计 算 机 体 系 结 构

第五讲

计算机科学与技术学院
舒燕君

Recap

- 量化设计的基本原则
  - ✓ 大概率事件优先原则
  - ✓ 并行性原理
  - ✓ **Amdahl**定律
  - ✓ 程序的局部性原理
  - ✓**CPU**性能公式
- 第一章作业和第二章作业一起交
- 第一章的阅读学习报告（可选做）

Quiz 1

1. 假设某台机器运行一个测试程序的执行时间为100秒，其中CPU处理时间占90%，I/O处理时间占10%，若CPU的执行速度每年能够提高50%，请问5年后在这台机器上，运行该测试程序将耗费多少秒？I/O处理时间占多少百分比？

解：

$T_{CPU} = 90s$，$T_{I/O} = 10s$

$$(1 - 0.9) + \frac{0.9}{1.5^5} = 0.1 + \frac{0.9 \times 32}{9 \times 27} = \frac{5.9}{27}$$

5年后运行该测试程序将耗费$\frac{5.9}{27} \times 100 \approx 21.85s$

I/O处理时间占$\frac{10}{21.85} \times 100\% \approx 45.77\%$

Quiz 1

2.某程序的条件判断的分支占10%，考虑运行该程序的CPU中条件分支的两种设计方案：

（1）CPU1通过比较指令设置条件码，然后测试条件码进行分支（测试分支）；

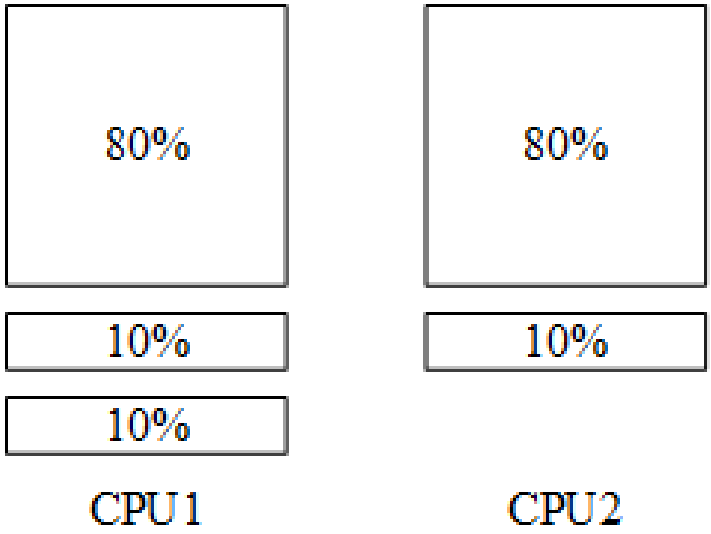
（2）CPU2在分支指令中就包含了比较过程（条件分支）。

假设条件分支指令和测试分支指令都需要2个时钟周期，其它所有指令需要1个时钟周期，由于CPU2在分支时需要比较，因此CPU1的时钟频率为CPU2的1.2倍，请对比两种CPU的设计方案。

解：该题有两种思路，答案均为CPU1比CPU2快

Quiz 1

第一种思路：

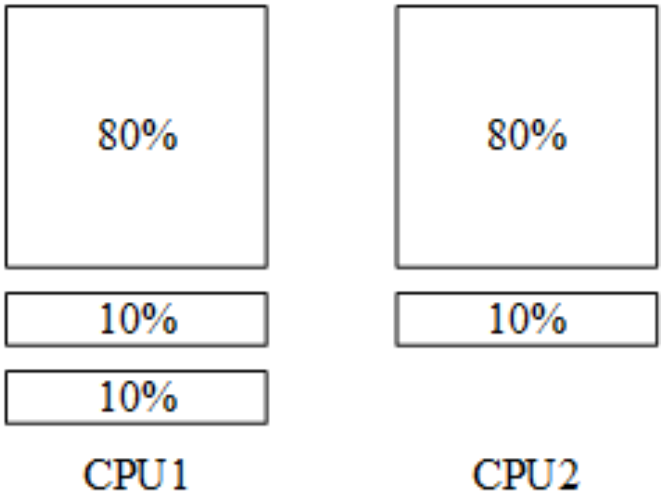


$$CPI1 = 0.8 * 1 + 0.1 * 2 + 0.1 * 1 = 1.1$$

$$CPI2 = 0.8 * 1/0.9 + 0.1 * 2/0.9 = 10/9$$

Quiz 1

第一种思路：



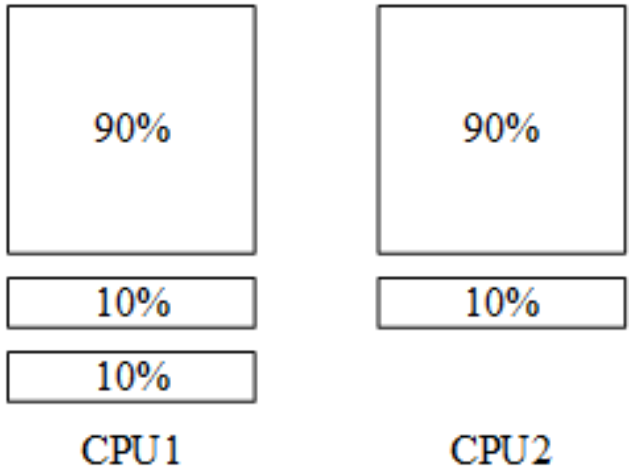
$$T_{CPU1} = IC_1 * 1.1 * Tclk_1$$

$$T_{CPU_2} = 0.9 \times IC_1 \times \frac{10}{9} \times 1.2T_{clk_1} = 1.2 \times IC_1 \times T_{clk_1}$$

$T_{CPU1} < T_{CPU2}$, CPU1比CPU2的设计方案更好

Quiz 1

第二种思路：

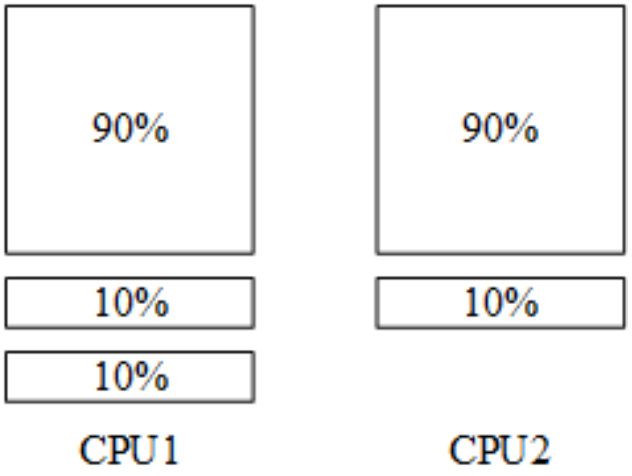


$$CPI_1 = \frac{9}{11} \times 1 + \frac{1}{11} \times 1 + \frac{1}{11} \times 2 = \frac{12}{11}$$

