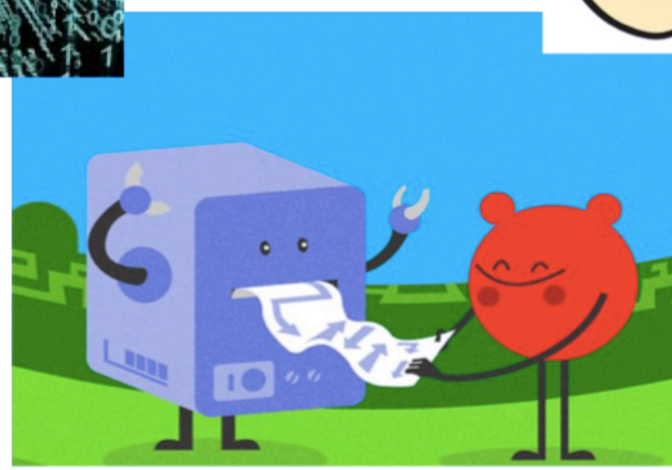
(13-1) Bits and Operations H&K Appendix C

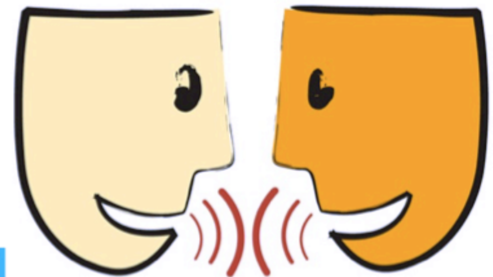
Instructor – Beiyu Lin

CptS 121 (June 5th, 2019)

Washington State University

Basic Memory Concepts (1)





Pictures are from:
https://catborrow.wordpress.com/2011/04/17/hello-world/
https://fcw.com/articles/2018/04/10/odni-cio-cdo-organization.aspx

Basic Memory Concepts (1)

https://www.alamy.com/binary-code-data-bit-screen-display-on-laptop-computer-screen-matrix-of-data-flow-rise-of-the-big-data-ai-age-artificial-intelligence-data-transfer-image207314809.html

Basic Memory Concepts (1)

- Recall when a variable is declared, memory is allocated based on its data type
- Recall some of the major data types in C include:
  - Char (**1 byte**), int (**4 bytes**), and double (**8 bytes**)
- A basic English character (char) requires less memory than an integer (int)
- An integer (int) requires less memory than a double precision floating-point value (double)
- sizeof (char) < sizeof (int) < sizeof (double)
  - Recall sizeof ( ) in C returns the number of bytes allocated for a variable or data type

We already talked about ”sizeof” to calculate the length of an array!
Thank you for your questions!

C. Hundhausen, A. O’Fallon, B. Lin

Basic Memory Concepts (2)

- All information is stored in memory as *bit(s)* of data
  - Bit is derived from binary digit
  - A binary digit or bit has two possible values; 0 or 1
- A sequence of 4-bits is called a *nibble*
  - One example of a nibble of data is 1111_2
    - This is the number 15 in decimal
    - Note the leftmost 1 is referred to as the most significant bit (msb) and the rightmost 1 is the least significant bit (lsb)
- A sequence of 8-bits is called a *byte* (***8 bits => a byte***)

'A' is a char => 1 Byte => 8 bits

"A" can be represented as: $0100\ 0001_2$
(This is the number 65 in decimal)

Number Systems (1)

- Decimal and binary systems are called *positional* number systems
- A digit from one of these systems has a *weight* dependent upon its position or location within the string of digits
- Each position is weighted as the *base* of the system to a power
  - Decimal is base 10
  - Binary is base 2
- A *binary number* consists of one or more bits

C. Hundhausen, A. O'Fallon

Number Systems (2)

- A decimal number 123_{10} actually means the following:
  - 10^2 10^1 10^0
  - 1 2 3
    - The 1 is in the hundreds or 10^2 position
    - The 2 is in the tens or 10^1 position
    - The 3 is in the ones or 10^0 position
- To evaluate a number in a positional number system; pick each digit and multiply by its weighted position and compute the sum
  - $1 * 10^2 + 2 * 10^1 + 3 * 10^0 = 123_{10}$

Details and examples were written on the white board

How Do We Convert from Decimal to Binary? (1)

Number	294		
divide by 2			
result	147	remainder	0 (LSB)
divide by 2			
result	73	remainder	1
divide by 2			
result	36	remainder	1
divide by 2			
result	18	remainder	0

divide by 2			
result	9	remainder	0
divide by 2			
result	4	remainder	1
divide by 2			
result	2	remainder	0
divide by 2			
result	1	remainder	0
divide by 2			
result	0	remainder	1 (MSB)

Given the decimal number 294_{10}

<=>

100100110_2 in binary

Details and examples were written on the white board.

