计 算 机 体 系 结 构

第六讲

计算机科学与技术学院

舒燕君

Recap

- 指令系统概述
  - ✓ 指令的表示方法
  - ✓ 指令系统与计算机的性能
- 指令系统的分类
  - ✓堆栈、累加器、寄存器
  - ✓ 通用寄存器型指令集结构分类及优缺点
- 寻址方式
- 操作数的类型和大小

2.2 指令集结构的分类

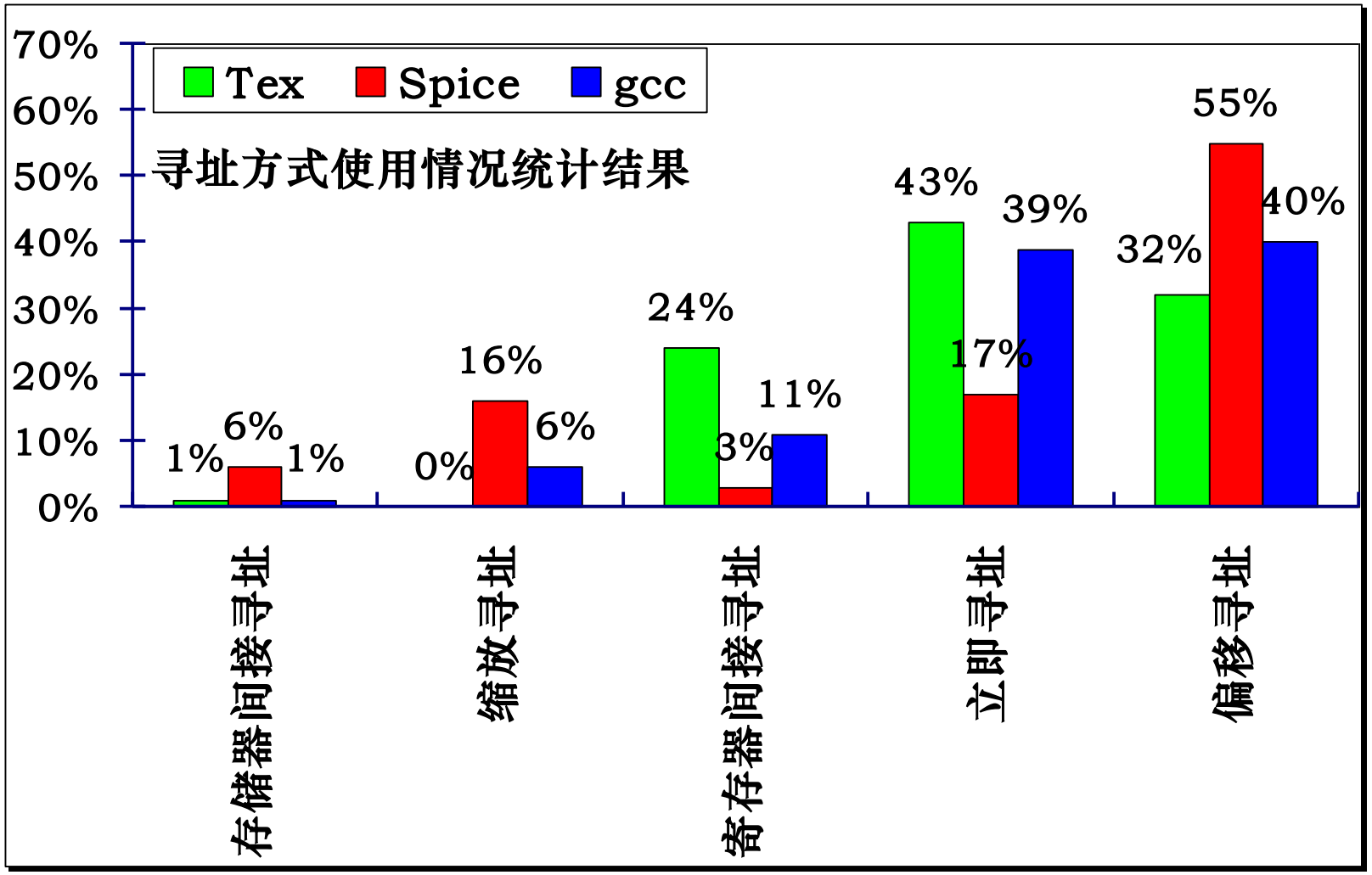
- Z=X+Y表达式在这三种类型指令集结构上的实现方法

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

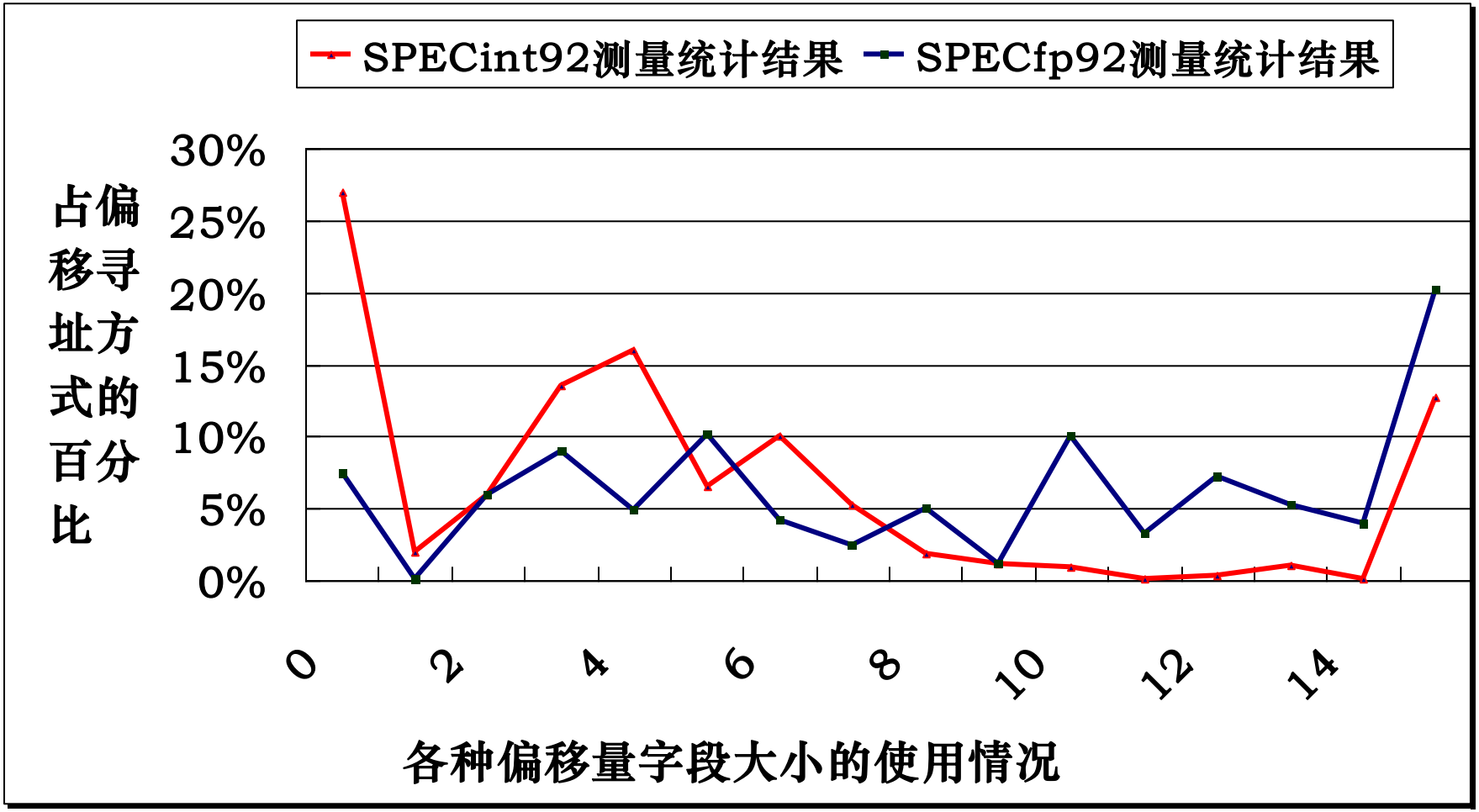
三种类型指令集结构的优缺点

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

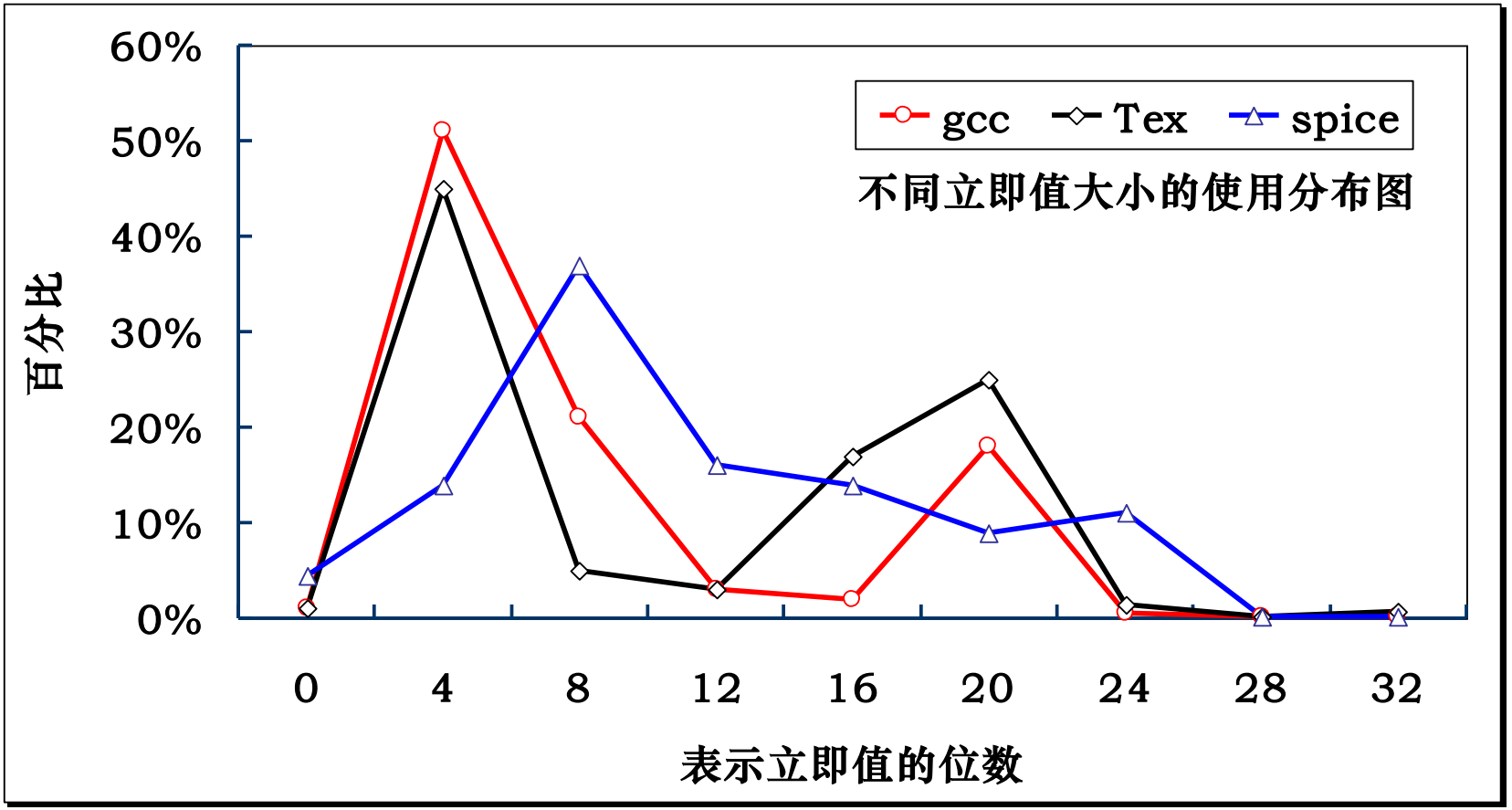
常用的一些操作数寻址方式



偏移寻址

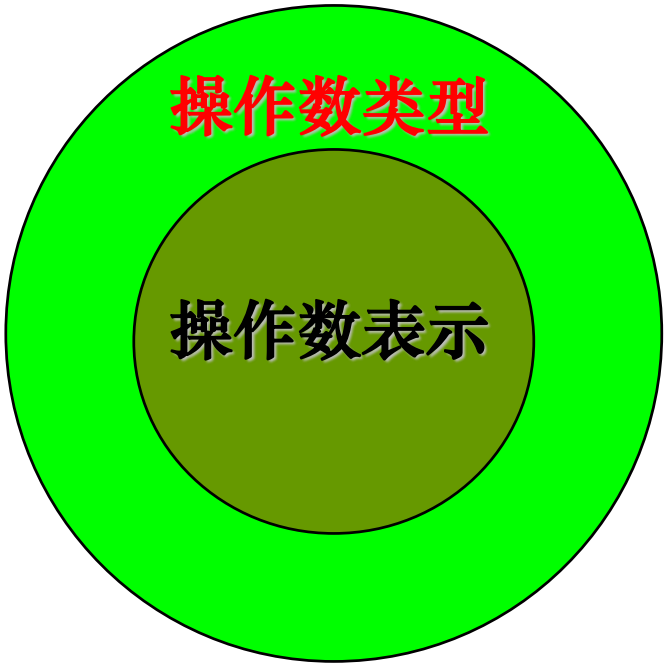

SPECint92测量统计结果
SPECfp92测量统计结果
占偏移寻址方式的百分比
30%
25%
20%
15%
10%
5%
0%
0
2
4
6
8
10
12
14
各种偏移量字段大小的使用情况

立即寻址


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

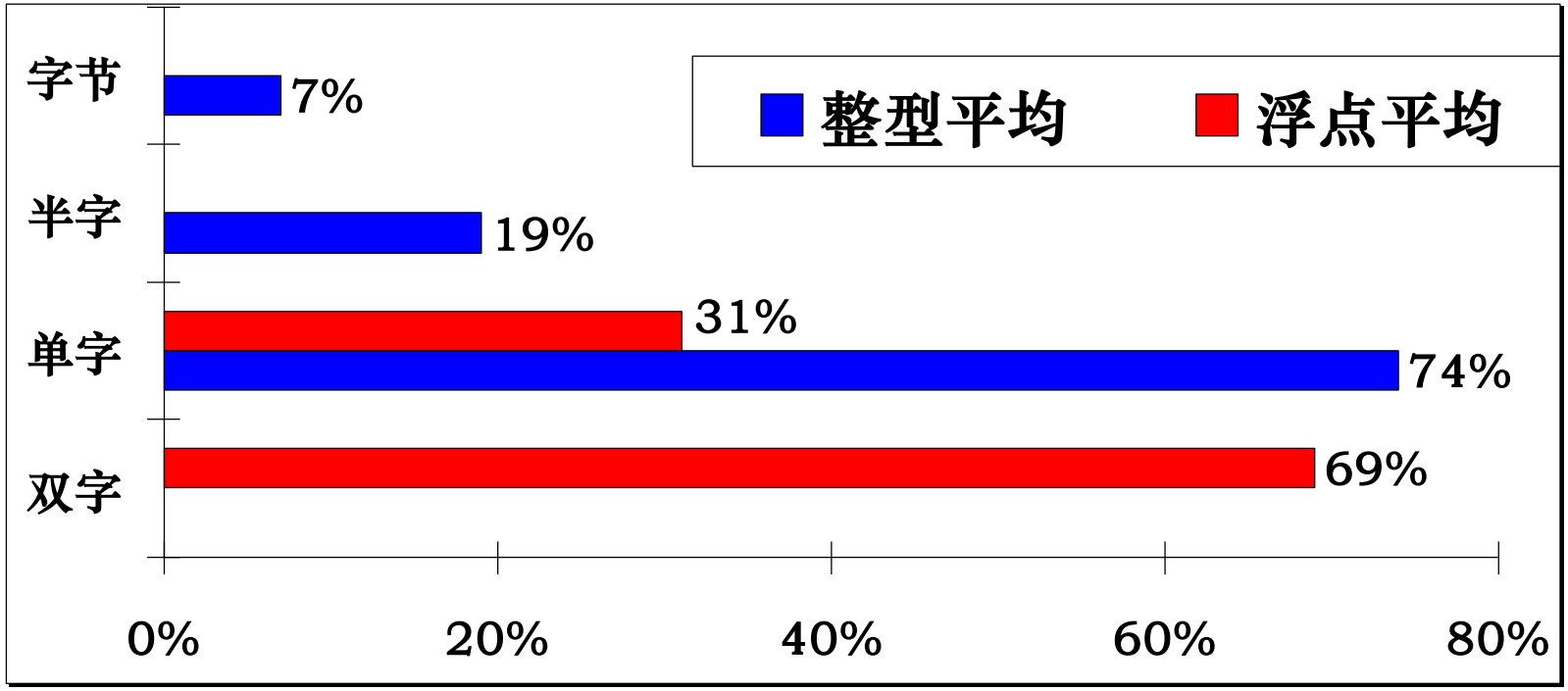
