16}$  to binary

## Converting Binary to Octal

---

- 1 octal digit = 3 binary digits
- Convert  $(10100101001101010011)_2$  to octal
  
- Convert  $(723642)_8$  to binary

## Converting Decimal to Octal/Hex

---

- Convert to binary, then to other base
- Convert  $(198)_{10}$  to Hexadecimal
  
- Convert  $(1983020)_{10}$  to Octal

## Arithmetic Operations

---

Decimal:

$$\begin{array}{r} 5\ 7\ 8\ 9\ 2 \\ +\ 7\ 8\ 9\ 5\ 6 \\ \hline \end{array}$$

Binary:

$$\begin{array}{r} 1\ 0\ 1\ 0\ 1\ 1\ 1 \\ +\ 0\ 1\ 0\ 0\ 1\ 0\ 1 \\ \hline \end{array}$$

Decimal:

$$\begin{array}{r} 5\ 7\ 8\ 9\ 2 \\ -\ 3\ 2\ 9\ 4\ 6 \\ \hline \end{array}$$

Binary:

$$\begin{array}{r} 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0 \\ -\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1 \\ \hline \end{array}$$

## Arithmetic Operations (cont.)

---

Binary:

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ * 1\ 0\ 1\ 1 \\ \hline \end{array}$$

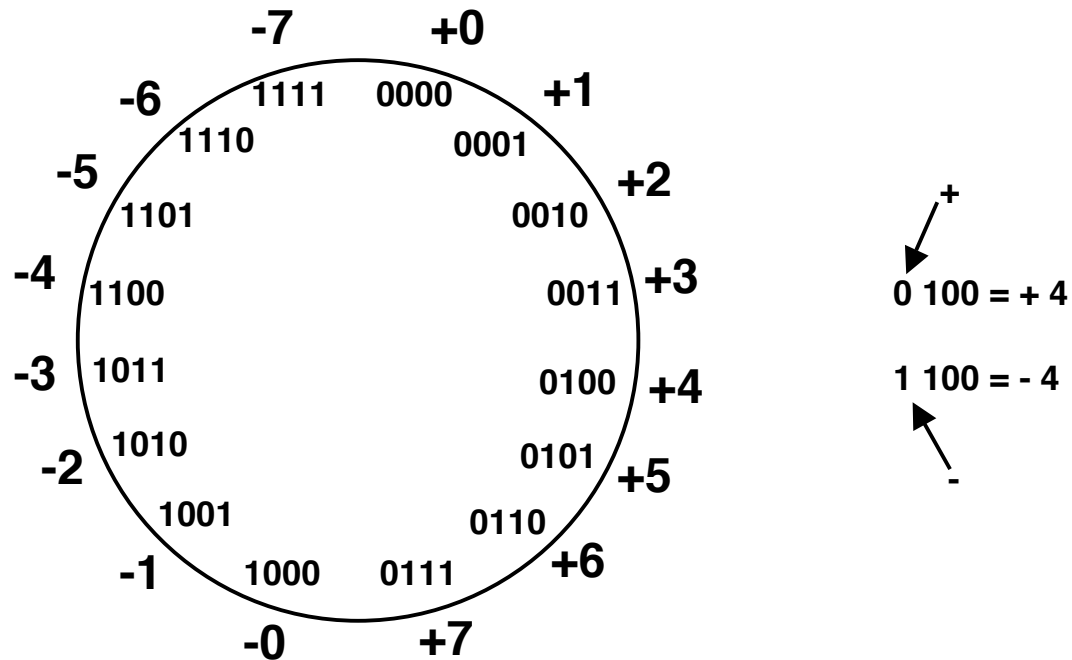
## Negative Numbers

---

- Need an efficient way to represent negative numbers in binary
  - Both positive & negative numbers will be strings of bits
  - Use fixed-width formats (4-bit, 16-bit, etc.)
- Must provide efficient mathematical operations
  - Addition & subtraction with potentially mixed signs
  - Negation (multiply by -1)



# Sign/Magnitude Representation



High order bit is sign: 0 = positive (or zero), 1 = negative

Three low order bits is the magnitude: 0 (000) thru 7 (111)

Number range for n bits =  $\pm 2^{n-1} - 1$

Representations for 0:

