CSE 31
Computer Organization

Lecture 2 – Integer Representations
C Programming

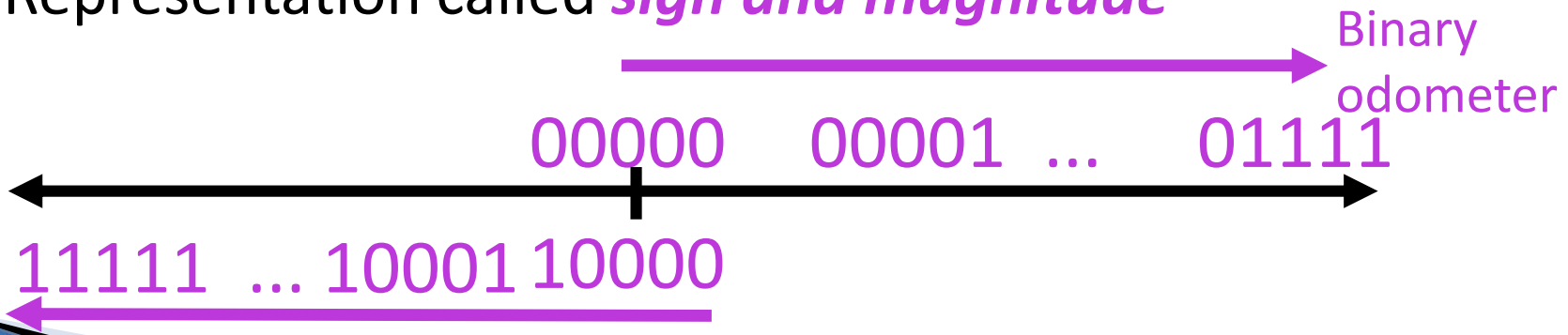
Announcement

- Lab #0 starts this week
  - Due in one week
  - Must demo your work to TA or instructor within a week after duedate
- Reading assignment
  - Chapter 1.1 – 1.3 of zyBook
    - Do all Participation Activities in each section
    - Access through CatCourses
    - Due Wednesday (8/29) at 11:59pm
  - Chapter 4-6 of K&R (C book) to review on C/C++ programming

Negative Numbers

- So far, ***unsigned numbers***

- Obvious solution: define leftmost bit to be sign!
  - 0 → + , 1 → –
  - Rest of bits can be numerical value of number
- Representation called ***sign and magnitude***

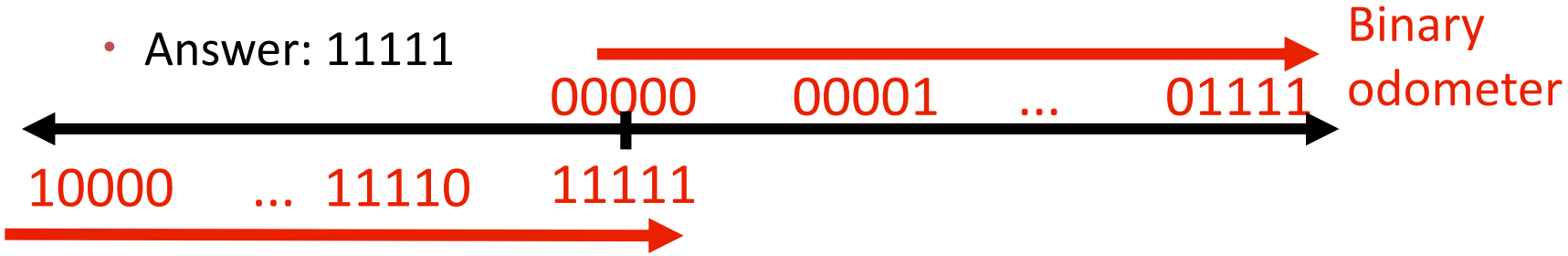


Shortcomings of Sign Magnitude?

- Arithmetic circuit complicated
  - Special steps depending whether signs are the same or not
- Also, two zeros
  - 0x00000000 = $+0_{ten}$
  - 0x80000000 = -0_{ten}
  - What would two 0s mean for programming?
- Also, incrementing "binary odometer", sometimes increases values, and sometimes decreases!
- Therefore sign and magnitude abandoned

Another try

- Complement the bits
  - Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
  - Note: positive numbers have leading 0s, negative numbers have leadings 1s.
  - Called ***One's Complement***
  - What is -00000?
    - Answer: 11111



  - How many positive numbers in N bits? 2^{N-1}
  - How many negative numbers? 2^{N-1}

Shortcomings of One's complement?

