信息科学与技术学院

School of Information Science and Technology

# CS 110
# Computer Architecture
## Everything is a Number


**Instructors:**

**Siting Liu & Chundong Wang**


Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html


**School of Information Science and Technology (SIST)**

**ShanghaiTech University**


2023/2/6

# Course Info

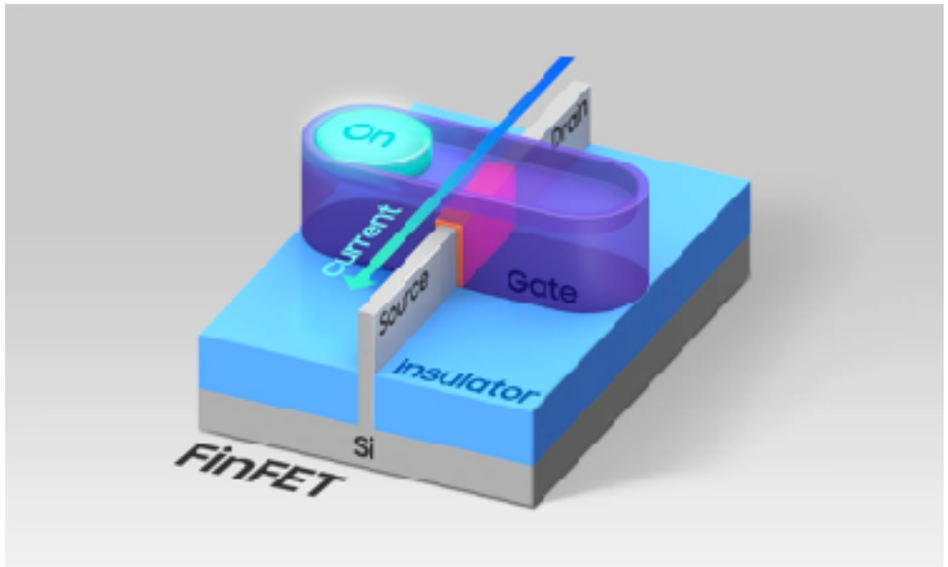
- **HW1 is available, Due Feb. 16th!**
- **Team (Lab & project) partners are required to be within the same lab session! Decide before Feb. 11th!**
- **Acknowledgement**: **UC Berkeley's CS61C: https://cs61c.org/**; 国科大一生一芯: https://ysyx.oscc.cc/
- https://piazza.com/shanghaitech.edu.cn/spring2023/cs110 (access code: **uutib6ruvql**)
- Textbooks: Average 15 pages of reading/week
  - Patterson & Hennessey, Computer Organization and Design **RISC-V edition**!
  - Kernighan & Ritchie, The C Programming Language, 2nd Edition
  - RTFM: C & RISC-V
- Materials this year are similar to previous years, but there might be differences!
- https://robotics.shanghaitech.edu.cn/courses/ca/22s/

# Outline

- Binary system
- Everything is a number
- Signed and Unsigned integers
- Tow's-complement representation

# Binary System

- 0 and 1
- Decided by the characteristic of semiconductor devices (bi-stable states)
- Resilient to noise (threshold)
  - Two branches of math theory
  - Can do logic and arithmetic
- Analogy to decimal (to represent values)
- Positionally weighted coding
- Binary-decimal conversion
- Extend to Hexadecimal (Base 16)/Octal (Base 8)



# Arithmetic

$$\begin{array}{rr} & 0011 \\ + & 0010 \\ \hline & 0101 \end{array}$$

$$\begin{array}{rr} & 1100 \\ - & 1010 \\ \hline & 0010 \end{array}$$

$$\begin{array}{rr} & 1101 \\ \times & 1110 \\ \hline & 1101 \\ & 1101 \\ & 1101 \\ \hline & 10110110 \end{array}$$

$$\begin{array}{r} 1101 \\ 110 \overline{)\,10101111} \\ 110 \\ \hline 1001 \\ 110 \\ \hline 111 \\ 110 \\ \hline 111 \\ 110 \\ \hline 1 \end{array}$$

# Everything is a Number

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 4-bit nibbles (rarely used), 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, ...

字节 半字 (半精) 字 (单精) 双字 (双精)

