CS 241
Data Organization using C
Binary and Bit Operators

Instructor: **Joel Castellanos**
e-mail: joel@unm.edu
Web: http://cs.unm.edu/~joel/
Office: Farris Engineering Center, Room 2110

9/5/2019

1

Workflow Tips

1) Use 2 or 3 PuTTy windows: code, compile, and input.
2) vim: If error on line 45, use **`<esc>:45`**.
3) vim: Go to top of file: **`<esc>:0`** or **`gg`**. Bottom: **`<esc>G`**.
4) vim: Search **`<esc>/`***string*.
5) When debugging, use many **`printf`** statements so you ***know*** where your program is flowing and what the values of intermediate fields are.
6) Edit data input file to just one line. When that line works, replace it with a new line – that way you do not generate too much debug output.

2

2

Quiz: Reading 2.9 Bitwise Operators

In the C programming language, the ^ operator performs:

a) Bitwise AND

b) Bitwise OR

c) Bitwise Exclusive OR

d) Two's Complement

e) One's Complement

3

Combinations and Permutations

In English we use the word "combination" loosely, without thinking if the order of things is important. In other words:

"*My fruit salad is a **combination** of apples, grapes and bananas*" In this statement, order does not matter: "bananas, grapes and apples" or "grapes, apples and bananas" make the same salad.

"*The **combination** to the safe is 472*". Here the order is important: "724" would not work, nor would "247".

In Computer Science we use more precise language:

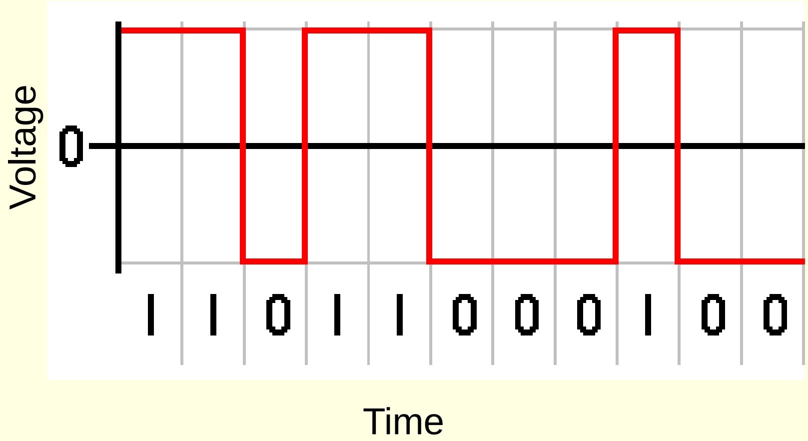
If the order doesn't matter, it is a ***Combination***.

If the order does matter it is a ***Permutation***.

1. ***Repetition is Allowed***: such as the lock above. It could be "333".
2. ***No Repetition***: for example the first three people in a running race. Order does matter, but you can't be first ***and*** second.

4

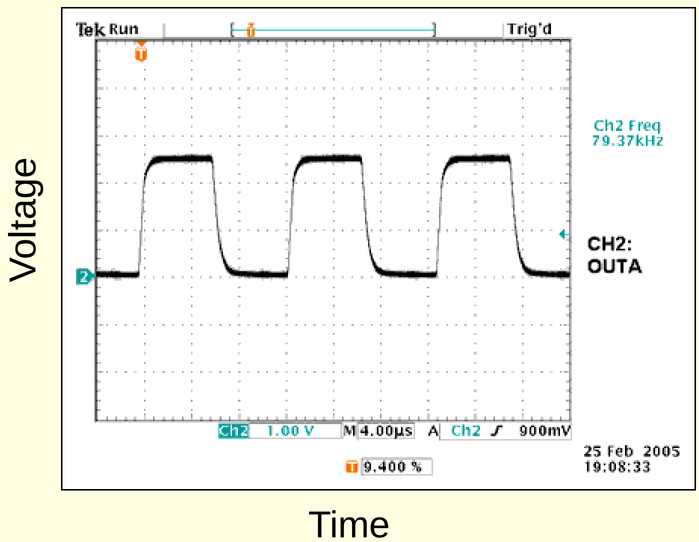
Ideal Serial Binary Signal



5

5

Real Binary Signal (Maxim MAX5581)



6

6

Information in a Binary Signal

1 Bit
2 Permutations

0
1

2 Bits
4 Permutations

0	0
0	1
1	0
1	1

3 Bits
8 Permutations

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

4 Bits
16 Permutations

0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	1	1	0	0
0	1	0	1	1	1	0	1
0	1	1	0	1	1	1	0
0	1	1	1	1	1	1	1