一、操作数类型

- 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。



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）。



第2章 指令系统

2.5 指令系统的设计和优化

2.5.1 指令系统设计的基本原则

2.5.2 控制指令

2.5.3 指令操作码的优化

2.5.1 指令系统设计的基本原则

1. 指令系统的设计

- 首先考虑所应实现的基本功能，确定哪些基本功能应该由硬件实现，哪些功能由软件实现比较合适。
- 包括
  - 指令的功能设计
  - 指令格式的设计

2.5.1 指令系统设计的基本原则

1. 指令系统的设计

– 一种指令系统中的指令到底要支持哪些类型的操作呢？这就是所谓的指令系统功能设计问题。

2.5.1 指令系统设计的基本原则

2. 在确定哪些基本功能用硬件来实现时，主要考虑3个因素：速度、成本、灵活性。

- 硬件实现的特点

  速度快、成本高、灵活性差

- 软件实现的特点

  速度慢、价格便宜、灵活性好

3. 对指令系统的基本要求

完整性、规整性、正交性、高效率、兼容性

2.5.1 指令系统设计的基本原则

- 完整性：在一个有限可用的存储空间内，对于任何可解的问题，编制计算程序时，指令系统所提供的指令足够使用。
  - 要求指令系统功能齐全、使用方便
  - 下表为许多指令系统结构都包含的一些指令类型
    - 前4类属于通用计算机系统的基本指令
    - 对于最后4种类型的操作，不同指令系统结构的支持大不相同 。

