全国高等教育自学考试

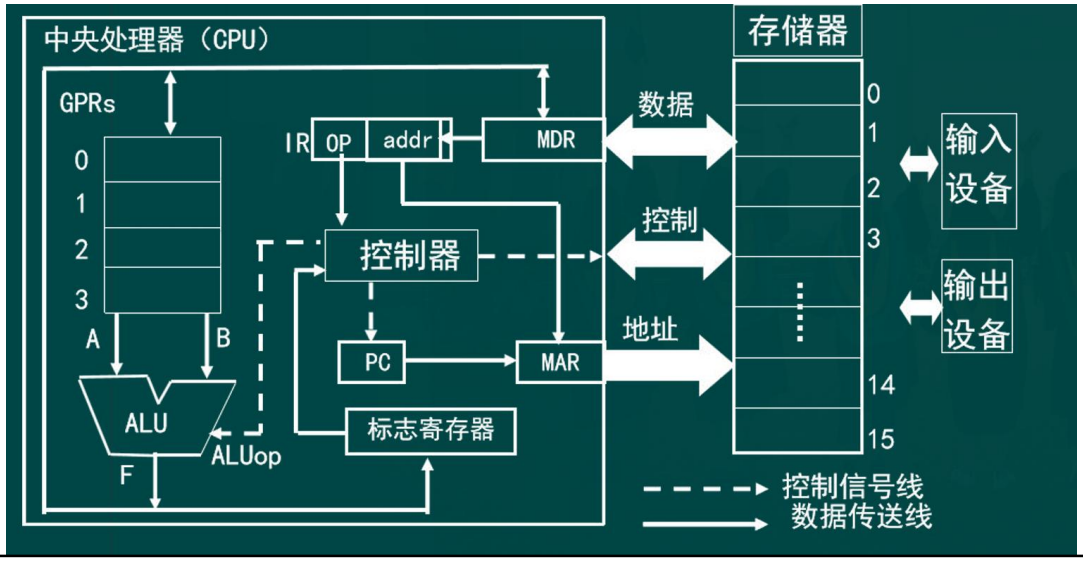
考前60分主观题带背

计算机系统原理

- ☐ 逻辑重点考学一点通
- ☑ 考前60分主观题带背
- ☐ 考前补漏百题斩
- ☐ 考前抢分黄金卷
- ☐ 考前速背决胜3小时
- ☐ 考前测练3年真题集锦

制作人 ○ 冯 文

第一章 计算机系统概述

知识点名称	内容
冯·诺依曼结构主要部件	1. ①用来存放指令和数据的主存储器，简称主存或内存②用来进行算术逻辑运算的部件，即算术逻辑部件（Arithmetic Logic Unit，ALU）在 ALU 操作控制信号 ALUop 的控制下，ALU 可以对输入端 A 和 B 进行不同的运算，得到结果 F③用于自动逐条取出指令并进行译码的部件，即控制元件（Control Unit，CU）也称控制器④用来和用户交互的输入设备和输出设备
冯·诺依曼结构总线	2. CPU 为了从主存取指令和存取数据。需要通过传输介质和主存相连，通常把连接不同部件进行信息传输的介质称为总线，其中，包含了用于传输地址信息、数据信息和控制信息的地址线、数据线和控制线 3. CPU 访问主存时，需先将主存地址、读/写命令分别送到总线的地址线、控制线，然后通过数据线发送或接收数据 4. CPU 送到地址线的主存地址应先存放在主存地址寄存器（MemoryAddress Register，MAR）中，发送到或从数据线取来的信息存放在主存数据寄存器（Memory Data Register，MDR）中
冯·诺依曼结构 模型计算机的硬件结构	5. 
程序和指令区别与联系	6.

程序	指令
由多个指令组成	单个命令或操作
高级组织形式	基本单位
用于完成复杂任务	用于执行具体的基本操作
体现整体逻辑和功能	体现具体的运算和操作步骤
例如：一个计算两个数相加的程序	例如：将值从一个寄存器加载到另一个寄存器

程序和指令的执行过程	7. 冯·诺依曼结构计算机的功能通过执行程序实现，程序的执行过程就是所包含的指令的执行过程。 8. 指令（instruction）是用 0 和 1 表示的一串 0/1 序列，用来指示 CPU 完成一个特定的原子操作，例如：（1）取数指令（load）从主存单元中取出数据存放到通用寄存器中（2）存数指令（store）将通用寄存器的内容写入主存单元（3）加法指令（add）将两个通用寄存器内容相加后送入结果寄存器（4）传送指令（mov）将一个通用寄存器的内容送到另一个通用寄存器，如

	此等等 9. 指令通常被划分为若干个字段，有操作码、地址码等字段：（1）操作码字段指出指令的操作类型，如取数、存数、加、减、传送、跳转等（2）地址码字段指出指令所处理的操作数的地址，如寄存器编号、主存单元编号等
翻译程序	10. 1）汇编程序（Assembler）：也称汇编器。用于将汇编语言源程序翻译成机器语言目标程序 2）解释程序（Interpreter）：也称解释器。用于将源程序中的语句按其执行顺序逐条翻译成机器指令并立即执行，如 PYTHON，BASIC 等 3）编译程序（Compiler）：也称编译器。用于将高级语言源程序翻译成汇编语言或机器语言目标程序，如 C，C++语言
程序设计语言和翻译程序	11. 1）预处理阶段：预处理程序（cpp）对源程序中以字符“#”开头的命令进行处理。例如，将#include 命令后面的.h 文件内容嵌入源程序文件中。预处理程序的输出结果还是一个源程序文件。以.i 为扩展名 2）编译阶段：编译程序（ccl）对预处理后的源程序进行编译，生成一个汇编语言源程序文件，以.s 为扩展名，例如，hello.s 是一个汇编语言程序文件。因为汇编语言与具体的机器结构有关，所以对同一台机器来说。不管什么高级语言。编译转换后的输出结果都是同一种机器语言对应的汇编语言源程序 3）汇编阶段：汇编程序（as）对汇编语言源程序进行汇编，生成一个可重定位目标文件（relocatable object file），以.o 为扩展名，例如，hello.o 是一个可重定位目标文件。它是一种二进制文件（binary file）因为其中的代码已经是机器指令，数据以及其他信息也都是用二进制表示的，所以它是不可读的。也即打开显示出来的是乱码 4）链接阶段：链接程序（ld）将多个可重定位目标文件和标准函数库中的可重定位目标文件合并成为一个可执行目标文件（executable object file）可执行目标文件简称为可执行文件。本例中，链接器将 hello.o 和标准库函数 printff（）所在的可重定位目标模块 printf.o 进行合并，生成可执行文件 hello
计算机的工作原理	12. 计算机的工作原理主要包括以下几个步骤：首先是输入阶段，用户通过输入设备将需要处理的数据输入到计算机中；然后是存储阶段，计算机把输入的数据暂时存放在主存储器中；接下来是运算阶段，计算机通过运算器对存储器中的数据进行运算处理；最后是输出阶段，计算机将处理完的数据通过输出设备输出给用户
计算机系统层次转换示意图	13. 
系统软件	14. 系统软件（System Software）包括为有效、安全地使用和管理计算机以及为开发和运行应用软件而提供的各种软件，介于计算机硬件与应用程序之间，它与具体应用关系不大。系统软件

