chap1

中央处理器简称 CPU 或处理器，是整个计算机的核心部件，主要用于指令的执行。

CPU 主要包含两种基本部件：数据通路和控制器。

数据通路主要包含算术逻辑部件(arithmetic and logic unit,ALU)和通用寄存器等

ALU 由一系列的数字逻辑电路组成，如门电路、触发器等，这些电路协同工作，执行复杂的算术和逻辑运算。

通用寄存器用来暂存指令所用的操作数或执行结果

控制器用来对指令进行译码，生成相应的控制信号，以控制数据通路进行正确的操作

寄存器堆(有些教材称为寄存器文件 register file 或通用寄存器组 GPRs) 通用寄存器组是 CPU 内部的高速存储单元，它们设计用于临时存储数据和在指令执行期间的计算结果。这些寄存器之所以被称为“通用”，是因为它们可以用于多种不同的目的，包括但不限于数据传送、运算操作、地址存储、指针和索引，以及函数调用和栈管理。

程序计数器（Program Counter，简称 PC）是计算机处理器中的寄存器，它包含当前正在执行的指令的地址（位置）

指令寄存器组是 CPU 内部用于暂存当前正在执行或即将执行的指令的存储单元。

主存储器用来存储指令和操作数。每个主存单元有一个编号，称为主存地址。通常一个主存单元存放一个字节(8 位),因此，每条指令可能占用多个主存单元，一个操作数也可能占用多个主存单元，指令的地址和操作数的地址都是指一组连续主存单元中最小的地址。

总线(bus)是传输信息的介质，用于在部件之间传输信息。CPU、主存和 I/O 模块通过总线互连，在 CPU 和 I/O 模块中都内含相应的存储部件，即各类寄存器或缓存器。

应用软件是为了满足用户不同领域、不同问题的应用需求而提供的软件部分。它拓宽了计算机系统的应用领域，放大了硬件的功能。

系统软件是计算机系统中不可或缺的一部分，它控制和协调计算机及外部设备，支持应用软件的运行。主要功能包括调度、监控和维护计算机系统，以及管理计算机系统中各种独立的硬件，使得它们可以协调工作。系统软件使得计算机使用者和其他软件将计算机当作一个整体而不需要顾及到底层每个硬件是如何工作的。

高级编程语言是一种使用接近于自然语言的语法和命令的编程语言，它允许程序员通过抽象的构造和功能来完成各种计算任务。

人们引入了一种机器语言的符号表示语言，通过用简短的英文符号和二进制代码建立对应关系，以方便程序员编写和阅读，这种语言称为汇编语言(assembly language)。

机器语言(machine language)是用二进制进行编码的机器指令(instruction),每条机器指令都是一个 0/1 序列。机器语言程序的可读性很差，也不易记忆，给程序员的编写和阅读带来极大的困难。

低级语言(low-level language)则和运行程序的计算机的底层结构密切相关，通常称为机器级语言，机器语言和汇编语言都是机器级语言。

源程序（Source Code）是指未编译的按照一定的程序设计语言规范书写的文本文件，是一系列人类可读的计算机语言指令。

目标程序，又称为“目的程序”，是源程序经过编译或汇编后生成的，能够被计算机直接识别和执行的机器语言代码。

翻译程序有以下 3 类

汇编程序(assembler)也称汇编器，用来将汇编语言源程序翻译成机器语言目标程序。

解释程序(interpreter)也称解释器，用来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。

编译程序(compiler)也称编译器，用来将高级语言源程序翻译成汇编语言或机器语言目

标程序。

语言处理程序是用户设计的编程服务软件，其主要功能是将高级语言源程序翻译成计算机能识别的目标程序。这一转换过程是通过翻译程序来完成的，而翻译程序统称为语言处理程序。

最终用户，也称为“终端用户”，是系统中系统所服务的最终受益者，通常用于区分参与产品或服务设计、开发和生产阶段的个人。

系统管理员（System Administrator，常简称为 sysadmin 或 admin）是负责维护和管理计算机系统、网络和相关技术设施的专业人员。

应用程序员，通常简称为“程序员”或“码农”，是负责编写、测试、调试和维护计算机应用程序的专业人员。

系统程序员是专门从事系统级软件开发、维护、优化和管理的专业人员。

指令集体系结构（ISA）是软件和硬件之间接口的一个完整定义。它定义了计算机可以执行的所有指令的集合，以及这些指令如何与计算机硬件进行交互。

微体系结构决定了处理器中各种部件，如指令集、地址总线、数据总线、控制单元等，如何被设计、分配和使用。它是计算机系统硬件设计的基础，是计算机体系结构的实现，并在将高级语言的指令翻译成底层硬件操作中发挥关键作用。

ISA 是对指令系统的一种规定或结构规范，而具体实现的组织(organization)称为微体系结构(microarchitecture),简称微架构

透明性是指在计算机系统中，某些事物或属性在某种特定的视角下看似不存在或不可见的状态。这种状态使得用户或程序员在关注某一层次或功能时，无需关心其底层的实现细节，从而简化了系统设计和使用的复杂性。

吞吐率表示在单位时间内所完成的工作量。

响应时间也称为执行时间(execution time)或等待时间(latency),是指从作业提交开始到作业完成所用的时间。

用户 CPU 时间指真正用于用户程序代码的执行时间;