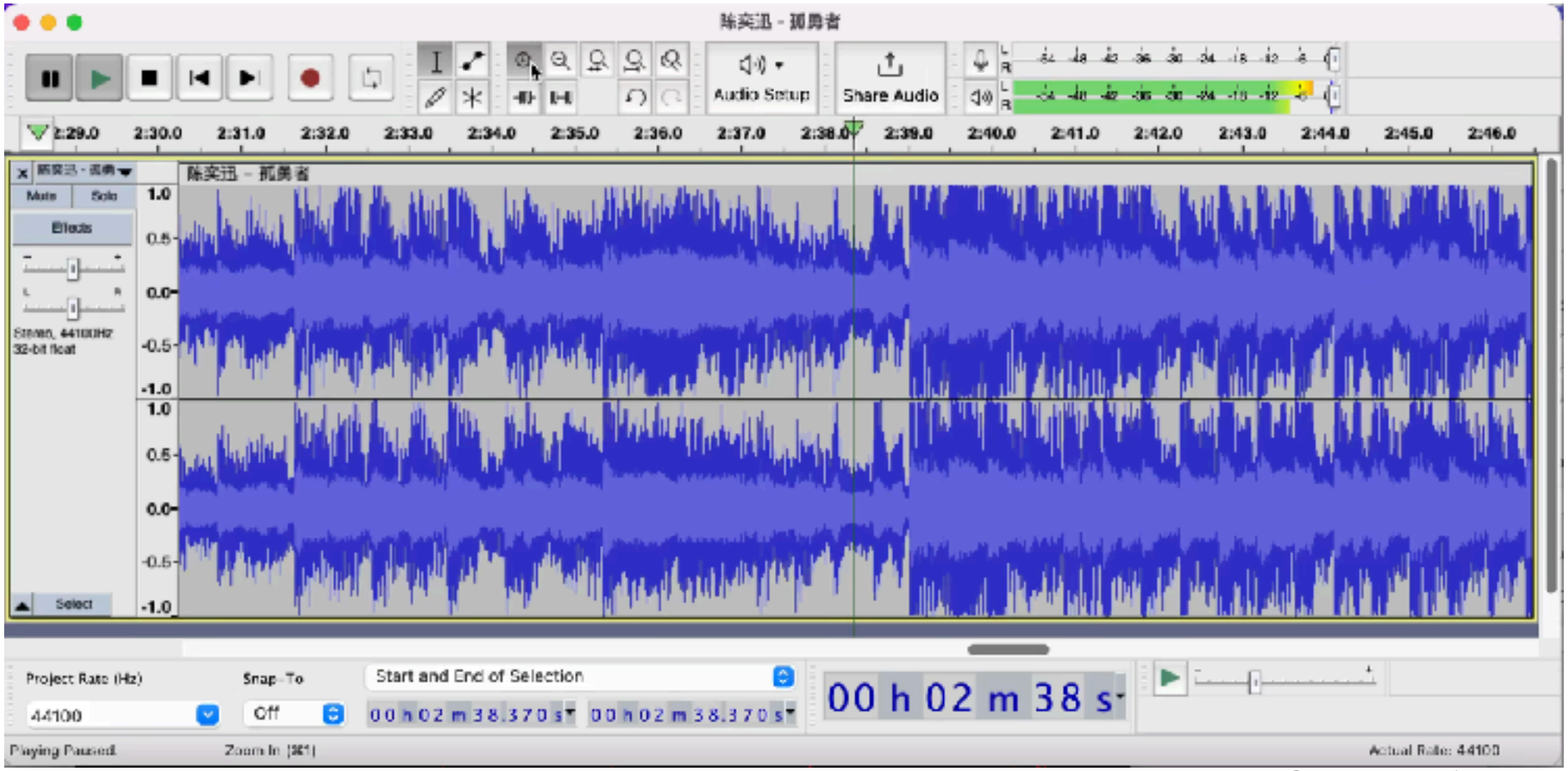
# Everything is a Number

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 4-bit nibbles (rarely used), 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, ...
- Integer and floating-point operations can lead to results too big/small to store within their representations: overflow 溢出
- To avoid overflow, use more bits or extend the range by interpreting a number differently

underflow
精度不足

# Everything is a Number

Soundtrack sampled at 44.1 kHz

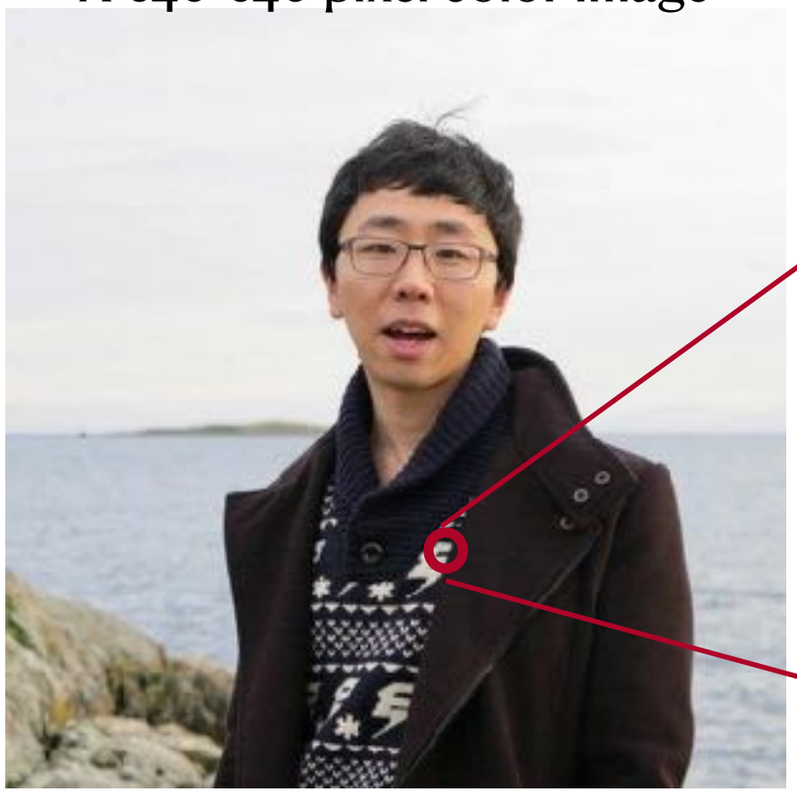



from Audacity

# Everything is a Number

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 4-bit nibbles (rarely used), 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, ...

