

Bits, Bytes, and Integers

15-213/18-243: Introduction to Computer Systems

2nd Lecture, 13 January 2010

Instructors:

Bill Nace and Gregory Kesden

The course that gives CMU its “Zip”!

Last Time: Course Overview

■ Course Theme:

Abstraction Is Good But Don't Forget Reality

■ 5 Great Realities

- Ints are not Integers, Floats are not Reals
- You've Got to Know Assembly
- Memory Matters
- There's more to performance than asymptotic complexity
- Computers do more than execute programs

■ Administrative / Logistics details

Bits, Bytes, and Integers

■ Topics

- Representing information as bits
- Bit-level manipulations
 - Boolean algebra
 - Expressing in C
- Representations of Integers
 - Basic properties and operations
 - Implications for C

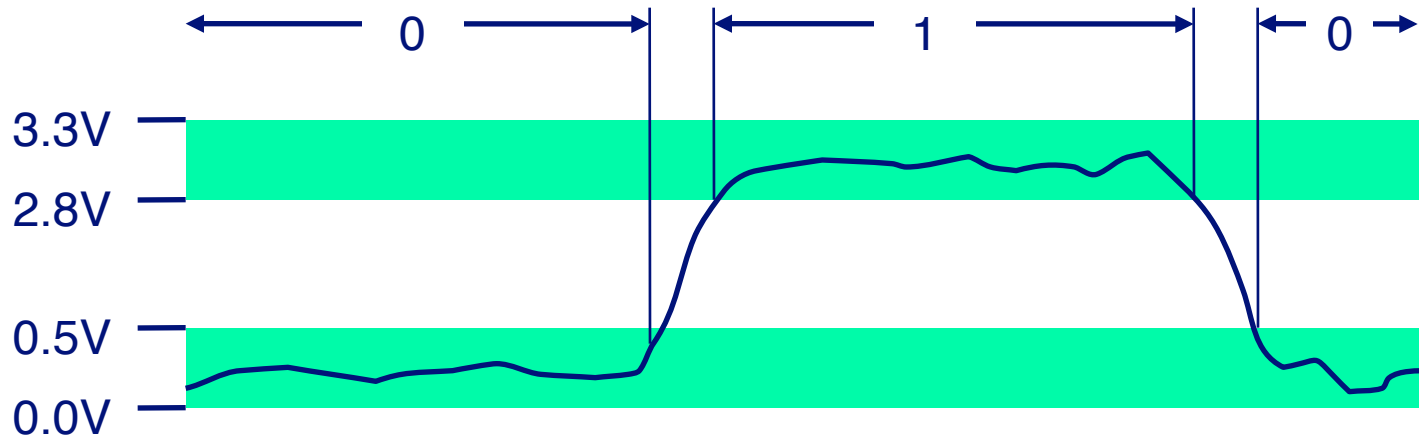
Binary Representations

■ Base 2 Number Representation

- Represent 15213_{10} as 11101101101101_2
- Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
- Represent 1.5213×10^4 (decimal) as $1.1101101101101_2 \times 2^{13}$

■ Electronic Implementation

- Easy to store with bistable elements
- Reliably transmitted on noisy and inaccurate wires