	包括操作系统（如 Windows、UNIX、Linux）、语言处理系统（如 Visual Studio、GCC）数据库管理系统（如 Oracle）和各类实用程序（如磁盘碎片整理程序、备份程序）等软件。操作系统主要用来管理整个计算机系统的资源，包括对它们进行调度、管理、监视和服务等，操作系统还提供计算机用户和硬件之间的人机交互界面，并提供对应用软件的支持。语言处理系统主要用于提供一个用高级语言编程的环境，包括源程序编辑、翻译、调试、链接、装入运行等功能
应用软件	15. 应用软件（Application Software）指专门为数据处理、科学计算、事务管理、多媒体处理、工程设计以及过程控制等应用所编写的各类程序。例如，人们平时经常使用的电子邮件收发软件、多媒体播放软件、游戏软件、炒股软件、文字处理软件、电子表格软件、演示文稿制作软件等都是应用软件
计算机系统的不同用户	16. 最终用户：使用应用程序完成特定任务的计算机用户称为最终用户（end user）几乎每个人都成为了计算机的最终用户。计算机最终用户使用键盘和鼠标等外设与计算机交互，通过操作系统提供的用户界面启动执行应用程序或系统命令，从而完成用户任务。因此，最终用户能够感知到的只是系统提供的简单人机交互界面和安装在计算机中的相关应用程序 17. 系统管理员：系统管理员（system administrator）是指利用操作系统等软件提供的功能对系统进行配置、管理和维护，以建立高效合理的系统环境供计算机用户使用的操作人员。其职责主要包括：安装、配置和维护系统的硬件和软件，建立和管理用户账户，升级软件，备份和恢复业务系统和数据等 18. 应用程序员：应用程序员（application programmer）是指使用高级编程语言编制应用软件的程序员。应用程序员大多使用高级语言编写程序。程序设计高级语言是面向算法设计的较接近于日常所用的英语书面语言的程序设计语言，例如：BASIC、C/C++、Fortran Java 等 19. 系统程序员：系统程序员（system programmer）指设计和开发系统软件的程序员。系统程序员开发操作系统、编译器和实用程序等系统软件时，需要熟悉计算机底层的相关硬件和系统结构，甚至可能需要直接与计算机硬件和指令系统打交道。因此，系统程序员必须熟悉指令系统、机器结构和相关的机器功能特性，有时还要直接用汇编语言等低级语言编写程序代码
未定义行为	20. 未定义行为：未定义行为指语言标准规范中没有明确指定其行为的情况。若编写了未定义行为的源程序，则每次执行结果可能不同，或在不同平台下执行结果可能不同。例如，C 语言标准指出，当格式说明符和参数类型不匹配时，输出结果是未定义的，因此，以下 C 程序段就属于未定义行为代码。int x=1234; printf（"%lf", x）;
未指定行为	21. 未指定行为：未指定行为是指语言标准规范中没有强制规定程序行为，而是列出多种结果供编译器选择，不同编译器可能选择不同行为结果。若源程序包含未指定行为，则采用不同编译器或同一编译器的不同版本，目标程序的运行结果都可能不同。例如，对于程序段“inti=1; f（i++, i++）;”C 语言标准规定，函数调用的参数求值顺序未指定，故编译器可能按“f（1, 2）”处理，也可能按“f（2, 1）”处理
实现定义行为	22. 实现定义行为：实现定义行为指语言标准规范的实现（如编译器）需要在文档中说明其选择的未指定行为。若源程序包含实现定义行为，在相同环境下运行可得到相同结果，但将程序移植到另一个环境时，运行结果可能不同。例如，C 语言标准规定，char 属于带符号整数还是无符号整数类型是实现定义行为。当程序员想当然地认为 char 类型一定按带符号整数运算时，编

	译器可能把 char 类型当成无符号整数处理，从而使程序得到非预期的结果
吞吐率	23. 吞吐率（Throughput）和响应时间（Response Time）是考量一个计算机系统性能的两个基本指标。Ø吞吐率表示在单位时间内所完成的工作量，类似的概念是带宽（Bandwidth）它表示单位时间内所传输的信息量。Ø响应时间是指从作业提交开始到作业完成所用的时间，类似的概念是执行时间（Execution Time）和等待时间（Latency）它们都是用来表示一个任务所用时间的度量值
时钟周期	24. 时钟周期：计算机执行一条指令的过程被分成若干步骤，由每一步中相应的操作来完成指令功能。每一步操作都要有相应的控制信号进行控制，用于对控制信号进行定时的同步信号就是 CPU 的时钟信号，其宽度为一个时钟周期（clock cycle，tick，clock tick，clock）
时钟频率	25. 时钟频率：CPU 的主频就是 CPU 时钟信号的时钟频率（clock rate）是 CPU 时钟周期的倒数。时钟频率的单位通常为 MHz 或 GHz。主频为 1.0 MHz 表示每秒钟发生 106 个时钟信号，因此时钟周期为 10-6s（秒）=1μs（微秒）；主频为 1.0 GHz 表示每秒钟发生 109 个时钟信号，因此时钟周期为 10-9s=1ns（纳秒）
CPI	26. CPI（cycles per instruction）表示执行一条指令所需的时钟周期数。由于不同指令的功能不同，因而执行不同指令所需的时钟周期数也不同，因此，对于一条特定指令而言，其 CPI 指执行该条指令所需的时钟周期数，此时 CPI 是一个确定的值 27. 对于一个程序或一台机器来说，其 CPI 指该程序或该机器指令集中的所有指令执行所需的平均时钟周期数，此时，CPI 是一个平均值，通常称为综合 CPI
指令速度	28. 指令速度所用的计量单位为 MIPS（Million Instructions Per Second）其含义是平均每秒钟执行多少百万（106）条指令
基准程序	29. 基准程序（benchmarks）是进行计算机性能评测的一种重要工具。基准程序是专门用来进行性能评价的一组程序，能够很好地反映机器在运行实际负载时的性能，可以通过在不同机器上运行相同的基准程序来比较在不同机器上的运行时间，从而评测其性能 30. 基准程序最好是用户经常使用的一些实际程序，或是某个应用领域的一些典型的简单程序。 对于不同的应用场合，应该选择不同的基准程序。例如，对用于软件开发的计算机进行评测时，最好选择包含编译器和文档处理软件的一组基准程序；而如果是对用于 CAD 处理的计算机进行评测时，最好选择一些典型的图形处理小程序作为一组基准程序

第二章 数据的表示和运算

知识点名称	内容
信息的二进制编码	1. 在计算机内部，所有信息都用二进制数字表示。这是因为：（1）二进制只有两种基本状态，而使用有两个稳定状态的物理器件可以容易地表示二进制数的每一位（2）二进制的编码、计数和运算规则都很简单，可用开关电路实现，简便易行（3）两个符号“1”和“0”正好与逻辑命题的两个值“真”和“假”相对应，为计算机中实现逻辑运算和程序中的逻辑判断提供了便利的条件，特别是能通过逻辑门电路方便地实现算术运算 2. 指令所处理的基本数据类型分为两种：数值数据和非数值数据。Ø数值数据可用来表示数量的多少，可比较其大小，分为整数和实数。Ø整数又分为无符号整数和带符号整数。在计算机内部，整数用定点数表示，实数用浮点数表示。Ø非数值数据没有大小之分，不表示数量的多少，

主要包括字符数据和逻辑数据

3. 表示一个数值数据要确定三个要素：（1）Ø进位记数制（2）Ø定/浮点表示（3）Ø编码规则

常用的几种进位记数制

4. 二进制 R=2，基本符号为 0 和 1
5. 八进制 R=8，基本符号为 0，1，2，3，4，5，6，7
6. 十六进制 R=16，基本符号为 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F
7. 十进制 R=10，基本符号为 0，1，2，3，4，5，6，7，8，9
8. 书写方式：（1）用下标 2，10，8，16 进行表示（2）使用后缀字母标识该数的进位数制，一般用 B 表示二进制，用 O 表示八进制，用 D 表示十进制（十进制数的后缀可以省略）而 H 则是十六进制数的后缀 （3）有时也在一个十六进制数之前用 0x 作为前缀
9.

