第一章 计算机基础

今天，微控制器已广泛应用于现代生产和生活中，如工业控制与自动化、电机控制、消费类电子、各种医疗设备等。微控制器往往作为控制核心，和软件及其他硬件集成到一起，形成一个功能专用，独立运行的应用设备或系统。

微控制器（Microcontroller Unit）并没有一个统一的定义。微控制器可以看成是围绕应用，其内部的硬件组成针对具体的功能、功耗、成本和可靠性要求进行优化的专用计算机系统。微控制器往往将计算机的各个组成部件如：CPU、存储器、定时器、AD/DC、串口通信和 PWM 等集成到一块芯片上。对于不同的应用领域，微控会对这些部件进行裁剪，因此微控制器有着众多的型号，各自配备不同大小的存储器，不同类型的外设。

微控制器是计算机的一个分支，学习微控制器，我们有必要了解计算机的基本组成，以及计算机是如何工作的。

1 计算机基本结构

1946 年 2 月 14 日，由美国军方定制的世界上第一台通用电子计算机 **ENIAC**（Electronic Numerical And Calculator，电子数字积分计算机）在美国宾夕法尼亚大学公布。ENIAC 是美国奥伯丁武器试验场为了满足计算弹道需要而研制的，这台计算器使用了 1.8 万支电子管，重达 30 余吨，功耗为 140kW，其运算速度为每秒 5000 次的加法运算。

ENIAC 是一台十进制（而非二进制）机器，也就是说其数字是以十进制表示，算法也是以十进制完成的。其存储器包含 20 个累加器，每个都能保存 1 个 10 位的十进制数。每一位数由 10 个真空管来表示，在任何时候，仅有一个真空管处于 ON 的状态，代表 10 个数字中的一个。**ENIAC** 的主要缺点是必须手动编程，一切都要通过设置开关和插拔电缆头来实现，输入和修改程序极其繁琐。

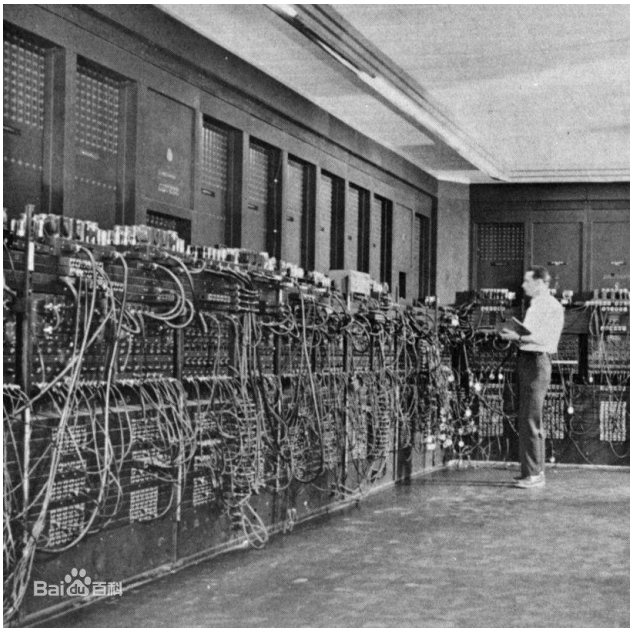


图 1-1 **ENIAC 电子**计算机

美籍匈牙利数学家冯·诺依曼于 1945 年 3 月在与他人共同讨论的基础上起草了 **EDVAC**（Electronic Discrete Variable Automatic Computer，电子离散变量自动计算机）的设计报告初

稿，在这份报告中，他提出了计算机设计的两个举足轻重的要点：**程序预存储**和**采用二进制系统**。将程序提前编写好，并存储在存储器中，这样计算机运行时可以连续的从存储器中读取指令，实现自动运行。采用二进制，使用容易实现的二进制电子器件和二进制逻辑运算来设计计算机，使得计算机电路大大简化。现代电子计算机仍然普遍采用冯·诺依曼计算机体系，冯·诺依曼也被尊称为现代电子计算机之父。

冯·诺依曼计算机体系结构的要点有：

1) 计算机中的程序和数据都采用二进制数字来传输、存储和运算。
2) 程序预存储，程序预先存储在存储器中，这样机器可以连续自动的执行。
3) 计算机包括运算器、控制器、存储器、输入和输出设备五个部分。

- **运算器：**又称算术逻辑单元 **ALU**(arithmetic and logic unit),对数据进行算术运算（加、减、乘、除及它们的复合运算）和逻辑运算（与、或、非、异或及比较、移位等等）。
- **控制器：**控制器每次从存储器读取一条指令，经过分析译码，产生一串控制信号，发向各个部件，以控制各部件动作，使计算机正常连续运行。
- **存储器：**存储程序和数据的部件。
- **输入\输出（I/O）：**在计算机及其外部环境之间传递数据。

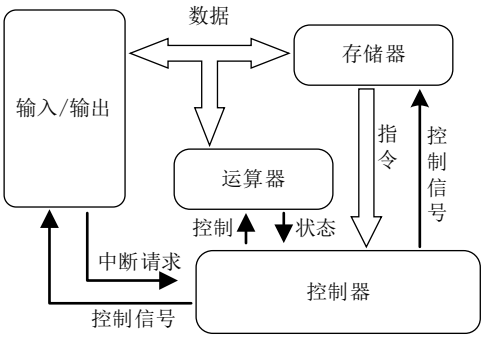


图 1-2 计算机的简单框图

我们知道，程序是一个指令序列。为了让 CPU **自动连续**地执行程序，指令和数据会先在存储器中保存起来。在 CPU 中有称为**程序计数器 PC**(Program Counter)的寄存器来保存下一条将要读取指令的地址。计算机运行时，CPU 会自动从 PC 指向的存储器单元中取出一条指令，并执行相应的操作，同时增加 **PC** 的值(PC+1)，以指向下一条指令（指令指针）。CPU 再对下一条指令执行上述操作，如此周而复始，除非遇到暂停指令才能停止执行，否则这个循环将一直继续下去。

