计算机组成与体系结构

数据表示、计算以及指令系统

计算机科学与技术

第二部分主要内容

- 数据表示
- 数据计算
- 指令系统

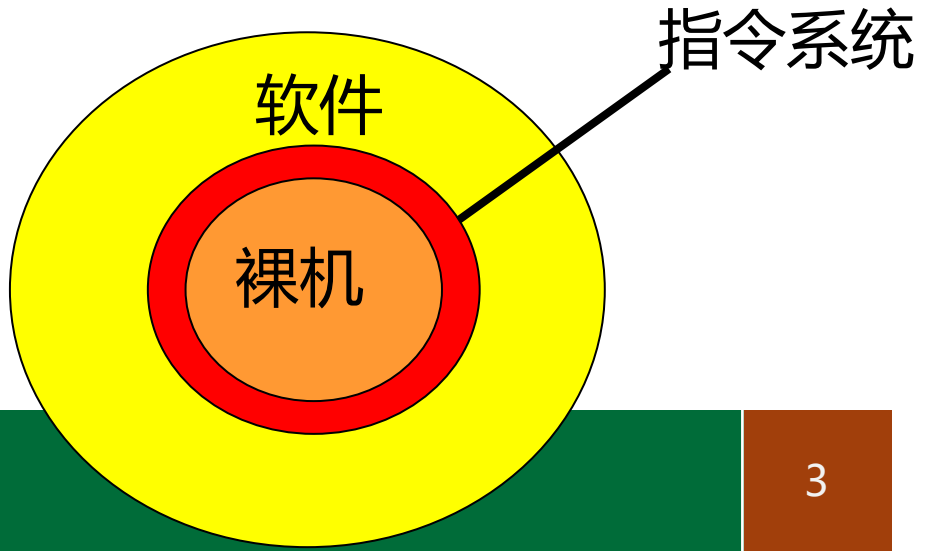
指令系统回顾

指令是指示计算机执行某些操作的命令，一台计算机的所有指令的集合构成该机的指令系统，也称指令集。指令系统处在软/硬件交界面，同时被硬件设计者和系统程序员看到。

硬件设计者角度：指令系统为CPU提供功能需求，要求易于硬件设计。
系统程序员角度：通过指令系统来使用硬件，要求易于编写编译器。
指令系统设计的好坏还决定了：计算机的性能和成本。

ISA：（Instruction Set Architecture）一组规范，一般说明寻址方式，指令编码，操作数类型，指令集合等内容。

ISA举例：RISC V，MIPS，X86（Intel IA-32，Intel 64，AMD64），ARM等



指令系统重点掌握

1. 寻址技术
2. 指令格式以及优化；
   ① 操作码字段编码：扩展操作码法；Huffman编码；
   ② 地址码字段：寻址技术、不同寻址方式有效地址（EA）的计算；
   ③ 设计与优化原则
3. 指令系统的发展：RISC和CISC。
4. 设计RISC的关键技术

设计新的指令系统

- 1）根据应用，拟出指令的分类和具体的指令。
- 2）编写出用该指令系统设计的各种高级语言的编译程序。
- 3）对各种算法编写大量测试程序，并且进行模拟测试，看指令系统的操作码和寻址方式是否都比较高。
- 4）将程序中出现的高频指令串复合，将其改成一条强功能新指令，用硬件实现；将低频指令改成用基本指令串完成的方式，即用软件实现。
- 重复以上过程，直到指令系统效能达到很高为止。

指令系统设计应考虑的问题

- 指令系统很大程度上决定了计算机具有的基本功能。设计和确定指令系统主要应考虑如何有利于满足系统的基本功能，有利于优化机器的性能价格比，有利于指令系统今后的发展和改进。指令系统的设计应能同时协调兼顾**编译程序设计者**和**系统结构设计者**两者要求。
- 指令设计常见问题：

1. 存储形式：如按边界对齐的存储方式；
2. 每条指令中显式说明的操作数个数；
3. 操作数保存的位置：寄存器、主存等；
4. 操作类型：运算类、传送类、控制类、输入输出类；
5. 操作数的类型和长短。

指令系统

1. 寻址技术
2. 指令系统设计与优化
3. 指令系统的发展和改进
4. 设计RISC的关键技术
5. 典型的RISC处理器（阅读内容）

寻址技术

- **寻址技术：** 寻找操作数及其他信息的地址的技术。
- **它是软件与硬件的一个主要分界面，是计算机系统结构的重要组成部分。**
- **研究内容：** 编址方式、寻址方式和定位方式。
- **研究对象：** 寄存器、主存储器、堆栈和输入输出设备。
- **前导课程中，** 已经学习了常见的寻址方式的基本原理和寻址技术的实现方法。

体系结构研究的重点： 分析各种寻址技术的优缺点，选择和确定寻址技术。

寻址技术-编址单位

- 1. 编址单位
  - ➢常用的编址单位：字、字节、位、块等。
- 字编址
  - ➢早期的大多数机器都采用这种编址方式。
    - □编址单位 = 访问单位
    - □每个编址单位所包含的二进制位数与读/写一次寄存器、主存的位数是相同。
  - ➢最简单。
  - ➢为进行字节或位操作，需要设置字节操作指令、位操作指令，在指令中指出操作数的字节编号或位编号。

寻址技术-编址单位

字节编址

- 是为了适应**非数值计算**的需要；
- 字节编址方式使编址单位与信息的基本单位（一个字节）相一致；
- 通常主存的访问单位是编址单位的若干倍（否则主存频带就太窄了）；
- **编址单位 < 访问单位。**

• 编址单位与访问单位（字长）

- 一般：字节编址，字访问；
- 部分机器：位编址，字访问；
- 辅助存储器：块编址。

字节编址字访问的问题

- (1)字节顺序问题
- 该数据内部的多个字节应该如何编址?
- 哪个字节顺序优先?
- (2) 对齐问题
- 向存储器访问多字节的数据，
- 其地址是否按照数据的自然边界对齐?

大端与小端模式

字节顺序：

在一个多字节数据内部的字节的排序方式。

两种排序方式：

- **高端字节顺序**（big-endian ordering）：地址低的字节顺序优先，又称为大端模式；即低字节在高地址，高字节在低地址。采用大端模式的计算机有IBM360/370、Motorola 68000等。
- **低端字节顺序**（little-endian ordering）：地址高的字节顺序优先，又称为小端模式，即低字节在低地址，高字节在高地址。采用小端模式的计算机有Intel 80X86、DEC VAX等。

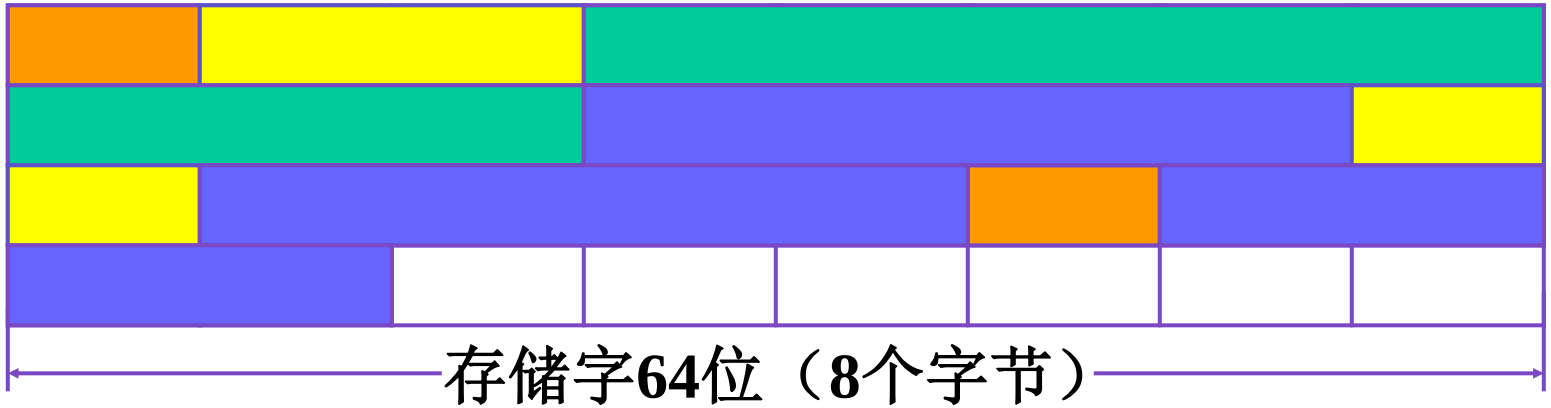
对齐问题

在采用字节编址的情况下，数据在主存储器中的三种不同存放方法。假设，存储字为64位（8个字节），读/写的数据有四种不同长度，它们分别是字节（8位）、半字（16位）、单字（32位）和双字（64位）。请注意：此例中数据字长（32位）不等于存储字长（64位）。

