第二章 指令系统

2.1指令系统概述

指令系统是指一台计算机所具有的全部**机器指令**的集合，它反映了该机所拥有的**基本功能。**

指令系统是计算机硬件的语言系统，也称为**机器语言。**

指令系统是软件和硬件的主要交界面，也是计算机**软件设计者和硬件设计者**之间沟通的桥梁。

指令系统决定了机器硬件所具有的**能力**，也决定**指令的格式**和机器的**硬件结构。**

指令系统也直接影响软件的**结构、复杂度和性能。**

指令系统的设计由**体系结构设计者**完成，指令系统的**逻辑实现是计算机组成的研究范畴。**

2.2指令系统的发展

2.2.1演变过程

指令系统经历了从简单到复杂，又从复杂到简单的螺旋式演变过程。

从简单到复杂：

50-60年代：电子管或晶体管计算机

硬件结构比较简单

仅有十几至几十条基本指令，且寻址方式简单。

60年代中期：集成电路计算机

硬件功耗、体积、价格下降，功能增强

指令数达100-200条，寻址方式多样化

60年代后期-70年代中期：半导体存储器出现

系列计算机诞生

新机器包含旧机种全部指令，软件向后兼容

70年代末期：超大规模集成电路和计算机

硬件成本下降，软件成本提高

指令系统更加复杂和完备，，指令数目可达300-500条，寻址方式多样化。

这样的计算机称为**复杂指令系统计算机（CISC）**

从复杂到简单

20世纪70年代

庞大的指令系统不但使计算机的**研制周期变长**，而且增加了**调试和维护**的难度，其结果还可能降低计算机系统的**性能**

1979年，美国加州伯克利分校Patterson教授领导的研究组，首次提出了**精简指令系统计算机（RISC）**的思想。

2.2.2CISC和RISC指令系统的特点

CISC指令系统的特征

软件硬化：用一条**复杂的新指令**来取代原先**一串指令**完成的功能

支持高级语言程序：增加新的**复杂指令**以及**复杂的寻址方式**

软件兼容：系列机软件要求**向上兼容和向后兼容**，指令系统不断扩大

CISC计算机存在的问题

（1）指令系统**庞大**，指令功能**复杂**，指令格式和寻址方式**多样性**，导致**编译程序复杂**，程序编译**速度慢**，**难以**用编译优化技术生成高效的目标程序。

（2）大多数指令**功能复杂**，且各种指令**都可以访问存储器**，使得绝大多数指令需要**多个机器周期**才能完成。

（3）为了实现复杂的指令系统，通常**采用微程序控制技术**设计控制器，由**微程序解释执行机器指令**，也影响了指令的执行速度。

（4）仅有**20%的指令使用频度比较高**，这些指令占据了80%的CPU时间。（导致硬件复杂、研制周期变长、难以调试和维护且可靠性变差）

RISC指令系统的特征

优先选取**使用频率较高**的简单指令

指令**长度固定**，指令**格式种类少**，**寻址方式少**

只有**取数/存数**指令访问存储器

cpu中通用**寄存器数量相当多**

cpu采用**流水线结构**，大部分指令在**一个时钟周期内**完成

控制单元设计以**硬布线控制逻辑**为主

采用**编译优化技术**，减少程序执行时间

2.3指令系统的功能

2.3.1指令系统设计原则

（1）**完备性**-功能需求

CISC指令系统的主要特点

RISC指令系统不强调完整性

（2）**规整性**-硬、软件设计需求

对称性：寻址方式

匀齐性：数据类型

一致性：指令格式和数据格式

（3）**高效性**-性能需求

CISC：完善指令系统功能，减小程序中指令的条数

RISC：降低每条指令的执行时间

（4）**兼容性**-通用性要求

2.3.2 数据类型

操作数类型

数据类型指的是面向应用或软件系统所处理的**各种数据结构**

基本数据类型：**整数、实数、布尔数、字符**等

复杂数据类型：**文件、图、表、树、阵列、队列、链表、栈、向量**等

数据表示指机器硬件能**直接识别**、指令能够**直接操作的数据结构**

例如**定点数（整数）、逻辑数（布尔数）、浮点数（实数）、十进制数、字符、字符串**等。

操作数指机器指令中的数据，即**硬件可以直接识别和处理的数据。**

确定操作数类型的原则

有利于缩短程序运行的时间

有利于减少cpu与主存储器之间的通信量

数据表示应具有通用性和较高利用率

常见操作数类型：

1.地址

操作数或指令被存放在数据存储设备的位置编码

主要数据存储设备有通用寄存器、主存储器和i/o设备

地址可认为是一个无符号整数

2.数字

计算机处理的最基本操作数类型

计算机中常用的数字类型有定点数、浮点数等

3.字符

在非数值计算领域表示和处理文本信息

将字符数字化表示，如ascll码

4.逻辑数

n位二进制数的组合，但各位之间可以没有任何关系

用于逻辑运算

地址空间

计算机中主要的数据存储设备有通用寄存器、主存储器和i/o设备，它们都包含多个可编址的数据访问单元，，对这些单元可以统一编址或者独立编址。

1.三个地址空间

对通用寄存器、主存储器和i/o设备分别进行独立编址

通用寄存器数量相对较少，地址码相对较短。

主存储器容量相对很大，因此地址码位数较长。存储器的编址单位分为按字编址、按字节编制和按位编址。

输入输出设备本身差异很大，编址方式不相同

risc只有访问操作可以在三类设备中进行，许多cisc中，所有操作都可以在三类设备中进行，因而指令系统比较复杂。

2.两个地址空间

对通用寄存器独立编址，主存储器和i/o设备统一编址。

在主存储器和i/o设备共用地址的高端划出一部分作为i/o设备的地址。

这样做可以简化指令系统，所有访问主存的指令都能访问i/o设备，但是这样做所有访问主存储器的指令都需要译码来判断是否访问i/o设备，影响指令的执行速度。

对于cisc处理器，很多指令对于i/o设备用不上

对于risc处理器，这样编址对于程序设计不利。

3.一个地址空间

所用存储设备统一编制。

通常是地址低端是存储器，高端是i/o设备。

这样编址地址编码最长，但是所有能访问寄存器的指令也能访问其他设备。

4.无地址空间

无地址空间也称为隐含编码方式。

在堆栈计算机中，指令是不需要地址的，有关设备不需要编址。

编址方式

编址方式是指主存单元的地址编排方式。编址方式决定了主存最小的访问单位。

1.按字编址方式

主存的**最小编址单位是一个存储字**，通常**存储字长等于机器字长**

对主存数据的**访问以字为单位**

主存容量=存储字数x存储字长，单位为**字（word）**或**位（bit）**

2.按字节编址方式

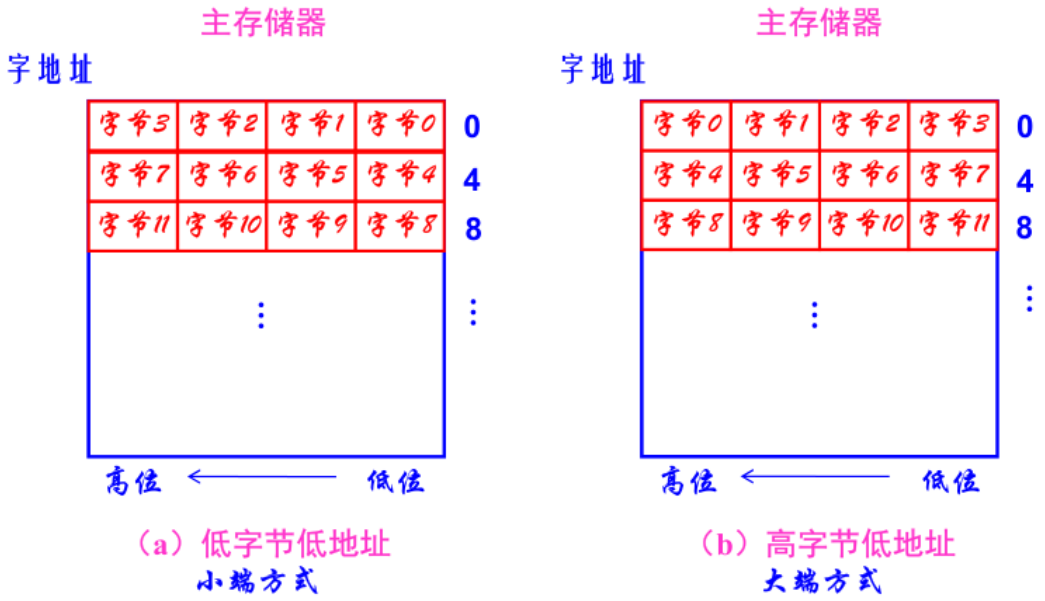
主存的最小编址单位是一个字节，描述主存储器容量时，以**字节（byte，b）**为单位