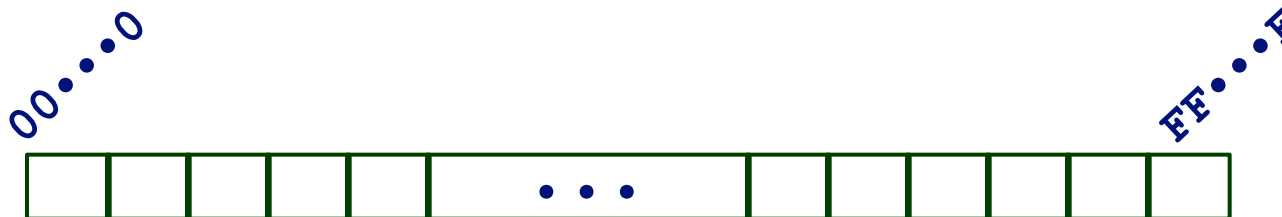
Encoding Byte Values

■ Byte = 8 bits

- Binary 00000000_2 to 11111111_2
- Decimal: 0_{10} to 255_{10}
 - First digit must not be 0 in C
- Hexadecimal 00_{16} to FF_{16}
 - Base 16 number representation
 - Use characters '0' to '9' and 'A' to 'F'
 - Write $FA1D37B_{16}$ in C as $0xFA1D37B$
 - Or $0xfa1d37b$
 - Or $0Xfa1d37b$ /* don't do this */

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Byte-Oriented Memory Organization



■ Programs Refer to Virtual Addresses

- Conceptually very large array of bytes
- Actually implemented with hierarchy of different memory types
- System provides address space private to particular “process”
 - Program being executed
 - Program can clobber its own data, but not that of others

■ Compiler + Run-Time System Control Allocation

- Where different program objects should be stored
- All allocation within single virtual address space

Machine Words

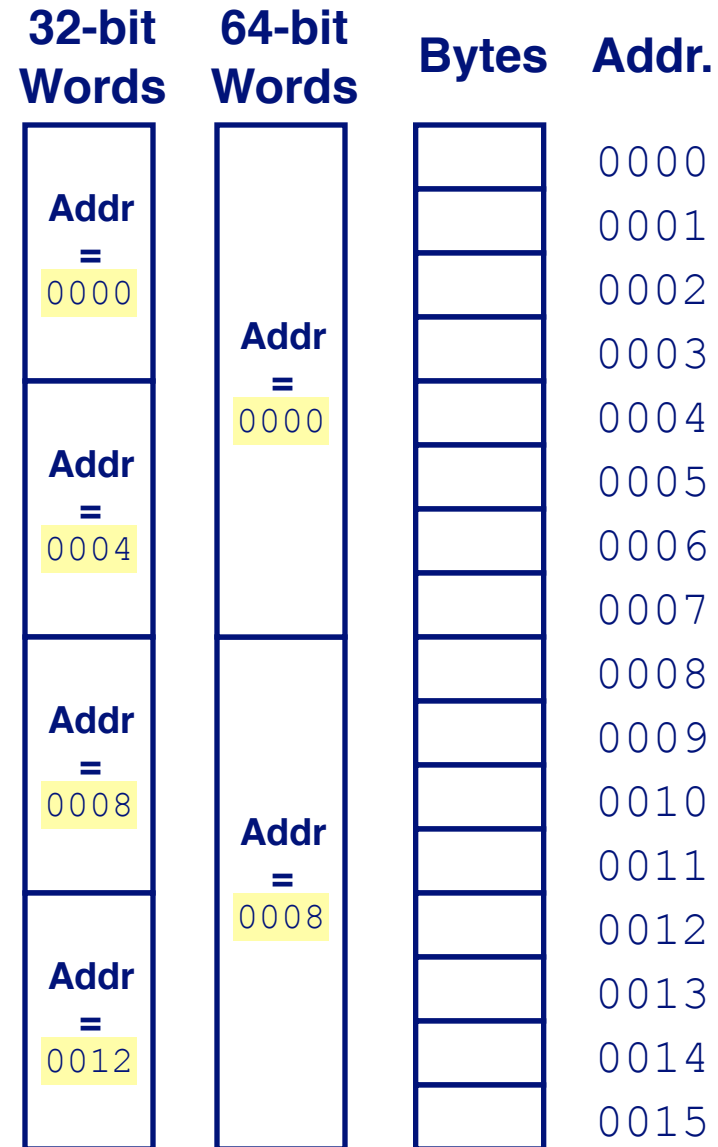
■ Machine Has “Word Size”

- Nominal size of integer-valued data
 - Including addresses
- Most current machines use 32 bits (4 bytes) words
 - Limits addresses to 4GB
 - Becoming too small for memory-intensive applications
- High-end systems use 64 bits (8 bytes) words
 - Potential address space $\approx 1.8 \times 10^{19}$ bytes
 - x86-64 machines support 48-bit addresses: 256 Terabytes
- Machines support multiple data formats
 - Fractions or multiples of word size
 - Always integral number of bytes

