C 语言

第一讲、概览

张晓平

武汉大学数学与统计学院

2017 年 2 月 21 日

1. 计算机的硬件介绍

2. 计算机的基本软件组成

3. 数制

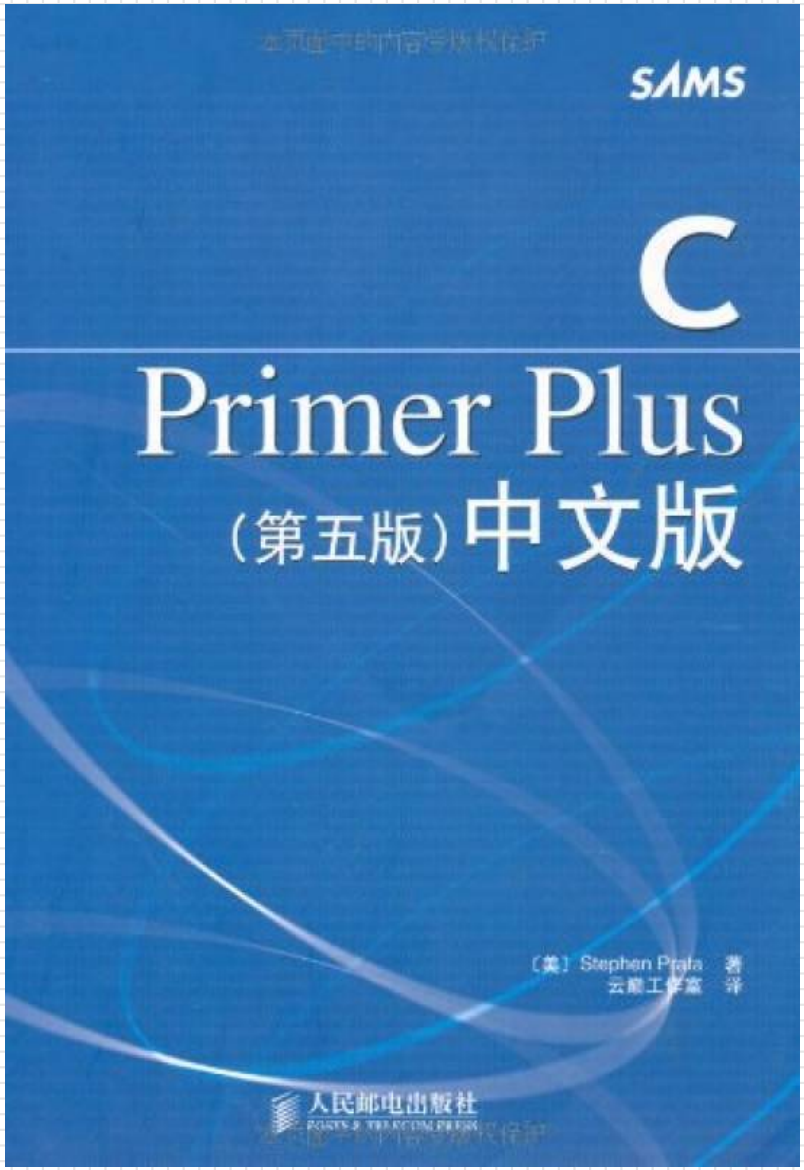
4. 汇编语言简介

5. C 语言简介

6. 使用 C 语言的七个步骤

7. 编译 C 程序的工作原理


```
#include<stdio.h>
int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```


SAMS
C
Primer Plus
(第五版)中文版
[美] Stephen Prata 著
云巅工作室 译
人民邮电出版社

1. 计算机的硬件介绍

硬件组成

- 中央处理器（Central Processing Unit, CPU）
- 存储器

  内存储器，即内存（Memory）

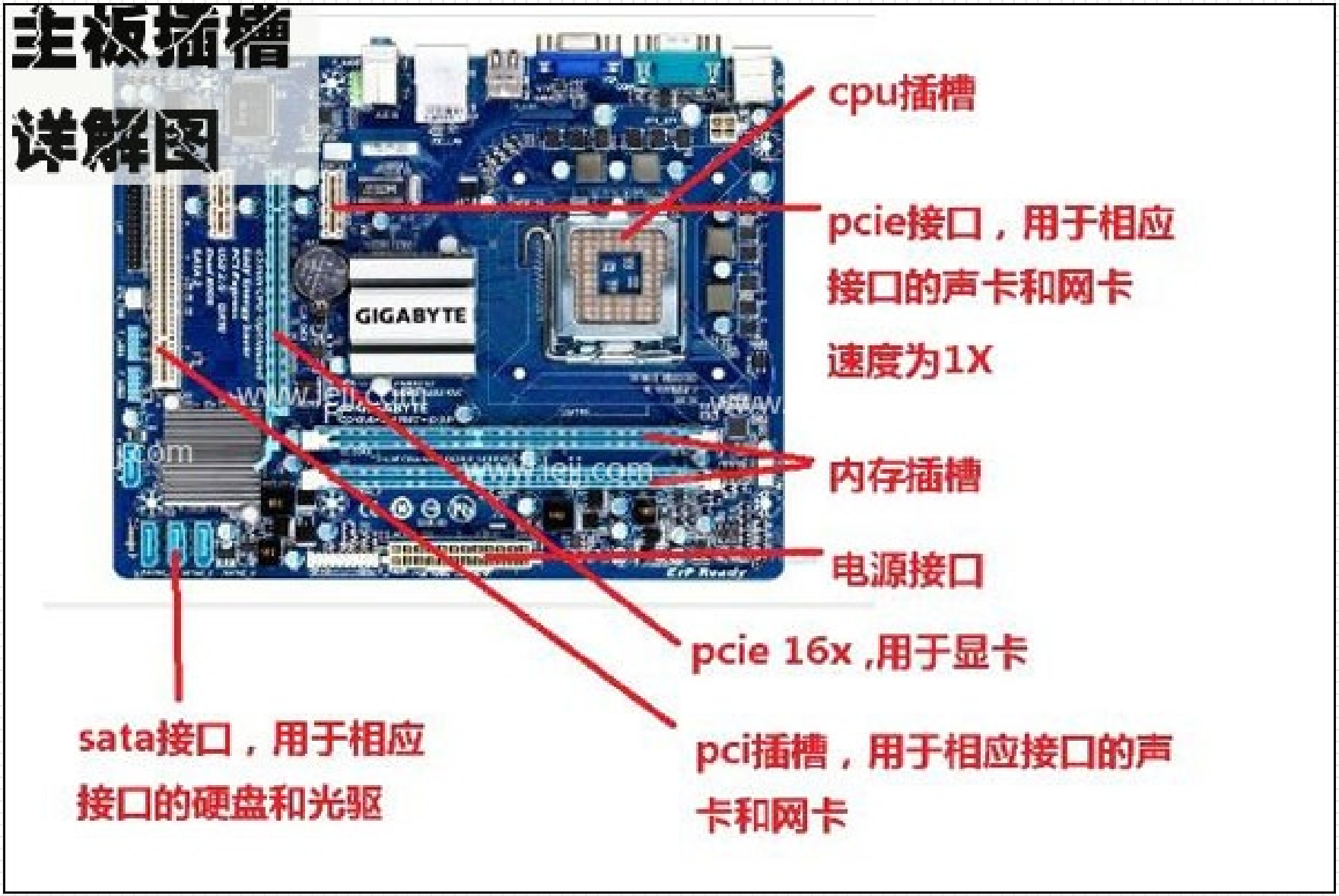
  外存储器：硬盘（Hard Disk）、U 盘
- 输入设备（Input Device）：键盘、鼠标、摄像头、扫描仪等
- 输出设备（Output Device）：显示器、打印机、音响等

主板

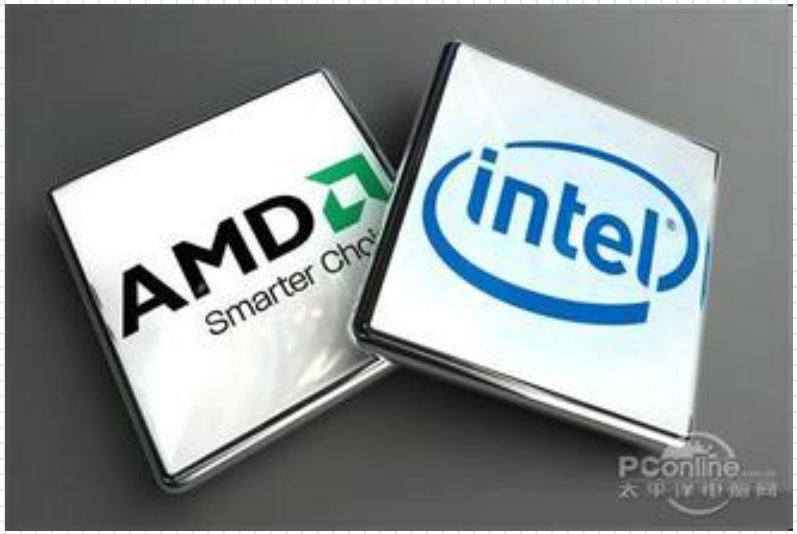
CPU、内存、硬盘、显卡和声卡等都必须安装在主板上才能运行。

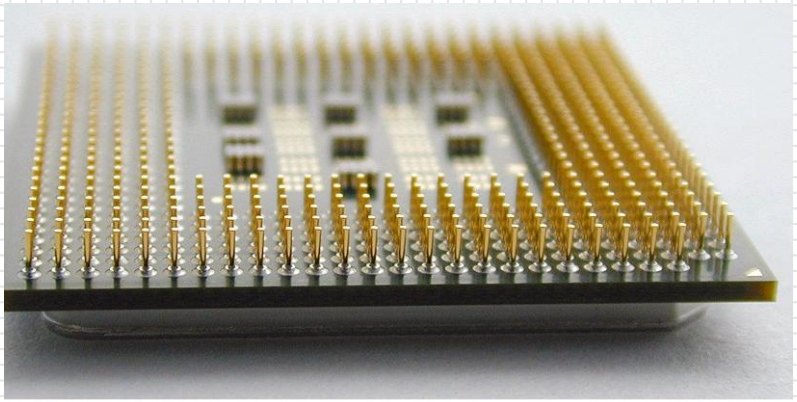
- 主板是电脑主机中的最大承载体，若把电脑比喻为一个人，主板就是人的躯干，它是把电脑各个部件连接起来的纽带。
- 主板同时还提供各种外接设备的接口，如 USB 接口、键盘接口、鼠标接口、网线接口，将鼠标、键盘、扫描仪、打印机等设备连接进来。

主板


主板插槽
详解图
cpu插槽
pcie接口，用于相应
接口的声卡和网卡
速度为1X
内存插槽
电源接口
pcie 16x ,用于显卡
sata接口，用于相应
接口的硬盘和光驱
pci插槽，用于相应接口的声
卡和网卡

CPU





CPU

CPU 是计算机的大脑，计算机处理数据的能力主要取决于 CPU。它主要执行三种基本的操作:

- 读出数据：一般从内存读取数据。
- 处理数据：通过算术逻辑单元对数据进行处理。
- 写入数据：将数据写入内存。

CPU

CPU 由运算器、控制器、寄存器和高速缓冲存储器组成。

CPU

- 运算器，又称算术逻辑单元：负责对数据进行加工处理，包括
  - 算术运算：加、减、乘、除等
  - 逻辑运算：与、或、非、异或、比较等。
- 控制器：主要是负责对指令译码，并且发出为完成每条指令所要执行的各个操作的控制信号。

CPU 的组成

- 寄存器，包括通用寄存器、专用寄存器和控制寄存器：来保存指令执行过程中临时存放的寄存器操作数和中间（或最终）的操作结果。
- 高速缓冲存储器（Cache）

  在 CPU 芯片内，是一个读写速度比内存更快的存储器。

内存

内存

- 内存是 CPU 能直接寻址的存储空间，所有程序的运行都是在内存中进行的。
- 只要计算机在运行中，CPU 就会把需要运算的数据调到内存中进行运算，当运算完成后 CPU 再将结果传送出来。
- 其作用是用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器交换的数据。

内存

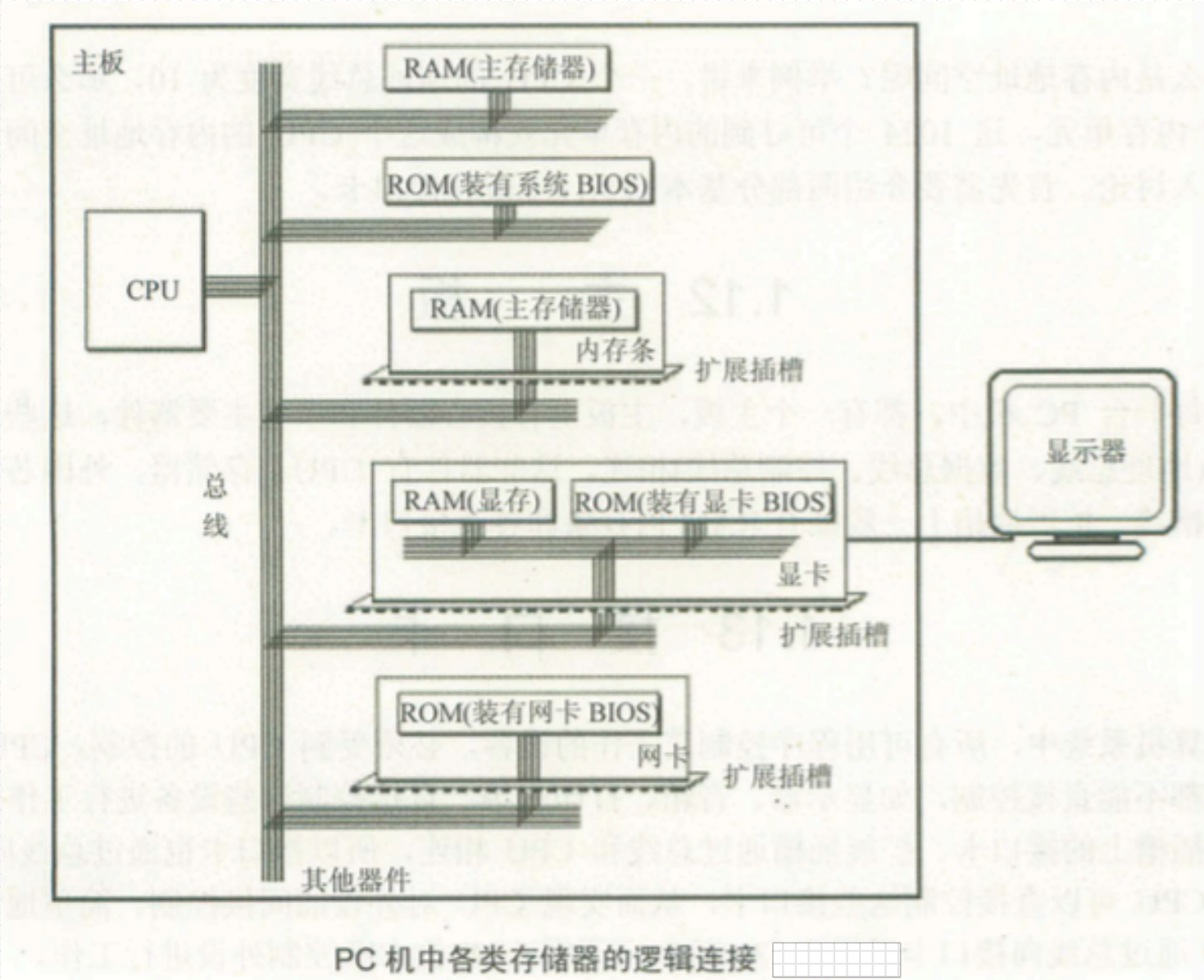
- 只读存储器（Read Only Memory, ROM）

  只能读取，不能写入，即使断电，存于其中的数据也不会丢失，一般用于存放计算机的基本程序和数据。

- 随机存储器（Random Access Memory, RAM）

  既可读取，也可写入，断电时，存于其中的数据就会丢失。内存条就是将 RAM 集成块集中在一起的一小块电路板。

各类存储器的逻辑连接



PC 机中各类存储器的逻辑连接

2. 计算机的基本软件组成

计算机的基本软件组成

软件是组成计算机系统的重要部分，分为系统软件和应用软件两大类。

计算机的基本软件组成

- 系统软件是由计算机生产厂商为使用该计算机而提供的基本软件。如操作系统、文字处理程序、计算机语言处理程序、数据库管理程序等。
- 应用软件是指用户为了自己的业务应用而使用系统开发出来的用户软件。如音频视频播放器、QQ、微信等。

系统软件依赖于机器，而应用软件则更接近用户业务。

操作系统（OS）

- 操作系统是最基本也是最重要的系统软件。
- 它负责管理计算机系统的各种硬件资源，如 CPU、内存空间、磁盘空间、外部设备等，并且负责解释用户对机器的管理命令，使它转换为机器实际的操作。
- 常见的操作系统：DOS、WINDOWS、UNIX(LINUX)、OS X 等。

计算机语言处理程序

计算机语言分为机器语言、汇编语言和高级语言。

- 机器语言：机器能直接认识的语言，是由“1”和“0”组成的一组代码指令。