A 640*640 pixel color image



Around 60 KB on disk

https://www.graphicsmill.com/blog/2014/11/06/Compression-ratio-for-different-JPEG-quality-values

HEX: #c6c3ba
RGB(198, 195, 186)

A 20*20 part of the color image

HEX: #292023
RGB(41, 32, 35)

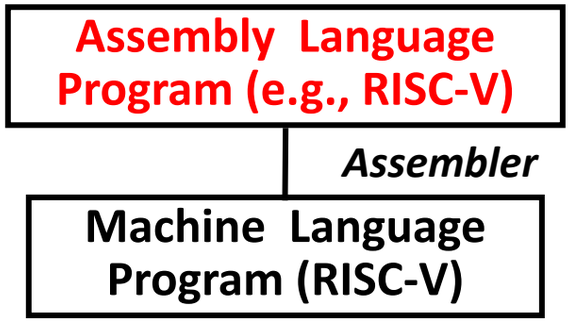
HEX: #d5d2c9
RGB(213, 210, 201)

# Everything is a Number

- Inside computers, everything is a number (but not necessary the value)
- But numbers usually stored with a fixed size
  - 4-bit nibbles (rarely used), 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, ...
- Identity, bank account, profile, ...
  - ID number, DoB (date of birth), criminal record, mobile, etc.
  - Bank account numbers, balance, loan, transaction records, etc.
  - Game account, coins, equipments, ...

# Everything is a Number

- Inside computers, everything is a number (but not necessary the value)
- But numbers usually stored with a fixed size
  - 4-bit nibbles (rarely used), 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, …
- Instructions: e.g., move direction: forward, backward, left, right; use $(00)_2$, $(01)_2$, $(10)_2$ and $(11)_2$




Stored program

```
lw   t0, 0(s2)
lw   t1, 4(s2)
sw   t1, 0(s2)
sw   t0, 4(s2)
```

Anything can be represented as a *number*, i.e., data or instructions

**ISA, defined by human, decides the meaning**

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

- It is how you interpret the numbers decides the meaning

# Signed and Unsigned Integers

- Commonly used in computers to represent integers
- C, C++ have signed integers, e.g., 7, -255:
  - int x, y, z;
- C, C++ also have unsigned integers, e.g. for addresses
- Unsigned integers use their values to represent numbers directly
- Unsigned integers in 32 bit word represent
  0 to $2^{32}$-1 (4,294,967,295) (4 Gibi)

# Unsigned Integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$

… …

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 2{,}147{,}483{,}648_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 2{,}147{,}483{,}649_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2{,}147{,}483{,}650_{ten}$

… …

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

$$(a_n a_{n-1} \ldots a_1 a_0)_2 = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

# Signed Integers

原码

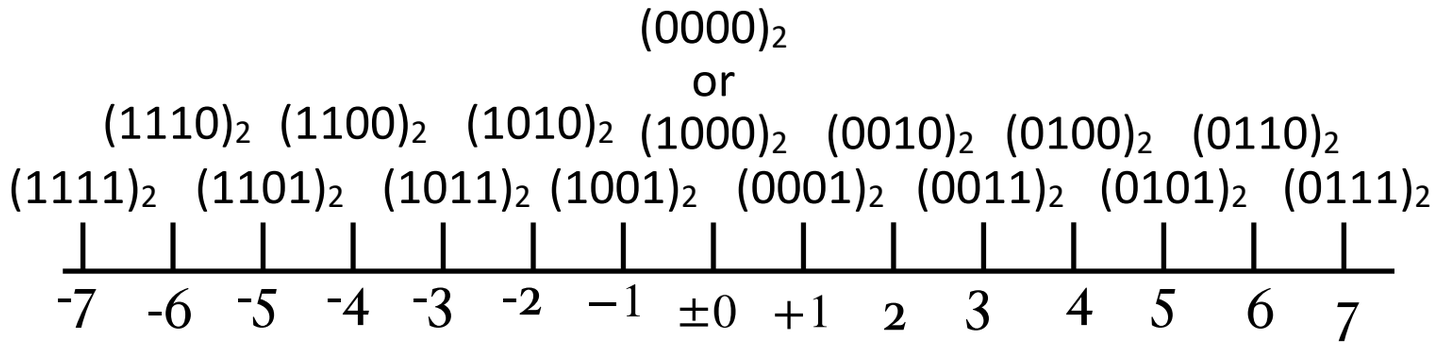
- A straight-forward method: add a sign bit (sign-magnitude)
- Most-significant bit (MSB, leftmost) is the sign bit, 0 means positive, 1 means negative; the other bits remain unchanged