$$CPI_2 = 0.9 \times 1 + 0.1 \times 2 = 1.1$$

Quiz 1

第二种思路：



$$T_{CPU_1} = 1.1IC_2 \times \frac{12}{11} \times T_{clk_1} = 1.2 \times IC_2 \times T_{Clk_1}$$

$$T_{CPU_2} = IC_2 \times 1.1 \times 1.2T_{Clk_1} = 1.32IC_2 \times T_{clk_1}$$

$T_{CPU1} < T_{CPU2}$, CPU1比CPU2的设计方案更好

第2章 指令系统

2.1 指令系统概述

2.2 指令系统结构的分类

2.3 寻址方式

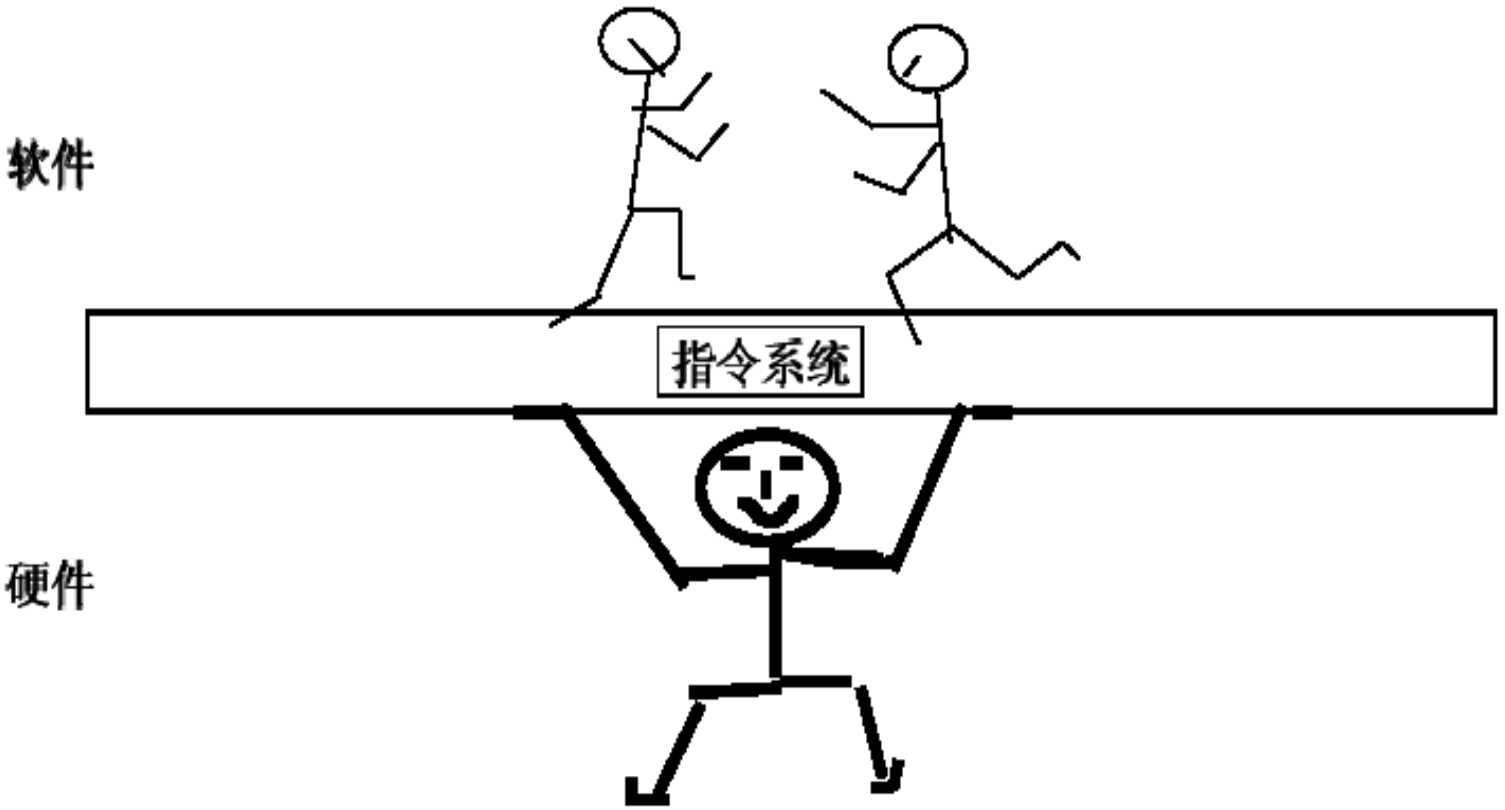
2.4 操作数类型和大小

2.5 指令系统的设计与优化

2.6 指令系统的发展和改进

2.7 指令格式举例

指令系统在计算机中的地位



2.1 指令系统概述

指令系统是计算机系统中软件与硬件交界的一个主要标志：

- 软件设计人员利用指令系统编制各种应用软件和系统软件；

- 硬件设计人员采用各种手段实现指令系统；

2.1 指令系统概述

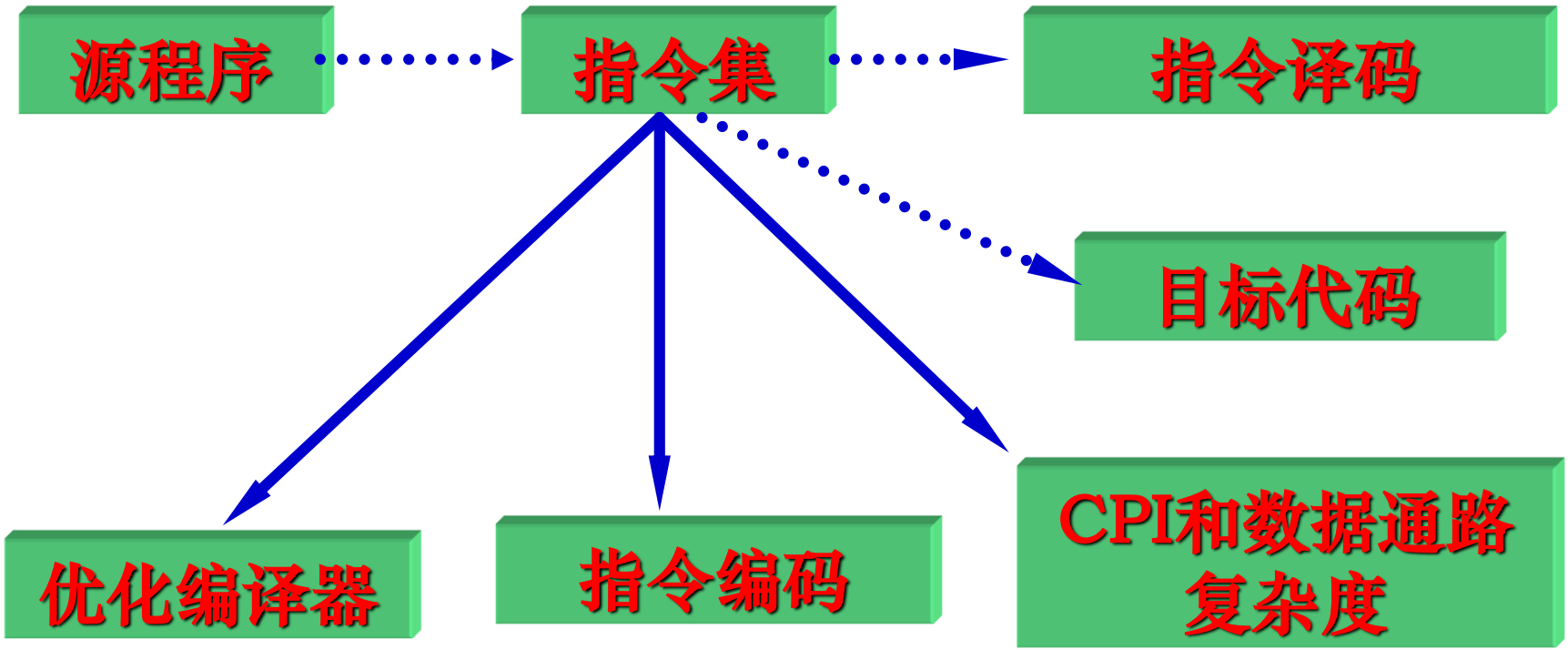
- 指令集：
  - 一些指令的集合；
  - 每条指令都是直接由CPU硬件执行。
- 指令的表示方法：
  - 二进制格式；
  - 物理存储空间组织方式是位、字节、字和多字等；
  - 当前的指令字长有：16、32、64位；
  - 可变长格式和固定长度格式。