四种进位计数之间的对应关系

二进制	八进制	十进制	十六进制	二进制	八进制	十进制	十六进制
0000	0	0	0	1000	10	8	8
0001	1	1	1	1001	11	9	9
0010	2	2	2	1010	12	10	A
0011	3	3	3	1011	13	11	C
0100	4	4	4	1100	14	12	C
0101	5	5	5	1101	15	13	D
0110	6	6	6	1110	16	14	E
0111	7	7	7	1111	17	15	F

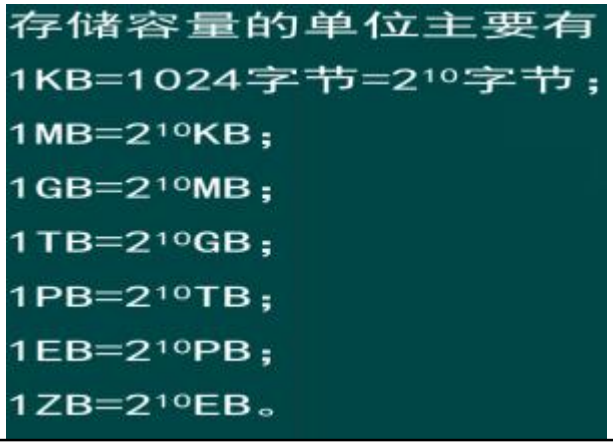
定点数的编码表示

10. 原码（Sign-and-Magnitude）是一种用于表示有符号数的编码方法。在这种表示方法中，最高位（最左边的一位）被用作符号位，其值为 0 表示正数，为 1 表示负数。其余的位数被用来表示数值的绝对值（即数的大小）
11. 补码（Two's Complement）是一种用于表示有符号数的二进制编码方法，它在计算机系统中被广泛使用，因为它能简化加减法运算，并且解决了原码表示存在的双重零问题。补码的表示方法，补码的表示方法规定：（1）正数的补码表示与其原码表示相同（2）负数的补码表示通过对其绝对值按位取反再加 1 来获得
12. 反码（One's Complement）是一种用于表示有符号数的二进制编码方法。相比于原码和补码，反码在历史上被使用过，但在现代计算机系统中已经较少采用。它主要用于简化一些特定计算尽管仍有局限性，例如两个零（+0 和-0）反码的表示方法，反码表示法的规则如下：（1）正数的反码表示与其原码相同（2）负数的反码表示通过对其绝对值的二进制码按位取反（即 0 变 1，1 变 0）来获得
13. 移码（Excess-N 或 Bias-N）是一种用于表示有符号数的二进制编码方法。与补码和反码的处理方式不同，移码通过将数值偏移一个固定的量（称为偏置值）来统一负数和正数的表示。这种表示方式在某些特定场景，如浮点数表示和特定硬件电路设计中，有其独特的优点

模运算

14. 在模运算系统中，若 A，B，M 满足下列关系：A=B+K×M（K 为整数）则记为：A=B（mod M）即 A、B 各除以 M 后的余数相同，故称 B 和 A 为模 M 同余。也就是说在一个模运算系统中，一个数

	与它除以“模”后得到的余数是等价的
补码定义	15. 补码的定义根据上述同余概念和数的互补关系，可引出补码表示方法：（1）正数的补码是它本身（2）负数的补码等于模与该负数绝对值之差
补码与真值之间的转换方法	16. 原码与真值之间的对应关系简单，只要对符号转换，数值部分不需改变。但对于补码来说，正数和负数的转换则不同。根据定义，求一个正数的补码时，只要将正号“+”转换为 0，数值部分无需改变；求一个负数的补码时，需要做减法运算，因而不太方便和直观 17. 因此，再求补码时，可以借助反码；有以下结论，正数：原码=反码=补码，负数：原码，反码：原码基础上，符号位不变，其余取反（0 变 1，1 变 0）补码=反码+1
无符号整数的表示	18. 当一个编码的所有二进位都用来表示数值而没有符号位时，该编码表示的就是无符号整数。此时，默认数的符号为正，所以无符号整数就是正整数或非负整数。 19. 一般在全部是正数运算且不出现负值结果的场合下，使用无符号整数表示。例如，可用无符号整数进行地址运算，或用来表示指针。通常把无符号整数简单地说成无符号数 20. 无符号整数没有符号位，在字长相同的情况下，它能表示的最大数比带符号整数所能表示的大，n 位无符号整数可表示的数的范围为 0～（2n-1）
带符号整数的表示	21. 带符号整数也被称为有符号整数，它必须用一个二进位来表示符号，虽然前面介绍的各种二进制定点数编码表示（包括原码、补码、反码和移码）都可以用来表示带符号整数 22. 但是补码表示有其突出的优点，主要体现在以下几方面：（1）与原码和反码相比，数 0 的补码表示形式唯一（2）与原码和移码相比，补码运算系统是一种模运算系统，因而可用加法实现减法运算，且符号位可以和数值位一起参加运算（3）与原码和反码相比，它比原码和反码多表示一个最小负数（4）与反码相比，不需要通过循环进位来调整结果 23. 现代计算机中带符号整数都用补码表示，故 n 位带符号整数可表示的数值范围为-2n-1～（2n-1-1）例如，8 位带符号整数的表示范围为-128～+127
实数的表示	24. 任意的一个实数 X，$X=(-1)^S \times M \times R^E$ 25. 其中 S 取值为 0 或 1，用来决定数 X 的符号，一般用 0 表示正，1 表示负；M 是一个二进制定点小数，称为数 X 的尾数；E 是一个二进制定点整数，称为数 X 的阶或指数；R 是基数，可以约定为 2、4、16 等。要确定一个实数的值，只要在默认基数 R 下，确定数符 S、尾数 M 和阶 E 就可以了
浮点数表示的计算步骤	26. （1）分解成符号、指数、尾数三部分（2）符号：0 正、1 负（3）把十进制转成二进制（4）写成科学计数法（5）指数偏移（6）记录尾数（7）写成计算机中的浮点数表示
C 语言中的浮点数类型	27. 当在 int、float 和 double 等类型数据之间进行强制类型转换时，程序将得到以下数值转换结果（假定 int 为 32 位）（1）从 int 转换为 float 时，不会发生溢出，但可能有有效数字被舍入（2）从 int 或 float 转换为 double 时，因为 double 的有效位数更多，故能保留精确值（3）从 double 转换为 float 时，因为 float 表示范围更小，故可能发生溢出，此外，由于有效位数变少，故可能被舍入（4）从 float 或 double 转换为 int 时，因为 int 没有小数部分，所以数据可能会向 0 方向被截断。例如，1.9999 9 被转换为 1，-1.999 9 被转换为-1。此外，因为 int 的表示范围更小，故可能发生溢出。将大的浮点数转换为整数可能会导致程序错误
逻辑值	28. 正常情况下，每个字或其他可寻址单位（字节，半字等）是作为一个整体数据单元看待的。但

	是，某些时候还需要将一个 n 位数据看成是由 n 个 1 位数据组成，每个取值为 0 或 1 29. 当数据以这种方式看待时，就被认为是逻辑数据。因此 n 位二进制数可表示 n 个逻辑值。逻辑数据只能参加逻辑运算，并且是按位进行的，如按位“与”按位“或”逻辑左移、逻辑右移等
汉字字符	30. （1）区位码：是表示某个字符在汉字字库中位置的编码（2）国标码：通常是指 GB2312 编码，用于表示简体中文字符（3）机内码：计算机内部使用的编码，这通常是在国标码的基础上增加偏移量得到的。 31. 在 GB2312 编码中：（1）区码：表示行号，范围是 0~93（2）位码：表示列号，范围是 01H 到 5EH（1 到 94） 32. 国标码的计算：国标码（GB2312 编码）是由区码+0x20H 和位码+0x20H 组成的，这样得到一个双字节编码，表示汉字中的一个字符。 33. 机内码：一种常见的方式是：机内码=国标码+0x8080H，所以，它将国标码再次转换到一个更高的值，以解决某些编码冲突或特定应用需求
存储容量的单位	34. 
大端	35. 大端（big endian）方式将数据的最高有效字节 MSB 存放在最小地址单元中，将最低有效字节 LSB 存放在最大地址单元中，即数据的地址就是 MSB 所在的地址
小端	36. 小端（little endian）方式将数据的最高有效字节 MSB 存放在高地址中，将最低有效字节 LSB 存放在低地址中，即数据的地址就是 LSB 所在的地址