- 汇编语言：实际是由一组与机器语言指令一一对应的符号指令和简单语法组成的。
- 高级语言：比较接近日常用语，对机器依赖性低，即适用于各种机器的计算机语言。如 Basic、Visual Basic、Fortran、C/C++、Java、Python 等。

计算机语言处理程序

将高级语言翻译为机器语言，有两种方式，一种叫“编译”，一种叫“解释”。

- 编译程序把高级语言程序作为一个整体进行处理，编译后与子程序库连接，形成一个完整的可执行程序。其缺点是编译、链接比较费时，但可执行程序运行速度很快。Fortran、C/C++ 语言等采用这种方式。
- 解释程序则对高级语言程序逐句解释执行。其特点是程序设计的灵活性大，但运行效率较低。Basic、Python、Matlab 语言属于解释型。

3. 数制

数制

定义 数制也称计数制，是用一组固定的符号和统一的规则来表示数值的方法。人们通常采用的数制有十进制、二进制、八进制和十六进制。

在一种数制中，只能使用一组固定的数字符号来表示数目的大小。具体使用多少个数字符号来表示数目的大小，就称为该数制的基数。

数制

- 十进制（Decimal）

  基数是 10，有 10 个数字符号，即 0，l，2，3，4，5，6，7，8，9。其中最大数码是基数减 1，即 9，最小数码是 0。

数制

- 二进制 (Binary)

  基数是 2，只有两个数字符号，即 0 和 1。这就是说，如果在给定的数中，除 0 和 1 外还有其它数，例如 1012，它就决不会是一个二进制数。

数制

- 八进制 (Octal)

  基数是 8，它有 8 个数字符号，即 0，l，2，3，4，5，6，7。最大的也是基数减 1，即 7，最小的是 0。

数制

- 十六进制（Hexadecimal）

  基数是 16，它有 16 个数字符号，除了十进制中的 10 个数可用外，还使用了 6 个英文字母。16 个数字符号依次是

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

  其中 A~F 分别代表十进制数的 10~15，最大的数字也是基数减 1。

数制

既然有不同的进制，那么在给出一个数时，需指明是什么数制里的数。

如：$(1010)_2$，$(1010)_8$，$(1010)_{10}$，$(1010)_{16}$ 所代表的数值就不同。

除了用下标表示外，还可用后缀字母来表示数制。

如：1A4EH, FEEDH, BADH 与 $(1A4E)_{16}$，$(FEED)_{16}$，$(BAD)_{16}$ 的意义相同。

数制

二进制	十进制	八进制	十六进制
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	10	8
1001	9	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F
10000	16	20	10

进制

在数制中，N 进制必须是逢 N 进一。

- 十进制数的特点是逢十进一。

$$(1010)_{10} = 1 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$$

- 二进制数的特点是逢二进一。

$$(1010)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (10)_{10}$$

- 八进制数的特点是逢八进一。

$$(1010)_8 = 1 \times 8^3 + 0 \times 8^2 + 1 \times 8^1 + 0 \times 8^0 = (520)_{10}$$

- 十六进制数的特点是逢十六进一。

$$(\mathrm{BAD})_{16} = 11 \times 16^2 + 10 \times 16^1 + 13 \times 16^0 = (2989)_{10}$$

二进制数的加法法则

$$0+0=0$$

$$0+1=1$$

$$1+0=1$$

$$1+1=10$$

$$1+1+1=10+1=11$$

$$\begin{array}{rr} & 1011 \\ + & 1010 \\ \hline = & 10101 \end{array}$$

二进制数的减法法则

$0-0=0$

$1-0=1$

$1-1=0$

$0-1=1$ 有借位，借 1 当 $(10)_2$

$0-1-1=0$ 有借位

$1-1-1=1$ 有借位

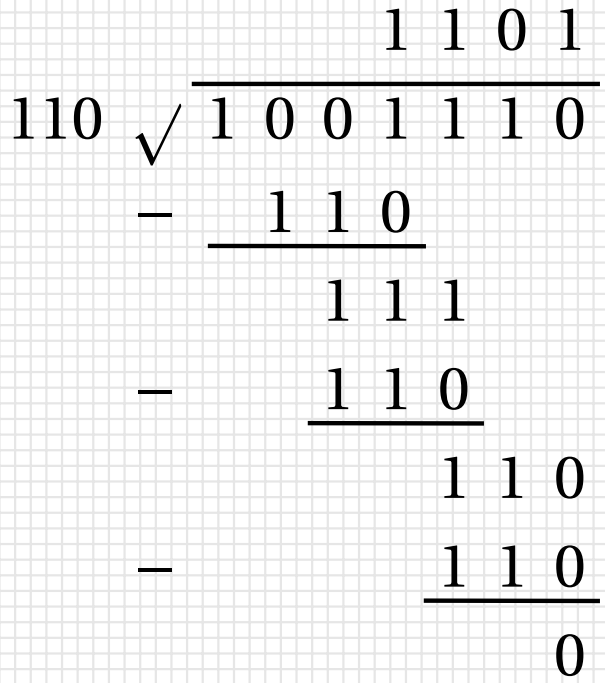
$$\begin{array}{rr} & 1\,1\,0\,0\,0\,0 \\ - & 1\,0\,1\,1\,1 \\ \hline = & 1\,1\,0\,0\,1 \end{array}$$

二进制的乘法法则

$$0\times 0=0,\quad 0\times 1=0$$
$$1\times 0=0,\quad 1\times 1=1$$

