CSSE132
Introduction to Computer Systems

2 : Bits and bytes

March 5, 2013

Today: Bits and Bytes

- **How is Linux going?**
- **Information in bits**
  - bits and Bytes
  - Hexadecimal
  - printf conversions
- **Memory**
  - Words
  - Machine addressing
  - Data sizes
- **Two's complement**

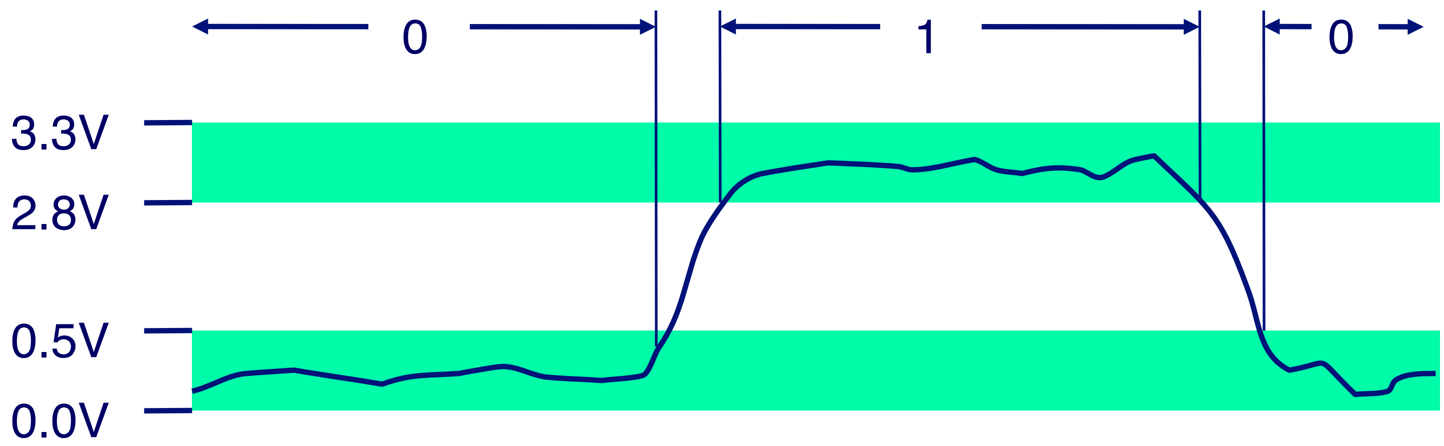
Binary and bits

- **Binary is a 2 digit numbering system (base 2)**
- **Decimal is a 10 digit numbering system (base 10)**
- **Hexadecimal is a 16 digit numbering system (base 16)**

- **Binary numbering is the basis for computing**
  - Easy to understand (switches on or off)
  - Represented in many domains
    - On/Off
    - 1/0
    - High voltage / low voltage
  - Less signal interpretation error
  - Simple physical representation

Binary Representations

- **Voltage representation**



Bits and Bytes

- **Bit : single binary number**
  - Either 1/0, On/Off, …
  - Not particularly useful by itself
  - Can be combined in series…
  - …with defined representation (encoding)

- **Byte : 8 bits**
  - Artifact of historical hardware design
  - Neither better nor worse than 7 bits or 9 bits
  - Just ‘happened’

Bytes

- **Have a bounded number of unique encodings**
  - 8 value places
  - 2 possible values for each place
- **Consider 1 bit**
  - 1 value place, 2 possible values
  - 2 unique encodings : 0, 1
- **Consider 2 bits**
  - 2 values places, 2 possible values
  - 4 unique encodings : 00, 01, 10, 11

Bytes

- **In general**
  - n value places, 2 possible values
  - 2^n possible unique encodings

- **For a single byte**
  - 8 value places, 2 possible values
  - 2^8 encodings (256)

Encoding numbers in binary

- **Similar to decimal, least-significant digit on the right**
  - 00_2 represents 0_{10}
  - 01_2 represents 1_{10}
  - 10_2 represents 2_{10}
  - 11_2 represents 3_{10}
  - and so on…

- **Convenient to represent place values as**

0	0	1	0	0	1	0	1	bit
128	64	32	16	8	4	2	1	Place value
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^n value

