Random News of the Day

Computer Program Recognizes Any Language

"If computers are rendered capable of recognising speech it will one day be the norm to give commands by voice rather than via a keyboard. 'Speaking' with a mobile phone is already commonplace for many people."

"A by-product is that this type of technology can be useful in contexts where several different languages are being used at once. It takes only in 30 to 60 seconds to identify a given spoken language. This can be helpful in instances where, for example, a person giving a presentation in one language cites a quote in another. It can also be significant in investigative work to determine quickly which language an individual is speaking."

http://www.sciencedaily.com/releases/2012/08/120821094125.htm

- Software Proposal

From Last Time

- Java has stack overflows: e.g., a bad recursive call
- LLDB: High-performance debugger
  - Multi-threaded programs, etc.
  - In “early development”
  - Mac OS X: C, Objective-C, C++

Bits

- 101
- 5 in binary
- 5 if it’s an integer
- $+\infty$ if float
- 0x101 = 257 in base 10 (decimal)
- A if it’s hex referring to ASCII

Bits + Context

- Integer
- Float
- Character
- Program instruction
- Memory Address

- Bytes
  - Trivia: *Nibble/Nybble*

Background: Hexadecimal

- 0xA4
  - 10100100 (binary)
  - 164 (decimal)

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

Take out some paper…

1. Convert 0x173A4C to binary
2. Convert 11 1100 1010 1101 1011 0011_2 to hex
3. Convert $7BB_{16}$ to base-10
4. Convert 60_{10} to base-16
5. Convert 600_{10} to base-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

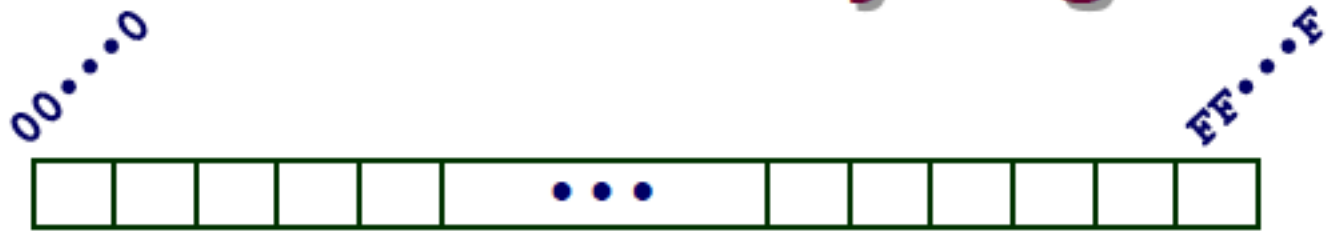
Take out some paper…

1. Convert 0x173A4C to binary
2. Convert 11 1100 1010 1101 1011 0011_2 to hex
3. Convert $7BB_{16}$ to base-10
4. Convert 60_{10} to base-16
5. Convert 600_{10} to base-16

Also, Google!

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

- Linux `ps` command?
- http://www.cheatengine.org/

Machine Words

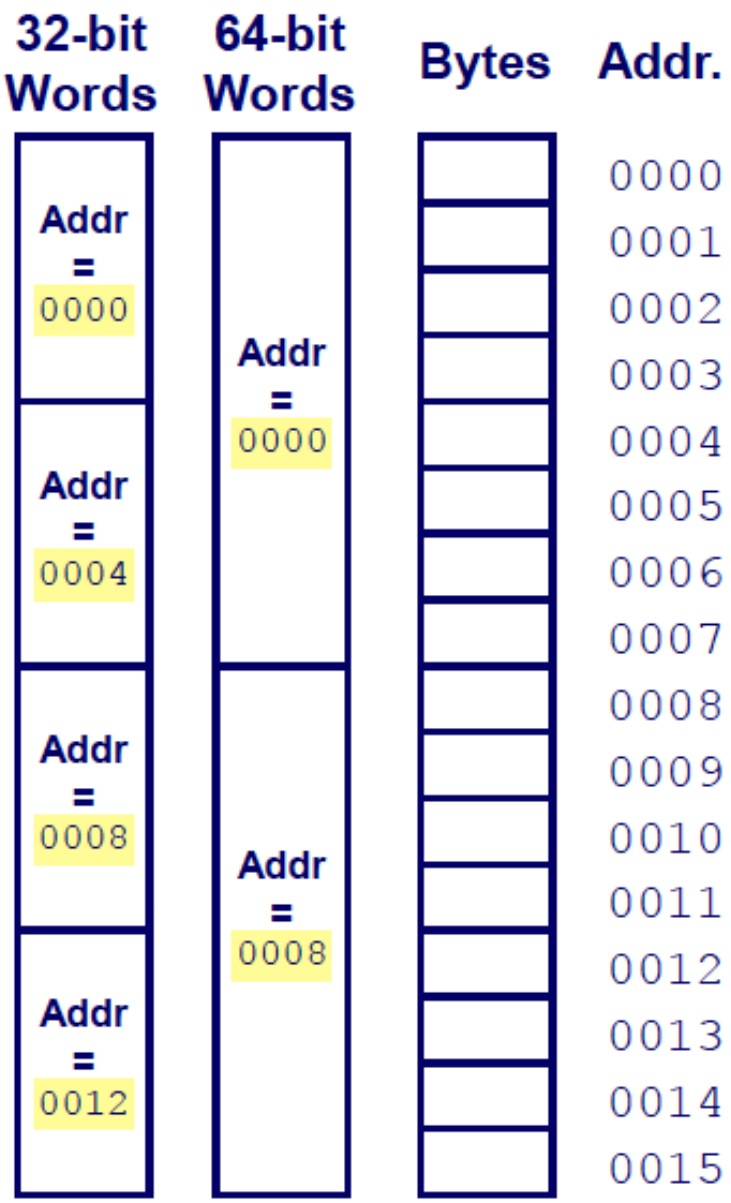
Machine Has "Word Size"