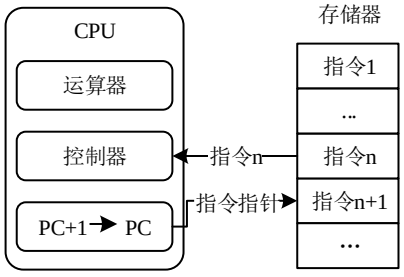


图 1-3 程序预存储

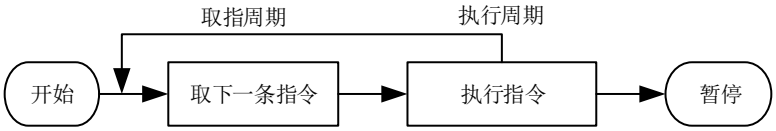


图 1-4 取指-执行指令周期

CPU：运算器和控制器是计算机的核心部件，常把他们合在一起称为中央处理器（CPU，Central Processing Unit）。

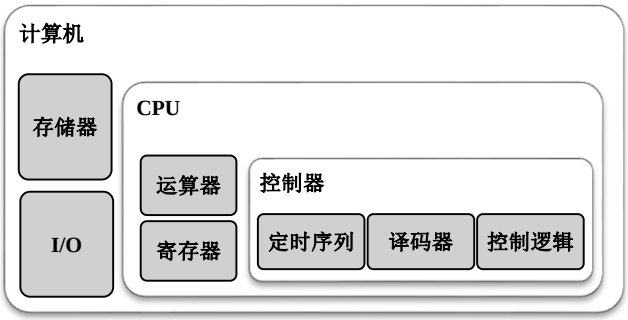


图 1-5 计算机顶层框图

单片机由芯片内仅有 **CPU** 的专用处理器发展而来，早期将 **CPU**、存储器和 **I/O** 外设集成到一个芯片上，便称为单片机（Single Chip Microcomputer）。嵌入式处理器的特征是针对具体领域的应用，只保留和应用密切相关的硬件，去除其他的多余部件，这样就以最低的功耗、成本和体积实现特定领域应用的要求。目前市面上具有嵌入式特点的处理器已经达到数千种，可将其划分成：微处理器、微控制器、DSP 处理器、片上系统 SOC。

1) **MPU**(Micro Processor unit)微处理器，性能较强，工作主频高，能外扩大容量内存，但功耗较小，体积小，适用于功能较强的便携设备，常见的如以 Cortex-A 为内核的智能手机处理器芯片，运行 Android、iOS 等轻量级操作系统。
2) **MCU**（Micro Controller Unit）微控制器，与微处理器相比，微控制器的特点是单片化，体积减小，功耗和成本低、能适应恶劣的工作环境、可靠性高。微控制器广泛应用于工业控制和消费电子，其片上的外设资源种类繁多，具体型号针对具体的应用场合对外设进行裁剪。
3) **DSP**(Digital Signal Processor)数字信号处理器，DSP 针对数字信号处理算法做了优化，如采用多总线、流水线、硬件乘法器、DSP 指令、多核并行运算等技术。DSP 在数字滤波、频谱分析、音频和视频处理等场合获得了广泛应用。
4) **SOC**(System On Chip)片上系统，是以一个特定应用为目标，把微处理器和相关外设集成到一块芯片上，其中还包括嵌入式软件。如许多蓝牙芯片、WiFi 芯片和汽车电子芯片都是 SOC。

2 计算机工作过程

2.1 运算器、存储器和 I/O 的简单模型

为了方便对计算机运行原理的理解，我们先对计算机的几个组成部件建立一个简化的模型。

运算器:计算机系统的其他部件(控制器、寄存器、存储器、I/O)为 ALU 传入数据，ALU 根据指令进行相应的运算，处理完成后输出运算结果。另外 ALU 可能根据运算结果设置一些标志。例如:如果计算结果超出了要保存它的寄存器位宽，那么溢出(overflow)标志将被置为 1。

存储器：通常，存储器模块由 N 个位数（如 8 位）相等的存储单元组成，每个存储单元分配了一个唯一的数值地址(0、1、…、N-1)，数据字可以从存储器中读出或写进存储器。操作的性质由读和写控制信号指示，操作的单元由地址指定。

