信息的存储

Information Storage

本章内容

Topic

- 使用比特表示信息

  Representing information as bits

- 位运算

  Bit-level manipulations

- 信息的存储和表示

  Information Storage and Representation

使用比特表示信息

Representing information as bits

万物皆比特
Everything is bits

- 二进制数据每一位只能用0或1来表示

  Each bit is 0 or 1

- 信息都可以使用二进制的编码进行表示

  Information can be encoded into sets of bits in various ways

  - 决定计算机要做什么（指令）

    Computers determine what to do (instructions)

  - 表示和处理各种数字、字符串等

    represent and manipulate numbers, strings, etc…

- 为什么要用二进制来表示？由计算机的电气实现所决定

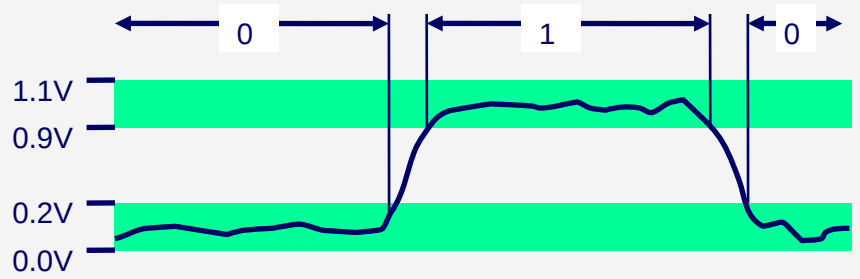
  Why bits? Electronic Implementation

  - 信号很容容易存储在双稳态单元中

    Easy to store with bistable elements

- 可以在存在噪声和不准确的信道中可靠地传输

  Reliably transmitted on noisy and inaccurate wires



大类必修课《数字逻辑与数字系统》

使用比特表示信息

Representing information as bits

举例：使用二进制计数
For example, can count in binary

- 二进制表示
  - 120_{10} 可以表示为 1111000_2
  - 1.20_{10} 可以表示为 $1.00110011001100110011[0011]_2$
  - 1.20×10^4 可以表示为 $1.0011[0011]_2 \times 2^{13}$

使用比特表示信息

Representing information as bits

字节数据编码
Encoding Byte Values

- 1 Byte = 8 bits
  - Binary （二进制）： 00000000_2 to 11111111_2
  - Decimal（十进制）: 0_{10} to 255_{10}
  - Hexadecimal（十六进制）： 00_{16} to FF_{16}
    - 以16位基数

      Base 16 number representation
    - 计数符号 0-9 A-F

      Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - C语言中 $FA1D37B_{16}$ 可表示为
      - 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

本章内容

Topic

- 使用比特表示信息

  Representing information as bits

- 位运算

  Bit-level manipulations

- 信息的存储和表示

  Information Storage and Representation

位运算

Bit-level manipulations

布尔代数
Boolean Algebra

- 由19世纪数学家乔治·布尔发明

  Developed by George Boole in 19th Century

- 逻辑的代数表示方法

  Algebraic representation of logic

  - “真”编码为1，“假”编码为0

    Encode “True” as 1 and “False” as 0

And （与）

A&B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or （或）

A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not （非）

~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor) （异或）

A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

相同为0， 不同为1

比特向量的布尔代数运算
Operate on Bit Vectors

- 逐位进行运算

  Operations applied bitwise

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

C语言中的位运算
Bit-Level Operations in C

- 四个运算符 (four operators defined in C)
  - | for Or （位或）
  - & for And （位与）
  - ~ for Not（位非）
  - ^ for eXclusive-Or （位异或）

C expression	Binary expression	Binary result	Hexadecimal result
~0x41	~[0100 0001]	[1011 1110]	0xBE
~0x00	~[0000 0000]	[1111 1111]	0xFF
0x69 & 0x55	[0110 1001] & [0101 0101]	[0100 0001]	0x41
0x69 | 0x55	[0110 1001] | [0101 0101]	[0111 1101]	0x7D

异或运算的应用：数据交换
XOR Operation Example：Swap Data

```
void funny(int *x, int *y)
{
    *x = *x ^ *y;     /* #1 */
    *y = *x ^ *y;     /* #2 */
    *x = *x ^ *y;     /* #3 */
}
```

