"With deliberate practice, however, the goal is not just to reach your potential but to build it, to make things possible that were not possible before. This requires challenging homeostasis - getting out of your comfort zone - and forcing your brain or your body to adapt. But once you do this, learning is no longer just a way of fulfilling some genetic destiny; it becomes a way of taking control of your destiny and shaping your potential in ways that you choose."

-- Peak: Secrets from the New Science of Expertise by Ericsson and Pool

1

Bits, Bytes, and Integers

CSCI 237: Computer Organization
4th Lecture, Friday, September 13

Kelly Shaw

2

Administrative Details

- Lab #1 due Tuesday at 11pm
  - Any questions?
  - Lab video on Glow – using gdb for bit puzzles
- Read CSAPP 2.2
- Weekly quiz on Glow due today at 2:30pm
- CS Colloquium today in Wege at 2:35pm
- Watch 15 minute recorded lecture before Monday

3

Last Class

- C syntax and examples
  - Control structures
- Terminology
- Representing information as bits
- Number representation
  - Conversion between decimal and binary
  - Conversion between hexadecimal, decimal, and binary
- Relationship between digits and range of unique values
- Bit-level manipulations

4

Today's Class

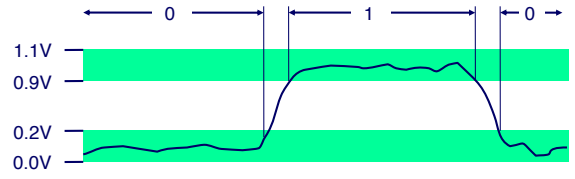
- Representing information as bits
- Number representation
  - Conversion between decimal and binary
  - Conversion between hexadecimal, decimal, and binary
- Relationship between digits and range of unique values
- Bit-level manipulations

5

5

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? Convenient for Electronic Implementation
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



6

6

We can represent (many) numbers 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 $1.1101101101101_2 \times 2^{13}$
- Representing really large numbers, really small numbers, or really precise numbers can take *many* binary digits
  - The largest number we can represent will be finite because of physical hardware constraints

7

7

Numbers are numbers but we can display them in different bases

Decimal	10	0, 1, …, 8, 9
Binary	2	0, 1
Octal	8	0, 1, 2, …, 6, 7
Hexadecimal	16	0, 1, 2, …, 8, 9, A, B, …, E, F

8

8

Converting from Binary to Decimal

- Decimal

$$\sum_{i=0}^{n-1} d_i \times 10^i \; where \; d_i \in \{0, 1, 2, 3, \ldots, 8, 9\}$$

e.g.) $123 = 1\times10^2 + 2\times10^1 + 3\times10^0$

- Binary

$$\sum_{i=0}^{n-1} b_i \times 2^i \; where \; b_i \in \{0, 1\}$$

e.g.) $110 = 1\times2^2 + 1\times2^1 + 0\times2^0$ = 6 in decimal

9

9

Converting Decimal to Binary

- Iterative process while remaining `number` != 0
  - Find largest power of 2 that fits into number and call it `x`.
  - Set to 1 the binary digit at position lg `x`
  - Subtract `x` from `number`

10

10

Example: Converting Decimal to Binary

- 11_{10}
  - Largest power of 2 that fits into 11 is 8
  - lg 8 is 3 so bit 3 will be set to 1
  - 11 – 8 = 3
  - Largest power of 2 that fits into 3 is 2
  - lg 2 is 1 so bit 1 will be set to 1
  - 3 – 2 = 1
  - Largest power of 2 that fits into 1 is 1
  - Lg 1 is 0 so bit 0 will be set to 1
  - 1 – 1 = 0
  - Final result: 1011_2

11

11

Converting Decimal to Binary (2nd algorithm)

- Iterative process while remaining `number` != 0
  - `bit = number % 2`
  - Prepend `bit` to result string
  - `number = number / 2`

12

12

Example: Converting Decimal to Binary (2nd algorithm)

- 11_{10}
  - `bit = 11 % 2 = 1`
  - `result = 1`
  - `number = 11 / 2 = 5`
  - `bit = 5 % 2 = 1`
  - `result = 11`
  - `number = 5 / 2 = 2`
  - `bit = 2 % 2 = 0`
  - `result = 011`
  - `number = 2 % 2 = 1`
  - `bit = 1 % 2 = 1`
  - `result = 1011`
  - `number = 1 / 2 = 0`

13

13

Hexadecimal

- Digits
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Converting hexadecimal to decimal

$$\sum_{i=0}^{n-1} h_i \times 16^i \; where \; b_i \in \{0, 1, \dots, 9, A, B, C, D, E, F\}$$

e.g.) $1B4 = 1\times16^2 + 11\times16^1 + 4\times16^0 = 256 + 176 + 4 = 436 \; in \; decimal$

14

14

Hexadecimal

- Convert decimal to hexadecimal using 2 algorithms for decimal to binary but switch to base 16 instead of base 2.