I/O：从计算机系统内部来看，I/O 在功能上与存储器相似，它们都有读和写两类操作；往往也给它分配一组地址。最后，I/O 模块可以给处理器发送中断信号。

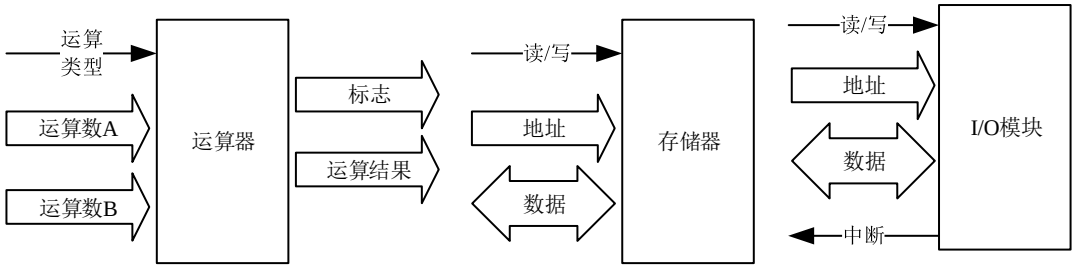


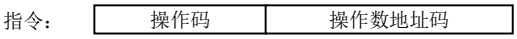
图 1-6 运算器、存储器和 I/O 的简单模型

2.2 指令的执行

指令（instruction）是规定计算机执行特定操作的命令，是计算机处理的最小单元。对于同一系列的计算机，其指令集是相同的，程序是由一系列的指令组合而成。

指令一般包括两个部分：操作码和操作数地址码。操作码表示该指令要操作的类型，不同的操作使用不同的操作码表示，如可用 001 表示加法，002 表示减法。

操作数地址码用来表明操作数的地址，例如做加法的加数的地址。



指令的执行由很多离散的步骤组成，包括：从存储器中取出指令、译码指令、取数据、执行算术和逻辑运算以及存数据。

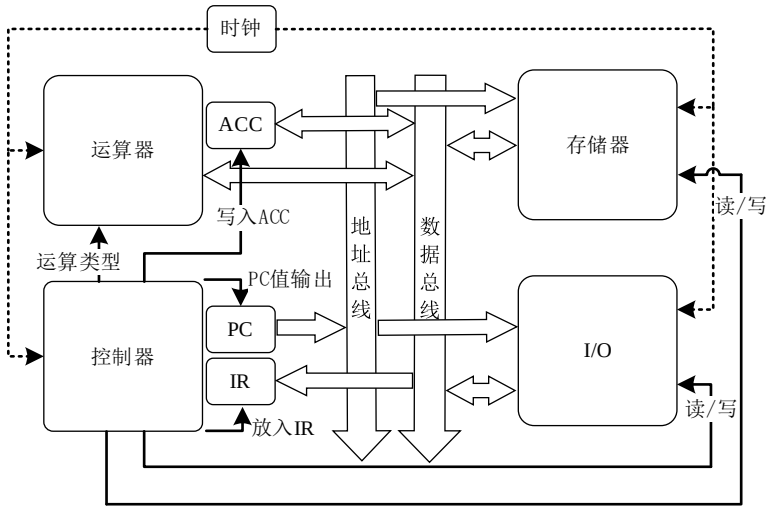


图 1-7 指令执行过程示意图（简化）

图 1-7 中可以看，CPU 内部有很多寄存器，如 PC、IR 和 Acc 寄存器。**寄存器**(Register)是 CPU 内部用来暂存数据的一些小型存储器，工作速度非常快。

程序计数器 PC：用来保存下一条要执行指令的地址。

指令寄存器 IR(Instruction Register)：用于保存从存储器中取出的指令操作码。

累加器 Acc：用于缓存计算过程中的结果。

下面以 MCS-51 指令集中的一条 ADD 指令的操作为例加以说明。

ADD A, 57H，此指令表示，将地址为 57H 中的数据和累加器 Acc 的值相加，并将结果保存到 Acc 中。其指令的机器码包含两个字节，为 25H 57H。其中前一个字节 25H 为操作码，规定了该指令是做加法操作，后一个字节 57H 是该操作需要的操作数地址码。该指令的操作过程如下（为了方便说明，此处做了简化）：

1) **取指**：将 PC 寄存器中的指令地址放到地址总线上，给出存储器读信号，从存储器读出指令操作码（25H）到 IP 寄存器，然后 PC+1。
2) **译码**：控制器对 IR 中的指令进行译码，识别指令类型。这里发现操作码为 25H 的指令是二字节指令，还需要读取操作数地址码。控制器给出存储器的读信号，读出操作数地址码（57H），并且 PC+1，指向下一条指令地址。
3) **取操作数**：从存储器 57H 的地址中读出操作数到运算器。
4) **执行**：控制器指挥运算器及其他部件，对操作数执行操作，这里执行加法操作。
5) **写回**：写回是将运算结果写到寄存器或存储器，这里是将运算结果放入 Acc 寄存器。

可见，这条指令的执行过程包括：取操作码(取指令第一字节)→译码(对指令操作码进行翻译，操作码将告诉 CPU 该指令的长短，再取出指令的第二字节操作数地址码)→取操作数→执行指令规定的操作→将结果写回寄存器或存储器。

时钟（Clock）：计算机各部分的工作需要使用时钟来同步，从而成为一个协调一致的系统。例如，我们指示算术逻辑单元(ALU)计算 x+y，其中 x 是与 ALU 邻近的寄存器的值，y 是距 ALU 较远寄存器的值。由于各种物理条件上的限制(距离、阻抗、干扰、随机噪声等等)，代表 x 和 y 的电信号可能会在不同的时刻到达 ALU。然而，对于 ALU 而言并没有时间概念——ALU 只负责不间断地把出现在输入端的值加起来。因此，ALU 的输出稳定到正确的 x+y 结果需要一些时间。在结果值稳定之前，则会产生垃圾值。

那么如何确保在结果稳定后读取呢？使用时钟就可以解决这个问题，只要时钟周期的长度大于稳定需要的最长时间，我们在下达指示后下一个时钟周期的开始读取，就一定能得到正确结果。

因此，指令的每个步骤都需要至少 1 个时钟周期来完成，整个指令则需要多个时钟周期来完成。有些指令可能只需要几个周期，而另一些指令需要几十个周期。

总线（bus）：连接计算机各个部件的通信线路。总线的特点是共享传输线路。总线上可以连接多个设备，一个设备发出的信号可以被其他所有设备所接收。多台设备不能同时发送信号．否则信号将会冲突。因此，一个时间只能有一个设备使用总线发送数据，其它的设备必须等到总线空闲时才能使用总线。

通常，总线由多条并行的线路组成，这样能够用来同时(并行地)传送二进制数字。例如，

一个 8 位的数据能通过总线中的 8 条线传送。总线的位数称为总线的宽度。

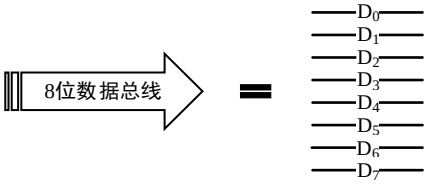


图 1-8 总线

计算机系统含有多种总线，它们在计算机系统的各个层次提供部件之间的通路。连接计算机的主要部件(CPU、存储器、I/O)的总线称为系统总线，CPU 内部也有总线，称为内部总线。系统总线一般包括数据总线、地址总线和控制总线。