C. Hundhausen, A. O'Fallon, B. Lin

Example is from: https://www.electronics-tutorials.ws/binary/bin_2.html

How Do We Convert from Decimal to Binary? (1) -- Coding

Part 1:
- A number % 2; store the remainder
- update the number by the division

```
int num, bi_arr[32], i;
while(num> 0)
{
  bi_arr[i] = num % 2;
  num = num / 2;
  i++;
}
```

Part 2:
- print out in the reversed order

```
  int j;
  for (j = 31; j > -1; j --)
  {
    printf("%d ", bi_arr[j]);
  }
```


Example is from: https://www.electronics-tutorials.ws/binary/bin_2.html

How Do We Convert from Decimal to Binary? (1)

- Let's convert 123_{10} to a binary number represented by one byte or 8-bits:
  - First note we need the following weights for an 8-bit number
    - 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
  - Then determine if the largest power of 2 (2^7 in this case) goes into 123_{10}
    - No it does not! Recall 2^7 is 128_{10}; so place a 0 in the 2^7 position
  - Next determine if 2^6 goes into 123_{10}
    - Yes it does! Recall 2^6 is 64_{10}; so place a 1 in the 2^6 position
    - Subtract 64_{10} from 123_{10}; result is 59_{10}

Details and examples were written on the white board

How Do We Convert from Decimal to Binary? (2)

- Next determine if 2^5 goes into 59_{10}
  - Yes it does! Recall 2^5 is 32_{10}; so place a 1 in the 2^5 position
  - Subtract 32_{10} from 59_{10}; result is 27_{10}
- Let's try one more; does 2^4 go into 27_{10}
  - Yes it does! Recall 2^4 is 16_{10}; so place a 1 in the 2^4 position
  - Subtract 16_{10} from 27_{10}; result is 11_{10}

C. Hundhausen, A. O'Fallon

How Do We Convert from Decimal to Binary? (3)

- Can you finish the rest of the process?

- The binary number should be:
  - 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
  - 0 1 1 1 1 0 1 1_2
  - Note the digit in the 2^0 position is 1; this means the number is odd; otherwise it would be 0

Details and examples were written on the white board

How Do We Convert From Binary to Decimal?

- Let's convert a nibble 1010_2 to a decimal number:
  - First note we need the following weights for a 4-bit number
    - 2^3 2^2 2^1 2^0, where the leftmost or msb 1 is in the 2^3 position, and the rightmost or lsb 0 is in the 2^0 position
  - Next pick off the each digit from the binary number and multiply by its corresponding positional weight
    - $1 * 2^3 = 8_{10}$
    - $0 * 2^2 = 0_{10}$
    - $1 * 2^1 = 2_{10}$
    - $0 * 2^0 = 0_{10}$
  - Lastly, sum up each individual result
    - $8_{10} + 0_{10} + 2_{10} + 0_{10} = 10_{10}$
  - The final result is 10_{10}

C. Hundhausen, A. O'Fallon

Getting Started with Bitwise Operations in C

- The C language supports several bit operations – i.e. operations that are applied to each individual bit in a number
  - These include: left shift (<<), right shift (>>), negation (~), bitwise AND (&), bitwise OR (|), and exclusive OR, also known as XOR, (^)

C. Hundhausen, A. O'Fallon

Applying Bitwise Operators (1)

- $1011_2 << 2$; means shift each bit in the number to the left by two positions and rotate in zeros
  - The result is 1100_2 if only a nibble of memory is available; otherwise it's 101100_2
- $1011_2 >> 1$; means shift each bit in the number to the right by one position and rotate in zeros
  - The result is 0101_2; note the lsb is lost in the result

Applying Bitwise Operators (2)

- 1010_2 & 0011_2; means AND each bit in each corresponding position
  - The result is 0010_2
- 1010_2 | 0011_2; means OR each bit in each corresponding position
  - The result is 1011_2

C. Hundhausen, A. O'Fallon

Applying Bitwise Operators (3)

- 1010_2 ^ 0011_2; means XOR each bit in each corresponding position
  - The result is 1001_2
- ~1010_2; means negate or "flip" each bit
  - The result is 0101_2

Why Apply Bitwise Operators? (1)

- Each position shifted to the left with a binary number indicates multiplication by 2
  - i.e. 8_{10} << 3 results in 64_{10}
- Each position shifted to the right with a binary number indicates division by 2
  - i.e. 4_{10} >> 1 results in 2_{10}
- Shift operations are much more efficient than multiplication and division operations!

Why Apply Bitwise Operators? (2)

- Bitwise AND may be used to clear bits; AND any bit with 0, the result is 0
- Bitwise OR may be used to set bits; OR any bit with 1, the result is 1
- XOR may be used to toggle bits; XOR any bit with 1, the result is the negation of the bit
  - 0 → 1 or 1 → 0, where → represents "becomes"

Basic Bit Manipulation

- Set union A | B
- Set intersection A & B
- Set subtraction A & ~B
- Set negation ALL_BITS ^ A or ~A
- Extract last bit A&-A or A&~(A-1) or x^(x&(x-1))
- Remove last bit A&(A-1)
- Get all 1-bits ~0

How to Interpret Bits?

- A bit may represent the state of a physical light switch
  - i.e. 1 means the light switch is on; 0 means the light switch is off
- A bit may also represent the state of an operation
  - i.e. is x == y? 1 means yes; 0 means no
- Can bitwise operators be used to encrypt/decrypt data?
- Many other interpretations exist. Be creative!

Next Lecture…

- Dynamic memory allocation

References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8th Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7th Ed.)*, Pearson Education , Inc., 2013.

C. Hundhausen, A. O'Fallon

Collaborators

- Chris Hundhausen
- Andrew O'Fallon