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = 3_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = -3_{ten}$

Sign bit

- Range:
  - Positive: 0 ~ $2^{(n-1)}-1$
  - Negative: -0 ~ -($2^{(n-1)}-1$)
  - Arithmetically unfriendly



# Signed Integers

## One's- & Two's-Complement Representation

- One's-complement representation 反码

  - Positive numbers, stay unchanged; Negative numbers, toggle all bits

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = 3_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two} = -3_{ten}$

Sign bit

除符号位均取反


- Range:

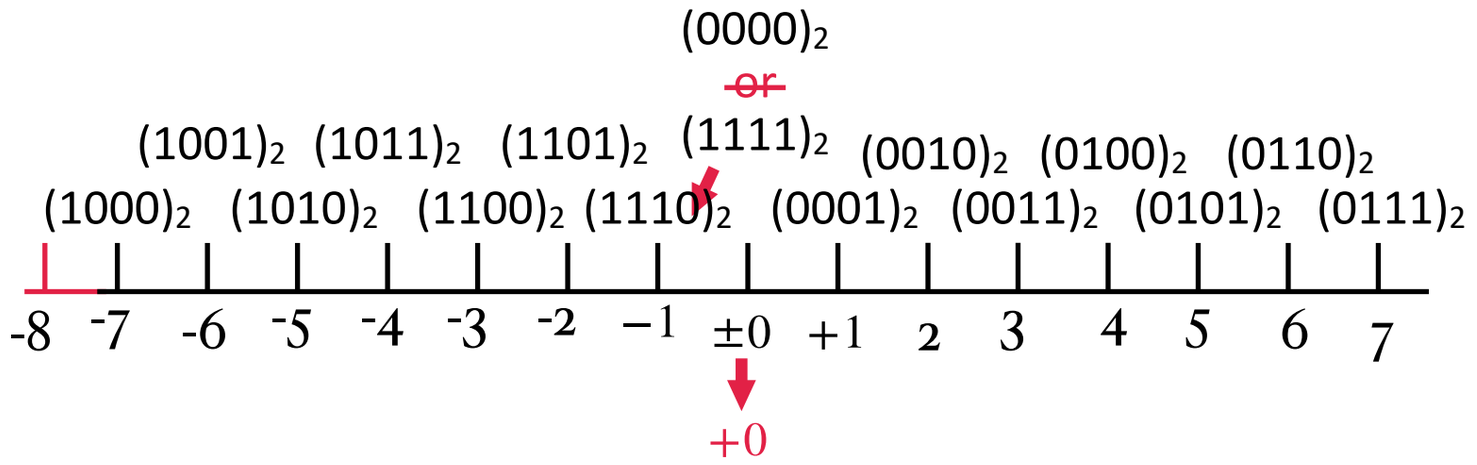
  - Positive: 0 ~ $2^{(n-1)}-1$
  - Negative: -0 ~ $-(2^{(n-1)}-1)$
  - Arithmetically unfriendly

$(-A)_{actual} = 2^n - 1 - A$

n：位数－1（最高位）

# Two's-Complement Representation (Signed Integer)

n：位数－1
（最高位）




$(-A)_{actual} = 2^n - A$

- Two's-complement representation: 补码 = 反码 + 1
- Positive numbers, stay unchanged; Negative numbers, apply two's complement (for an n-bit number A, complement to $2^n$ is $2^n$-A, or toggling all bits and adding 1)

