指令：计算机能识别和执行的最小功能单位

一地址指令：指定一个操作数，另一个操作数地址是隐含的（隐含在某个寄存器中）。

要先对寄存器进行赋值，再调用该指令。

零地址指令：在指令中指看到操作码，没有操作数，所有的操作数都是被隐含的。

通常是用堆栈来隐含操作数，默认操作数在堆栈中。所以要先把操作数放入堆栈之中，目标操作数隐含在堆栈顶部。也常用累加器来隐含操作数。

操作数越多，地址数越多，指令越复杂，长度越长，能存储到寄存器的数据越多，后期对寄存器操作也就越快。或者说能寻址的地址空间越多。

操作数越少，地址数越少，同一个程序需要用到的指令就越多，存储空间越小，存取速度越快。

指令集的设计：

指令集是计算机体系结构中最为重要的属性，定义了CPU能够完成的、汇编语言程序员可见的功能。

设计指令集需要考虑：

1. 指令清单：总共提供了多少操作、是什么功能、指令的复杂度；
2. 数据类型：每一条指令能够处理的数据类型
3. 指令格式：指令的长度（用多少个二进制位来描述指令，二进制位越多，指令功能越强大）、给操作码的分配的二进制位的长度（长度越长，指令的数量越长）、地址的个数
4. 寄存器：CPU可访问的寄存器的数目及其用途
5. 寻址方式（AM）：我们找到提供的操作数的地址的方式
6. 指令集的结构：RISC或者CISC

指令中操作数的类型：

1. 地址类型：操作数是通过给出它的地址指示的。

是一个无符号的整数。

1. 数值类型：直接提供要处理的数据，如整数、浮点数、二进制数
2. 字符型：直接提供要处理的字符，如ASCII码
3. 逻辑数据：以位来表示，1与0代表两种对立的事实。

标志寄存器中每个标志有两个位模式“1”或“0”来标志状态。

处理器中，寻址是以8个二进制位（一个字节）为一个单位来进行寻址的。

一个32位的双字要读取或写入，需要寻址4次。（对于数据总线宽度为8位的系统）

某些指令规定存储32位为一个字，但是处理器寻址依然是8位为一个单位。

计算机的四种基本操作：

1. 数据传送
2. 数据处理
3. 数据存储
4. 控制

进一步分类：

1. 数据传输指令

提供的要素：数据的来源、传输的目的地、数据块的大小长度、数据的寻址方式

1. 算术运算指令

基本的算术运算，加减乘除、自增自减

1. 逻辑运算指令（与或非、位处理如移位和循环移位）
2. 系统控制指令

一般是提供给操作系统使用

1. 控制传输类指令

如中断服务程序、条件判断打断程序的顺序执行、子程序调用

子程序调用：调用指令、返回指令（在子程序的内部）

1. 有些调用时会使用特定的寄存器来保存当前执行指令的地址的值
2. 有些是直接把地址放到子程序的最前面
3. 更多的是用堆栈来存储地址信息。

堆栈是内存中特殊开辟出来的空间，用来存储临时信息。

1. 输入输出指令

会在接口部分详细介绍

1. 格式转换指令

数据表示方法的转换，如二进制到十进制的转换。

汇编BRZ指令：BR代表Branch分支指令，Z指ZF标志寄存器中ZF标志，该标志有1和0两种可能值，1代表上一步执行结果为零，0代表上一步执行结果非零。

BRZ 211：若ZF标志寄存器为1则跳转到地址为211的指令。

BR指令：无条件跳转指令。

BR 211：跳转到地址为211的指令。

BRE指令：E代表Equal，判断是否相等，如果相等则跳转到指定的指令。

BRE R1 R2 211：若R1等于R2，则跳转到地址为211的指令。

端的规则：如何存储数据

如果有个4位的16进制数，如2345，存储则需要2个字节，需要分别存储在2个单元中，第一个单元是23，第二个单元是45，读取的时候如果没有规则，则可能读取成4523。