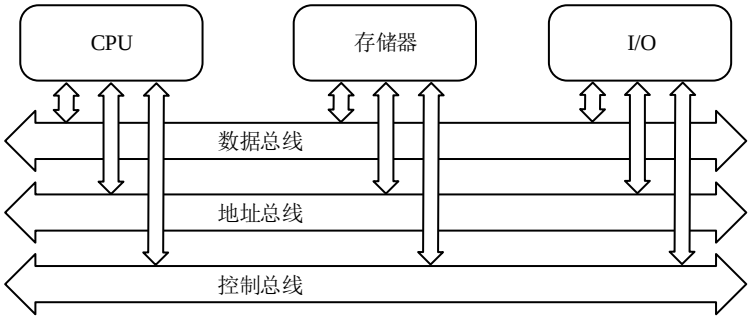


图 1-9 系统总线框图

字长：运算器一次运算处理的二进制位数称为字长，常用的计算机字长如 8 位、16 位、32 位、64 位等。一般计算机的字长和数据总线、寄存器和存储器的宽度是相同的。

2.3 控制器的设计

由前面的分析可知，控制器就是要按照指令的要求，在一定的时序下，发出一系列的控制信号。下面介绍一种硬布线方案的控制器设计，它主要由译码器、定时序列和控制器逻辑组成。

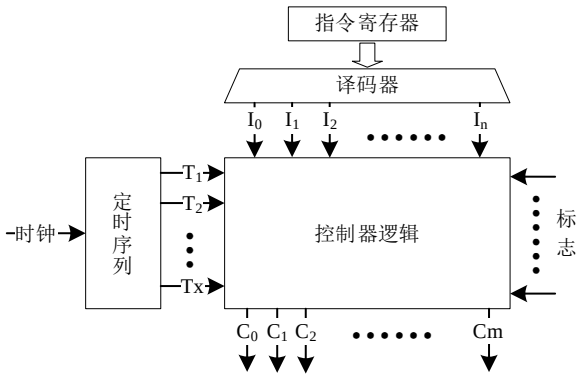


图 1-10 带译码器的硬布线方案控制器

译码器：通过指令寄存器，控制器使用指令的操作码，并将为不同的指令完成不同的动作(发出不同的控制信号组合)。为简化控制器逻辑，应使每一操作码有一个唯一的逻辑输入。译码器(decoder)能完成这个功能，它接收一个编码了的输入并产生单一的输出，即一条指令对应图 1-10 中 I_0 到 I_n 中的一个。

定时序列：时钟发出的是一个重复的脉冲信号，在一个指令周期内，控制器要在不同时

间单位发送不同的控制信号。于是，我们希望通过定时序列得到当前是第几个时钟。指令周期内的每一个时钟都对应定时序列信号 T_1 到 T_X 中的 1 个，例如在第 1 个时钟 T_1 有效，而其它的定时序列信号都无效。在指令周期结束时，控制器必须复位计数器，以使它从 T_1 重新开始。

控制器逻辑：得到指令译码信号和时钟序列信号后，接着就是产生控制信号，显然这是一个组合逻辑。我们根据某个控制信号要在哪些指令的哪些时钟序列发出，来确定其逻辑表达式。

例如：如果控制信号 C_0 在指令 I_0 的 T_1 和 T_4 时刻有效，则$C_0 = I_0 \cdot T_1 + I_0 \cdot T_4$

如果C_0还在指令I_4的T_4和T_5时刻有效，则$C_0 = I_0 \cdot T_1 + I_0 \cdot T_4 + I_4 \cdot T_4 + I_4 \cdot T_5$

2.4 中断

通常，控制器按顺序执行存储器中的指令，但这一顺序能通过程序控制类指令来加以改变，例如条件转移指令、无条件转移指令、循环指令和函数调用指令。以转移指令为例：条件转移可以依据条件（通常是运算的标志）来决定是否跳转到新的地址执行，比如 C 语言中的 if 语句就是用条件转移指令实现的；无条件转移指令可以使程序直接跳转到新的地址执行，比如 C 语言中的 goto 语句。另外一种打断顺序执行的情况是中断。

2.4.1 中断执行过程

中断机制能打断计算机的正常程序运行，而去紧急处理外设（如 I/O、定时器等）的一些请求。由于大部分外设比处理器的运行速度慢很多，中断能大大提高处理器的使用效率。以打印机为例：处理器可以很快的将数据传送给打印机，而打印机打印这些数据则需要比较长的时间。如果没有中断，处理器在传送完部分数据后（打印机缓存空间有限），就必须等待（空闲）到这些数据打印完成，再传送剩下的数据，这就造成了处理器使用的巨大浪费。

如果使用中断，当打印机准备好接收数据时，则发送中断请求信号给处理器。处理器暂停当前程序的运行，跳转去运行打印机的中断服务程序（向打印机发送数据），并且在中断服务完成（数据传输完成）后恢复原来程序的执行，此时打印机开始打印接收的数据。这样，处理器就可以在外设进行操作的时候执行其他程序。

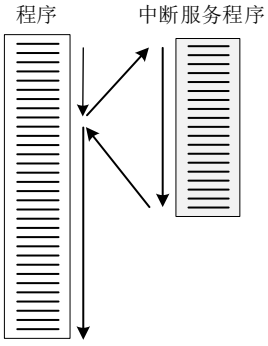


图 1-11 中断执行过程

中断机制意味着要在指令周期中则要加入中断相关环节。如图 1-12 所示，在指令执行完后要检查是否有中断信号，如果没有或者中断被禁止，则去读取下一条指令，继续执行。如果发现了中断请求，则要进行下列操作：

