3. Data Representation (1/2)

Prof. Jong-Chan Kim

Dept. Automobile and IT Convergence

KMU 국민대학교
KOOKMIN UNIVERSITY

Various Kinds of Data in Computer Systems

Real Number

String

3 3.14 ‘h’ “hello”

Integer

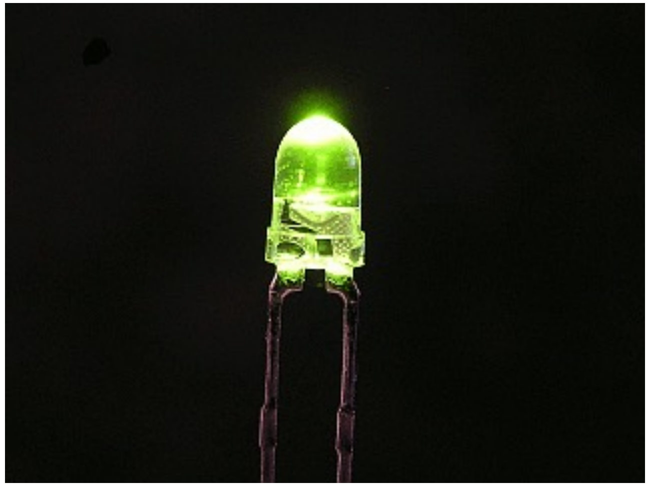
Character

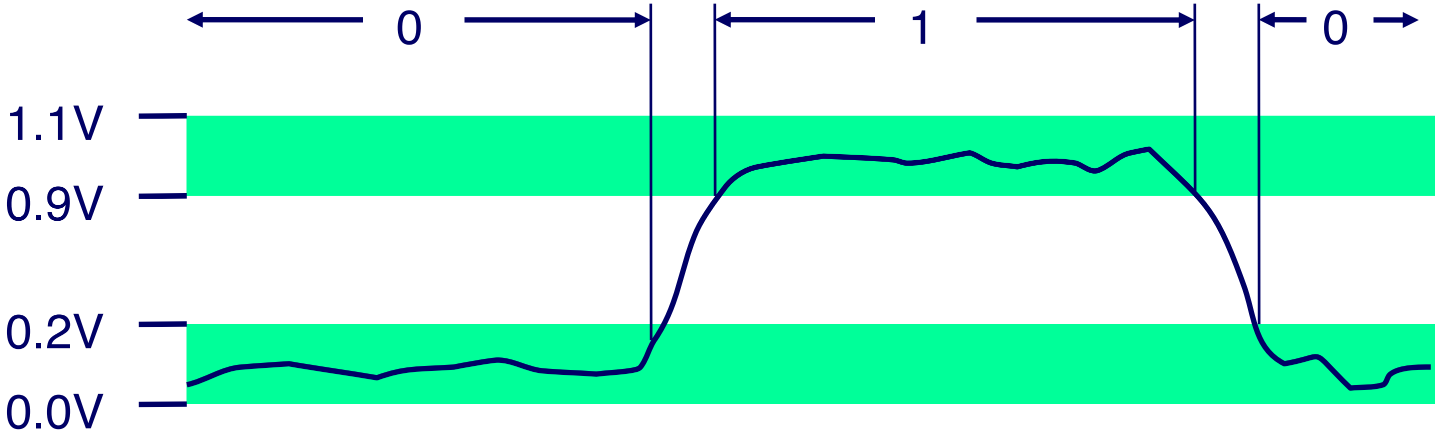
Image

Visit http://www.lenna.org/
for the story of this image

Bit

- A data holder that can be either 0 or 1
- The smallest unit of data in computers
- Can represent two different states
  - Yes or No, Black or White, On or Off, ...
- Can be implemented by the voltage level of an electrical component





Source: Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Source: https://syamtrd.wordpress.com/2015/04/13/understanding-led-blinking-in-arduino/

Byte, Word, and Double Word

- Byte: a sequence of eight adjacent bits

Can represent 2^8 = 256 different states

0	1	0	1	1	0	0	0

- Can store a single character
- The unit of memory addressing

- Word: two adjacent bytes (16 bits)

Can represent 2^16 = 65536 different states

$$\underbrace{01011000}_{\text{Byte}}\ \underbrace{10111001}_{\text{Byte}}$$

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

- Another meaning of word: "the natural data size of a processor"
- 32-bit processor's word size is 32 bits
- 64-bit processor's word size is 64 bits