Step	*x	*y
Begin	A	B
1	A^B	B
2	A^B	(A^B)^B = A^(B^B) = A^0 = A
3	(A^B)^A = (B^A)^A = B^(A^A) = B^0 = B	A
End	B	A

- 交换时不使用额外的变量

  Swap data without extra variables

C语言中的逻辑运算
Logical Operations in C

- 定义了三种逻辑运算

three logical operators defined in C

  - || （logical or, 逻辑或）
  - && （logical and, 逻辑与）
  - ！（logical not, 逻辑非）

短路效应

Short Circuit

1.X && 5/X 可以用于避免除0运算

will never cause a division by zero

2.p && *p++ 可以避免空指针运算

will never cause the dereferencing of a null pointer

3. 5 || X=Y 赋值语句将不会被执行

assignment statement will never be executed

位运算

Bit-level manipulations

C语言中的移位运算
Shift Operations in C

- 逻辑移位 和 算术移位
  Logical shift & arithmetic shift
  - 右移运算有逻辑移位（左侧补0）和算术移位（左侧补原最高位值）两种操作
- C语言标准中并没有详细地定义编译器具体应使用哪一种右移运算
  The C standards do not precisely define which type of right shift should be used by complier.
- 对于无符号数，右移一定是逻辑的
  For unsigned data, right shifts must be logical.
- 对于有符号数，理论上编译器采用逻辑右移和算术右移都是符合规范的
  For signed data (the default), either arithmetic or logical shifts may be used.
- 在实践中，几乎所有的编译器针对有符号数的右移都采用的是算术右移，大多数程序员也将这种情况视为默认
  In practice, however, almost all compiler/machine combinations use arithmetic right shifts for signed data, and many programmers assume this to be the case.

Operation	Values	
Argument x	[01100011]	[10010101]
x << 4	[0110*0000*]	[1010*0000*]
x >> 4 (logical)	[*0000*0110]	[*0000*1001]
x >> 4 (arithmetic)	[*0000*0110]	[*1111*1001]

位运算

Bit-level manipulations

```
int a = 1;
b = ++a+++a+++a;
b = ?
```

小知识：未定义行为
Tips: Undefined Behavior

- C语言规范中没有被明确定义的行为称为未定义行为

  Behaviors that are not explicitly defined in the C language specification are called undefined behaviors (UB)

- 编程时应避免使用未定义行为

  Should avoid undefined behavior

  - 有符号数算术右移除外

    Except for arithmetic right shift of signed numbers

- 例如：移位k，当k大于等于变量位长时

  Shifting by k, for large values of k ⩾ w

  (w: for data type consisting of w bit)

```
// GCC中的实现
int aval = 0x0EDCBA98 >> 36;
movl  $0, -8(%ebp)
unsigned uval = 0xFEDCBA98u << 40;
movl  $0, -4(%ebp)
```

位运算

Bit-level manipulations

要注意运算优先级问题
Operator precedence issues

- 1<<2 + 3<<4 = ?

X (1<<2) + (3<<4)

✓ 1<<(2 + 3)<<4

✓ (1 << (2+3)) << 4

本章内容

Topic

- 使用比特表示信息

  Representing information as bits

- 位运算

  Bit-level manipulations

- 信息的存储和表示

  Information Storage and Representation

信息的存储和表示

Information Storage and Representation

字长
Word Size

- 每台计算机都有一个字长的属性

  Any given computer has a “Word Size”

  - 指针数据的标称大小（虚拟地址宽度）

    Nominal size of a pointer data （Width of ritual Address）

  - 十多年前，大部分的计算机字长都是32位

    More than ten years ago, most machines used 32 bits (4 bytes) as word size

    - 限制了地址空间为4GiB（2^{32} 字节）

      Limits addresses to 4GiB (2^{32} bytes)

  - 64位字长目前已成为主流

    Until recently, machines have 64-bit word size

    - 寻址能力达到了18EiB

      Potentially, could have 18 EiB (exabytes) of addressable memory

1EiB = 1024PiB

1PiB = 1024TiB

C语言中的各数据类型位宽
Bit Width of Data Types in C Language

- 计算机和编译器支持多种数据类型

  Machines still support multiple data formats

- 或是小于字长，或大于字长

  Fractions or multiples of word size

- 但长度都是整数个字节

  Always integral number of bytes

C语言数据类型 C Data Type	典型32位系统 Typical 32-bit	典型64位系统 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