2.1 指令系统概述

- 指令的操作十分简单，其操作由操作码编码表示。
- 每个操作需要的操作数个数为0-3个不等。
  - 操作数是一些存储单元的地址；
  - 典型的存储单元通常有：主存、寄存器、堆栈和累加器。
- 操作数地址隐含表示或显式表示。

指令系统与计算机的性能

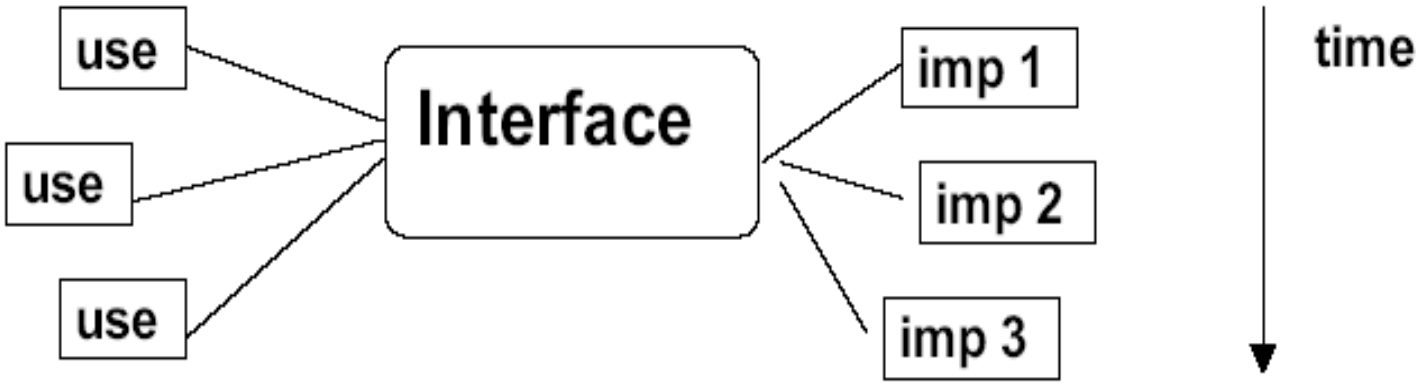
$$T_{CPU} = IC \times CPI \times T_{CLK}$$



2.1 指令系统概述

好的指令系统应该具备的条件

- **面向应用设计**
- **方便高层软件的功能实现！**
- **为底层硬件的高效率实现提供方便！**



第2章 指令系统

2.2 指令集结构的分类

- 一般来说，可以从如下五个因素考虑对计算机指令集结构进行分类，即：
  - 在CPU中操作数的存储方法；
  - 指令中显式表示的操作数个数；
  - 操作数的寻址方式；
  - 指令集所提供的操作类型；
  - 操作数的类型和大小。

2.2 指令集结构的分类

- CPU中用来存储操作数的存储单元主要有：
  - 堆栈；
  - 累加器；
  - 一组寄存器。
- 指令中的操作数可以被明确地显式给出，也可以按照某种约定隐式地给出。

2.2 指令集结构的分类

- Z=X+Y表达式在这三种类型指令集结构上的实现方法

堆栈	累加器	寄存器 (寄存器-存储器)	寄存器 (寄存器-寄存器)
PUSH X PUSH Y ADD POP Z	LOAD X ADD Y Store Z	LOAD R1,X ADD R1,Y Store R1,Z	LOAD R1,X LOAD R2,Y ADD R3,R1,R2 Store R3,Z

2.2 指令集结构的分类

- 早期的大多数机器都是采用堆栈型或累加器型指令集结构，但是自1980年以来的大多数机器均采用的是寄存器型指令集结构。主要有三个方面的原因:
  - 集成电路技术飞速发展
  - 寄存器和CPU内部其它存储单元一样，要比存储器快
  - 对编译器而言，可以更容易有效地分配和使用寄存器

通用寄存器型指令集结构

- 通用寄存器型指令集结构的主要优点：
  - 在表达式求值方面，比其它类型指令集结构都具有更大的灵活性；
  - 寄存器可以用来存放变量；
    - 减少存储器的通信量，加快程序的执行速度（因为寄存器比存储器快）
    - 可以用更少的地址位来寻址寄存器，从而可以有效改进程序的目标代码大小。

通用寄存器型指令集结构

- 两种主要的指令特性能够将通用寄存器型指令集结构（GPR）进一步细分。
  - ALU指令到底有两个或是三个操作数？
  - 在ALU指令中，有多少个操作数可以用存储器来寻址，也即有多少个存储器操作数？

通用寄存器型指令集结构

ALU指令中存储器操作数的个数	ALU指令中操作数的最多个数	结构类型	机器实例
0	3	RR	MIPS, SPARC, Alpha, PowerPC, ARM
1	2	RM	IBM 360/370, Intel 80x86, Motorola 6800
	3	RM	IBM 360/370
2	2	MM	VAX
3	3	MM	VAX

通用寄存器型指令集结构的分类

- 可以将当前大多数通用寄存器型指令集结构进一步细分为三种类型：
  - 寄存器－寄存器型
    (R－R：register-register)
  - 寄存器－存储器型
    (R－M：register-memory)
  - 存储器－存储器型
    (M－M：memory-memory)

(m，n)表示指令的n个操作数中有m个存储器操作数

三种通用寄存器型指令集结构的优缺点

- 寄存器－寄存器型（0,3）
  - 优点：
    - 指令字长固定，指令结构简洁，是一种简单的代码生成模型，各种指令的执行时钟周期数相近。
  - 缺点：
    - 与指令中含存储器操作数的指令系统结构相比，指令条数多，目标代码不够紧凑，因而程序占用的空间比较大。

