Bits, Bytes, and Integers

CSCI 237: Computer Organization
2nd Lecture, Feb 10, 2025

Jeannie Albrecht

Administrative Details

- TA hours will start this week
- Navigating the course webpage
- Lab on Wed/Thur
  - You need a CS account – should have received an email from Lida
  - Let me know if not!
- Please read through lab and prelab in advance!
  - Lab will be posted on webpage by tomorrow (at the latest)
- I'll talk a bit more about lab logistics on Wed/Thur
- Office hours this week: Wed 9-10:30
  - Regular schedule coming soon!

Last Time

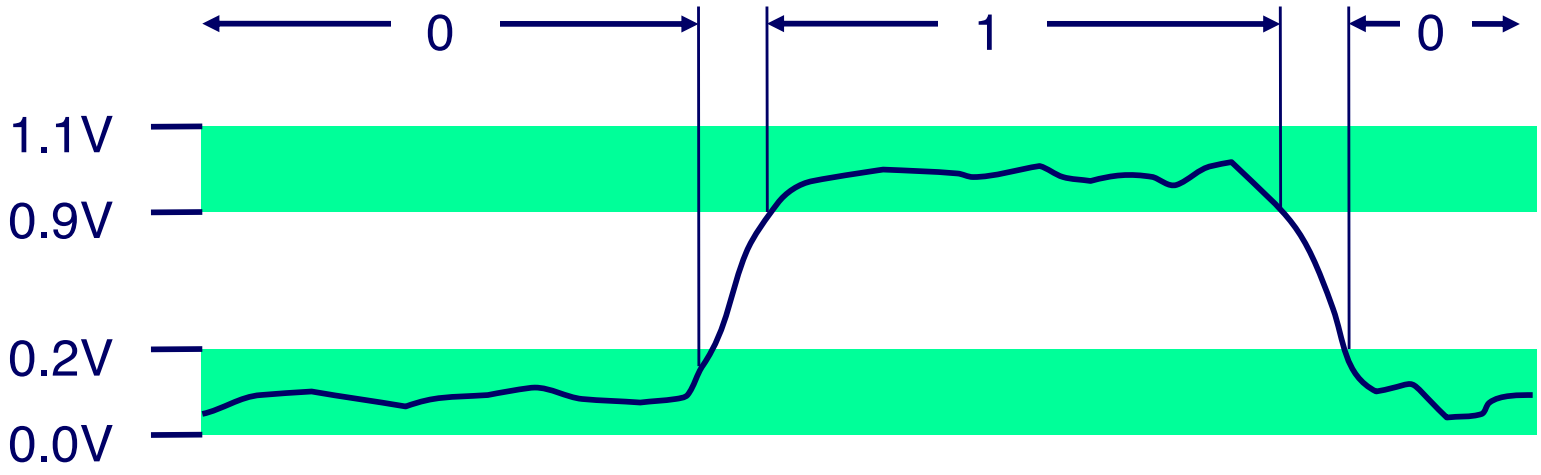
- Five realities
- Course logistics and overview
  - Any questions?

This week: Bits, Bytes, and Integers (Ch 2)

- 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

Everything is bits!

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- Why bits? Electronic implementation
  - Easy to store with *bistable* elements
  - Reliably transmitted on noisy and inaccurate wires



Counting in Binary

- Base 2 Number Representation
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.00110011001100110011[0011]\ldots_2$
  - Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

- How do we convert binary to decimal?
- Recall that decimal is base 10
  - $417_{10} \rightarrow 4 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$

- Binary is the same using base 2
  - $110_2 \rightarrow 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6_{10}$

Counting in Binary

- Base 2 Number Representation
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.00110011001100110011[0011]\ldots_2$
  - Represent $1.5213 \text{ X } 10^4$ as $1.1101101101101_2 \text{ X } 2^{13}$

- How do we convert binary to decimal?

$$11101101101101_2 =$$

$$1x2^{13} + 1x2^{12} + 1x2^{11} + 0x2^{10} + 1x2^9 + 1x2^8 + 0x2^7$$
$$+ 1x2^6 + 1x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 =$$

$$8192 + 4096 + 2048 + 0 + 512 + 256 + 0$$
$$+ 64 + 32 + 0 + 8 + 4 + 0 + 1 =$$

$$15213$$

Counting in Binary

- Base 2 Number Representation
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.0011001100110011[0011]\ldots_2$
  - Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

- How do we convert decimal to binary?