- DWord: two adjacent words (32 bits)

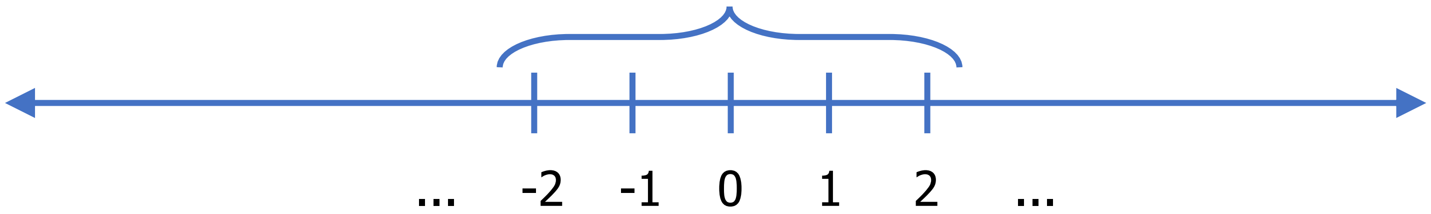
Can represent 2^32 different states

$$\underbrace{0101100010111001}_{\text{Word}}\ \underbrace{0100011110001111}_{\text{Word}}$$

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

Integers

- Numbers without fractional components



- Given a certain range, there is only a finite number of integers in it
  - Within -2.1 and 2.1, there are only five integers
  - 3 bits are enough (2^3) to represent five different integers

Integer Representation

- Like children using their fingers
  - Decimal notation
- Computers express numbers using bits
  - Binary notation (base 2) is the natural choice
- For example, four bits can represent integers like
  - 0 ~ 15
  - 1 ~ 16
  - -8 ~ 7
  - …
- For a given range of integer numbers, we can
  - Find the required number of bits
  - Define a mapping bet'n bit sequences and numbers

Mapping between four bits and 0 ~ 15 using the simple base-2 or binary numerical system

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Another possible mapping

Decimal	Binary
-8	0000
-7	0001
-6	0010
-5	0011
-4	0100
-3	0101
-2	0110
-1	0111
0	1000
1	1001
2	1010
3	1011
4	1100
5	1101
6	1110
7	1111

Data Type

- A combination of
  - Data length n (number of bits)
  - Mapping bet'n 2^n bit sequences and the set of numbers to be represented
- Integer data types
  - Memory sizes can vary depending on the compiler implementation
  - Portability problem when moving to another C compiler

Data Type	Typical Memory Size (bytes)	Value Range
(signed) char	1	-128 ~ 127
unsigned char	1	0 ~ 255
(signed) short	2	-32768 ~ 32767
unsigned short	2	0 ~ 65535
(signed) int	4	-2147483648 ~ 2147483647
unsigned int	4	0 ~ 4294967295
(signed) long	8	-9223372036854775808 ~ 9223372036854775807
unsigned long	8	0 ~ 18446744073709551615

Data Size Portability Issue

Try size.c

- In 64-bit compiler, sizeof(long) is 8
- In 32-bit compiler, sizeof(long) is 4

Do not assume about the length of data types

```c
#include <stdio.h>

int main(void)
{
    long l;

    printf("%zu\n", sizeof(l));

    return 0;
}
```

Do not use "%d" or "%ld" for sizeof(), which are not portable

For 32-bit compiler

```
$ sudo apt update
$ sudo apt install gcc-multilib
$ gcc size.c
$ ./a.out
8
$ gcc size.c -m32
$ ./a.out
4
```