其他时间包括 CPU 运行操作系统程序的时间以及等待 I/O 操作完成的时间或 CPU 用于其他用户程序的执行时间

系统性能是指系统的响应时间，它与 CPU 性能相关，同时也与 CPU 以外的其他部分有关

CPU 性能是指用户 CPU 时间，它只包含 CPU 运行用户程序代码时的执行时间

时钟周期(clock cycle,tick,clock tick,clock)。计算机执行一条指令的过程被分成若干步骤来完成，每一步都要有相应的控制信号进行控制，这些控制信号何时发出、作用时间多长，都要有相应的定时信号进行同步。因此，计算机必须能够产生同步的时钟定时信号，也就是 CPU 的主脉冲信号，其宽度称为时钟周期

CPU 的主频就是主脉冲信号的时钟频率，它是 CPU 时钟周期的倒数。

CPI 表示执行一条指令所需的时钟周期数。

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。

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条指令。

MFLOPS (million floating point operations per second)。它表示每秒钟所执行的浮点运算有多少万次，它是基于所完成的操作次数而不是指令数来衡量的。GFLOPS(10°次/s)、TFLOPS(10 次/s)、PFLOPS(10 次/s)和 EFLOPS(10*次/s)等

chap2

通常将在计算机内部编码表示后的数称为机器数

机器数真正的值(即现实世界中带有正负号的数)称为机器数的真值

数值数据主要是带符号整数和浮点数。数值型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，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。

非数值数据（Non-Numerical Data）指的是不能以数值方式表示的数据，逻辑值、字符等数据都是非数值数据，

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。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。用于地址计算等，范围较大

带符号整数也称为有符号整数，它必须用一个二进位来表示符号 p.33 补码表示的好处

定点数是指在计算机运算过程中，数据中小数点的位置固定不变。这种固定性是由计算机设计者在机器的结构中指定一个不变的位置来实现的，而不是通过专门的器件来表示小数点。

源码（或原码）是数值按照绝对值大小转换成的二进制数，并用一位符号位来表示正负。正数的符号位为 0，负数的符号位为 1。其余位则直接表示该数的绝对值。

补码是计算机系统中最为常用的有符号数表示方法。正数的补码与其源码和反码相同；负数的补码是在其反码的基础上加 1。补码的设计是为了简化计算机中的加法和减法运算。

补码表示法也称“2-补码”(two's complement)表示法，由符号位后跟上

真值的模 2”补码构成

负数的补码可采用“各位取反，末位加 1”的方法得到

反码是正数和负数的另一种二进制表示方法。对于正数，反码就是其源码；对于负数，反码是在源码的基础上，除符号位外，其余各位取反（0 变 1，1 变 0）。

为了便于判断运算结果是否溢出，某些计算机中还采用了一种双符号位的补码表示方式，称为变形补码，也称为模 4-补码。在双符号位中，左符是真正的符号位，右符用于判断“溢出”

浮点数是属于有理数中某特定子集的数的数字表示，它在计算机中用以近似表示任意某个实数。这个实数由一个整数或定点数（即尾数）乘以某个基数（在计算机中通常是 2）的整数次幂得到，这种表示方法类似于基数为 10 的科学计数法。

任意一个浮点数都可以用一个定点小数和一个定点整数来表示

尾数，又称为有效数（Significand）或小数部分，是浮点数中表示数字实际大小（精度）的部分。在浮点数表示法中，尾数与指数（Exponent）一起使用，以科学计数法的形式表示实数。

阶码（Excess Code），也称为偏移量或溢出码，是一种用来表示有符号整数的编码方式

一般情况下，浮点数的阶用一种称之为“移码”的编码表示。通常，将阶的编码表示称为阶码

移码（又称增码或偏置码）通常用于表示浮点数的阶码，定义：设由 1 位符号位和 n 位数值位组成的阶码，则移码表示为 $[E]_{移} = 2^n + X$

假设用来表示阶 E 的移码的位数为 n,则[E]=偏置常数+E,通常，偏置常数取 2^{n-1}或 $2^{n-1}-1$

溢出通常指的是算术溢出，即计算机进行算术运算产生的结果超出了机器所能表示的

范围。

下溢则是指运算结果低于了负方向的最小值或接近于零的值，例如两个负数相加，若结果为正，则称为下溢（即计算结果小于机器所能表示的最小负数）。

上溢是指运算结果超过了正方向的最大值，例如两个正数相加，若结果为负，则称为上溢（即计算结果大于机器所能表示的最大正数）

从理论上来讲，规格化数的标志是真值的尾数部分中最高位具有非零数字。也就是说，若基数为 R,则规格化数的标志是，尾数部分真值的绝对值大于或等于 1/R。若浮点数的基数为 2,则尾数规格化的浮点数形式应为±0.1bb…b×2(这里 b 是 0 或 1)。

经过规格化的浮点数被称为规格化数。为了在尾数中表示最多的有效数据位，同时使浮点数具有固定的表示方式，浮点数的编码应当采用一定的规范。

规格化操作有两种：“左规”和“右规”。当有效数位进到小数点前面时，需要进行右规。右规时，尾数每右移一位，阶码加 1,直到尾数变成规格化形式为止，右规时指数会增加，因此有可能溢出；当出现形如±0.0…0bb…b×2F 的运算结果时，需要进行左规，左规时，尾数每左移一位，阶码减 1,直到尾数变成规格化形式为止。

非规格化数的特点是阶码为全 0,尾数高位有一个或几个连续的 0,但不全为 0。因此非规格化数的隐藏位为 0,并且单精度和双精度浮点数的指数分别为一 126 或—1022,故数值分别为(一 1)'×0.f×2~126 和(一 1)'×0.f×2~1。非规格化数可用于处理阶码下溢，使得出现比最小规格化数还小的数时程序也能继续进行下去。