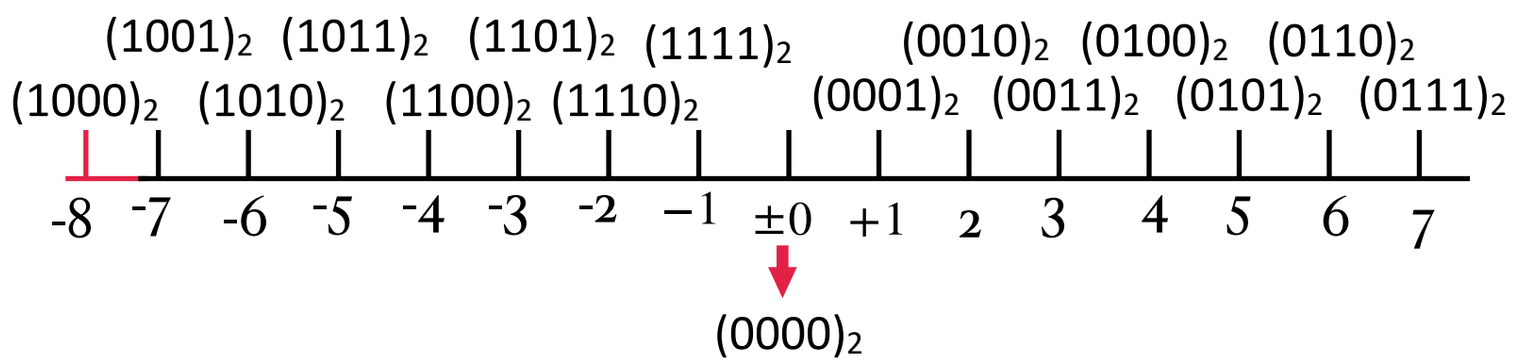
正数不变
负数：取补码

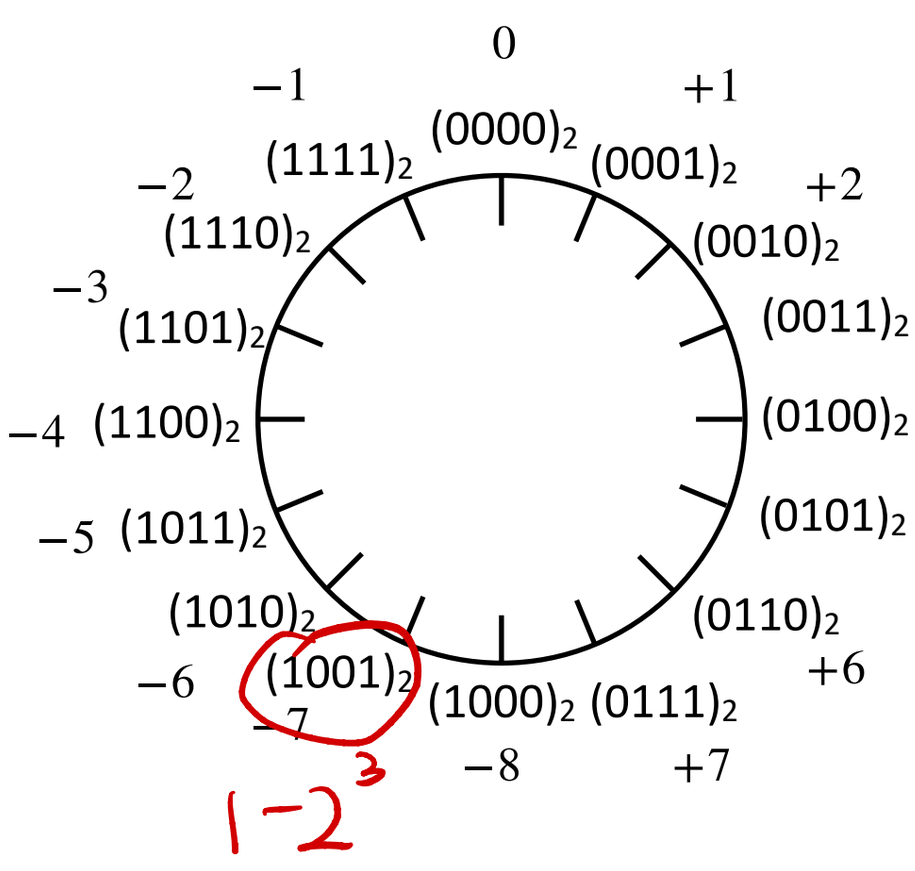
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two} = 3_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -3_{ten}$

Sign bit

# Two's-Complement Representation (Signed Integer)

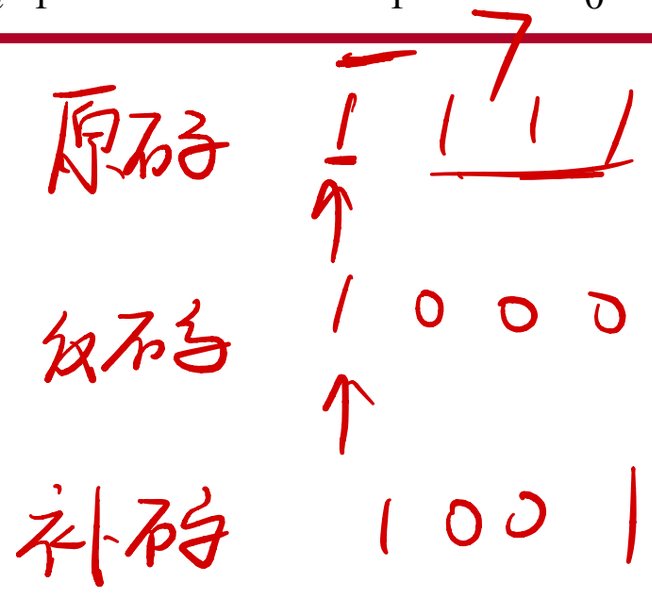



- 2'complement number $(a_n a_{n-1} \ldots a_1 a_0)_2$ represents

$$(a_n a_{n-1} \ldots a_1 a_0)_2 = -a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

- Sign extension
- Arithmetics



# Two's-Complement Arithmetic (Addition & Subtraction)

(-2) 1010 取反→ 1101 补码→ 1110

| | | |
|---|---|---|
| 3 0011 | 3 0011 | -3 1101 |
| +2 0010 | + (-2) 1110 | + (-2) 1110 |
| +5 0101 | +1 [1] 0001 | [1] 1011 |
| | | -5 |

$3-2^3=-5$

| | |
|---|---|
| 7 0111 | -8 1000 |
| +1 0001 | + (-1) 1111 |
| 1000 | 7 [1] 0111 |
| -8 | overflow |

- Overflow check!