$$\begin{array}{rr}
 & 1\ 1\ 1\ 0 \\
\times & 0\ 1\ 1\ 0 \\
\hline
 & 0\ 0\ 0\ 0 \\
 & 1\ 1\ 1\ 0 \\
 & 1\ 1\ 1\ 0 \\
+ & 0\ 0\ 0\ 0 \\
\hline
 & 1\ 0\ 1\ 0\ 1\ 0\ 0
\end{array}$$

二进制的除法法则



十进制数到二进制数的转换

（1）整数部分 除 2 取余，直至商为 0，最后将所得余数按逆序排列。

2	2	3	
2	1	1	1
	2	5	1
	2	2	1
	2	1	0
		0	1

结果为 $(23)_{10} = (10111)_2$.

十进制数到二进制数的转换

（2）小数部分 乘 2 取整数，若小数部分是 5 的倍数，则以最后小数部分为 0 为止，否则以约定的精确度为准，最后将所取整数按顺序排列。

$$
\begin{array}{rrrrrl}
 & 0 & . & 2 & 5 & \\
\times & & & & 2 & \\
\hline
 & 0 & . & 5 & 0 & \text{取整数位0} \\
\times & & & & 2 & \\
\hline
 & 1 & . & 0 & 0 & \text{取整数位1}
\end{array}
$$

结果为 $(0.25)_{10} = (0.01)_2$.

十进制数到二进制数的转换

例　将十进制数 125.24 转换为二进制数（取四位小数）。

十进制数到二进制数的转换

例　将十进制数 125.24 转换为二进制数（取四位小数）。

2 | 125		
2 | 62	1	
2 | 31	0	
2 | 15	1	
2 | 7	1	
2 | 3	1	
2 | 1	1	
0	1	

0.24	
× 2	
0.48	取整数位0
× 2	
0.96	取整数位0
× 2	
1.92	取整数位1
× 2	
1.84	取整数位1

十进制数到二进制数的转换

例　将十进制数 125.24 转换为二进制数（取四位小数）。

2 | 125		
2 | 62	1	
2 | 31	0	
2 | 15	1	
2 | 7	1	
2 | 3	1	
2 | 1	1	
0	1	

0.24 × 2	
0.48	取整数位0
0.48 × 2	
0.96	取整数位0
0.96 × 2	
1.92	取整数位1
1.92 × 2	
1.84	取整数位1

结果为

$$(125.24)_{10} = (1111101.0011)_2.$$

二进制数到十进制数的转换

基本原理：

- 将二进制数从小数点开始，往左从 0 开始对各位进行正序编号，往右序号则分别为 $-1, -2, -3, \cdots$，直到最末位。
- 然后分别将各位上的数乘以 2 的 k 次幂所得的值进行求和，其中 k 的值为各个位所对应的上述编号。

二进制数到十进制数的转换

例　将二进制数 1101.101 转换为十进制数（取四位小数）。

二进制数到十进制数的转换

例　将二进制数 1101.101 转换为十进制数（取四位小数）。

$$
\begin{aligned}
& (1101.101)_2 \\
= \quad & 1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3} \\
= \quad & 8+4+1+0.5+0.125 \\
= \quad & 13.625
\end{aligned}
$$

二进制数到十六进制数的转换

基本原理： 由于十六进制数基数是 2 的四次幂，所以一个二进制转换为十六进制，

- 如果是整数，只要从它的低位到高位每 4 位组成一组，然后将每组二进制数所对应的数用十六进制表示出来。
- 如果有小数部分，则从小数点开始，分别向左右两边按照上述方法进行分组计算。

二进制数到十六进制数的转换

例　将二进制数 11 1010 1111 0001 0111 转换为十六进制数。

二进制数到十六进制数的转换

例　将二进制数 11 1010 1111 0001 0111 转换为十六进制数。

二进制数	11	1010	1111	0001	0111
十六进制数	3	A	F	1	7

结果为

$$(11\ 1010\ 1111\ 0001\ 0111)_2 = (3AF17)_{16}$$

4. 汇编语言简介

机器语言

机器语言是机器指令的集合。机器指令是计算机可以正确执行的命令，它是一组二进制数字。计算机将其转变为一组高低电平，以使电子器件收到驱动，进行运算。

机器语言

8086CPU 完成运算 $s = 768 + 12288 - 1280$ 的机器码如下：

1011 0000 0000 0000 0000 0011

0000 0101 0000 0000 0011 0000

0010 1101 0000 0000 0000 0101

机器语言

8086CPU 完成运算 $s = 768 + 12288 - 1280$ 的机器码如下：