对主存数据**既能以字节为单位访问，也能以字为单位访问**

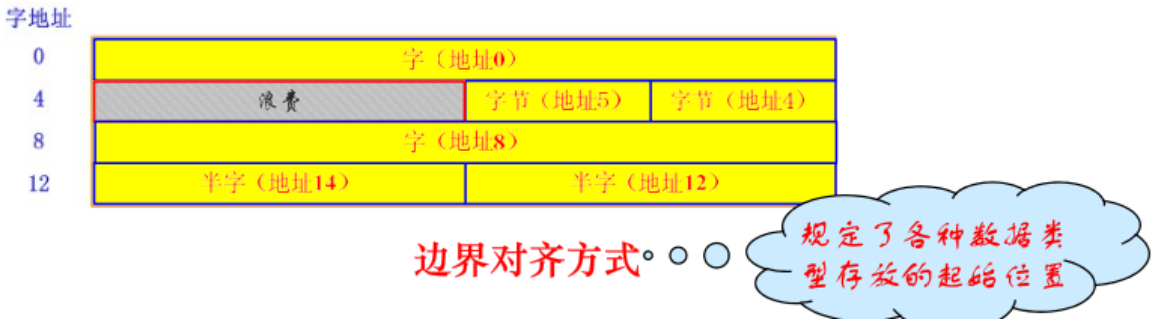
按字节访问主存时，使用字节地址；按字访问主存时，使用字地址

通常**存储字长是字节的整数倍，字节地址是连续的，字地址是不连续的**

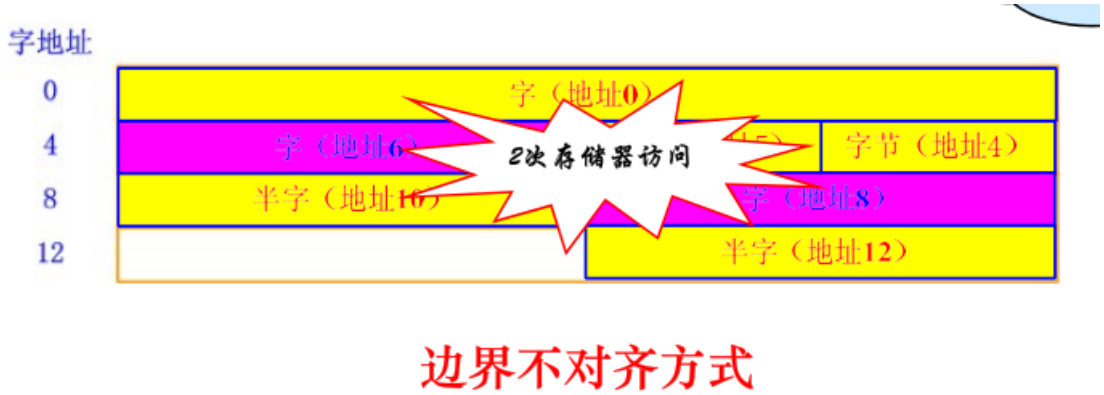
多个字节数据存放在一个字单元，有两种编址顺序：**低字节低地址（小端方式）、高字节低地址（大端方式）；也有存放边界问题：边界对齐、边界不对齐。**



对于边界对齐方式，规定了字节、半字、单字或者双字存放的起始位置。优点是无论访问一个字节、一个半字或一个单字都可以在一个存储周期完成，读写数据的控制简单，但是可能会造成空间的浪费。



对于边界不对齐方式，这是一种不浪费存储空间的存储方式，不同长度的数据一个接一个存放，但是这种存放方式储存在两个问题：一是除了访问一个字节外，其他数据可能跨两个字单元存放，有可能花费两个存储周期，主存储器访问速度降低一半；二是存储器的读写控制比较复杂。



通常，按字节编址的机器硬件支持边界不对齐方式，为了保证程序执行速度，软件可以选择采用对齐或不对齐方式。

2.3.3操作类型

所谓操作类型就是把指令系统按功能进行分类。一般指令分为五类：**数据传送指令、数据运算指令、程序控制指令、输入输出指令、其他指令。**

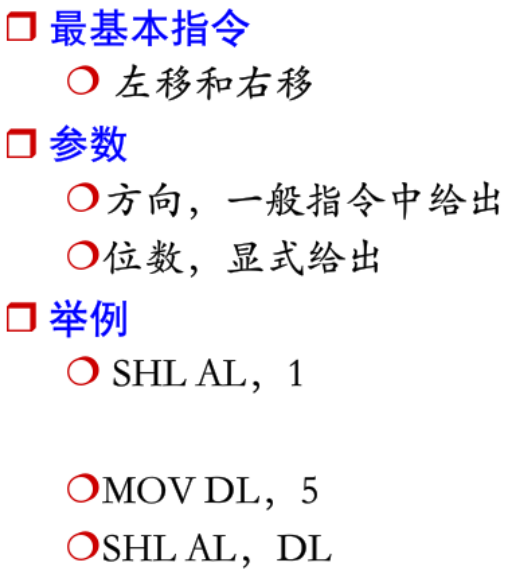
数据传送指令：在寄存器之间、主存单元之间以及寄存器与主存单元之间传送数据，包括数据源到目的之间的传送、对存储器的读和写交换源与目的内容、堆栈的入栈和出站。

- 传输主体
  - 主存<->主存
  - 主存<->寄存器
  - 寄存器<->寄存器
- IA-32
  - MOV AL, BL
  - MOV AX, [BL]
- MIPS
  - LW $ s1,($ s2)
  - SW $ s1,($ s2)

数据运算指令：cisc指令系统通常支持源与目的操作数在内存单元之中的指令，risc运算类指令源与目的操作数都在寄存器中。包括算术运算指令、逻辑运算指令、移位指令、位操作指令。

- 最基本指令
  - ADD
  - SUB
- IA-32
  - ADD AX, BX
  - SUB AX, BX
- MIPS
  - ADD $ s1, $ s2, $ s3
  - SUB $ s1, $ s2, $ s3

- 最基本指令
  - AND 、NOT 或者
  - OR、NOT
- IA-32
  - AND AX, BX
  - OR AX, BX
- MIPS
  - AND $ s1, $ s2, $ s3
  - OR $ s1, $ s2, $ s3



程序控制指令包括：

无条件转移指令（无需条件）

条件转移指令（满足一定条件，一般是标志位的信息）

调用（call）与返回指令（return）：为调用某些需要重复执行的子程序，call用于将当前程序转至子程序入口，return用于返回原程序断点。return需要记录返回地址，返回地址通常可以存储在以下三处：

1.寄存器内（但是难以支持子程序嵌套）；

2.子程序的入口单元内（可以支持子程序嵌套，无法支持子程序递归调用）；

3.堆栈栈顶，call将返回地址压入堆栈，return弹出栈顶元素获得返回地址。

陷阱指令（trap）：程序运行中的意外事故，作用是暂停当前程序，转入故障处理程序的故障处理，一般不提供给用户直接使用，而是作为隐指令由cpu自动执行。有的机器（如IA-32）设置用户使用的陷阱指令和访管指令，利用它完成系统调用和程序请求。

输入输出指令：对于输入输出设备单独编址的设备来说，，通常设有i/o指令，作用为从i/o接口的寄存器中读入一个数据到cpu的寄存器内或者相反。

其他指令：包括等待指令、停机指令、空操作指令、开中断指令、关中断指令、置条件码指令等，为某些需求制定的指令。

2.4指令格式

指令格式是指令的二进制表示形式。

2.4.1指令的组成

指令格式是二进制代码表示的指令字结构形式，**操作码（功能特性）与地址码（硬件结构）**是指令的组成部分。

操作码字段指出指令的操作性质，即指令要完成的功能。

地址码字段指出操作数的地址，即指令操作对象所在的位置，或者下一条指令在主存储器中的地址。

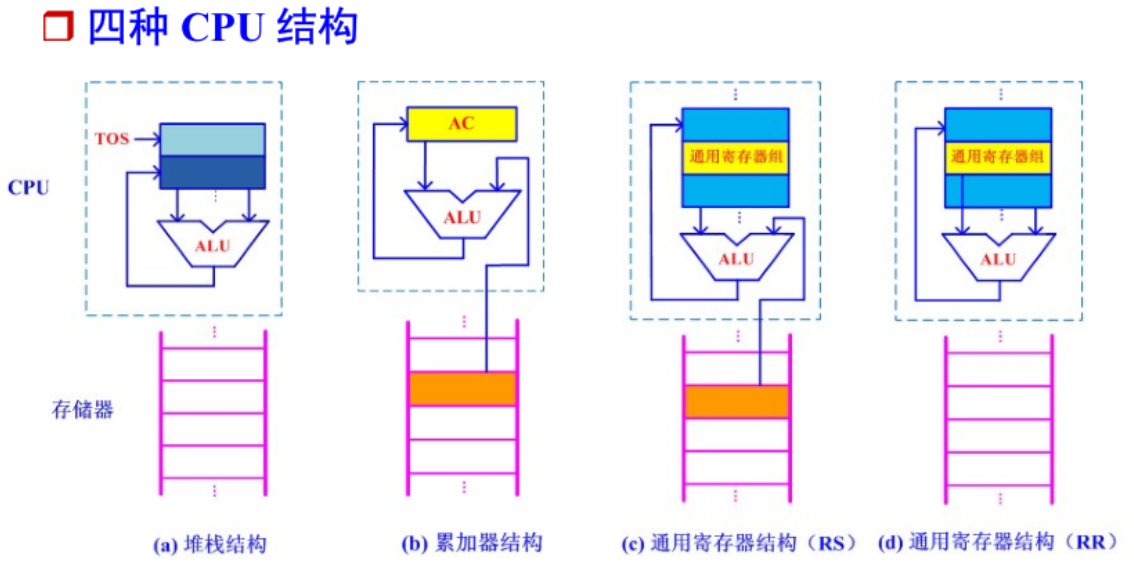
1.操作码

通常提供两部分信息：指令操作的类型和所用操作数的类型。

2.地址码

地址码指令中设计的地址包括寄存器地址、主存单元地址及i/o设备地址。

指令中的地址码形式取决于cpu的结构，分为堆栈结构、累加器结构和通用寄存器结构。



通用寄存器结构分为：寄存器—寄存器型（RR型）、寄存器—存储器型（RS型）

堆栈结构中，操作数都是隐式的，源操作数来自栈顶和次栈顶，只能通过push/pop指令访问存储器。不能随机访问堆栈，难以生成有效的代码。早期计算机都是采用堆栈结构，20世纪80年代以后陆续采用了通用寄存器结构，因其在灵活性和性能方面的优势。

累加器结构中，一个源操作数是隐式的，即累加器，另一个来自存储单元，运算结果送回累加器。程序需要频繁访问存储器。

在堆栈型和累加器型计算机中，指令字比较短，程序占用空间少。

在通用寄存器结构中，所有操作数都显式给出，rs型一个来自通用寄存器组，另一个来自存储单元，rr型都来自通用寄存器组，结果都存入通用寄存器组。

由于ALU最多涉及三个操作数：两个源操作数一个目的操作数，可以进一步把通用寄存器型指令系统分为三种类型：RR型、RS型、SS型。（现代计算机中已经不采用ss型）

寄存器-寄存器型（RR型）：

优点：指令字长固定，指令结构简单，各种指令执行时钟周期数相近。

缺点：指令条数较多，目标代码不够紧凑，程序占用空间大。

寄存器-存储器型（RS型）：

优点：可以在ALU指令中直接对存储器操作数进行引用，不必用load指令进行加载。容易对指令进行编码，目标代码比较紧凑。

缺点：由于一个操作数的内容将被破坏，所以指令中的两操作数不对称。在一条指令中，同时对寄存器操作数和存储器操作数进行编码，可能限制指令能够表示的寄存器个数。指令的时钟周期数因操作数来源有很大不同。