Word-Oriented Memory Organization

■ Addresses Specify Byte Locations

- Address of first byte in word
- Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)



Data Representations

■ Sizes of C Objects (in Bytes)

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8

Byte Ordering

- **How should bytes within a multi-byte word be ordered in memory?**
- **Conventions**
 - Big Endian: Sun, PPC Mac, Internet
 - Least significant byte has highest address
 - Little Endian: x86
 - Least significant byte has lowest address

Byte Ordering Example

■ Big Endian

- Least significant byte has highest address

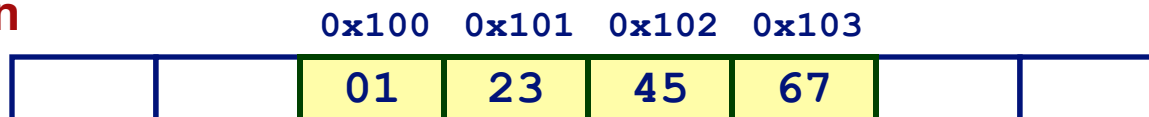
■ Little Endian

- Least significant byte has lowest address

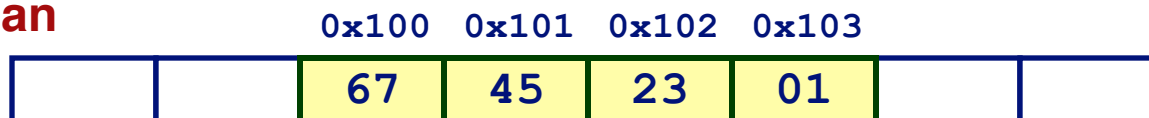
■ Example

- Variable x has 4-byte representation 0x01234567
- Address given by &x is 0x100

Big Endian



Little Endian



Reading Byte-Reversed Listings

■ Disassembly

- Text representation of binary machine code
- Generated by program that reads the machine code

■ Example Fragment

Address	Instruction Code	Assembly Rendition
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add \$0x12ab, %ebx
804836c:	83 bb 28 00 00 00 00	cmpl \$0x0, 0x28(%ebx)

■ Deciphering Numbers

- Value: 0x12ab
- Pad to 32 bits: 0x000012ab
- Split into bytes: 00 00 12 ab
- Reverse: ab 12 00 00

Examining Data Representations

■ Code to Print Byte Representation of Data

- Casting pointer to unsigned char * creates byte array

```
typedef unsigned char *pointer;

void show_bytes(pointer start, int len){
    int i;
    for (i = 0; i < len; i++)
        printf("0x%p\t0x%.2x\n", start+i, start[i]);
    printf("\n");
}
```

Printf directives:

%p: Print pointer

%x: Print Hexadecimal

show_bytes Execution Example

```
int a = 15213;  
printf("int a = 15213;\n");  
show_bytes((pointer) &a, sizeof(int));
```

Result (Linux):

```
int a = 15213;  
0x11ffffcb8 0x6d  
0x11ffffcb9 0x3b  
0x11ffffcba 0x00  
0x11ffffcbb 0x00
```