指令集操作的分类

算术和逻辑运算	整数的算术和逻辑操作：加、减、与、或等
数据传输	存数/取数
控制	分支、跳转、过程调用和返回、自陷等
系统	操作系统调用、虚拟存储器管理等
浮点	浮点操作：加、乘等
十进制	十进制加、十进制乘、十进制到字符的转换
字符串	字符串移动、字符串比较、字符串搜索等
图形	像素操作、压缩/解压操作等

2.5.1 指令系统设计的基本原则

- **规整性：**主要包括对称性和均匀性。
  - 对称性：所有与指令系统有关的存储单元的使用、操作码的设置等都是对称的。
    - ✓例如，在存储单元的使用上，所有通用寄存器都要同等对待。在操作码的设计上，如果设置了**A-B**的指令，就应该也设置**B-A**的指令。
  - 均匀性：指对于各种不同的操作数类型、字长、操作种类和数据存储单元，指令的设置都要同等对待。
    - ✓例如，如果某机器有**5**种数据表示，**4**种字长，两种存储单元，则要设置**5*4*2=40**种同一操作的指令（如加法指令）。

2.5.1 指令系统设计的基本原则

- 正交性：在指令中各个不同含义的字段，如操作类型、数据类型、寻址方式字段等，在编码时应互不相关、相互独立。
- 高效率：指指令的执行速度快、使用频度高。
- 兼容性：主要是要实现向后兼容，指令系统可以增加新指令，但不能删除指令或更改指令的功能。

2.5 指令系统的设计和优化

2.5.1 指令系统设计的基本原则

2.5.2 控制指令

2.5.3 指令操作码的优化

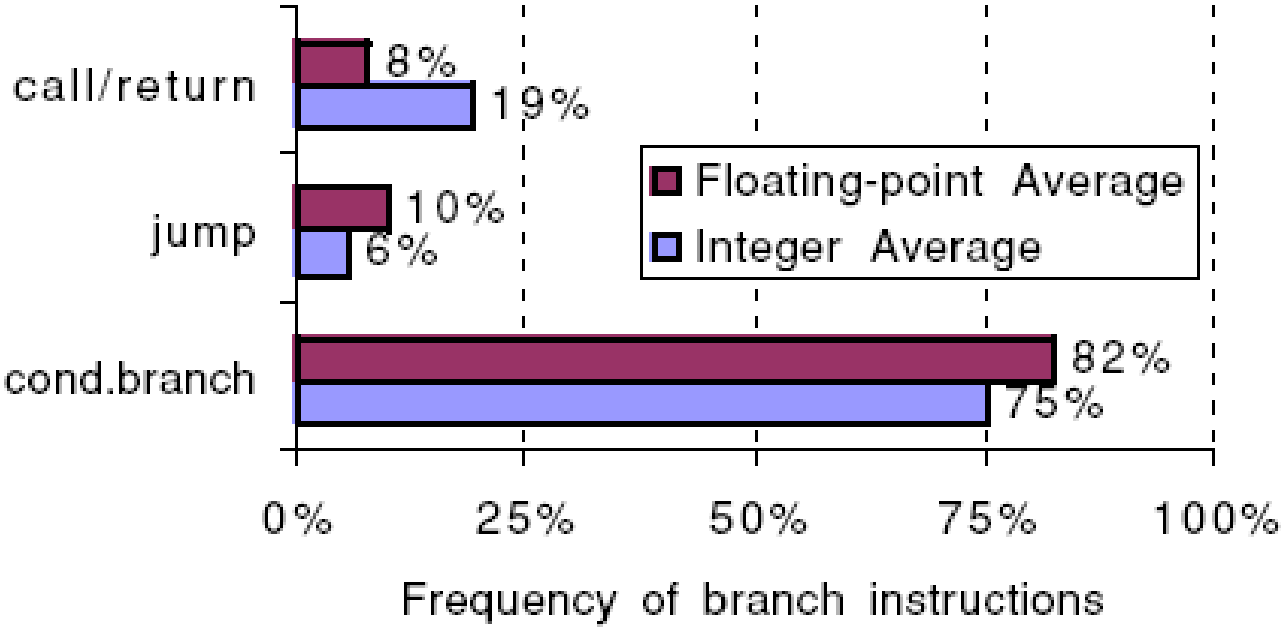
2.5.2 控制指令

- “跳转”（Jump）：当控制指令为无条件改变控制流时，我们称之为“跳转”。
- “分支”（Branch）：而当控制指令是有条件改变控制流时，我们称之为“分支”。