三种通用寄存器型指令集结构的优缺点

- 寄存器－存储器型（1,2）
  - 优点：
    - 可以在ALU指令中直接对存储器操作数进行引用，而不必先用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：
    - 由于有一个操作数的内容将被破坏，所以指令中的两个操作数不对称。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，有可能限制指令所能够表示的寄存器个数。指令的执行时钟周期因操作数的来源（寄存器或存储器）的不同而差别比较大。

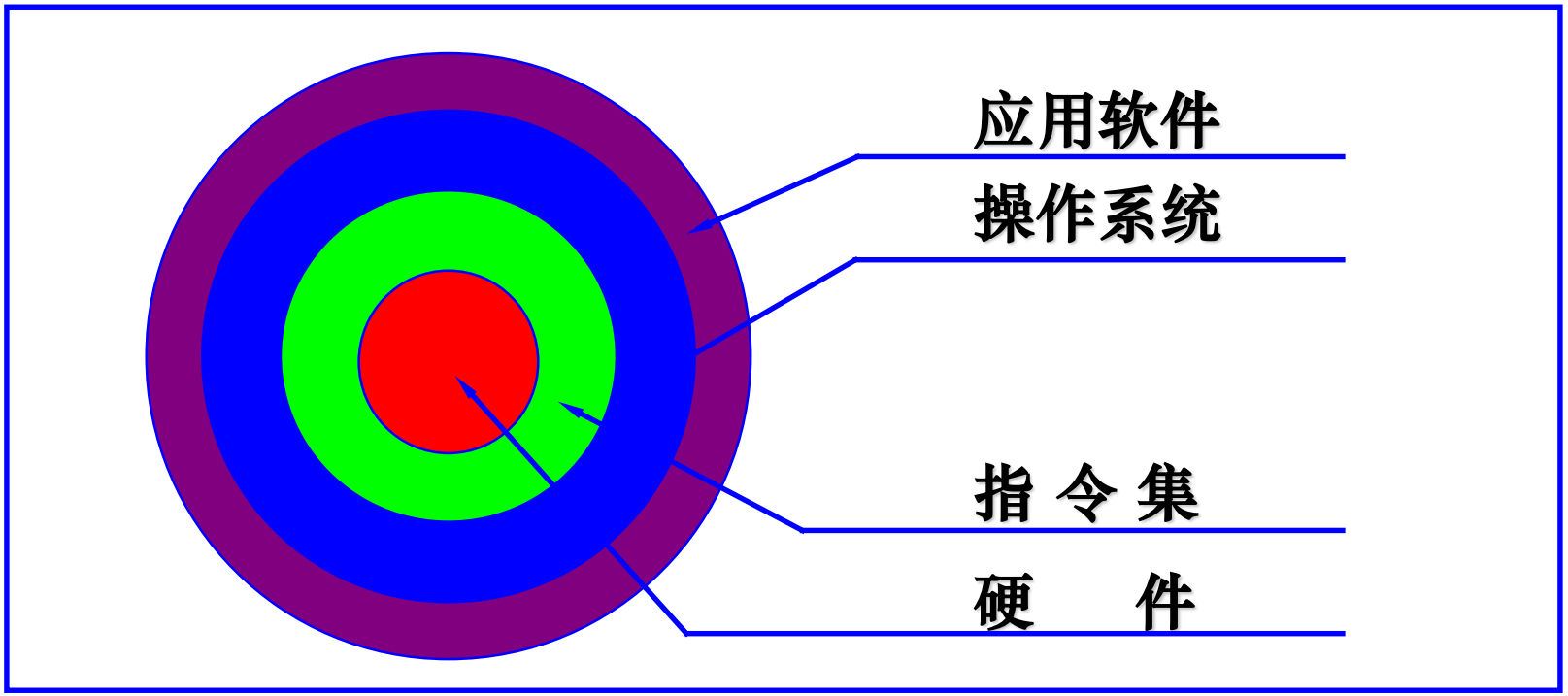
三种通用寄存器型指令集结构的优缺点

- 存储器—存储器型（（2，2）或（3，3））
  - 优点：
    - 目标代码最紧凑，不需要设置存储器来保存变量。
  - 缺点：
    - 指令字长变换很大，特别是3个操作数指令。而且每条指令完成的工作也差别很大。对存储器的频率访问会使存储器成为瓶颈。这种类型的指令系统现在已经不用了。

三种类型指令集结构的优缺点

指令集结构类型	优点	缺点
寄存器-寄存器（0，3）	指令字长固定，指令结构简洁，是一种简单的代码生成模型，各种指令的执行时钟周期数相近	与指令中含存储器操作数的指令系统结构相比，指令条数多，目标代码不够紧凑，因而程序占用的空间比较大
寄存器-存储器型（1，2）	可以在ALU指令中直接对存储器操作数进行引用，而不必先用load指令进行加载，容易对指令进行编码，目标代码比较紧凑	由于有一个操作数的内容将被破坏，所以指令中的两个操作数不对称。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，有可能限制指令所能够表示的寄存器个数。指令的执行时钟周期因操作数的来源（寄存器或存储器）的不同而差别比较大
存储器-存储器（2，2）或（3，3）	目标代码最紧凑，不需要设置存储器来保存变量	指令字长变换很大，特别是3个操作数指令。而且每条指令完成的工作也差别很大。对存储器的频率访问会使存储器成为瓶颈。这种类型的指令系统现在已经不用了

指令集结构设计概观



操作码	寻址方式	操作数	寻址方式	操作数

第2章 指令系统

2.1 指令系统概述

2.2 指令系统结构的分类

2.3 寻址方式

2.4 操作数的类型和大小

2.5 指令系统的设计与优化

2.6 指令系统的发展和改进

2.7 指令格式举例

2.3 寻址方式

- 在通用寄存器型指令集结构中，一般是利用寻址方式指明指令中的操作数是一个常数、一个寄存器操作数，或是一个存储器操作数。

2.3 寻址方式

- 寻址实际上是从形式地址到实际地址的转换。形式地址由指令描述，实际地址也称为有效地址。
- 有效地址指明的是存储器单元的地址或寄存器地址。
- 必须加速有效地址生成。

2.3 寻址方式

- 常用的一些操作数寻址方式
  - ←：赋值操作
  - Mem：存储器
  - Regs：寄存器组
  - 方括号：表示内容
    - Mem[ ]：存储器的内容
    - Regs[ ]：寄存器的内容
    - Mem[Regs[R1]]：以寄存器R1中的内容作为地址的存储器单元中的内容

常用的一些操作数寻址方式

- 寄存器寻址
  - 指令实例：**Add R4 , R3**
  - 含义：**Regs[R4]←Regs[R4]＋Regs[R3]**
- 立即值寻址
  - 指令实例：**Add R4 , #3**
  - 含义：**Regs[R4]←Regs[R4]＋3**

常用的一些操作数寻址方式

- 偏移寻址
  - 指令实例：**Add R4 , 100(R1)**
  - 含义：
    - **Regs[R4]←Regs[R4]＋Mem[100+Regs[R1]]**
- 寄存器间接寻址
  - 指令实例：**Add R4 , (R1)**
  - 含义：
    - **Regs[R4]←Regs[R4]＋Mem[Regs[R1]]**