现有一批数据，它们依次为：字节、半字、双字、单字、半字、单字、字节、单字。

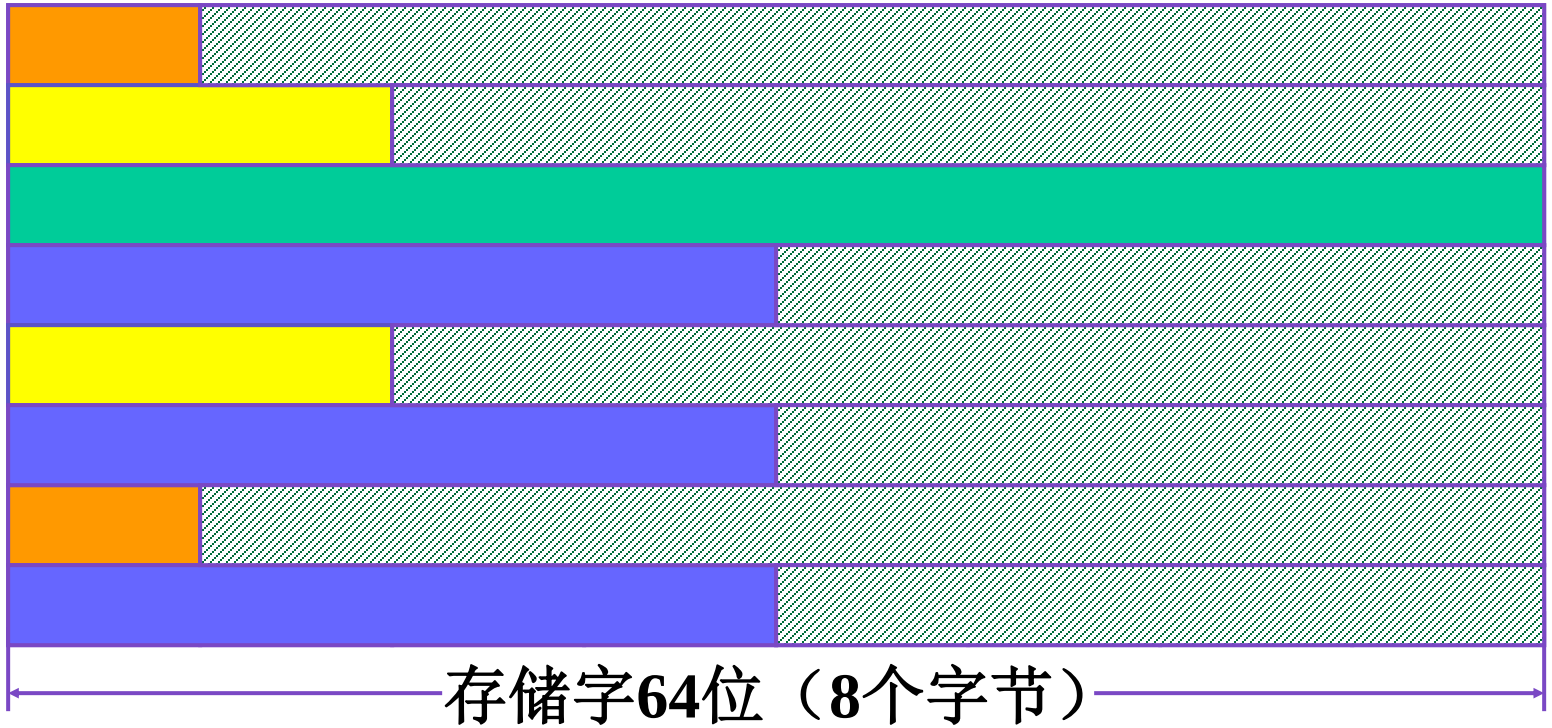
(1)不浪费存储器资源的存放方法

四种不同长度的数据一个紧接着一个存放。优点是不浪费宝贵的主存资源，但存在的问题是：当访问的一个双字、单字或半字跨越两个存储字时，存储器的工作速度降低了一倍，而且读写控制比较复杂。



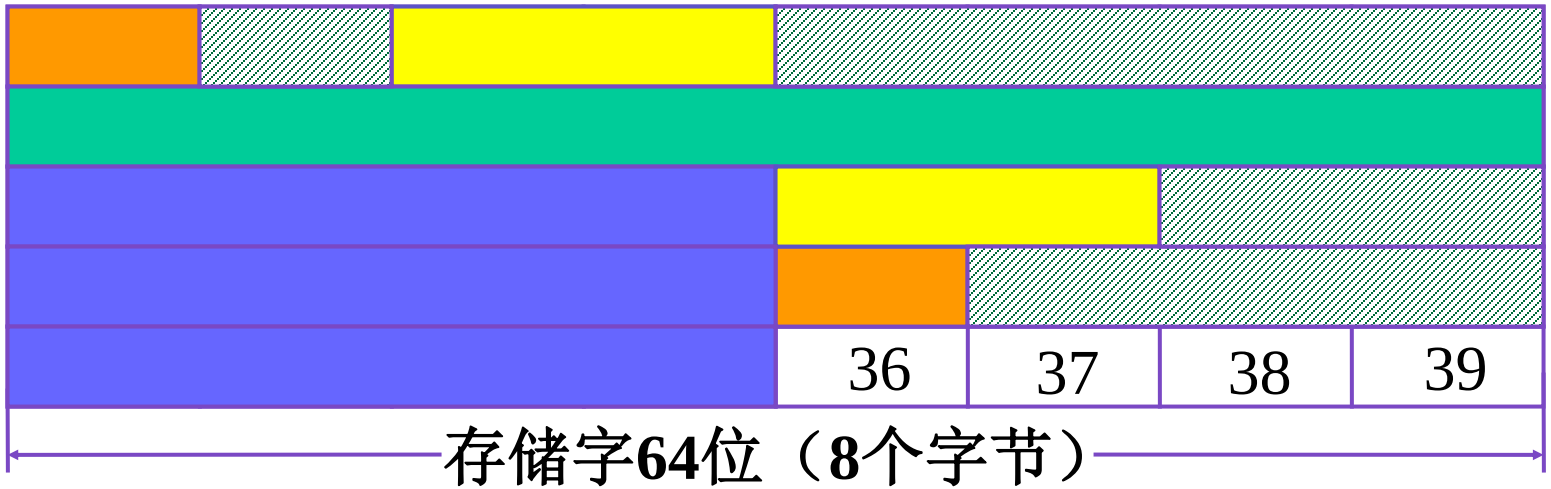
(2)从存储字的起始位置开始存放方法

无论要存放的是字节、半字、单字或双字，都必须从存储字的起始位置开始存放，而空余部分浪费不用。优点是：无论访问一个字节、半字、单字或双字都可以在一个存取周期内完成，读写数据的控制比较简单。缺点是：浪费了宝贵的存储器资源。


存储字64位（8个字节）

(3)边界对齐的数据存放方法

双字地址的最末三个二进制位必须为000，单字地址的最末两位必须为00，半字地址的最末一位必须为0。它能够保证无论访问双字、单字、半字或字节，都在一个存取周期内完成，尽管存储器资源仍然有浪费，但是浪费比第(2)种存放方法要少得多。

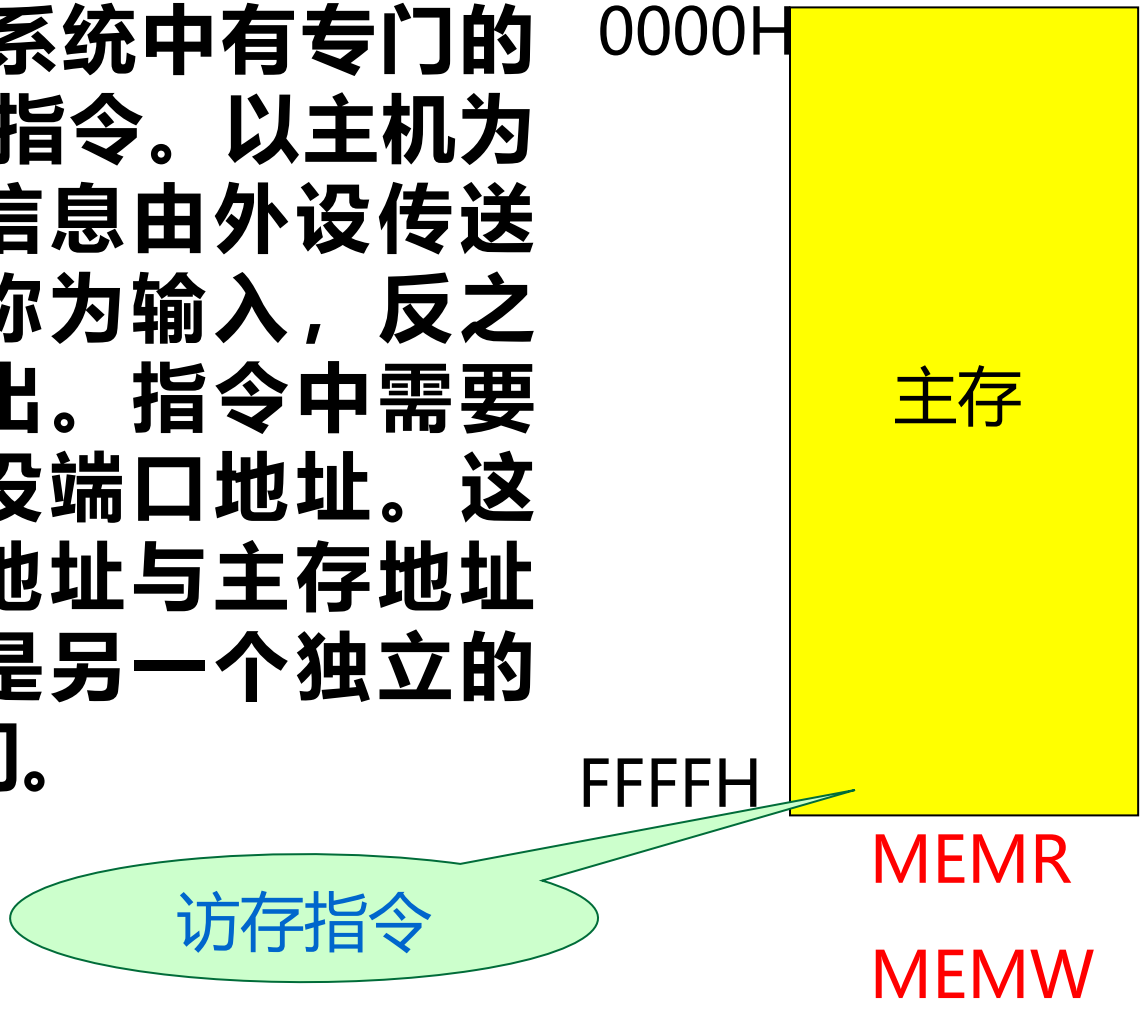


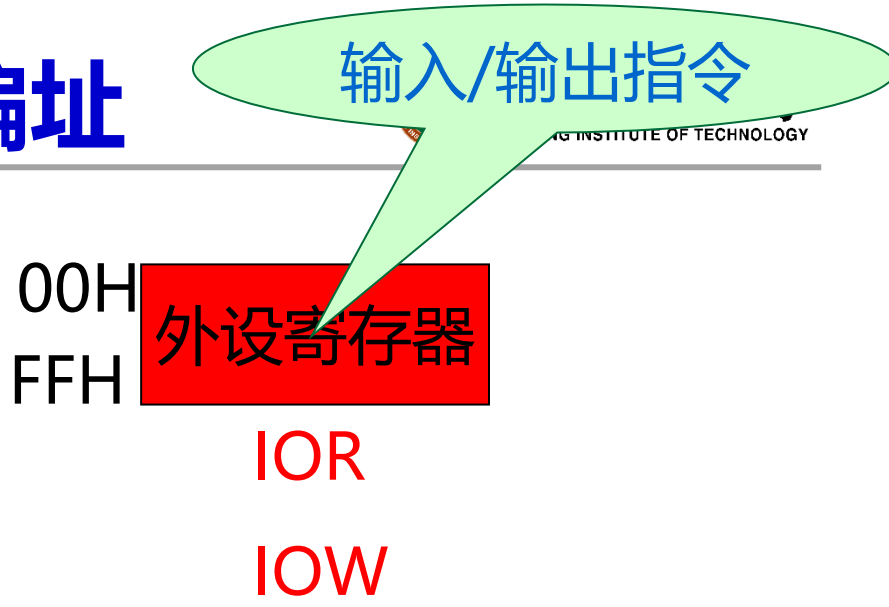
编址方式-分类编址

- 分类编址
  - 又叫独立编址，将部件适当分类，每类各自从“0”开始单独编址。
  - 通用寄存器、主存储器和输入输出设备均独立编址，形成三个零地址空间。
  - 通用寄存器独立编址，主存储器与输入输出设备统一编址，形成两个零地址空间。

主存储器和输入输出设备独立编址

指令系统中有专门的IN/OUT指令。以主机为基准，信息由外设传送到主机称为输入，反之称为输出。指令中需要给出外设端口地址。这些端口地址与主存地址无关，是另一个独立的地址空间。