程序中控制流程的改变情况包括：

- 跳转（jump）；
- 条件分支（conditional branch）；
- 过程调用（call）；
- 过程返回（return）。

控制指令的使用频率

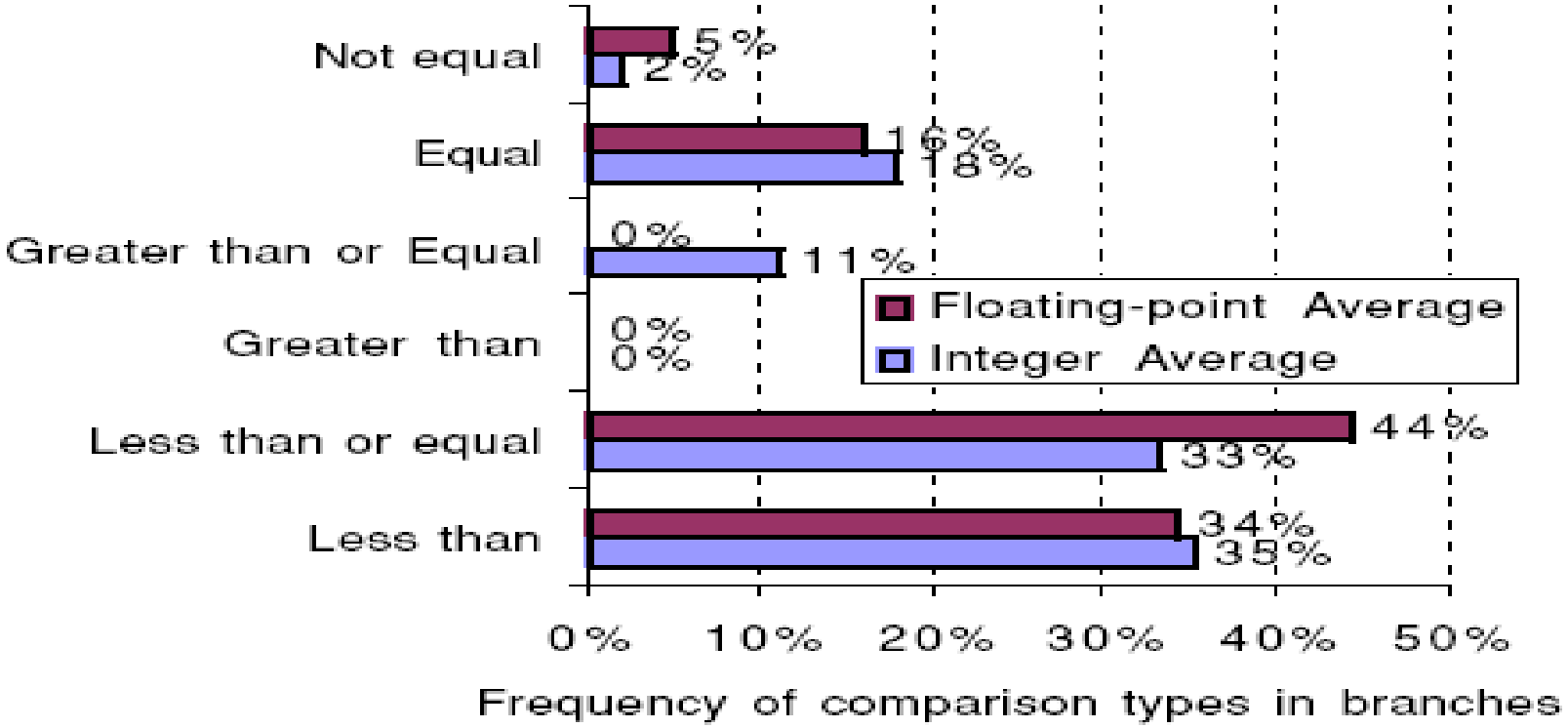


改变控制流的大部分指令是分支指令（条件转移）。

条件分支指令的表示

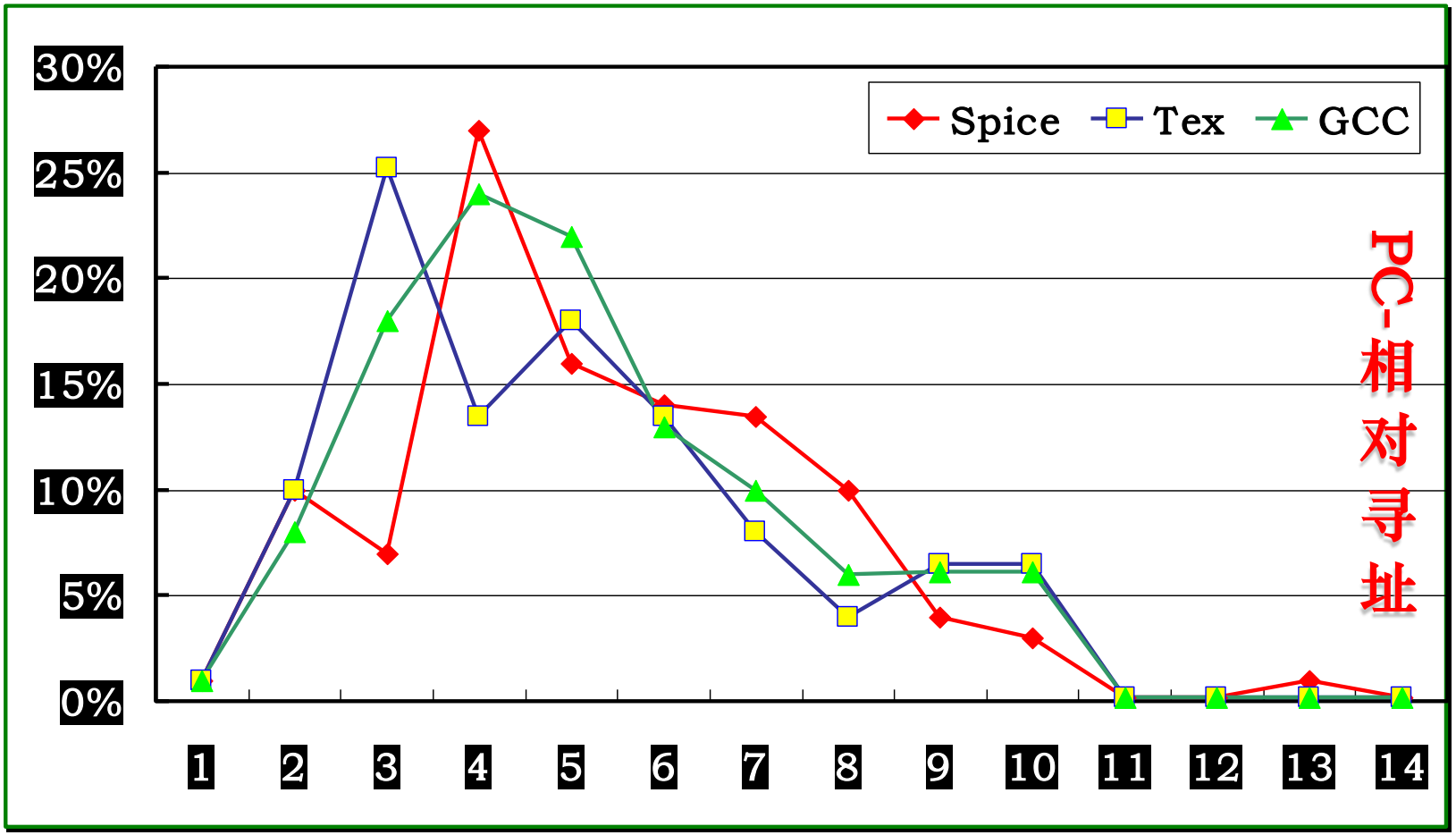
分支条件表示	优　点	缺　点
条件码（CC）：在程序的控制下，由ALU操作设置特殊的位	可以自由设置分支条件	必须从一条指令将分支条件信息传送到分支指令，所以CC是额外状态，条件码限制了指令执行顺序
条件寄存器：比较指令把比较结果放入任何一个寄存器，检测时就检测该寄存器	简单	占用了一个寄存器
比较分支：比较操作是分支指令的一部分，比较受限制	一条指令完成了两条指令的功能	分支指令的操作增多

分支比较类型



小于或等于分支占主导地位。

分支目标地址的表示

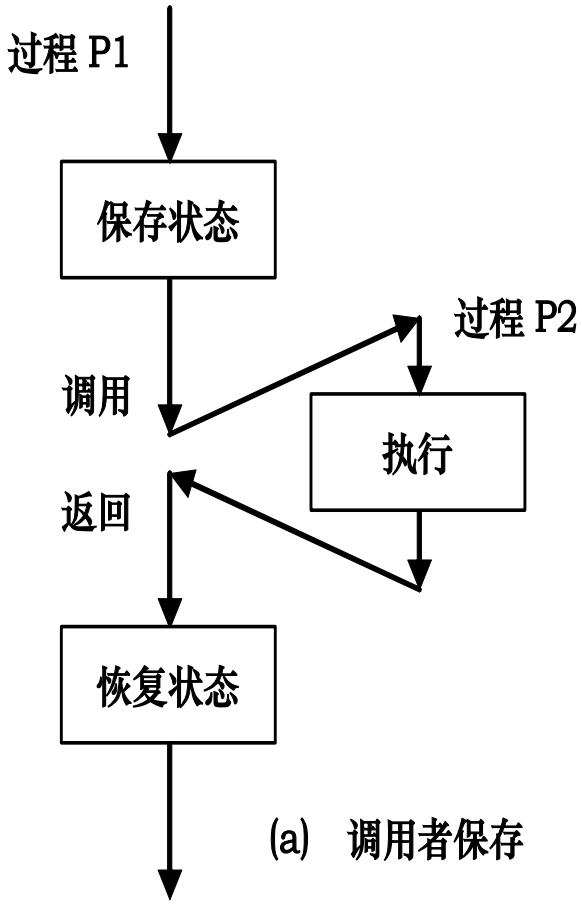


– 采用4～8位的偏移量字段（以指令字为单位）就能表示大多数控制指令的转移目标地址了

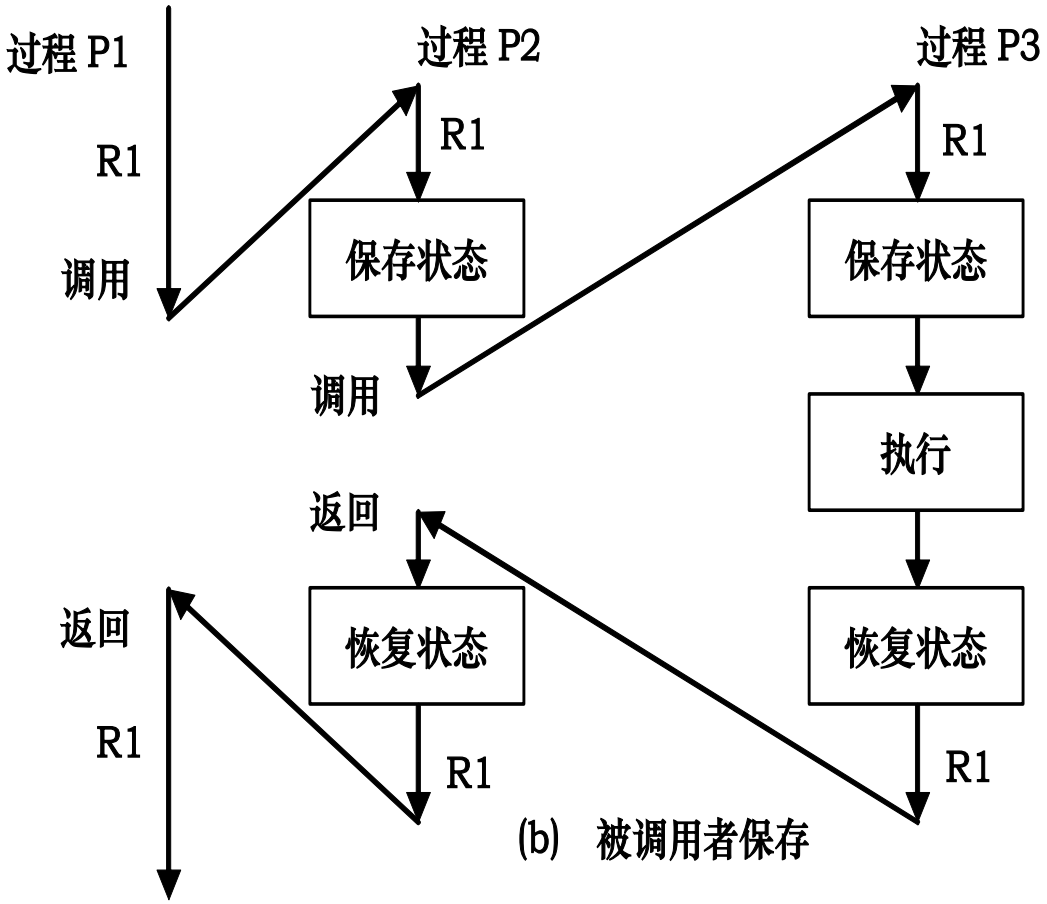
过程调用和返回的状态保存

- “调用者保存”（caller saving）方法：如果采用调用者保存策略，那么在一个调用者调用别的过程时，必须保存调用者所要保存的寄存器，以备调用结束返回后，能够再次访问调用者。
- “被调用者保存”（callee saving）方法：如果采用被调用者保存策略，那么被调用的过程必须保存它要用的寄存器，保证不会破坏过程调用者的程序执行环境，并在过程调用结束返回时，恢复这些寄存器的内容。

