0010
Number Systems

ENGR 3410 – Computer Architecture
Fall 2010

Decimal (Base 10) Numbers

- Positional system: Each digit (0, 1, 2, 3, 4, 5, 6, 7, 8, or 9) has a value that depends on its position.

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

- Value of Digit in 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)] (need a glyph for each value)
  For R = 16 (base 16, hexadecimal), each digit is in
  { 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 = Digit * R^i
  $D_3 D_2 D_1 D_0$ (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 non-negative integers
- 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 non-negative integers
- Repeatedly divide number by 2. Remainder becomes the binary digits (right to left)

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

Decimal to Binary Method 2

This works for any base R. Why?

Decimal to Binary Method 2

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

Converting Binary to Hexadecimal

- 1 hex digit = 4 binary digits (start grouping from right)
- Convert $(11100011010111010011)_2$ to hex

- Convert $(A3FF2A)_{16}$ to binary

Converting Binary to Octal

- 1 octal digit = 3 binary digits (start grouping from right)
- Convert $(10100101001101010011)_2$ to octal

- Convert $(723642)_8$ to binary

Converting Decimal to Octal/Hex

- Could divide by powers of 8/16 or use successive division.
- Let's convert to binary, then to other base
- Convert $(198)_{10}$ to Hexadecimal

- Convert $(1983020)_{10}$ to Octal

Arithmetic Operations

Decimal:

```
  5 7 8 9 2
+ 7 8 9 5 6
```

Binary:

```
  1 0 1 0 1 1 1
+ 0 1 0 0 1 0 1
```

Decimal:

```
  5 7 8 9 2
- 3 2 9 4 6
```

Binary:

```
  1 0 1 0 0 1 1 0
- 0 0 1 1 0 1 1 1
```

Arithmetic Operations (cont.)

Binary:

```
  1 0 0 1
* 1 0 1 1
---------
```

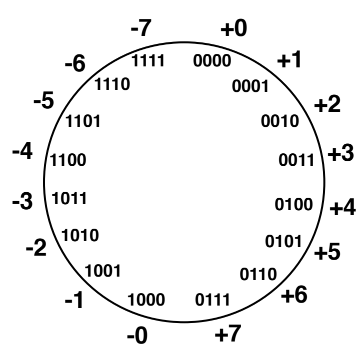

Fly in the ointment: Negative numbers

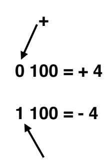
- Addition, subtraction, multiplication work for non-negative numbers.
- But there *are* negative numbers. How to represent them?
  - Sign-Magnitude
  - Ones Complement
  - Twos Complement

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 = +/- (2^{n-1} - 1)

Representations for 0:

Sign/Magnitude Addition

Idea: Pick negatives so that addition/subtraction works

```
  0 0 1 0 (+2)
+ 0 1 0 0 (+4)
```

```
  1 0 1 0 (-2)
+ 1 1 0 0 (-4)
```

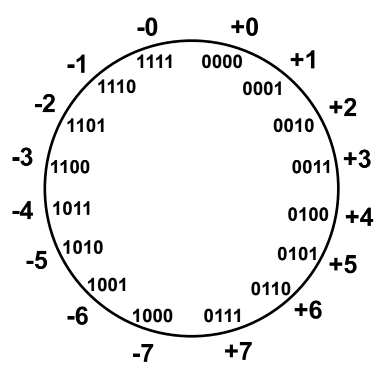
```
  0 0 1 0 (+2)
+ 1 1 0 0 ( -4)
```

```
  1 0 1 0 ( -2)
+ 0 1 0 0 (+4)
```

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

Ones Complement

- Sign bit, as in sign/magnitude
- Negative number has bits flipped: -2 is represented as 1101




```
  0 0 1 0 (+2)
+ 1 1 0 1 (-2)
```

```
  0 0 1 0 (+2)
+ 1 0 1 1 (-4)
```

```
  1 1 0 1 (-2)
+ 1 0 1 1 (-4)
```