- Arithmetic is less complicate than sign & magnitude.
- Still two zeros
  - 0x00000000 = $+0_{ten}$
  - 0xFFFFFFFF = -0_{ten}
- Although used for a while on some computer products, one's complement was eventually abandoned because another solution was better.

Standard Negative # Representation

- Problem is the negative mappings "overlap" with the positive ones (the two 0s). Want to shift the negative mappings left by one.
  - Solution! For negative numbers, complement, then add 1 to the result
- As with sign and magnitude, & one's complement, leading 0s → positive, leading 1s → negative
  - 000000...xxx is ≥ 0, 111111...xxx is < 0
  - except 1…1111 is -1, not -0
- This representation is ***Two's Complement***
- This makes the hardware simple!

In C: short, int, long long, intN_t (C99) are all signed integers.

Two's Complement Formula

- Can represent positive and negative numbers in terms of the bit value times a power of 2:

  $d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$

- Example: 1101_{two}

  $= 1 \times -(2^3) + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

  $= -2^3 + 2^2 + 0 + 2^0$

  $= -8 + 4 + 0 + 1$

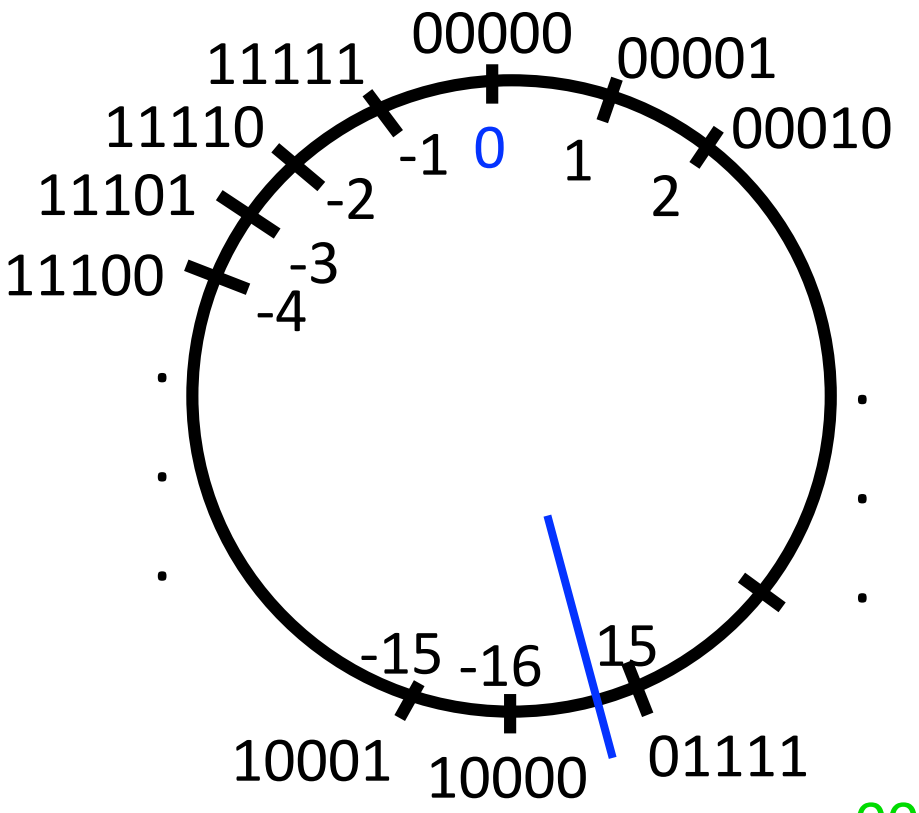
  $= -8 + 5$

  $= -3_{ten}$

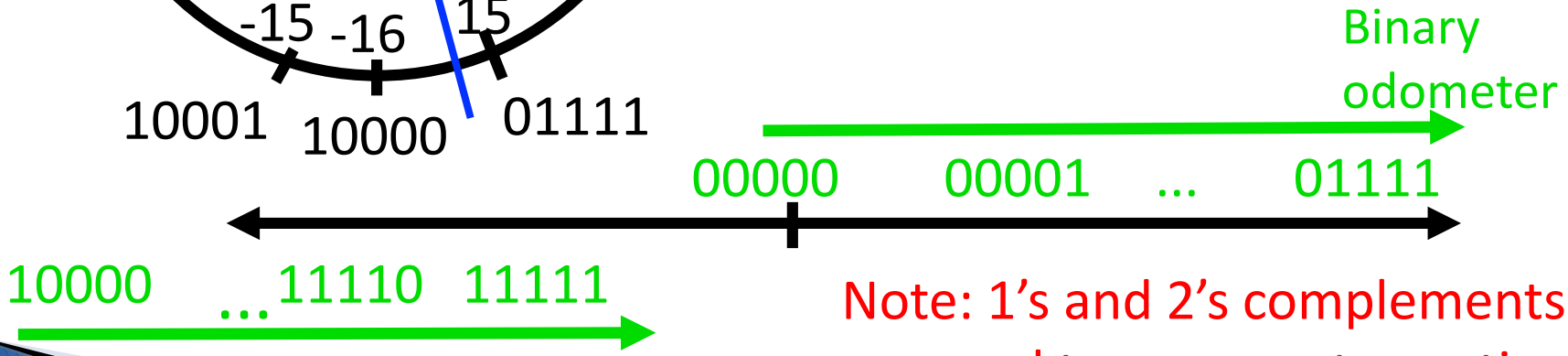
Example: -3 to +3 to -3:

x :	1101_{two} (-3)
x':	0010_{two}
+1:	0011_{two} (3)
()':	1100_{two}
+1:	1101_{two} (-3)

2's Complement Number "line": N = 5

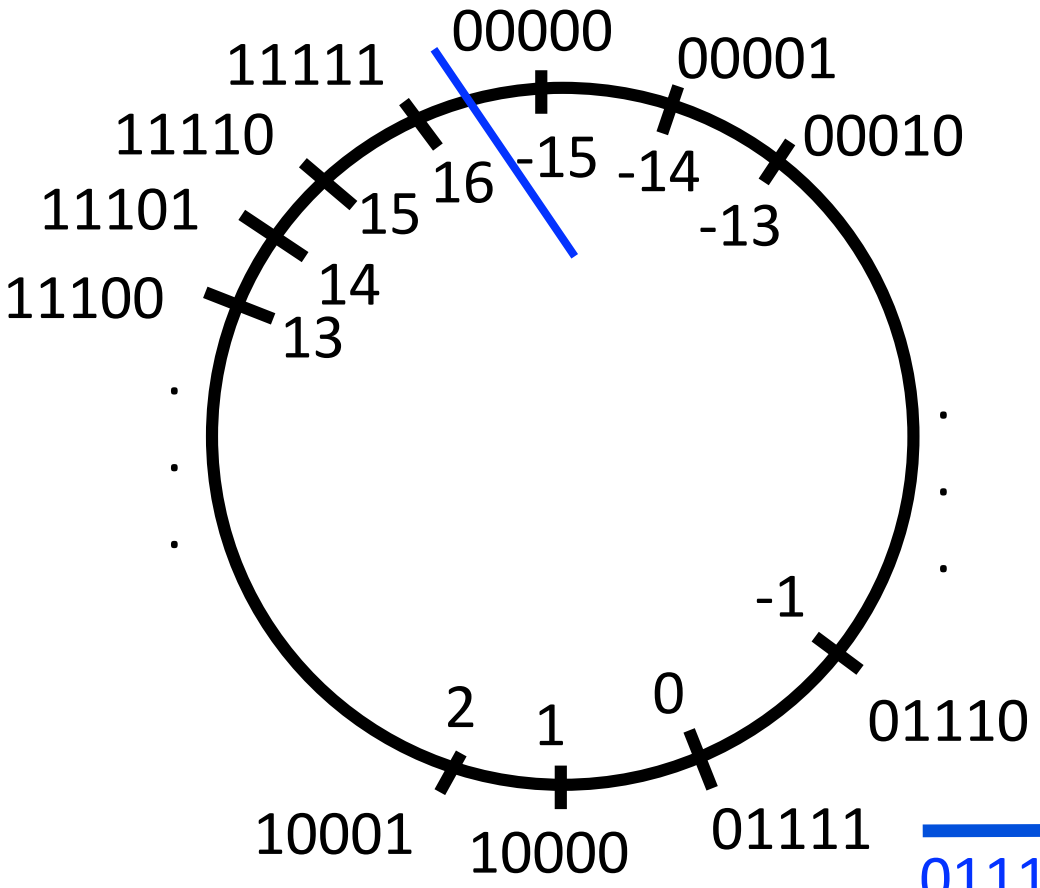


- 2^{N-1} non-negatives
- 2^{N-1} negatives
- one zero
- how many positives?
  - $2^{N-1} - 1$