机器零在数轴上表示为 0 点及其附近的一段区域。即在计算机中，小到机器数的精度达不到的数均被视为“机器零”，这与真值零（即数学上严格的 0 点）有所区别。

NaN 是计算机科学中数值数据类型的一类值，表示未定义或不可表示的值。它常在浮点数运算中使用，以指示某些运算结果无法用正常的数值表示。

在某些情况下需要将一个 n 位数据看成是由 n 个 1 位数据组成，每个取值为 0 或 1。例如，有时需要存储一个二进制数据阵列，阵列中的每项只能取值 1 或 0;有时可能需要提取一个数据项中的某位进行诸如“置位”或“清 0”等操作。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。逻辑数据只能进行按位逻辑运算，如按位“与”、按位“或”、逻辑左移、逻辑右移等。

在计算机中，ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）是一种字符编码标准，用于电子通信。ASCII 码定义了 128 个特定的字符，每个字符都有一个唯一的数字代码与之对应。这些字符包括英文字母（大写和小写）、数字（0-9）、标点符号和一些控制字符。

汉字输入码，也称为汉字外码或中文输入法编码，是为将汉字输入到计算机而设计的代码。由于汉字数量庞大且字形复杂，无法直接通过西文键盘上的键来一一对应输入，因此需要通过特定的编码方式，将汉字转换为计算机能够识别和处理的代码。

汉字内码，又称“汉字机内码”或“汉字 ASCII 码”，简称“内码”，是计算机内部存储、处理加工和传输汉字时所用的由 0 和 1 符号组成的代码。当汉字输入计算机后，它会被转换为内码以便进行存储、处理和传输。

汉字内码不能有二义性，即不能和 ASCII 码有相同的编码，要与汉字在字库中的位置有关系，以便于汉字的处理、查找，编码应尽量短

机器字长是指计算机进行一次整数运算所能处理的二进制数据的位数（整数运算即定点整数运算）。具体来说，它反映了计算机运算器进行定点数运算的字长，同时也是 CPU 内部数据通道的宽度。

大端方式将数据的最高有效字节 MSB 存放在低地址单元中，即数据的地址就是 MSB 所在的地址。IBM360/370、Motorola 68k、MIPS、Sparc、HP PA 等机器都采用大端方式。

小端方式将数据的最低有效字节 LSB 存放在低地址单元中，即数据的地址就是 LSB 所在的地址。Intel 80x86、DEC VAX 等都采用小端方式

最高有效字节，在多字节序列中，具有最大权重的字节。在二进制数中，它通常位于最左边，代表数值的最高位部分。

最低有效字节（LSB），在多字节序列中，具有最小权重的字节。在二进制数中，它通常位于最右边，代表数值的最低位部分

chap3

ALU 由一系列的数字逻辑电路组成，如门电路、触发器等，这些电路协同工作，执行复杂的算术和逻辑运算。

行波进位加法器是一种基于行波进位思想的数字电路，用于将两个二进制数字相加。它的特点在于，每一位的进位都是由上一位的进位和当前位的输入信号共同决定的，这种行波进位的传递方式有效地减少了延迟，提高了运算速度。

零标志（Zero Flag）是状态标志寄存器（如 x86 架构中的 EFLAGS 或 RFLAGS 寄存器）中的一位，用于表示某个算术或逻辑操作的结果是否为零。零标志通常在算术逻辑单元（ALU）执行指令后设置或清除。

溢出标志（OF）是处理器状态寄存器（如 x86 架构中的 EFLAGS 或 RFLAGS 寄存器）中的一位。当执行有符号整数运算（如加法、减法、乘法等）时，如果结果超出了目标数据类型能够表示的范围，那么 OF 标志就会被置位（设置为 1）。

进位/借位标志位（CF）是处理器状态寄存器（如 x86 架构中的 EFLAGS 或 RFLAGS 寄存器）中的一位。当执行算术运算（如加法、减法、移位等）时，如果运算结果的最高位（对于字节运算来说是第 8 位，对于字运算来说是第 16 位，以此类推）产生了进位或借位，CF 标志就会被置位（设置为 1）。否则，CF 标志保持为 0。

符号标志（Sign Flag, SF）：用于指示有符号数运算结果的符号（正或负）。

Booth 提出了一种补码相乘算法，可以将符号位与数值位合在一起参与运算，直接得出用补码表示的乘积，且正数和负数同等对待。这种算法被称为布斯(Booth)乘法，该算法的核心思想是通过移位和加法操作来实现乘法，具有简单、易于实现的优点

基 4 布斯算法（Bi-quaternary Booth's Algorithm）是一种优化的乘法算法，主要用于数字信号处理中的乘法器，其核心思想是将四进制数字转换为精心设计的二进制字符串，以简化乘法操作，提高计算速度。

阵列乘法器的基本原理是将两个数字分解成一组二进制数，并将每个数的每个位相乘。这些乘积被组合在一起，并以正确的顺序相加，以产生最终的乘积。阵列乘法器通常由多个阵列单元构成，每个单元都包含一组乘法器，可以同时执行多个位的乘法。这些单元被排列在一个网络上，以便乘积可以在每个单元之间传递和组合。

对阶的目的是使 x 和 y 的阶码相等，以使尾数可以相加减。对阶的原则是：小阶向大阶看齐，阶小的那个数的尾数右移，右移的位数等于两个阶的差的绝对值。

舍入是一种数学和计算中的修约规则，用于在有限精度下表示实数。当实数的表示超出了计算机可以精确处理的范围时，舍入被用来找到最接近的可表示值。这通常涉及到删除或修改一些数字，以便用最少的位数来近似表示原始数值。

在浮点数的中间计算过程中，保护位位于右边多保留的两位中的首位。其主要作用是提高舍入精度。在浮点数的计算过程中，为了得到更精确的结果，通常会多保留几位小数进行运算，保护位就是这些额外保留的小数中的最高位。