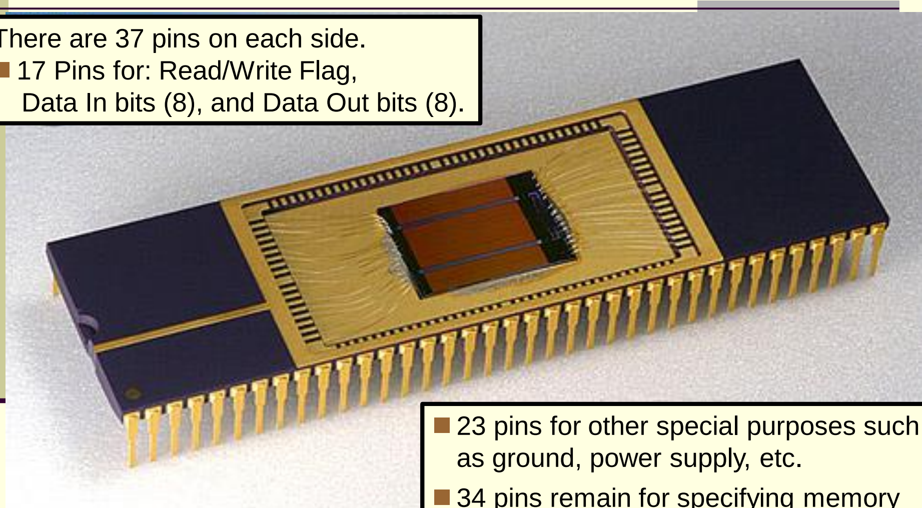
7

7

Parallel Binary Signals: Flash Memory Chip

There are 37 pins on each side.

- 17 Pins for: Read/Write Flag, Data In bits (8), and Data Out bits (8).



- 23 pins for other special purposes such as ground, power supply, etc.
- 34 pins remain for specifying memory addresses.

How many bytes can be addressed?

8

8

Numbers in Base Ten and Base Two

Base 10

$$5307 = 5\times10^3 + 3\times10^2 + 0\times10^1 + 7\times10^0$$
$$= 5000 + 300 + 0 + 7$$

Base 2

$$1011 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$
$$= 8 + 0 + 2 + 1$$

9

Examples of Binary Numbers

0	0	0	0	1	0	0	0	1	1
512	256	128	64	32	16	8	4	2	1

= 35

0	0	0	0	1	1	1	1	1	1
512	256	128	64	32	16	8	4	2	1

= 63

0	0	0	1	0	0	0	0	0	0
512	256	128	64	32	16	8	4	2	1

= 64

1	1	0	1	1	0	0	0	1	1
512	256	128	64	32	16	8	4	2	1

= 867

10

Quiz: Binary

The binary number, 00101010, in base-ten is:

a) 101010

b) 1010

c) 128

d) 75

e) 42

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

Quiz: Binary

The binary number, 10101010, in base-ten is:

a) 170

b) 76

c) 52

d) 47

e) 42

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

Hexadecimal: Base-16

Hexadecimal (or hex) is a base-16 system that uses sixteen distinct symbols, most often the symbols 0–9 to represent values zero to nine, and A, B, C, D, E, F to represent values ten to fifteen.

Base 16

$$
\begin{aligned}
\text{0x53AC} &= 5\times16^3 + 3\times16^2 + 10\times16^1 + 12\times16^0 \\
&= 5\times4096 + 3\times256 + 10\times16 + 12\times1 \\
&= 20{,}480 + 768 + 160 + 12 \\
&= 21{,}420
\end{aligned}
$$

13

13

Why Hexadecimal?

Hexadecimal is trivially more compact than base-10, but ***significantly more compact*** than base-2.

Since 16 is a power of 2, it is very easy to convert between Binary and Hexadecimal

Base 16

0x01239ACF: Four bytes:

01 23 9A CF

0000 0001 0010 0011 1001 1010 1100 1111

14

14

Hexadecimal Literals

```
#include <stdio.h>
void main(void)
{ printf("%d\n", 0x1);
  printf("%d\n", 0x2);
  printf("%d\n", 0x3);
  printf("%d\n", 0x8);
  printf("%d\n", 0x9);
  printf("%d\n", 0xA);
  printf("%d\n", 0xB);
  printf("%d\n", 0xC);
  printf("%d\n", 0xD);
  printf("%d\n", 0xE);
  printf("%d\n", 0xF);
  printf("%d\n", 0x10);
  printf("%d\n", 0x11);
  printf("%d\n", 0x12);
}
```

```
1
2
3
8
9
10
11
12
13
14
15
16
17
18
```