两种保存策略的比较



(a) 调用者保存



(b) 被调用者保存

2.5 指令系统的设计和优化

2.5.1 指令系统设计的基本原则

2.5.2 控制指令

2.5.3 指令操作码的优化

2.5.3 指令操作码的优化

- 指令由两部分组成：操作码、地址码
- 指令格式的设计

  确定指令字的编码方式，包括操作码字段和地址码字段的编码和表示方式。

- 指令格式的优化：如何用最短的位数来表示指令的操作信息和地址信息。

2.5.3 指令操作码的优化

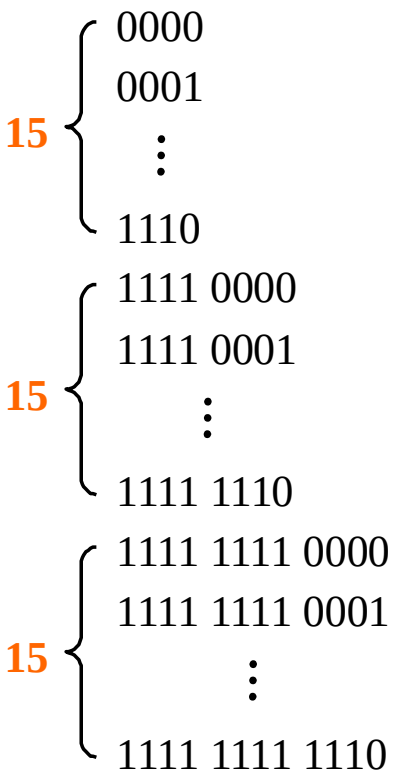
1. 等长扩展码

– 为了便于分级译码，一般都采用等长扩展码。（在早期的计算机上）

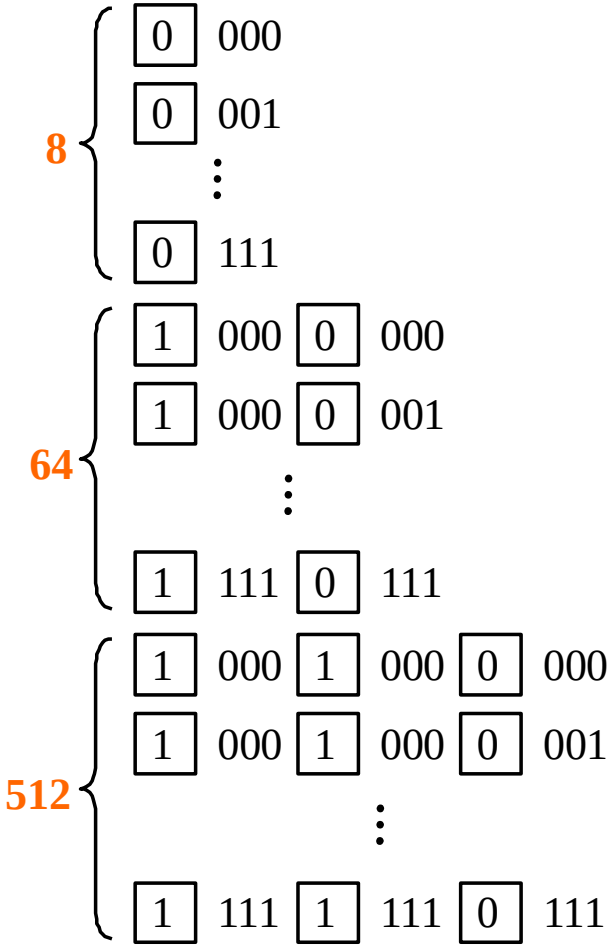
例如：15/15/15法和8/64/512法

- 选用哪种编码法取决于指令使用频度p_i的分布。若在头15种指令中p_i的值都比较大，但在后30种指令后急剧减少，则应选择15/15/15法；若p_i的值在头8种指令中较大，之后的64种指令的p_i值也不太低，则应选择8/64/512法。
- 衡量标准：看哪种编码法能使平均码长最短。

2.5.3 指令操作码的优化



15/15/15 编码法



8/64/512 编码法

2.5.3 指令操作码的优化

2. 定长操作码

- 固定长度的操作码：所有指令的操作码都是同一的长度（如8位）。

  许多计算机都采用（特别是RISC结构的计算机）

- 保证操作码的译码速度、减少译码的复杂度。
- 以程序的存储空间为代价来换取硬件实现上的好处。

2.5.3 指令操作码的优化

例题：某机器的指令字长为16位，设有单地址指令和二地址指令。若每个地址字段均为5位，且二地址指令有A条，问单地址指令最多可以有多少条？

解：

二地址指令的操作码为6位，有A条指令

单地址指令可以用作操作码的位数为11位

单地址指令最多可以有$2^{11} - \mathrm{A} \times 2^5$

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

2.6 指令系统的发展和改进

- 一个方向是强化指令功能，实现软件功能向硬件功能转移，基于这种指令集结构而设计实现的计算机系统称为复杂指令集计算机（CISC）。
- 八十年代发展起来的精简指令集计算机（RISC），其目的是尽可能地降低指令集结构的复杂性，以达到简化实现，提高性能的目的。

CISC指令集功能设计

1 面向目标程序增强指令功能

- 对大量的目标程序及其执行情况进行统计分析，找出那些使用频度高、执行时间长的指令或指令串。对于使用频度高的指令，用硬件加快其执行；对于使用频度高的指令串，用一条新的指令来替代。
- 既能减少目标程序的执行时间，也能有效地缩短程序的长度。
- 可以从以下几个方面来改进：

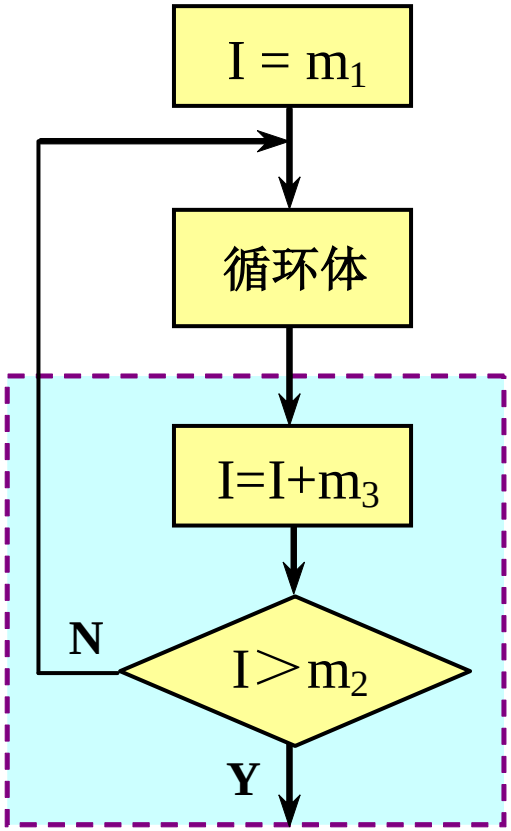
  (1) 增强运算型指令的功能

  (2) 增强数据传送指令的功能

  (3) 增强程序控制指令的功能

例如：循环在程序中占有相当大的比例，所以在指令上提供专门的支持。

- 循环控制占有较大比例
- 循环控制部分通常用3条指令完成：
  - 一条加法指令
  - 一条比较指令
  - 一条分支指令
- 设置循环控制指令，用一条指令完成上述3条指令的功能。



一般循环程序的结构

CISC指令集功能设计

2 面向高级语言的优化实现来改进指令系统

（缩小高级语言与机器语言的语义差距）

高级语言与一般的机器语言的语义差距非常大，为高级语言程序的编译带来了一些问题。

- ✓ 编译器本身比较复杂；
- ✓ 编译生成的目标代码比较难以达到很好的优化。

CISC指令集功能设计

(1) 增强对高级语言和编译器的支持

- ✓ 对高级语言中使用频度高、执行时间长的语句，增强有关指令的功能，加快这些指令的执行速度，或者增加专门的指令，可以达到减少目标程序的执行时间和减少目标程序长度的目的。
- ✓ 增强系统结构的规整性，减少系统结构中的各种例外情况。

CISC指令集功能设计

(2) 高级语言计算机

① 间接执行高级语言机器

高级语言作为机器的汇编语言。

② 直接执行高级语言的机器

直接把高级语言作为机器语言。

（一种比较激进的方法）

采用“比较简单的系统结构＋软件”的做法能够在较低成本和复杂度的前提下，提供更高的性能和灵活性。

CISC指令集功能设计

3 面向操作系统的优化实现改进指令系统

(1) 处理机工作状态和访问方式的切换；

(2) 进程的管理和切换；

(3) 存储管理和信息保护；

(4) 进程的同步与互斥，信号的管理等。

RISC指令集功能设计

- CISC结构存在着如下缺点：
  1. 在CISC结构的指令系统中，各种指令的使用频率相差悬殊。据统计，有20％的指令使用频率最大，占运行时间的80％。也就是说，有80％的指令在20％的运行时间内才会用到。