= 32 + 4 + 1 = 37

  - We will see another encoding/context by the end of this lecture

Byte representation practice

0	0	0	0	1	1	1	1
128	64	32	16	8	4	2	1

0	1	0	0	1	0	0	0
128	64	32	16	8	4	2	1

0	1	1	0	0	1	0	0
128	64	32	16	8	4	2	1

Encoding Byte Values as Hexadecimal

- **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 as Hexadecimal

- **Binary hex conversion**
  - Binary to hex
    - Partition bits into groups of 4
    - From least-sig side
    - Convert each group into hex digit
  - Hex to binary
    - Convert each hex digit to 4 bits
  - 2 hex digits represent 1 byte (8 bits)

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

Hexadecimal conversion

- **Base 16, so each place value is 16 times larger**
- **Multiply by place value to convert to decimal**

0	**1**	**0**	**1**
4096	256	16	1

$= 1*16^2 + 0*16 + 1*1 = 257$

0	**0**	**1**	**F**
4096	256	16	1

0	**0**	**2**	**A**
4096	256	16	1

$=2*16 + A*1 = 32 + 10 = 42$

0	**1**	**4**	**A**
4096	256	16	1

Hexadecimal conversion

Convert decimal to hex by repeated division (factoring)

523_{10}

523 = 32*16 + 11 : B

32 = 2*16 + 0 : 0

2 = 0*16 + 2 : 2

0	2	0	B
4096	256	16	1

$4,004_{10}$

4004 = 250*16 + 4 : 4

250 = 15*16 + 10 : A

15 = 0*16 + 15 : F

0	F	A	4
4096	256	16	1

printf() conversion

- **printf() can easily convert hexadecimal and decimal**
  - %d : signed decimal integer (also %i)
  - %u : unsigned decimal integer
  - %x : lowercase hexadecimal integer
  - %X : uppercase hexadecimal integer

Today: Bits, Bytes, and Integers

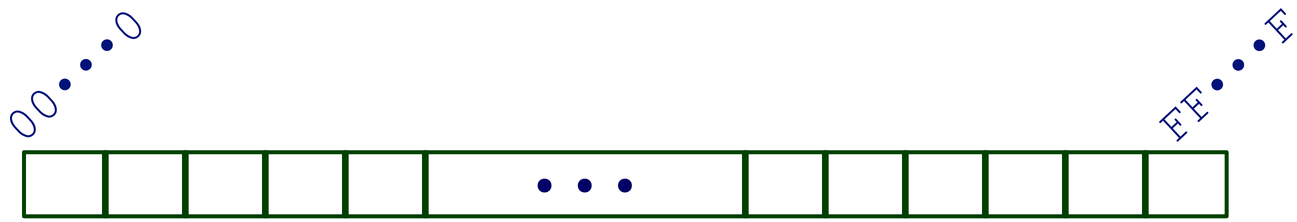
- **How is Linux going?**
- **Information in bits**
  - bits and Bytes
  - Hexadecimal
  - printf conversions
- **Memory**
  - Words
  - Machine addressing
  - Data sizes
- **Two's complement**

Memory

- **Storage bank for data**
  - Byte is the smallest unit of storage
  - Each byte has an 'address'
  - Addresses start at 0 and go up

- **Memory abstractions are hidden**
  - OS handles some memory abstractions (virtual address space)
  - Hardware handles other (caching hierarchy)