存储器-存储器型（SS型）：

优点：目标代码最紧凑，不需要设置寄存器保存变量。

缺点：指令字长变化很大，特别是三操作数指令。每条指令完成的工作差别很大。对存储器的频繁访问导致存储器成为瓶颈。

综合不同功能的指令，按照指令中地址字段的个数可将指令分为：**三地址指令、二地址指令、一地址指令和零地址指令。**

三地址指令两个源操作数内容不变

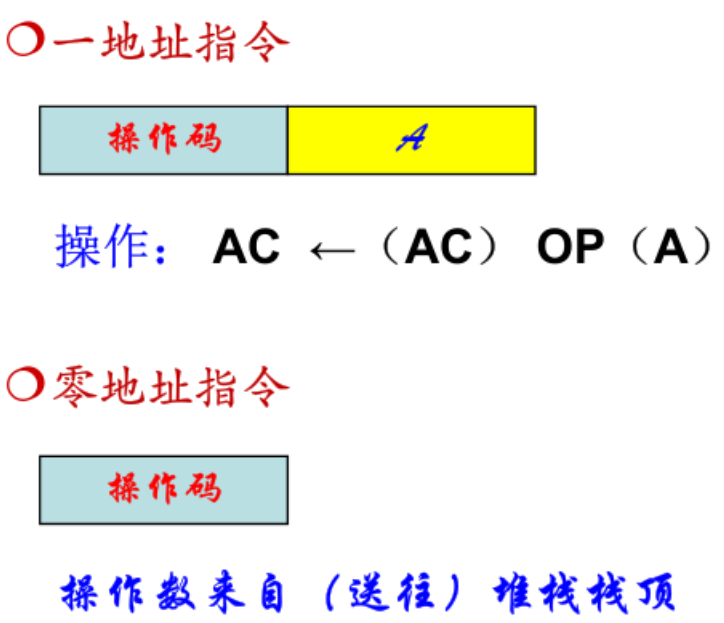
二地址指令通过减少一个地址字段减少了指令长度或增加地址码位数扩大寻址范围或者为操作码扩展技术提供支持。但是会改变一个源操作数的内容，影响程序灵活性。

这两种指令格式与通用寄存器组cpu组织呼应。

将源操作数或目的操作数设定到一个专用寄存器中，形成了**一地址指令格式。**支持一地址指令格式的计算机结构称为基于累加器结构。

省略二地址指令的两个地址码字段，得到**零地址指令。**通常所有源操作数和目的操作数都放在一个堆栈中。支持零地址指令格式的计算机称为堆栈机。

零地址指令也包括**空操作（NOP）、停机（HLT）**这类只有操作码的指令，也包括隐含的单操作数指令通常隐含在ACC或堆栈栈顶。



地址码个数选择标准有两个：

一个是程序所占的存储尽可能小（所有指令长度总和尽可能短）

二是执行速度尽可能快（访问主存储器的时间和尽可能短）

在CISC中，多种指令格式混合使用。RISC中除了(LOAD/STORE)是RS型，其他指令均为RR型，地址格式比较单一。

2.4.2指令字长

指令字长指一条指令中包含的**二进制代码位数。**它取决于**操作码的长度、地址码的长度和个数。**

早期计算机指令字长、机器字长、存储字长均相等，称为单字长指令。访问某个存储单元便可以取出一条完整的指令或者一个完整的数据。控制方式比较简单。

随着计算机的发展，一台机器的指令系统可以采用位数不同的指令，即指令字长是可变的，如单字长指令、多字长指令以及半字长指令。控制这类指令读取的电路比较复杂，且多字长指令需要多次访问存储器才能取出一条完整的指令，降低了指令执行的速度。

多种指令字长并存是CISC指令系统的特点，目的是提供功能丰富、数据类型和寻址方式多样的指令系统。

RISC指令寻址方式少，指令字长固定（通常选取单字长指令），指令格式种类少。

指令字长选取的基本原则:

指令字长尽可能短，以节省存储空间和提高处理速度。

指令中各种信息位利用率尽可能高，以有效压缩指令字长。

常见指令结构:

等长指令字结构：所有指令字长均相等，通常是指令字长等于机器字长。

变长指令字结构：各种指令长度不等。

IA-32采用变长指令字结构，指令字长是字节的整数倍，可取1-16字节。MIPS采用单字长的等长指令字结构，，指令字长为32位。

2.4.3操作码拓展技术

操作码字段

为了表示不同功能的指令，每一条指令安排唯一的操作码。操作码字段位数选取原则是能够表示指令系统的全部指令。

操作码长度

定长操作码：所有指令的操作码位数相同，并将操作码集中安排在指令字的一个固定字段中。

这样做硬件设计简单，指令译码结构简单，译码时间短，但是浪费了许多信息位。

变长操作码：各种指令的操作码位数不一致，并且操作码可以分散在指令字的不同字段中。

这种结构会增大指令译码和分析的难度，使控制器设计复杂，但是可以充分利用指令信息位，有效压缩操作码长度，广泛应用于字长较短的计算机中。

操作码拓展技术

基本思想：当采用定长指令字格式，且多种地址码结构混合使用时，可利用地址码个数较少的指令空出的地址码字段来增加操作码的位数。

操作码扩展方法

等长扩展法：

4-8-12扩展法：保留一个码点标志的（1111）15/15/15拓展法，每次保留一个标志位（0111）的8/64/512/8192拓展法

3-6-9扩展法

不等长扩展法

如4-6-10拓展法

<table>
<tr><th colspan="4">指令格式及操作码编码</th><th>说明</th></tr>
<tr><td>OP_Code</td><td>A1</td><td>A2</td><td>A3</td><td rowspan="2">4位操作码的三地址指令
15条</td></tr>
<tr><td>0000
0001
…
1110</td><td>A1</td><td>A2</td><td>A3</td></tr>
<tr><td colspan="2">OP_Code</td><td>A1</td><td>A2</td><td rowspan="2">8位操作码的二地址指令
15条</td></tr>
<tr><td>1111 0000
1111 0001
… …
1111 1110</td><td></td><td>A1</td><td>A2</td></tr>
<tr><td colspan="3">OP_Code</td><td>A</td><td rowspan="2">12位操作码的一地址指令
15条</td></tr>
<tr><td>1111 1111 0000
1111 1111 0001
… … …
1111 1111 1110</td><td></td><td></td><td>A</td></tr>
<tr><td colspan="4">OP_Code</td><td rowspan="2">16位操作码的零地址指令
16条</td></tr>
<tr><td>1111 1111 1111 0000
1111 1111 1111 0001
… … … …
1111 1111 1111 1111</td><td></td><td></td><td></td><td></td></tr>
</table>

在设计操作码时，具体采用哪种拓展编码格式要根据所设计的系统中各种指令出现的概率分布情况确定，衡量的标准是操作码的平均长度是否最短，或者信息的冗余量是否最小。

❒ MIPS 32 指令格式

32位定长指令格式，操作码字段也是固定长度，没有专门的寻址方式字段，由指令格式确定寻址方式。

(a) R-型指令

(b) I-型指令

(c) J-型指令

2.5寻址方式

计算源操作数、目的操作数以及下一条指令所在存储设备**具体位置**的方法称为寻址方式。

与地址相关的概念

形式地址：也称符号地址，通常指由指令中显式给出的地址。

有效地址：也称逻辑地址，它通常可以根据形式地址通过某种变换得到，变换规则由具体的寻址方式来确定。有效地址EA由寻址方式和形式地址共同来确定的，通过形式地址求有效地址所采用的算法就是寻址方式。

2.5.1寻址方式类型

寻址方式分为**指令寻址方式**和**数据寻址方式**两类。

指令寻址方式

确定下一条将要执行的指令所在主存单元地址的方法。其目的是为 PC设置新值。按照程序运行轨迹可分为：

顺序指令寻址

下一条指令地址由当前指令地址自增产生，即PC =（PC）+ 1

跳跃指令寻址

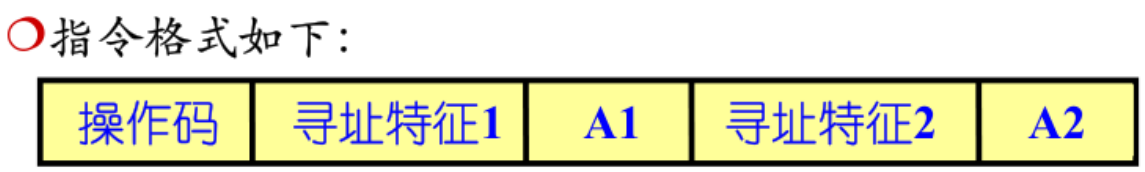
由转移类指令给出下一条指令的地址信息。 即PC内容按转移地址重新设置，而不是由PC顺序计数提供。