Run the 32-bit compiler

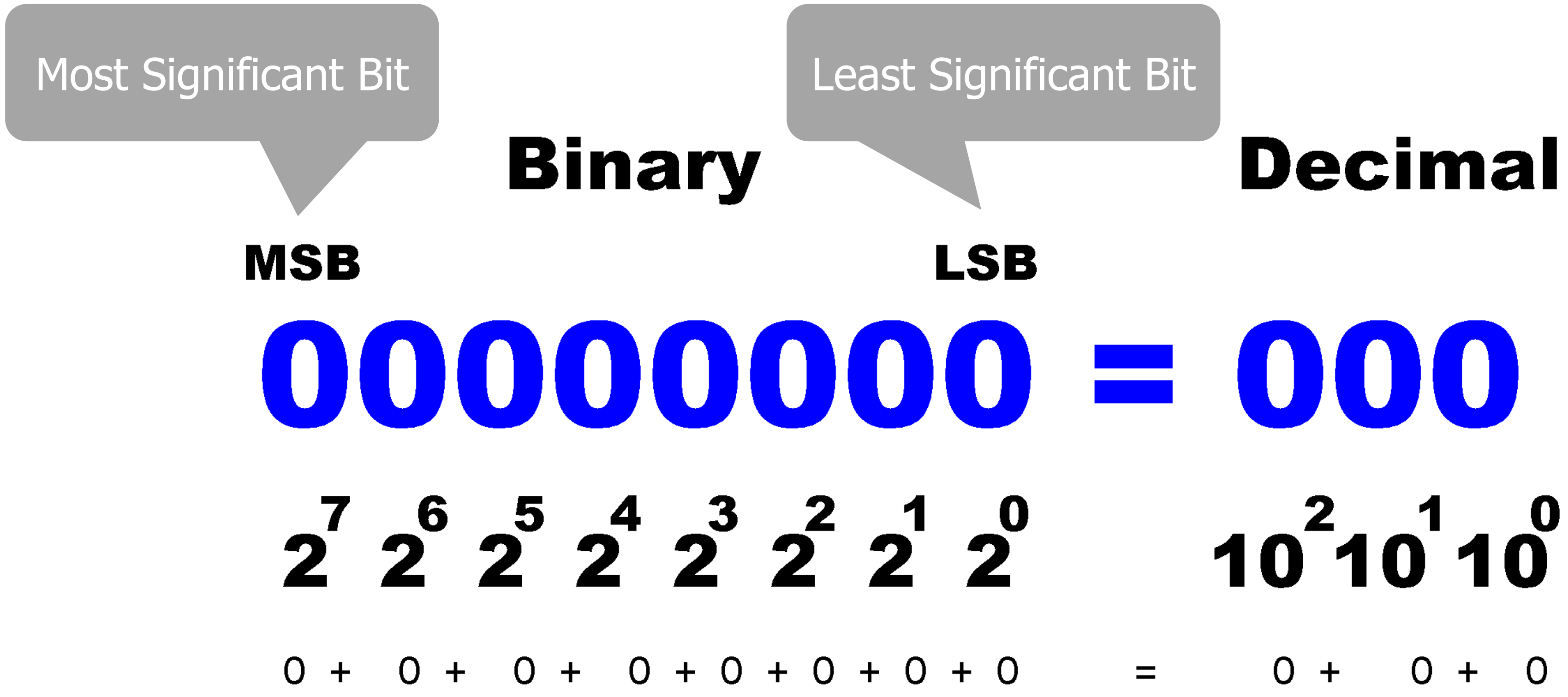
Portable Integer Data Types (stdint.h)

- C99 standard newly defined stdint.h with portable integer data types

`int8_t`	8-bit signed integers
`int16_t`	16-bit signed integers
`int32_t`	32-bit signed integers
`int64_t`	64-bit signed integers
`uint8_t`	8-bit unsigned integers
`uint16_t`	16-bit unsigned integers
`uint32_t`	32-bit unsigned integers
`uint64_t`	64-bit unsigned integers

Mapping for Unsigned Types

- A single byte can represent 256 different states
- We can interpret these eight bits as it is a binary number



Source: https://gifer.com/en/QYHd

Signed Types: A Naïve method

- Interprets the MSB as a sign bit (0:+, 1:-) and the remainings as it is

1	1	1	1	1	1	1	0

Sign bit indicates minus (-)   Then interpret as 1111110_2

4-bit Binary Expressions	Signed Decimal (Naïve Method)
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-0
1001	-1
1010	-2
1011	-3
1100	-4
1101	-5
1110	-6
1111	-7

What is the problem of the naïve method?