IEEE 754 标准规定，浮点数运算的中间结果右边必须至少额外保留两位附加位。这两位附加位中，紧跟在浮点数尾数右边那一位为保护位或警戒位(guard),用以保护尾数右移的

位，紧跟保护位右边的是舍入位(round),左规时可以根据其值进行舍入。

在 IEEE754 标准中，为了更进一步提高计算精度，在保护位和舍入位后面还可以引入额外的一个数位，称为粘位(sticky),只要舍入位的右边有任何非 0 数字，粘位就被置 1;否则，粘位被置为 0。

规格化浮点数是指把一个浮点数按指定的格式进行转换，确保浮点数具有固定的表示方式，并在尾数中表示最多的有效数据位。

规格化操作有两种：“左规”和“右规”。当有效数位进到小数点前面时，需要进行右规。右规时，尾数每右移一位，阶码加 1,直到尾数变成规格化形式为止，右规时指数会增加，因此有可能溢出；当出现形如±0.0…0bb…b×2F 的运算结果时，需要进行左规，左规时，尾数每左移一位，阶码减 1,直到尾数变成规格化形式为止。

非规格化浮点数用于表示接近零的非常小的浮点数。与规格化浮点数不同，非规格化浮点数的尾数部分允许包含前导零，从而能够表示更小的数值。

若一个正指数超过了最大允许值(127 或 1023),则发生指数上溢

若一个负指数超过了最小允许值(一 149 或-1074),则发生指数下溢

chap4

指令是计算机执行某种操作的命令，是计算机运行的最小功能单位。

指令集体系结构（ISA）是软件和硬件之间接口的一个完整定义。它定义了计算机可以执行的所有指令的集合，以及这些指令如何与计算机硬件进行交互。

操作码是指令的核心部分，用于指明该指令所要完成的操作类型，如加法、减法、传送、移位、转移等。操作码的长度可以是固定的，也可以是变化的。

地址码用来指出该指令的源操作数的地址（一个或两个）、结果的地址以及下一条指令的地址。这里的“地址”可以是主存的地址，也可以是寄存器的地址，甚至可以是 I/O 设备的地址。

程序计数器（Program Counter，简称 PC）是计算机处理器中的寄存器，它包含当前正在执行的指令的地址（位置）

指令指针是一个用于存放下一条要执行指令的地址的寄存器。用于指示 CPU 从哪个地址开始读取和执行下一条指令。

立即寻址：在指令中直接给出操作数本身，这种操作数称为立即数。mov ax 1234H

直接寻址：指令中发给出的地址码是操作数的有效地址，这种地址称为直接地址或者绝对地址 mov ax,[1234H]

间接寻址：指令中给出的地址码是存放操作数有效地址的存储单元地址，由于速度慢，已经基本淘汰。inc[200]

寄存器间接寻址：指令中给出的地址码是一个寄存器编号，该寄存器中存放的是操作数的有效地址。如，Intel8086 指令“MOV AX,[BX]”

寄存器寻址方式也称为寄存器直接寻址方式，其指令中给出的地址码是操作数所在寄存器的编号。mov ax,bx

变址寻址方式主要用于对线性表之类的数组元素进行访问。采用变址寻址方式时，指令中的地址码字段 A 给出一个基准地址，相对于基准地址的偏移量在指令中明显或隐含地由变址寄存器 1 给出，地址 EA=(I)+A。 mov ax,[SI+1000] si/di

变址寄存器是指寄存器 ESI、EDI、SI 和 DI 的寄存器，它们主要用于存放存储单元在段内的偏移量。

基址寻址方式下，指令中的地址码字段 A 给出一个偏移量，基准地址可以明显或隐含地由基址寄存器 B 给出。操作数有效地址 EA=(B)+A。mov ax,[bx+1000] bx/bp

基址寄存器是一个特殊的地址寄存器，用于存放操作数或中间结果，以减少对存储器的访问次数。它也可以用于存放基地址，在基数位移定址系统中，基地址寄存器用于存放基地址。执行指令时，基地址与相对地址相加，从而得到实际的内存地址。

如果某指令的操作数的有效地址或转移目标地址位于该指令所在位置的前、后某个固定位置上，则该操作数或转移目标可用相对寻址方式。采用相对寻址方式时，指令中的地址码字段 A 给出一个偏移量，基准地址隐含由 PC 给出。即操作数有效地址或转移目标地址 EA=(PC)+A。这里的偏移量 A 是形式地址，有效地址或目标地址可以在当前指令之前或之后，因而偏移量 A 是一个带符号整数。相对寻址方式可用于实现共享代码的浮动或实现程序的跳转执行

通用寄存器用来暂存指令所用的操作数或执行结果

CISC（复杂指令集计算）是一种计算机指令集的设计策略，其指令系统包含了多种不同功能、不同操作数的复杂指令。这种设计允许处理器通过执行一条指令来完成复杂的操作，从而减少了程序所需的指令数量，但同时也增加了处理器设计的复杂性。所以一般采用微程序控制

RISC（精简指令集计算）是一种计算机指令集的设计策略，其指令系统只包含少量简单、通用的指令。这种设计通过简化指令集来降低处理器的复杂性，提高处理器的执行效率。一般采用硬布线方式

内部异常是指在计算机系统内部发生的异常事件或错误。当计算机执行指令或进行操作时，如果发生了不符合规范或意外的情况，就会引发内部异常。

外部中断是由 CPU 外部设备或事件触发的中断请求。当外部设备需要处理或响应时，会向 CPU 发送一个中断信号，请求 CPU 暂停当前的工作并转向处理该中断事件。