信息的存储和表示

Information Storage and Representation

字节序
Byte Ordering

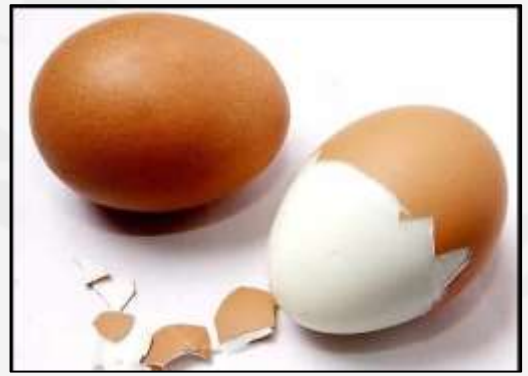
- 小端序（Little endian）: Intel
  - 低地址存放低位数据，高地址存放高位数据
- 大端序（Big endian）: IBM, Sun Microsystem（Oracle）
  - 低地址存放高位数据，高地址存放低位数据
- Example: 0x1234567

Big endian

	0x100	0x101	0x102	0x103	
...	01	23	45	67	...

Little endian

	0x100	0x101	0x102	0x103	
...	67	45	23	01	...



信息的存储和表示

Information Storage and Representation

探索数据在存储器中的存储方式
Explore Data Stored in Memory

- 这段代码用于打印各变量的字节表示形式

  Code to print the byte representation of program objects.

- show_bytes函数用于绕开C语言中的类型系统

  Function show_bytes uses casting to circumvent the type system.

- 其他的数据类型也可以使用类似的方法探索

  Similar functions are easily defined for other data types.

```
1   #include <stdio.h>
2
3   typedef unsigned char *byte_pointer;
4
5   void show_bytes(byte_pointer start, int len) {
6       int i;
7       for (i = 0; i < len; i++)
8           printf(" %.2x", start[i]);
9       printf("\n");
10  }
11
12  void show_int(int x) {
13      show_bytes((byte_pointer) &x, sizeof(int));
14  }
15
16  void show_float(float x) {
17      show_bytes((byte_pointer) &x, sizeof(float));
18  }
19
20  void show_pointer(void *x) {
21      show_bytes((byte_pointer) &x, sizeof(void *));
22  }
```

信息的存储和表示

Information Storage and Representation

Linux32/64（小端）、Win32（小端）和Sun（32位，大端）系统下的结果
Result on Linux32/64（LE）; Win32（LE）; Sun（32b，BE）

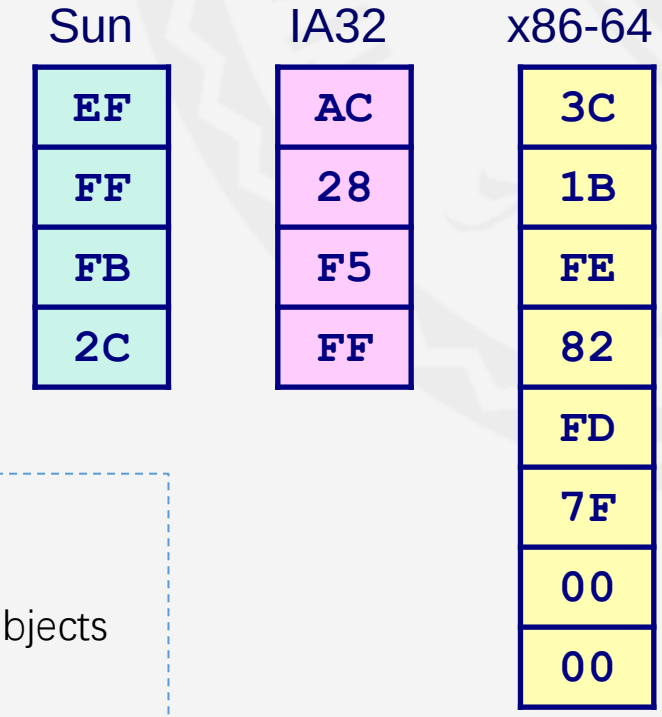
```
void test_show_bytes (int val)
{
    int ival = val;
    float fval = (float)ival;
    int *pval = &ival;

    show_int(ival);
    show_float(fval);
    show_pointer(pval);
}
```