## Sign/Magnitude Addition

---

Idea: Pick negatives so that addition/subtraction works

$$\begin{array}{r} 0\ 0\ 1\ 0\ (+2) \\ +\ 0\ 1\ 0\ 0\ (+4) \\ \hline \end{array}$$

$$\begin{array}{r} 1\ 0\ 1\ 0\ (-2) \\ +\ 1\ 1\ 0\ 0\ (-4) \\ \hline \end{array}$$

$$\begin{array}{r} 0\ 0\ 1\ 0\ (+2) \\ +\ 1\ 1\ 0\ 0\ (-4) \\ \hline \end{array}$$

$$\begin{array}{r} 1\ 0\ 1\ 0\ (-2) \\ +\ 0\ 1\ 0\ 0\ (+4) \\ \hline \end{array}$$

Bottom line: Basic mathematics are too complex in Sign/Magnitude

## Idea: Pick negatives so that addition works

---

- Let  $-1 = 0 - (+1)$ :

$$\begin{array}{r} 0\ 0\ 0\ 0\ (0) \\ -\ 0\ 0\ 0\ 1\ (+1) \\ \hline \end{array}$$

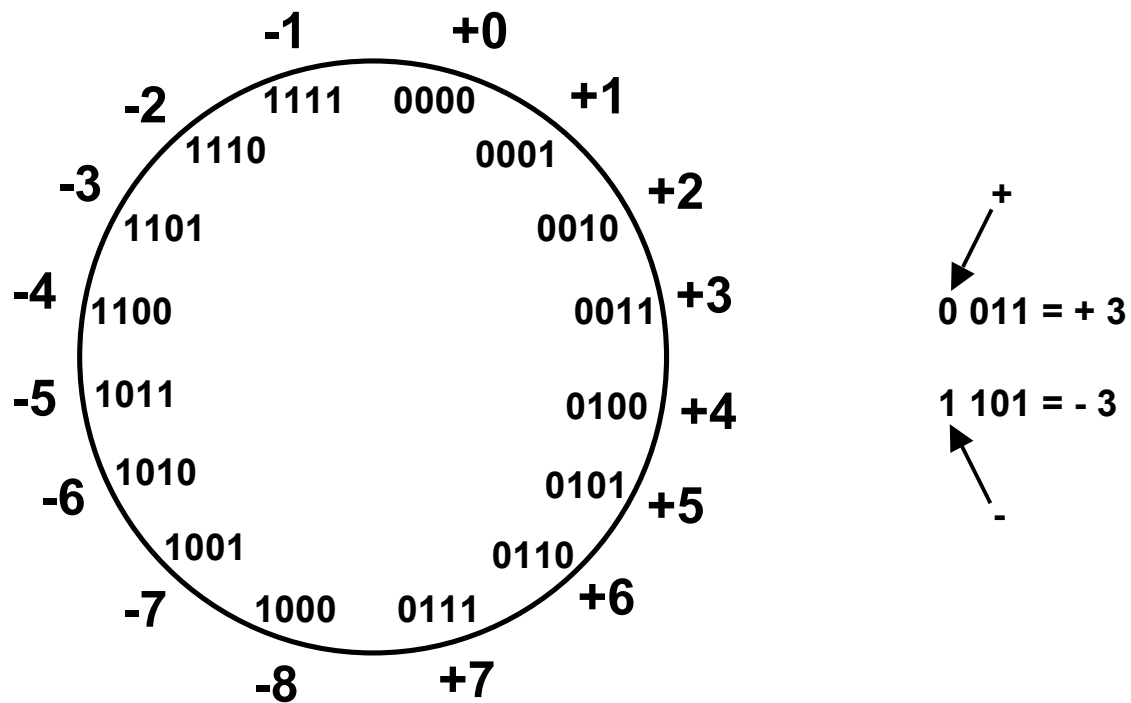
- Does addition work?