Signed Types: Two’s Complement Method

- If sign bit is 0 (+)
  - Interpret it as

    +(As it is)

- If sign bit is 1 (-)
  - Interpret it as

    -(Two’s complement of it)

Sign bit

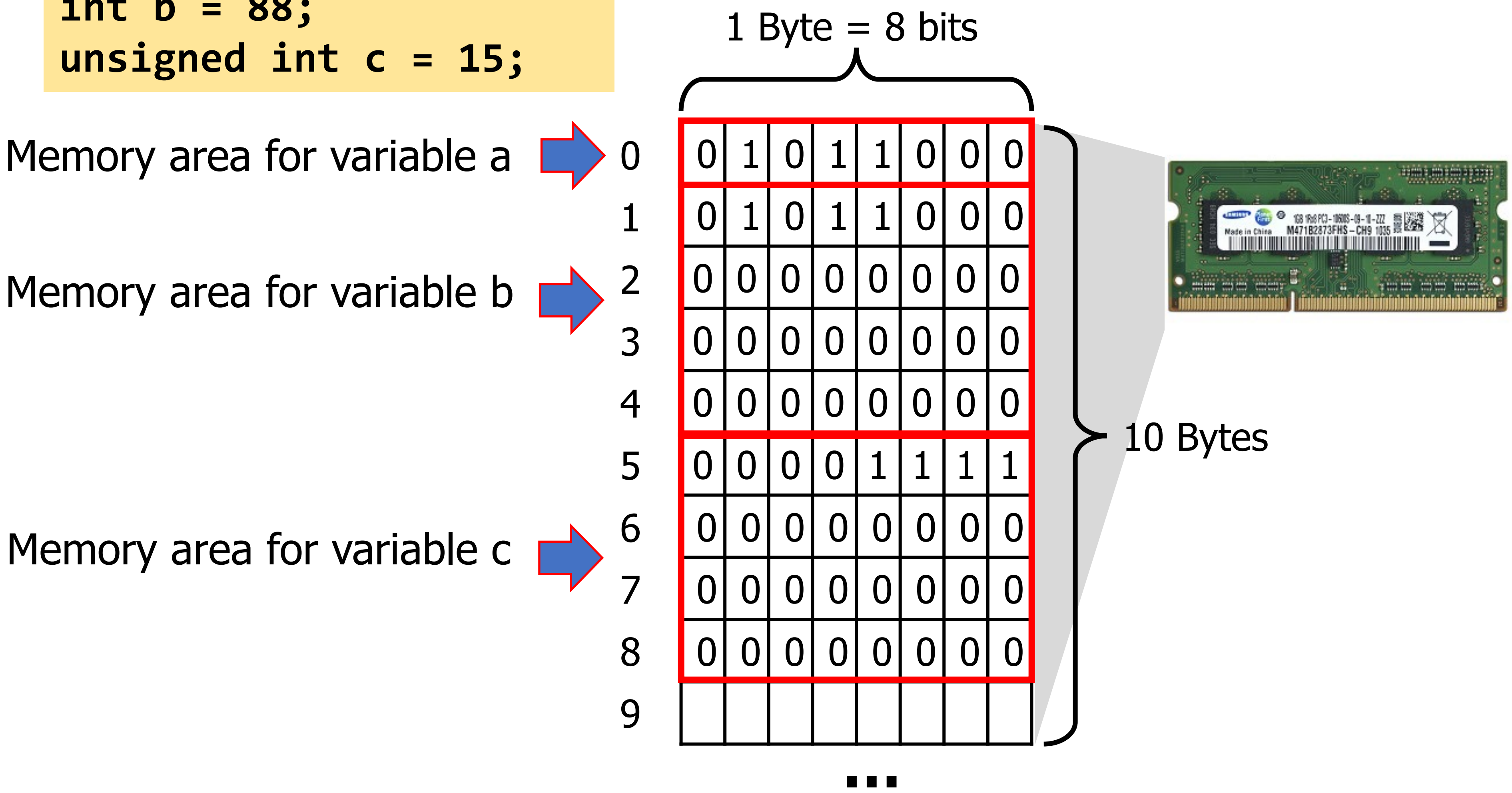
Inverting all bits

One’s complement + 1

4-bit Binary Expressions	Signed Decimal (Naïve Method)	One’s complement	Two’s complement	Signed Decimal (Two’s Complement)
0000	+0	1111	0000	0
0001	+1	1110	1111	+1
0010	+2	1101	1110	+2
0011	+3	1100	1101	+3
0100	+4	1011	1100	+4
0101	+5	1010	1011	+5
0110	+6	1001	1010	+6
0111	+7	1000	1001	+7
1000	-0	0111	1000	-8
1001	-1	0110	0111	-7
1010	-2	0101	0110	-6
1011	-3	0100	0101	-5
1100	-4	0011	0100	-4
1101	-5	0010	0011	-3
1110	-6	0001	0010	-2
1111	-7	0000	0001	-1