Byte-Oriented Memory Organization



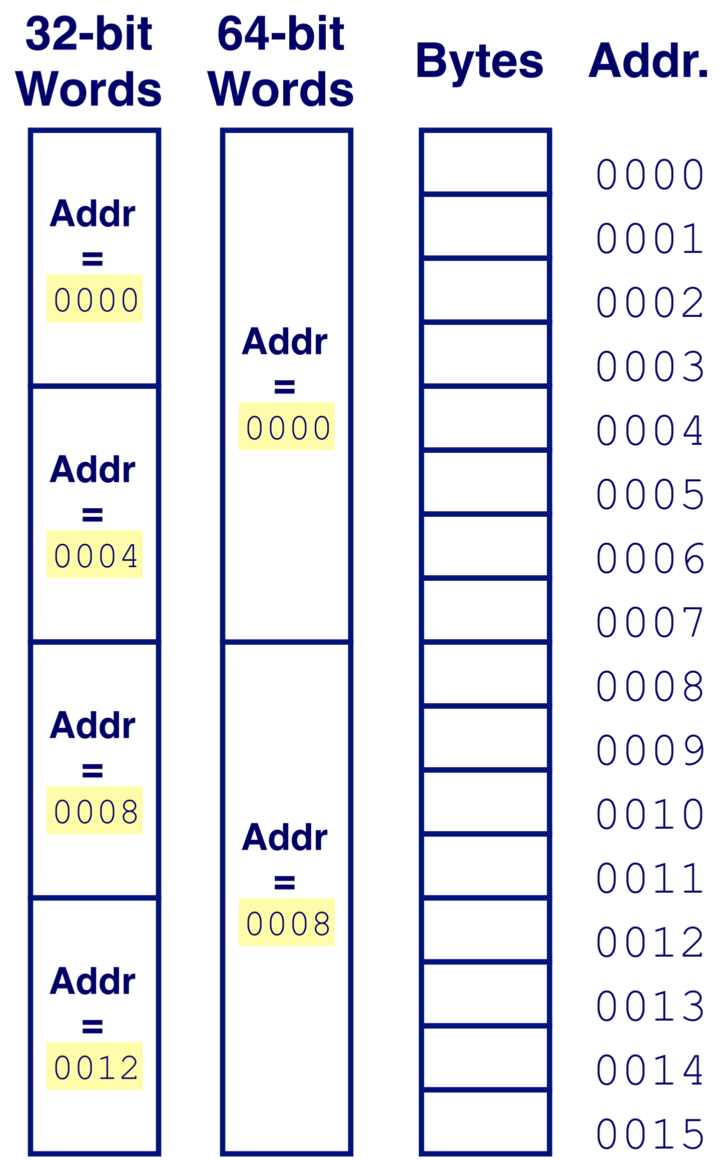
- **Programs Refer to Virtual Addresses**
  - Conceptually very large array of bytes
  - Actually implemented with hierarchy of different memory types
  - System provides address space private to particular "process"
    - Program being executed
    - Program can clobber its own data, but not that of 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 phones use 32 bits (4 bytes) words
    - Limits addresses to 4GB
    - Becoming too small for memory-intensive applications
  - Most current PCs 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 Representations (byte count)

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8

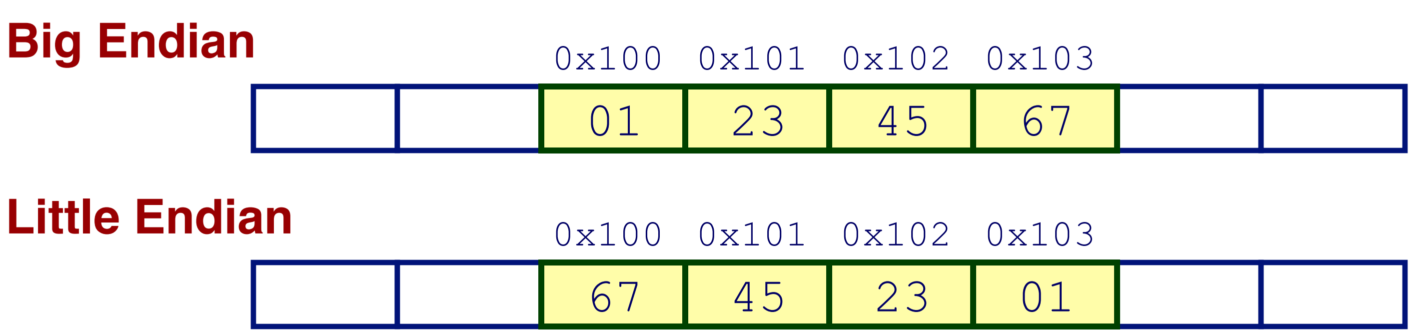
Byte Ordering

- **How should bytes within a multi-byte word be ordered in memory?**
- **Conventions**
  - Big Endian: Sun, PPC Mac, Internet
    - Least significant byte has highest address
  - Little Endian: x86, ARM phones
    - Least significant byte has lowest address
  - Bi-Endian: General ARM, general PPC, Itanium
    - Can switch between endianness

- Endianness is arbitrary!
  - No hardware reason that one is better!