```
1011 0000 0000 0000 0000 0011
0000 0101 0000 0000 0011 0000
0010 1101 0000 0000 0000 0101
```

若将程序错写成以下形式，请指出错误：

```
1011 0000 0000 0000 0000 0011
0000 0101 0000 0000 0011 0000
0001 0110 1000 0000 0000 0101
```

机器语言

书写和阅读机器码程序不是一件简单的工作，需要记住抽象的二进制码。上面只是一个非常简单的小程序，就暴露了机器码的晦涩难懂和不易查找。

早期程序员很快发现了使用机器语言带来的麻烦，于是汇编语言产生了。

汇编语言简介

汇编语言的主体是汇编指令。汇编指令和机器指令的差别在于指令的表示方法上。汇编指令是机器指令便于记忆的书写形式。

汇编语言简介

汇编语言的主体是汇编指令。汇编指令和机器指令的差别在于指令的表示方法上。汇编指令是机器指令便于记忆的书写形式。

操作：寄存器 BX 的内容送到 AX 中

机器指令：1000 1001 1101 1000

汇编指令：mov ax,bx

汇编语言简介

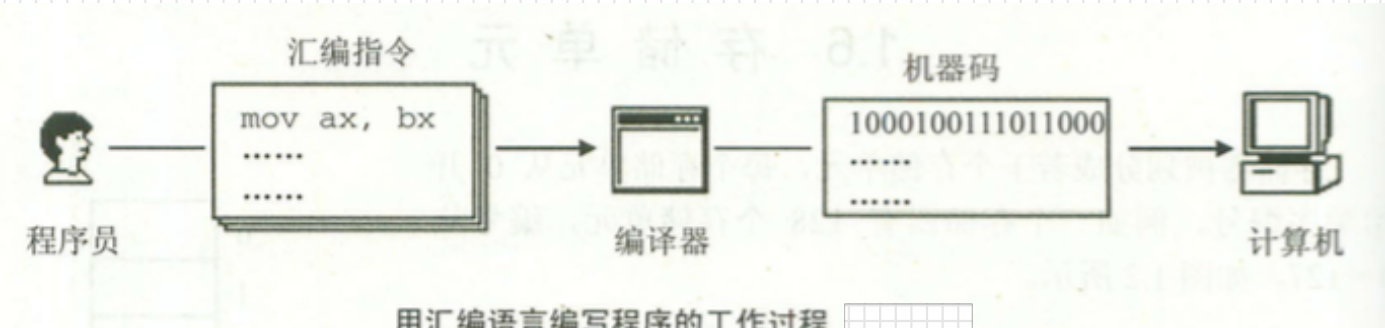
问题　计算机只能读懂机器指令，那么如何让计算机执行汇编指令编写的程序呢？

汇编语言简介

问题　计算机只能读懂机器指令，那么如何让计算机执行汇编指令编写的程序呢？

需要用到一个能将汇编指令转换为机器指令的翻译程序，即编译器。

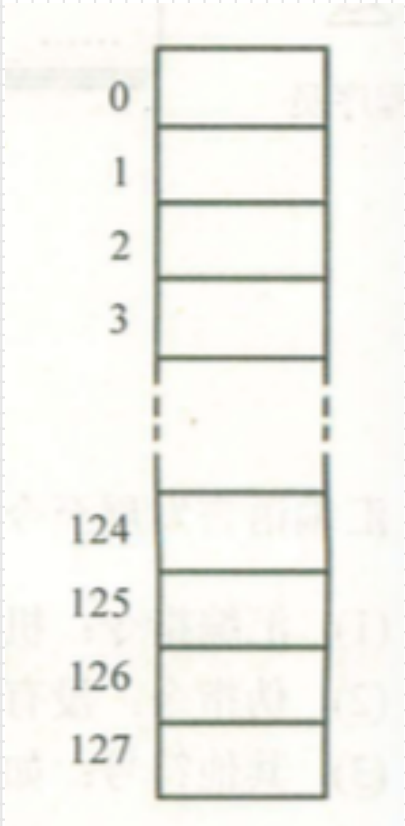
程序员用汇编语言写出源程序，再用汇编编译器将其编译为机器码，由计算机最终执行。



用汇编语言编写程序的工作过程

存储单元

存储器被划分为若干个存储单元，每个存储单元从 0 开始顺序编号。例如，假设一个存储器，编号从 0～127，如下图：



存储单元

计算机的最小信息单位是 Bit，也就是一个二进制位。8 个 Bit 组成一个 Byte，即一个字节。一个存储单元可存储一个字节。若一个存储器有 128 个存储单元，它可以存储 128 个字节。存储器的容量以字节为最小单位来计算。对于拥有 128 个存储单元的存储器，其容量为 128 个字节。

对于大容量的存储器，还用以下单位来计算容量（以下用 B 来代表 Byte）：

$$1KB = 2^{10}B = 1024B, \quad 1MB = 1025KB,$$
$$1GB = 1024KB, \quad 1TB = 1025GB,$$

CPU 对存储器的读写

- CPU 要从内存中读数据，首先要指定单元地址。也就是说要先确定要读哪个单元中的内容。
- 存储器不止一种。CPU 在读写数据时还要指明，对哪一个存储器进行操作，进行哪种操作，是从中读取数据，还是向里面写入数据。

CPU 对存储器的读写

CPU 要想进行数据的读写，必须与外部器件（芯片）进行以下 3 类信息的交互：

- 存储单元的地址（地址信息）；
- 器件的选择，读或写的命令（控制信息）；
- 读或写的数据（数据信息）。

CPU 对存储器的读写

问题　CPU 通过什么将地址、数据和控制信息传给存储器芯片呢？

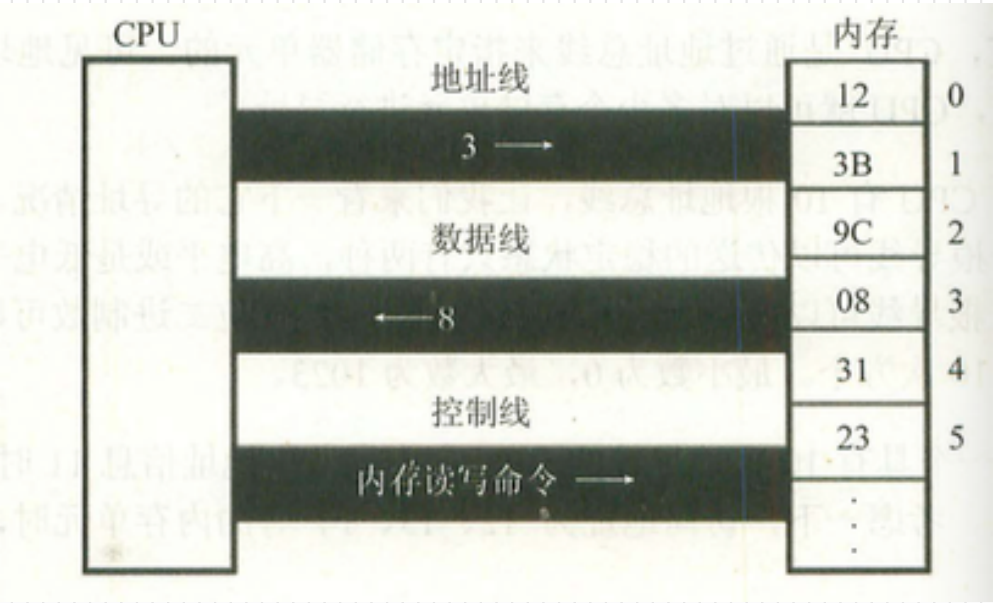
CPU 对存储器的读写

问题　CPU 通过什么将地址、数据和控制信息传给存储器芯片呢？