15

15

Powers of 2: char, int

```
#include <stdio.h>
void main(void)
{
  char i=0;
  char a=1;
  unsigned char b=1;
  int c = 1;

  for (i=1; i<22; i++)
  {
    a = a * 2;
    b = b * 2;
    c = c * 2;
    printf("%2d) %4d %3d %7d\n",
      i, a, b, c);
  }
}
```

```
 1)    2   2       2
 2)    4   4       4
 3)    8   8       8
 4)   16  16      16
 5)   32  32      32
 6)   64  64      64
 7) -128 128     128
 8)    0   0     256
 9)    0   0     512
10)    0   0    1024
11)    0   0    2048
12)    0   0    4096
13)    0   0    8192
14)    0   0   16384
15)    0   0   32768
16)    0   0   65536
17)    0   0  131072
18)    0   0  262144
19)    0   0  524288
20)    0   0 1048576
21)    0   0 2097152
```

16

16

Powers of 2: int, long

```
#include <stdio.h>

void main(void)
{
  char i=0;
  int  c=1;
  long d = 1;


  for (i=1; i<65;
    i++)
  {
    c = c * 2;
    d = d * 2;
    printf("%2d) %11d %20ld\n", i, c, d);
  }
}
```

```
...
29)   536870912             536870912
30)  1073741824            1073741824
31) -2147483648            2147483648
32)           0            4294967296
33)           0            8589934592
...
61)           0   2305843009213693952
62)           0   4611686018427387904
63)           0  -9223372036854775808
64)           0                     0
```

Format code: **ld** for long decimal

17

Bitwise Operators

- **&** bitwise AND **1010 & 0011 = 0010**
- **|** bitwise inclusive OR **1010 | 0011 = 1011**
- **^** bitwise exclusive OR **1010 ^ 0011 = 1001**
- **~** bitwise NOT **~1010 = 0101**

- **<<** left shift **00000100 << 3 = 00100000**
- **>>** right shift **00000100 >> 2 = 00000001**

18

Shift Operator Example

```c
void main(void)
{ int i;
  for (i=0; i<8; i++)
  { unsigned char n = 1 << i;
    printf("n=%d\n", n);
  }
}
```

Output:

```
n=1
n=2
n=4
n=8
n=16
n=32
n=64
n=128
```

19

Quiz: Bitwise AND Operator

```c
#include <stdio.h>

void main(void)
{
  printf("%d\n", 26 & 28);
}
```

The output is:
a) 0
b) 4
c) 8
d) 12
e) 24

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

20

Quiz: Bitwise OR Operator

```
#include <stdio.h>

void main(void)
{
   printf("%d\n", 26 | 28);
}
```

The output is:

a) 26
b) 28
c) 30
d) 42
e) 54

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

21

21

Convert 77 to an 8-bit Binary String

2^7 = 128 is > 77 put a '0' in the 128's place.

0							

2^6 = 64 is <= 77, put a '1' in the 64's place, AND, subtract 64: **77 - 64 = 13**.

0	1						

2^5 = 32 is > 13. Put a '0' in the 32's place.

0	1	0					

2^4 = 16 is > 13. Put a '0' in the 16's place.

0	1	0	0				

2^3 = 8 is <= 13. Put a '1' in the 8's place AND subtract 8: **13 – 8 = 5**.

0	1	0	0	1			

2^2 = 4 is <= 5. Put a '1' in the 4's place.

0	1	0	0	1	1		

AND subtract 2: **5 – 4 = 1**.

2^1 = 2 is > 1. Put a '0' in the 2's place

0	1	0	0	1	1	0	

2^0 = 1 is <=1. Put a '1' in the 1's place AND subtract 1: **1 – 1 = 0**.

0	1	0	0	1	1	0	1

22

22

Convert unsigned char to Binary Array

```c
#include <stdio.h>
void main(void)
{
  char bits[9];
  bits[8] = '\0';
  unsigned char n=83;
  unsigned char powerOf2 = 128;
  int i;
  for (i=0; i<=7; i++)
  { if (n >= powerOf2)
    { bits[i] = '1';
      n = n-powerOf2;
    }
    else bits[i] = '0';
    powerOf2 /= 2;
  }
  printf("%s\n", bits);
}
```

```
01010011
```

23

23

The Mask

In computer science, a *mask* or *bitmask* is data that is used for a bitwise AND operation.

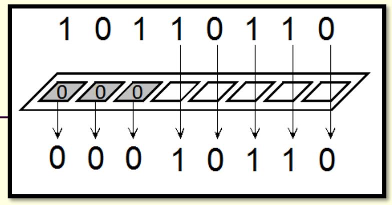
Using a mask, multiple bits in a byte, short, int, long or other machine type can be set blocked in a single bitwise operation.