数据寻址方式

确定当前指令所涉及的操作数在数据存储设备中的地址的方法。

现代计算机中，寻址方式较多，为了正确、有效地获得操作数，通常在指令中安排几位标志位表示所用的寻址方式，称为“寻址方式码”或“寻址特征码”。

指令格式如下：

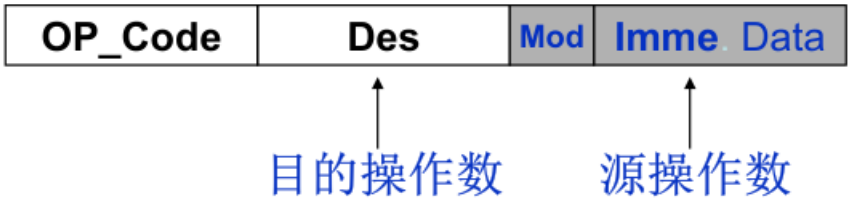


数据寻址方式可分为基本寻址方式和复合寻址方式。

基本寻址方式根据数据所在的位置不同可分为四大类：立即寻址、寄存器寻址、存储器寻址和堆栈寻址。

（1）立即寻址方式

所需的操作数在指令的地址码部分直接给出，操作数直接在指令本身，称为立即数，通常立即数用补码表示。



立即寻址通常只适用于源操作数，，用于给寄存器赋值或者作为常数用于运算。

优点是只要取出指令就立即获得操作数，执行阶段不必再访问存储器，指令执行速度快。

缺点是I的位数限制了立即数的表示范围，数据的长度不能太长，大量使用立即寻址会导致程序通用性下降。

Operand = Imme. Data
例如，IA-32中，mov eax,100
MIPS 32 中， addi $s1,$s2,100

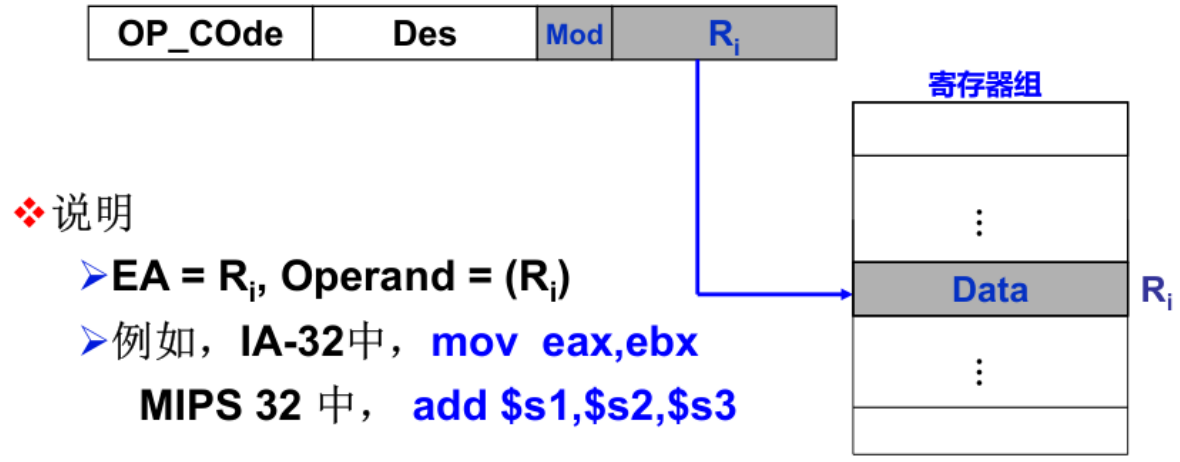
IA-32中采用的立即寻址方式可以给寄存器或内存单元赋值，立即数也可以参与运算

MIPS中I-型指令就是采用立即寻址方式。

（2）寄存器寻址方式

操作数在寄存器中，指令执行速度快。

地址码字段的形式地址部分给出通用寄存器编号 Ri，地址码字段短。



寄存器寻址方式为获取操作数无需访问内存，指令执行时间短，同时只需标注寄存器编号，指令字长短，节省存储空间。

这种寻址方式在所有的RISC和大部分CISC中得到广泛应用。

（3）存储器寻址方式

根据有效地址的不同形成方法，又分为：

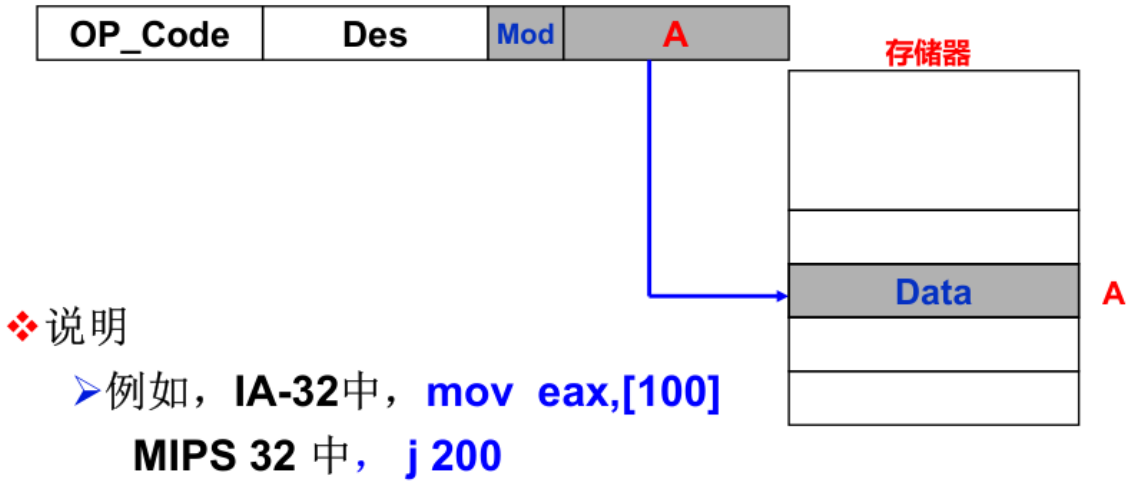
直接寻址，存储器间接寻址，寄存器间接寻址，偏移寻址以及段寻址

这些寻址方式也可以用于跳跃执行的指令寻址

直接寻址方式

指令地址字段直接给出操作数在存储器中的地址

寻址速度快，但寻址范围小。

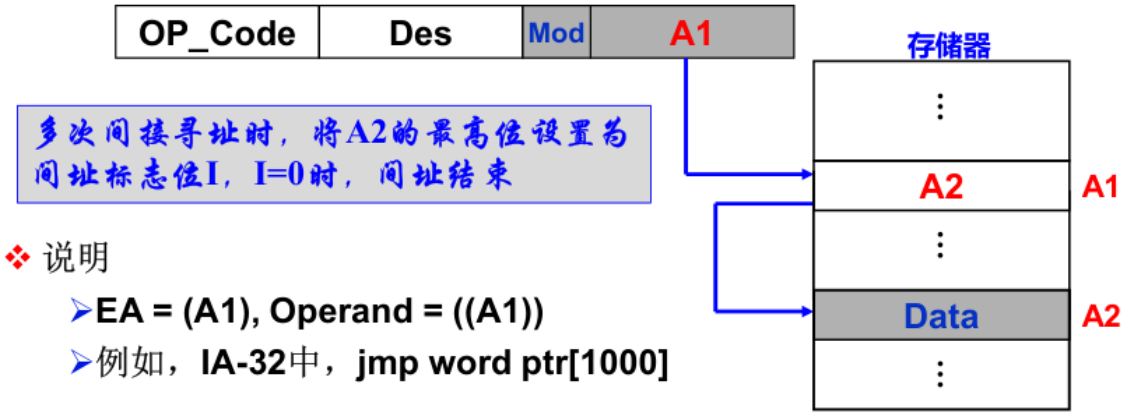


间接寻址方式

地址码字段给出的内容既不是操作数，也不是操作数的地址，而是操作数地址的地址。

分为一次间接和多次间接寻址。

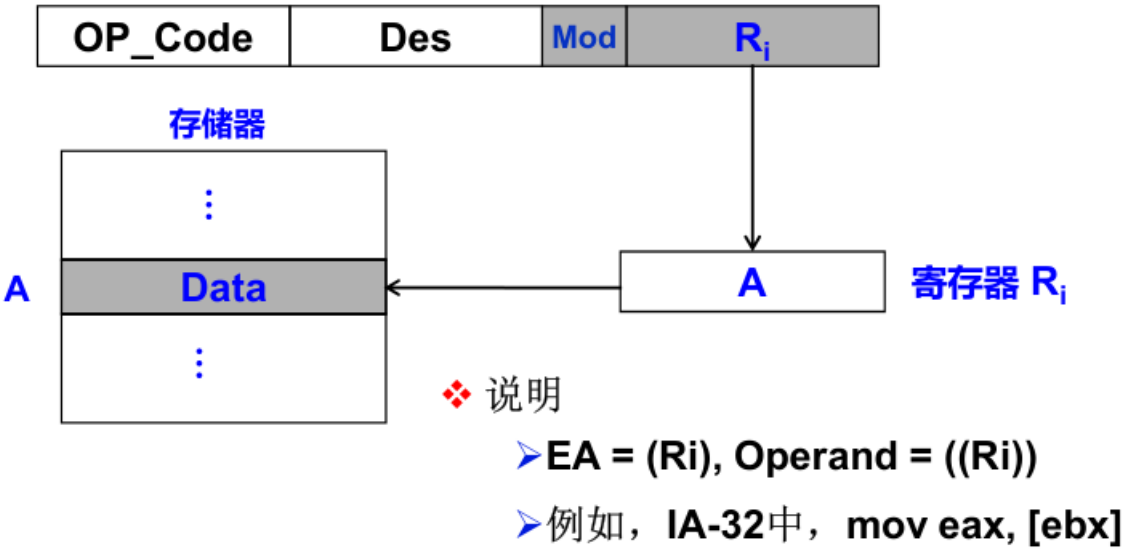
可扩大指令寻址范围，但指令执行速度慢。



寄存器间接寻址方式

地址码字段给出某一通用寄存器的编号

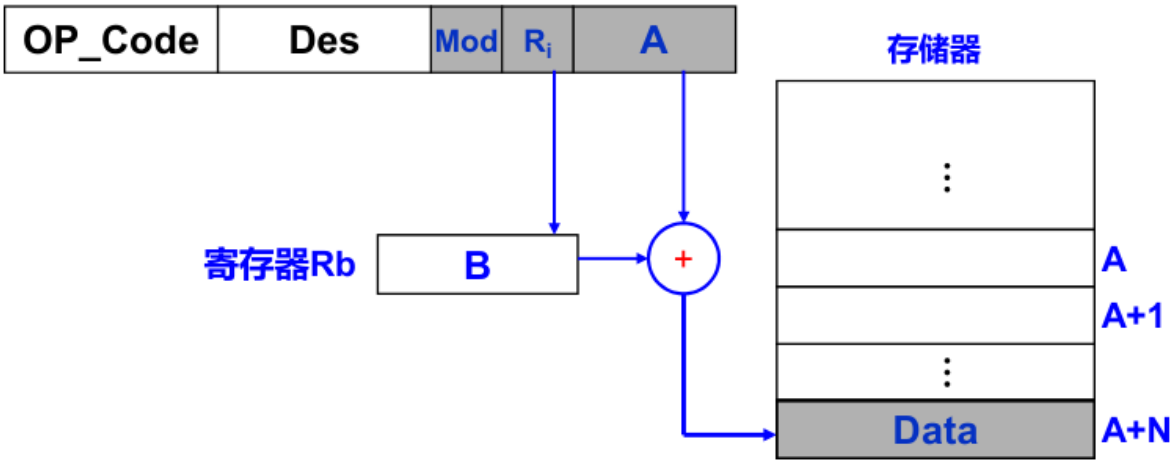
该寄存器中存放的是操作数在主存单元的地址。



偏移寻址方式

形式上可认为是直接寻址和寄存器间接寻址的结合

地址码字段既要给出形式地址，也要指出引用哪一个寄存器 Ri内容实现偏移。
EA=（Ri）+ A



常用的偏移寻址方式：

相对寻址
引用专门的程序计数器 PC，即 EA =（PC）+A

指令中只需要给出偏移量 A

有利于程序在内存中浮动

变址寻址

引用一个变址寄存器 Rx, EA =（Rx）+ A

变址寄存器 Rx 可以是专用寄存器或通用寄存器

若变址寄存器 Rx 是专用寄存器，地址字段中就不需要指出该寄存器（默认）

若变址寄存器 Rx是通用寄存器，地址字段要给出寄存器的编号 Ri

基址寻址

引用一个基址寄存器 Rb, EA =（Rb）+ A

基址寄存器 Rb 可以是专用寄存器或通用寄存器

若基址寄存器 Rb 是专用寄存器，地址字段中就不需要指出该寄存器（默认）

若基址寄存器 Rb 是通用寄存器，地址字段要给出寄存器的编号 Ri

变址寻址与基址寻址的区别：

基址寄存器中存放基地址，一旦由系统设定一般用户不能改变，程序中指令或数据的改变由不同的位移量完成；

