Bits and Bytes

CS-392-A Systems Programming

Instructors:

Georgios Portokalidis *(This section)*

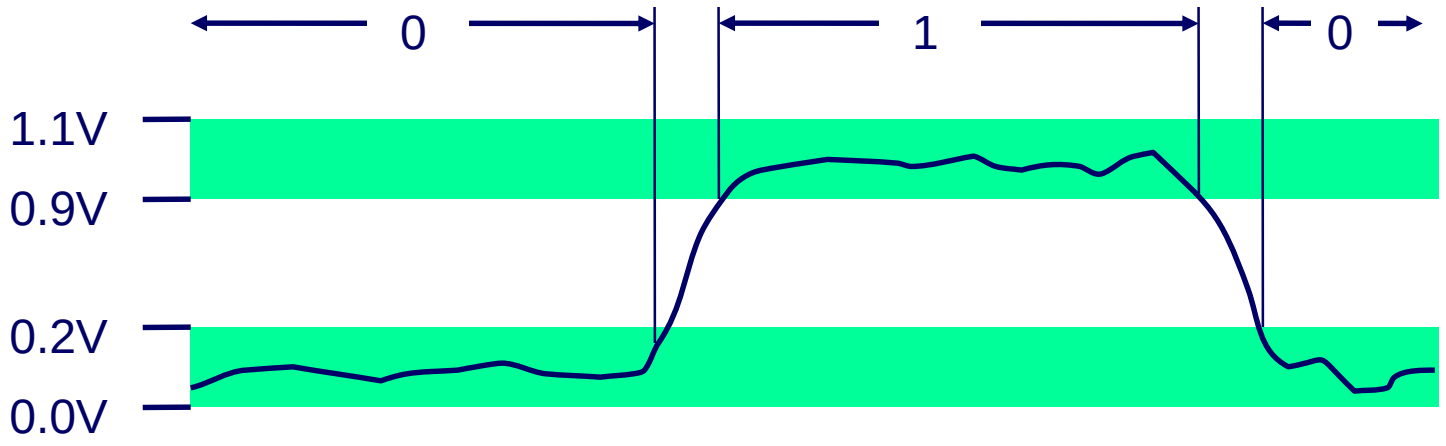
Srinivas Sridharan (Other section)

Bits and Bytes

- **Representing information as bits**
- **Bit-level manipulations**

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



Encoding Byte Values

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - 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
      - 0xfa1d37b

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

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - 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
      - 0xfa1d37b
  - Octal 0_8 to 377_8
    - Base 8 number representation '0'-'7'
    - Octal constants in C are prefixed with '0'
      - 010 is 8_{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

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

Bits and Bytes

- Representing information as bits
- **Bit-level manipulations**

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

- **~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 }
    - *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 &, |, ~, ^ 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
  - p && *p (avoids null pointer access)

Contrast: Logic Operations in C

- **Contrast to Logical Operators**
  - &&, ||, !
    - View 0 as "Fal
    - Anything nonze
    - Alway
    - Early
- **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 << 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