优点：

指令字长较短

地址形成简单

主存的编址空间较大

缺点：

指令中应有区分每类部件的标志或约定

编址方式——统一编址

- 把各种部件统一编成一个从“0”开始的一维线性地址空间（一个零地址空间）。
- 对不同部件的访问反映在对这个空间不同地址范围的访问。
- 优点：
  - 可简化指令系统。
  - 例如若将I/O设备与主存统一编址，就不必设置单独的I/O指令。
- 缺点：
  - 在一定程度上使地址形成复杂化。

主存储器和输入输出设备统一编址

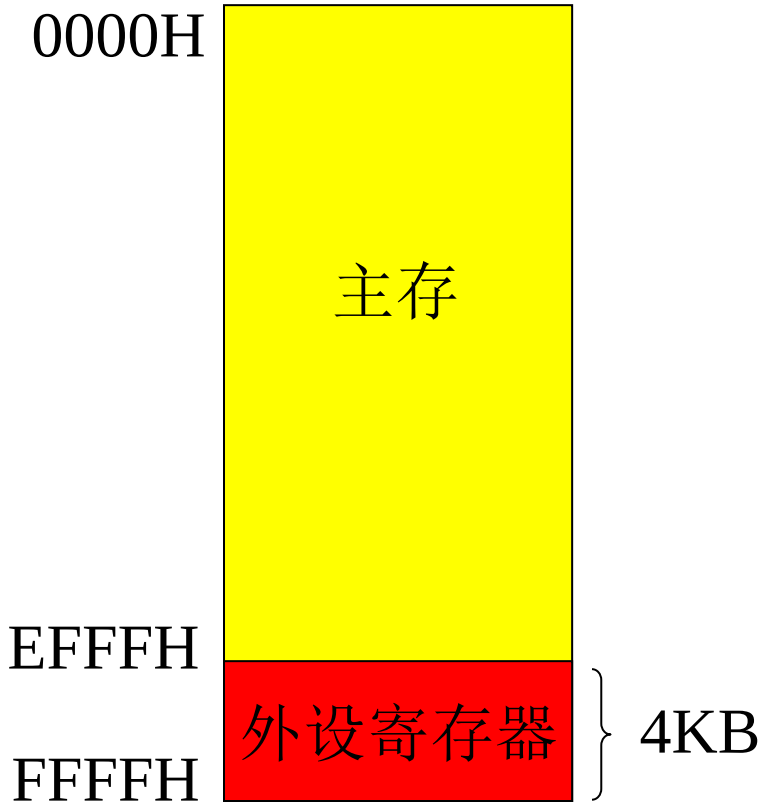
指令系统中没有专门的I/O指令，就用一般的数据传送类指令来实现I/O操作。

数据传送指令

MOV R_0,(50H)

MOV R_0,(F000H)

输入指令



编址方式——隐含编址

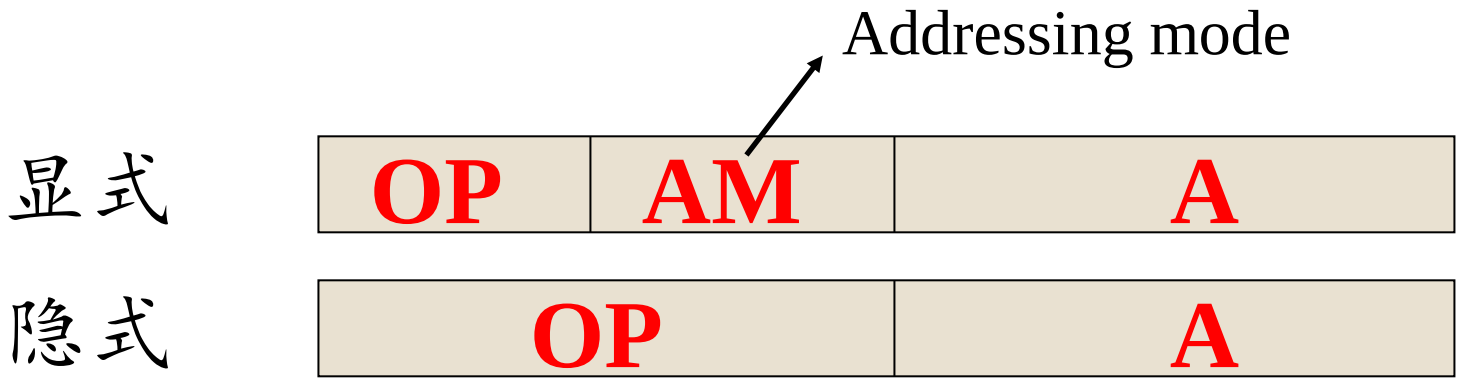
- 采用事先约定的编址方式隐含寻址。
- 例如堆栈、某些专用寄存器、Cache等可以采用这种编址方式。
- **无零地址空间。**
- **优点：**
  - 不必进行地址计算，访问速度比较快。
- **缺点：**
  - 有时会使指令的设计不规范。

寻址方式

- 大多数计算机都将主存、通用寄存器、堆栈分类编址，因此就有分别面向寄存器、堆栈和主存的寻址方式。面向寄存器的寻址方式操作数可以取自寄存器或主存，结果大多保存在寄存器中，少量的送入主存。面向堆栈的寻址方式主要访问堆栈，少量访问主存或寄存器。面向主存的寻址方式主要访问主存，少量访问寄存器。
- **寻址方式：**指令寻找（或访问）到所需数据的方法。
- **主要内容：**寻址方式的设计思想和设计方法。
- 寻址的多样性、灵活性、寻址空间范围大小、地址变换速度等都有了很大的发展。

寻址方式

- 寻址方式的指明
  - 口显式：在指令中设置专门的寻址方式字段，用二进制代码来表明寻址方式类型。
  - 口隐式：由指令的操作码字段说明指令格式并隐含约定寻址方式。



寻址方式

- **立即数寻址**

OP	立即数

  - 优点：不需要数据存储单元，指令执行速度快。
  - 缺点：只能用于源操作数寻址，数据比较短。

- **寄存器寻址**

$$EA = R_i$$

  - □OPC　R　　一地址
  - □OPC　R，R　　二地址
  - □OPC　R，R，R　三地址
  - □OPC　R，M　　用于Load和Store或一般运算指令
  - 优点：指令字长短，执行速度快，支持向量、矩阵运算。
  - 缺点：不利于优化编译，现场切换困难，硬件复杂。

- **主存储器寻址**
  - □OPC　M
  - □OPC　M，M
  - □OPC　M，M，M
  - ➢包括直接、间接、变址寻址等。
- **堆栈寻址**
  - □OPC
  - □OPC　　M
  - ➢主要优点：支持高级语言，有利于编译程序；节省存储空间；支持程序的嵌套和递归调用，支持中断处理。
  - ➢主要缺点：运算速度比较低。

复习回顾：

直接寻址：EA=A

间接寻址：EA=（A）　单级间接与多级间接

寄存器间接寻址：EA=（Ri）

变址寻址：$EA=A+(R_X)$　$A\geq 0$

相对寻址：EA=D+(PC)　D可正可负

基址寻址：$EA=D+(R_b)$　D可正可负

变址寻址与基址寻址方式的比较

基址寻址和变址寻址在形成有效地址时所用的算法是相同的，而且在一些计算机中，这两种寻址方式都是由同样的硬件来实现的。

但这两种寻址方式应用的场合不同，变址寻址是**面向用户的**，用于访问字符串、向量和数组等成批数据；而基址寻址**面向系统**，主要用于逻辑地址和物理地址的变换，用以解决程序在主存中的再定位和扩大寻址空间等问题。在某些大型机中，基址寄存器只能由特权指令来管理，用户指令无权操作和修改。

间接寻址方式与变址寻址方式的比较

- **目的相同：**
  - ➢都是为了解决操作数地址的修改问题。
  - ➢都能做到不改变程序而修改操作数地址。
  - ➢原则上，一种处理机中只需设置间接址寻址方式与变址寻址方式中的任何一种即可，有些处理机两种寻址方式都设置。
  - ➢**如何选取间址寻址方式与变址寻址方式？优缺点怎样？**

例：一个由N个元素组成的数组，已经存放在起始地址为AS的主存连续单元中，现要把它搬到起始地址为AD的主存连续单元中。不必考虑可能出现的存储单元的重叠问题。为了编程简单，采用一般的两地址指令编写程序。

间接寻址方式与变址寻址方式的比较

- 采用变址寻址方式编写的程序简单、易读。
- 对于程序员，两种寻址方式的主要差别是：
  - 间址寻址方式：间接地址在主存储器中，没有偏移量；
  - 变址寻址方式：基地址在变址寄存器中，有偏移量；
- 主要优缺点比较：
  - 实现的难易程度：间址寻址方式容易
  - 指令的执行速度：间址寻址方式慢
  - 对数组运算的支持：变址寻址方式比较好

需要注意的问题：第一点见教科书。

间接寻址方式与变址寻址方式的比较

- 自动变址
  - 在访问间接地址过程中，地址自动增/减。
  - 原来程序中紧跟其后的对变址寄存器做增/减量的指令可以省去。
  - 地址增/减量单位根据具体机器所采用的编址方式和数据元素的长度等关系来确定。
  - 地址增/减量的先后次序要注意。
- 前变址与后变址
  - 在既有变址寻址，又有间址寻址方式的计算机系统中，先变址还是先间址需要实现定义
  - 前变址寻址方式：EA = ((X) + A)
  - 后变址寻址方式：EA = (X) + (A)