Byte Ordering Example

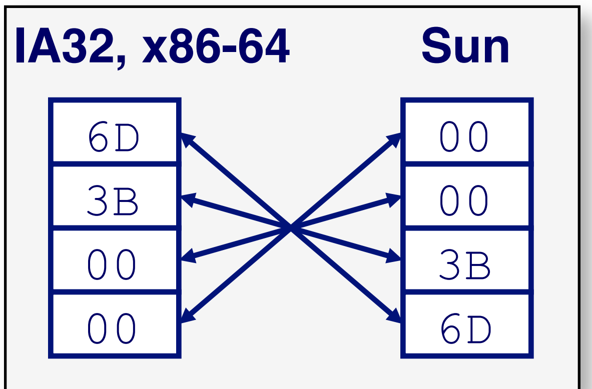
- **Big Endian**
  - Least significant byte has highest address
- **Little Endian**
  - Least significant byte has lowest address
- **Example**
  - Variable x has 4-byte representation 0x01234567
  - Address given by &x is 0x100



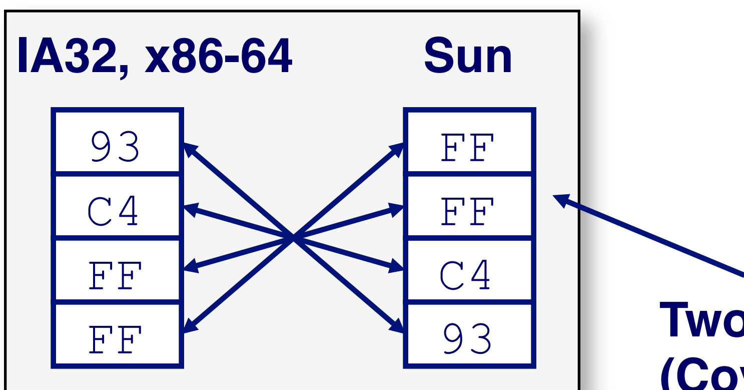
Representing Integers

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

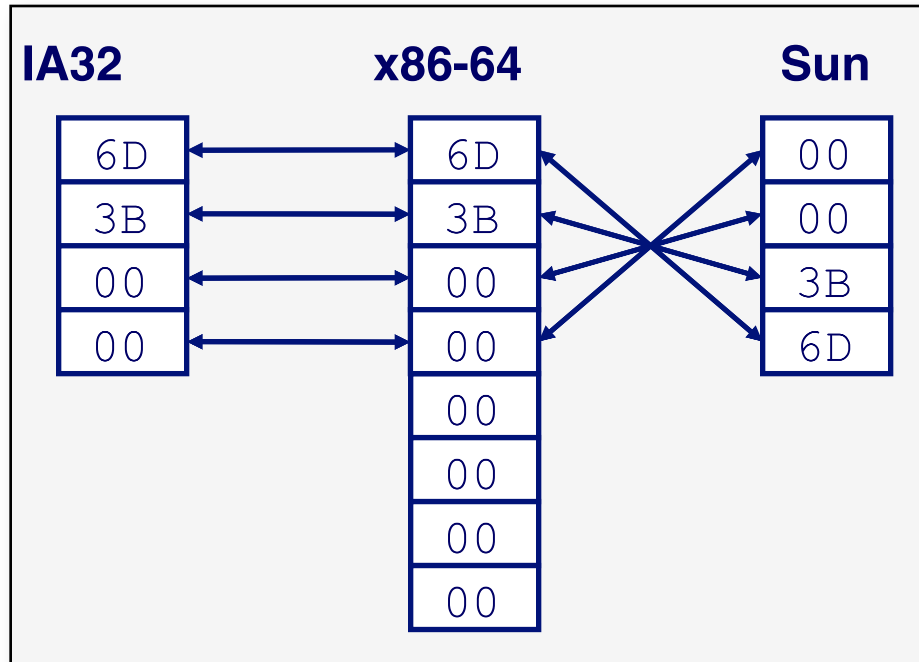
`int A = 15213;`



`int B = -15213;`



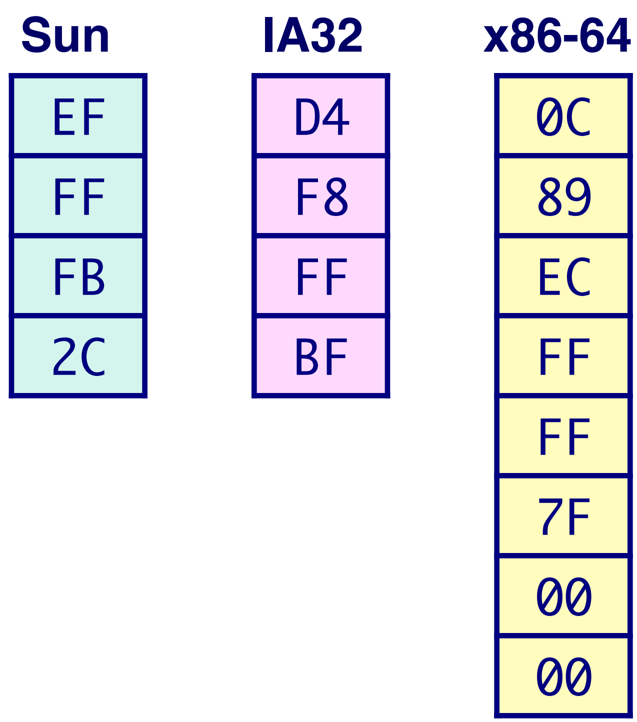
`long int C = 15213;`



Two's complement representation
(Covered later)

Representing Pointers

```
int B = -15213;
int *P = &B;
```

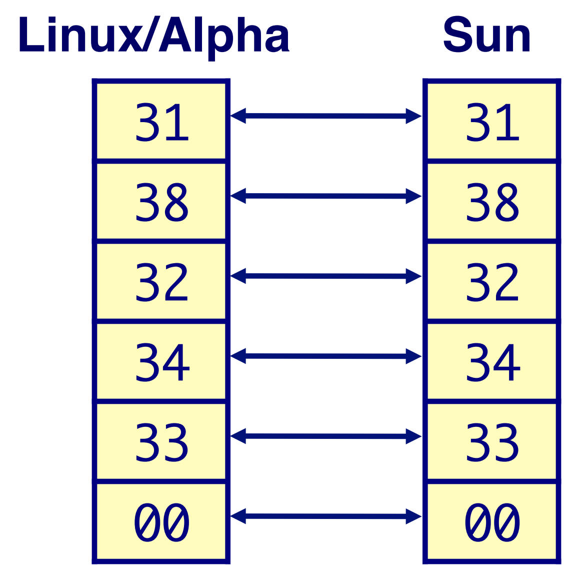


Different compilers & machines assign different locations to objects

Representing Strings

```
char S[6] = "18243";
```

- **Strings in C**
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Character “0” has code 0x30
      - Digit i has code 0x30+i
  - String should be null-terminated
    - Final character = 0
- **Compatibility**
  - Byte ordering not an issue



Today: Bits, Bytes, and Integers

- **How is Linux going?**
- **Information in bits**
  - bits and Bytes
  - Hexadecimal
  - printf conversions
- **Memory**
  - Words
  - Machine addressing
  - Data sizes
- **Two's complement**

Signed numbers preview

- **We will use 'Two's complement'**
  - Most significant bit represents negative value

- **So, for 4 bits**

1	**1**	**1**	**1**
-8	4	2	1

= -8 + 4 + 2 + 1 = -1

0	**1**	**1**	**1**
-8	4	2	1

= 4 + 2 + 1 = 7

Two's complement

- **Full bytes**

1	0	0	0	1	0	0	1
-128	64	32	16	8	4	2	1

-128 + 8 + 1 = -119

1	1	0	1	0	1	1	0
-128	64	32	16	8	4	2	1

-128 + 64 + 16 + 4 + 2 = -42

Encoding Example (Cont.)

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**