故障也称为失效，它是引起故障的指令在执行过程中被检测到的一类异常事件。

自陷也称为陷阱或陷入，与故障等其他异常事件不同，它是预先安排的一种“异常”事件，就像预先设定的“陷阱”一样。通常的做法是，事先在程序中用一条特殊指令或通过某种方式设定特殊控制标志来人为设置一个“陷阱”,当执行到满足条件的自陷指令时，CPU 自动根据不同“陷阱”类型进行相应的处理，自陷异常处理结束后，将返回到自陷指令的下条指令执行

中断请求（Interrupt ReQuest，IRQ）是指计算机硬件或软件向中央处理器（CPU）发出的一种信号，要求 CPU 暂停当前正在执行的程序，转而处理该请求所对应的事件或任务。这种“申请”过程，即为中断请求。

伪指令是用于对汇编过程进行控制的指令，它们并不是可执行指令，没有对应的机器代码，只存在于汇编语言中，用于在汇编过程中为汇编程序提供汇编信息。

叶子过程（Leaf Procedure）是指一个没有调用其他函数或过程，只会被其他函数或过程调用的函数或过程。简单来说，叶子过程就是程序执行树中的叶子节点，它们不会进一步展开（调用其他函数），而是直接执行具体的指令或操作。

栈指针寄存器（Extended Stack Pointer），简称为 ESP，是用来指示堆栈的最顶端位置的寄存器。在 x86 架构的计算机中，栈向下增长，即栈顶地址逐渐减小。ESP 寄存器通常用于入栈（PUSH）和出栈（POP）操作，以及在函数调用时保存和恢复栈顶指针。

帧指针寄存器（Extended Base Pointer），简称为 EBP，指向当前栈帧的底部。栈帧包含了函数的返回地址、参数、局部变量等信息。EBP 寄存器作为一个固定的参照物，使得可以通过它来定位栈帧中的其他数据。在函数调用期间，EBP 寄存器通常用来保存旧的栈底地址，以便在函数返回时能够正确地恢复栈的状态

栈帧是用于支持虚拟机进行方法调用和方法执行的数据结构。它存储了方法的局部变量表、操作数栈、动态连接和方法返回地址等信息。每一个方法从调用至执行完成的过程，

都对应着一个栈帧在虚拟机栈里从入栈到出栈的过程。

chap5

CPU 取出并执行一条指令的时间称为指令周期，不同指令的指令周期可能不同。

CPU 内部数据通路是指 CPU 内部的数据流经的路径以及路径上的部件，主要包括用于运算的 ALU 和用于存储中间结果的通用寄存器等，这些部件的宽度一致才能相互匹配。因此，字长等于 CPU 内部用于整数运算的运算器位数和通用寄存器宽度。

控制器也称为控制部件。控制器用来对指令进行译码，生成相应的控制信号，以控制数据通路进行正确的操作

通常把数据通路中专门进行数据运算的部件称为执行部件。指令执行所用到的元件有两类：组合逻辑元件(也称操作元件)和存储元件(也称状态元件)。连接这些元件的方式有两种：总线方式和分散连接方式。数据通路就是由操作元件和状态元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。

操作元件属于组合逻辑元件，其输出只取决于当前的输入。数据通路中最常用的操作元件有多路选择器(MUX)、加法器(Adder)、算术逻辑部件(ALU)等

状态元件属于时序逻辑电路，具有存储功能，输入状态在时钟控制下被写到电路中，并保持电路的输出值不变，直到下一个时钟到达。

在时钟下降沿到达前一段时间内，输入端 D 必须稳定有效，这段时间称为建立时间(setup time)。

在时钟下降沿到达后一段时间内，输入端 D 必须继续保持稳定不变，这段时间称为保持时间(hold time)

时钟周期(clock cycle,tick,clock tick,clock)。计算机执行一条指令的过程被分成若干步骤来完成，每一步都要有相应的控制信号进行控制，这些控制信号何时发出、作用时间多长，都要有相应的定时信号进行同步。因此，计算机必须能够产生同步的时钟定时信号，也就是 CPU 的主脉冲信号，其宽度称为时钟周期

边沿触发是一种信号处理技术，它基于输入信号的边缘（上升沿或下降沿）到达设定阈值时触发相应动作或事件的过程。与电平触发不同，边沿触发更关注信号变化的瞬间而非信号的绝对水平。

锁存延迟，也称为锁存时间（Latch Time），是指从输入信号变化到输出信号稳定反映这一变化所需的时间。在数字电路中，特别是在时序逻辑电路中，这一延迟是不可避免的，因为它涉及到信号的传播、电路的响应和内部状态的稳定。

多路选择器是一种能够从多个输入中选择一个输出到单一输出端的数字电路。它根据控制信号（选择线）的状态来决定将哪个输入信号传递到输出端。

拓展器是用于将输入数据或信号扩展到更大位宽或更大容量范围的装置或方法。它根据具体的应用场景，可以在数据的位数、存储器的容量或信号的传输带宽等方面进行扩展。

寄存器堆(有些教材称为寄存器文件 register file 或通用寄存器组 GPRs) 通用寄存器组是 CPU 内部的高速存储单元，它们设计用于临时存储数据和在指令执行期间的计算结果。这些寄存器之所以被称为“通用”，是因为它们可以用于多种不同的目的，包括但不限于数据传送、运算操作、地址存储、指针和索引，以及函数调用和栈管理。

暂存寄存器是中央处理器（CPU）内部用于暂时存储数据或指令的存储设备。它直接与 CPU 进行数据交换，是计算机系统中速度最快的存储设备之一。

指令译码器是控制器中的主要部件之一，它的主要任务是对计算机指令中的操作码字段进行分析和解释。指令是计算机能够执行的基本操作单元，由操作码和地址码组成。操作码指明了计算机要执行的操作性质和功能，而地址码则指明了操作对象（操作数）的存

