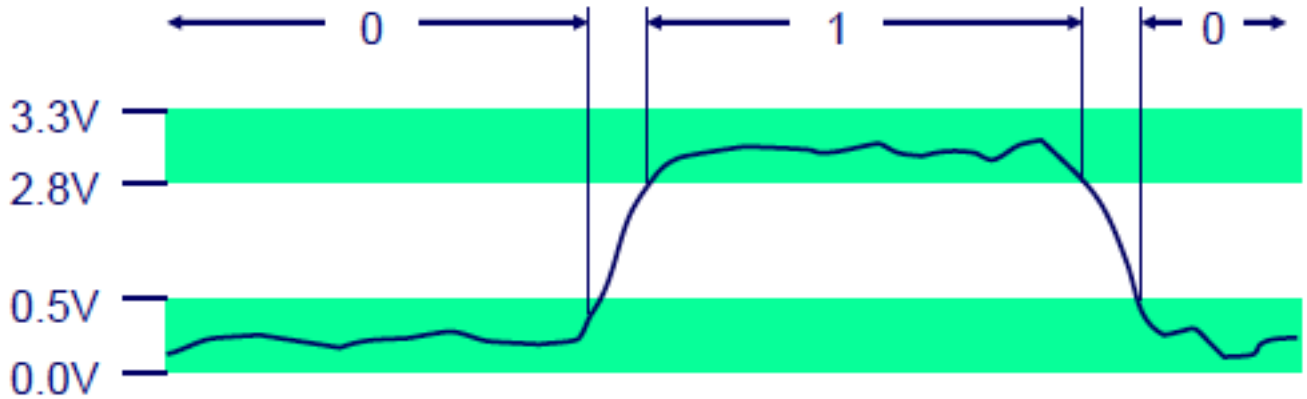
- Question: Big Endian

Bits

Electronic Implementation

- Easy to store with bistable elements
- Reliably transmitted on noisy and inaccurate wires



Bits

- 101
- 5 in binary
- 5 if it’s an integer
- $+\infty$ if float
- 0x101 = 65 in dec
- A if it’s hex referring to ASCII

Bits + Context

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

- *Bytes*

Background: Hexadecimal

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

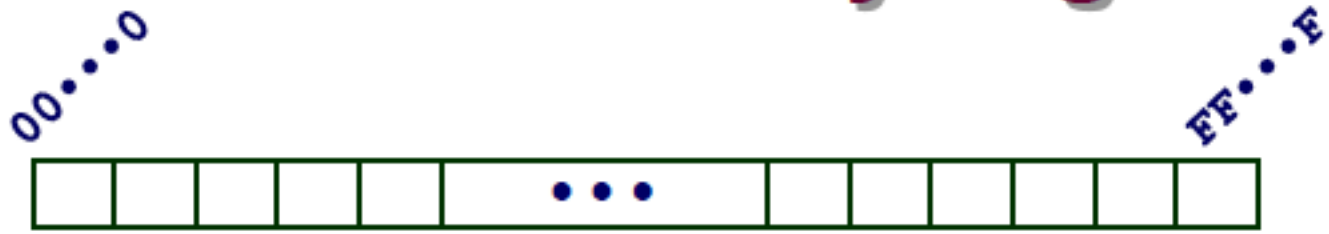
Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Take out some paper…

1. Convert 0x173A4C to binary
2. Convert 11 1100 1010 1101 1011 0011_2 to hex
3. Convert $7AF_{16}$ 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