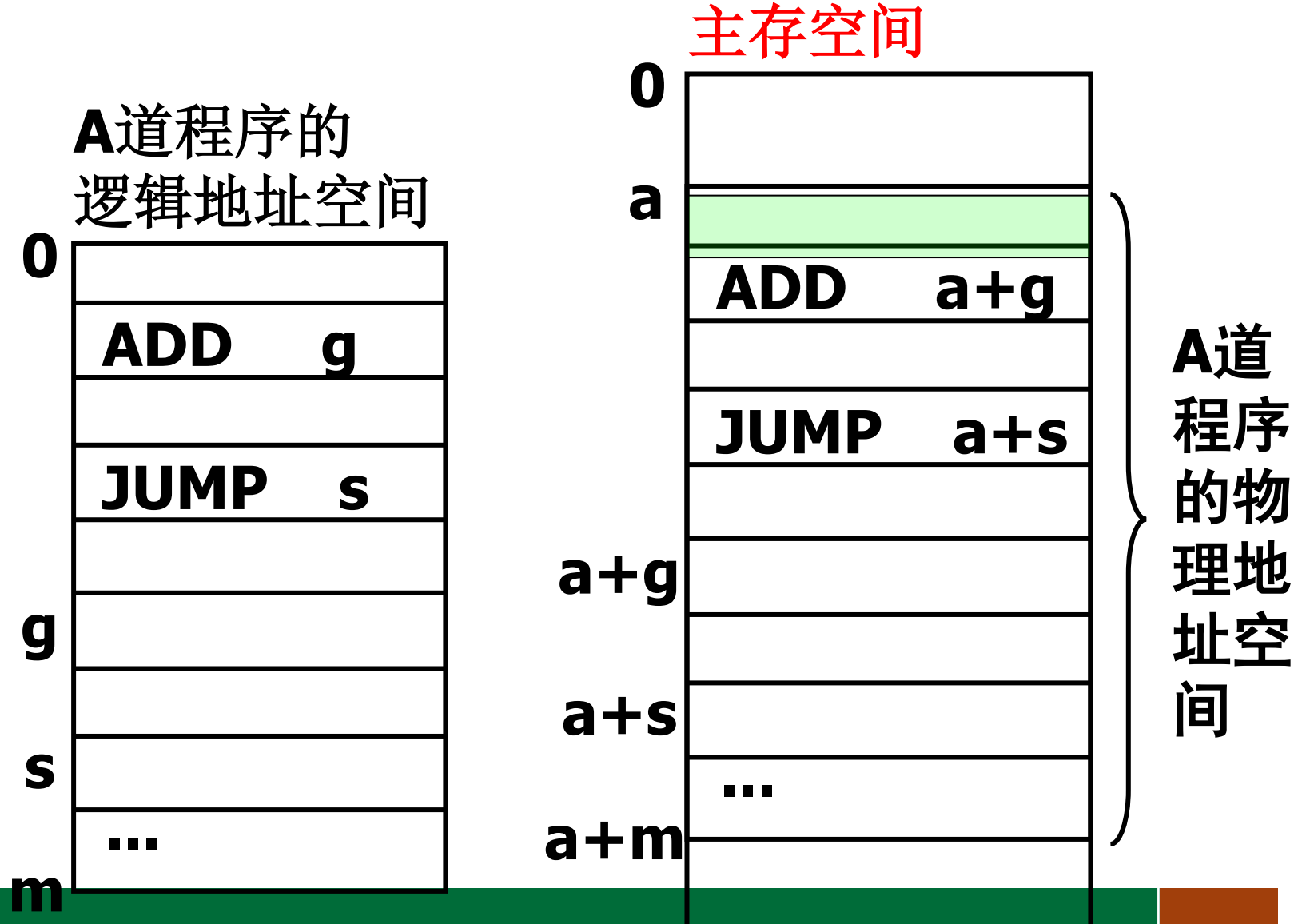
程序在主存中的定位技术

- 何时确定程序在主存中的物理地址?
- 以何种方法确定程序在主存中的物理地址?
- 根据程序的独立性要求和模块化设计思想，为了解决程序空间与主存空间大小不同的矛盾、以及多道程序共享主存空间等问题，引入了新的“地址”概念。
- **逻辑地址**
  - 程序员编写程序时所使用的地址。
- **主存物理地址**
  - 程序在主存中的实际地址。

程序在主存中的定位技术

早期，主存物理地址与逻辑地址是一致的。

现在，主存物理地址是从0开始编址的一维线性空间，所以**逻辑地址空间和物理地址空间是不一致的**，故需要进行逻辑地址空间到物理地址空间的转换。



程序在主存中的定位技术

- 主要有三种变换（定位）技术
- **直接定位方式**：在程序装入主存储器之前，程序中的指令和数据的主存物理地址就已经确定了的称为直接定位方式。
- **静态再定位**：在程序装入主存储器的过程中随即进行地址变换，确定指令和数据的主存物理地址的称为静态定位方式。
- **动态再定位**：在程序执行过程中，当访问到相应的指令或数据时才进行地址变换，确定指令和数据的主存物理地址的称为动态定位方式。

程序在主存中的定位技术

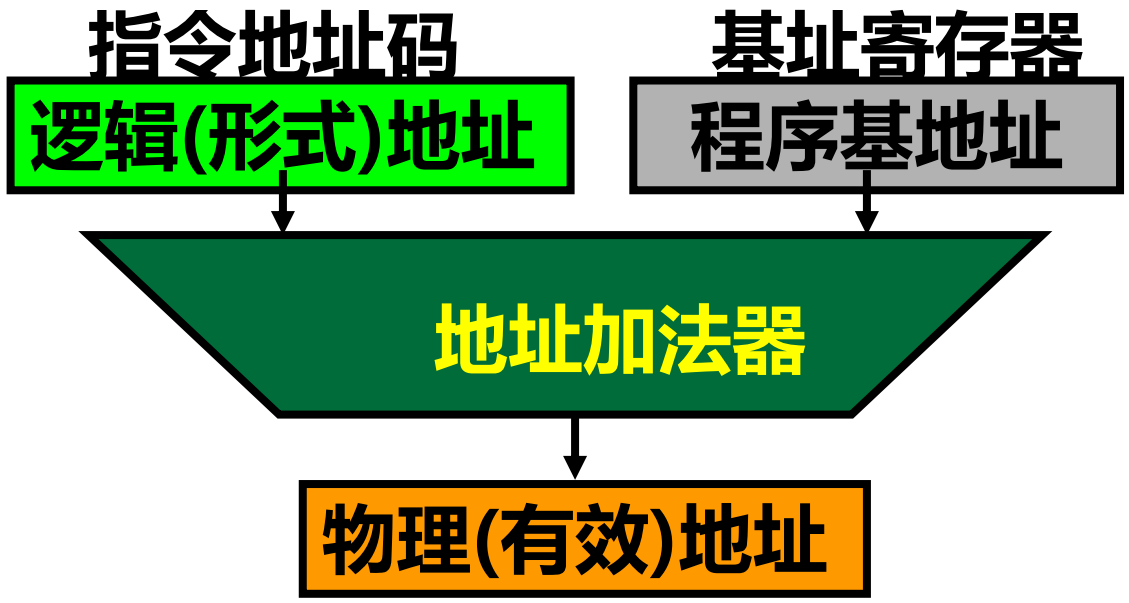


图　基址寻址工作原理

程序执行时，通过地址加法器将逻辑地址加上基址寄存器的内容(程序基地址)，形成物理地址，然后访存，地址变换速度快。

1. 寻址技术
2. 指令系统设计与优化
3. 指令系统的发展和改进
4. 设计RISC的关键技术
5. 典型的RISC处理器（阅读内容）

指令系统设计

- **指令的功能设计和指令的格式设计。**

- 确定哪些基本功能使用硬件实现，考虑速度、成本、灵活性等因素；
- 对指令系统的五个基本要求:完整性、规整性、正交性、高效率性和兼容性

机器指令基本格式

一条指令就是机器语言的一个语句，它是一组有意义的二进制代码。指令的基本格式如下：

操作码：指明操作的性质及功能。

地址码：指明操作数的地址，特殊情况下也可能直接给出操作数本身。

指令长度可以等于机器字长，也可以大于或小于机器字长。

在一个指令系统中，若所有指令的长度都是相等的，称为定长指令字结构；若各种指令的长度随指令功能而异，称为变长指令字结构。

指令格式的设计

- **主要目标：**
  - 节省程序的存储空间
  - 指令格式尽量规整，便于译码
- **研究内容：**
  - 操作码的表示以及优化；
  - 地址码的表示以及优化；

指令的操作码

指令系统中的每一条指令都有一个唯一确定的操作码，指令不同，其操作码的编码也不同。为了能表示整个指令系统中的全部指令，指令的操作码字段应当具有足够的位数。

指令操作码的编码可以分为**规整型**和**非规整型**两类：

规整型（定长编码）

非规整型（变长编码）

规整型

操作码字段的位数和位置是固定的。

假定：指令系统共有m条指令，指令中操作码字段的位数为N位，则有如下关系式：

$$N \geq \log_2 m$$

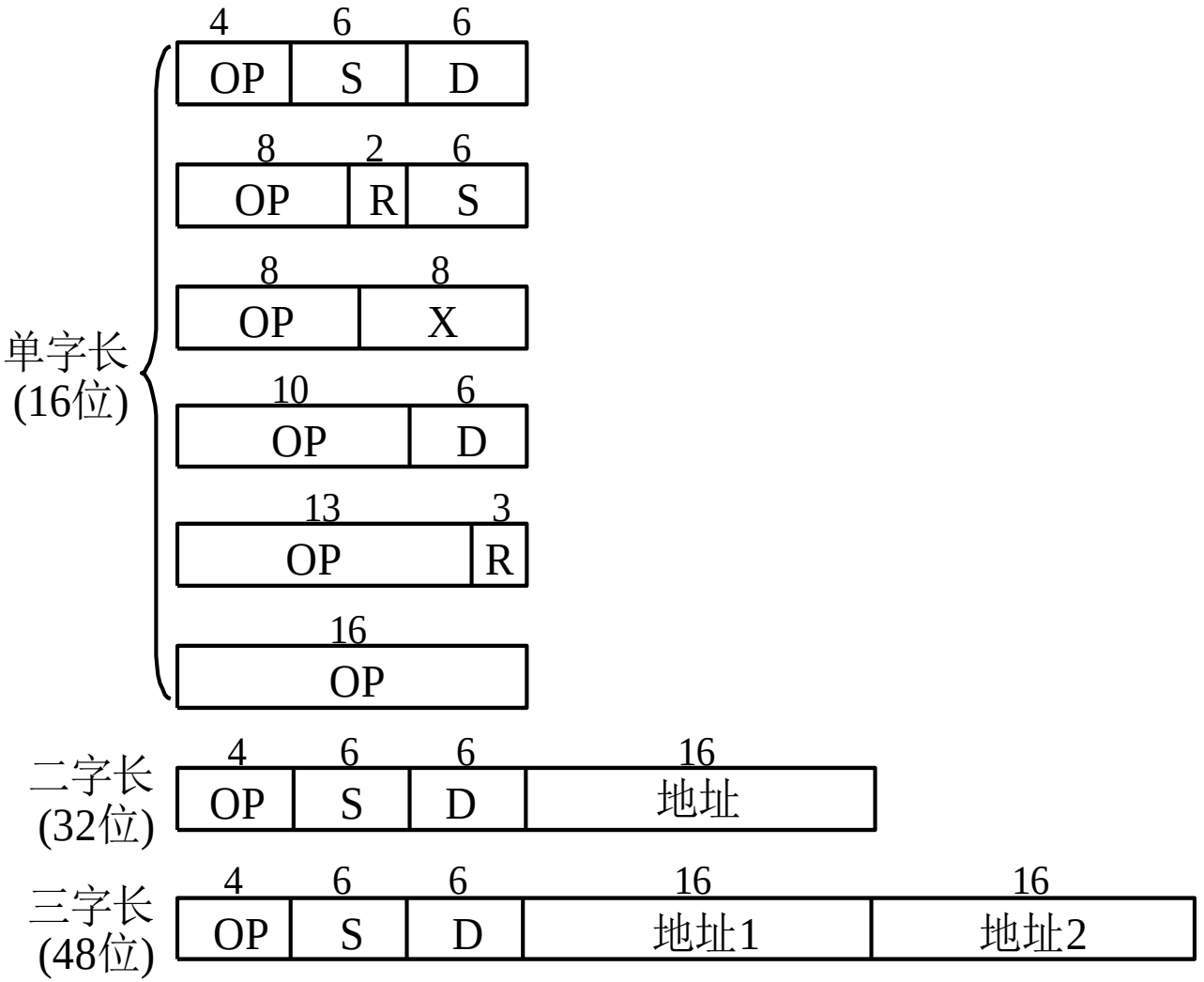
IBM 370机（字长32位）的指令可分为三种不同的长度形式：半字长指令、单字长指令和一个半字长指令。不论指令的长度为多少位，其中操作码字段**一律都是8位**，8位操作码字段允许容纳256条指令，实际上在IBM 370机中仅有183条指令。