储位置。

转移目标地址是指在程序执行过程中，当遇到分支指令（如跳转指令、条件跳转指令等）时，程序将要跳转到的目标地址。这个地址指明了程序接下来要执行的指令位置。

控制信号是计算机系统实施各类动态操作时生成的信号，这些信号用于指导计算机内部各个部件（如 CPU、存储器、输入输出设备等）按照预定的规则和时序进行工作。

硬线控制器是将控制部件设计成产生专门固定时序控制信号的逻辑电路，这些逻辑电路主要由门电路和触发器构成，用以产生各种控制信号，进而驱动计算机的各个功能部件协同工作。硬线控制器的设计目标是在使用最少元件的同时取得最高的操作速度。

微程序控制器是一种控制器，它采用微程序设计技术，将控制逻辑以微程序的形式存储在控制存储器（CM）中。微程序由多条微指令组成，而微指令则是若干微命令的集合，用于实现机器指令的一个操作。

控制存储器（CM）是一个只读存储器（ROM），专门用于存储微程序。微程序是一系列微指令的集合，这些微指令是实现机器指令操作的基础。

微代码，也称为微指令或微码，是计算机或处理器用来模拟和实现复杂指令集的一种低级编程语言。它通常用于将复杂的机器指令翻译成具体的硬件操作序列。

微程序是一种计算机指令集，它是一系列的简单指令序列，用于执行特定的任务。这些简单指令序列被称为微指令，每个微指令都对应一个或多个微操作，这些微操作是计算机硬件执行的基本操作。

固件是一种嵌入在硬件设备中的软件，用于控制设备的基本功能和操作。它通常被存储在非易失性存储器（如 ROM、EEPROM 或闪存）中，并且在设备启动时首先加载。固件是硬件设备和操作系统之间的桥梁，负责初始化硬件、提供硬件接口以及执行某些低级任务。

异常是指在 CPU 执行指令过程中，由 CPU 内部检测到的与正在执行指令相关的意外事件。这些事件可能会导致指令无法正常执行或需要特殊处理。

中断是指 CPU 外部的设备向 CPU 发出的中断请求。这些请求通常与当前正在执行的指令无关，是异步事件。

chap7

SRAM 是一种基于硅集成电路的内部存储器类型，用于计算机和其他电子设备中的数据存储和访问。

DRAM 是一种半导体存储器，用于存储数据和程序指令以供 CPU（中央处理器）快速访问。它属于 RAM（随机存取存储器）的一种，这意味着可以在任意时间访问内存中的任何位置。

易失性存储器，也称为挥发性存储器，是一种计算机存储器，它需要持续的电力供应以保持存储的数据。一旦电源被切断或发生故障，易失性存储器中的数据将无法保留。

相联存储器（也称为关联存储器或按内容访问的存储器）是一种特殊的计算机存储器，其数据访问方式不依赖于传统的物理地址，而是根据存储内容来进行存取。

记忆单元也称为存储元、位元，它是具有两种稳态的能表示二进制 0 和 1 的物理器件。

记忆单元是计算机内存的基本组成部分，用于存储数据和信息。在计算机科学中，它通常指的是构成存储器的基本成分，如中央处理器（CPU）中的缓存器、主存储器（RAM）以及各种辅助内存。

编址方式是指计算机系统中对各种存储设备进行编码的方法。

编址单位(addressing unit)是指具有相同地址的那些位元构成的一个单位，可以是一字节或一个字。对存储单元进行编号的方式称为编址方式(addressing mode),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字节编址方式，

刷新周期定义为，上次对整个存储器刷新结束的时刻作为开始点到下次对整个存储器全部刷新一遍为止的时间间隔，也就是对某一个特定的行进行相邻两次刷新的时间间隔。

片选信号（Chip Select Signal），简称 CS 或 SS，指的是地址线和数据线分开的 BIOS 芯片里的特定信号，通常位于该芯片的 22 脚。这个信号是由 CPU 发出，经过北桥和南桥到达的。

地址引脚复用是指某个引脚（Pin）可以配置为不同的功能，既可以作为地址线（Address Line）使用，也可以作为其他类型的数据线或控制信号线。这种设计旨在减少芯片上的引脚数量，同时保持足够的灵活性。

行选通信号，也称为行地址选通脉冲（RAS, Row Address Strobe），是计算机内存内部结构中用于标示存储单元行地址的信号。简单来说，它就像是内存内部的一个横坐标，用于定位特定的数据行。

列选通信号是发送给动态随机存取控制器（DRAM）的一个信号，用于关联的地址是列地址。它就像是内存内部的一个纵坐标，与行选通信号（RAS）结合，共同确定内存中的特定数据单元。

SDRAM 是 DRAM 的一种，但与传统的 DRAM 不同，它有一个同步接口，这意味着在响应控制输入前会等待一个时钟信号，以与计算机的系统总线同步。

行缓冲是指在进行 I/O 操作时，数据首先被缓冲到内存中，而不是直接写入到输出设备或从输入设备读取。当缓冲区满或者遇到特定的条件（如换行符）时，缓冲区的内容才会被一次性地写入到输出设备或从输入设备读取。

多模块存储器是由多个具有相同容量和存取速度的模块组成的存储器系统。每个模块都有独立的地址寄存器、数据寄存器、地址译码、驱动电路和读/写电路，因此它们既可以并行工作，也可以交叉工作。 是一种空间并行技术，利用多个结构完全相同的存储模块的并行工作来增加存储器的吞吐率。