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

- Result: Two's Complement Numbers

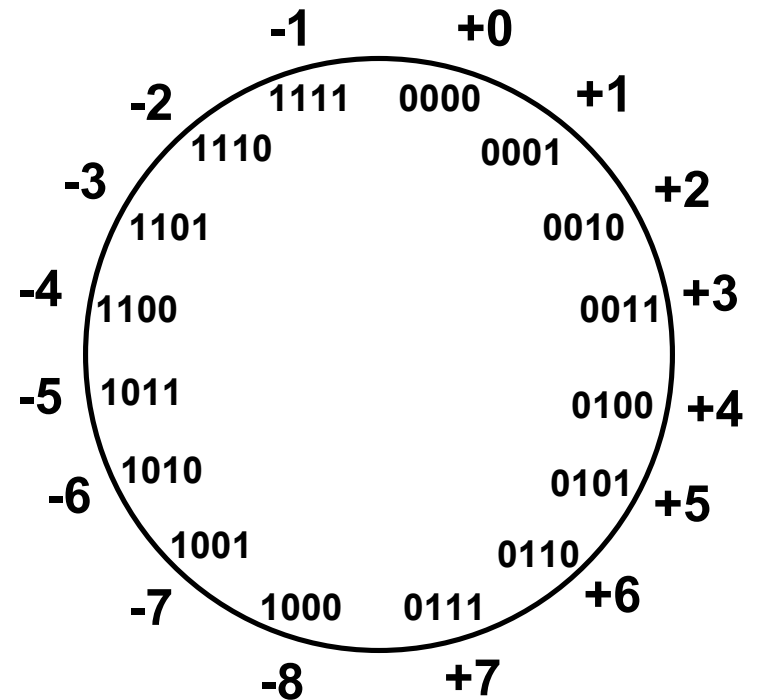
# Two's Complement

- Only one representation for 0
- One more negative number than positive number
- Fixed width format for both pos. & neg. numbers



## Negating in Two's Complement

- Flip bits & Add 1
- Negate  $(0010)_2$  (+2)
  
- Negate  $(1110)_2$  (-2)