RISC指令集功能设计

2. CISC结构指令系统的复杂性带来了计算机体系结构的复杂性。大量占用芯片面积，给VLSI设计造成很大困难。这不仅增加了研制时间和成本，而且还容易造成设计错误。
3. CISC结构的指令系统中，许多复杂指令需要很复杂的操作，因而运行速度慢。
4. 在CISC结构的指令系统中，由于各条指令的功能不均衡性，不利于采用先进的计算机体系结构技术（如流水技术）来提高系统的性能。

RISC指令集功能设计

执行频率排序	80X86指令	指令执行频率
1	Load	22%
2	条件分支	20%
3	比较	16%
4	Store	12%
5	加	8%
6	与	6%
7	减	5%
8	寄存器－寄存器间数据移动	4%
9	调用	1%
10	返回	1%
合　计		95%

RISC指令集功能设计

- 进行RISC计算机指令集结构的功能设计时，我们并不能简单地着眼于精简指令系统上，更重要的目的是使得计算机体系结构更加简单、更加合理和更加有效，克服CISC结构的缺点，使机器速度更快，程序运行时间缩短，从而提高计算机系统的性能。

RISC指令集功能设计原则

- 选取使用频率最高的指令，并补充一些最有用的指令；
- 每条指令的功能应尽可能简单，并在一个机器周期内完成（采用流水线技术后）；
- 所有指令长度均相同；
- 只有load和store操作指令才访问存储器，其它指令操作均在寄存器之间进行；
- 大多数指令都采用硬连线技术；
- 以简单有效的方式支持高级语言；
- 充分利用流水线技术。

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

四大主流ISA

序号	架构	特点	代表性厂商	运营机构	发明时间
1	x86	性能高、速度快、兼容性好、实用性强	Intel、AMD	Intel	1978
2	ARM	成本低、低功耗	高通、Nivda、Pythium、Huawei	ARM	1983
3	RISC-V	完全开源、架构简单、模块化、极简、可扩展	Samsung、阿里	RISC-V	2014
4	MIPS	简洁、优化方便包含大量寄存器、高拓展性	Loongson、MIPS	MIPS	1981

Intel 8086

(1) 指令字长　1 ~ 6 个字节

INC　AX　　1 字节

MOV　WORD PTR[0204], 0138H　　6 字节

(2) 地址格式

零地址	NOP		1 字节	
一地址	CALL	段间调用	5 字节	
	CALL	段内调用	3 字节	
二地址	ADD AX，BX		2 字节	寄存器 – 寄存器
	ADD AX，3048H		3 字节	寄存器 – 立即数
	ADD AX，[3048H]		4 字节	寄存器 – 存储器

2.7 MIPS指令分析

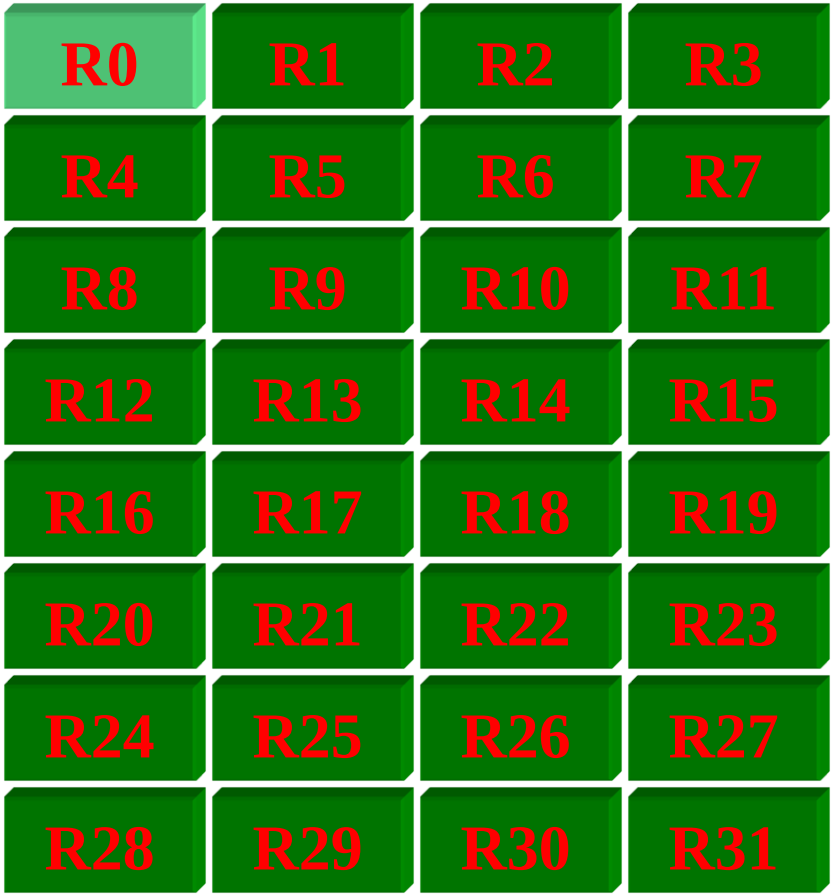
介绍MIPS32的一个子集，简称为MIPS。

- Load/Store型指令集结构
- MIPS是一种多元指令集结构
  - 体现了当今多种机器（AMD29K、DEC station 3100、HP850、IBM 801、MIPS M/120A、MIPS M/1000、Motorola 88k、RISC I、SGI4D/60、SPARC station 1、Sun 4/110、Sun 4/260等）的指令集结构的共同特点。
  - 还将会体现未来一些机器的指令集结构的特点。

MIPS指令集结构

- 具有一个简单的Load/Store指令集；
- 注重指令流水效率；
- 简化指令的译码；
- 高效支持编译器。

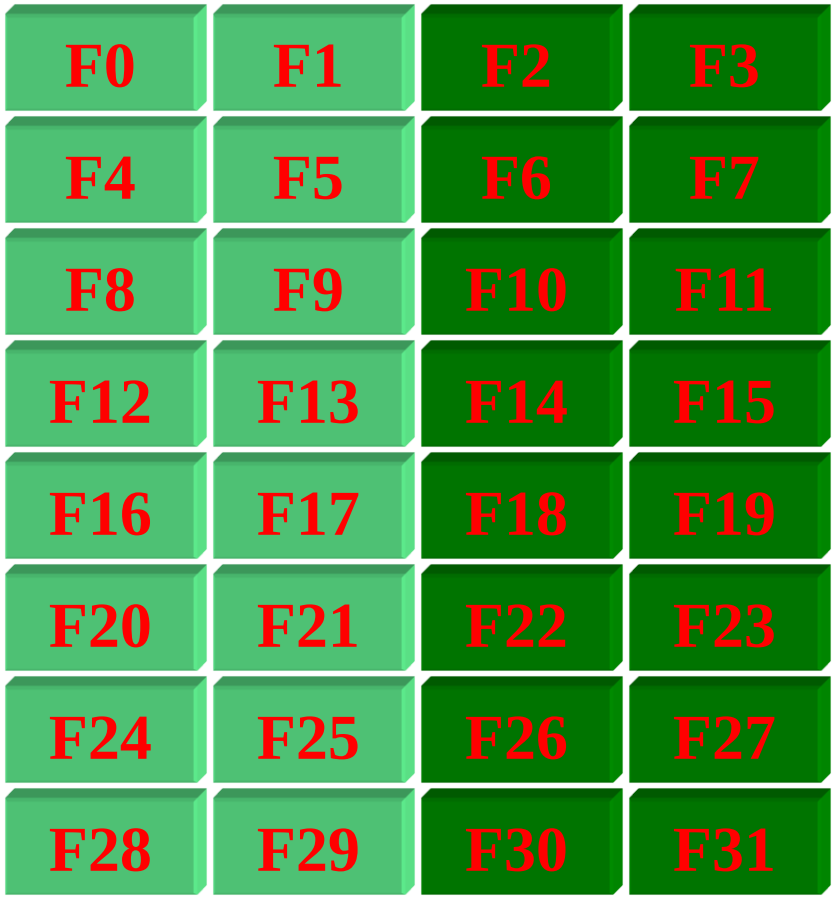
MIPS指令集结构：寄存器



32个32位的通用寄存器（GPRs）。

寄存器R0的内容恒为全0。

MIPS指令集结构：寄存器



32个32位浮点寄存器（FPRs）。

单精度浮点数表示和双精度浮点数表示。

MIPS指令集结构：数据类型

- 整型数据：
  - 8位、16位、32位。
- 浮点数据：
  - 32位单精度浮点；
  - 64位双精度浮点；
  - IEEE 754标准。

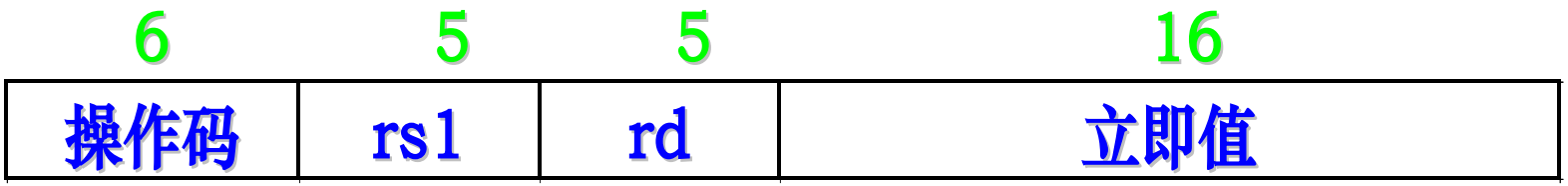
00000000	00000000	00000000	01100100
11111111	11111111	10011000	01100100