连续编址方式在连续编址的多模块主存储器中，主存地址的高位表示模块号(或体号),低位表示模块内地址(或体内地址),因此，也称为按高位地址划分方式，地址在模块内连续。

在交叉编址的多模块存储器中，主存地址的低位表示模块(体)号，高位表示模块(体)内地址，因此，也称按低位地址划分方式。每个模块按“模 m”交叉方式编址。假定有 m 个模块(体),每个模块有 n 个单元，则第 i(i=0～m-1)模块中的地址为 i,i+m,i+2m,…，i+(n-1)*m。

—

在计算机磁盘驱动器中，柱面是由所有磁面（包含上下两个盘面）上编号相同的磁道组成的集合。

磁头和盘片相对运动形成的圆，构成一个磁道(track),磁头位于不同的半径上，则得到不同的磁道。多个盘面上的相同磁道形成一个柱面(cylinder),磁道号就是柱面号。

扇区（Sector）是硬盘或固态驱动器上的基本数据存储单元。它是磁盘上划分的区域，用于存储数据。

记录密度通常是指单位长度或单位面积内所存储的二进制信息量。这是评估存储介质性能的一个重要参数，它决定了存储介质能够容纳的数据量。

磁盘平均访问时间指的是硬盘在接收到系统指令后，磁头从当前位置移动到数据所在磁道，然后等待指定数据扇区旋转到磁头下方，最后读取或写入数据所需的总时间的平均值。

PROM（Programmable Read-Only Memory，可编程只读存储器）是一种计算机存储芯片，其特点是在出厂时其内容为空，但允许用户通过特定的编程设备（如 PROM 编程器）对其进行一次性的编程，之后内容便无法更改。PROM 通常用于存储计算机启动时需要的

关键信息，如引导加载程序（bootloader）或 BIOS（Basic Input/Output System）等。

EPROM（Erasable Programmable Read-Only Memory，可擦除可编程只读存储器）是一种计算机存储芯片，它允许用户通过特定的编程设备（如 EPROM 编程器）进行编程，并且其内容可以通过紫外线照射来擦除，从而允许进行多次编程。EPROM 在计算机系统、嵌入式系统以及需要频繁更新或修改代码的应用中非常有用。

EEPROM（Electrically Erasable Programmable Read-Only Memory，电可擦除可编程只读存储器）是一种非易失性存储器，它可以在计算机或专用设备上被电子信号擦除和重新编程，而无需移除或更换芯片。EEPROM 具有一系列独特的特点和优势

闪存（Flash Memory）是一种电子式可清除程序化只读存储器的形式，具有非易失性特点，即掉电后保存的数据不丢失。

码制，即将一定位数的数码按一定的规则排列起来，以表示特定的对象，这样的规则就称为码制。其主要功能是确保信息能被电路识别，并便于运算和存储。

码制，即将一定位数的数码按一定的规则排列起来，以表示特定的对象，这样的规则就称为码制。其主要功能是确保信息能被电路识别，并便于运算和存储。

码距是指两个等长字符串（或码字）在相同位置上不同字符（或比特位）的个数。在信息编码中，两个合法代码对应位上编码不同的位数也称为码距或海明距离。

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

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

在访存过程中，需要判断所访问信息是否在 cache 中。若在 cache 中，则称 cache 命中(hit),命中的概率称为命中率 p(hit rate),它等于命中次数与访问总次数之比；若不在 cache 中，则为不命中(miss),其概率称为缺失率(miss rate),它等于不命中次数与访问总次数之比。

命中时，CPU 在 cache 中直接存取信息，所用的时间开销就是访问 cache 的时间 T,称为命中时间(hit time);缺失时，需要从主存读取一个主存块送 cache,同时将所需信息送 CPU,因此，所用时间开销为访问主存的时间 T。和 T.之和。通常把从主存读入一个主存块到 cache 的时间 T。称为缺失损失(miss penalty)。

虚拟地址（也称为虚地址/逻辑地址 VA）是内存管理的一种方式，为每个进程提供一个独立的、连续的内存地址空间。这些地址并不直接对应物理内存中的实际位置，而是由操作系统和硬件共同管理的中间层。

虚拟页号（VPN）是虚拟地址空间中的页面号，用于标识虚拟内存中的页面。在分页内存管理机制中，虚拟地址空间被划分为固定大小的块，每个块称为一个页面。虚拟页号就是这些页面的编号。

物理地址是内存中各存储单元的编号，即存储单元的真实地址，它是可识别、可寻址并实际存在的。物理地址也被称为实地址（Real Address）或二进制地址（Binary Address），它是在地址总线上，以电子形式存在的，使得数据总线可以访问主存的某个特定存储单元的内存地址。

页框（Page Frame）是内存管理中的一个概念，指的是物理内存中用于存放数据块（页）的固定大小的连续区域。通常，为了简化地址转换电路，物理内存被划分为长度相等的块，这些块就被称为页框。页框的大小通常是 4KB 或 8KB，但具体大小取决于系统和硬件的设计。

物理页号（Physical Page Number）是物理内存中页框的编号。在地址转换过程中，逻辑地址（或虚拟地址）中的页号部分被映射到物理页号，以确定数据实际存储在物理内存的哪个页框中。

地址转换是指将逻辑地址（或虚拟地址）转换为物理地址的过程。在计算机系统中，

程序使用的地址通常是逻辑地址，而数据实际存储在物理内存中，因此需要通过地址转换来找到数据的实际存储位置。

页表（Page Table）是一种数据结构，用于存储逻辑页号到物理页号的映射关系。每个进程都有自己的页表，以便系统能够将该进程的逻辑地址转换为物理地址。页表通常包含有效位、脏位、引用位和物理页号等信息。