计算机传输的信息都是电信号，电信号当然要用导线传送。在计算机中专门有连接 CPU 和其它芯片的导线，通常称为总线。根据传送信息的不同，总线从逻辑上分为三类：

- 地址总线
- 控制总线
- 数据总线

CPU 从内存中读取数据



1. CPU 通过地址总线将地址信息 3 发出；
2. CPU 通过控制总线发出内存读命令，选中存储器芯片，并通知它，将要从中读取数据；
3. 存储器将 3 号单元的数据 8 通过数据总线送入 CPU。

CPU 对存储器的读写

写操作与读操作的步骤类似。如向 3 号单元写入数据 26。

1. CPU 通过地址总线将地址信息 3 发出；
2. CPU 通过控制总线发出内存写命令，选中存储器芯片，并通知它，将要从中写入数据；
3. CPU 通过数据总线将数据 26 送入内存的 3 号单元中。

CPU 对存储器的读写

问题　我们知道了 CPU 如何进行数据的读写。那么，我们又如何命令计算机进行数据的读写呢？

CPU 对存储器的读写

问题　我们知道了 CPU 如何进行数据的读写。那么，我们又如何命令计算机进行数据的读写呢？

要让计算机工作，应向它输入能驱动其进行工作的电平信息，即机器码。

机器指令：1010 0001 0000 0011 0000 0000

汇编指令：mov ax, [3]

含义：传送 3 号单元的内容入 ax。

地址总线

- CPU 通过地址总线指定存储器单元。由此可见，地址总线能传送多少个不同的信息，CPU 就可以对多少个存储单元进行寻址。

地址总线

- CPU 通过地址总线指定存储器单元。由此可见，地址总线能传送多少个不同的信息，CPU 就可以对多少个存储单元进行寻址。
- 现假设一个 CPU 有 10 根地址总线，可以传送 10 位二进制数据，共 2^{10} 个不同数据，最小为 0，最大为 1023。

地址总线

- CPU 通过地址总线指定存储器单元。由此可见，地址总线能传送多少个不同的信息，CPU 就可以对多少个存储单元进行寻址。
- 现假设一个 CPU 有 10 根地址总线，可以传送 10 位二进制数据，共 2^{10} 个不同数据，最小为 0，最大为 1023。
- 一个 CPU 有 N 根地址线，则称 CPU 的地址总线的宽度为 N，最多可以寻找 2^N 个内存单元。

地址总线

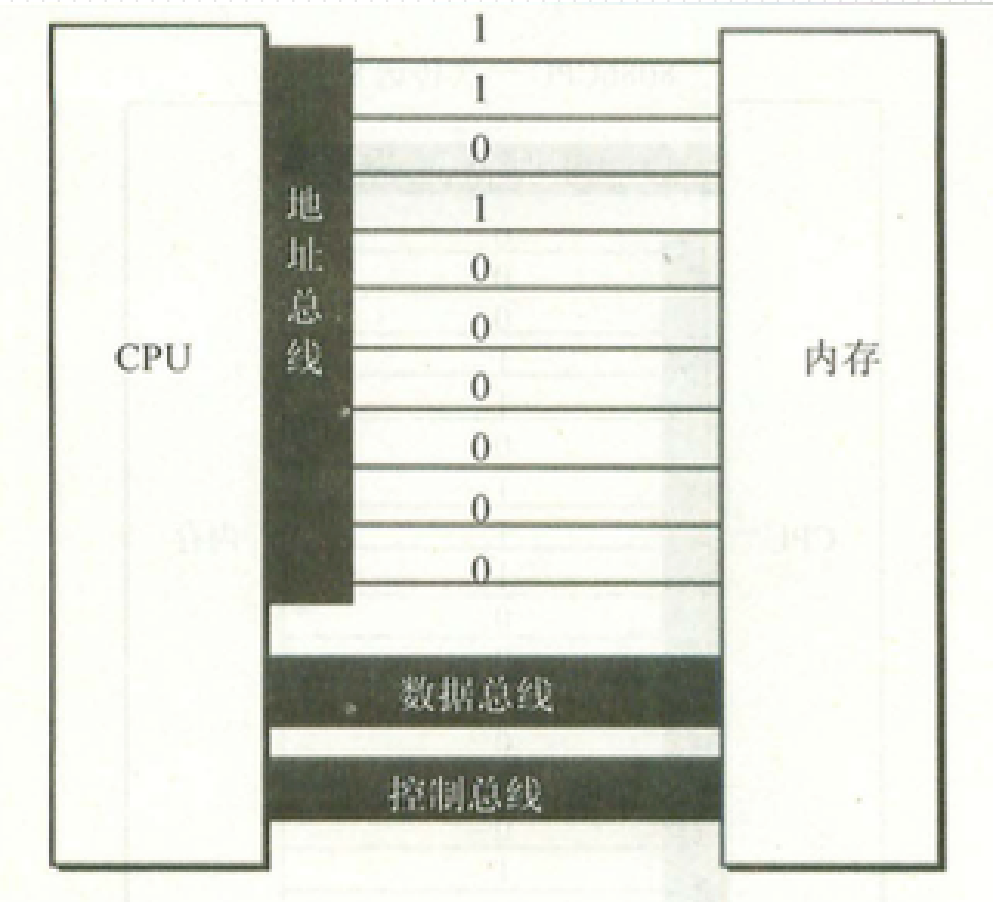


图: 地址总线发送的地址信息

数据总线

CPU 与内存和其它器件之间的数据传输通过数据总线来进行。数据总线的宽度决定了 CPU 与外界的数据传输速度。

数据总线

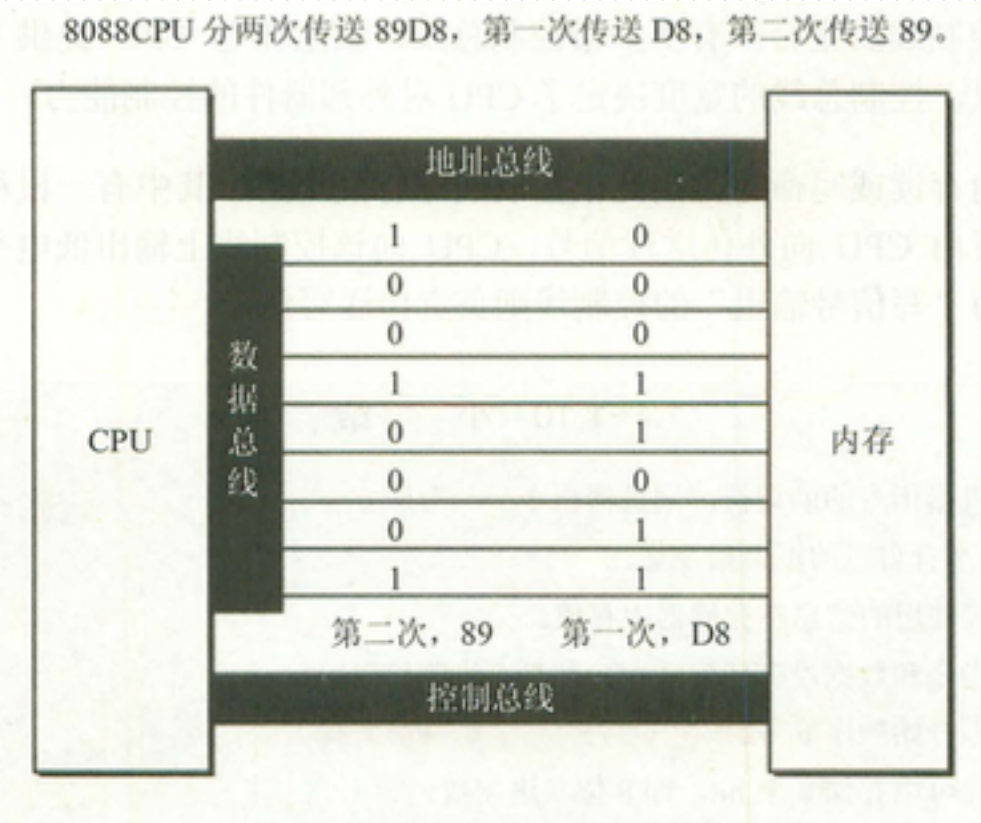


图: 8 根数据总线一次可传输一个字节

数据总线

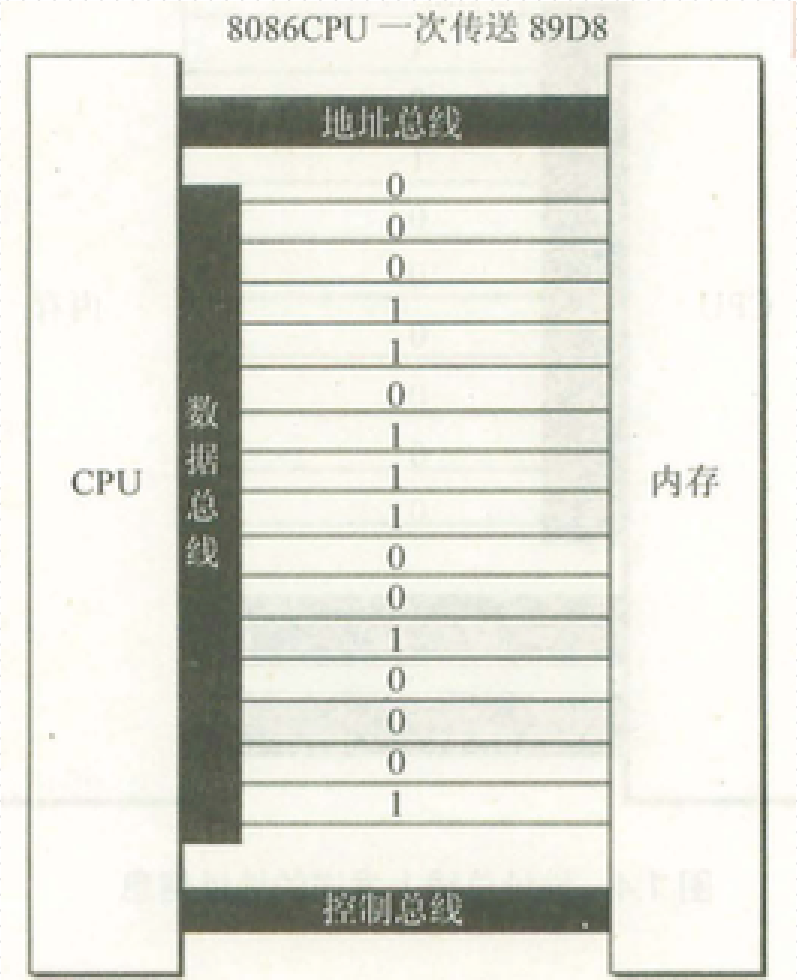


图: 16 根数据总线一次可传输两个字节

控制总线

CPU 对外部器件的控制通过控制总线来进行。有多少根控制总线，就意味着 CPU 提供了对外部器件的多少种控制。因此，控制总线的宽度决定了 CPU 对外部器件的控制能力。

寄存器

一个典型的 CPU 由运算器、控制器、寄存器等器件构成，这些器件通过内部总线相连。内部总线实现 CPU 内部各个器件之间的联系，而外部总线实现 CPU 与主板上其它器件的联系。

在 CPU 中：

- 运算器进行信息处理；
- 寄存器进行信息存储；
- 控制器控制各种器件进行工作；