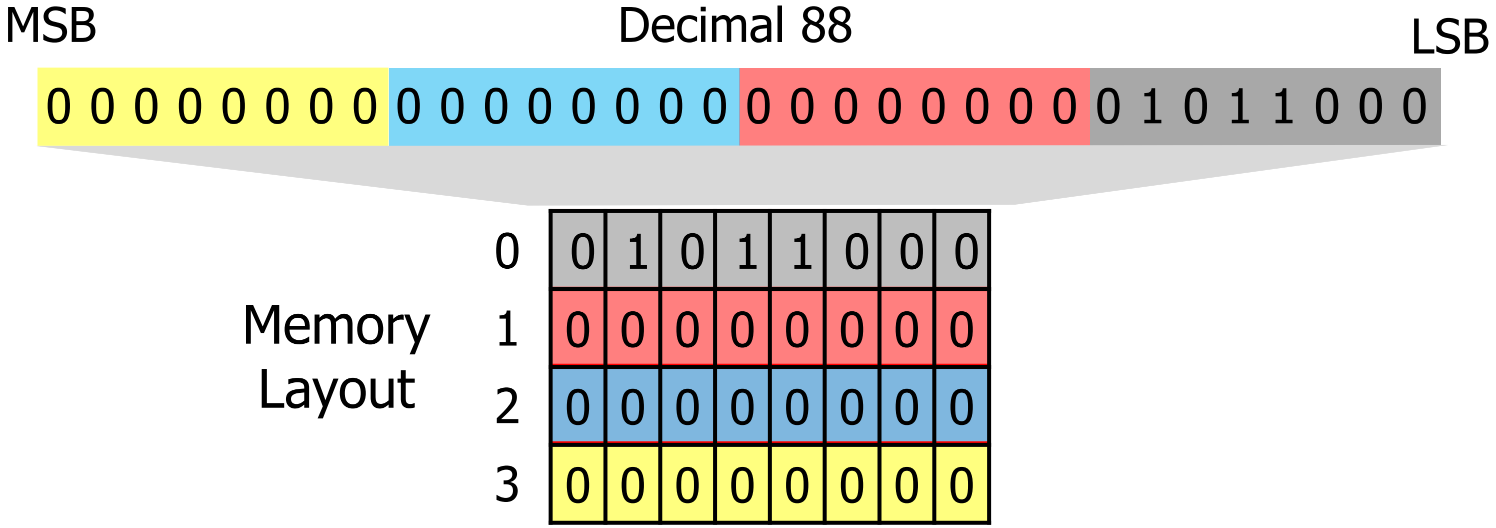
Data Types Spanning Multiple Bytes

```
char a = 88;
int b = 88;
unsigned int c = 15;
```