出现的码全是补码

# Comparison

| Sign-magnitude | One's-complement | Two's-complement |
|---|---|---|
| 5 0101 | 5 0101 | 5 0101 |
| + (-3) 1011 | + (-3) 1100 | + (-3) 1101 |
| 0000 | 0001 | 0010 |
| ✗ | ✗ | ✓ |

# Two's-Complement Representation (Signed Integer)

- Two's complement treats 0 as positive, so 32-bit word represents $2^{32}$ integers from
  $-2^{31}$ (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - **Every computer uses two's complement today**
- Most-significant bit (MSB) (leftmost) is the sign bit, since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant (MSB), bit 0 is least significant (LSB)

# Two's-Complement Representation (Signed Integer)

- Two's complement treats 0 as positive, so 32-bit word represents $2^{32}$ integers from
  $-2^{31}$ (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - **Every computer uses two's complement today**
- Most-significant bit (MSB) (leftmost) is the sign bit, since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant (MSB), bit 0 is least significant (LSB)

信息科学与技术学院
School of Information Science and Technology

# CS 110
# Computer Architecture
# Intro to C I

**Instructors:**

**Siting Liu & Chundong Wang**

Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2023/index.html

**School of Information Science and Technology (SIST)**

**ShanghaiTech University**

2023/2/6

# Introduction to C
# “The Universal Assembly Language”

# Intro to C

- C *is not a "very high-level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*
- Enabled first operating system not written in assembly language: UNIX - A portable OS!

# Intro to C

- *Why C?: we can write programs that allow us to exploit underlying features of the architecture – memory management, special instructions, parallelism*
- C and derivatives (C++/Obj-C/C#) still one of the most popular application programming languages after >40 years!

# Disclaimer

- You will not learn how to fully code in C in these lectures! You'll still need your C reference for this course
  - K&R C is a recommendation
    - Check online for more sources
      - ANSI/ISO C standard manual (RTFM) (https://web.archive.org/web/20200909074736if_/https://www.pdf-archive.com/2014/10/02/ansi-iso-9899-1990-1/ansi-iso-9899-1990-1.pdf; http://web.archive.org/web/20030222051144/http://home.earthlink.net/~bobbitts/c89.txt)
- Key C concepts: Pointers, Arrays, Implications for Memory management
- We will use ANSI C89 – original "old school" C
  - Because it is closest to Assembly

# How C program works?

**High Level Language Program (e.g., C)**

*Compiler*

**Assembly Language Program (e.g., RISC-V)**

*Assembler*

**Machine Language Program (RISC-V)**

***Machine Interpretation***

**Hardware Architecture Description (e.g., block diagrams)**

***Architecture Implementation***

**Logic Circuit Description (Circuit Schematic Diagrams)**

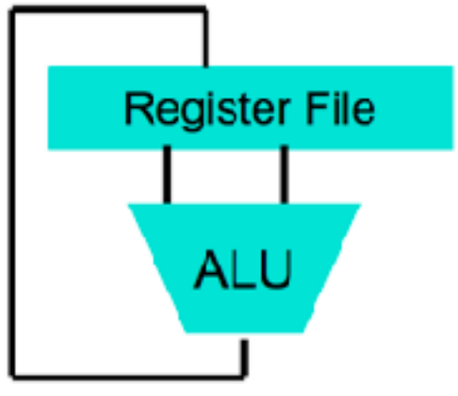
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

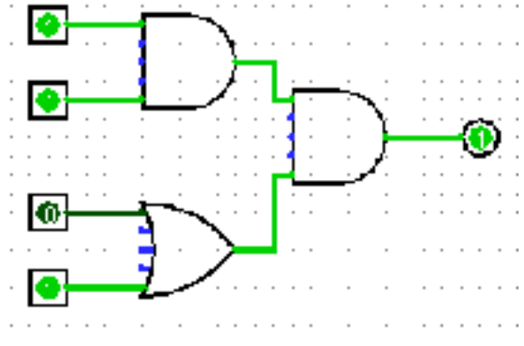
*We are here!*

```
lw    t0, 0(s2)
lw    t1, 4(s2)
sw    t1, 0(s2)
sw    t0, 4(s2)
```

Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```



# Compilation: Overview

- C compilers map C programs into architecture (OS & ISA)-specific machine code (strings of 1s and 0s)
  - Unlike Java, which converts to architecture-independent bytecode
  - Unlike Python environments, which interpret the code
  - These differ mainly in exactly when your program is converted to low-level machine instructions ("levels of interpretation")
  - For C, generally a two part process of compiling .c files to .o files, then linking the .o files into executables;
  - Assembling is also done (but is hidden, i.e., done automatically, by default); we'll talk about that later