- 内部总线连接各种器件，在它们之间进行数据的传送。

寄存器

寄存器是 CPU 中程序员可以用指令读写的部件，程序员通过改变各种寄存器中的内容来实现对 CPU 的控制。

不同的 CPU，寄存器的个数、结构不尽相同。8086CPU 有 14 个寄存器，每个寄存器都有一个名称，分别是

AX、BX、CX、DX、SI、DI、SP、BP、IP、CS、SS、DS、ES、P

通用寄存器

8086CPU 的所有寄存器都是 16 位的，可以存放两个字节。

AX、BX、CX、DX 这四个寄存器通常用来存放一般性的数据，被称为通用寄存器。

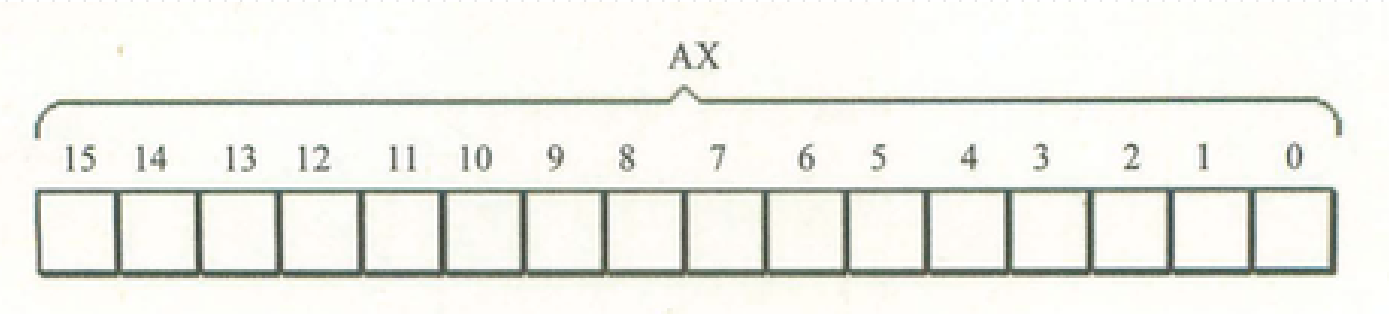


图: 16 位寄存器的逻辑结构

通用寄存器

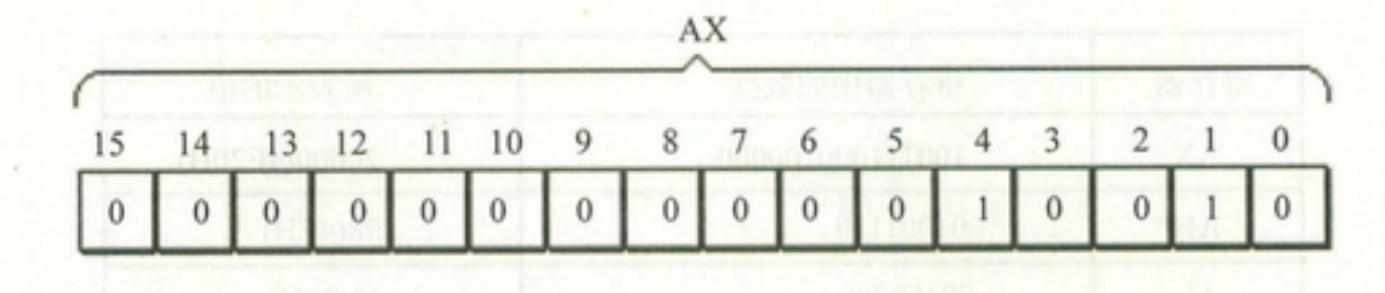


图: $(10010)_2$ 在寄存器 AX 中的存储

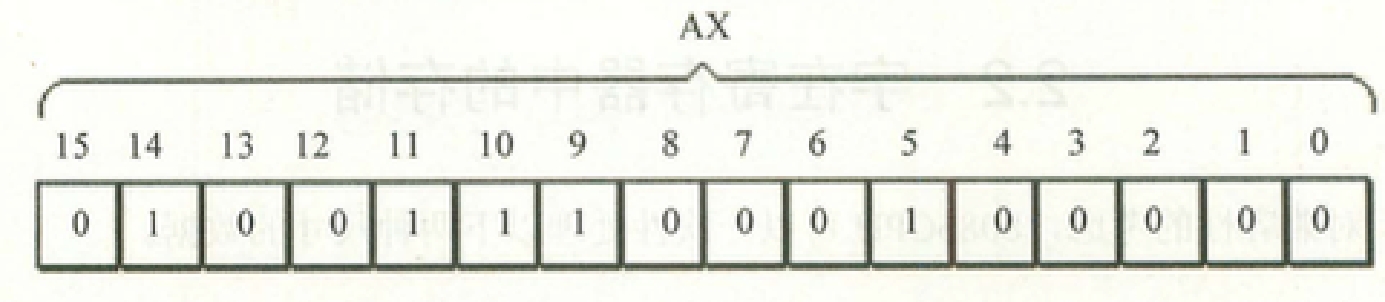


图: $(1001110001000000)_2$ 在寄存器 AX 中的存储

通用寄存器

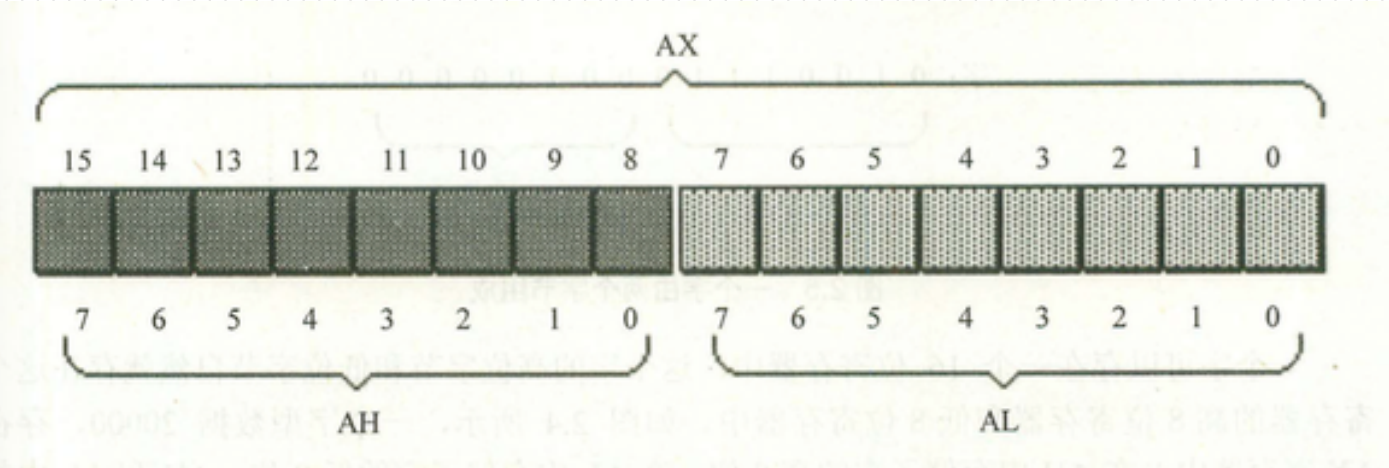


图: 16 位寄存器可分为两个 8 位寄存器

几条汇编指令

汇编指令	控制 CPU 完成的操作	用高级语言的语法描述
mov ax,18	将 18 送入寄存器 AX	AX=18
mov ah,78	将 78 送入寄存器 AH	AH=78
add ax,8	将寄存器 AX 中的数值加上 8	AX=AX+8
mov ax,bx	将寄存器 BX 中的数据送入寄存器 AX	AX=BX
add ax,bx	将 AX 和 BX 中的数值相加，结果存在 AX 中	AX=AX+BX

图: 汇编指令举例

物理地址

CPU 访问内存单元时，要给出内存单元的地址。所有的内存单元构成的存储空间是一个一维的线性空间，每一个内存单元在这个空间中都有唯一的地址，称为物理地址。

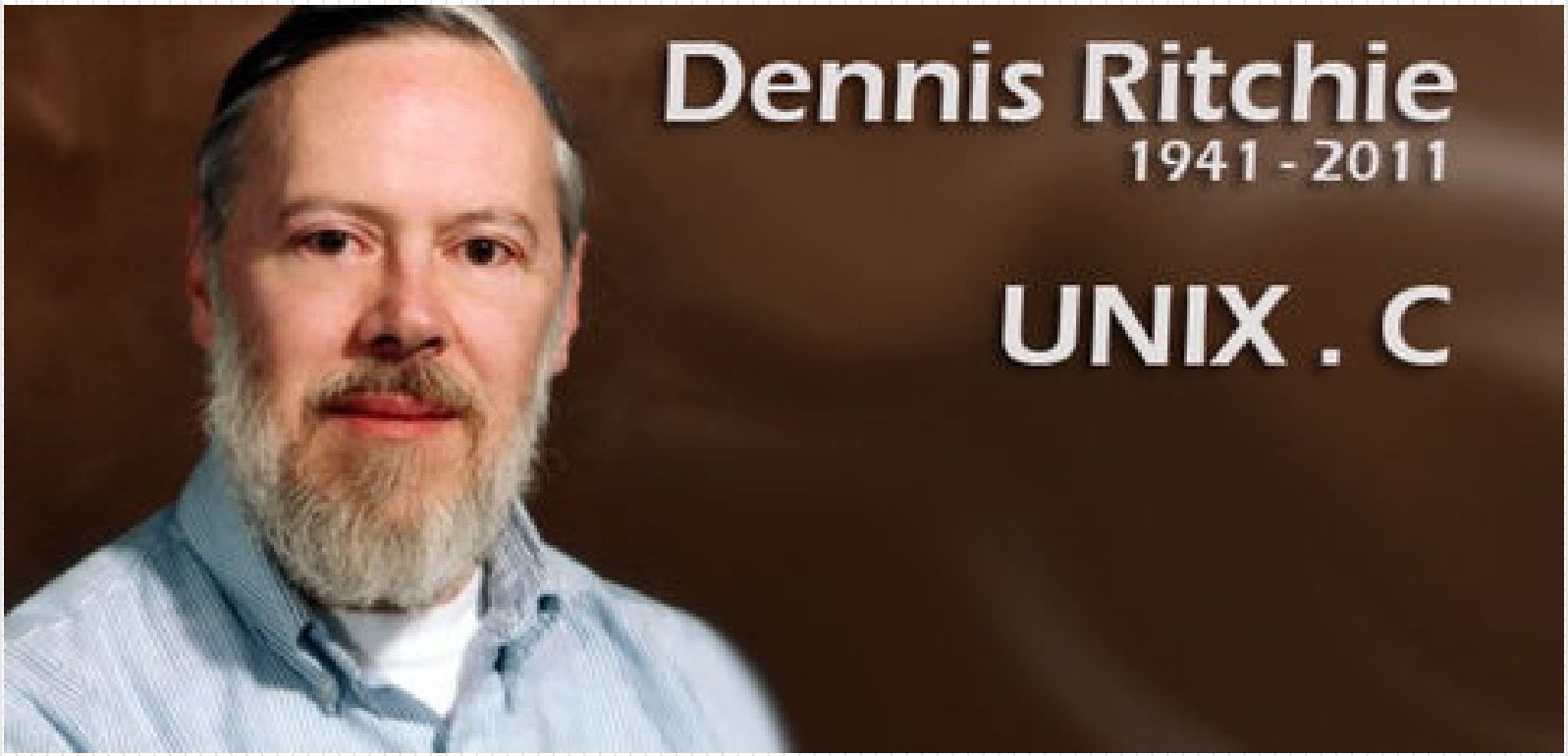
CPU 通过地址总线送入存储器的，必须是一个内存单元的物理地址。在 CPU 向地址总线上发出物理地址之前，必须要在内部先形成这个物理地址。不同的 CPU 可以有不同的形成物理地址的方式。

5. C 语言简介

C 的起源

- 产生时间：1972-1973 年
- 产生地点：美国贝尔实验室
- 创始人：Dennis Ritchie & Ken Thompson
- 目的：改写 Unix 系统
- 荣誉：美国国家技术奖章 (1999)

C 的起源



C 的起源

图: Ken Thompson (1942-)

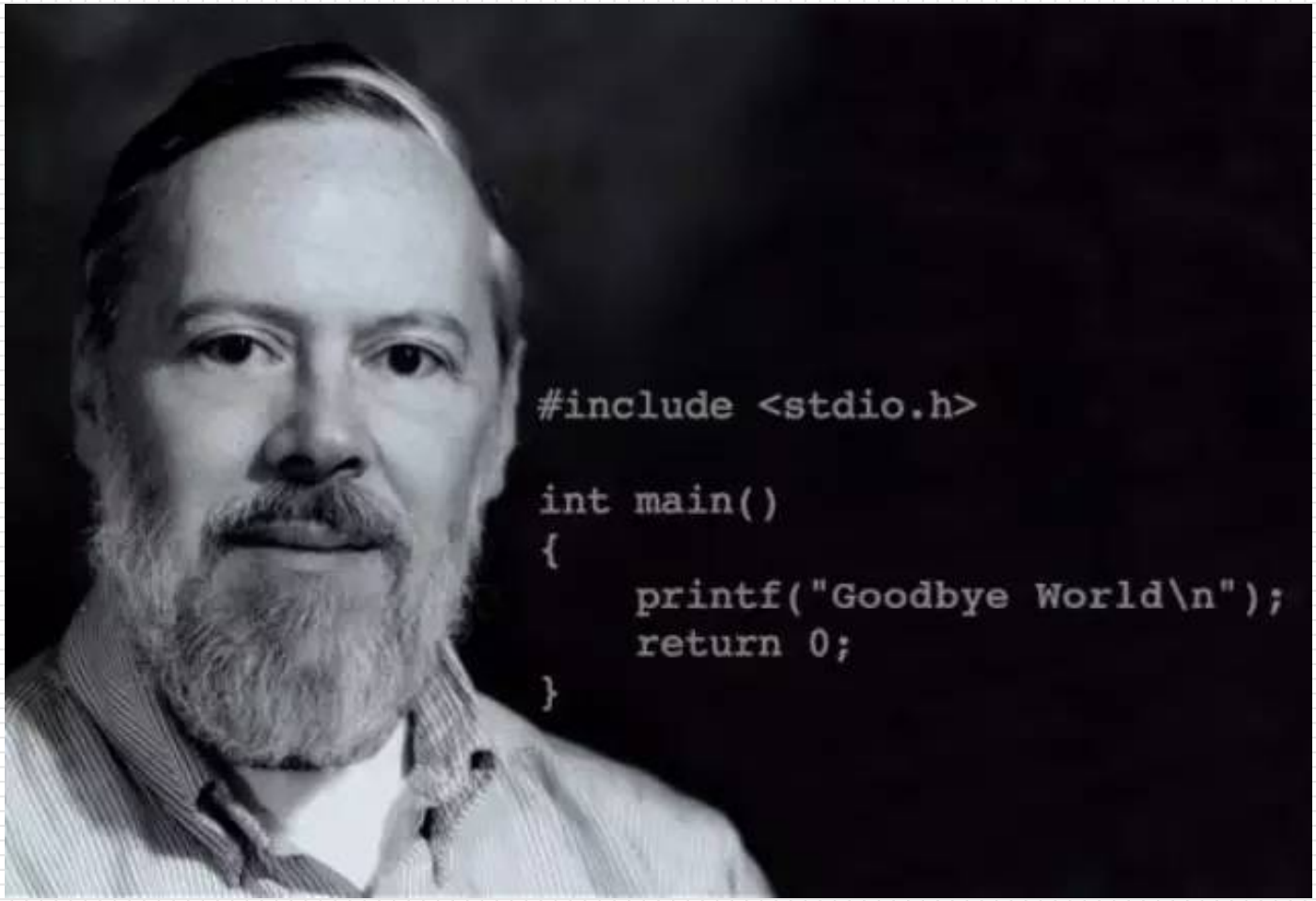
C 的起源

图: Dennis Ritchie 和 Ken Thompson(1972 年)

C 的起源

1983 年，Dennis Ritchie 和 Ken Thompson 一起获得了图灵奖，理由是 :“研究发展了通用的操作系统理论，尤其是实现了 Unix 操作系统”。

关于 Ritche 的评价



关于 Ritche 的评价 1

“当乔布斯去世时，享受到了声势浩大的追思。相形之下，里奇先生对当代科技进程做出了更大的贡献，可公众甚至不知道他是谁，这十分不公平。”

关于 Ritche 的评价 2