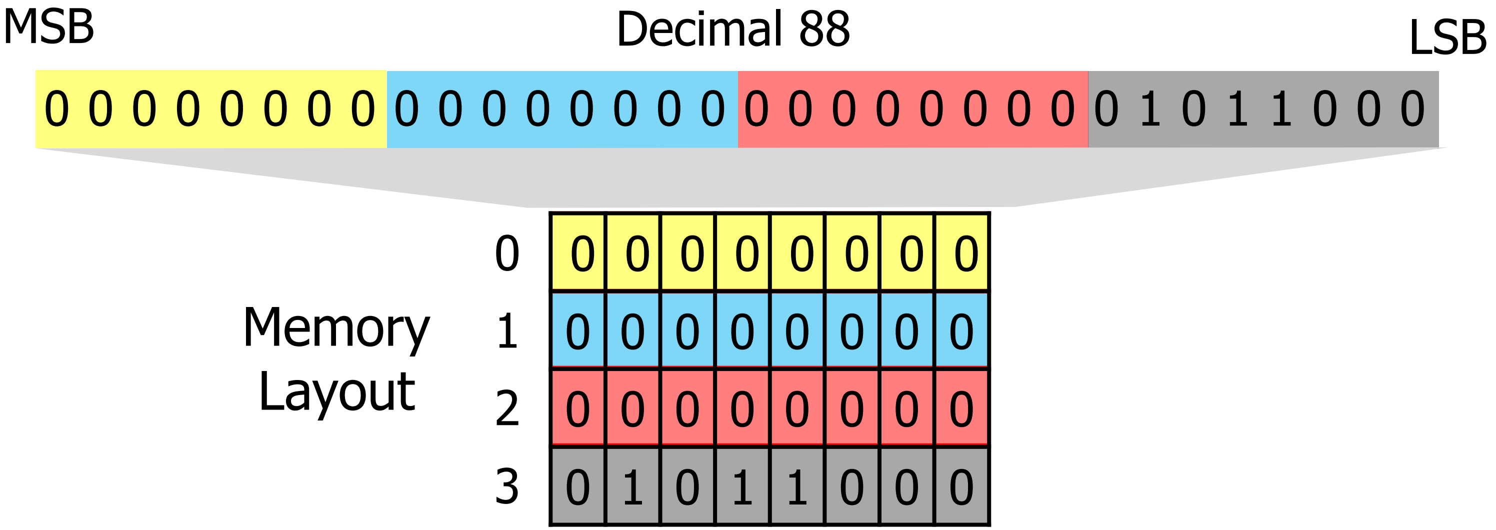


Byte Ordering

- Little-Endian Ordering
  - e.g.) Intel CPUs

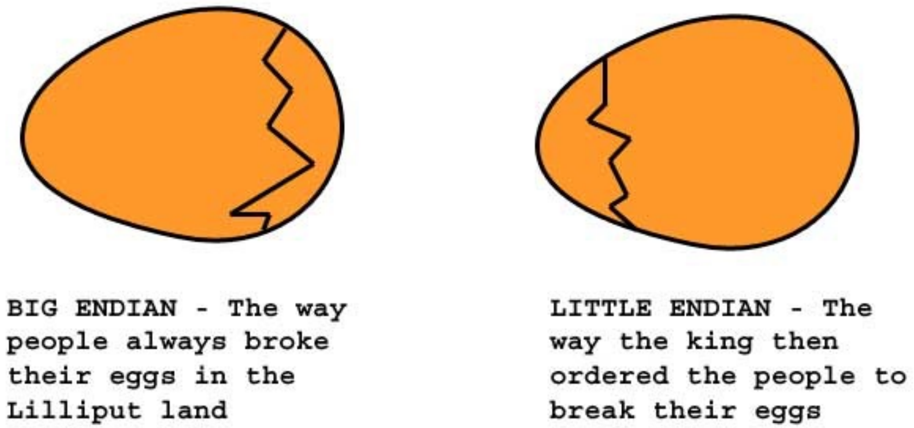


- Big-Endian Ordering
  - e.g.) Most ARM CPUs



Origin of Endianness

http://www.conceptualfiction.com/gullivers_travels.html



http://hardboiledpoker.blogspot.nl/2011/06/order-of-flop.html

You remember, the factions Gulliver encounters in the land of Lilliput, with one group insisting upon first breaking eggs on the big end and the other adamant about breaking them on the small end? War erupts between Lilliput and nearby Blefuscu over the matter, and thousands die as a result.