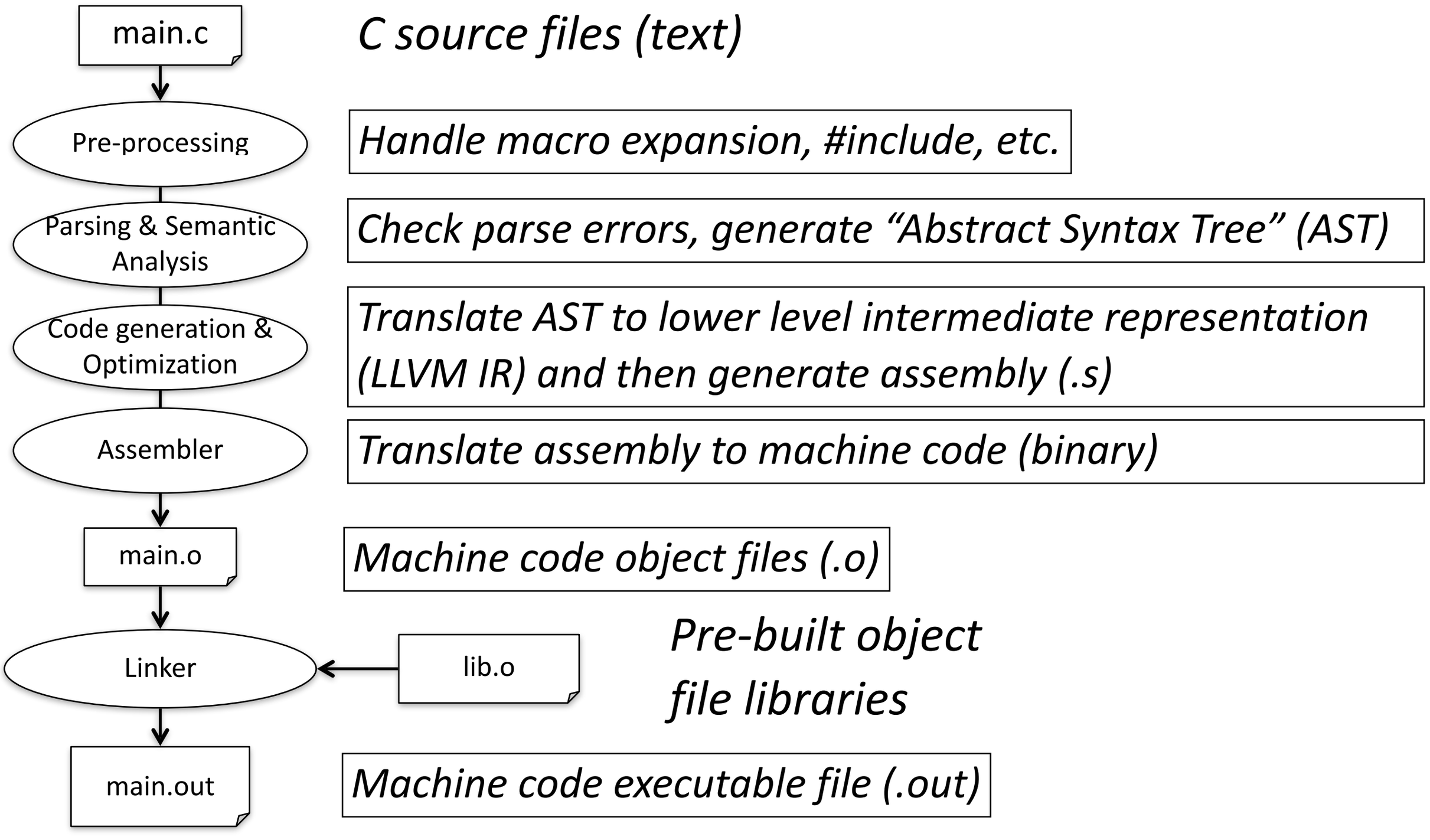
# Compilation: Advantages

- Excellent run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- Reasonable compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled