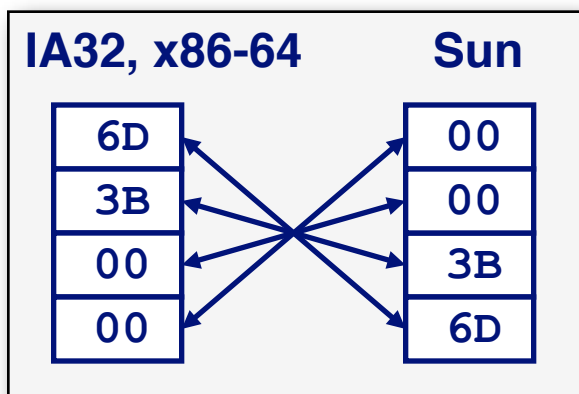
Representing Integers

Decimal: 15213

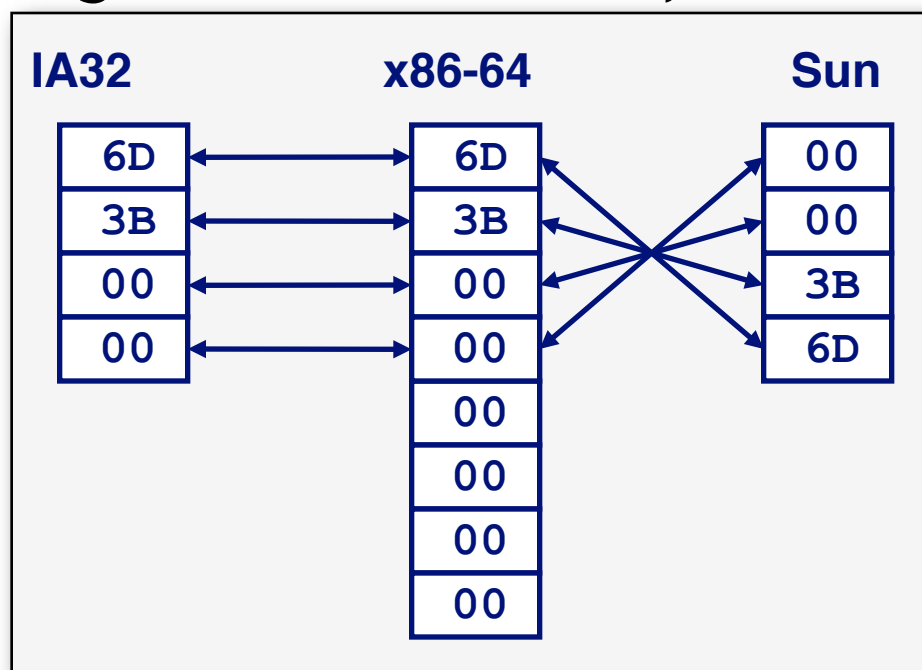
Binary: 0011 1011 0110 1101

Hex: 3 B 6 D

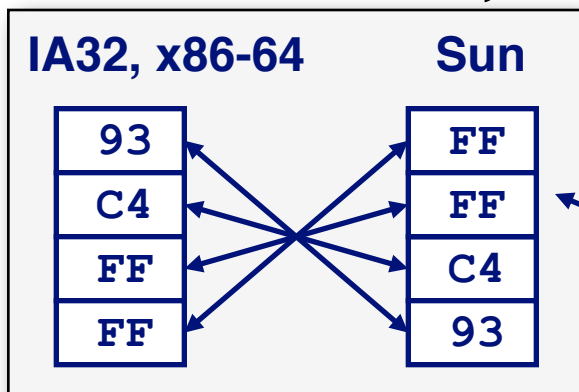
int A = 15213;



long int C = 15213;



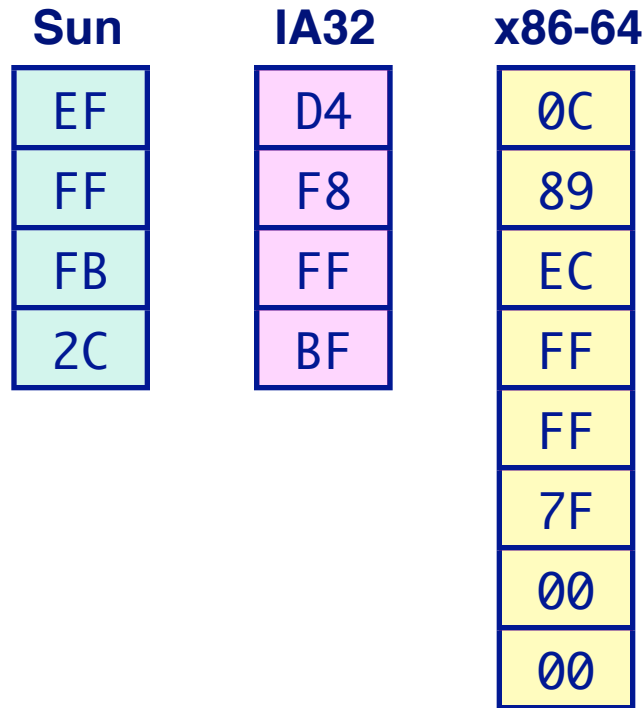
int B = -15213;