Source: http://hardboiledpoker.blogspot.nl/2011/06/order-of-flop.html

Checking Endianness

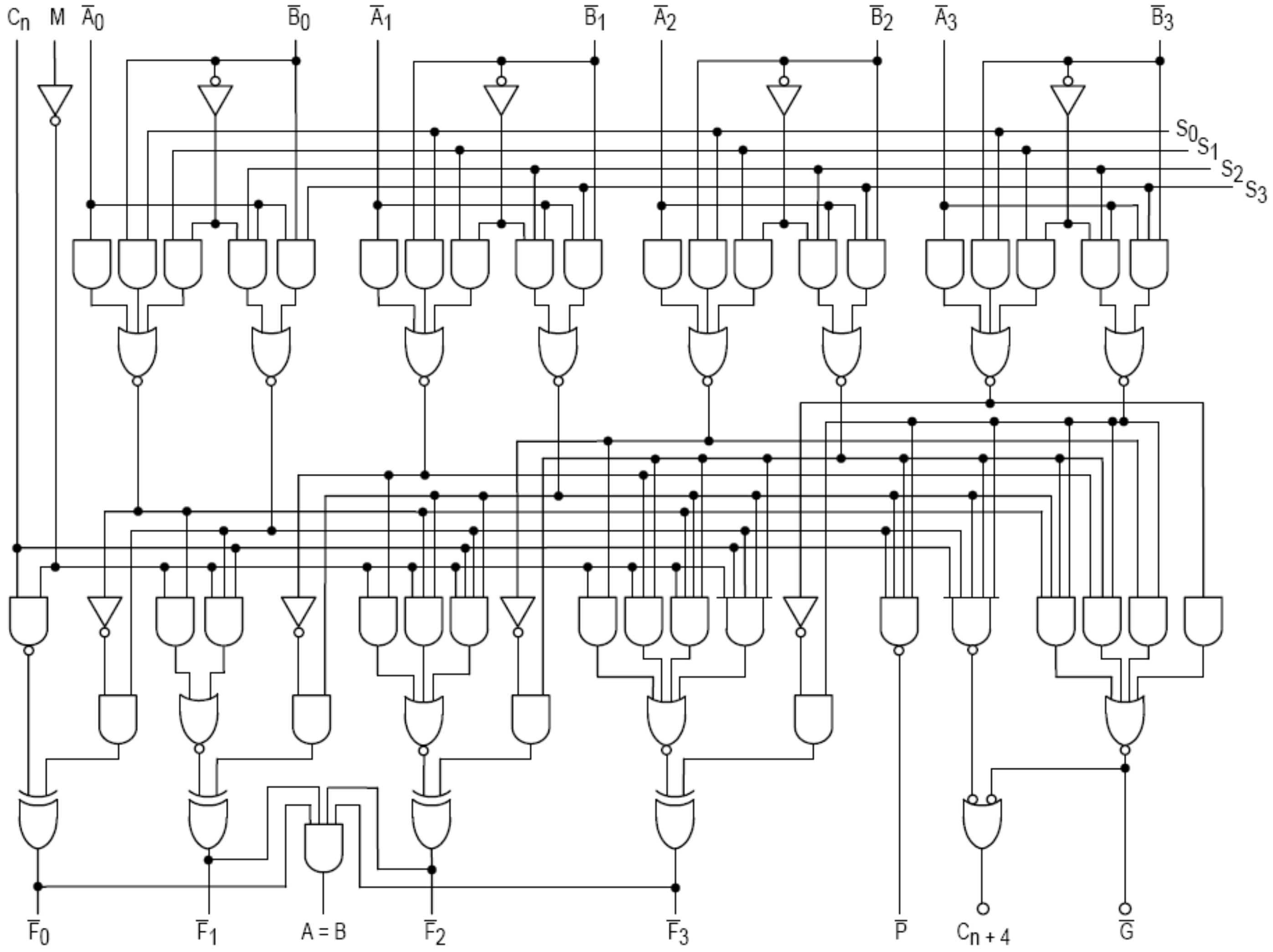
Try endian.c

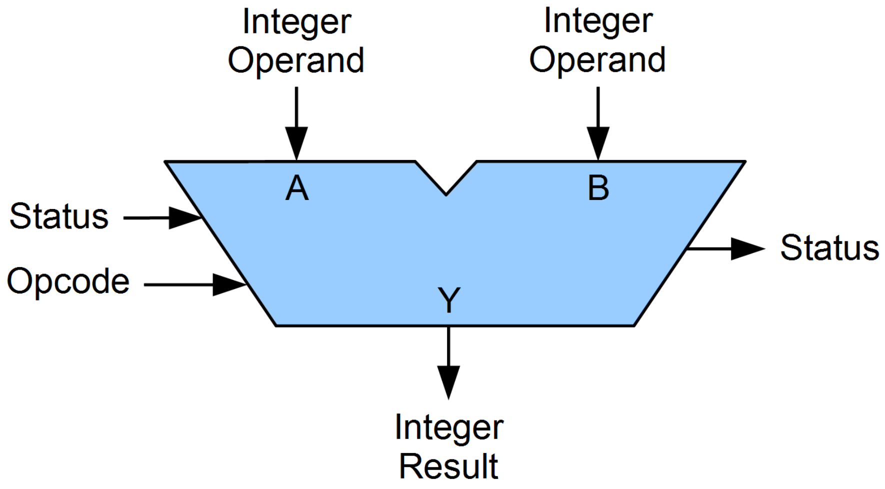
```
#include <stdio.h>

int main(void)
{
    unsigned int a = 1;
    printf("%02x %02x %02x %02x\n",
        ((unsigned char *)&a)[0],
        ((unsigned char *)&a)[1],
        ((unsigned char *)&a)[2],
        ((unsigned char *)&a)[3]);
    return 0;
}
```