第三章 程序的转换及机器级表示

知识点名称	内容
机器级语言	1. 机器语言和汇编语言统称为机器级语言；用机器指令表示的机器语言程序和用汇编指令表示的汇编语言程序统称为机器级程序，是对应高级语言程序的机器级表示。任何一个高级语言程序一定存在一个与之对应的机器级程序，而且是不唯一的。因此，如何将高级语言程序生成对应的机器级程序，并在时间和空间上达到最优，是编译优化要解决的问题
指令集体系结构	2. 在计算机系统的抽象层中，最重要的抽象层就是指令集体系结构（Instruction Set Architecture，ISA）它作为计算机硬件之上的抽象层，对使用硬件的软件屏蔽了底层硬件的实现细节，将物理上的计算机硬件抽象成一个逻辑上的虚拟计算机，称为机器语言级虚拟机。 3. ISA 定义了机器语言级虚拟机的属性和功能特性，主要包括如下信息：（1）可执行的指令的集合，包括指令格式、操作种类以及每种操作对应的操作数的相应规定（2）指令可以接受的操作数的类型（3）操作数或其地址所能存放的寄存器组的结构，包括每个寄存器的名称、编号、

	长度和用途（4）操作数所能存放的存储空间的大小和编址方式（5）操作数在存储空间存放时按照大端还是小端方式存放（6）指令获取操作数以及下一条指令的方式，即寻址方式（7）指令执行过程的控制方式，包括程序计数器、条件码定义等
指令类型	4. （1）RR 型（两个操作数都来自寄存器）（2）RS 型（两个操作数分别来自寄存器和存储单元）（3）SI 型（两个操作数分别来自存储单元和立即数）（4）SS 型（两个操作数都来自存储单元） 5. 按指令格式的复杂度来分，可分为 CISC 与 RISC 两种类型指令系统
RISC 风格指令系统	6. RISC 的着眼点不是简单地放在简化指令系统上，而是通过简化指令使计算机结构更加简单合理，从而提高机器的性能 7. 与 CISC 相比、RISC 指令系统的主要特点如下：①指令数目少②指令格式规整，采用定长指令字方式，操作码和操作数地址等字段的长度固定③只有 Load/Store 指令中的数据需要访存、这种称为 Load/Store 型指令风格④采用大量通用寄存器
使用 GCC 工具将一个 C 语言程序转换为可执行目标代码的步骤	8. ①预处理：例如、在 C 语言源程序中有一些以#开头的语句，可以在预处理阶段对这些语句进行处理，在源程序中插入所有用#include 命令指定的文件和用#define 声明指定的宏②编译：将预处理后的源程序文件编译生成相应的汇编语言程序③汇编：由汇编程序将汇编语言源程序文件转换为可重定位的机器语言目标代码文件④链接：由链接器将多个可重定位的机器语言目标文件以及库例程（如 printf（）库函数）链接起来，生成最终的可执行文
C 语言程序中的基本数据类型	9. 指针或地址：用来表示字符串或其他数据区域的指针或存储地址，可声明为 char *等类型，其宽度为 32 位，对应 IA-32 中的双字 10. 序数、位串等：用来表示序号、元素个数、元素总长度、位串等的无符号数，可声明为 unsigned char、unsigned short [int]、unsigned [int]、unsigned long[int]（括号中的 int 可省略）类型，分别对应 IA-32 中的字节、字、双字和双字。因为 IA-32 是 32 位架构，所以，编译器把 long 型数据定义为 32 位。ISO C99 规定 long long 型数据至少是 64 位，而 IA-32 中没有能处理 64 位数据的指令，因而编译器大多将 unsigned long long 型数据运算转换为多条 32 位运算指令来实现 11. 带符号整数：它是 C 语言中运用最广泛的基本数据类型，可声明为 signed char、short[int]、int、long[int]类型，分别对应 IA-32 中的字节、字、双字和双字，用补码表示。与对待 unsigned long long 型数据一样，编译器将 long long 型数据运算转换为多条 32 位运算指令来实现 12. 浮点数：用来表示实数，可声明为 float、double 和 long double 类型，分别采用 IEEE754 的单精度、双精度和扩展精度标准表示。long double 类型是 ISO C99 中新引入的，对于许多处理器和编译器来说，它等价于 double 类型，但是由于与 x86 处理器配合的协处理器 x87 中使用了深度为 8 的 80 位的浮点寄存器栈，对于 Intel 兼容机来说，GCC 采用了 80 位的“扩展精度”格式表示
控制标志的含义	13. DF（Direction Flag）：方向标志。用来确定串操作指令执行时变址寄存器 SI（ESI）和 DI（EDI）中的内容是自动递增还是递减。若 DF=1，则为递减；否则为递增。可用 std 指令和 cld 指令分别将 DF 置 1 和清 0 14. IF（Interrupt Flag）：中断允许标志。若 IF=1，表示允许响应中断；否则禁止响应中断。IF 对非屏蔽中断和内部异常不起作用，仅对外部可屏蔽中断起作用。可用 sti 指令和 cli 指令分别将 IF 置 1 和清 0

	15. TF（Trap Flag）：陷阱标志。用来控制单步执行操作。TF=1 时，CPU 按单步方式执行指令，此时，可以控制在每执行完一条指令后，就把该指令执行得到的机器状态（包括各寄存器和存储单元的值等）显示出来。没有专门的指令用于对该标志的修改，但可用栈操作指令（如 pushf/pushfd 和 popf/popfd）来改变其值。EFLAGS 寄存器的第 12～31 位中的其他状态或控制信息是从 80286 以后逐步添加的。包括用于表示当前程序的 I/O 特权级（IOPL）、当前任务是否是嵌套任务（NT）、当前处理器是否处于虚拟 8086 方式（VM）等一些状态或控制信息
IA-32 中的通用寄存器	16. （1)8 个 8/16/32 位定点通用寄存器(2)8 个 MMX 指令/x87 FPU 使用的 64 位/80 位寄存器 MMO/ST（0）～MM7/ST（7）（3）8 个 SSE 指令使用的 128 位寄存器 XMMO～XMM7
传送指令	17. 传送指令用于寄存器、存储单元或 I/O 端口之间传送信息，分为通用数据传送、地址传送、标志传送和 I/O 信息传送等几类，除了部分标志传送指令外，其他指令均不影响标志位的状态
通用数据传送指令	18. 通用数据传送指令传送的是寄存器或存储器中的数据。 19. 主要有以下几种：（1）MOV：一般的传送指令。包括 movb、movw 和 movl 等（2）MOVS：符号扩展传送指令。将短的源数据高位符号扩展后传送到目的地址，如 movsbw 表示把一个字节进行符号扩展后送到一个 16 位寄存器中（3）MOVZ：零扩展传送指令，将短的源数据高位零扩展后传送到目的地址，如 movzwl 表示把一个字的高位进行零扩展后送到一个 32 位寄存器中 20. 注意：MOVS 和 MOVZ 指令的目的地址只能是寄存器编号
输入/输出指令	21. 输入/输出指令专门用于在累加寄存器 AL/AX/EAX 和 I/O 端口之间进行数据传送。例如，in 指令用于将 I/O 端口内容送累加器，out 指令将累加器内容送 I/O 端口
标志传送指令	22. 标志传送指令专门用于对标志寄存器进行操作。如 pushf 指令用于将标志寄存器的内容压栈，popf 指令将栈顶内容送标志寄存器，因而 popf 指令可能会改变标志
加/减运算指令	23. 加/减类指令（ADD/SUB）用于对给定长度的两个位串进行相加或相减。两个操作数中最多只能有一个是存储器操作数，不区分是无符号数还是带符号整数，产生的和/差送到目的地。生成的标志信息送标志寄存器 FLAGS/EFLAGS
增/减运算指令	24. 增/减类（INC/DEC）指令对给定长度的一个位串加 1 或减 1，给定操作数既是源操作数也是目的操作数，不区分是无符号数还是带符号整数，生成的标志信息送标志寄存器 FLAGS/EFLACS，注意不生成 CF 标志
乘/除运算指令	25. 乘法指令分成 MUL（无符号整数乘）和 IMUL（带符号整数乘）两类。对于 IMUL 指令，可以显式地给出一个、两个或三个操作数。对于 MUL 指令，只须显式给出一个操作数
除法指令	26. 除法指令分成 DIV（无符号整数除）和 IDIV（带符号整数除）两类。指令中只明显指出除数，用累加器 AL/AX/EAX 中的内容除以指令中指定的除数。 27. （1）若源操作数为 8 位。则 16 位的被除数隐含在 AX 寄存器中，商送 AL，余数在 AH 中（2）若源操作数为 16 位，则 32 位的被除数隐含在 DX-AX 寄存器中，商送 AX，余数在 DX 中（3）若源操作数是 32 位。则 64 位的被除数在 EDX-EAX 寄存器中。商送 EAX。余数在 EDX 中。 28. 需要说明的是。如果商超过目的寄存器能存放的最大值，系统就产生类型号为 0 的异常。并且商和余数均不确定
按位运算指令	29. 按位运算指令用来对不同长度的操作数进行按位操作，立即数只能作源操作数，不能作为目的操作数，并且最多只能有一个为存储器操作数