MIPS指令集结构：寻址方式

- 寄存器寻址；如 **ADD R1, R2, R3**
- 立即值寻址；如 **ADD R1, R2, #42**
- 偏移寻址； 如 **LW R2, 40(R3)**
  - 寄存器间接寻址； 如 **LW R2, 0(R3)**
  - 直接寻址；如 **LW R2, 40(R0)**

MIPS指令集结构：指令格式

I 类型指令



字节、半字、字的载入和存储；

rd ←rs1　op　立即值。

MIPS指令集结构：指令格式

R 类型指令

寄存器－寄存器 ALU 操作：rd←rs1 func rs2；
函数对数据的操作进行编码：加、减、…；
对特殊寄存器的读/写和移动。

MIPS指令集结构：指令格式

J 类型指令

跳转，跳转并链接，从异常（exception）处自陷和返回。

MIPS指令集结构：操作类型

- Load和Store操作；
- ALU操作；
- 分支和跳转操作；
- 浮点操作。

MIPS指令集结构：操作类型

- 符号“←”表示数据传送操作，其后附带一个下标n，也即“←n” 表示传送一个n位数据。
- 符号“##”用来表示两个域的串联操作，它可以出现在数据传送操作的任何一边。

MIPS指令集结构：操作类型

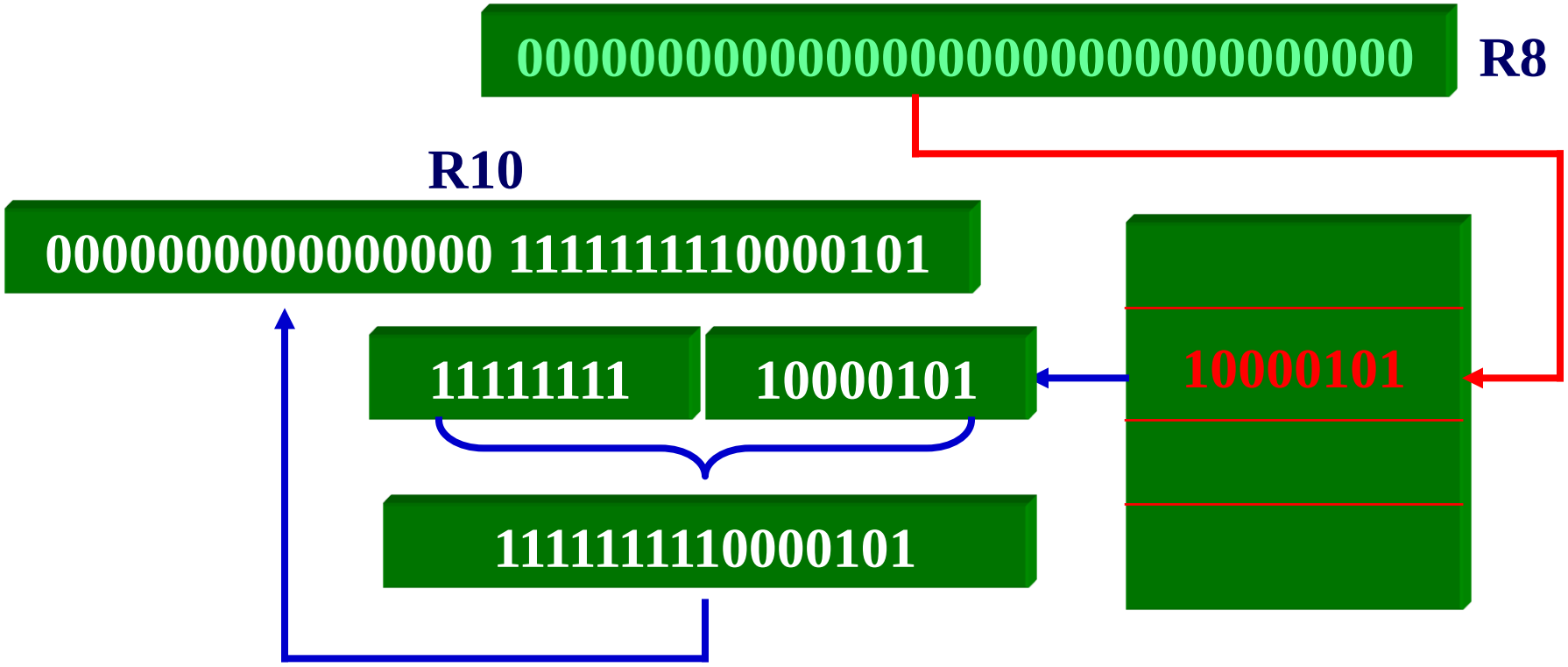
- 域的下标用来表明从该域中选择某一位。域中位的标记是从最高位开始标记，并且起始标记为0。下标可以是一个单独的数字，如$Regs[R4]_0$表示选择寄存器R4中内容的符号位；下标也可以是一个范围，如$Regs[R3]_{24..31}$表示选择寄存器R3中内容的最低一个字节。

MIPS指令集结构：操作类型

- 上标表示复制一个域，如0^{24}可以得到一个24位全为0的一个域。
- 变量Mem用来表示存储器中的一个数组，存储器按照字节寻址，它可以传送任何数目的字节。

MIPS指令集结构：操作类型

$$Regs[R10]_{16..31} \leftarrow_{16} (Mem[Regs[R8]]_0)^8 \,\#\#\, Mem[Regs[R8]]$$



MIPS指令集结构：Load和Store

- **Load和Store操作**：可以对MIPS的所有通用寄存器和浮点寄存器进行Load（载入）和Store（储存）操作，但是对通用寄存器R0的Load操作没有任何效果。

指令举例	指令名称	含 义
LW R2，40(R3)	装入字	Regs[R2]$\leftarrow_{32}$Mem[40+Regs[R3]]
LB R2，30(R3)	装入字节	Regs[R2]$\leftarrow_{32}$ (Mem[30+Regs[R3]]$_0$)^{24} ## Mem[30+Regs[R3]]
LBU R2，40(R3)	装入无符号字节	Regs[R2]$\leftarrow_{32}$ 0^{24} ## Mem[40+Regs[R3]]
LH R2，30(R3)	装入半字	Regs[R2]$\leftarrow_{32}$ (Mem[30+Regs[R3]]$_0$)^{16} ## Mem[30+Regs[R3]]## Mem[31+Regs[R3]]
LS F2，60(R4)	装入单精度浮点	Regs[F2]$\leftarrow_{32}$ Mem[60+Regs[R4]]
SW 300(R5)，R4	保存字	Mem[300+Regs[R5]]$\leftarrow_{32}$ Regs[R4]
SH 502(R4)，R5	保存半字	Mem[502+Regs[R4]]$\leftarrow_{16}$ Regs[R5]$_{16..31}$
SB 41(R3)，R2	保存字节	Mem[41+Regs[R3]]$\leftarrow_{8}$ Regs[R2]$_{24..31}$
SS 40(R2)，F2	保存单精度浮点数	Mem[40+Regs[R2]]$\leftarrow_{32}$ Regs[F2]

MIPS指令集结构：ALU操作

- **ALU操作**：在MIPS中，所有的ALU指令都是寄存器—寄存器型指令，其运算包含了简单的算术和逻辑运算，如加、减、AND、OR、XOR和移位。
- “设置相等”、“设置不等”、“设置小于”：寄存器比较指令（=，≠，<，>，≤，≥），如果比较结果为真，这些指令就在目标寄存器中填入1（表示真），否则填入0（表示假）。

MIPS指令集结构：ALU操作

指令实例	指令名称	含 义
ADD R1,R2,R3	加	Regs[R1] ← Regs[R2] + Regs[R3]
ADDI R1,R2,#3	和立即值相加	Regs[R1] ← Regs[R2] + 3
LHI R1,#42	载入高位立即值	Regs[R1] ← 42 ## 0^{16}
SLLI R1,R2,#5	逻辑左移立即值形式	Regs[R1] ← Regs[R2] <<5
SLT R1,R2,R3	设置小于	if (Regs[R2] <Regs[R3]) Regs[R1] ← 1 else Regs[R1] ← 0