Idea: Pick negatives so that addition works

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

```
    0   0   0   0    (  0)
 –  0   0   0   1    (+1)
```

- Generally, represent -N by the n-bit binary representation of 2^n - N

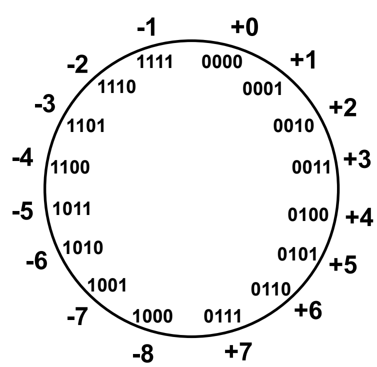
- Does addition work?

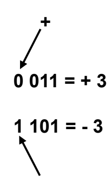
```
    0   0   1   0    (+2)
 +  1   1   1   1    (−1)
```

- Result: Two's Complement Numbers

Two's Complement

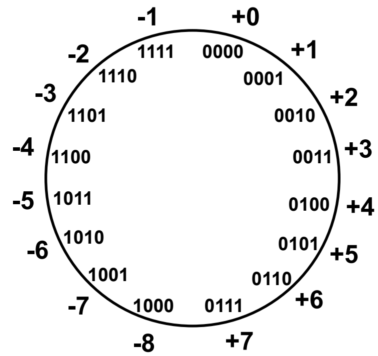
- Only one representation for 0
- One more negative number than positive number
- Fixed width format for both pos. & neg. numbers
- Bits(-N) = Bits(2^n – N) (limited to n bits)





Negating in Two's Complement

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



- Negate $(1110)_2$ (-2)

Addition in Two's Complement

```
  0 0 1 0 (+2)
+ 0 1 0 0 (+4)
```

```
  1 1 1 0 (-2)
+ 1 1 0 0 (-4)
```

```
  0 0 1 0 (+2)
+ 1 1 0 0 ( -4)
```

```
  1 1 1 0 ( -2)
+ 0 1 0 0 (+4)
```

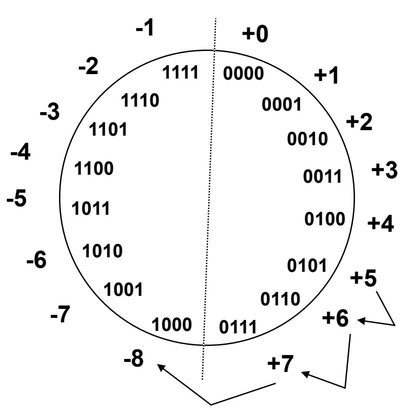
Subtraction in Two's Complement

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

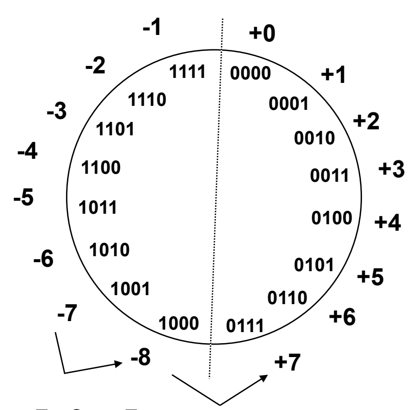
Overflows in Two's Complement

Add two positive numbers to get a negative number

or two negative numbers to get a positive number



5 + 3 = -8



-7 - 2 = +7

Overflow Detection in Two's Complement

5	0 1 0 1	-7	1 0 0 1
3	0 0 1 1	-2	1 1 1 0
-8		7	
Overflow		**Overflow**	

5	0 1 0 1	-3	1 1 0 1
2	0 0 1 0	-5	1 0 1 1
7		-8	
No overflow		**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 (M > N), simply duplicate sign bit:
- Convert $(1011)_2$ to 8-bit Two's Complement
- Convert $(0010)_2$ to 8-bit Two's Complement