非规整型

操作码字段的**位数不固定**，且分散地放在指令字的不同位置上。

PDP-11机（字长16位）的指令分为单字长、两字长、三字长三种，操作码字段占4～16位不等，可遍及整个指令长度。

操作码字段的位数和位置不固定将增加指令译码和分析的难度，使控制器的设计复杂化。

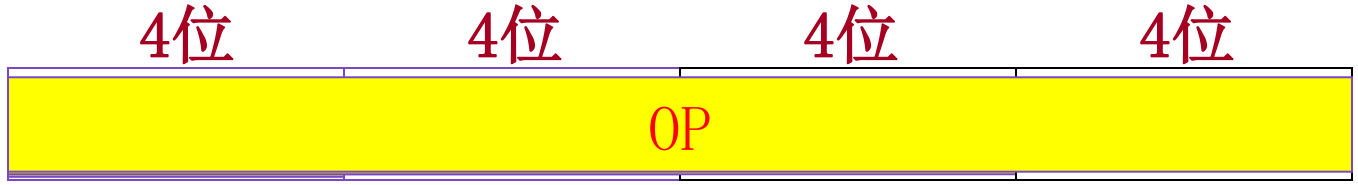


- 最常用的非规整型编码方式是**扩展操作码法。**
- **主要思路**：让操作数地址个数多的指令（如三地址指令）的操作码字段短些，操作数地址个数少的指令（如一或零地址指令）的操作码字段长些。
- **例如**：设某机的指令长度为16位，操作码字段为4位，有三个4位的地址码字段，其格式为：

4位	4位	4位	4位
OP	A_1	A_2	A_3

- 若要表示15条三地址指令，15条二地址指令，15条一地址指令和16条零地址指令该如何编码?

解答：如果按照定长编码的方法，4位操作码字段最多只能表示16条不同的三地址指令。故选择扩展操作码法。



要点：
1、短码不能是长码的前缀。
2、各条指令的操作码唯一。

扩展窗口

扩展窗口

扩展窗口

0000 XXXX YYYY ZZZZ
⋮
1110 XXXX YYYY ZZZZ — 15条三地址指令

1111 0000 XXXX YYYY
⋮
1111 1110 XXXX YYYY — 15条二地址指令

1111 1111 0000 XXXX
⋮
1111 1111 1110 XXXX — 15条一地址指令

1111 1111 1111 0000
⋮
1111 1111 1111 1111 — 16条零地址指令

指令操作码的优化

指令操作码的优化就是要在足够表达全部指令的前提下，使操作码字段占用的位数最少。改进操作码编码方式能够节省程序存储空间。

- Huffman编码
  - Huffman压缩，1952年由Huffman首先提出。
  - 对发生概率最高的事件用最短的位数（时间）来表示（处理），对发生概率较低的，允许用较长的位数（时间）来表示（处理）。
  - 前提条件：必须事先知道事件发生的概率。

指令操作码的优化举例

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

现有一模型机，共有7条指令，使用频度如下。

指令	使用频度（p_i）
I_1	0.40
I_2	0.30
I_3	0.15
I_4	0.05
I_5	0.04
I_6	0.03
I_7	0.03

操作码的**最短平均长度（理想情况）**，又称信息源熵，可通过下式计算：

$$H = -\sum_{i=1}^{n} p_i \cdot \log_2 p_i$$

其中：p_i表示第i种操作码在程序中出现的概率。

按表 2.4 的数据，得

H=0.40×1.32+0.30×1.74+0.15×2.74+0.05×4.32+0.04×4.64+0.03×5.06+0.03×5.06 =2.17(理想值)

Huffman编码

- **信息冗余量：**

$$R = 1 - \frac{H}{\text{实际平均码长}}$$

- **采用 3 位定长操作码表示的信息冗余量**

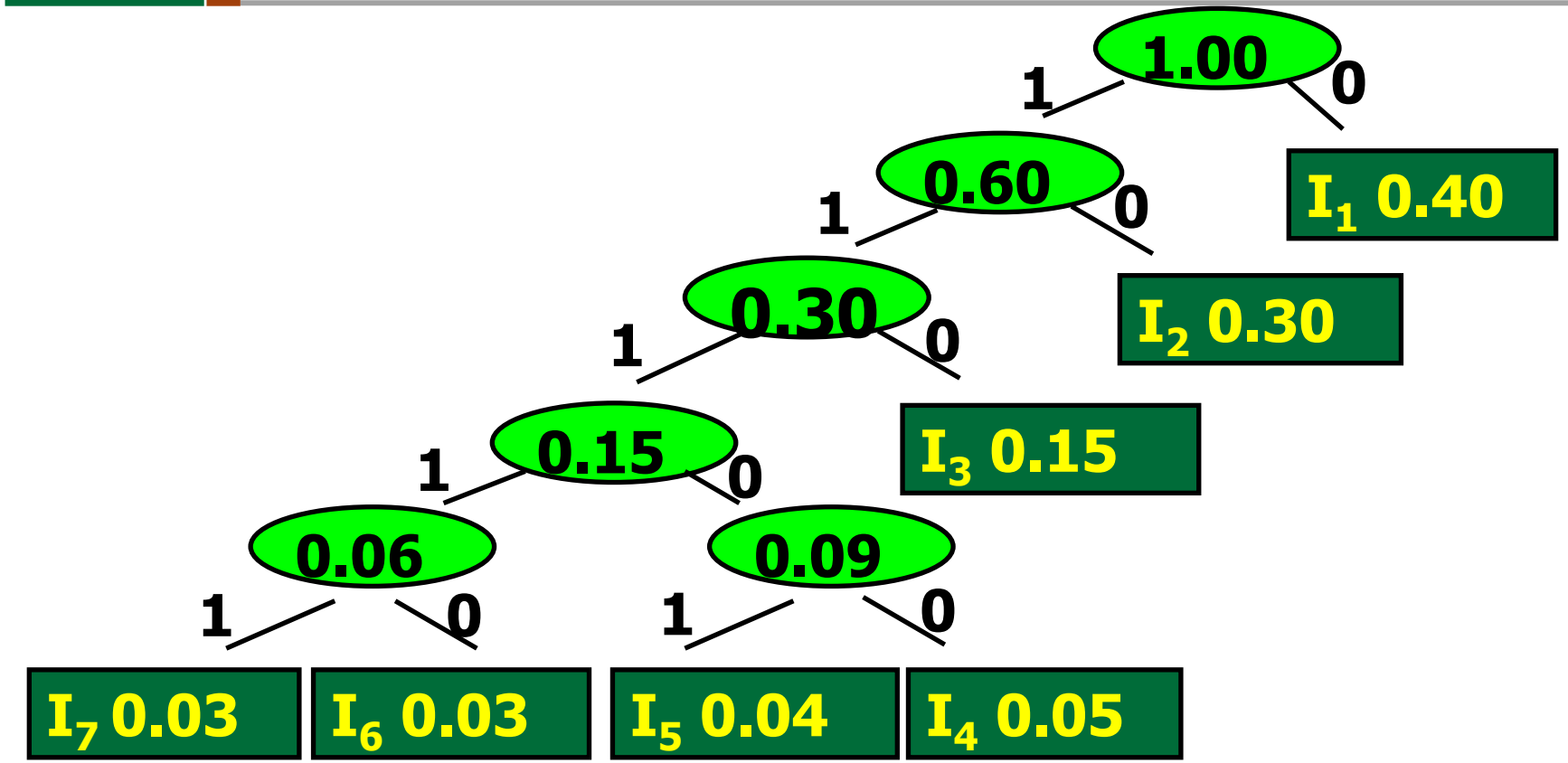
$$\frac{\text{操作码的实际平均长度} - H}{\text{操作码的实际平均长度}} = \frac{3 - 2.17}{3} = 0.28 \quad (\text{即}28\%)$$

- **为减少信息冗余， 可改用哈夫曼编码。**
- **Huffman编码首先要构造Huffman树，这种编码方法，又称为最小概率合并法。**

Huffman编码法编码流程

- ① 把所有指令按照操作码在程序中出现的概率，自左向右从小到大排列好。
- ② 选取两个概率最小的节点合并成一个概率值是两者之和的新节点，并把这个新节点与其它还没有合并的节点一起形成新节点集合。
- ③ 在新节点集合中选取两个概率最小的节点进行合并，如此继续进行下去，直至全部节点合并完毕。
- ④ 最后得到的根节点的概率值为1。
- ⑤ 每个节点都有两个分支，分别用一位代码0和1表示。
- ⑥ 从根节点开始，沿箭头所指方向，到达属于该指令的概率节点，把沿线所经过的代码组合起来得到这条指令的操作码编码。

Huffman树



短码不能是长码前缀，
保证唯一和实时性。

Huffman编码

- 只要采用全哈夫曼编码，操作码的平均码长肯定是唯一的。如此例，操作码的平均码长

$$\sum_{i=1}^{7} p_i \cdot l_i = 0.40\times1+0.30\times2+0.15\times3+0.05\times5$$

$$+0.04\times5+0.03\times5+0.03\times5$$

$$= 2.20 \quad (\text{位})$$

- 非常接近于可能的最短位数(H)2.17位。这种编码的信息冗余为

$$\frac{2.20-2.17}{2.20}\approx1.36\%$$

Huffman编码

- 每条指令的长度

指令序号	概率	Huffman编码法	操作码长度
I_1	0.40	0	1位
I_2	0.30	10	2位
I_3	0.15	110	3位
I_4	0.05	11100	5位
I_5	0.04	11101	5位
I_6	0.03	11110	5位
I_7	0.03	11111	5位

Huffman编码

Huffman编码的具体码值不惟一。

主要优点：实际平均码长最短。

主要缺点：

- **① 操作码长度很不规整，硬件译码困难。**
- **② 与地址码共同组成固定长的指令比较困难。**

扩展编码

- 扩展操作码编码
  - 一种介于定长编码和全哈夫曼编码之间的一种编码方法。
  - 操作码长度不固定，但只有有限的几种长度。
  - 使用频度高的指令用短操作码表示，使用频度低的指令用长操作码表示(哈夫曼压缩思想)。
  - 等长扩展编码和不等长扩展编码两种方式，前者更容易译码。

  - 例如：将上例改为2-4等长扩展编码（2位和4位码长）。
    - 使用频度高的指令I_1、I_2、I_3用2位编码表示。
    - 使用频度低的指令I_4、I_5、I_6、I_7用4位编码表示。

扩展编码

2-4等长扩展编码：

指令序号	概率	Huffman编码法	操作码长度
I_1	0.40	00	2位
I_2	0.30	01	2位
I_3	0.15	10	2位
I_4	0.05	1100	4位
I_5	0.04	1101	4位
I_6	0.03	1110	4位
I_7	0.03	1111	4位