	30. 逻辑运算指令：以下 5 类逻辑运算指令中，仅 NOT 指令不影响条件标志位，其他指令执行后，OF=CF=0，而 ZF 和 SF 则根据运算结果来设置：若结果为全 0，则 ZF=1；若最高位为 1，则 SF=1（1）NOT：单操作数的取反指令。它将操作数每一位取反，然后把结果送回对应位（2）AND：对双操作数按位逻辑“与”，主要用来实现“掩码”操作 31. 移位指令：移位指令将寄存器或存储单元中的 8、16 或 32 位二进制数进行算术移位、逻辑移位或循环移位。在移位过程中，把 CF 看作扩展位，用它接收从操作数最左或最右移出的一个二进制位。只能移动 1～31 位，所移位数可以是立即数或存放在 CL 寄存器中的一个数值：（1）SHL：逻辑左移，每左移一次，最高位送入 CF，并在低位补 0（2）SHR：逻辑右移，每右移一次，最低位送入 CF，并在高位补 0（3）SAL：算术左移，操作与 SHL 指令类似，每次移位，最高位送入 CF，并在低位补 0。执行 SAL 指令时，如果移位前后符号位发生变化，则 OF=1，表示左移后结果溢出。这是 SAL 与 SHL 的不同之处（4）SAR：算术右移，每右移一次，操作数的最低位送入 CF，并在高位补符号（5）ROL：循环左移，每左移一次，最高位移到最低位，并送入 CF（6）ROR：循环右移，每右移一次，最低位移到最高位，并送入 CF（7）RCL：带循环左移将 CF 作为操作数的一部分循环左移（8）RCR：带循环右移，将 CF 作为操作数的一部分循环右移
程序执行流控制指令	32. 指令执行的顺序在 IA-32 中由 EIP 确定。正常情况下，指令按照它们在存储器中的存放顺序一条一条地按序执行，但是，在有些情况下，程序需要跳转到另一段代码去执行，此时可通过直接将指令指定的跳转目标地址送 EIP 的方法实现跳转。 33. 有直接跳转和间接跳转两种方式：（1）直接跳转指跳转目标地址由出现在指令机器码中的立即数作为偏移量而计算得到（2）间接跳转则是指跳转目标地址间接存储在某寄存器或存储单元中 34. 跳转目标地址的计算方法有两种：（1）一种是通过将当前 EIP 的值加偏移量计算得到。因为偏移量是带符号整数。因此跳转目标地址为 EIP 内容增加或减少某一个数值得到。也就是采用相对寻址方式得到。可以看成是以当前 EIP 内容为基准往前或往后跳转。称为相对跳转（2）另一种是直接将指令中设置的目标地址设置到 EIP 中，称为绝对跳转
条件跳转指令	35. 条件跳转指令 Jcc（其中 cc 为条件助记符）以标志位或标志位组合作为跳转依据。如果满足条件，则跳转到由标号 label 确定的目标地址处执行；否则继续执行下一条指令。这类指令都采用相对寻址方式的直接跳转
调用和返回指令	36. 调用指令：调用指令 CALL 是一种无条件跳转指令，跳转方式与 JMP 指令类似。它包含两个操作①将返回地址入栈（相当于 PUSH 操作）②跳转到指定地址处执行。执行时。首先将当前 EIP 或 CS：EIP 的内容（即返回地址，相当于 CALL 指令下面一条指令的地址）入栈。然后将调用目标地址（即子程序的首地址）装入 EIP 或 CS：EIP、以跳转到被调用的子程序执行。显然。CALL 指令会修改找指针 ESP。 37. 返回指令：返回指令 RET 也是一种无条件跳转指令，通常放在子程序的末尾，使子程序执行后返回主程序继续执行。该指令执行过程中，返回地址被从栈顶取出（相当于 POP 指令），并送到 EIP 寄有器（段内或段间调用时）和 CS 寄存器（仅段间调用）。显然，RET 指令会修改栈指针。若 RET 指令带有一个立即数 n，则当它完成上述操作后。还会执行 R[ep]←R[sp]+n 操作，从而实现预定的修改栈指针 ESP 的目的
陷阱指令	38. 陷阱也称为自陷或陷入。它是预先安排的一种“异常”事件、就像预先设定的“陷阱”一样。

	当执行到陷附指令（也称自陷指令）时、CPU 就调出特定的程序进行相应处理，处理结束后返回到陷阱指令的下一条指令执行
过程调用的执行步骤	39. 假定过程 P 调用过程 Q，则 P 称为调用者（Caller）Q 称为被调用者（Callee）过程调用的执行步骤如下：1）P 将入口参数（实参）放到 Q 能访问到的地方 2）P 将返回地址存到特定的地方，然后将控制转移到 Q3）Q 保存 P 的现场，并为自己的非静态局部变量分配空间 4）执行 Q 的过程体（函数体）5）Q 恢复 P 的现场，并释放局部变量所占空间 6）Q 取出返回地址，将控制转移到 P
按值传递参数和按地址传递	40. (1)按值传递：形参是基本类型变量名时,采用按值传递方式(2)按地址传递：形参是指针类型变量名或构造类型变量名时，采用按地址传递

do～whILe 循环的机器级表示	41.
	C 语言中的do～while语句形式如下。
	do{
	Loop body-statement ;
	}whlle (cond_expr)
	该循环结构的执行过程可以用以下更接近机器级语言的低级行为来描述。
	loop : Loop_ body_statement
	c=cond_expr ;
	if (c) goto loop ;
	上述结构对应的机器级代码中，loop_body_statement用一个指令序列来完成，然后用一个指令序列实现对
	cond_expr 的计算，并将计算或比较的结果记录在标志寄存器中，然后用一条条件跳转指令来实现 "if (c) gotoloop ; "
	的功能。

C 语言中的 for 语句形式	42.
	for (begin_expr ; cond_expr ; update_expr)
	loop_body_statement ;
	for 循环结构的执行过程大多可以用以下更接近于机器级语言的低级行为来描述。
	begin_expr ;
	c=cond_expr ;
	if (!c) goto done ;
	loop : loop_body_statement
	update_expr ;
	c=cond_expr ;
	if (c) goto loop ;
	done :
	从上述结构可看出，与while循环结构相比，for循环仅在两个地方多了一段指令序列。一个是开头多了一段循环
	变量赋初值的指令序列，另一个是循环体中多了更新循环变量值的指令序列，其余地方与while语句一样。

第四章 可执行文件的生成与加载执行

知识点名称	内容
预处理，编译和汇编	1. 预处理：是从源程序变成可执行文件的第一步，C 预处理程序为 cpp（即 CPreprocessor）主要用于 C 语言编译器对各种预处理命令进行处理，包括对头文件的包含、宏定义的扩展、条件编译的选择等 2. C 编译器在进行具体的程序翻译前，会先对源程序进行词法分析、语法分析和语义分析，然后根据分析的结果进行代码优化和存储分配，最终把 C 语言源程序翻译成汇编语言程序。编译器通常采用对源程序进行多次扫描的方式进行处理，每次扫描集中完成一项或几项任务，也可以将一项任务分散到几次扫描去完成。例如，可以按照以下四趟扫描进行处理：第一趟扫描进行