- Nominal size of integer-valued data
  - Including addresses
- Many machines use 32 bits (4 bytes) words
  - Limits addresses to 4GB
  - Becoming too small for memory-intensive applications
- Newer systems use 64 bits (8 bytes) words
  - Potential address space ≈ 1.8×10^{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 Sizes

C declaration	32-bit	64-bit
char	1	1
short int	2	2
int	4	4
long int	4	8
long long int	8	8
char*	4	8
float	4	4
double	8	8

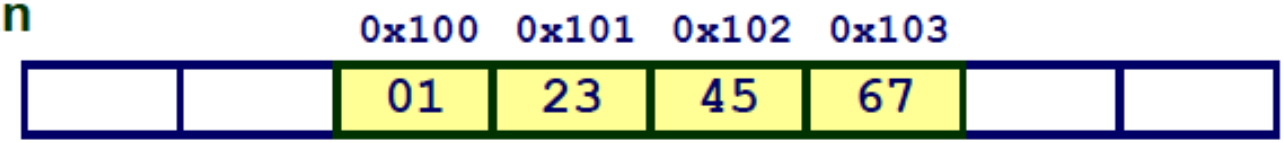
Byte Ordering

- Big Endian: Sun, PPC, ARM, PDP-10
  - Most significant byte has highest address
- Little Endian: x86, x86-64, VAX, PDP-11
  - Least significant byte has lowest address

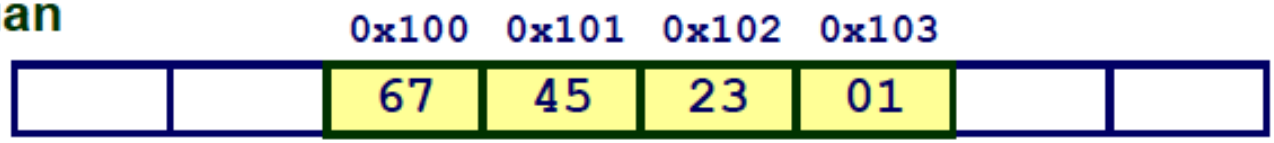
Example

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

Big Endian



Little Endian



Blame George Bool & Claude Shannon

- And
- Or
- Xor
- Not

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

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Bitwise_operators

Aside

- Swap without a temporary variable?

a = b

b = a

- Now a and b have same value, want to be swapped….

Aside: XOR Swap

```
void xorSwap (int *x, int *y) {
     if (x != y) {
          *x ^= *y;
          *y ^= *x;
          *x ^= *y;
     }
 }
```

x		y		
1010	$\oplus$	0011	=	1001 $\rightarrow x$
1001	$\oplus$	0011	=	1010 $\rightarrow y$
1001	$\oplus$	1010	=	0011 $\rightarrow x$
0011		1010		

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

- `~0x41 -->`
  $\sim 01000001_2$ -->
- `~0x00 -->`
  $\sim 00000000_2$ -->
- `0x69 & 0x55 -->`
  01101001_2 & 01010101_2 -->
- `0x69 | 0x55 -->`
  01101001_2 | 01010101_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

- `!0x41 --> 0x00`
- `!0x00 --> 0x01`
- `!!0x41 --> 0x01`

- `0x69 && 0x55 --> 0x01`
- `0x69 || 0x55 --> 0x01`

```
if (0)

if (1)



#if 0

#endif


#if VERBOSE > 2
  printf("…");
#endif

#define PI 3.14159
```

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

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

Right Shift: `x >> y`

- Shift bit-vector `x` right `y` positions
  - Throw away extra bits on right

Unsigned

Signed (typically)

Logical shift

- Fill with 0's on left

Arithmetic shift

- Replicate most significant bit on right

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

Undefined Behavior

- Shift amount < 0 or ≥ word size

Representing Integers

```
int A = 15213;
int B = -15213;
long int C = 15213;
```

Decimal:	15213
Binary:	0011 1011 0110 1101
Hex:	3 B 6 D