常用的一些操作数寻址方式

- 索引寻址
  - 指令实例：**Add R3 , (R1 + R2)**
  - 含义：
    - **Regs[R3]←Regs[R3]＋Mem[Regs[R1]+Regs[R2]]**
- 直接寻址或绝对寻址
  - 指令实例：**Add R1 , (1001)**
  - 含义：
    - **Regs[R1]←Regs[R1]＋Mem[1001]**

常用的一些操作数寻址方式

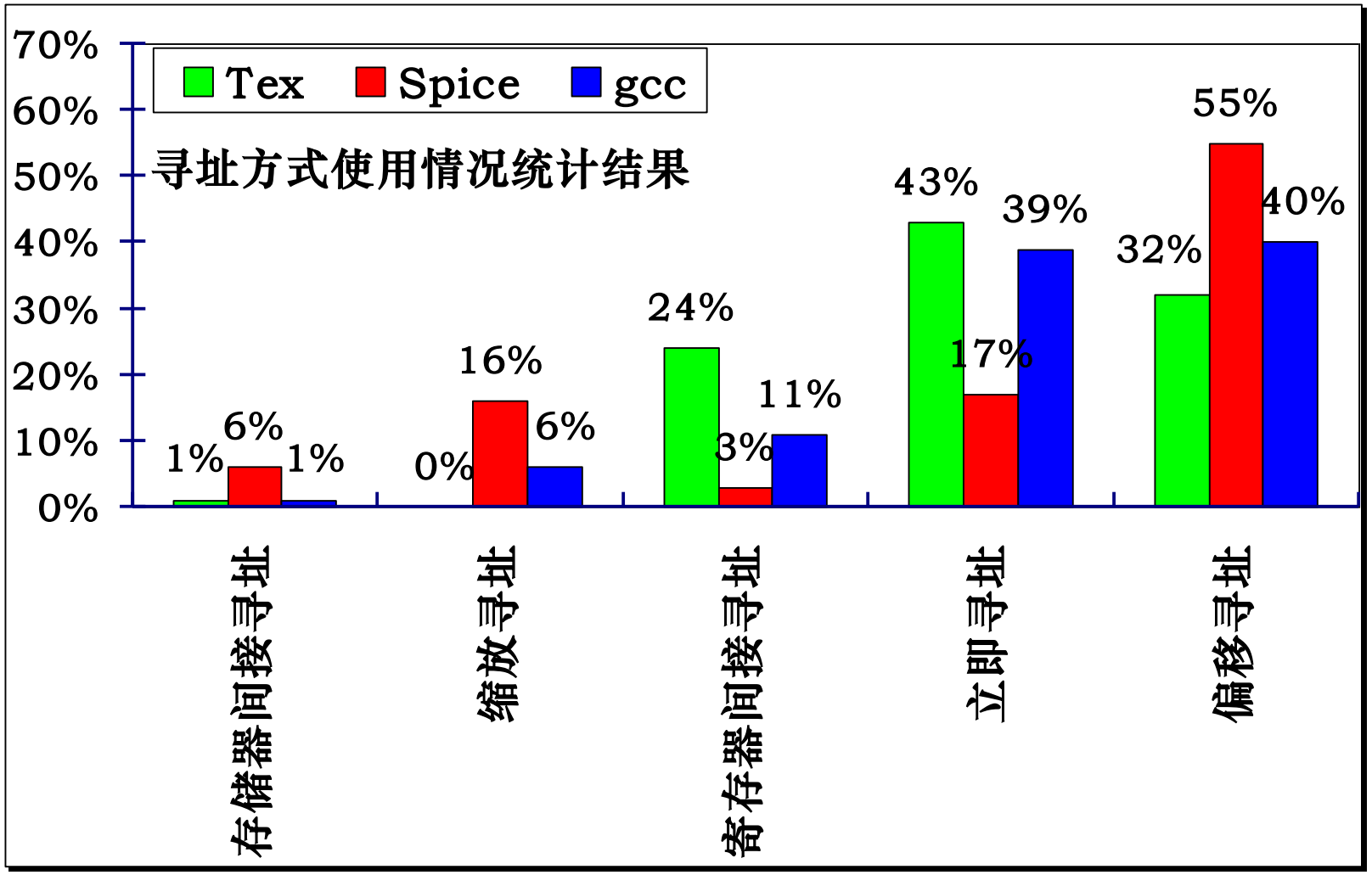
- 存储器间接寻址
  - 指令实例：**Add R1 , @(R3)**
  - 含义：
    - **Regs[R1]←Regs[R1]＋Mem[Mem[Regs[R3]]]**
- 自增寻址
  - 指令实例：**Add R1 , (R2)+**
  - 含义：
    - **Regs[R1]←Regs[R1]＋Mem[Regs[R2]]**
    - **Regs[R2]←Regs[R2]＋d**