	词法分析；第二趟扫描进行语法分析；第三趟扫描进行代码优化和存储分配；第四趟扫描生成代码。GCC 可以直接产生机器语言代码，也可以先产生汇编语言代码，然后再通过汇编程序将汇编语言代码转换为机器语言代码。GCC 中的编译命令是“gcc-S”或“ccl”， 3. 汇编的功能是将编译生成的汇编语言代码转换为机器语言代码。因为通常最终的可执行文件由多个不同模块对应的机器语言目标代码组合而成，所以，在生成单个模块的机器语言目标代码时，不可能确定每条指令或每个数据最终的地址，即单个模块的机器语言目标代码需要重新定位，因此，通常把汇编生成的机器语言目标文件称为可重定位目标文件。GCC 中的汇编命令是“gcc-c”或“as”命令
符号解析	4. 符号解析的目的是将每个符号的引用与一个确定的符号定义建立关联。符号包括全局静态变量名和函数名，而非静态局部变量名则不是符号
ELF 目标文件格式	5. 目标代码（ObjectCode）指编译器或汇编器处理源代码后所生成的机器语言目标代码 6. 目标文件（ObjectFile）指存放目标代码的文件
可重定位目标文件格式	7. ELF 头：ELF 头位于目标文件的起始位置，包含文件结构说明信息。ELF 头的数据结构分 32 位系统对应结构和 64 位系统对应结构。 8. 节：节是 ELF 文件中的主体信息，包含了链接过程所用的目标代码信息，包括指令、数据符号表和重定位信息等 9. 节头表：节头表由若干个表项组成，每个表项描述相应节的节名、在文件中的偏移、大小、访问属性、对齐方式等，目标文件中的每个节都有一个表项与之对应。除 ELF 头之外，节头表是 ELF 可重定位目标文件中最重要的一部分内容
符号表	10. 目标文件中有一个符号表。表中包含了在程序模块中定义的所有符号的相关信息。对于某个 C 程序模块 m 来说。包含在符号表中的符号有以下三种不同类型：1）在 m 中定义并被其他模块引用的全局符号（GlobalSymbol）。这类符号包括非静态的函数名和全局变量名 2）由其他模块定义并被 m 引用的全局符号，称为 m 的外部符号（ExternalSymbol）包括在 m 中引用的在其他模块定义的外部函数名和外部变量名 3)在 m 中定义并在 m 中引用的本地符号(LocalSymbol)。这类符号包括带 static 属性的函数名和全局变量名。这类在模块内部定义的带 static 属性的本地变量不在栈中管理，而是分配在静态数据区。即编译器为它们在节.data 或.bss 中分配空间。如果在 m 内有两个函数使用了同名 static 本地变量，则需要为这两个变量都分配空间，并作为两个不同的符号记录到符号表中 11. 注意，上述三类符号不包括分配在栈中的非静态局部变量（auto 变量）链接器不需要这类变量的信息，因而它们不包含在由节.symtab 定义的符号表中
全局符号的解析规则	12. 编译器在对源程序编译时，会把每个全局符号的定义输出到汇编代码文件中，汇编器通过对汇编代码文件的处理，在可重定位文件的符号表中记录全局符号的特性，以供链接时全局符号的符号解析所用。一个全局符号可能是函数，或者是.data 节中具有特定初始值的全局变量，或者是.bss 节中被初始化为 0 的全局变量，或者是说明为 COMMON 伪节的未初始化全局变量（即 COMMON 符号），还可能是绑定属性为 WEAK 的弱符号 13. 为便于说明全局符号的多重定义问题，本书将前三类全局符号（即函数、.data 节和.bss 节中的全局变量）统称为强符号 14. 规则 1：强符号不能多次定义，否重链接错误。规则 2：若出现一次强符号定义和多次 COMMON

	符号和弱符号定义，则按强符号定义为准。规则 3：若同时出现 COMMON 符号定义和弱符号定义，则按 COMMON 符号定义为准。规则 4：若一个 COMMON 符号出现多次定义。则以其中占空间最大的一个为准。因为符号表中仅记录 COMMON 符号的最小长度，而不会记录变量的类型，因此在链接器确定多重 COMMON 符号的唯一定义时，以最小长度中的最大值为准进行符号解析.能够保证满足所有同名 COMMON 符号的空间要求。规则 5：若使用编译选项-fno-common，则不考虑 COMMON 符号，相当于将 COMMON 符号作为强符号处理
重定位的工作	15. 节和定义符号的重定位：链接器将相互关联的所有可重定位文件中相同类型的节合并，生成一个同一类型的新节。并根据合并后的新节在虚拟地址空间中的起始位置以及新节中定义的每个符号的位置，确定每个符号的存储地址。例如，将所有模块中的.data 节合并后作为可执行文件中的.data 节，并重新确定其中每个定义符号在虚拟地址空间中的位置 16. 引用处符号的重定位：链接器对合并后的新代码节（.text）和新数据节（.data）中所有符号引用处进行重定位，使其指向对应的定义符号的起始位置。为了实现这一步工作，链接器需要知道可重定位目标文件中存在哪些需要重定位的符号引用、所引用的是哪个定义符号等，这些称为重定位信息，放在重定位节（.rel.text 和.rel.data）的重定位表项中。重定位过程中，根据重定位节，rel.text 和.rel.data 中的重定位表项，分别对新的.text 节和.data 节中的符号引用进行重定位处理
共享库动态链接的特点	17. “共享性”是指共享库中的代码段在内存只有一个副本，当应用程序在其代码中需要引用共享库中的符号时，在引用处通过某种方式确定指向共享库中对应定义符号的地址即可 18. “动态性”是指共享库只在使用它的程序被加载或执行时才加载到内存，因而在共享库重新后并不需要重新对程序进行链接，每次加载或执行程序时所链接的共享库总是最新的
程序和进程的概念	19. 程序（Program）：就是代码和数据的集合，程序的代码是一个机器指令序列，因而程序是一种静态的概念，它可以作为文件存放在硬盘中。进程（Process）可以看成是程序的一次运行过程，因此进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动，因而进程具有动态的含义。计算机处理的所有任务实际上是由进程完成的。每个应用程序在系统中运行时均有属于它自己的存储空间，用来存储它自己的程序代码和数据，包括只读区（代码和只读数据）、可读/写数据区（初始化数据和未初始化数据）、动态的堆区和栈区等 20. 进程：是操作系统对处理器中程序运行过程的一种抽象。进程有自己的生命周期，它由于任务的启动而创建，随着任务的完成（或终止）而消亡，它所占用的资源也随着进程的终止而释放。一个可执行文件可以被多次加载执行，也就是说，一个程序可能对应多个不同的进程
CPU 的基本功能和组成	21. CPU 的基本职能是周而复始地执行指令，机器指令执行过程中的全部操作都是由 CPU 中的控制器控制执行的。 22. 随着超大规模集成电路技术的发展，更多的功能逻辑被集成到 CPU 芯片中，包括 cache、MMU、浮点运算逻辑、异常和中断处理逻辑等，因而 CPU 的内部组成越来越复杂，甚至可以在一个 CPU 芯片中集成多个处理器核。 23. 但是，不管 CPU 多复杂，它最基本的部件是数据通路（Datapath）和控制部件（ControlUnit）。控制部件根据每条指令功能的不同生成对数据通路的控制信号，并正确控制指令的执行流程 24. 1）程序计数器（PC）：PC 又称指令计数器或指令指针寄存器（IP）用来存放即将执行指令的地址。顺序执行时，PC+“1”形成下一条指令地址（这里的“1”是指一条指令的字节数）需