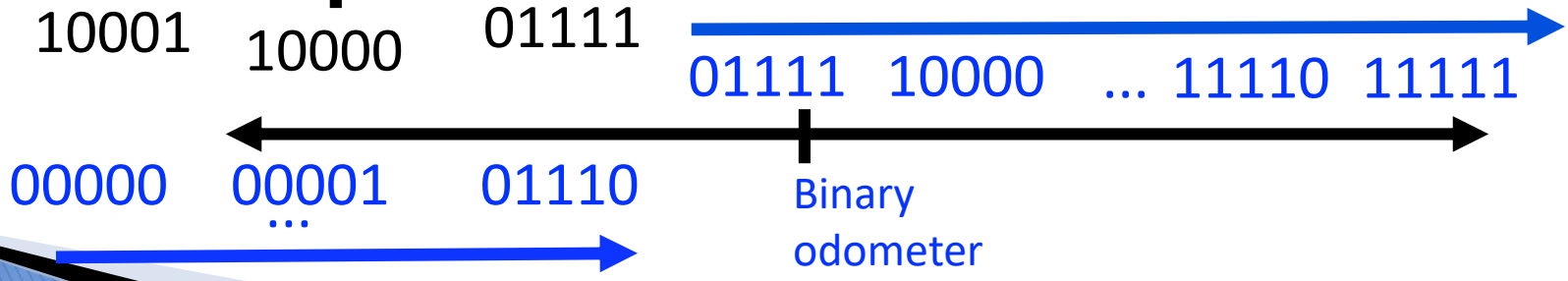


Note: 1's and 2's complements are used to represent negative numbers only!

Bias Encoding: N = 5 (bias = 15)



- Want 00… to represent the smallest number
- value = unsigned - bias
- Bias for N bits = $2^{N-1} - 1$
- one zero
- how many positives?
  - 2^{N-1}
  - (more than 2's complement)



Floating Point Numbers

- How best to represent: 2.75_{10}?
  - 2s Complement (but shift binary pt)
  - Bias (but shift binary pt)
  - Combination of 2 encodings
  - Combination of 3 encodings
  - We can't

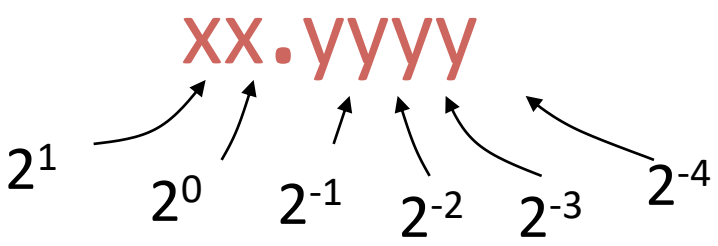
Shifting ***binary point*** means "divide" number by some power of 2.

$11_{10} = 1011.0_2$ ➔ $10.110_2 = (11/4)_{10} = 2.75_{10}$

Representation of Fractions

"Binary Point" like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:



$10.1010_2 = 1x2^1 + 1x2^{-1} + 1x2^{-3} = 2.625_{10}$

If we assume "**fixed binary point**", range of 6-bit representations with this format:

0 to 3.9375 (almost 4)

Fractional Powers of 2

i	2^{-i}	
0.	1.0	1
1.	0.5	1/2
2.	0.25	1/4
3.	0.125	1/8
4.	0.0625	1/16
5.	0.03125	1/32
6.	0.015625	
7.	0.0078125	
8.	0.00390625	
9.	0.001953125	
10.	0.0009765625	
11.	0.00048828125	
12.	0.000244140625	
13.	0.0001220703125	
14.	0.00006103515625	
15.	0.000030517578125	

History Lesson

- C developed by Dennis Ritchie at AT&T Bell Labs in the 1970s.
  - Used to maintain UNIX systems
  - C was derived from the B language
  - B was derived from the BCPL (Basic Combined Programming Language)
  - Many commercial applications are still written in C
- Current standard updates
  - C11: improved Unicode support, cross-platform multi-threading API
  - C99 or C9x remains the common standard

History Lesson

- References
  - `http://en.wikipedia.org/wiki/C99`
- Highlights
  - Declarations in for loops, like Java
  - Java-like `//` comments (to end of line)
  - Variable-length non-global arrays
  - `<inttypes.h>`: explicit integer types (intN_t, unintN_t)
  - `<stdbool.h>` for boolean logic def's

Disclaimer

- Important: You will not learn how to fully code in C in these lectures! You'll still need your C reference for this course:
  - K&R is a must-have reference
  - Check online for more sources

Compilation : Overview

C compilers take C and convert it into an architecture specific machine code (string of 1s and 0s).