常用的一些操作数寻址方式

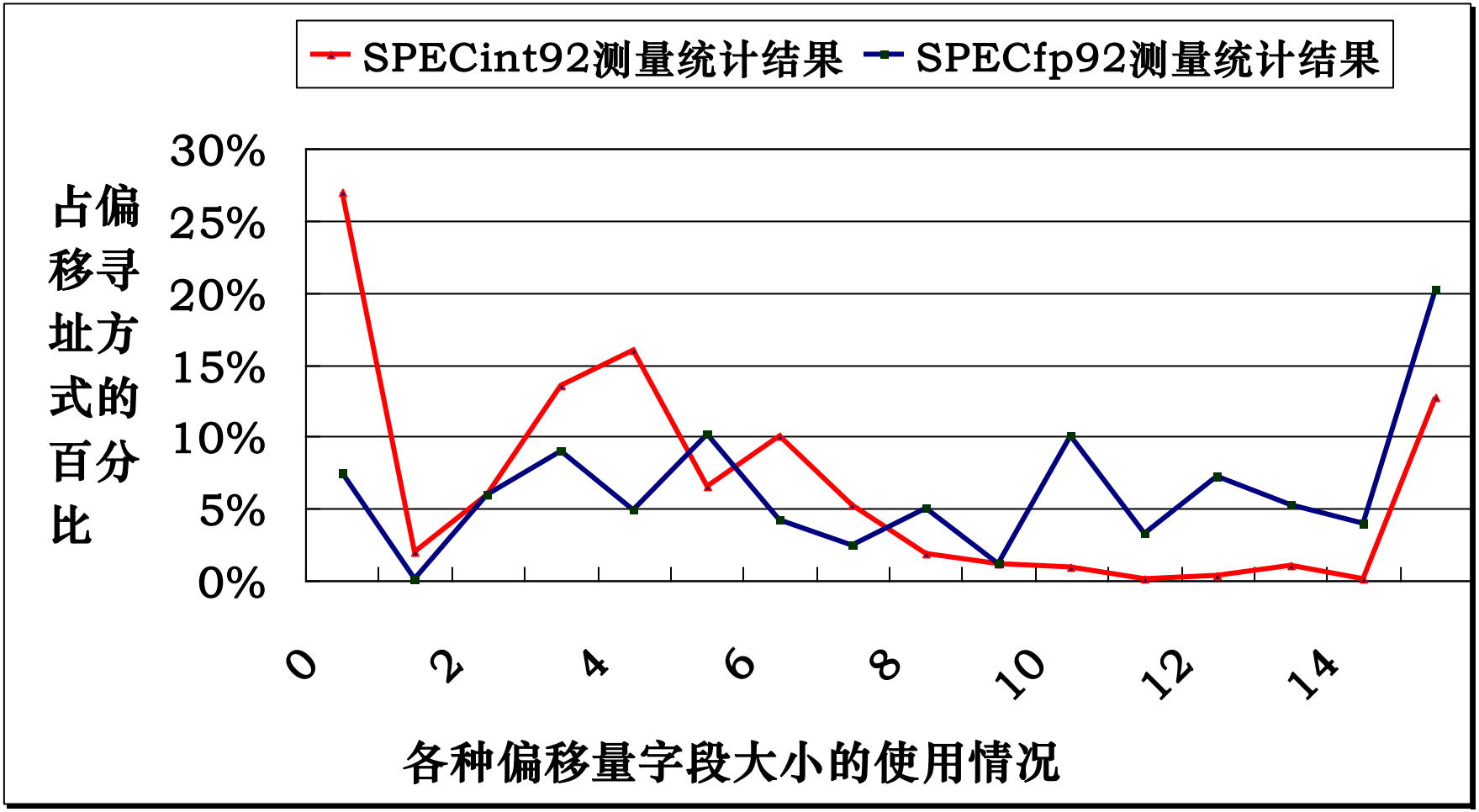
- 自减寻址
  - 指令实例：**Add R1, -(R2)**
  - 含义：
    - **Regs[R2]←Regs[R2]－d**
    - **Regs[R1]←Regs[R1]+Mem[Regs[R2]]**
- 缩放寻址
  - 指令实例：**Add R1 , 100(R2)[R3]**
  - 含义：
    - **Regs[R1]←Regs[R1]＋Mem[100＋Regs[R2]+ Regs[R3]*d]**

寻址方式	指令实例	含　　义
寄存器寻址	ADD R1 , R2	Regs[R1]←Regs[R1] + Regs[R2]
立即值寻址	ADD R3 , #6	Regs[R3]←Regs[R3] + 6
偏移寻址	ADD R3 , 120(R2)	Regs[R3]←Regs[R3] + Mem[120+Regs[R2]]
寄存器间接寻址	ADD R4 , (R2)	Regs[R4]←Regs[R4] + Mem[Regs[R2]]
索引寻址	ADD R4 , (R2 + R3)	Regs[R4]←Regs[R4] + Mem[Regs[R2]+Regs[R3]]
直接寻址或 绝对寻址	ADD R4 , (1010)	Regs[R4]←Regs[R4] + Mem[1010]
存储器间接寻址	ADD R2 , @(R4)	Regs[R2]←Regs[R2] + Mem[Mem[Regs[R4]]]
自增寻址	ADD R1 , (R2)+	Regs[R1]←Regs[R1] + Mem[Regs[R2]] Regs[R2]←Regs[R2] + d
自减寻址	ADD R1, -(R2)	Regs[R2]←Regs[R2] - d Regs[R1]←Regs[R1]+Mem[Regs[R2]]
缩放寻址	ADD R1 , 80(R2)[R3]	Regs[R1]←Regs[R1] + Mem[80 + Regs[R2] + Regs[R3]*d]