变址寄存器存放地址修改量（变址值），而形式地址给出基本地址值，操作数地址的变化由变址值增、减量完成（典型的变址寻址其变址值的增、减量由硬件自动完成）；

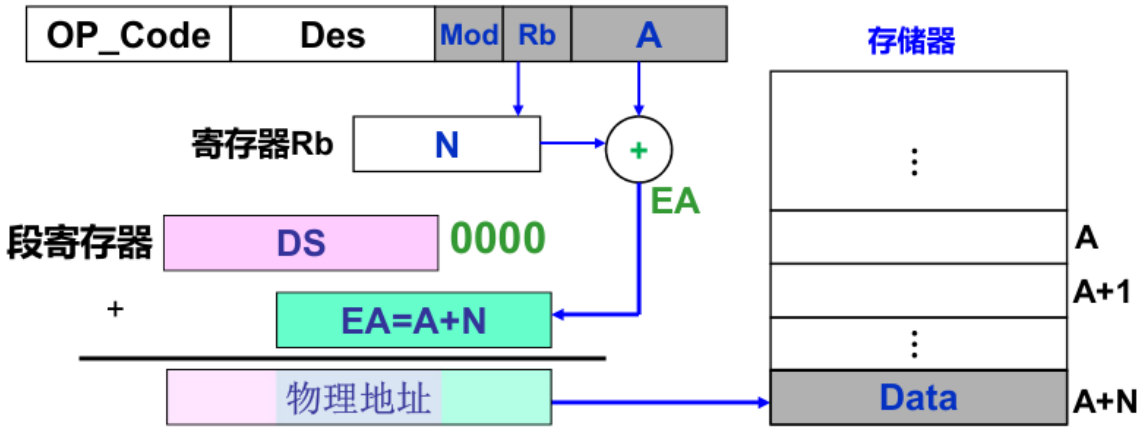
基址寄存器的内容通常由系统程序设定，而变址寄存器的内容通常由用户设定；

基址寻址常用来实现对用户程序的动态定位，而变址寻址常用于数组处理及串操作。

段寻址方式（基址寻址的特例）

用于地址长度超过机器字长的场合

将与机器字长相等的段地址和段内位移量错位相加，以获得更长的主存地址。



（4）堆栈寻址方式

堆栈是一种后进先出（LIFO）的存储装置

有寄存器堆栈（硬堆栈）和存储器堆栈（软堆栈）

存储器堆栈是在主存中开辟一块区域，该区域一端固定，称为栈底；另一端是浮动的，称为栈顶。

栈顶是数据唯一的出入口。堆栈指针（SP）始终指向栈顶。

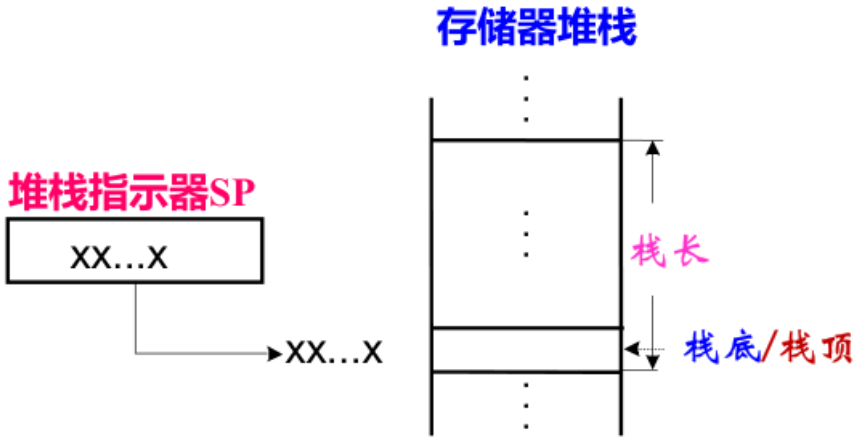
栈底与栈顶的地址设定方法：

栈底设在栈区的**低址端**，栈顶设在**高址端**，堆栈向上生长。

栈底设在栈区的**高址端**，栈顶设在**低址端**，堆栈向下生长。

❒ 堆栈的实现

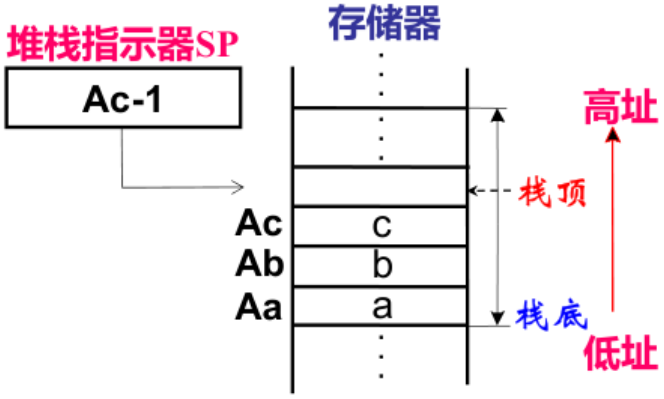
❍SP指向栈顶空单元



空栈情况

❒ 堆栈的实现

❍SP指向栈顶空单元



堆栈操作

压栈操作（PUSH）：

(SP) ← 数据

SP ← (SP) +1

出栈操作（POP）：

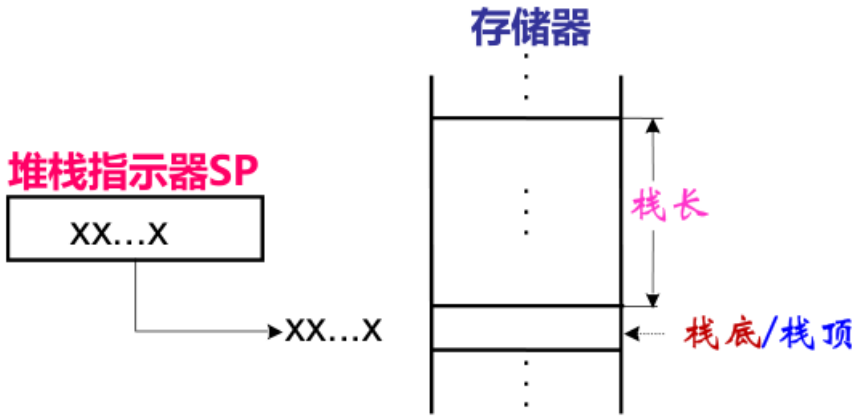
SP ← (SP) - 1

[(SP)] 出栈

非空栈情况

❒ 堆栈的实现

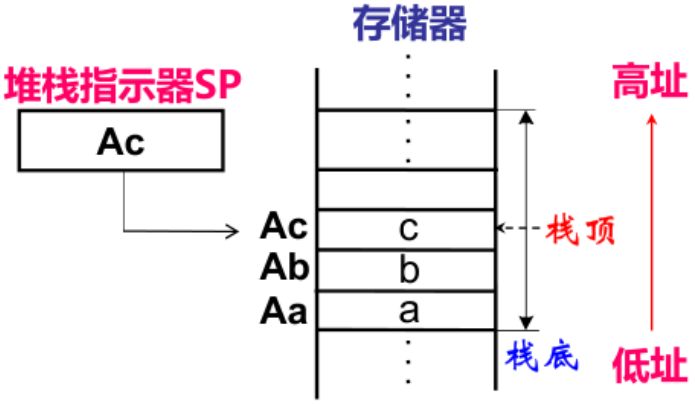
❍SP指向栈顶非空单元



空栈情况

❒ 堆栈的实现

❍SP指向栈顶非空单元



堆栈操作

压栈操作（PUSH）：

SP ← (SP) + 1

(SP) ← 数据

出栈操作（POP）：

[(SP)] 出栈

SP ← (SP) - 1

非空栈情况

(5) 复合寻址方式

两种以上寻址方式联合使用，称为复合寻址

关键是地址计算的顺序，习惯从名称上加以反映

变址间接寻址：先变址，后间接。即EA=[(Rx)+A]， Rx ← (Rx)+□

间接变址寻址：先间接，后变址。即EA= (Rx)+(A)， Rx ← (Rx)+□

以上基本寻址方式的介绍是原理性的，不涉及具体机器。这些寻址方式在各种实际机器中实现时有许多变通的命名和规则，在使用时应遵循实际机器汇编语言的具体规定。

2.5.2程序定位方式

定位方式是指程序中指令和数据的逻辑地址到物理地址的变换时间和实现方式。

程序定位方式：

直接定位方式

直接使用主存物理地址来编写或编译程序，即无需地址变换。

将主存物理空间划分为若干个固定且大小相同的分区，为每个任务分配相应的分区。

若程序比较大超出了分配给它的主存物理空间，把它分割成若干个程序段，在程序运行过程中逐段调入主存物理空间，称为“覆盖”。

静态定位方式

在程序加载到主存时，一次性为指令和数据分配主存物理地址。

由加载程序（操作系统程序）完成定位功能。

若程序比较大超出了分配给它的主存物理空间，采用“覆盖”技术。

程序多次运行时可以装入到不同的主存物理空间，一旦装入，执行期间不能在主存中移动。

动态定位方式

在程序执行过程中，进行逻辑地址到物理地址的转换。

需要硬件支持，采用基址寻址方式实现。

一个程序可以被分配在多个不连续的主存物理空间内。

多个程序可以共享存放在主存中的同一个程序段。

支持虚拟存储器，为用户提供一个比实际主存储器的物理空间大得多的逻辑地址空间。

思考题和习题

1.对CPU中的寄存器、主存储器以及IO设备采用同一编址或者独立编址对指令系统的设计有哪些影响?