Integer Arithmetic

- Arithmetic logic unit (ALU) inside CPU does the arithmetic operations





Source: https://en.wikipedia.org/wiki/Arithmetic_logic_unit

Integer Overflow and Underflow

- Overflow
  - Goes above the range of a data type

1	1	1	1	1	1	1	1

+1 ➡

0	0	0	0	0	0	0	0

255 + 1 = 0

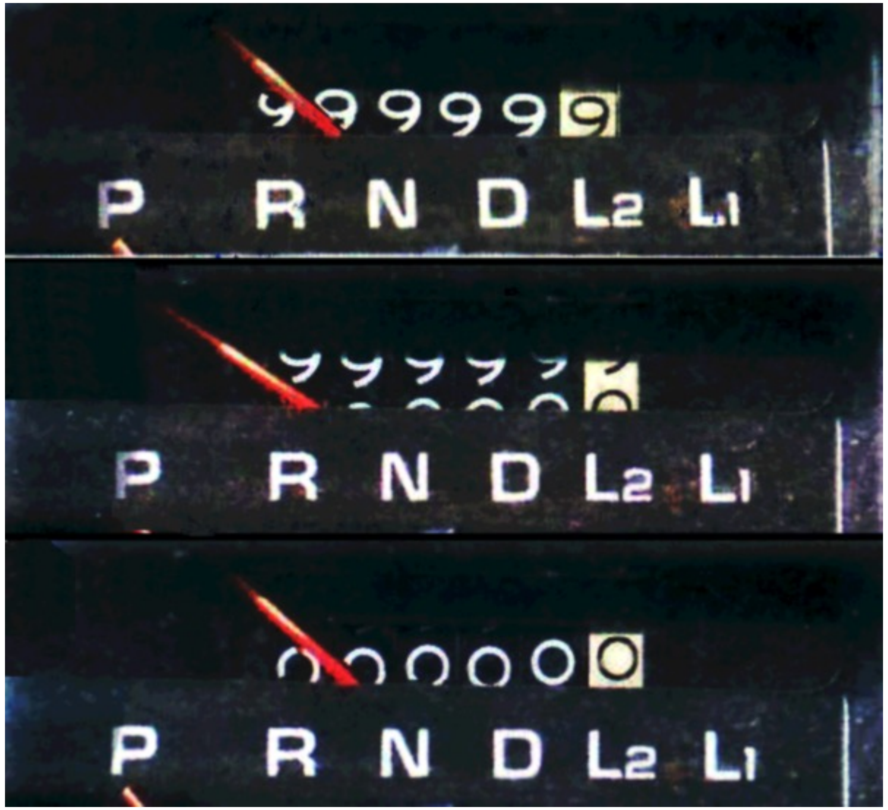
- Underflow
  - Goes below the range of a data type

0	0	0	0	0	0	0	0

-1 ➡

1	1	1	1	1	1	1	1

0 - 1 = 255



Source: https://en.wikipedia.org/wiki/Integer_overflow

Integer Overflow and Underflow

Try overflow.c

- What are the results of the following codes:

```
unsigned char uc = 255;
uc++;
printf("%u\n", uc);

uc = 0;
uc--;
printf("%u\n", uc);

char sc = 127;
sc++;
printf("%d\n", sc);

sc = -128;
sc--;
printf("%d\n", sc);
```

Hex Code

- A byte can be broken into two 4-bit nibbles (or half-bytes)
- A nibble can be represented by a hexadecimal digit (0 ~ F)

$\underbrace{0\,1\,0\,1}_{5}\,\underbrace{1\,1\,1\,1}_{F}$

`unsigned char u = 01011111₂;` → `unsigned char u = 0x5F;`

In most programming languages, there is no such way of expressing binary numbers as it is

Hexadecimal notation is the standard way for expressing binary numbers

- Hex Editor

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  42 4D 7C 00 00 00 00 00 00 00 1A 00 00 00 0C 00  BM|.............
00000010  00 00 04 00 04 00 01 00 18 00 00 00 FF FF FF FF  ............ÿÿÿÿ
00000020  00 00 FF FF FF FF FF FF FF 00 00 00 FF FF FF 00  ..ÿÿÿÿÿÿÿ...ÿÿÿ.
00000030  00 00 FF 00 00 FF FF FF FF 00 00 FF FF FF FF FF  ..ÿ..ÿÿÿÿ..ÿÿÿÿÿ
00000040  FF 00 00 00 FF FF FF 00 00 00 00 00              ÿ...ÿÿÿ.....
```

Summary

- Integer Data Types
- Signed vs Unsigned
- Byte Ordering