扩展编码

- **实际平均码长：**

$$L=\sum p_i l_i = 2.30 \text{（位）}$$

- **信息源熵：**

$$H=-\sum p_i log_2 p_i = 2.17 \text{（位）}$$

- **信息冗余量：**

$$R=1-\frac{H}{\text{实际平均码长}}=1-\frac{2.17}{2.30}\approx 5.65\%$$

指令地址码字段优化

- 只对操作码进行优化，而没有对**地址码和寻址方式**采取相应的措施，程序总位数还是难以减少。

➢**地址码优化：**

□**一方面，**希望寻址范围越大越好。例如采用虚拟存储器可以使用比物理地址空间更大的逻辑地址空间。

□**另一方面，**通过采取各种方法，在满足很大寻址范围的前提下，地址码的长度不一定非要那么宽。

指令地址码字段优化

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

问题：

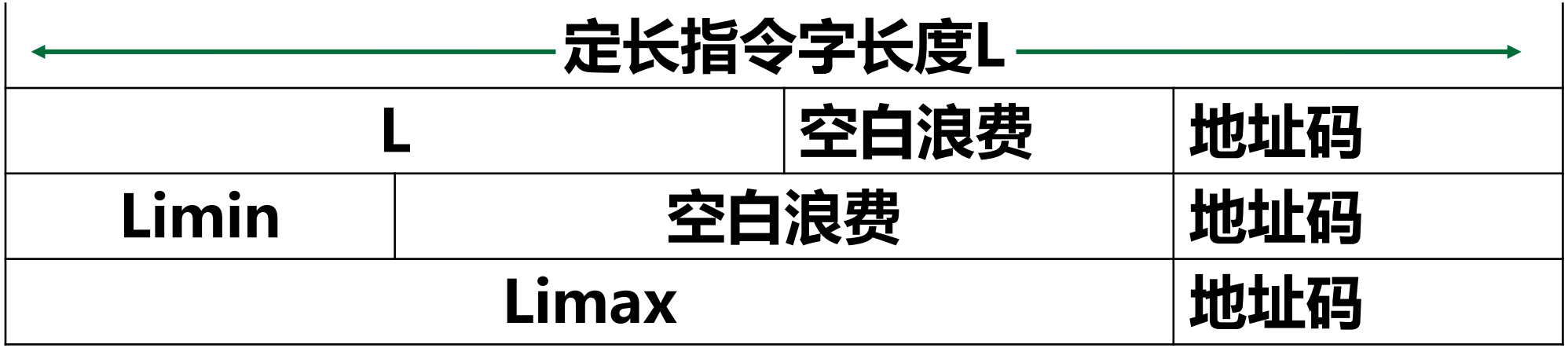


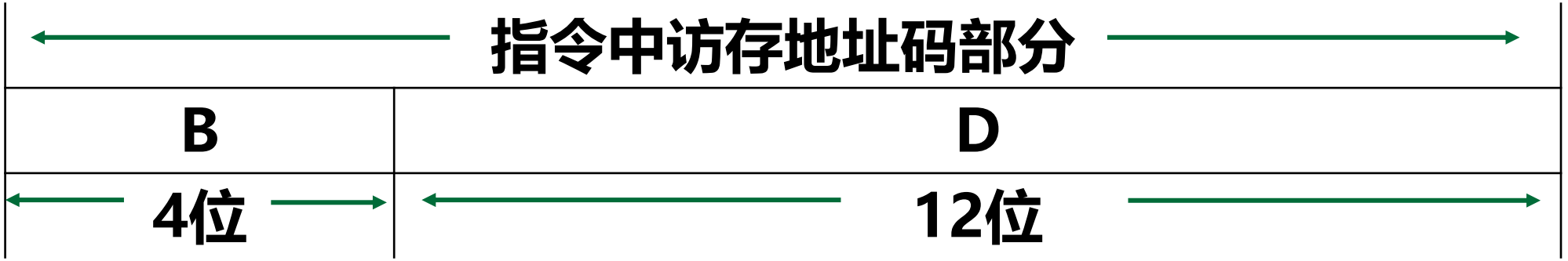
图 2.10　等长地址码发挥不出操作码优化表示的作用

优化思路：地址码可变。

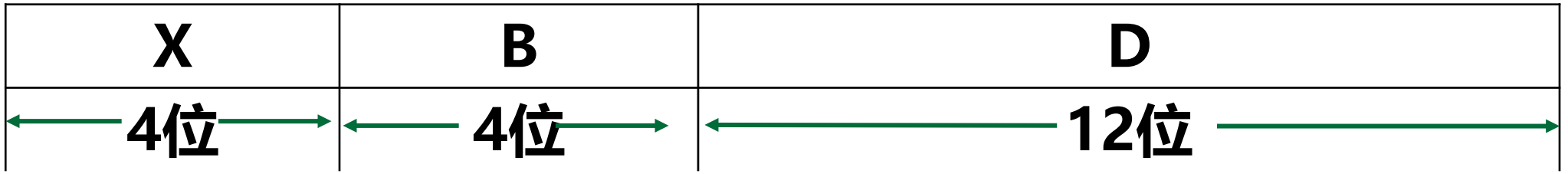
不同寻址方式对地址码位数的影响

如IBM 370 的指令中**基址寻址**，16位地址→24位物理地址。

$EA=(B)_{8\sim31}+D$



如IBM 370 指令中**基址变址寻址**，20位地址→24位物理地址。



$EA=(X)+(B)+D$

不同寻址方式对地址码位数的影响

➢**转移指令**：将访存地址空间分为若干个段，访存地址就由段号和段内地址两部分组成。

- 口段内转移：只需段内地址
- 口段间转移：只需转到段基址

段 号	段内地址

➢**寄存器寻址与寄存器间接寻址**：地址码为寄存器号。

指令地址码字段优化

➢对于操作数不同的定长指令采用多种地址制：

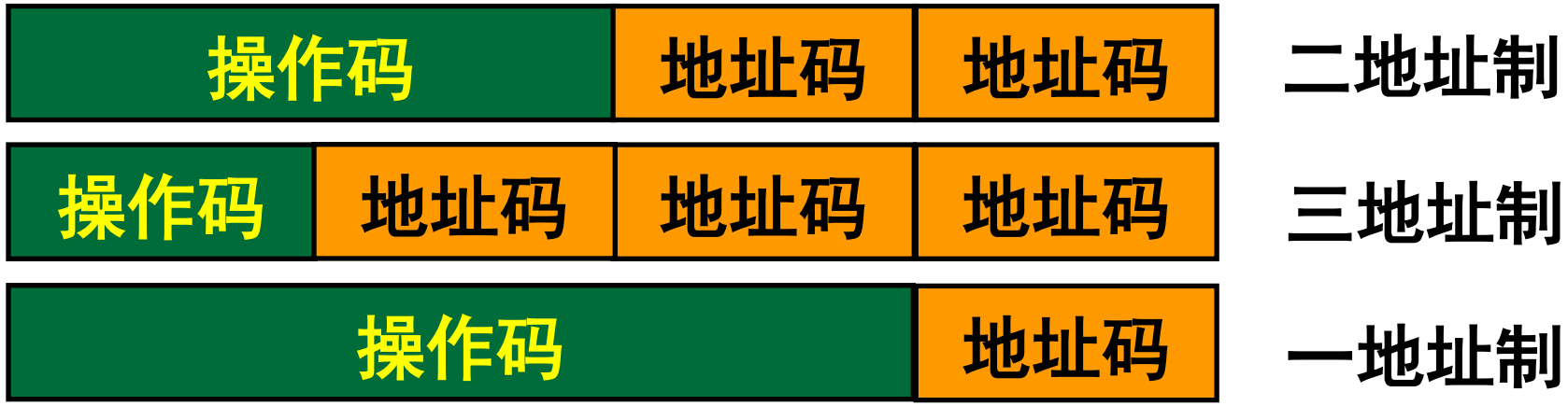


图　　定长指令字中采用多种地址制

指令地址码字段优化

- 即使为同一种地址制，还可以采用多种地址形式和长度，可以利用空白处存放直接操作数或常数等

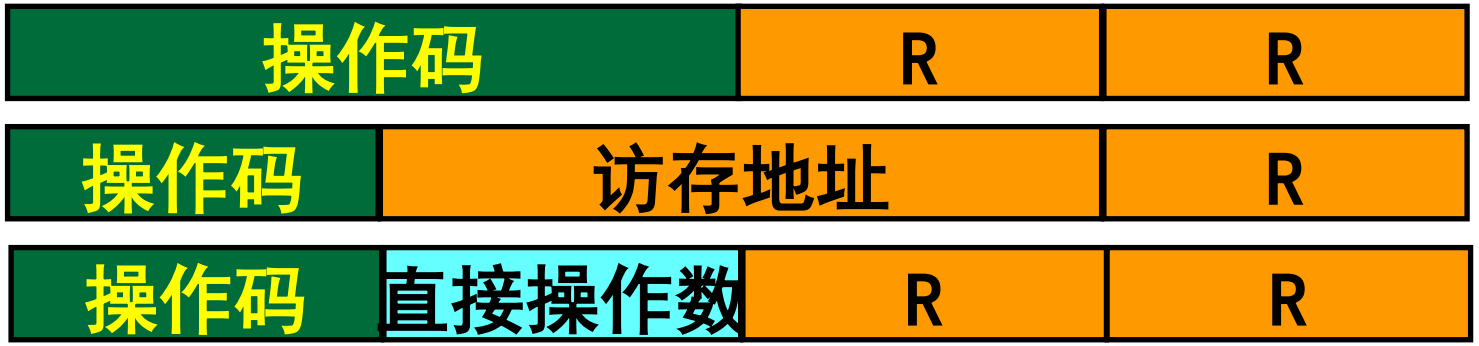


图 多种地址形式和长度

- 如果让最常用的操作码最短，其地址码字段个数越多，就越能使指令的功能增强，越可以从宏观上减少所需的指令条数。

➢例如，为实现A+B→C，采用单地址指令需经取A、加B、 送C 3 条指令完成，而采用 3 地址指令

加	A	B	C

则只需一条指令即可完成。这不仅进一步缩短了程序的占用空间，也会因为减少了访存取指令次数而加快程序执行的速度。

➢**多种指令字长**有利于减少信息冗余，缩短程序的长度。

指令字格式优化的措施总结

- (1)选择合适的编码方式**缩短操作码的平均码长**，如采用扩展操作码，根据指令的频度的分布状况；
- (2)采用诸如基址、变址、相对、寄存器、寄存器间接、段式存放、隐式指明等多种寻址方式，以**缩短地址码的长度**，并在有限的地址长度内提供更多的地址信息；
- (3) 采用0、1、2、3等多种地址制，以增强指令的功能，这样，从宏观上就越能**缩短程序的长度**，并加快程序的执行速度；
- (4)在同种地址制内再采用**多种地址形式**，如寄存器-寄存器、寄存器-主存、主存-主存等，让每种地址字段可以有多种长度，且让长操作码与短地址码进行组配；
- (5) 在维持指令字在存储器中按整数边界存储的前提下，使用**多种不同的指令长度。**