统一编址（Memory-Mapped I/O）

在统一编址模式下，寄存器、主存储器和I/O设备都共享一个统一的地址空间，也就是说它们都通过相同的地址总线进行访问。这种方式的特点和影响包括：

简化的指令集设计：

因为所有的存储单元和I/O设备都通过相同的地址空间访问，CPU只需要一套指令来处理所有的访问操作。读取或写入I/O设备与访问内存和寄存器没有本质的区别，通常只需要特定的地址范围区分即可。

这使得指令集的设计相对简单，因为I/O操作可以像普通的内存操作一样处理，不需要专门的I/O指令。

灵活性高：

可以通过简单地访问特定的内存地址来控制I/O设备或获取数据，这种统一的模型使编程更加灵活和直观。

当需要访问一个新的I/O设备时，只需分配一个新的地址区间即可，不需要额外的硬件或软件上的改变。

性能潜在下降：

由于I/O设备通常较慢，与内存共用总线可能导致总线带宽的竞争，从而影响系统性能。特别是在高I/O操作频繁的情况下，这种带宽竞争可能成为系统瓶颈。

缓存一致性问题也会变得复杂，因为I/O设备和内存可能同时被缓存，处理同步需要更多的设计考虑。

独立编址（Isolated I/O）

在独立编址模式下，寄存器、主存储器和I/O设备各自拥有独立的地址空间，通常使用不同的指令和总线来进行访问。这种方式的特点和影响包括：

复杂的指令集：

因为需要区分对内存和对I/O设备的访问，指令集必须包含专门用于I/O操作的指令，例如IN和OUT指令。这增加了指令集的复杂性。

CPU需要更多的指令来处理不同类型的操作，可能会导致指令系统的设计更加复杂和庞大。

性能优势：

由于I/O设备和内存使用独立的总线系统，I/O操作不会与内存操作直接竞争带宽，从而可能提高系统的整体性能。

独立的I/O操作可以避免一些与内存相关的缓存一致性问题，这简化了缓存系统的设计。

灵活性降低：

独立编址使得程序需要明确区分内存和I/O操作，在编程时需要使用特定的指令来访问I/O设备，增加了编程复杂性。

添加新设备时，需要更新指令集或者设计额外的硬件接口，相比于统一编址系统，灵活性有所降低。

2.请描述主存储器按字节编址方式的特点，并分析主存储器数据存放采用边界对齐和边界不对齐的优缺点。

主存的最小编址单位是一个字节，描述主存储器容量时，以字节（byte，b）为单位

对主存数据既能以字节为单位访问，也能以字为单位访问

按字节访问主存时，使用字节地址；按字访问主存时，使用字地址

通常存储字长是字节的整数倍，字节地址是连续的，字地址是不连续的

多个字节数据存放在一个字单元，有两种编址顺序：低字节低地址（小端方式）、高字节低地址（大端方式）；也有存放边界问题：边界对齐、边界不对齐。

对于边界对齐方式，规定了字节、半字、单字或者双字存放的起始位置。优点是无论访问一个字节、一个半字或一个单字都可以在一个存储周期完成，读写数据的控制简单，但是可能会造成空间的浪费。

对于边界不对齐方式，这是一种不浪费存储空间的存储方式，不同长度的数据一个接一个存放，但是这种存放方式储存在两个问题：一是除了访问一个字节外，其他数据可能跨两个字单元存放，有可能花费两个存储周期，主存储器访问速度降低一半；二是存储器的读写控制比较复杂。

3.请说明在指令格式设计时，分别采用等长指令字结构和变长指令字结构的优缺点。

等长指令字结构

等长指令字结构是指所有指令的长度相同，通常是一个固定的字长（如32位、64位等）。这种结构主要用于RISC（精简指令集计算机）架构。

优点：

指令解码简单、快速：

由于所有指令长度相同，CPU可以在每个时钟周期内解码和执行一条指令，而不需要判断指令的长度或进行复杂的分支操作。指令流水线设计也因此变得更加简单高效。

硬件设计简化：

等长指令字使得指令提取和解码逻辑较为简单，硬件电路设计相对容易，减少了设计和实现的复杂性。

高效的流水线处理：

在等长指令的情况下，指令流水线可以高效地工作，因为每个流水线阶段都可以在固定的时钟周期内处理一部分指令，不会因为指令长度的差异而出现空转或气泡。

指令执行的预测性：

由于指令长度固定，指令的执行时间较为一致，有助于提高指令执行的预测性，优化CPU的性能。

缺点：

指令编码效率较低：

对于简单指令（如加载、存储、加法等），固定长度的指令可能会浪费位空间。例如，一个简单的操作可能只需要很少的位来编码，但由于指令长度固定，多余的位就会浪费掉。

可表达性有限：

由于所有指令长度相同，复杂操作可能无法在单条指令中完全表达，必须通过多条指令来实现，这可能导致代码长度增加，从而影响整体执行效率。

变长指令字结构

变长指令字结构是指指令的长度可以不同，长度由指令的复杂度决定。CISC（复杂指令集计算机）架构通常采用这种结构。

优点：

指令编码效率高：

变长指令允许根据操作的复杂性灵活分配位数，简单指令可以用较短的编码，复杂指令则使用更长的编码，从而提高指令编码的效率，减少程序的整体大小。

丰富的指令集：

变长指令格式允许设计更复杂和功能更强大的指令，能够直接支持更复杂的操作，减少了执行复杂任务所需的指令数量，从而可能提高整体程序的执行效率。

程序紧凑性：

由于短指令可以用较少的字节表示，整体程序的体积可能较小，这对存储和传输有利，尤其在存储空间有限的环境中（如嵌入式系统）。

缺点：

指令解码复杂：

由于指令长度不固定，CPU在解码时需要判断每条指令的长度，这增加了解码的复杂性，可能需要额外的时钟周期，降低指令解码的速度。

流水线设计复杂：

变长指令导致流水线处理的复杂性增加，指令长度的变化可能导致流水线阶段出现空转或气泡，影响流水线的效率。

硬件实现复杂：

变长指令需要更加复杂的硬件电路来支持灵活的解码和执行，这可能增加芯片设计和制造的复杂性以及功耗。

执行时间不确定性：

不同长度的指令可能需要不同的执行时间，导致指令执行的不可预测性，进而影响CPU性能的优化。

4.一般来说，CISC比RISC的指令复杂，因此可以用较少的指令完成相同的任务。然而，由于指令的复杂，一条CISC指令需要花费比RISC更多的时间来完成。假设一个任务需要P条CISC指令或者2P条RISC指令，完成每条CISC指令花费8Tns，每条RISC指令花费2Tns。在此假设下，哪一种指令系统性能更好?

P×8 > 2P×2

RISC性能更好

5.ASCII码是7位，如果设计主存单元字长为31位，指令字长为12位，是否合理？为什么?

此设计方案不合理。其原因是：

① ASCII码是7位，通常加一位校验位为8位，以字节为单位进行处理比较方便。故主存应设计成按字节编址，这种编址方式下一般主存单元字长应取字节长度的2、4、8倍。若按8位标准字节设计，主存字长取32位比较合适，取31位显然不合理。

② 一般指令字长应与机器字长或字节长度间有整数倍关系，若主存设计成按字节编址方式，则指令字长取单字节、双字节等较合适，取12位显然不合理。

6.在某些计算机中，子程序调用是以下述方法实现的：转子指令将返回地址（即主程序中该指令的下一条指令地址）存入子程序的第一单元，然后转到第二个单元开始执行子程序。

（1）设计一条相应的从子程序返回主程序的指令；

（2）在这种情况下，你怎样在主、子程序间进行参数的传递?

（3）上述调用方法是否可用于子程序嵌套?

（4）上述调用方法是否可用于子程序多重嵌套时的递归调用（即某个子程序调用它本身）？如果改用堆栈链接方法，是否可实现此问题?

(1) 返回指令是一地址指令，其格式如下：

JMP	I	K

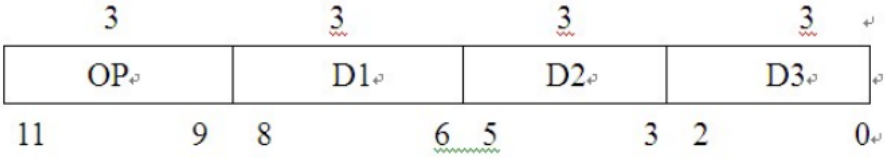
这是一条间接寻址的无条件转移指令。其中，I为间接寻址标志，K为子程序在主存第一单元的地址。

(2)在这种情况下，可利用寄存器或主存单元进行主、子程序之间的参数传递。

(3)可以用于子程序的嵌套(多重转子程序)，因为每个返回地址都存放在被调用的子程序的第一个单元中。

(4)不可以用于子程序的递归，因为当某个子程序自己调用自己时，子程序的第一个单元的内容将被破坏。如果改为堆栈方法．可以实现子程序的递归，因为堆栈具有后进先出的功能。

7.设某指令系统基本指令格式如下图示。图中，指令总字长12位，其中OP表示操作码字段，占3位；Di（i=1、2、3）表示地址码字段，每个分别占3位。请利用扩展操作码法，试提出一种编码方案使该指令系统有5条三地址指令，8条二地址指令，120条单地址指令，60条零地址指令。要求具体分配每条指令的操作码编码。



解：该指令系统操作码编码分配方案如下：

000 XXX YYY ZZZ
↓ ↓ ↓ ↓
100 XXX YYY ZZZ　　5条三地址指令

101 000 YYY ZZZ
↓ ↓ ↓ ↓
101 111 YYY ZZZ　　8条二地址指令

110 000 000 ZZZ
↓ ↓ ↓ ↓
110 111 111 ZZZ

111 000 000 ZZZ
↓ ↓ ↓ ↓
111 110 111 ZZZ　　120条单地址指令

111 111 000 000
↓ ↓ ↓ ↓
111 111 110 111
111 111 111 000
↓ ↓ ↓ ↓
111 111 111 011　　60条零地址指令