## Addition in Two's Complement

---

$$\begin{array}{r} 0010 (+2) \\ + 0100 (+4) \\ \hline \end{array}$$

$$\begin{array}{r} 1110 (-2) \\ + 1100 (-4) \\ \hline \end{array}$$

$$\begin{array}{r} 0010 (+2) \\ + 1100 (-4) \\ \hline \end{array}$$

$$\begin{array}{r} 1110 (-2) \\ + 0100 (+4) \\ \hline \end{array}$$

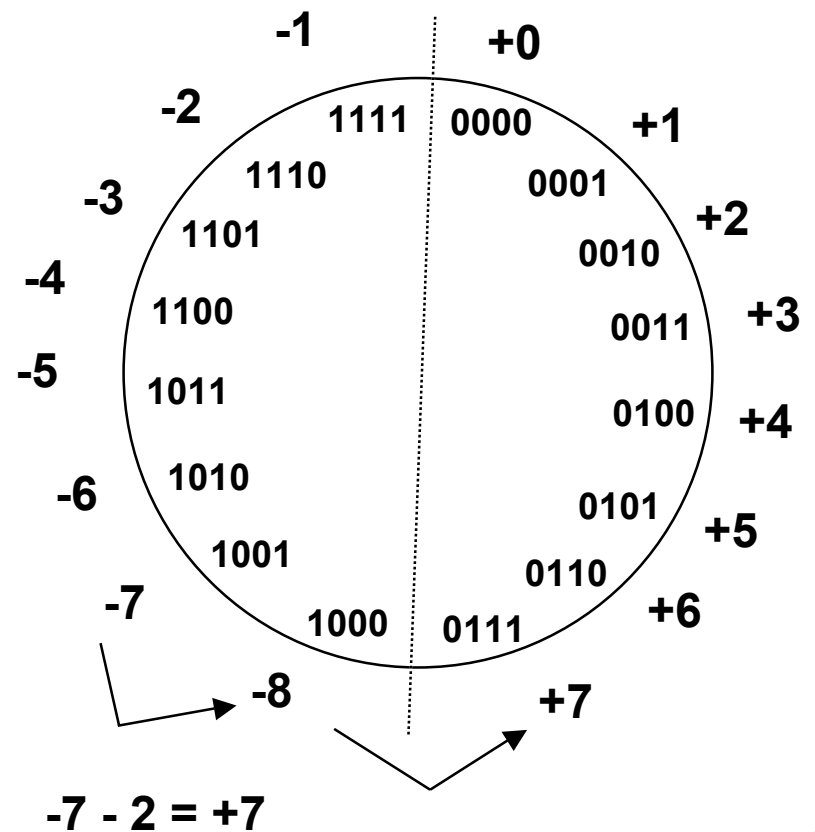
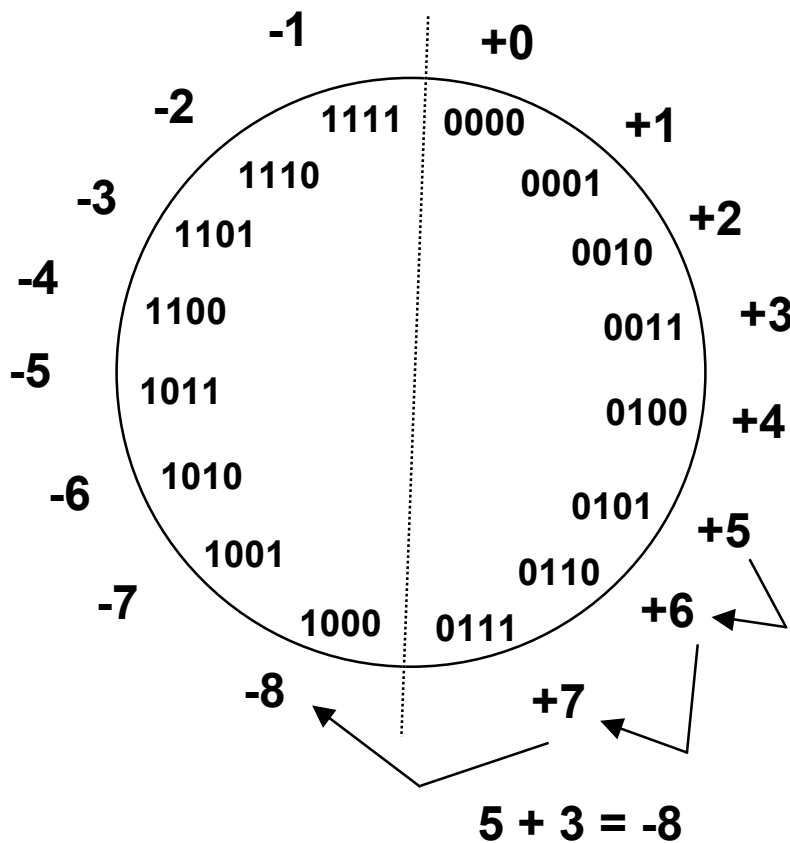
## Subtraction in Two's Complement

---

- $A - B = A + (-B) = A + \overline{B} + 1$
- 0010 - 0110
- 1011 - 1001
- 1011 - 0001

# Overflows in Two's Complement

Add two positive numbers to get a negative number  
or two negative numbers to get a positive number





## Overflow Detection in Two's Complement

---

$$\begin{array}{r} 5 \\ \underline{3} \end{array} \quad \begin{array}{r} 0101 \\ \underline{0011} \end{array}$$

-8

Overflow

$$\begin{array}{r} -7 \\ \underline{-2} \end{array} \quad \begin{array}{r} 1001 \\ \underline{1110} \end{array}$$

7

Overflow

$$\begin{array}{r} 5 \\ \underline{2} \end{array} \quad \begin{array}{r} 0101 \\ \underline{0010} \end{array}$$

7

No overflow

$$\begin{array}{r} -3 \\ \underline{-5} \end{array} \quad \begin{array}{r} 1101 \\ \underline{1011} \end{array}$$

-8

No overflow

Overflow when carry in to sign does not equal carry out

## Converting Decimal to Two's Complement

---

- Convert absolute value to binary, then negate if necessary
- Convert  $(-9)_{10}$  to 6-bit Two's Complement
  
- Convert  $(9)_{10}$  to 6-bit Two's Complement

## Converting Two's Complement to Decimal

---

- If Positive, convert as normal;  
If Negative, negate then convert.
- Convert  $(11010)_2$  to Decimal
- Convert  $(01011)_2$  to Decimal

## Sign Extension

---

- To convert from N-bit to M-bit Two's Complement ( $N > M$ ), simply duplicate sign bit:
- Convert  $(1011)_2$  to 8-bit Two's Complement
- Convert  $(0010)_2$  to 8-bit Two's Complement