1. 寻址技术
2. 指令系统设计与优化
3. **指令系统的发展和改进**
4. 设计RISC的关键技术
5. 典型的RISC处理器（阅读内容）

指令系统的发展和改进

复杂指令系统计算机 CISC (Complex Instruction Set Computer)

早期计算机部件昂贵、速度慢，为了扩展硬件功能，不得不将更多更复杂指令加入到指令系统，以提高计算机的处理能力，如早期x86架构的指令集。

主要特点

①指令数量多；

②指令长度可以不固定，指令格式和寻址方式多样；

③很多指令会涉及存储器读写操作，指令周期长；

④多采用微程序控制方式。

从CISC到RISC

1. CISC结构和思路面临的主要问题

日趋庞大的指令系统，控制器硬件复杂，不规整，**设计与实现困难，调试和维护难度增加**，且因指令操作复杂而增加机器周期，CPI值较大，**执行时间长，不利于采用流水线技术**。

2. CISC指令集的“20%-80%律”

在程序中各种指令出现的频率悬殊很大，占指令总数20%的简单指令在程序中的出现频率占80%，而占指令总数80%的复杂指令在程序中的出现频率只占约20%。

3. VLSI技术的发展带来机遇。

VLSI工艺要求规整性，主存与控存的速度相当…

从CISC到RISC

针对CISC结构存在的这些问题，Patterson等人提出了精简指令系统计算机的设想。通过精减指令来使计算机结构变得简单、合理、有效，并克服CISC结构的上述缺点。他们提出了设计RISC机器应当遵循的一般原则。

这些原则包括：

(1) 确定指令系统时，只选择使用频度很高的那些指令，在此基础上增加少量能有效支持操作系统和高级语言实现及其他功能的最有用的指令，让指令的条数大大减少，一般不超过 100 条。

(2) 大大减少指令系统可采用的寻址方式的种类，一般不超过两种。简化指令的格式，使之也限制在两种之内，并让全部指令都具有相同的长度。

从CISC到RISC

(3) 让所有指令都在一个机器周期内完成。

(4) 扩大通用寄存器的个数，一般不少于 32 个寄存器，以尽可能减少访存操作，所有指令中只有存(STORE)、取(LOAD)指令才可访存，其他指令的操作一律都在寄存器间进行。

(5) 为提高指令执行速度，大多数指令都采用硬联控制实现， 少数指令采用微程序实现。

(6) 通过精简指令和优化设计编译程序，以简单有效的方式来支持高级语言的实现。

指令系统的发展和改进

精简指令系统计算机 RISC (Reduce Instruction Set Computer)

现代新的指令集大多采用RISC体系结构，如ARM、MIPS、RISC(-I到-V）指令集。X86发展过程中也借鉴了RISC思想。

主要特点

①指令数量少；

②指令长度固定，指令格式和寻址方式种类少；

③大部分采用RR寻址，采用Load/Store指令读写存储器，指令周期短；

④采用组合逻辑电路控制，不用或少用微程序控制。

两个不同设计方向

口复杂指令系统计算机CISC

- ✓增强指令功能，设置功能复杂的指令
- ✓面向目标代码、高级语言和操作系统
- ✓用一条指令代替一串指令

口精简指令系统计算机RISC

- ✓只保留功能简单的指令
- ✓功能较复杂的指令用子程序来实现

按CISC方向发展与改进指令系统

- **1. 面向目标程序的优化实现来改进**
  - ➢对计算机已有机器指令系统进行分析，看哪些功能仍用基本指令串实现，哪些功能改用一条新指令实现。这样既减少目标程序占用的程序空间，提高程序的运行速度，又使实现起来更容易。
  - ➢改进的思路：
    - 口(1)按统计出的各种指令和指令串的使用频度来分析改进。
    - 口(2)通过增设强功能复合指令来取代原先由常用宏指令或子程序实现的功能。复合指令用微程序解释实现，极大提高运算速度，减少了程序调用的额外开销，缩减了子程序存储空间。

按CISC方向发展与改进指令系统

- **静态使用频度：对程序中出现的各种指令及指令串进行统计得出的百分比。**

  按静态使用频度改进指令系统是着眼于减少目标程序所占用的存储空间。

- **动态使用频度：在目标程序执行过程中对出现的各种指令及指令串进行统计得出的百分比。**

  按动态使用频度改进指令系统是着眼于减少目标程序的执行时间。

按CISC方向发展与改进指令系统

- **2. 面向高级语言的优化实现来改进**
  - 尽可能缩短高级语言和机器语言的语义差距，以利于支持高级语言编译系统，缩短编译程序的长度和编译所需的时间。
  - 改进的思路：

    (1)对源程序中各种高级语言语句的使用频度进行统计分析。

    (2)面向编译，优化代码生成。

    (3)改进指令系统，使它与各种语言间的语义差距都有共同的缩小。

    (4)让机器具有分别面向各种高级语言的多种指令系统、多种系统结构，并能动态地切换。

    (5)发展高级语言计算机。

按CISC方向发展与改进指令系统

- **3. 面向操作系统的优化实现来改进**
  - 缩短操作系统与计算机系统结构之间的语义差距，以利于进一步减少运行操作系统所需要的辅助操作时间和节省操作系统软件做占用的存储空间。
  - 改进的思路：

    (1)通过对操作系统中常用的指令和指令串的使用频度进行统计和分析来改进。

    (2) 增设专用于操作系统的新指令。

    (3)把操作系统由软件子程序实现的某些功能进行硬化或固化。

    (4)发展让操作系统由专门的处理机来完成的功能分布处理系统结构。

按RISC方向发展与改进指令系统

1.减少CPI是RISC思想的精华

- 程序执行时间的计算公式：P = I• CPI • T

  其中：

  - P是执行这个程序所使用的总的时间；
  - I是这个程序所需执行的总的指令条数；
  - CPI (Cycles Per Instruction)是每条指令执行的平均周期数
  - T是一个周期的时间长度。

- RISC的速度要比CISC快3倍左右，关键是RISC的CPI减小了

按RISC方向发展与改进指令系统

2.RISC设计思想也可以用于CISC中。

- 例如：Intel公司80x86处理机的CPI在不断缩小，如：
  - 8088的CPI大于20
  - 80286的CPI大约是5.5
  - 80386的CPI进一步减小到4左右
  - 80486的CPI已经接近2
  - Pentium处理机的CPI已经与RISC十分接近
  - 目前，超标量、超流水线处理机的CPI已经达到0.5，实际上用IPC (Instruction Per Cycle)更确切。

RISC结构优势与不足总结

- 采用RISC结构后可以带来如下明显的好处：
  - 简化指令系统设计， 适合超大规模集成电路实现。
  - 提高机器的执行速度和效率。
  - 降低设计成本， 提高了系统的可靠性。
  - 可以提供直接支持高级语言的能力， 简化编译程序的设计。

- RISC结构也还存在某些不足和问题， 主要是：
  - 由于指令少，使原CISC上由单一指令完成的某些复杂功能现在需要用多条RISC指令才能完成，这实际上加重了汇编语言程序员的负担，增加了机器语言程序的长度，从而占用了较大的存储空间，加大了指令的信息流量。
  - 对浮点运算和虚拟存储器的支持虽有很大加强，但仍不够理想。
  - 相对来说， RISC机器上的编译程序要比CISC机器上的难写。