页表基址寄存器（Page Table Base Register）用于存储页表在物理内存中的起始地址。当系统需要进行地址转换时，它会首先读取页表基址寄存器中的值，以确定页表的位置，然后查找相应的页表项以获取物理页号。

有效位（Valid Bit，也称装入位）是页表项中的一个标志位，用于表示该页是否已装入物理内存中。如果有效位为 1，则表示页面已装入物理内存；如果为 0，则表示页面尚未装入，需要从磁盘等辅助存储器中调入。

修改位（Dirty Bit）也是页表项中的一个标志位，用于表示页面在物理内存中是否被修改过。如果修改位为 1，则表示页面内容已被修改；如果为 0，则表示页面内容自上次从磁盘调入后未被修改。这对于实现写回（Write-Back）策略非常重要。

缺页是指当系统尝试访问一个尚未装入物理内存的页面时发生的情况。此时，系统需要暂停当前进程的执行，从磁盘等辅助存储器中调入所需的页面，并将其装入物理内存中。这个过程称为缺页中断或缺页异常。

请求分页是一种动态内存管理技术，它允许程序在运行时根据需要动态地分配和释放内存。当程序访问一个尚未装入物理内存的页面时，系统会触发缺页中断，并自动从磁盘等辅助存储器中调入所需的页面。

FIFO（First In, First Out）和 LRU（Least Recently Used）是两种常用的页面替换算法。FIFO 算法按照页面进入内存的顺序进行置换，最先进入内存的页面最先被置换出去。而 LRU 算法则根据页面被访问的频繁程度进行置换，最长时间未被访问的页面最先被置换出去。

快表（Translation Lookaside Buffer，TLB）是一种高速缓存机制，用于存储最近访问过的页表项。当系统需要进行地址转换时，它会首先查找 TLB 中是否有匹配的页表项。如果 TLB 命中，则可以快速完成地址转换；如果 TLB 未命中，则需要访问主存中的页表来查找物理页号。

在计算机系统中，通常存在两种操作模式：管理模式（也称为内核模式或系统模式）和用户模式（也称为用户空间或应用模式）。在管理模式下，CPU 可以执行任何指令，并访问任何内存地址；而在用户模式下，CPU 只能执行受限的指令集，并且只能访问用户空间的内存地址。

异常返回是指当 CPU 在执行程序时遇到异常情况（如缺页中断、除零错误等）时，它会暂停当前程序的执行，并跳转到异常处理程序进行处理。异常处理程序完成处理后，会通过异常返回指令将控制权返回给被中断的程序。

存储保护是计算机系统中的一种机制，旨在防止程序或进程间相互越界访问，从而保护内存中的数据和程序不受破坏。

地址越界是指在程序执行过程中，由于某种原因（如错误的指针操作、数组下标越界等），程序试图访问其内存分配范围之外的内存地址。

访问越权是指用户或程序在不具备相应权限的情况下，试图访问或操作受限制的资源或执行不允许的操作。这种行为违反了系统的安全策略，可能导致数据泄露、系统被非法控制等严重后果

chap6

指令流水线（Instruction Pipeline）是一种将指令的执行过程分解为多个独立阶段的技术，每个阶段都由专门的硬件电路完成。这样，当一条指令在某个阶段进行处理时，下一条指令可以同时进入流水线的下一个阶段，从而提高了处理器的吞吐量和执行效率。

流水线深度指的是在指令流水线中可以同时流经的阶段的数量，即流水线的段数。每个阶段负责执行指令的特定部分，如取指、译码、执行、访存和写回等。

指令吞吐量是指在给定时间内，系统或处理器成功执行并完成指令的平均数量。它反映了系统的执行效率和性能。

流水段寄存器是流水线操作中用于保存各流水段处理结果的存储设备。在流水线操作中，一个任务（如一条指令或一个操作）被分解为多个有联系的子任务，每个子任务由一个专门的功能部件来实现。为了保存每个流水段的处理结果，并在需要时向后传递这些结果，每个流水段后面都会有一个流水段寄存器。

流水线冒险是指在现代 CPU 的流水线结构中，由于指令的并行执行而可能导致的一种错误状态，即下一条指令无法按照预期开始执行。流水线冒险主要分为三种类型：结构冒险、数据冒险和控制冒险。

结构冒险：当一条指令需要的硬件部件还在为之前的指令工作，而无法为这条指令提供服务时，就产生了结构冒险。这里的“结构”指的是硬件电路中的某个部件或资源。

数据冒险：当指令在流水线中重叠执行时，后面的指令需要用到前面的指令的执行结果，而前面的指令尚未写回导致的冲突，称为数据冒险（也称为数据相关性）。

气泡：在指令流水线中，当某条指令不能按正常时序进入下一个阶段时，流水线中的相应位置会出现空档，这个空档被形象地称为“气泡”或“流水线冒泡”（Pipeline Bubble）。

空操作指令（NOP）是计算机指令集中的一种指令，它本身不改变系统的状态或执行任何实质性的操作。当 CPU 执行 NOP 指令时，它仅仅是将程序计数器（PC）的值增加，以便跳转到下一条指令，而不进行任何数据处理或内存访问。

转发（Forwarding）或旁路（Bypassing）：是一种在 CPU 硬件中实现的机制，用于将一条指令的执行结果直接传递给下一条需要该结果的指令，而不需要等待该结果首先写入寄存器然后再从寄存器中读取。这种技术允许流水线继续执行，从而提高了 CPU 的性能。

控制冒险：如果现在要执行哪条指令，是由之前指令的运行结果来决定的，而现在之前指令的结果还没有产生，就导致了控制冒险。这主要由跳转分支等控制指令引起。