15213 / 2 = 7606	r1	59 / 2 = 29	r1
7606 / 2 = 3803	r0	29 / 2 = 14	r1
3803 / 2 = 1901	r1	14 / 2 = 7	r0
1901 / 2 = 950	r1	7 / 2 = 3	r1
950 / 2 = 475	r0	3 / 2 = 1	r1
475 / 2 = 237	r1	1 / 2 = 0	r1
237 / 2 = 118	r1		
118 / 2 = 59	r0		

Counting in Binary

- What about decimal values?
- 1.2_{10} ➔ $1.00110011001100110011001100110011001100110011001100110_2$

```
.2 × 2 = .4      r0
.4 × 2 = .8      r0
.8 × 2 = 1.6     r1
.6 × 2 = 1.2     r1

.2 × 2 = .4      r0
.4 × 2 = .8      r0
.8 × 2 = 1.6     r1
.6 × 2 = 1.2     r1

                 .
                 .
                 .
```

Counting in Binary

- 1.22_{10} → $1.0011100001010001111010111000010100011110 1_2$

.22 × 2 = .44	r0	.32 × 2 = .64	r0	.92 × 2 = 1.84	r1
.44 × 2 = .88	r0	.64 × 2 = 1.28	r1	.84 × 2 = 1.68	r1
.88 × 2 = 1.76	r1	.28 × 2 = .56	r0	.68 × 2 = 1.36	r1
.76 × 2 = 1.52	r1	.56 × 2 = 1.12	r1	.36 × 2 = .72	r0
.52 × 2 = 1.04	r1	.12 × 2 = .24	r0	.72 × 2 = 1.44	r1
.04 × 2 = .08	r0	.24 × 2 = .48	r0	.44 × 2 = .88	r0
.08 × 2 = .16	r0	.48 × 2 = .96	r0		
.16 × 2 = .32	r0	.96 × 2 = 1.92	r1		

Encoding Byte Values

- 1 Byte = 8 bits
  - Binary (base 2): 00000000_2 to 11111111_2
  - Decimal (base 10): 0_{10} to 255_{10}
  - Hexadecimal (base 16): 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
- Hexadecimal to decimal
  - $9BD_{16}$ ➔ $9 \times 16^2 + 11 \times 16^1 + 13 \times 16^0$

    $= 2493_{10}$

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

Encoding Byte Values

- 1 Byte = 8 bits
  - Binary (base 2): 00000000_2 to 11111111_2
  - Decimal (base 10): 0_{10} to 255_{10}
  - Hexadecimal (base 16): 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
- Binary to hexadecimal:
  - $15213_{10} = 11101101101101_2 = ??_{16}$

$$11\ 1011\ 0110\ 1101_2$$
$$0011\ 1011\ 0110\ 1101_2$$
$$3\quad B\quad 6\quad D$$
$$= 0x3B6D \text{ or } 3B6D_{16}$$

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

Example Data Representations

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

(number of bytes)

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 (Ch 2.1.6)

- 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

- ~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

General Boolean Algebras

- Operate on Bit Vectors
  - Operations applied bitwise

```
  01101001      01101001      01101001
& 01010101    | 01010101    ^ 01010101    ~ 01010101
  --------      --------      --------      --------
  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, …, w–1}
  - $a_j = 1$ if $j \in A$
    - 01101001 { 0, 3, 5, 6 } = A
    - *76543210*
    - 01010101 { 0, 2, 4, 6 } = B
    - *76543210*
- Operations

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

Bit-Level Operations in C (Lab 1!)

- Operations &, |, ~, ^ 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)
  - `~0x41 → 0xBE`
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - `~0x00 → 0xFF`
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - `0x69 & 0x55 → 0x41`
    - $01101001_2 \;\&\; 01010101_2 \rightarrow 01000001_2$
  - `0x69 | 0x55 → 0x7D`
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

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`

Contrast: Logic Operations in C

- Contrast to Logical Operators
  - `&&, ||, !`
    - View 0 as "Fals
    - Anything
    - Alway
    - Early t
- Examples
  - `!0x41`
  - `!0x00`
  - `!!0x41`
  - `0x69 && 0x55  →  0x01`
  - `0x69 || 0x55  →  0x01`

Watch out for && vs & (and || vs |)…

One of the more common mistakes in beginner C programming!

Shift Operations

- Left Shift: `x << y`
  - Shift bit-vector **x** left **y** positions
    - Throw away extra bits on left
    - Fill with 0's on right
- Right Shift: `x >> 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 ≥ *word size*

Argument x	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

Extra Practice

- Convert the following decimal number to binary
  - 27
- Convert the following binary number to decimal
  - 11001011
- Convert the following binary number to hexadecimal
  - 1011101
- Convert the following decimal number to hexadecimal
  - 93

Extra Practice

- Convert the following decimal number to binary
  - $27_{10} = 11011_2$
- Convert the following binary number to decimal
  - $11001011_2 = 203_{10}$
- Convert the following binary number to hexadecimal
  - $1011101_2 = 0x5D$
- Convert the following decimal number to hexadecimal
  - $93_{10} = 0x5D$