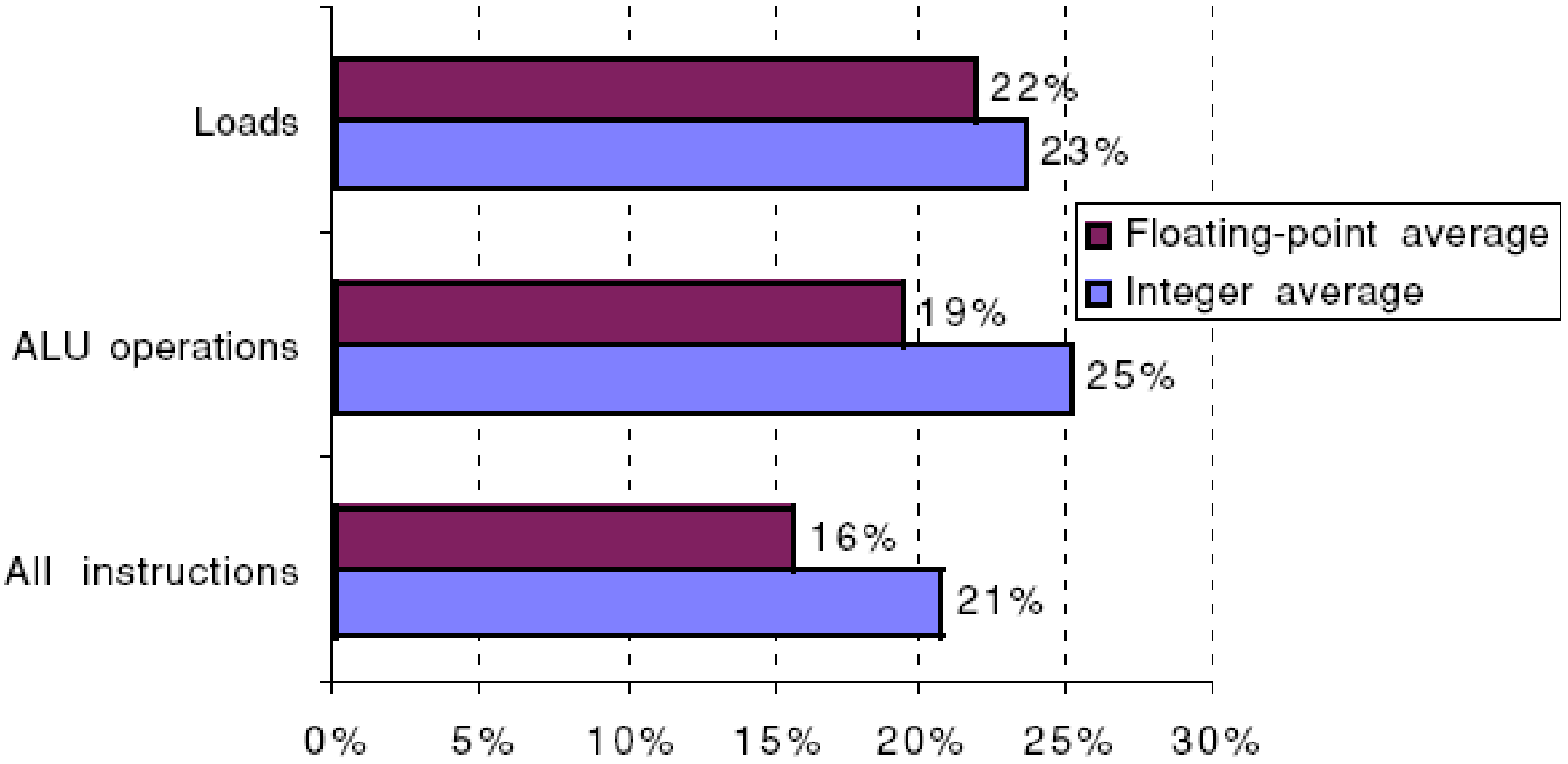
常用的一些操作数寻址方式



偏移寻址

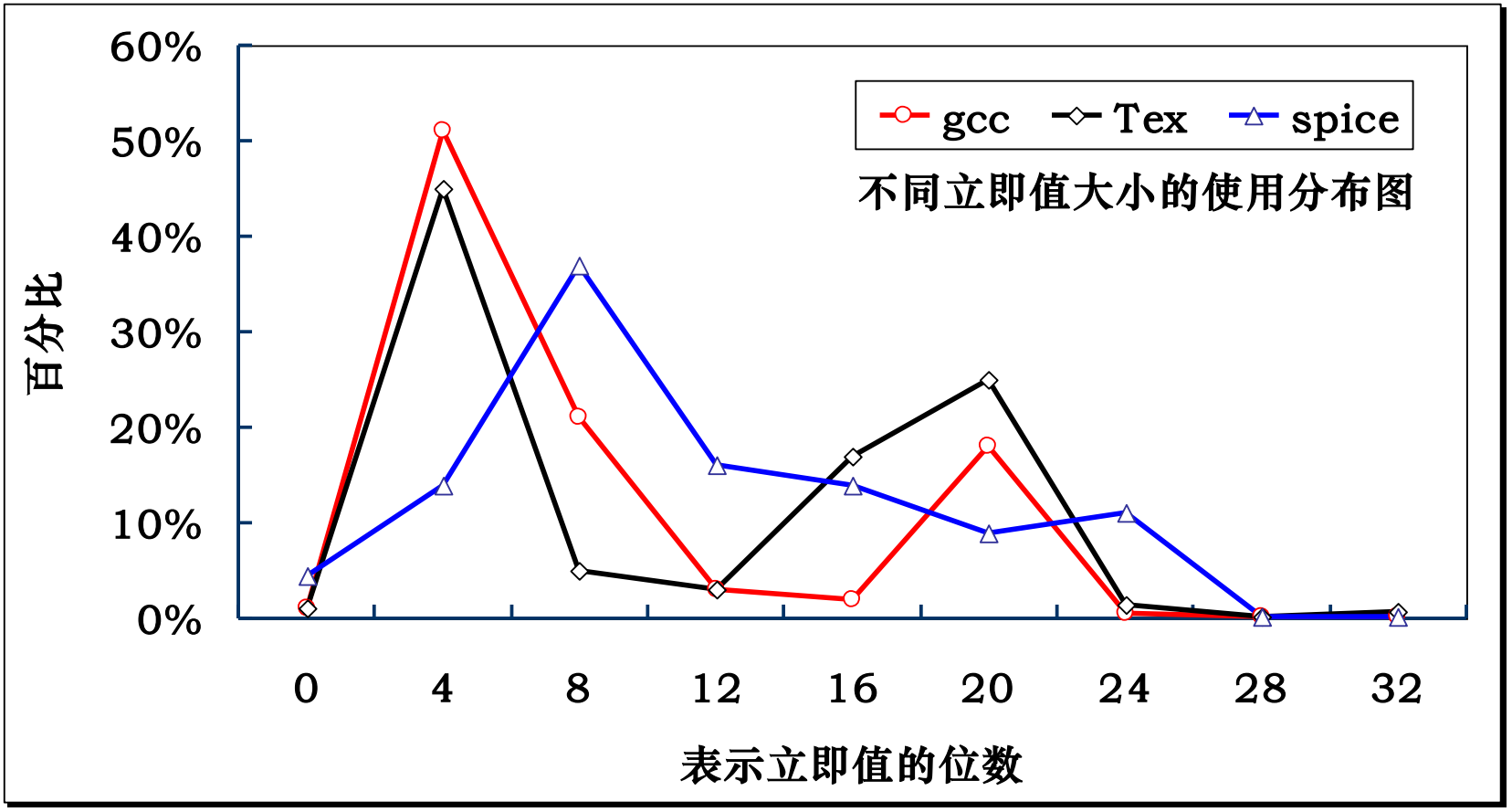

SPECint92测量统计结果
SPECfp92测量统计结果
占偏移寻址方式的百分比
30%
25%
20%
15%
10%
5%
0%
0
2
4
6
8
10
12
14
各种偏移量字段大小的使用情况

立即寻址



大约1/4的load指令和ALU指令采用了立即数寻址。

立即寻址


60%
50%
40%
30%
20%
10%
0%
百分比
gcc
Tex
spice
不同立即值大小的使用分布图
0
4
8
12
16
20
24
28
32
表示立即值的位数

2.3 寻址方式

- 两种表示寻址方式的方法
  - 将寻址方式编码于操作码中，由操作码描述相应操作的寻址方式。
  - 适合：处理机采用load-store结构，寻址方式只有很少几种。
    - ✓ MIPS指令集

2.3 寻址方式

- 两种表示寻址方式的方法
  - 在指令字中设置专门的寻址字段，用以直接指出寻址方式。
    - 灵活，操作码短，但需要设置专门的寻址方式字段，而且操作码和寻址方式字段合起来所需要的总位数可能会比隐含方法的总位数多。

适合：处理机具有多种寻址方式，且指令有多个操作数。

✓ VAX11中源操作数和目的操作数各有4位寻址方式位

2.3 寻址方式

- 一个需要注意的问题：物理地址空间的信息如何存放？

  如何在存储器中存放不同宽度的信息？

  以IBM370为例子进行讨论。

  - 信息有字节、半字（双字节）、单字（4字节）和双字（8字节）等宽度。
  - 主存宽度为8个字节。采用按字节编址，各类信息都是用该信息的首字节地址来寻址。
  - 允许它们任意存储
  - 很可能会出现一个信息跨存储字边界而存储于两个存储单元中