111 111 111 100
↓ ↓ ↓ ↓
111 111 111 111　　4条冗余编码备用

8.某32位计算机，CPU中有32个通用寄存器，主存容量为4GB。指令字长等于机器字长，若该机指令系统可完成138种操作，操作码位数固定，且具有立即寻址、直接寻址、间接寻址、寄存器间接寻址、变址寻址、基址寻址和相对寻址7种方式，试回答：（要求：答案中数据分别用2的幂形式表示）

（1）画出一地址指令格式，并指出各字段的作用；

（2）该指令立即数的最大范围；

（3）直接寻址的最大范围；

（4）一次间接寻址和多次间接寻址的寻址范围；

（5）寄存器间接寻址的范围；

（6）分别采用专用寄存器和通用寄存器作为变址寄存器时，变址寻址的位移量范围；

（7）分别采用专用寄存器和通用寄存器作为基址寄存器时，基址寻址的位移量范围；

（8）相对寻址的位移量。

（1）

（2）立即数最多可以用21位，所以最大取值范围：

$-2^{20} \sim 2^{20}-1$(有符号数)或者$0 \sim 2^{21}-1$ (无符号数)

（3）直接寻址时，形式地址位数最多21位，所以寻址最大分范围：

$2^{21}B = 2MB$

（4）一次间接寻址的寻址范围：4GB

多次间接寻址的寻址范围：2GB

（5）寄存器间接寻址的范围：4GB

（6）采用专用寄存器作为变址寄存器时，变址寻址的位移量范围：4GB

采用通用寄存器作为变址寄存器时，变址寻址的位移量范围：4GB

（7）采用专用寄存器作为基址寄存器时，基址寻址的位移量范围：

$-2^{20} \sim 2^{20}-1B$，即-1M ~ 1M-1B

采用通用寄存器作为基址寄存器时，基址寻址的位移量范围：

$-2^{15} \sim 2^{15}-1B$，即-32K ~ 32K-1B

（8）相对寻址的位移量：$-2^{20} \sim 2^{20}-1B$，即-1M ~ 1M-1B

9.某16位机，采用单字长单地址指令格式，其中形式地址码字段占7位。若基址寄存器的内容为2000H，变址寄存器的内容为23A0H，指令的形式地址码部分是3FH，当前正在执行的指令所在地址为2B00H。

请回答下列问题：

（1）变址寻址、基址寻址和相对寻址三种情况下的访存有效地址；

（2）设变址寻址用于取数指令，相对寻址用于转移指令，存储器内存放的相关内容如下

地　址	内　容
003FH	2300H
2000H	2400H
203FH	2500H
2B3FH	2600H
23A0H	2700H
23DFH	2800H
2B00H	063FH

请写出从存储器中所取的数据以及转移地址。

（3）若采用直接寻址，请写出从存储器中取出的数据。

（1）相对寻址：EA = （PC） + A = 2B01H + 003FH = 2B40H

变址寻址：EA = （Rx） + A = 23A0H + 003FH = 23DFH

基址寻址：EA = （Rb） + A = 2000H + 003FH = 203FH

（2）变址寻址时EA = 23DFH 因此从23DFH单元取出的数据为2800H；相对寻址时转移地址 = 2B40H

（3）直接寻址时EA = 003FH，S=(EA)=2300,因此取出的数据为2300H

10.某计算机字长16位，主存按字编址，采用单字长单地址指令格式，指令的一般格式如下所示：

OP Code	I	X	D
操作码	间址位	基址寄存器号	形式地址

用户程序中某条指令K格式如下：

θ	1	3	401

主存某几个单元的内容如下：（参数均为十进制表示）

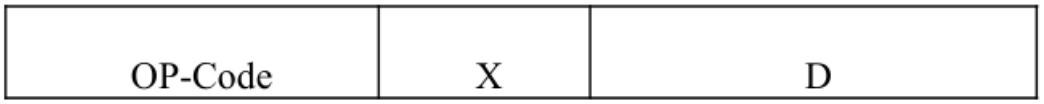
地　址	内　容
⋮	⋮
4016	3528
⋮	⋮
4300	2053
⋮	⋮
4416	1764
4417	4300
⋮	⋮

若3号基址寄存器内容是4016，试用先基址后间址（一次）的复合寻址方式，求指令K的操作数P。

先基址寻址：EA’=（Rb） + A= 4016 + 401 = 4417

后间接寻址：EA=(EA’) =(4417)=4300　　P = (4300)=2053

11.某计算机字长16位，主存按字编址，采用单字长单地址指令格式，其格式如下所示：



OP-Code：操作码。D：形式地址。

X：寻址方式码，X=00：直接寻址；

X=01：用变址寄存器X1变址；

X=10：用变址寄存器X2变址；

X=11：相对寻址；

若执行指令时，机器状态如下：

（PC）=1548H，（X1）=036AH，（X2）=46B2H

请分别确定下列指令的有效地址EA。

① 3056H ② 42A0H ③ 1347H ④ 4598H ⑤ 67CEH

①指令码=0011 0000 0101 0110，

直接寻址EA=D=0101 0110B = 0056H

②指令码=0100 0010 1010 0000，

用变址寄存器X2变址EA=（X2）+D = 46B2H + A0H= 4752H

③指令码=0001 0011 0100 0111，

相对寻址EA=（PC）+D = 1548H + 47H=158FH

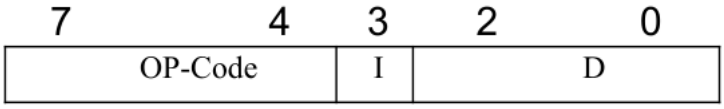
④指令码=0100 0101 1001 1000，

用变址寄存器X1变址EA=（X1）+D = 036AH + 98H=0402H

⑤指令码=0110 0111 1100 1110，

相对寻址EA=（PC）+D = 1548H + FFCEH=1516H

12.某8位计算机，其指令格式如下图所示：



其中，OP-Code为操作码；I为间址特征位，只允许一次间址；D为形式地址。假设主存储器部分单元内容如下：

地　址	内　容
00H	9DH
01H	04H
02H	A4H
03H	5EH
04H	15H
05H	76H
06H	B8H
07H	23H

指出下列指令的有效地址：

① A7H ② DFH ③ B2H ④ CEH

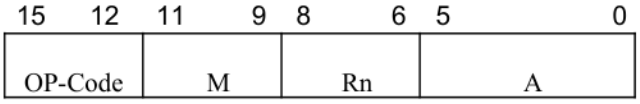
①指令码=1010 0111，直接寻址 EA=D=07H

②指令码=1101 1111，间接寻址 EA=(D)=(07H)=23H

③指令码=1011 0010，直接寻址 EA=D=02H

④指令码=1100 1110，间接寻址 EA =(D)=(06H)=B8H

13.某计算机字长16位，主存按字编址，采用单字长单地址指令格式，指令各字段定义如下：



其中，OP-Code为操作码，M为寻址方式码，Rn为通用寄存器编号，A为形式地址。寻址方式码定义如下：

M	寻址方式	有效地址表达式
000B	一次间接	EA=（A）
001B	寄存器间接	EA=（Rn）
010B	变址	EA=（Rn）+A，Rn←（Rn）+1
011B	相对	EA=（PC）+A

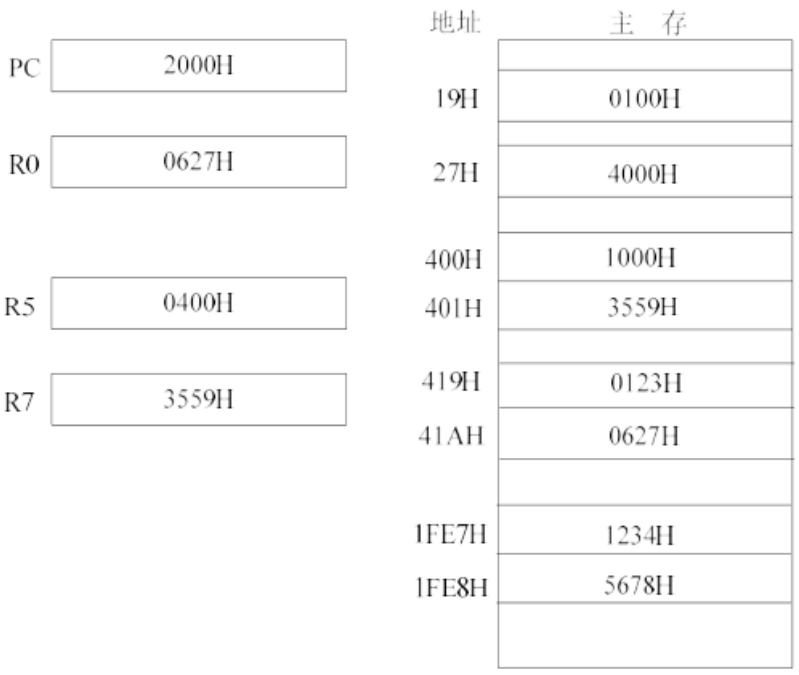
注：有效地址表达式中（X）表示存储器地址X或寄存器X的内容；指令中Rn字段和A字段是否使用视寻址方式而定；位移量用补码表示。

请回答下列问题：

(1)该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？

(2)上表中各种寻址方式的寻址范围多大（不包括相对寻址）？相对寻址的浮动范 围多大?

(3)设开始取指令时，对应寄存器和主存相关单元的内容如下图，图中的数字均为十六进制表示，请写出指令
0627H和3559H的操作数各为多少？分别单独执行这两条指令后相关寄存器的内容各是多少?