1. 寻址技术
2. 指令系统设计与优化
3. 指令系统的发展和改进
4. 设计RISC的关键技术
5. 典型的RISC处理器（阅读内容教材2.5和补充资料）

设计RISC的关键技术

1.重叠寄存器窗口（Overlapping Register Window）技术。

重叠寄存器窗口技术由美国加洲大学伯克利分校的F .Baskett提出。

实现方法：设置一个数量比较大的寄存器堆，并把它划分成很多个窗口。在每个过程使用的几个窗口中有一个窗口是与前一个过程共用，还有个窗口是与下一个过程共用。

效果：可以减少大量的访存操作。另外，要在主存中开辟一个堆栈，当调用层数超过规定层数（寄存器溢出）时，把溢出部分的寄存器中内容压入堆栈。

设计RISC的关键技术

- RISCII 共设138个寄存器，每个程序或过程可以直接访问32个寄存器。
-

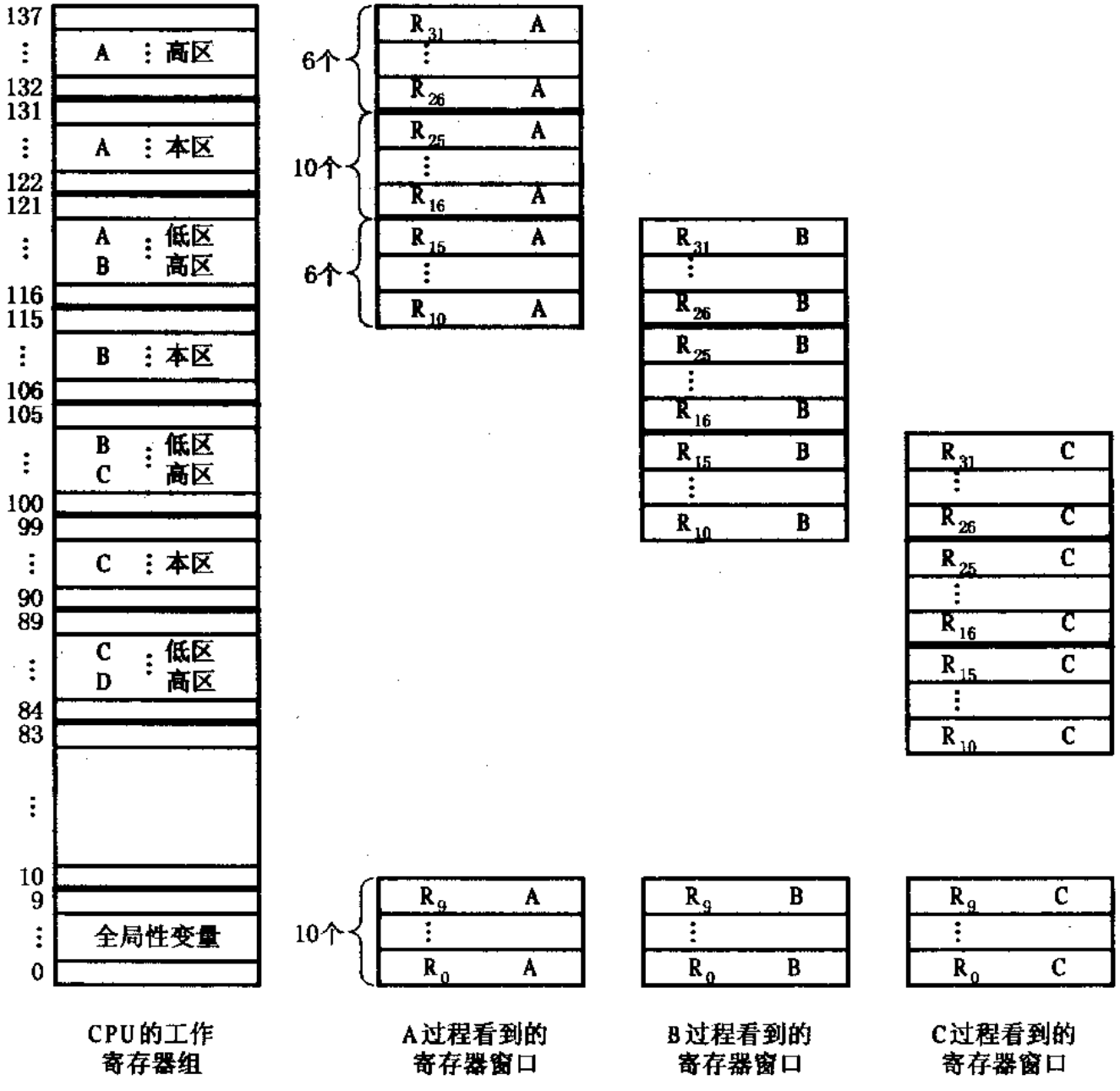


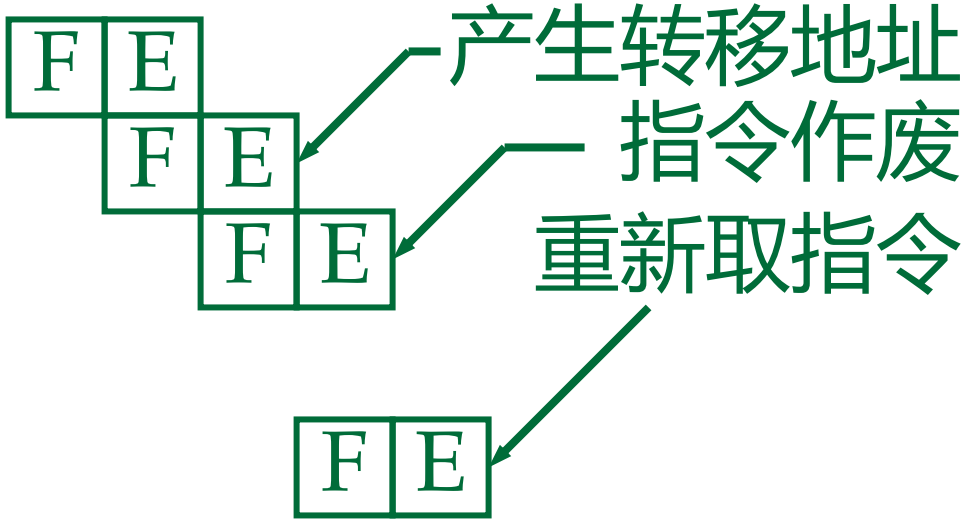
图 2.23 RISCII的重叠寄存器窗口

2.延迟转移技术（Delayed Branch）

定义：为了使指令流水线不断流，在转移指令之后插入一条不相关的有效的指令，而转移指令被延迟执行，这种技术称为延迟转移技术（延迟槽）（Delay Slot）。

- ✓采用指令延迟转移技术时，指令序列的调整由**编译器**自动进行。
- ✓无条件转移：

```
1:          add r1, r2
2:          jmp next2
3:  next1:  sub r3, r4
            ……
n:  next2:  move r4, a
```

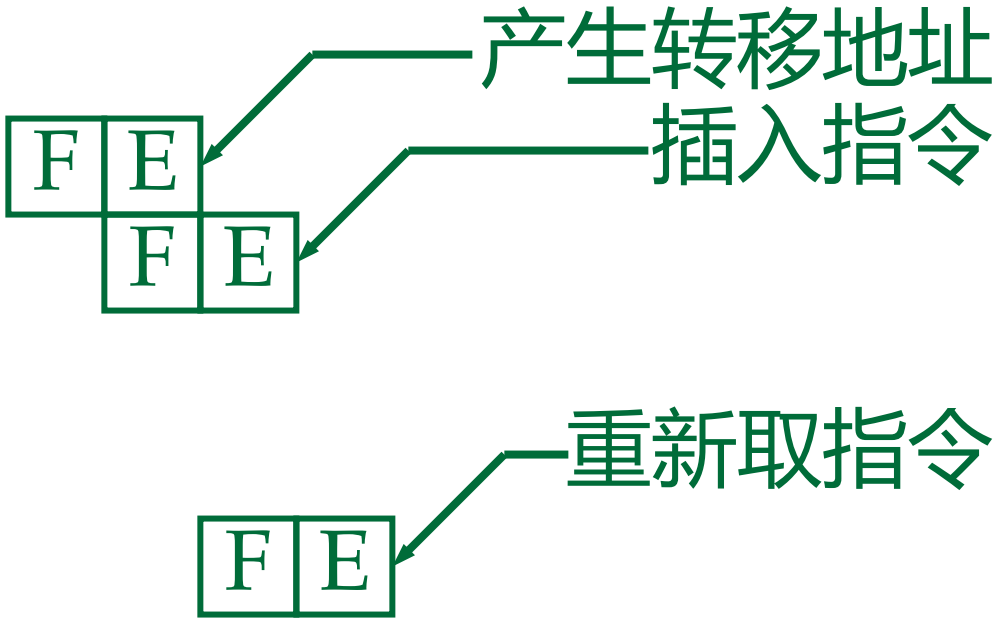


因转移指令引起的流水线断流

设计RISC的关键技术

把jmp next2指令提前执行，即把第一和第二条指令交换位置，这时，流水线不会出现断流。

```
1:          jmp next2
2:          add r1, r2
3:  next1:  sub r3, r4
            ......
n:  next2:  move r4, a
```



采用指令延迟转移技术的指令流水线

设计RISC的关键技术

条件转移指令：

调整前的指令序列：

```
1:          move r1, r2
2:          cmp r3, r4      ;(r3)与(r4)比较
3:          beq  next       ;如果(r3) = (r4)则转移到next
4:          add  r4, r5

……
n:  next: move r4, a
```

设计RISC的关键技术

调整后的指令序列：

```
1:          cmp    r3, r4        ;(r3)与(r4)比较
2:          beq    next ;如果(r3) = (r4)则转移到next
3:          move r1, r2
4:       add  r4, r5
……
n: next: move r4, a
```

采用延迟转移技术的两个限制条件

1.被移动指令在移动过程中与所经过的指令之间不能有数据相关。

2.被移动指令不破坏条件码，至少不影响后面的指令使用条件码。

如果找不到符合条件的指令，必须在条件转移指令后面插入空操作；如果指令的执行过程分为多个流水段，则要插入多条指令。

3.指令取消技术

- 采用指令延时技术，在许多情况下找不到可以用来调整的指令，故有些RISC采用指令取消技术，分为三种情况：
- 向后转移（循环程序）
  - ✓实现方法：循环体的第一条指令经调整后安排在两个位置，第一个位置是在循环体的前面，第二个位置安排在循环体的后面。
  - ✓如果转移成功，则执行循环体后面的指令，然后返回到循环体开始；否则，则取消循环体后面的指令，继续执行后面的指令。

设计RISC的关键技术

□向后转移（循环程序）

✓例如：

调整前

```
loop:  X X X
       Y Y Y
       ……
       Z Z Z
       cmp r1, r2, loop
       W W W
```

调整后

```
       X X X
loop:  Y Y Y
       ……
       Z Z Z
       cmp r1, r2, loop
       X X X
       W W W
```

效果：能够使指令流水线在绝大多数情况下不断流，由于绝大多数情况下，转移是成功的。

设计RISC的关键技术

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

口向前转移(if-then)

✓实现方法：如果转移不成功执行下条指令，否则取消下条指令。

例如：R R R ;If部分的程序代码

……

S S S ;If部分的程序代码

cmp r1, r2, thru ;若转移，则取消TTT

T T T

……

U U U

thru: VVV ;Then部分的程序代码

效果：成功与不成功的概率通常各为50%

4.指令取消技术

口隐含转移技术

- ✓应用场合：用于if..then..结构，且then部分只有一条指令
- ✓实现方法：把IF的条件取反，如果取反后的条件成立则取消下条指令，否则执行下条指令。
- ✓例子：if (a<b) then b=b+1

```
cmp >=, ra, rb ;若(ra)>=(rb)则取消下条指令
inc rb
```

设计RISC的关键技术

- 目标：通过变量重新命名消除数据相关，提高流水线执行效率

- 例：调整后的指令序列比原指令序列的执行速度快一倍

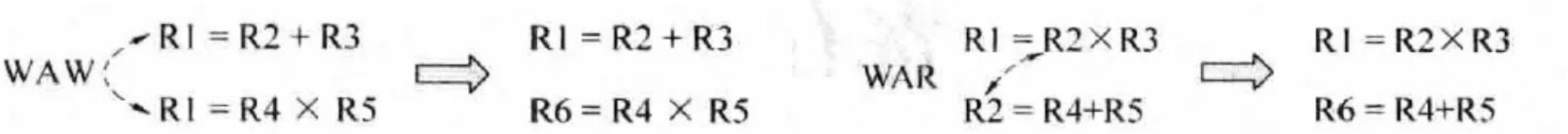
```
add  r1, r2, r3          add  r1, r2, r3
add  r3, r4, r5          mul  r6, r7, r0
mul  r6, r7, r3          add  r3, r4, r5
mul  r3, r8, r9          mul  r0, r8, r9
```

调整前　　调整后

设计RISC的关键技术

口寄存器重命名

- 寄存器重命名是将逻辑寄存器（也叫架构寄存器、通用寄存器）重命名为物理寄存器，从而消除由于逻辑寄存器的重复使用而产生的不必要的数据依赖。**物理寄存器的数量要多于逻辑寄存器的数量。**
- 数据相关性有三种类型：RAW（read after write）、WAW（write after write）、WAR（write after read）。其中只有RAW是真相关（必须按照顺序计算才能得到，与寄存器名称无关），WAW和WAR都是由乱序引发的假相关，可通过更换寄存器名字解决，如下图所示。



设计RISC的关键技术

➢5.采用认真设计和优化编译系统设计的技术。

口设A、 A+1, B, B+1 为主存单元，则程序

取A， R_a ；(A)→R_a

存R_a, B ；(R_a)→B

取A+1， R_a ；(A+1)→R_a

存R_a, B+1 ；(R_a)→B+1

该程序段实现的是将A和A+1 两个主存单元的内容转存到B和B+1 两个主存单元。由于取和存两条指令交替进行，又使用同一个寄存器R_a, 出现寄存器R_a必须先取得A的内容，然后才能由R_a存入B，即上条指令未结束之前，下条指令无法开始。后面的指令也是如此。因此，指令之间实际上不能流水，每条指令均需两个机器周期。

有办法改进吗?

设计RISC的关键技术

口可以支持流水的改进版本：

取A, R_a ; (A)→R_a
取A+1, R_b ; (A+1)→R_b
存R_a, B ; (R_a)→B
存R_b, B+1 ; (R_b)→B+1