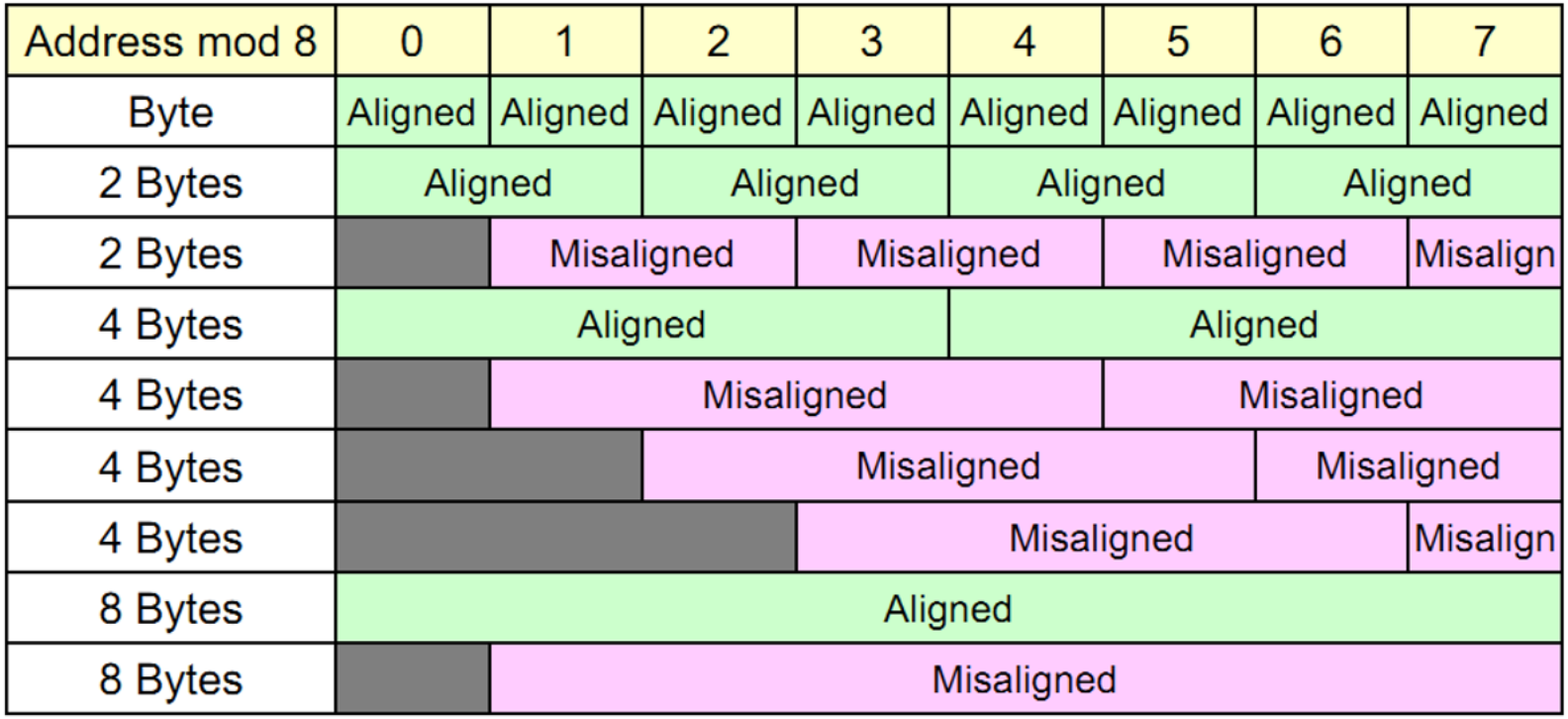
2.3 寻址方式

- 信息宽度不超过主存宽度的信息必须存放在一个存储字内，不能跨边界。
  - 必须做到：信息在主存中存放的起始地址必须是该信息宽度（字节数）的整数倍

    信息存储的整数边界概念
  - 满足以下条件
    - 字节信息的起始地址为：×…××××
    - 半字信息的起始地址为：×…×××0
    - 单字信息的起始地址为：×…××00
    - 双字信息的起始地址为：×…×000
  - 存在存储空间的浪费，但保证访问速度。

2.3 寻址方式

- 对s字节的对象访问地址为A，如果A mod s =0 称为边界对齐。
- 边界对齐的原因是存储器本身读写的要求，存储器本身读写通常就是边界对齐的，对于不是边界对齐的对象的访问可能要导致存储器的两次访问，然后再拼接出所需要的数（或发生异常）。



第2章 指令系统

2.1 指令系统概述

2.2 指令系统结构的分类

2.3 寻址方式

2.4 操作数类型和大小

2.5 指令系统的设计与优化

2.6 指令系统的发展和改进

2.7 指令格式举例

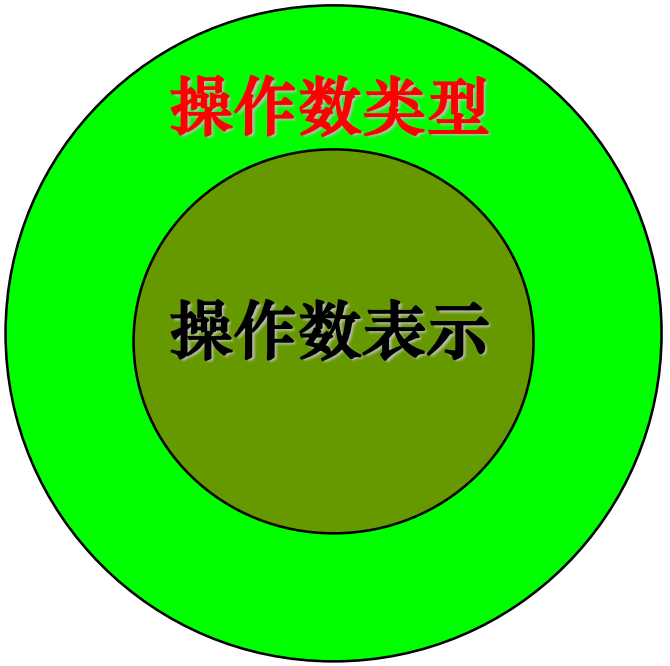
2.4 操作数类型和大小

一、操作数类型

- 操作数类型和操作数表示也是软硬件主要界面之一。
- 操作数类型是面向应用、面向软件系统所处理的各种数据结构。
- 操作数表示是硬件结构能够识别、指令系统可以直接调用的那些结构。

一、操作数类型

- 操作数表示所表征的那些操作数类型，是应用软件和系统软件所处理的操作数类型的子集。



一、操作数类型

- 确定操作数表示实际上也是软硬件取舍折衷的问题
  - 计算机即使只具有最简单的操作数表示，如只有整数（定点）表示法，也可以通过软件方法处理各种复杂的操作数类型，但是这样会大大降低系统的效率。
  - 如果各种复杂的操作数类型均包含在操作数表示之中，无疑会大大提高系统的效率，但是所花费的硬件代价也很高。

一、操作数类型

- 整数（定点）：二进制补码表示；其大小可以是字节（8位）、半字（16位）或单字（32位）。
- 浮点：可以分为单精度浮点（单字大小）和双精度浮点（双字大小）。当前普遍采用的是IEEE 754浮点操作数表示标准。
- 字符和字符串：8位ASCII码表示。

一、操作数类型

- 十进制：面向商业应用，通常采用“压缩十进制”或“二进制编码十进制（BCD）”表示。压缩十进制数据表示用4位编码数字0～9，然后将两个十进制数字压缩在一个字节中存储。如果将十进制数字直接用字符串来表示，就叫做“非压缩十进制”表示法。

一、操作数类型

- 操作数类型的表示主要有如下两种方法：
  - 操作数的类型可以由操作码的编码指定，这也是最常见的一种方法；
  - 数据可以附上由硬件解释的标记（tag），由这些标记指定操作数的类型，从而选择适当的运算。然而有标记数据的机器却非常少见。

二、操作数大小

- 一般的操作数类型大小选择主要有：字节、半字（16位）、单字（32位）、和双字（64位）。