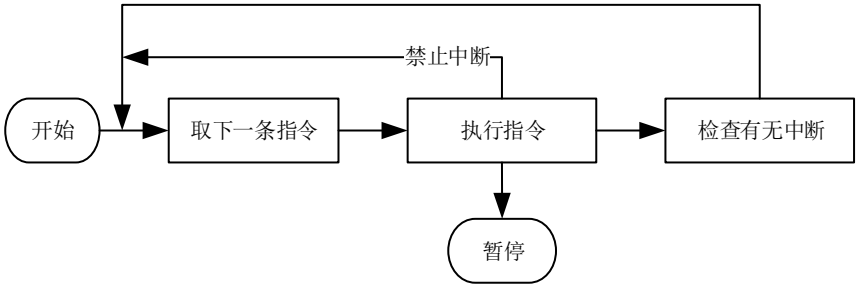


图 1-12 加入中断的指令周期

1) 暂停当前正在执行的程序，并**保护现场**。保存下一条指令的地址（程序计数器 PC 内的当前值），以及处理器当前处理的相关数据，一般包括一些寄存器的值和运算状态标志。

2) 将 PC 的值设置为中断服务程序的起始地址，则处理器从中断服务程序的第 1 条指令开始执行。当中断程序执行完后，要**恢复现场。**将 PC 重置为保存的原程序断点地址，并恢复相关寄存器的数据，从而继续原来程序的执行。

2.4.2 多重中断

实际情况中往往会有多个中断同时发生，如一个中断未处理完又来了新的中断。为能及时响应并处理发生的所有中断，可以将中断源分为若干个级别，称作中断优先级，从而使中断响应能有一个优先次序。多重中断的处理方式分为顺序中断处理和抢占式中断处理。

顺序中断处理：在中断处理过程中忽略新的中断请求信号，新中断会被挂起等待，等中断处理完成后再进行检查，如果有新中断等待，则继续执行新中断。如图 1-13 所示，处理器处理完中断程序 A 后就会检测到挂起的中断 B，于是继续执行中断服务程序 B。如果有多个中断在同一个时刻到达，或者处理器检测到多个挂起的中断，则先处理其中高优先级的中断。

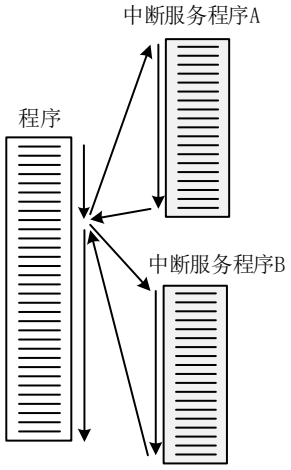


图 1-13 顺序中断处理

抢占式中断处理：抢占式中断处理允许优先级高的中断打断低级中断处理程序。如图 1-14 所示，用户程序发生了 A 中断，开始执行中断服务程序 A。当程序 A 仍在执行时，发生了更高优先级的 B 中断。由于 A 的优先级相对较低,只好打断中断程序 A,而 B 中断得到响

应，中断程序 B 执行完后，再继续执行中断程序 A。

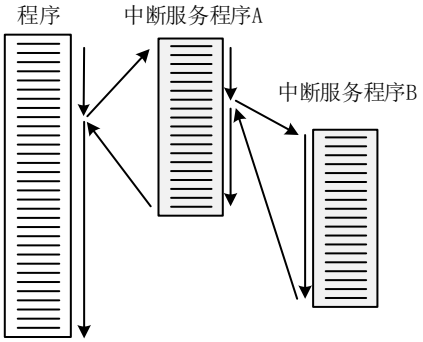


图 1-14 抢占式中断处理

3 计算机的架构改进

计算机硬件速度的提高很大程度上要归功于芯片工艺制程的发展，使得计算机芯片的逻辑门电路尺寸越来越小，门电路之间的距离也越来越近，信号传输延迟变小，芯片功耗降低，这都使得时钟频率可以变得更高。时钟频率提高意味着每个操作步骤能以更快的速度进行。另外，计算机处理数据和传输数据的位数的提高，如 8 位计算机到 16 位、32 位和 64 位计算机也大大提高了计算机的性能。

改变处理器的组成和体系结构也是提高指令执行速度的有效途径。特别是在计算机架构中采用各种并行处理技术。由于在性能和功耗上有着越来越高的要求，微控制器常常采用了流水线、哈佛结构和 RISC 指令集等处理器架构技术。

3.1 流水线

每条指令的执行都分为取指、译码、取操作数、执行和存储器操作写回 5 个步骤。假如每个步骤用时为 T 个时钟周期，则一条指令需要 5T 个时钟周期，3 条指令需要 15T 个时钟周期。但取指令是通过总线完成的，而译码和执行指令大部分是在 CPU 内部的译码部件和执行部件串行完成的。因此，就有可能在一条指令的执行过程中，同时译码下一条指令，并预读取第三条指令。这样 7T 个时钟内可能完成 3 条指令的执行，从而大大提高了指令的执行速度。

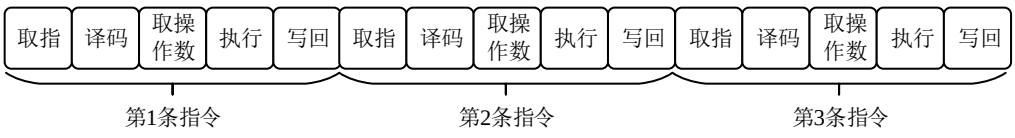


图 1-15 顺序执行

图 1-16 流水线执行

CPU 流水线技术将指令分解为多个步骤，让数条指令的不同步骤同步操作，从而实现

几条指令并行处理，以加速程序运行过程的并行执行。流水线技术要求各功能部件能相对独立地工作，这就要增加硬件。例如要能预取指令，就需增加指令缓冲器，从而可以将读取的指令存放到指令缓冲器中。类似的，其它部件也需要缓冲器，以缓存各自的处理结果。这使得微处理器能同时进行取指令和译码、执行指令的操作。