Machine	Value	Type	Bytes (hex)
Linux 32	12,345	int	39 30 00 00
Windows	12,345	int	39 30 00 00
Sun	12,345	int	00 00 30 39
Linux 64	12,345	int	39 30 00 00
Linux 32	12,345.0	float	00 e4 40 46
Windows	12,345.0	float	00 e4 40 46
Sun	12,345.0	float	46 40 e4 00
Linux 64	12,345.0	float	00 e4 40 46
Linux 32	&ival	int *	e4 f9 ff bf
Windows	&ival	int *	b4 cc 22 00
Sun	&ival	int *	ef ff fa 0c
Linux 64	&ival	int *	b8 11 e5 ff ff 7f 00 00

指针的存储方法
Representing Pointers

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



- 注意：不同的编译器和计算机可能会分配不同的地址

  Different compilers & machines assign different locations to objects

- 甚至每一次运行时得到的结果都不相同

  Even get different results each time run program

字符串的表示
Representing Strings

- C语言的字符串使用char数组表示

  Strings in C are represented by array of characters
  - 每个字符都被编码成ASCII码

    Each character encoded in ASCII format
  - 一个7比特的字符编码集 （扩展集为8比特）

    Standard 7-bit encoding of character set
  - 字符“0” 的编码是0x30

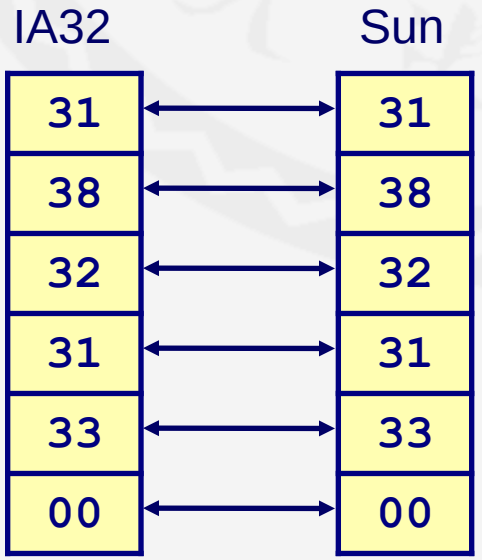
    Character “0” has code 0x30
  - 数字字符 i 的编码是 0x30 +i

    Digit i has code 0x30+i
  - 字符串的结尾应为空字符，即ASCII编码为0

    String should be null-terminated, Final character = 0
- 字符串的表示与字节序无关，大小端兼容

  Compatibility，byte ordering not an issue

```
char S[6] = "18213";
```



信息的存储和表示

Information Storage and Representation

程序的表示
Representing Code

- 不同类型的机器使用不同的且不兼容的指令和指令编码

Different machine types use different and incompatible instructions and encodings.

- 在相同处理器不同的操作系统中，由于编码规范存在差异，同样代码所生成的程序也不是二进制兼容的

Even identical processors running different operating systems have differences in their coding conventions and hence are not binary compatible.

- 程序很少能够在不同类型机器和不同操作系统中实现二进制水平上移植

Binary code is seldom portable across different combinations of machine and operating system.

```
int sum(int x, int y) {
    return x + y;
}
```

Linux 32: 55 89 e5 8b 45 0c 03 45 08 c9 c3
Windows: 55 89 e5 8b 45 0c 03 45 08 5d c3
Sun: 81 c3 e0 08 90 02 00 09
Linux 64: 55 48 89 e5 89 7d fc 89 75 f8 03 45 fc c9 c3

小知识：PE和ELF格式

- Windows操作系统下常用的可执行文件格式是PE:
  - Portable Executable (PE)
- Unix家族（含Linux）操作系统下可执行文件格式为ELF
  - Executable and Linkable Format (ELF)

Portable Executable

Filename extension	.acm, .ax, .cpl, .dll, .drv, .efi, .exe, .mui, .ocx, .scr, .sys, .tsp
Internet media type	application/vnd.microsoft.portable-executable
Developed by	Currently: Microsoft
Type of format	Binary, executable, object, shared libraries
Extended from	DOS MZ executable COFF

ELF

Filename extension	none, .axf, .bin, .elf, .o, .prx, .puff and .so
Magic number	0x7F 'E' 'L' 'F'
Developed by	Unix System Laboratories[1]:3
Type of format	Binary, executable, object, shared libraries, core dump
Container for	Many executable binary formats