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

Machine Words

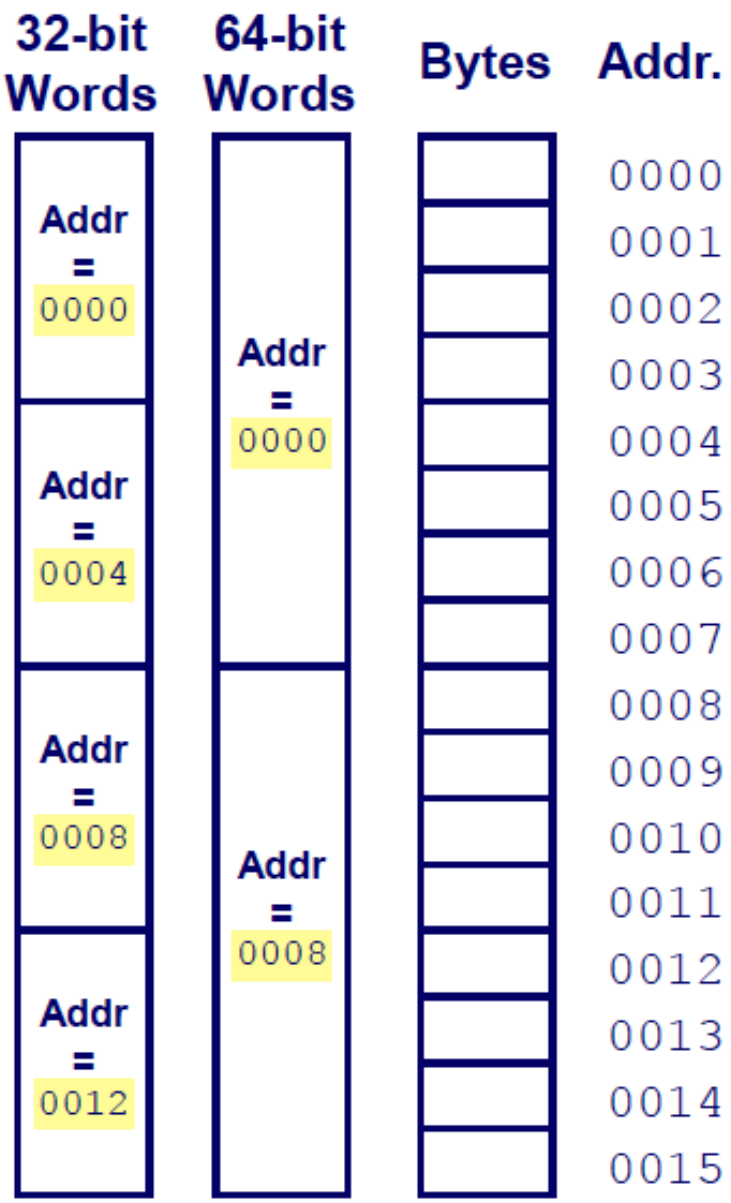
Machine Has "Word Size"

- Nominal size of integer-valued data
  - Including addresses
- Most current machines use 32 bits (4 bytes) words
  - Limits addresses to 4GB
  - Becoming too small for memory-intensive applications
- High-end 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

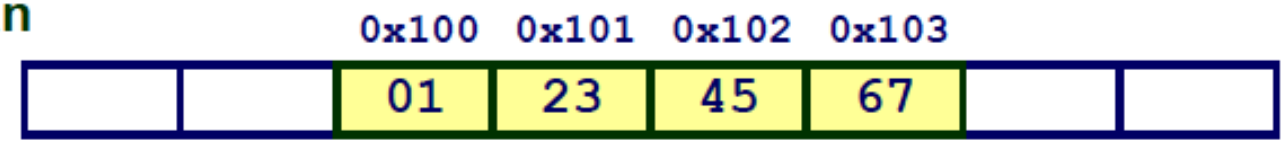
Byte Ordering

- Big Endian: Sun, PPC, Internet
  - Least significant byte has highest address
- Little Endian: x86
  - Least significant byte has lowest address

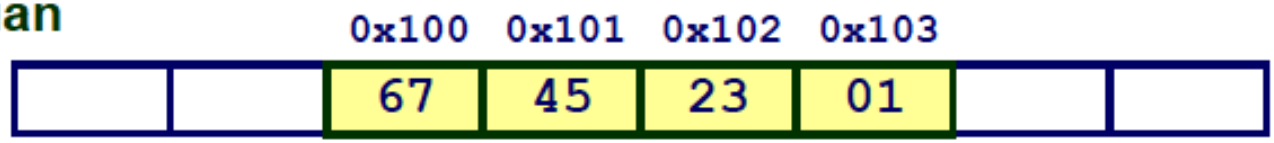
Example

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

Big Endian



Little Endian



Blame George Bool & Claude Shannon

- And
- Or
- Not
- Xor

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

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`