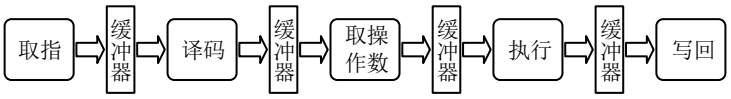


图 1-17 流水线硬件示意图

3.2 普林斯顿结构和哈佛结构

普林斯顿结构

普林斯顿结构也称冯·诺依曼结构，其程序指令和数据都存储在同一存储器结构的不同位置。指令和操作数都通过同一条总线访问，这就必须使用分时复用的方式进行。缺点是在流水线执行时，不能同时预取指和存取数据，形成传输过程的瓶颈。

哈佛结构

哈佛结构将程序指令存储器和数据存储器分开，有各自独立的总线供 CPU 访问。哈佛结构的程序指令和数据指令是分开组织和存储的，使得其预取指操作具有较高的执行效率。目前，几乎所有的 32 位 RISC 流水线处理器都采用了哈佛结构。

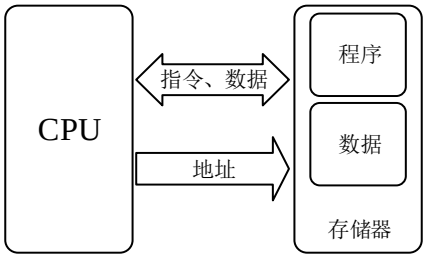


图 1-18 普林斯顿结构

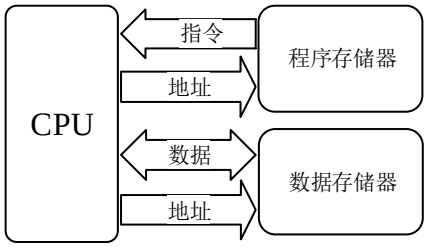


图 1-19 哈佛结构

3.3 RISC 指令集

在计算机行业的早期，编译器技术尚不发达，程序都是采用机器语言和汇编语言来编写。为了方便编程，更喜欢采用功能强大的指令。另外在那个年代，内存容量小、价格高且访问速度慢，为了减少代码的长度，以及降低访问内存的频率，更倾向于采用复杂的指令。即能用一条指令完成一系列的功能，例如：将数据的读写和计算合并到一起，组成 1 条指令。这样产业界就倾向于复杂指令集 CISC (Complex Instruction Set Computing)，桌面计算机流行的 x86 计算机体系采用的就是 CISC 指令集。CISC 指令的特点是高度编码、长度不等、执行多个操作。

IBM 公司于 1975 年对指令系统的合理性问题进行了研究。对 CISC 的测试表明，使用最多的是只占指令总数 20%的一些简单指令，在程序中的使用频率占到 80%。由此可见，那些复杂指令实际上很少使用，而且这些复杂指令带来了处理器结构复杂，功耗高，设计时间和成本高的弊端。于是出现了指令集设计的另一种思路，其基本思想是简化计算机指令功能，只保留那些使用频率高的简单指令，而把较复杂的功能用一组简单指令来实现，这种指令集

就被称为精简指令集 RISC (Reduced Instruction Set Computing)。

大部分 RISC 指令集具有下述一些特点，使其很适合于高效的流水线操作。

1) 指令数量少，选取使用频率高的简单指令，以及一些很有用但不复杂的指令。
2) 统一指令格式，指令长度一般固定，只采用少数寻址方式。
3) 采用专用指令访问存储器，如取数/存数指令(1oad / store)，负责将数据在寄存器和存储器之间传送。其余指令不访问存储器，其操作都是在寄存器之间进行。
4) 指令执行时间比较固定，大部分指令在一个或小于一个机器周期内完成。

4 计算机数制及存储

计算机可以看成是信息处理机，对输入的信息进行处理，再输出结果信息。输入的信息必须能被计算机识别和存储，而计算机是采用二进制的，所以任何数据和信息都要转换成二进制的形式。下面讨论计算机中数据的表示。

4.1 二进制、十进制和十六进制

一个数值可以用不同进制的数表示，如二进制的 1010 和十进制的 10 表示的是同一个数值。通常用数字后面跟一个英文字母来表示该数的数制。

十进制（Decimal）数据加后缀 D 表示，D 可以省略，如：11D，或 11。

二进制（Binary）数据加后缀 B 表示，如 1011B。

十六进制（Hexadecimal）数据加后缀 H 表示,如:BH。在 C/C++语言中，十六进制数也往往用加前缀 0x 表示，如 0xB。

表 1-1 进制转换

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

例如：1011B=BH=11D

1) 每种进位制都有一个确定的基数 R，每一位的系数 Ki 有 R 种可能的取值，例如二进制每位只有 0 和 1 两种可能。
2) 按“逢 R 进一”方式计数，在混合小数中，小数点右移一位相当于乘以 R，左移一位相当于除以 R。

对于 16 进制，有十六个不同的数字符号：0～9、A、B、C、D、E、F。R=16, K_i 为 16 个数码中的任意一个，逢十六进一。

例 1.1：将A2.3H转换为十进制数

解： $\mathrm{A2.3H} = 10 \times 16^1 + 2 \times 16^0 + 3 \times 16^{-1} = 162.1875$

例 1.2：将1101.001B转换为十进制数

解 ： $1101.001\text{B} = (1101.001)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+0\times2^{-2}+1\times2^{-3} = (13.125)_{10} = 13.125$

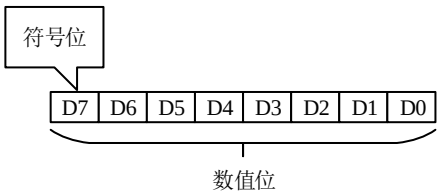
在计算机内，所有的信息均以二进制的形式表示，它们有一定的位数（bit）。例如：11101001B 和 E9H 都表示的是 8 位的数据，需要 8 位的存储空间，而 00E9H 则是 1 个 16 位的数据。