	要改变程序执行顺序时，CPU 根据跳转类指令提供的信息生成跳转目标指令的地址，并将其作为下一条指令地址送 PC。2）指令寄存器（IR）：IR 用以存放现行指令。上文提到，每条指令总是先从存储器取出后才能在 CPU 中执行，指令取出后存放在指令寄存器中，以便送指令译码器进行译码。3）指令译码器（ID）：ID 对 IR 中的操作码部分进行译码，产生的译码信号提供给操作控制信号形成部件，以产生控制信号。4）启停控制逻辑：脉冲源产生一定频率的脉冲信号作为 CPU 的时钟信号。启停控制逻辑在需要时能保证可靠地开放或封锁时钟信号，实现对机器的启动与停机。5）时序信号产生部件：该部件以时钟信号为基础，产生不同指令对应的时序信号，以实现机器指令执行过程的时序控制。6）操作控制信号形成部件：该部件综合时序信号、指令译码信号和执行部件反馈的条件标志（如 CF、SF、ZF 和 OF）等，形成不同指令操作所需要的控制信号。7）总线控制逻辑：实现对总线传输的控制，包括对数据和地址信息的缓冲与控制。CPU 对于存储器的访问通过总线进行，CPU 将存储访问命令（即读/写控制信号）送到控制线，将要访问的存储单元地址送到地址线，并通过数据线取指令或者与存储器交换数据信息。8）中断机构：实现对异常情况和外部中断请求的处理

第五章 程序的存储访问

知识点名称	内容
存储器概述	2. 按使用材料分类：半导体存储器，磁表面存储器，光介质存储器 3. 按信息可更改性分类：读写存储器，只读存储器 4. 按断电后可保存性分类：非易失性存储器（rom，磁盘），易失性存储器（主存，cache） 5. 随机存取存储器 RAM：只读存储器 ROM：都采用随机存取方式进行访问 6. 主存储器由 DRAM 芯片组成，Cache 由 SRAM 芯片组成 7. CPU 跟主存交换信息，主存跟外部辅助存储器交换信息
编址单位	8. 编址单位是指具有相同地址的那些位元构成的一个单位，可以是一个字节或一个字。对各存储单元进行编号的方式称为编址方式，可以按字节编址，也可以按字编址。现在大多数通用计算机都采用按字节编址方式，此时，存储体内一个地址中有一个字节。每一个存储单元由 8 个记忆单元组成，因此，是按字节编址方式
程序访问的局部性	9. CPU 和主存之间设置高速缓存（cache）也可以提高 CPU 访问指令和数据的速度。 10. 工作原理：程序访问的局部性大量结果表明，在较短时间间隔内，程序产生的地址往往集中在存储空间的一个很小范围，这种现象称为程序访问的局部性，可细分为时间局部性和空间局部性 11. 时间局部性是指被访问的某个存储单元在一个较短的时间间隔内很可能又被访问 12. 空间局部性是指被访问的某个存储单元的邻近单元在一个较短的时间间隔内很可能也被访问 13. 为了更好地利用程序访问的空间局部性，通常把当前访问单元以及邻近单元作为一个主存块一起调入 cache。这个主存块的大小以及程序对数组元素的访问顺序等都对程序的性能有一定的影响
访存指令	14. I 取数指令（Load）用于将存储单元内容装入 CPU 的寄存器中 15. I 存数指令（Store）用于将 CPU 寄存器的内容存储到存储单元中
cache 的有效位	16. 在系统启动或复位时，每个 cache 行都为空，其中的信息无效，只有装入了主存块后信息才有效。为了说明 cache 行中的信息是否有效，每个 cache 行需要一个有效位（ValidBit）有了有

	效位，就可通过将有效位清 0 来淘汰某 cache 行中的主存块，称为冲刷（Flush）装入一个新主存块时，再将有效位置 1
cache 行和主存块之间的映射方式	17. 主存块和 cache 行之间必须遵循一定的映射规则，这样，CPU 要访问某个主存单元时，可以依据映射规则，到 cache 对应的行中查找要访问的信息 18. 根据不同的映射规则，主存块和 cache 行之间有以下三种映射方式：（1）直接映射：每个主存块映射到 cache 的固定行中（2）全相联映射：每个主存块映射到 cache 的任意行中（3）组相联映射：每个主存块映射到 cache 的固定组的任意行中
cache 中主存块的替换算法	19. cache 行数比主存块数少得多，因此，往往多个主存块会映射到同一个 cache 行中。当新的一个主存块复制到 cache 时，cache 中的对应行可能已经全部被占满，此时，必须选择淘汰掉一个 cache 行中的主存块 20. （1）先进先出算法（FIFO）的基本思想是：总是选择最早装入 cache 的主存块被替换掉。这种算法实现起来较方便，但不能正确反映程序的访问局部性，因为最先进入的主存块也可能是目前经常要用的，因此，这种算法有可能产生较大的缺失率（2）最近最少用算法（LRU）的基本思想是：总是选择近期最少使用的主存块被替换掉。这种算法能比较正确地反映程序的访问局部性，因为当前最少使用的块一般来说也是将来最少被访问的。但是，它的实现比 FIFO 算法要复杂一些。LRU 算法用计数值来记录主存块的使用情况，通过硬件修改计数值，并根据计数值选择淘汰某个 cache 行中的主存块。这个计数值称为 LRU 位，其位数与 cache 组大小有关（3）最不经常用算法（LFU）的基本思想：是替换掉 cache 中引用次数最少的块。LFU 也用与每个行相关的计数器来实现。这种算法与 LRU 有点类似，但不完全相同（4）随机替换算法：从候选行的主存块中随机选取一个淘汰掉，与使用情况无关。模拟试验表明，随机替换算法在性能上只稍逊于基于使用情况的算法，而且代价低
cache 的写策略	21. 全写法：全写法的基本做法是：若写命中，则同时写 cache 和主存；若写不命中，则有以下两种处理方式。（1）写分配法：先在主存块中更新相应存储单元，然后分配一个 cache 行，将更新后的主存块装入到分配的 cache 行中。这种方式可以充分利用空间局部性，但每次写不命中都要从主存读一个块到 cache 中，增加了读主存块的开销（2）非写分配法：仅更新主存单元而不装入主存块到 cache 中。这种方式可以减少读入主存块的时间，但没有很好利用空间局部性。 22. 回写法：回写法的基本做法是：若写命中，则信息只被写入 cache 而不被写入主存；若写不命中，则在 cache 中分配一行，将主存块调入该 cache 行中并更新相应单元的内容。该方式下在写不命中时，通常采用写分配法进行写操作。由此可见，该方式实际上采用的是回头再写或最后一次性写的做法，因此，该方式通常被称为回写法或一次性写方式，也有教材称之为写回法。写操作时，回写法不会更新主存单元，只有当 cache 行中的主存块被替换时，才将该块内容一次性写回主存。这种方式的好处在于减少了写主存的次数，因而大大降低了主存带宽带求。为了减少写回主存块的开销，每个 cache 行设置了一个修改位（有时也称为“脏位”）若修改位为 1、则说明对应 cache 行中的主存块被修改过，替换时需要写回主存，若修改位为 0，则说明对应主存块未被修改过，替换时无需写回主存。由于回写法没有同步更新 cache 和主存内容，所以存在 cache 和主存内容不一致而常来的潜在隐患。通常需要其他的同步机制来保证存储信息的一致性

长期股权投资后续计量	23. 期股权投资的可收回金额：每年年末，企业应对期股权投资的账面价值进行检查。如果出现减值迹象，应对其可收回金额进行估计。可收回金额应当根据期股权投资的公允价值减去处置费用后的净额与期股权投资预计未来现金流量的现值两者之间较高者确定 24. 期股权投资减值损失的确认：如果期股权投资可收回金额的计量结果表明其可收回金额低于其账面价值，说明期股权投资已发生减值，应当将其账面价值减记至可收回金额，借：资产减值损失，贷：期股权投资减值准备，期股权投资减值损失一经确认，在以后期间不得转回