	1	0	1	1	0	1	0	1	← Original Value
&	0	0	0	0	1	1	1	1	← Mask
	0	0	0	0	0	1	0	1	← Result

24

24

The Mask




```
void main(void)
{
  unsigned short x = 30; //00011110
  unsigned short mask = 0xF7; //11110111

  //Turn OFF 8 bit. If 8 bit is already OFF,  x is unchanged.
  x = x & mask;
  printf("%d\n", x); //prints: 22

  //Turn OFF all bits except the first 3.
  x = x & 0x07; // 0x07 = 00000111
  printf("%d\n", x); // prints: 6 (00000110)

  //True (non-zero) if 8 bit is ON in x.
  if (x & (~mask)) printf("true\n");
}
```


Using the Mask: Binary Array

```
#include <stdio.h>
void main(void)
{
  char bits[9];
  bits[8] = '\0';
  unsigned char n = 83;
  unsigned char powerOf2 = 128;
  int i;
  for (i=0; i<=7; i++)
  { if (n & powerOf2) bits[i] = '1';
    else bits[i] = '0';
    powerOf2 = powerOf2 >> 1;
  }
  printf("%s\n", bits);
}
```

In the first method, whenever a power of 2 is found, it is subtracted from **n**.
This method never changes **n**.

Output:

```
00010001
```

Addition: Base 10 and Binary

	1
	2 9
+	5 6
	8 5

		1	1	1				
	0	0	0	1	1	1	0	1
+	0	0	1	1	1	0	0	0
	0	1	0	1	0	1	0	1

27

Overflow Addition

```
#include <stdio.h>
void main (void)
{
  char i=0;
  char a = 123, b = 252;
  unsigned char x = 123, y = 252;
  for (i=1; i<=7; i++)
  {
    a++; b++; x++; y++;
    printf("%4d %4d %4d %4d\n", a,   x,   b,   y);
  }
}
```

124	124	-3	253
125	125	-2	254
126	126	-1	255
127	127	0	0
-128	128	1	1
-127	129	2	2
-126	130	3	3

	0	1	1	1	1	1	1	1
+	0	0	0	0	0	0	0	1
	1	0	0	0	0	0	0	0

28

Quiz: Bitwise Operator

```c
#include <stdio.h>

void main(void)
{
  printf("%d\n", 70 & 73);
}
```

The output is:
- a) 3
- b) 42
- c) 64
- d) 142
- e) 143

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

29

29

Quiz: << and &

```c
#include <stdio.h>
void main(void)
{
  char bits[9];
  bits[8] = '\0';
  unsigned char n=37;
  unsigned char p = 128;
  int i;
  for (i=0; i<=7; i++)
  { if (n & p) bits[i] = '1';
    else bits[i] = '0';
    p = p >> 1;
  }
  printf("%s\n", bits);
}
```

- a) 00110101
- b) 00100101
- c) 00101101
- d) 00100111
- e) 00111101

30

30

Quiz: << and &

```
void main(void)
{ unsigned char a=37;
  int i;

  for (i=7; i>=0; i--)
  { unsigned char n = 1 << i;
    if (!(a & n)) printf("%d, ", n);
  }
  printf("\n");
```

a) 64, 32, 16, 4, 1,
b) 64, 16, 8, 4, 2, 1,
c) 64, 16, 8, 4, 1,
d) 128, 64, 16, 8, 1,
e) 128, 64, 16, 8, 2,

Two's Complement

From ordinary binary:
Flip the bits and Add 1.

5:	0	0	0	0	1	0	1
Flip Bits:	1	1	1	1	0	1	0
Add 1:	0	0	0	0	0	0	1
-5:	1	1	1	1	0	1	1

Ordinary Binary	Decimal
0000 0001	1
0000 0010	2
0000 0011	3
0000 0100	4
0000 0101	5
0000 0010	6
0000 0111	7

Two's Complement	Decimal
1111 1111	−1
1111 1110	−2
1111 1101	−3
1111 1100	−4
1111 1011	−5
1111 1010	−6
1111 1001	−7

Two's Complement Addition

29		0	0	0	1	1	1	0	1

	1	1	1	1	1	1	1	1	
29		0	0	0	1	1	1	0	1
+ -29	+	1	1	1	0	0	0	1	1
0		0	0	0	0	0	0	0	0

	1	1	1	1	1	1			
7		0	0	0	0	0	1	1	1
+ -4	+	1	1	1	1	1	1	0	0
3		0	0	0	0	0	0	1	1

33

33

There are 10 types of people: those who understand binary and those who don't.

34

34