Number Bases and C's Bitwise Operators

This lecture reviews conversion between decimal, binary and hexadecimal numbers, describes C's bitwise logical operators and shows how binary numbers are represented by voltage levels.
After this lecture you should be able to: convert a decimal value to and from binary, convert a binary value to and from hexadecimal, evaluate expressions that use C's bitwise logical and shift operators, and convert between bus signal levels and a numeric value.

Number Systems

We commonly use decimal notation to express the value of a number. However, in digital logic hardware (including computers) numbers are represented with two-valued (binary) values. To be able to deal with the hardware representation of numbers we need to be able to convert between the decimal and binary representation of numbers. Note that *the value of a number does not change when we express it in a different base*.

exponent	value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

Binary Representation

In the decimal number system each digit position represents a different power of ten. The rightmost digit gives the number of multiples of 10^0 in the number, the next digit gives the multiples of 10^1, and so on. For example, $105 = 5 \times 10^0 + 0 \times 10^1 + 1 \times 10^2$.

Binary numbers work in the same way, but since only two values are possible for each digit, each position represents a different power of two. The rightmost digit gives the number of multiples of 2^0, the next digit gives the multiples of 2^1, and so on. For example, $110 = 0 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = 0 \times 1 + 1 \times 2 + 1 \times 4 = 6_{\text{(base 10)}}$.

The table below shows the decimal values of powers of 2 for exponents from 0 to 15:

To convert from binary to decimal we just need to add up the powers of two corresponding to the bit positions that are '1's.

Exercise: Convert the binary number 1001 0110 to a decimal number.

To convert from decimal to binary it is necessary to find the appropriate combination of powers of two that will add up to the desired decimal number. This is done by repeatedly subtracting the largest possible power of two until the remaining value is zero.

For example, to convert the decimal value 35, we find the largest power of two less than or equal to 35: 32 (2^5). The remainder is $35 - 32 = 3$. The next largest power of that can be subtracted is $2^1 = 2$. Subtracting this value leaves $3 - 2 = 1$. The next possible power of two is $2^0 = 1$. Subtracting this value leaves 0. Therefore $35 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^5$ which is 10011 in binary.

Exercise: Convert the decimal number 86 to a binary number.

Hexadecimal Representation

Binary numbers are too verbose for many purposes so we often use use hexadecimal (base 16) numbers. Hex numbers are less verbose but also easier to convert to binary. Since the base is 16, we need 16 different digits. We use the digits 0 to 9 and the letters A to F. The following table shows the the 16 hexadecimal digits and the corresponding values in decimal and binary.

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

To convert from binary to hex we group the binary digits into groups of 4 starting from the least significant (rightmost) digit. The we just look up the corresponding hexadecimal value in the table.

Exercise: Convert the binary number 1101000 to hexadecimal.

Hex notation is used because the conversion between hex and binary is much simpler than between decimal and binary. This is becase each hex digit represents exactly 4 bits.

To convert a decimal number to hex you can first convert the number to binary and then group the bits into groups of 4 bits starting from the right. These 4-bit patterns are then easily converted to hex digits.

Similarly, to convert a hex number back to decimal first convert it to a binary number and find the corresponding decimal value.

Exercise: What are the binary and decimal representations of the hexadecimal value 0x3F?

Hexadecimal and Octal Constants in C

We can use hexadecimal notation in C programs by prefixing the constant using the characters '0x'. The hex digits may be in upper of lower case. For example, the constant `0x21` has the value 33 (decimal).

C also has octal (base 8) constants which are denoted by prefixing a number with a zero. For example the constant `010` has the value 8 (decimal). Octal notation is no longer widely used.

Unfortunately, C does not have binary constants.

C's Bitwise Logical Operators

These operators operate bit-by-bit on the binary representation of their operands.

The bitwise complement operator, `~`, is a unary operator similar to the logical negation operator and has the same precedence. However, it inverts the values of the bits in the binary representation of the operand. If a bit is 0, the bitwise negation sets that bit in the result to 1 and vice versa. For example, `~ 2` has the value `1` (2 is 10 and 1 is 01 in binary).

The bitwise 'and', 'exclusive-or'[1] and 'or' operators are `&`, `^`, and `|`. They result in the operation being applied to the *bits* in the binary representations of their operands. Both operators have lower precedence than the comparison operators but higher precedence than the logical operators. The bitwise 'and' has a higher precedence than the bitwise 'or'.

Exercise: What are the values of the following expressions?

```
( 7 ^ 5 ) | 5
0xAA & 15
```

Note that there is an important difference between logical and bitwise logical operators. For example, the expression `5 && 2` is `1` while `5 & 2` is `0`.

Exercise: Why?

Bit Shift Operators

C also has bit-shift operators. These operators shift the bits in the left operand left (`<<`) or right (`>>`) by the number of bit positions given by the second operand. For example, `x>>2` shifts bits in the binary

[1] The exclusive or operator gives 0 if the two bits are the same and 1 otherwise.

representation of x right by 2 bits. If `x` had the value 0x4 (0100 binary), the value of `x>>2` is 0x1 (0001).

Exercise: What is the value of `0x4 >> 2`? How about `0x4 << 1`? How does shifting the bits in a number left by 1 position affect the value of a number? How about shifting them right?

Each size of integer can only hold a limited number of bits (8, 16 or 32). Thus each shift causes one bit to be shifted "off the end" and another bit to be shifted in at the other end. The bit that is shifted out disappears. For unsigned integers a zero is always shifted in. For right shifts on signed integers the value of the most significant bit is duplicated and shifted in.

Binary Numbers and Logic Levels

Computers represent binary values by using two voltages. For example, one way is to use 0 volts to represent a binary '0' and 5 volts to represent a binary '1'. These voltages, sometimes also called low (L) and high (H), are called logic levels.

A binary number can be represented by a collection of 8 (for a `char`) or 16 (for an `int`) signals[2]. Each signal represents a particular bit of a binary number. A group of related signals is called a *bus*.

Exercise: The connector used between a PC and a printer has 25 signal pins. Pin numbers 9 to 2[3] carry signals generated by the PC that provide the printer with the ASCII value of the character that should be printed. You measure the following voltages: 0,5,0,0, 5,0,0,0 on the signal pins. What character is the printer trying to print?

Note that the opposite convention for logic levels (H for 0 and L for 1) is also often used.

[2] A signal is a voltage that carries information.

[3] in order from most significant to least significant bit.