第六章 程序中的 I/O 操作实现

知识点名称	内容
I/O 子系统的特性	2. 共享性：I/O 子系统被多个进程共享，因此必须由操作系统对共享的 I/O 资源统一调度管理，以保证用户程序只能访问自己有权访问的那部分/0 设备或文件，并使系统的吞吐率达到最佳 3. 复杂性：I/O 设备控制的细节比较复杂，如果由最上层的用户程序直接控制，会给广大的应用程序开发人员带来麻烦，因而需操作系统提供专门的驱动程序进行控制，这样可以对应用程序开发人员屏蔽设备控制的细节，简化应用程序的开发 4. 异步性：I/O 子系统的速度较慢，而且不同设备之间的速度也相差较大，因而，I/O 设备与主机之间的信息交换方式通常使用异步的中断 I/O 方式，中断导致从用户态向内核态转移，因此，I/O 处理须在内核态完成，通常由操作系统提供中断服务程序来处理 I/O
文件的基本概念	5. 创建文件：通常，用户程序在读写一个文件前，必须告知系统将要对该文件进行何种操作，是读、写、添加还是可读可写，该告知操作通过打开或创建一个文件来实现。可以直接打开一个已存在的文件，若文件不存在，则应先创建。创建一个新文件时，用户应指定其文件名和访问权限，系统将返回一个非负整数，称为文件描述符（FileDescriptor）文件描述符是进程中被打开文件的唯一标识，可用于后续读/写等操作 6. 打开文件：打开文件时，系统会检测文件是否存在、用户是否有访问权限等。若成功，则系统会返回一个非负整数作为文件描述符。创建每个进程时，都会预先打开三个标准文件：标准输入（描述符为 0）、标准输出（描述符为 1）和标准错误（描述符为 2）键盘和显示器可以分别抽象成标准输入文件和标准输出文件 7. 设置文件读/写位置：每个文件都有一个当前读/写位置，表示相对于文件最开始处的字节偏移量，初始时为 0。用户程序中可通过系统调用封装函数 lseek（）设置文件读/写位置 8. 读文件和写文件：用户程序可以向被创建的新文件中写入信息，也可以从一个已存在且打开后的文件中读或写信息。写文件操作将从当前读/写位置 k（k≥0）处写入 n（n>0）个字节，因而写入后文件当前读/写位置为 k+n。读文件操作将从文件当前读/写位置 k（k≥0）处读出 n（n>0）个字节，因而读出后文件当前读/写位置为 k+n。假设文件大小为 m 字节，若执行读文件操作时 k=m，则当前位置为结尾处，这种情况称为文件结束（EndOfFile，EOF） 9. 关闭文件：完成文件读/写等操作后，用户程序需要通知系统关闭文件，表示用户程序不再对该文件进行任何操作。关闭文件时，系统将释放文件创建或打开相关的数据结构所在存储区，并回收文件描述符。无论一个进程因为何种原因终止，系统都会关闭其打开的所有文件，以释放相应的存储资源
系统级 I/O 函数	10. （1）creat 函数（2）open 函数（3）read 函数（4）write 函数（5）lseek 函数（6）stat/fstat 函数（7）close 函数

输出缓冲区的属性	11. 全缓冲的含义是，即使遇到换行符也不会写文件，只有当缓冲区满时才会将缓冲区内容真正写入文件 fd 中 12. 行缓冲的含义是，遇到换行符或者缓冲区满就将缓冲区内容写文件 fd 13. 非缓冲的含义是直接写到文件 fd 中。普通文件的缓冲区属性为全缓冲。 14. 对于全缓冲属性，每次执行写操作时，先判断当前缓冲区是否已写满（即 cnt=0）对于行缓冲属性，则判断本次写的字节流中是否有换行符\n 或者是否缓冲区已满。若是，则将缓冲区信息一次性写到文件 fd 中，并置 ptr 等于 base，cnt 等于 1024。若写入缓冲区 n 字节，则新的 ptr 等于 ptr 加 n，cnt 等于 cnt 减 n
缓存层	15. I/O 设备的工作速度较慢，为了提升 I/O 请求的处理效率，操作系统充分利用数据访问的局部性特点，在内核空间对应主存区中开辟一块空间作为高速缓存，用于存储最近访问的文件数据。作为高速缓存的主存 RAM 区可称为高速缓存 RAM 16. 传统的外部存储器是磁盘，因此上述高速缓存也称为磁盘高速缓存（DiskCache） 17. 虚拟文件系统首先检查用户请求访问的数据是否在该缓存中，若是，则直接访问缓存，无须通过 I/O 请求访问外存中的数据；否则调用逻辑文件系统提供的功能，将该请求翻译成访问外存中若干存储块的 I/O 请求，并提交到通用块设备 I/O 层进行后续处理。缓存中存放的信息包括写入文件的数据、从磁盘读出的磁盘块等信息，缓存通常采用回写策略，操作系统每隔一段时间将缓存内容真正写人设备中，以保证数据的永久存储。 18. 有了磁盘高速缓存，磁盘读/写次数可大幅减少，用户的 I/O 请求能得到快速响应
中断控制 I/O 方式	19. 中断控制 I/O 方式的基本思想是，当需要进行 I/O 操作时，首先启动外设进行第一个数据的 I/O 操作，然后阻塞请求 I/O 的用户进程，并调度其他进程到 CPU 上执行，期间，外设在设备控制器的控制下工作。外设完成 I/O 操作后。向 CPU 发送一个中断请求信号，CPU 检测到该信号后，则进行上下文切换，调出相应的中断服务程序执行。中断服务程序将启动后续数据的 I/O 操作，然后返回到被打断的进程继续执行
中断服务程序	20. 准备阶段：需要将寄存现场保存到栈上，并根据实际情况决定是否需要在中断处理过程中响应并处理其他中断，若是、则进行以下操作：①保存当前的中断屏蔽字，中断屏蔽字用于指示是否允许响应新的中断源：②设置新的中断屏蔽字，从而指定允许在后续的处理阶段中响应哪些中断源；③开中断，允许 CPU 响应中断。若否，则可省略上述三步操作。中断屏蔽字寄存器是中断控制器中的一个 I/O 端口，由 CPU 通过 I/O 指令进行设置。 21. 处理阶段：需要从中断控制器中读出触发本次中断的中断号并根据中断号查询相应的中断服务程序然后调用具体的中断服务。具体的中断服务首先通知设备本次中断已收到。清除中断请求，然后根据设备的具体功能进行处理。由于具体的中断服务与设备紧密相关。因此通常作为设备驱动程序的一部分功能，设备驱动程序在初始化时会向操作系统注册相应的中断服务，同时将中断号作为参数，指示将该中断服务绑定到该中断号 22. 恢复阶段：的工作与准备阶段相反，包括关中断、恢复现场和旧屏蔽字等。最后通过指令集提供的中断返回指令从中断处理过程返回。通常中断返回指令除了返回到程序的断点外同时还会自动恢复处理器的中断使能
输入/输出设备	23. I/O 设备又称外围设备、外部设备，简称外设，是计算机系统与人类或其他计算机系统之间交换信息的装置。操作系统为了统一管理 I/O 设备，通常将 I/O 设备分成两类：字符设备和块设备。

	24. 字符设备是以字符为单位向主机发送或从主机接收字符流的设备。字符设备传送的字符流不能形成数据块，无法定位和寻址 25. 块设备以一个固定大小的数据块为单位与主机交换信息。块设备中的数据块大小通常在 512 字节以上，通常按照某种组织方式对其进行读/写，每个数据块都有唯一的位置信息，因而是可寻址的
中断系统	26. 现代计算机系统的中断处理功能相当丰富，每个计算机系统的中断系统功能可能不完全相同，但其基本功能主要包括以下几个方面：①及时记录各种中断请求，通常用一个中断请求寄存器来记录②自动响应中断请求。CPU 在“开中断”状态下，执行一条指令后会自动检测中断请求引脚，发现有中断请求后会自动响应中断③同时有多个中断请求时，能自动选择并响应优先级最高的中断请求④保护被打断程序的断点和现场。断点指被打断程序中将要执行的下一条指令的地址，由 CPU 保存，现场指被打断程序在断点处各通用寄存器的内容，由中断服务程序保存⑤通过中断屏蔽实现多重中断的嵌套执行