题解：

- **1、该指令系统最多可有2^4=16条指令，该计算机最多有2^3=8个通用寄存器**
- **2、一次间接寻址范围：2^{16}=64K字**
  寄存器间接寻址范围：2^{16}=64K字
  变址寻址范围：-2^{15}~ $+2^{15}$ =-32K~+32K字（Rx内容为偏移量，带符号整数）
  相对寻址的浮动范围=-32~+31字
- **3、a、指令0627H展开：0000 011 0 00 10 0111B**
  OP=0000B，M=011B=相对寻址，
  Rn=000B（无用），A=10 0111B（负数补码）
  EA=(PC)+A=2001H+FFE7H=1FE8H
  （取指后(PC)+1，且A符号扩展）
  操作数a=（EA）=（1FE8H）=5678H
  指令执行后：（PC）=2001H

b、指令3559H展开：0011 010 1 01 01 1001B

OP=0011B，M=010B=变址寻址，

Rn=101B（R5），A=01 1001B（正数补码）

EA=(R5)+A=0400H+0019H=0419H（A符号扩展）

操作数b=（EA）=（0419H）=0123H

（R5）=（R5）+1=0401H

指令执行后：（PC）=2001H，（R5）=0401H

14.某机字长16位，主存容量为1M字，采用单字长指令格式，共有50条指令，采用立即寻址、直接寻址、间接等寻址方式。CPU中有PC，IR，MAR，MDR等专用寄存器和4个通用寄存器。问：

（1）指令格式如何安排？

（2）立即寻址的数据范围是多大？

（3）为使指令能寻址到主存的任一单元，可采取什么措施？

（4）能否增加其它寻址方式？

（1）据题意，该机指令应能表示出50种操作码，3种以上寻址方式，该指令格式为单字长单地址，如下图示

其中，寻址方式码M分配如下：

M = 00， 直接寻址， EA = D

M= 01， 间接寻址， EA=（D）

M= 10， 立即寻址

M= 11， 备用

（2） $-2^7 \sim 2^7-1$(有符号数)或者$0 \sim 2^8-1$ (无符号数)；

（3）由于机器字长限制，上述格式求出的有效地址EA 为8～16位长，但题意所给主存容量为1M，需20位地址。要将EA扩展成20位主存实际地址，还需使用段寻址方式。为简化设计，在此设段寻址方式为默认的，既无需指令格式给出，由硬件隐含完成。设硬件配置有段寄存器DS，其长度 = 字长 = 16位，其内容为段地址，则：

$$物理地址 =（DS）\times 2^4 + EA$$

由此式可得20位主存物理地址。

（4） 由于剩一种寻址方式码未用，故在寻址方式码M位数不增加的前提下，还可增加一种寻址方式。

例如： M=11， 相对寻址， EA =（PC）+ D

（1）据题意，该机指令应能表示出50种操作码，3种以上寻址方式，该指令格式为单字长单地址，如下图示：

6位	3位	2位	5位
OP_Code	Mod	R	D
操作码	寻址码	通用寄存器号	形式地址

其中，寻址方式码M分配如下：

M = 000， 直接寻址， EA = D

M= 001， 间接寻址， EA=（D）

M= 010， 立即寻址

M= 其它， 备用。

上述格式中未给PC，IR，AR，DR四个寄存器分配地址码，这是因为这四个寄存器是专用寄存器，硬件会自动访问，不需编程指定。

（2） $-2^4 \sim 2^4-1$(有符号数)或者$0 \sim 2^5-1$ (无符号数)；

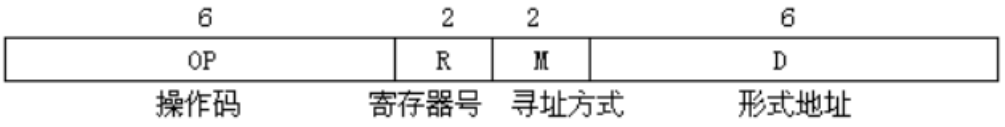
（3）由于机器字长限制，上述格式求出的有效地址EA 为5～16位长，但题意所给主存容量为1M，需20位地址。要将EA扩展成20位主存实际地址，还需使用段寻址方式。为简化设计，在此设段寻址方式为默认的，既无需指令格式给出，由硬件隐含完成。设硬件配置有段寄存器DS，其长度 = 字长 = 16位，其内容为段地址，则：

$$物理地址 =（DS）\times 2^4 + EA$$

由此式可得20位主存物理地址。

（4） 由于剩5种寻址方式码未用，故在寻址方式码M位数不增加的前提下，还可增加5种寻址方式。例如，寄存器，寄存器间接寻址和偏移寻址。

（1）据题意，该机指令应能表示出50种操作码，3种以上寻址方式，可寻址4个通用寄存器和主存，因此，该指令格式可以设计为RS型单字长二地址指令。如下图示：



其中，寻址方式码M分配如下：

M = 00， 直接寻址， EA = D

M= 01， 间接寻址， EA=（D）

M= 10， 立即寻址

M= 11， 备用。

上述格式中未给PC，IR，AR，DR四个寄存器分配地址码，这是因为这四个寄存器是专用寄存器，硬件会自动访问，不需编程指定。

（2） $-2^5 \sim 2^5-1$(有符号数)或者$0 \sim 2^6-1$ (无符号数)；

（3）由于机器字长限制，上述格式求出的有效地址EA 为6～16位长，但题意所给主存容量为1M，需20位地址。要将EA扩展成20位主存实际地址，还需使用段寻址方式。为简化设计，在此设段寻址方式为默认的，既无需指令格式给出，由硬件隐含完成。设硬件配置有段寄存器DS，其长度 = 字长 = 16位，其内容为段地址，则：

$$物理地址 =（DS）\times 2^4 + EA$$

由此式可得20位主存物理地址。

（4） 由于剩一种寻址方式码未用，故在寻址方式码M位数不增加的前提下，还可增加一种寻址方式。

例如： M=11， 相对寻址， EA =（PC）+ D

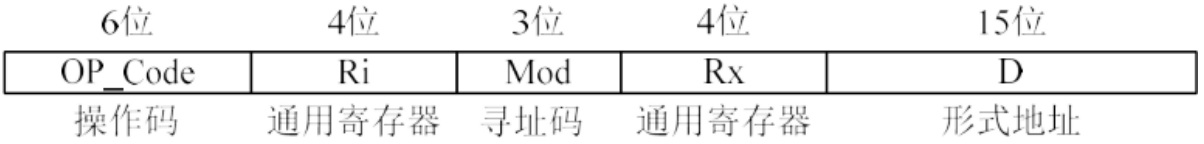
15.设某机字长32位，CPU中有16个32位的通用寄存器，主存按字编址，欲设计一种能容纳64种操作的指令系统，存储器寻址可提供8种方式，采用通用寄存器作变址寄存器，若取指令字长与机器字长相等，请安排RS型指令的格式，并回答下述问题：

（1）如果采用直接寻址方式，指令可寻址的最大存储空间是多少？

（2）如果采用一次间接寻址方式，指令可寻址的最大存储空间是多少？

（3）如果采用变址寻址，指令可寻址的最大存储空间又是多少？

据题意，指令格式可安排如下：



(1) 直接寻址时，不需要指出变址寄存器，所以形式地址可以扩展为19位，EA = D，则指令可寻址的最大存储空间2^{19} = 512K字。

(2) 间接寻址时，也不需要指出变址寄存器，所以形式地址也是19位，EA =（D），则指令可寻址的最大存储空间是2^{32} = 4G字。

注意：EA的位数与存储字长有关，与形式地址D的长度无关。

(3) 变址寻址时，EA =(Rx)+ D，则该RS型指令的最大存储空间是-2^{31}~ 2^{31}字（4G字）。注意：EA的位数仅与Rx的位数有关，与形式地址D的长度无关。Rx内容为偏移量（有符号整数）

16.对于一个按字节编址的存储器，存储字长32位。请问：

（1）第42个字的字节地址是什么？

（2）单字长数据0xFF223344按照大端或小端方式存储在第42个字中，画出数据在主存中放置的示意图，并标出与每个字节数据对应的字节地址。

题解：

- (1). 通常说的编号是从0开始连续编号的，所以第42个字的字节地址是41 * 4 = 164。
- (2). 大端

小端

FF	22	33	44
164	165	166	167

FF	22	33	44
167	166	165	164

17.在某32位计算机中，存储器按字节编址，采用小端方式存放数据。假设C语言编译器规定int型和short型长度分别为32位和16位，并且数据按边界对齐存储。某C语言程序段如下：

```
struct{
        char  x;
        short  y;
        int  z;
 } data;
data.x='0';
data.y=1026;
data.z=258;
```

若程序加载时，将data分配在以0x0A000012为首地址的主存区域内，请画图示意该主存区域中存放的数据值及对应的地址编码，要求用16进制表示。

解答：

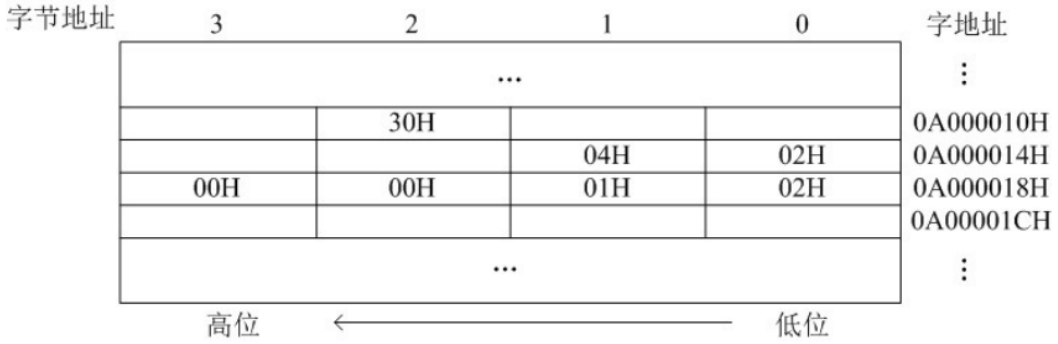
字符‘0’的ASCII码是：30H

短整型十进制数1026的十六进制值是：0402H

长整型十进制数 258 的十六进制值是：0000 0102H

所以，主存区域中存放的数据值及对应的地址编码如下：

(用十六进制表示)



18.就你对PC相对寻址的理解，解释为何汇编器在下面代码序列中直接实现分支指令时可能会有问题：

here: beq $s0, $s2, there

…

there: add $s0, $s0, $s0

说明汇编器可能如何重写该代码序列来解决这些问题

在PC相对寻址中EA=（PC）+ A，实际的跳转位置为

Here+There，在Here不为0的情况下，程序无法正确跳转到期望的There地址。

要实现正确跳转，需要汇编器计算Here到There之间的地址差，即代码为：

here: beq $s0, $s2, there-here

…

there: add $s0, $s0, $s0