Two's complement representation
(Covered later)

Representing Pointers

```
int B = -15213;  
int *P = &B;
```



Different compilers & machines assign different locations to objects

Representing Strings

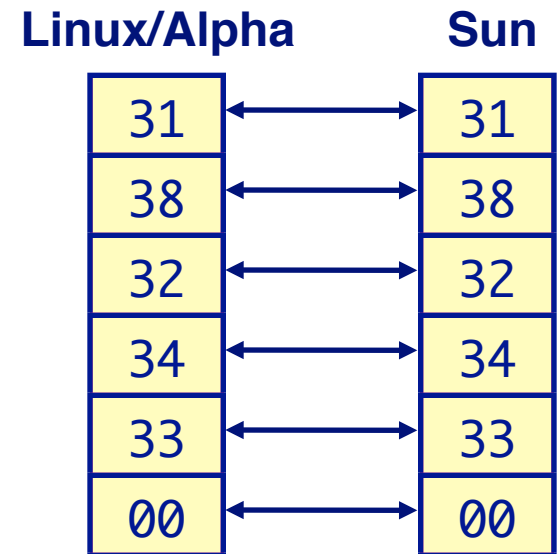
■ Strings in C

- Represented by array of characters
- Each character encoded in ASCII format
 - Standard 7-bit encoding of character set
 - Character "0" has code 0x30
 - Digit i has code $0x30+i$
- String should be null-terminated
 - Final character = 0

■ Compatibility

- Byte ordering not an issue

```
char S[6] = "18243";
```



Boolean Algebra

■ Developed by George Boole in 19th Century

- Algebraic representation of logic
 - Encode “True” as 1 and “False” as 0

And

- $A \& B = 1$ when both $A=1$ and $B=1$

$\&$	0	1
0	0	0
1	0	1

Or

- $A | B = 1$ when either $A=1$ or $B=1$

	0	1
0	0	1
1	1	1

Not

- $\sim A = 1$ when $A=0$

\sim	
0	1
1	0

Exclusive-Or (Xor)

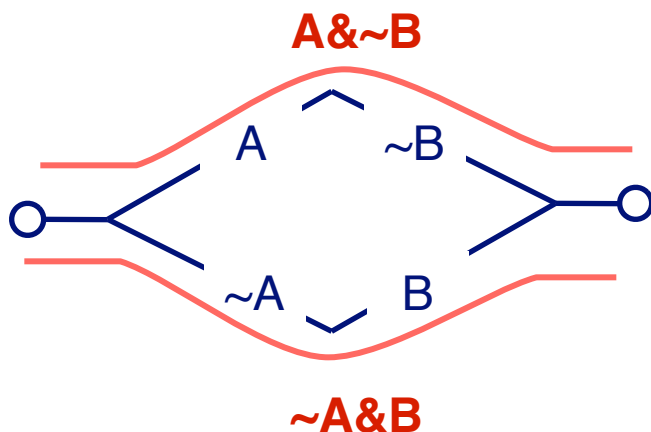
- $A \wedge B = 1$ when either $A=1$ or $B=1$, but not both

\wedge	0	1
0	0	1
1	1	0

Application of Boolean Algebra

■ Applied to Digital Systems by Claude Shannon

- 1937 MIT Master's Thesis
- Reason about networks of relay switches
 - Encode closed switch as 1, open switch as 0



