

Today: Bits, Bytes, and Integers

- Representing information as bits
- **Bit-level manipulations**
- **Integers**
 - Representation: unsigned and signed
 - Conversion, casting
 - Expanding, truncating
 - Addition, negation, multiplication, shifting
 - Summary
- Representations in memory, pointers, strings

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

General Boolean Algebras

■ Operate on Bit Vectors

- Operations applied bitwise

01101001	01101001	01101001	
<u>& 01010101</u>	<u> 01010101</u>	<u>^ 01010101</u>	<u>~ 01010101</u>
01000001	01111101	00111100	10101010

■ All of the Properties of Boolean Algebra Apply

Example: 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)

Contrast: Logic Operations in C

■ Contrast to Logical Operators

- `&&`, `||`, `!`
 - View 0 as “False”
 - Anything nonzero
 - Always returns 0 or 1
 - **Early** evaluation

■ Example

- `!0x41`
- `!0x00`
- `!!0x41`

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

**Watch out for `&&` vs. `&` (and `||` vs. `|`)...
one of the more common oopsies in
C programming**

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 left

■ 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