4.2 有符号数和无符号数

计算机中的数据分为无符号数和有符号数。前面所涉及的都是无符号数，无符号数只能表示正数；而有符号数则可表示正数和负数。在计算机中，为便于计算机识别，需将带符号数的正、负号数字化。有符号数有原码、补码和反码多种表示方法。

4.2.1 原码

原码的作法是在数值前面加一符号位，即二进制数的最高位（最左边）为符号位，且用“0”表示正数，用“1”表示负数。数值部分即为该带符号数的二进制值。



$$D_7 = \begin{cases} 0 & \text{正数} \\ 1 & \text{负数} \end{cases}$$

图 1-20 原码格式

例如 82 和-82的原码表示为：

$$82 = 01010010\text{B}$$

$$-82 = 11010010\text{B}$$

原码的表示法有两个缺点，一个是在做加减运算时，既要考虑数的符号位，也要考虑数值位。另一个缺点是 0 有两种表示方法：$+0$和-0，判断是否为 0 的操作更为繁琐。所以实际上最常用的方法是补码。

4.2.2 补码

补码也采用最高位为其符号位（正为“0”，负为“1”），不同在于其数值位的定义。

正数的补码：数值位和原码相同，如：$82 = 01010010\text{B}$。

负数的补码：数值位逐位取反后加 1，如：$-82 = 10101110\text{B}$。

数值位：1010010B

逐位取反：0101101B

数值位补码：$0101101\text{B} + 1\text{B} = 0101110\text{B}$

补码的可以将减法转换为加法，并可以连同符号位一起参与运算。

$$[\text{A} + \text{B}]_{补} = [\text{A}]_{补} + [\text{B}]_{补}$$

$$[\text{A} - \text{B}]_{补} = [\text{A}]_{补} + [-\text{B}]_{补}$$

例 1.3： 用 8 位补码计算 $y = 99 - 58$ 。

解： $y = 99 - 58 = 99 + (-58) = 41$

$$[99]_{补} = 0110\ \ 0011\text{B}$$

$$[-58]_{补} = 1100\ \ 0110\text{B}$$

$$01100011\text{B}\ \ +\ \ 11000110\text{B} = 1\ \ 0010\ \ 1001\text{B}$$

最高位的 1 自动丢失：

$$\text{y}\ \ =\ \ [\text{y}]_{补} = 0010\ \ 1001\text{B} = 41$$

溢出：若运算结果超过了某个字长所能表示数的范围，称为溢出。此时运算结果出错。例如：1 个 8 位有符号数的表示范围为–127～+127,如两个 8 位有符号数 126+5=131，大于 127，便会发生溢出。

$126_{补} + 5_{补} = 0111\ \ 1110\text{B} + 0000\ \ 0101\text{B} = 1000\ \ 0011\text{B} = -125_{补}$，发生了溢出错误。

4.3 定点数和浮点数

4.3.1 定点数

所谓定点数是指小数点的位置是固定不变的数，整数便可看为其小数点的位置固定在数据的最低位之后的定点数。

事先约定其小数点的位置固定于某位之后，则可以表示小数。如例 1.2 中固定于第 3 位之后。

$$1101.001\text{B} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 13.125$$

4.3.2 浮点数

定点数能表示的数值范围是比较有限的，不能表示较大的数据，也不能表示较小的小数，如何解决这个问题呢。在十进制中，我们通过科学计数法用较少的数字表示大范围的数据。比如 537 000 000 000 可表示为5.37×10^{11}，而$0.000\ \ 000\ \ 000\ \ 057\ \ 3$可表示为$5.73 \times 10^{-11}$，这里只用了 5、3、7、11 等四个数字就表示了一个很大的数和一个很小的小数，11 和-11 表示了小数点的位置，是可以变化的。

同样，也可以在二进制中使用相同的方法。与定点数相反，若小数点的位置在数中不是固定的,而是浮动可变的，则称这类数为浮点数。

计算机的浮点数可看为两个部分组成：一部分是尾数，一般将数的符号与尾数放在一起考虑，即尾数为一带符号的定点小数，它的符号称为数符；第二部分是阶码，它是一个带符号的整数。阶码实际上是指示尾数中的小数点应向左或向右移动的位数。

我们以常用的 **IEEE-754** 标准的 **32** 位单精度浮点数为例来说明。其数据格如下：

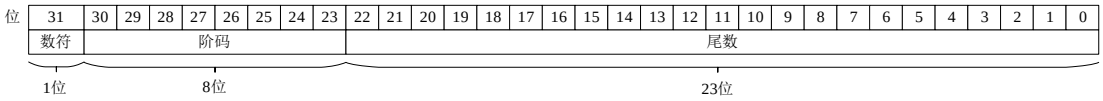


图 1-21 单精度浮点数格式

尾数（**Fraction，f**）：共 23 位，是小数点在其最高位之前的定点数，即表示一个纯小数。

数符（**sign，s**）：在最高位，表示数的正负。

阶码（**exponent，e**）：共 8 位。

其表示的数值可以用以下公式计算，其中 bias=127：

$$(-1)^s \times (1+f) \times 2^{e-bias}$$

例：

图 1-22 浮点数实例

十进制数值 $= (1 + 0.7189599) \times 2^{133-127} = 1.7189599 \times 64 = 110.0134336$

值得注意的是，虽然浮点数能表示的范围扩大了，但其能准确表示的数值数量还是固定的，仍然是2^{32}个。因此，很多时候我们都需要用近似的浮点数来表示想要的数值，这就存在了一定的误差。