Connection when

$$A \& \sim B \mid \sim A \& B$$

General Boolean Algebras

■ Operate on Bit Vectors

- Operations applied bitwise

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01010101

■ All of the Properties of Boolean Algebra Apply

Representing & Manipulating Sets

■ Representation

- Width w bit vector represents subsets of $\{0, \dots, w-1\}$
- $a_j = 1$ if $j \in A$

01101001 { 0, 3, 5, 6 }

76543210

01010101 { 0, 2, 4, 6 }

76543210

■ Operations

- & Intersection 01000001 { 0, 6 }
- | Union 01111101 { 0, 2, 3, 4, 5, 6 }
- ^ Symmetric difference 00111100 { 2, 3, 4, 5 }
- ~ Complement 10101010 { 1, 3, 5, 7 }

Bit-Level Operations in C

■ Operations $\&$, $|$, \sim , \wedge Available in C

- Apply to any “integral” data type
 - long, int, short, char, unsigned
- View arguments as bit vectors
- Arguments applied bit-wise

■ Examples (Char data type)

- $\sim 0x41 \rightarrow 0xBE$
 - $\sim 01000001_2 \rightarrow 10111110_2$
- $\sim 0x00 \rightarrow 0xFF$
 - $\sim 00000000_2 \rightarrow 11111111_2$
- $0x69 \ \& \ 0x55 \rightarrow 0x41$
 - $01101001_2 \ \& \ 01010101_2 \rightarrow 01000001_2$
- $0x69 \ | \ 0x55 \rightarrow 0x7D$
 - $01101001_2 \ | \ 01010101_2 \rightarrow 01111101_2$

Contrast: Logic Operations in C

■ Contrast to Logical Operators

- `&&`, `||`, `!`
 - View 0 as “False”
 - Anything nonzero as “True”
 - Always return 0 or 1
 - **Early termination**

■ Examples (char data type)

- `!0x41` → `0x00`
- `!0x00` → `0x01`
- `!!0x41` → `0x01`

- `0x69 && 0x55` → `0x01`
- `0x69 || 0x55` → `0x01`
- `p && *p` (avoids null pointer access)

Shift Operations

- **Left Shift:** $x \ll y$
 - Shift bit-vector x left y positions
 - Throw away extra bits on left
 - Fill with 0's on right
- **Right Shift:** $x \gg y$
 - Shift bit-vector x right y positions
 - Throw away extra bits on right
 - Logical shift
 - Fill with 0's on left
 - Arithmetic shift
 - Replicate most significant bit on right
- **Undefined Behavior**
 - Shift amount < 0 or \geq word size

Argument x	01100010
$\ll 3$	00010000
Log. $\gg 2$	00011000
Arith. $\gg 2$	00011000

Argument x	10100010
$\ll 3$	00010000
Log. $\gg 2$	00101000
Arith. $\gg 2$	11101000

Integer C Puzzles

- Assume 32-bit word size, two's complement integers
- For each of the following C expressions, either:
 - Argue that is true for all argument values
 - Give example where not true

- `x < 0` $\Rightarrow ((x*2) < 0)$
- `ux >= 0`
- `x & 7 == 7` $\Rightarrow (x \ll 30) < 0$
- `ux > -1`
- `x > y` $\Rightarrow -x < -y$
- `x * x >= 0`
- `x > 0 && y > 0` $\Rightarrow x + y > 0$
- `x >= 0` $\Rightarrow -x <= 0$
- `x <= 0` $\Rightarrow -x >= 0$
- `(x|-x)>>31 == -1`
- `ux >> 3 == ux/8`
- `x >> 3 == x/8`
- `x & (x-1) != 0`

Initialization

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

Encoding Integers

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \times 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \times 2^{w-1} + \sum_{i=0}^{w-2} x_i \times 2^i$$

```
short int x = 15213;
short int y = -15213;
```

← Sign Bit

- For 2 byte long short

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

■ Sign Bit

- For 2's complement, most significant bit indicates sign
 - 0 for nonnegative
 - 1 for negative

Encoding Example

short int x = 15213: 00111011 01101101

short int y = -15213: 11000100 10010011

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		15213		-15213