小端规则：最低有效字节存储在最低的地址编号的单元中（从小到大）

2345存储为：45 23

大端规则：最低有效字节存储在最高的地址编号的单元中（从大到小）

2345存储为：23 45

注意最右边才是最低有效字节。23代表的是23（高位字节），45代表的是45

存储是从最低有效位依次字节（8位）往高位，进行分组。

寻址分为两类：指令的寻址和操作数的寻址

指令的寻址：顺序寻址（隐含的寄存器PC）和跳跃寻址

操作数的寻址：

在指令中要把寻址方式指示出来，通过预留的AM（Addressing Mode）指示出来，在设计的时候已经定义清楚。

A（Address field）地址域：通过规定寻址方式来寻址的地址

EA是操作数的实际地址

（X）寄存器的编号，或者间接寻址的次数（一个括号代表间接寻址一次）

操作数的寻址方式：

1. 立即寻址（A就是操作数本身的值，操作数的大小受到分配给A的二进制位的位数限制）
2. 直接寻址（A是操作数的地址，寻址单元的数量受到分配给A的二进制位的位数限制）
3. 间接寻址（A是操作数地址的地址，可以有一次间接，二次间接，多次间接，寻址单元的数量受到存储器的二进制位的位数限制）
4. 寄存器的寻址（A的值是寄存器的编号，从寄存器中找到操作数）
5. 寄存器间接寻址（寄存器中为操作数的地址）
6. 偏移寻址（A是基准的与操作数的偏移量）
7. 堆栈寻址（直接在堆栈中把操作数读取出来）

引入指令流水的目的：使指令间能并行执行，提高指令的运行速度。

要求：

1. 任务是可分解的
2. 每一个被分解出来的步骤需要用到一个独立的物理单元来实现，并且该物理单元有独立的控制逻辑
3. 每一个物理单元在完成任务的时候应该在给定的时间内完成

能引起指令流水的冲突的有：

1. 资源冲突
2. 数据相关性
3. 跳转指令

某一个任务既可以用硬件来实现（在硬件的层面完成，由CPU设计者考虑），也可以用软件来实现（程序员调用指令来完成）。

CISC：复杂指令集的处理器

要减轻汇编语言程序员的负担，CPU提供的指令越多越复杂越好。

尽量提供更多更复杂的指令。

CPU的设计会更复杂。

RISC：精简指令集的处理器

把过于复杂的功能从CPU中剔除出去，把不必要的指令精减掉。把处理器中空出来的晶体管用于提高剩余指令的运行效率、增加CPU的寄存器、增强处理器的功能。

RISC的特点：

1. 提供的指令提供的指令的长度往往是固定长度的。CPU对指令进行译码的时候也会快很多，指令流水的效率也会提高。
2. 只能用load和store对内存进行读写，不能通过其他指令进行读写。简化了对内存的操作。执行计算的时候操作数只能位于CPU寄存器或者指令中，寻址方式减少了很多，只剩下立即寻址和寄存器直接寻址。只能先从内存load操作数到CPU寄存器再进行计算。由于要操作的操作数必须先存入寄存器才能进行运算，故能有效减少资源冲突，进而提高指令流水的效率。
3. CPU内寄存器更多了。
4. 指令少了，只有一些基本功能的指令，编译器更复杂了。
5. 绝大多数指令都可以在一个时钟周期内完成
6. 指令更多是通过硬件来实现的，速度远远快于用微程序来实现。
7. 参数的传递：操作数是通过片内（CPU内部）的寄存器来传递的，不需要多一次对内存的访问。

冯诺依曼架构：控制单元（Control Unit）、ALU（算术逻辑单元）、存储器、输入设备、输出设备

设计以ALU为中心，ALU的负荷很大。

CPU有70%的时间在对内存的操作，因为不管是指令还是数据都是在存储在内存中。

改进的架构的以存储器为中心的。只有涉及到算术运算或者逻辑运算才会启动ALU。

逻辑上将指令和数据分成两部分存储，物理上还是存储在一起的，因为它们共用同样的数据总线和地址总线。所以不能同时读取指令和数据。

Harvard架构：把存储器从物理上分成独立的两部分，分别存储指令和数据，把指令和数据从物理上分开。这两个部分与CPU有各自独立的总线连接。

可以解决资源冲突问题，解决指令和数据不能同时读取的问题。

Intel结构：

1. 采用CISC的设计模式
2. 小端规则
3. 两地址结构
4. 变长指令
5. 寄存器-存储器结构（操作数可能位于存储器也可能位于寄存器中）
6. 所有的寻址方式都可能用到

MIPS结构：

1. 采用RISC的设计模式
2. 小端规则
3. 采用32位的单字长
4. 固定长度的指令
5. 三地址结构
6. 只有load和store可以对内存操作
7. 只提供基本的寻址方式