4.4 字符和文字的计算机编码

相应的，为了让计算机能够识别，文字、声音、图像和其它各种各样的信息都需要转化为二进制数据。

ASCⅡ编码

对于英文字母和其它一些常用字符，一般采用 ASCⅡ编码来表示。ASCII ((American Standard Code for Information Interchange): 美国信息交换标准代码是由是由美国国家标准学会(American National Standard Institute , ANSI)制定，使用单字节对西文字符编码，后来它被国际标准化组织（International Organization for Standardization, ISO）定为国际标准，即 ISO 646 标准。

标准 ASCII 编码使用单字节中的低 7 位二进制数字来表示 128 个字符，包括 26 个英文字母的大小写、数字 0~9、标点符号和一些特殊控制符。最高位(bit7)被用作奇偶校验位，来检查数据通信中是否出现错误。但在扩展 ASCII 编码中，最高位也被用来表示字符，扩展出来的 128 个二进制数字用来表示一些外来语字符和图形符号。

汉字编码

对于汉字在计算机中的表示，普遍使用的是 GB2312 字符集（国家标准字符集），其中包括汉字 6763 个，符号 715 个。GB2312 编码采用双字节对汉字编码，即用一个双字节的数字表示 1 个汉字。此外还有 GBK 编码和 GB18030 编码，对能表示的汉字在 GB2312 编码的基础上进行了扩充，其中 GBK 编码是双字节编码，GB18030 采用单字节、双字节和四字节三种方式编码。

国际标准字符集 Unicode 编码

Unicode 编码为世界各种语言的每一个字符定义了一个唯一的编码，这使得软件在不同的语言环境中得到了兼容,这也使得在跨语言的平台中进行文本信息转换变得更方便。如今，大多数操作系统、浏览器和其它软件产品都支持 Unicode 编码。最新的 Unicode 编码版本是 2020 年 3 月发布的 13.0 版。

表 1-2 ASCII 编码

编码	字符/缩写	编码	字符	编码	字符	编码	字符
0	NULL	20	Space	40	@	60	`
1	SOH	21	!	41	A	61	a
2	STX	22	"	42	B	62	b
3	ETX	23	#	43	C	63	c
4	EOT	24	$	44	D	64	d
5	ENQ	25	%	45	E	65	e
6	ACK	26	&	46	F	66	f
7	BELL	27	'	47	G	67	g
8	Backspace	28	(	48	H	68	h
9	Horizontal Tab	29	)	49	I	69	i
0A	LF/NL	2A	*	4A	J	6A	j
0B	Vertical Tab	2B	+	4B	K	6B	k
0C	FF/NP	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1/XON	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3/XOFF	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	Cancel	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

4.5 字节编址和大、小端模式

计算机系统通常采用字节（Byte）编址，即给每个字节赋予一个地址。如图 1-22 所示，每个地址对应的存储单元都存储 1 个字节（8 位）的数据，则一个 32 位的数据，因为有 4 个字节，则占用了 4 个地址的存储空间，如从地址 00 到 03 的存储单元。

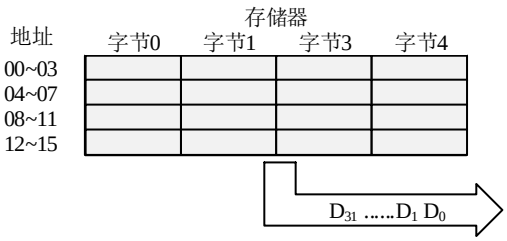


图 1-22 字节编址

32 位数据在存储时，其占用的 4 个字节存放有两种方式：大端模式和小端模式。

大端模式：存储器的低地址单元存放数据的高字节。如图 1-23 所示，32 位数据 0x12345678 的最高字节 0x12 存放在低地址单元 0。则 32 位数据总线的高字节数据位 D24~D31 连接到低地址存储单元（地址为 0、4、8 和 12）。

小端模式：存储器的低地址单元存放数据的低字节。32 位数据 0x12345678 的最低字节 0x78 存放在低地址单元 0。则 32 位数据总线的低字节数据位 D0~D7 连接到低地址存储单元（地址为 0、4、8 和 12）。

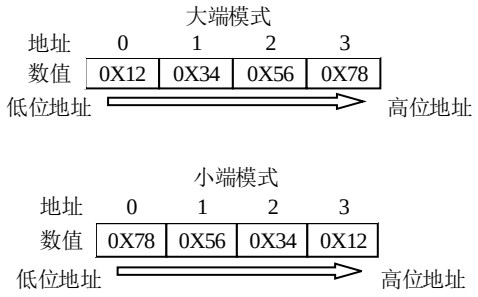


图 1-23 大端模式和小端模式

ARM 处理器大端模式和小端模式都支持，默认为小端模式，可以通过修改配置设为大端模式。在常见的应用中，Intel 的 X86 系列芯片使用小端模式；ARM 芯片两种模式都支持；IP 协议规定网络数据传输一般采用大端模式。

5 习题

1) 个人电脑（PC）是由哪些基本部件组成的？
2) 什么是单片机？单片机和微控制器的关系是怎样的？
3) 简述 CISC 和 RISC 的特点和区别?
4) 指令的机器码的典型元素是什么？
5) 如果你设计计算机，你会设计那些类型的指令？
6) 中断的保护现场可能有哪些信息需要被保存？
7) 在嵌入式 C 语言代码中，main 函数的最后都有一个 while 循环，为什么？
8) +0 和-0 的补码分别是什么？
9) 用补码计算 8 位的有符号数 25-37。

6 实验

1) 设计一个实验，编写代码，在 Tiva 中通过 CCS 的调试手段观察：

a) C 语言中的 char、short、int、long 和 double 型变量的数据长度为多少位？

b) C 语言中有符号数采用的是什么码？

c) C 语言中单精度浮点数 0.00001 的 32 位数值（16 进制）表示是多少？

d) 存储在内存中的一个 32 位数据 0x90345678，当其为有符号整数，无符号整数，字符时分别表示什么？