15

15

Today: Bits, Bytes, and Integers

- Representing information as bits
- Number representation
  - Conversion between decimal and binary
  - Conversion between hexadecimal, decimal, and binary
- Relationship between digits and range of unique values
- Bit-level manipulations

16

16

Bytes: the smallest unit of memory transfer

- 1 Byte = 8 bits
  - Binary: 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal 00_{16} to FF_{16}
    - Hex: 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

17

17

Bytes: the smallest unit of memory transfer

- Converting binary <-> hex:
  - One hex character represents 4 bits
    - Split into groups of 4 binary bits
    - "pad" leftmost digit with 0s
- Examples:
  - 11001_2 in hex?
  - 0xfa01 in binary?

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

18

18

Practice On Your Own (Solns on Glow)

- 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

19

19

Relationship between Digits and Unique Values

- What is the total number of unique values we can represent with **w** digits
  - in decimal?
  - in binary?
- Definition of $\log_n x$?
  - $\log_2 x$?
  - $\log_{10} x$?

20

20

C Primitive types are multiples of bytes

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
pointer	4	8	8

(number of bytes)

**word* size : size of pointer data which is number of bits to specify an address

21

21

Number of Unique values

- 1 byte char
  - 2^8 = 256
- 2 byte short
  - 2^16 = 65536
- 4 byte int
  - 2^32 = 4,294,967,296
- 8 byte long
  - 2^64 = 18446744073709551616

22

22

Practice on Your Own

- Why do you think there are so many different ways of representing whole numbers (i.e., char, short, int, long) in C?

23

23

Today: Bits, Bytes, and Integers

- Number representation
  - Conversion between decimal and binary
  - Conversion between hexadecimal, decimal, and binary
- Relationship between digits and range of unique values
- Bit-level manipulations

24

24

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

25

25

General Boolean Algebras

- Operate on sequence of bits
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

- All of the properties of Boolean Algebra apply, but on a per-bit basis

26

26

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 sequence of bits
  - 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`
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

28

28

Contrast: Logic Operations in C

- Contrast to Logical Operators
  - `&&`, `||`, `!`
    - View 0 as "False"
    - Anything nonzero as "True"
    - Operators always return 0 or 1
    - Early termination (short circuit)
- Examples (char data type)
  - `!0x41` → `0x00`
  - `!0x00` → `0x01`
  - `!!0x41` → `0x01`
  - `0x69 && 0x55` → `0x01`
  - `0x69 || 0x55` → `0x01`

29

29

Contrast: Logic Operations in C

- Contrast to Logical Operators
  - `&&, ||, !`
    - View 0 as "False"
    - Anything nonzero
    - Opera
    - 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!

30

30

Shift Operations

- Left Shift: `x << y`
  - Shift bit-vector **x** left by **y** positions
    - Throw away extra bits on left
    - Fill with 0's on right
- Right Shift: `x >> y`
  - Shift bit-vector **x** right by **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
  - If shift amount is less than 0 or larger than word size
    - Can't shift left more than or equal to number of bits in type

Argument **x**	01100010
<< 3	00010000
Log. >> 2	00011000
Arith. >> 2	00011000

Argument **x**	10100010
<< 3	00010000
Log. >> 2	00101000
Arith. >> 2	11101000

31

31

Practice On Your Own

- Assume numbers are represented as 8 bits:
- 0xa7 | 0xc2
- 0xa7 || 0xc2
- 0xa7 & 0xc2
- 0xa7 && 0xc2
- ~0xa7
- !0xa7
- 0xa7 >> 3 (logical right shift)
- 0xa7 >> 3 (arithmetic right shift)

32

32

In-class Problem

- Using only operators: ~ , !, &, |, <<, >>, +, -
- int moduloPowerOfTwo(int num, int powerOfTwo)
  - powerOfTwo is an int specifying which power of 2. It cannot be larger than 31.

33

33