“如果说，乔布斯是可视化产品中的国王，那么里奇就是不可见王国中的君主。乔布斯的贡献在于，他如此了解用户的需求和渴求，以至于创造出了让当代人乐不思蜀的科技产品。然而，却是里奇先生为这些产品提供了最核心的部件，人们看不到这些部件，却每天都在使用着。”

关于 Ritche 的评价 3

“牛顿说他是站在巨人的肩膀上，如今，我们都站在里奇的肩膀上。”

C 的地位

- C 拥有汇编语言的力量和便利性，其运行方式更接近于硬件系统；
- C 所提供的数据结构，力发千钧，足以贯穿所有高层和底层的语言；

C 的优点

- C 的开发是科技史上不可磨灭的伟大贡献，因为这个语言把握住了计算机科技中一个至关重要的并且是恰到好处的中间点，一方面它具备搭建高层产品的能力，另一方面又能够对于底层数据进行有效控制。正是由于这种关联性和枢纽性作用，决定了 C 所导向的近三十年来计算机编程主流方式。

C 的优点

(1) 设计特性：C 融合了控制特性，使得用户可以采用自顶而下的结构化编程，以及模块化的设计，从而使编写出的程序更可靠、更易懂。

(2) 高效性：C 程序紧凑且运行速度快，可以表现出只有汇编语言才具有的精细控制能力。

(3) 可移植性：在一个系统上编写的 C 程序经过很少改动或不经修改就可以在其他系统上运行。

C 的优点

(4) 强大的功能与灵活性

- 强大而灵活的 UNIX 操作系统便是用 C 编写的。
- 其他语言（如 Fortran、Python、Pascal 等）的许多编译器和解释器也都是用 C 编写的。

(5) 面向程序员

- 它允许你访问硬件，并可以操纵内存中的特定位。
- 它具有丰富的运算符供选择，让你能够简洁地表达自己的意图。

C 的缺点

(1) C 在表达方面的自由会增加风险。

(2) C 对指针的使用，可能导致你会犯难以追踪的编程错误。

自由的代价是永远的警惕。

(3) C 的简洁性与丰富的运算符相结合，可能会产生极难理解的代码。

含糊代码竞赛 www.ioccc.org

6. 使用 C 语言的七个步骤

使用 C 语言的七个步骤

(1) 定义程序目标

开始时应对程序做什么有一个清晰的想法。

考虑程序需要的信息、需要进行的计算和操作，以及程序应该向你报告的信息。

使用 C 语言的七个步骤

(2) 设计程序

如何表示数据

用什么方法处理数据

选择一个好的方式来表示信息可以使程序设计和数据处理容易很多

使用 C 语言的七个步骤

(3) 编写代码

用文本编辑器创建一种称为源代码的文件。

常见的文本编辑器 Ultraedit、Emacs、Vi、集成开发环境 (IDE) 中自带的编辑器等。

使用 C 语言的七个步骤

(4) 编译

编译器将源代码转换为可执行文件，可执行文件是机器语言表示代码。

(5) 运行程序

在 Dos、Unix 和 Linux 系统中，可在命令行中直接键入可执行文件名即可运行程序。

在 Windows 和 MAC 环境提供的 IDE 中可通过选择菜单选项或特定快捷键来执行程序。

使用 C 语言的七个步骤

(6) 测试和调试程序

Bug

Debug

(7) 维护和修改程序

7. 编译 C 程序的工作原理

编译 C 程序的工作原理

C 是一种高级语言，它需要编译器将其转换为可执行代码，以使得程序能在机器上运行。

以下介绍在 MAC 或 Linux 上使用 gcc 编译器的几个步骤。

编译 C 程序的工作原理

(1) 首先使用编辑器（如 vi 或 emacs 等）创建一个 C 程序，并将其保存为 add2num.c。