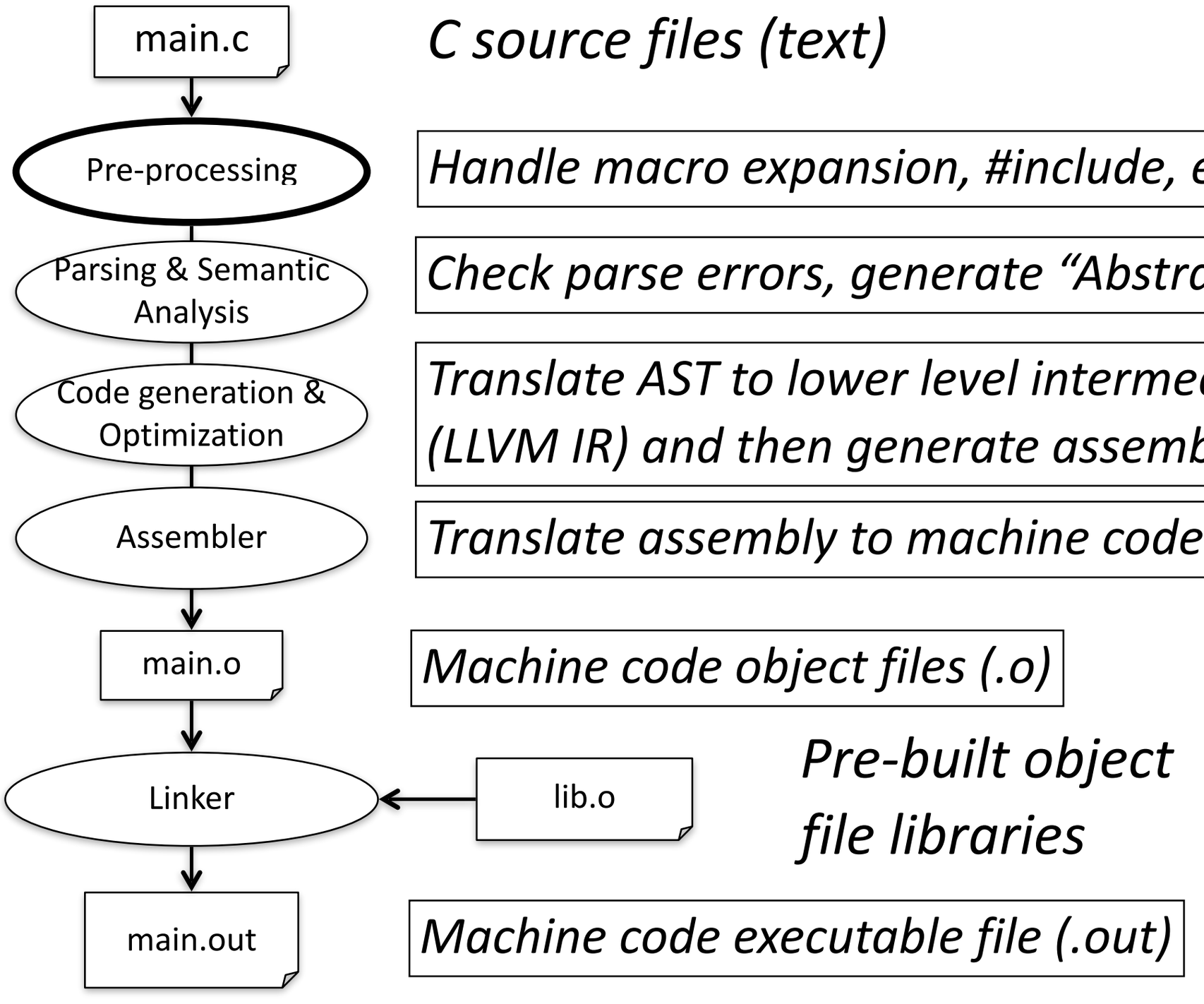
# Compilation

- Mainstream C compiler in Linux:
  - GNU Complier Collection (gcc, not only for C family);
  - clang/LLVM (for C language family)
  - In terminal/command line tool/shell, “man clang/gcc”

# C Compilation Simplified Overview



# C Compilation Simplified Overview




*C source files (text)*

*Handle macro expansion, #include, etc.*

*Check parse errors, generate "Abstract Syntax Tree" (AST)*

*Translate AST to lower level intermediate representation (LLVM IR) and then generate assembly (.s)*

*Translate assembly to machine code (binary)*

*Machine code object files (.o)*

*Pre-built object file libraries*

*Machine code executable file (.out)*

# C Pre-Processing (CPP)

- C source files first pass through CPP, before compiler sees code (mainly text editing)
- CPP replaces comments with a single space
- CPP commands begin with “#”
- #include “file.h” /* Inserts file.h into output */
- #include <stdio.h> /* Looks for file in standard location */
- #define M_PI (3.14159) /* Define constant */
- #if/#endif /* Conditional inclusion of text */
- Use -save-temps (-E) option to gcc to see result of preprocessing

# Consts. and Enums. in C

- Constant is assigned a typed value once in the declaration; value can't change during entire execution of program

```
const float golden_ratio = 1.618;
const int days_in_week = 7;
```

- You can have a constant version of any of the standard C variable types
- Enums: a group of related integer constants. Ex:

```
enum cardsuit {CLUBS,DIAMONDS,HEARTS,SPADES};
enum color {RED, GREEN, BLUE};
```

Compare "`#define PI 3.14`" and "`const float pi=3.14`" – which is true?

A: Constants "PI" and "pi" have same type

B: Can assign to "PI" but not "pi"

C: Code runs at about the same speed using "PI" or "pi"

**D: "pi" takes more memory space than "PI"**

**E: Both behave the same in all situations**

Compare "`#define PI 3.14`" and "`const float pi=3.14`" – which is true?

A: Constants "PI" and "pi" have same type

B: Can assign to "PI" but not "pi"

C: Code runs at about the same speed using "PI" or "pi"

D: "pi" takes more memory space than "PI"

E: Both behave the same in all situations