- Unlike Java which converts to architecture independent bytecode.
- Unlike most functional programing languages (ex. Scheme) which interpret the code.
- These differ mainly in when your program is converted to machine instructions.
- For C, generally a 2 part process of compiling .c files to .o files, then linking the .o files into executables. Assembling is also done (but is hidden, i.e., done automatically, by default)

Compilation : Advantages

- Great run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- OK compilation time: enhancements in compilation procedure (`Makefiles`) allow only modified files to be recompiled

Compilation : Disadvantages

- All compiled files (including the executable) are architecture specific, depending on both the CPU type and the operating system
- Executable must be rebuilt on each new system.
  - Called "porting your code" to a new architecture.
- The "change→compile→run [repeat]" iteration cycle is slow

C vs. Java™ Overview (1/2)

Java	C
Object-oriented (OOP)	No built-in object abstraction. Data separate from methods.
“Methods”	“Functions”
Class libraries of data structures	C libraries are lower-level
Automatic memory management	Manual memory management

C vs. Java™ Overview (1/2)

Java	C
High memory overhead from class libraries	Low memory overhead
Relatively Slow	Relatively Fast
Arrays initialize to zero	Arrays initialize to garbage

Syntax:
`/* comment */`
`// comment`
`System.out.print`
	Syntax:
	`/* comment */`
	`// comment`
	`printf`

You need newer C compilers to allow Java style comments, or just use C99

C Syntax: `main`

- To get the main function to accept arguments, use this:
  `int main (int argc, char *argv[])`
- What does this mean?
  - **`argc`** will contain the number of strings on the command line (the executable counts as one, plus one for each argument). Here `argc` is 2:

    ```
    ./sort myFile
    ```

  - **`argv`** is a pointer to an array containing the arguments as strings (more on pointers later).
  - Always return a value according to ANSI (American National Standard Institute)

C Syntax: Variable Declarations

- Very similar to Java, but with a few minor but important differences
- All variable declarations must go before they are used (at the beginning of the block)*
- A variable may be initialized in its declaration; if not, it holds garbage!
- Examples of declarations:
  - correct:
```
int a = 0, b = 10;
...
```
  - Incorrect:*
```
for (int i = 0; i < 10; i++)
```

*C99 overcomes these limitations

C Syntax: True or False?

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (pointer: more on this later)
  - no such thing as a Boolean*
- What evaluates to TRUE in C?
  - everything else…

Boolean types provided by C99's `stdbool.h`

C syntax : flow control

- Within a function, remarkably close to Java constructs in methods (shows its legacy) in terms of flow control
  - `if-else`
  - `switch`
  - `while` and `for`
  - `do-while`

Common C Error

- There is a difference between assignment and equality

  `a = b` is assignment

  `a == b` is an equality test

- This is one of the most common errors for beginning programmers!
  - One solution (when comparing with constant) is to put the var on the right!
    If you happen to use =, it won't compile.

```
if (3 == a) { ... }
```

All objects have a size

- The size of their representation
- The size of static objects is given by sizeof operator (in Bytes)

```
#include <stdio.h>
int main() {
  char c = 'a';
  int  x = 34;
  int  y[4];
  printf("sizeof(c)=%d\n",   sizeof(c) );
  printf("sizeof(char)=%d\n",sizeof(char));
  printf("sizeof(x)=%d\n",   sizeof(x) );
  printf("sizeof(int)=%d\n", sizeof(int) );
  printf("sizeof(y)=%d\n",   sizeof(y) );
  printf("sizeof(7)=%d\n",   sizeof(7) );
}
```

Output:

```
sizeof(c)=1
sizeof(char)=1
sizeof(x)=4
sizeof(int)=4
sizeof(y)=16
sizeof(7)=4
```

Quiz:

```
void main(); {
   int *p, x=5, y; // init
   y = *(p = &x) + 1;
   int z;
   flip-sign(p);
   printf("x=%d,y=%d,p=%d\n",x,y,p);
}
flip-sign(int *n){*n = -(*n)}
```

How many syntax+logic errors in this C99 code?

#Errors
a) 1
b) 2
c) 3
d) 4
e) 5

Quiz: Answer

```
void main(); {
   int *p, x=5, y; // init
   y = *(p = &x) + 1;
   int z;
   flip-sign(p);
   printf("x=%d,y=%d,p=%d\n",x,y,*p);
}
flip-sign(int *n){*n = -(*n);}
```

How many syntax+logic errors in this C99 code?

5…

(signed ptr print is logical err)

#Errors
a) 1
b) 2
c) 3
d) 4
e) 5