```
$ emacs add2num.c
```

编译 C 程序的工作原理

在编辑界面输入以下内容：

```
// program for the addition of two numbers
#include<stdio.h>
#define add(a,b) (a+b) //using macros
int main(void)
{
  int a = 5, b = 4;
  printf("Addition␣is:␣%d\n", add(a,b));
  return 0;
}
```

编译 C 程序的工作原理

(2) 然后用以下命令编译，并查看当前目录下的文件。

```
$ gcc -Wall add2num.c  o add2num
$ ls
add2num     add2num.c
```

选项-Wall启动所有编译器的警告信息。建议使用该选项以生成更好的代码。选项-o用来制定输出文件名。如果缺省该选项，则输出文件将默认为a.out。

编译 C 程序的工作原理

(3) 编译通过后，将会生成可执行文件，可使用以下命令来运行生成的可执行文件。

```
$ ./add2num
Addition is: 9
```

编译 C 程序的工作原理

编译器将一个 C 程序转换为一个可执行文件，需经历了 4 个阶段：

- 预处理
- 编译
- 汇编
- 链接

编译 C 程序的工作原理

执行以下命令，在当前目录下会生成所有的中间文件以及可执行文件。

```
$ gcc -Wall -save-temps filename.c  o filename
$ ls
add2num      add2num.c   add2num.o
add2num.bc   add2num.i   add2num.s
```

编译 C 程序的工作原理

接下来，让我们一个个地来看这些中间文件中的内容。

(1) 预处理

这是第一阶段，包括

- 去掉注释
- 宏的展开
- 头文件的展开

预处理的输出保存在文件 add2num.i 中。

编译 C 程序的工作原理

```
$ less add2num.i
$ ls
...
int printf(const char * restrict, ...) __attribute__
((__format__ (__printf__, 1, 2)));
# 499 "/usr/include/stdio.h" 2 3 4
# 3 "add2num.c" 2

int main(void)
{
  int a = 5, b = 4;
  printf("Addition is: %d\n", (a+b));
  return 0;
}
```

编译 C 程序的工作原理

分析：在以上内容中，源文件被附加了很多信息，但在末尾代码仍被保留。`printf()` 函数中包含了 `a+b` 而非 `add(a,b)`，因为宏已被展开。注释被去除掉。`#include<stdio.h>` 没有了，取而代之的是很多代码。因此，头文件已被展开，并且被包含到了源文件中。

编译 C 程序的工作原理

(2) 编译

接下来就是编译 `add2num.i`，并生成一个编译过的输出文件 `add2num.s`。该文件为汇编指令，可用以下命令查看：

编译 C 程序的工作原理

```
$ less add2num.s
...
        movl    $0, -4(%rbp)
        movl    $5, -8(%rbp)
        movl    $4, -12(%rbp)
        movl    -8(%rbp), %eax
        addl    -12(%rbp), %eax
        movl    %eax, %esi
        movb    $0, %al
        callq   _printf
        xorl    %esi, %esi
...
```

以上内容表明这是汇编语言，能为汇编器所识别。

编译 C 程序的工作原理

(3) 汇编

这一阶段，将通过汇编器将 `filename.s` 转换成 `filename.o`，该文件包含机器指令。需要注意的是，该阶段只会将现有代码转换成机器语言，而诸如 `printf()` 的函数调用则不会。

编译 C 程序的工作原理

```
$ less add2num.o
<CF><FA><ED><FE>^G^@^@^A^C^@^@^@^A^@^@^@^D^@^@^@
^@^B^@^@^@ ^@^@^@^@^@^@^Y^@^@^@<88>^A^@^@^@^@^@
@^@^
...
```

编译 C 程序的工作原理

(4) 链接

这是最后一个阶段，将完成所有函数调用及其定义的链接工作。链接器知道所有这些函数在何处执行。链接器也会做一些额外的工作，以添加一些启动和结束程序所需的额外代码。

编译 C 程序的工作原理

在命令行中输入以下命令，可看出从目标文件到可执行文件时文件大小的变化。这是因为链接器为我们的程序添加了额外的代码。

```
$ size add2num.o
__TEXT   __DATA   __OBJC   others     dec          hex
  145      0        0         32       177          b1
$ size add2num
__TEXT   __DATA   __OBJC   others     dec          hex
4096       4096     0   4294971392 4294979584 100003000
```