MIPS指令集结构：转移操作

- 描述目标地址的方法：
  - 用带符号位的26位偏移量加上程序计数器的值来确定跳转的目标地址；
  - 指定一个寄存器，由寄存器中的内容决定跳转的目标地址。

MIPS指令集结构：转移操作

- 两种跳转类型：
  - 一种是简单跳转；
  - 另一种是跳转并链接（用于过程调用），它将下一条顺序指令地址（返回地址）保存在寄存器R31中。

MIPS指令集结构：转移操作

指令实例	指令名称	含义
J name	跳转	PC ←name+PC+4; $-2^{25} \le$ name $\le 2^{25}$
JAL name	跳转并链接	Regs[R31] ← PC+4; PC ←name +PC+4; $-2^{25} \le$ name $\le 2^{25}$
JR R3	寄存器型跳转	PC ← Regs[R3];
JALR R2	寄存器型跳转并链接	Regs[R31]←PC+4; PC← Regs[R2];
BEQZ R4 , name	“等于0”分支	if (Regs[R4]==0) PC ← name+ PC+4; $-2^{15} \le$ name $\le 2^{15}$
BNQZ R4 , name	“不等于0”分支	if (Regs[R4]!=0) PC ← name+ PC+4; $-2^{15} \le$ name $\le 2^{15}$

MIPS指令集结构：浮点操作

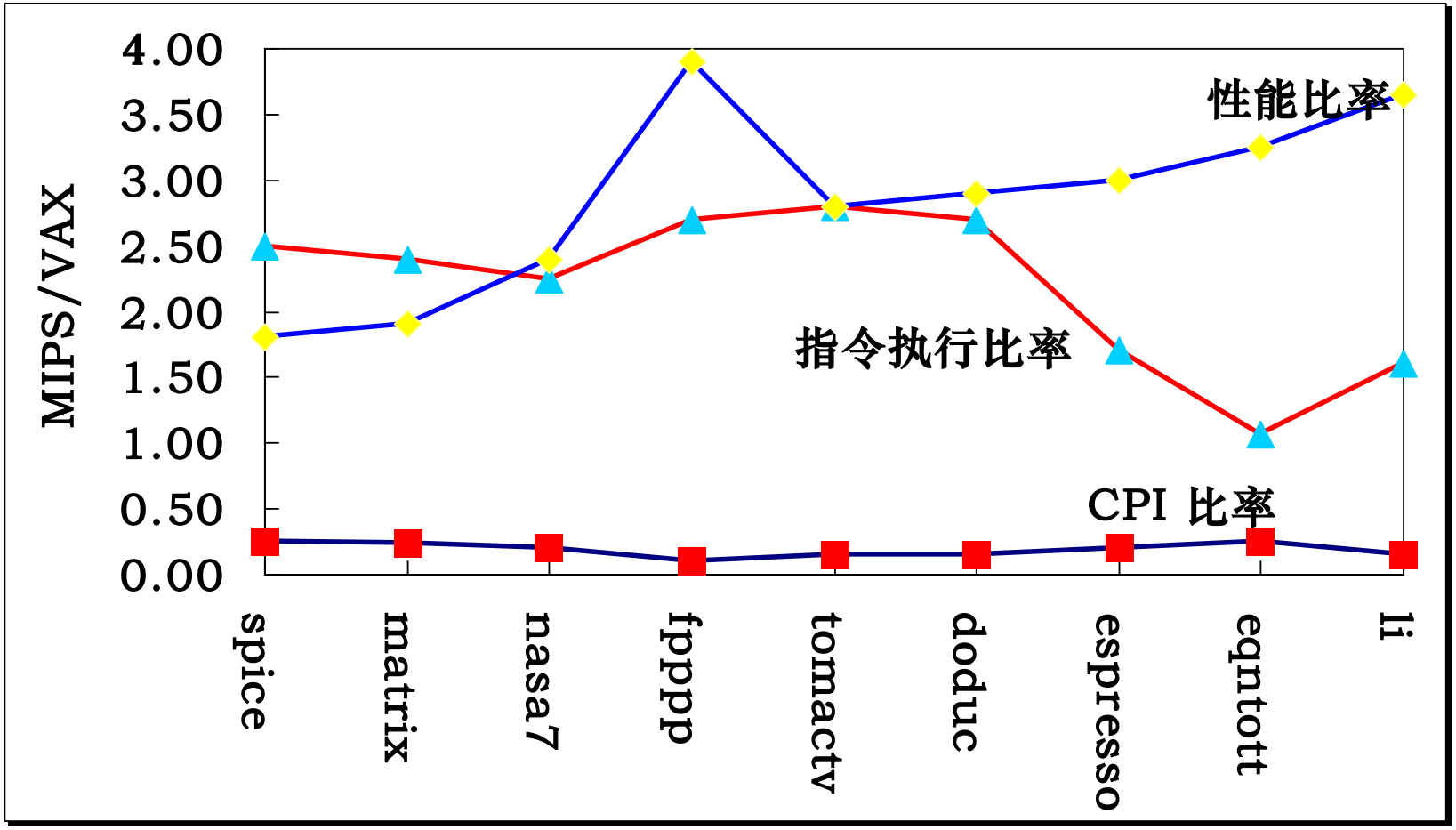
- 浮点操作：浮点指令的操作数来源于浮点寄存器，同时它还指明了相应的操作是单精度浮点操作还是双精度浮点操作。
  - 后缀D代表双精度浮点操作
  - 而后缀S代表单精度浮点操作
  - 如：ADDD、ADDS、SUBD、SUBS、MULTD、MULTS、DIVD、DIVS。
- MIPS的浮点操作有：加、减、乘、除。

MIPS的效能分析

- 问题的提出：
  - MIPS指令集结构的指令格式、寻址方式和操作都非常简单。也许有人会担心，这些特性会使得目标代码中指令条数增多，导致程序运行时间加长，从而使这种指令集结构的机器性能并不会太高。

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

MIPS的效能分析


MIPS/VAX
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
性能比率
指令执行比率
CPI 比率
spice
matrix
nasa7
fpppp
tomactv
doduc
espresso
eqntott
li

ARM

1 基本格式 <opcode>{<cond>}{S}<Rd><Rn><operand2>

<> 为必选项，{}为可选项
{} 决定指令执行条件域
{S} 决定指令执行是否影响当前程序状态寄存器CPSR的值

2 指令字长32位为例

cond	00	X	opcode	S	Rn	R_d	Shifter-operand
4	2	1	4	2	4	4	12

opcode：指令操作码
cond：指令执行条件
S：指令的操作是否影响CPSR的值
Rn：第一个操作数的寄存器地址
Rd：目标寄存器地址
Shifter_operand:第二个操作数

RISC-V

1 指令集组成

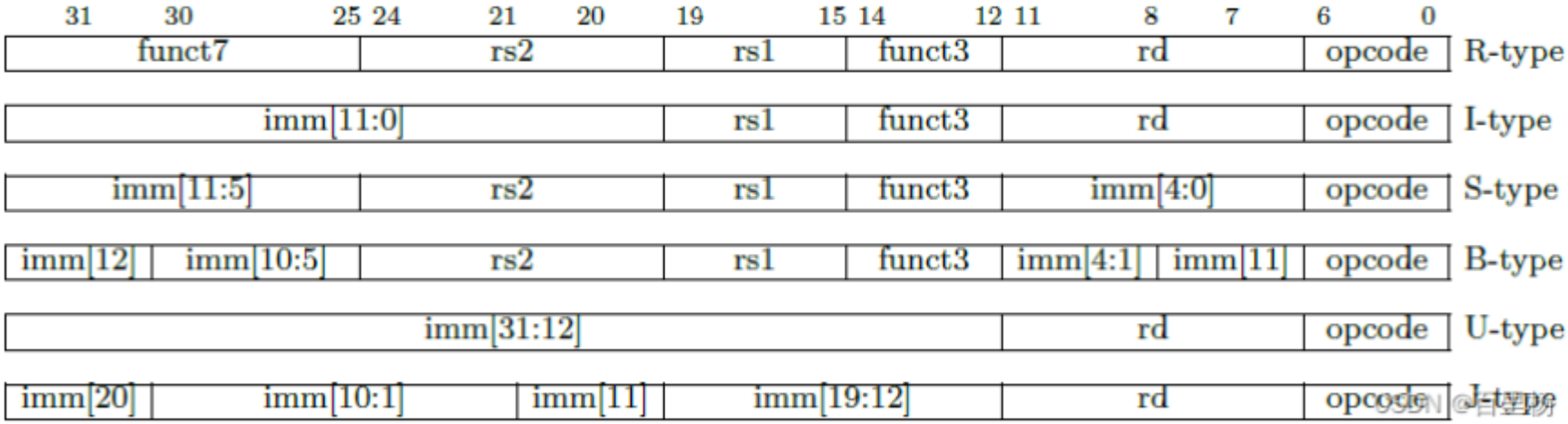
一个基础整数指令集为核心，根据需要再选择多个扩展指令集

G = I + M + A + D + F

I：基础整数指令集，M：乘法和除法指令，A：原子指令

F：当精度浮点，D：双精度浮点

2 指令字长32位为例



R型：寄存器间操作，I型：短立即数和取数，S型：存数

B型：条件分支跳转，U型：长立即数，J型：无条件跳转

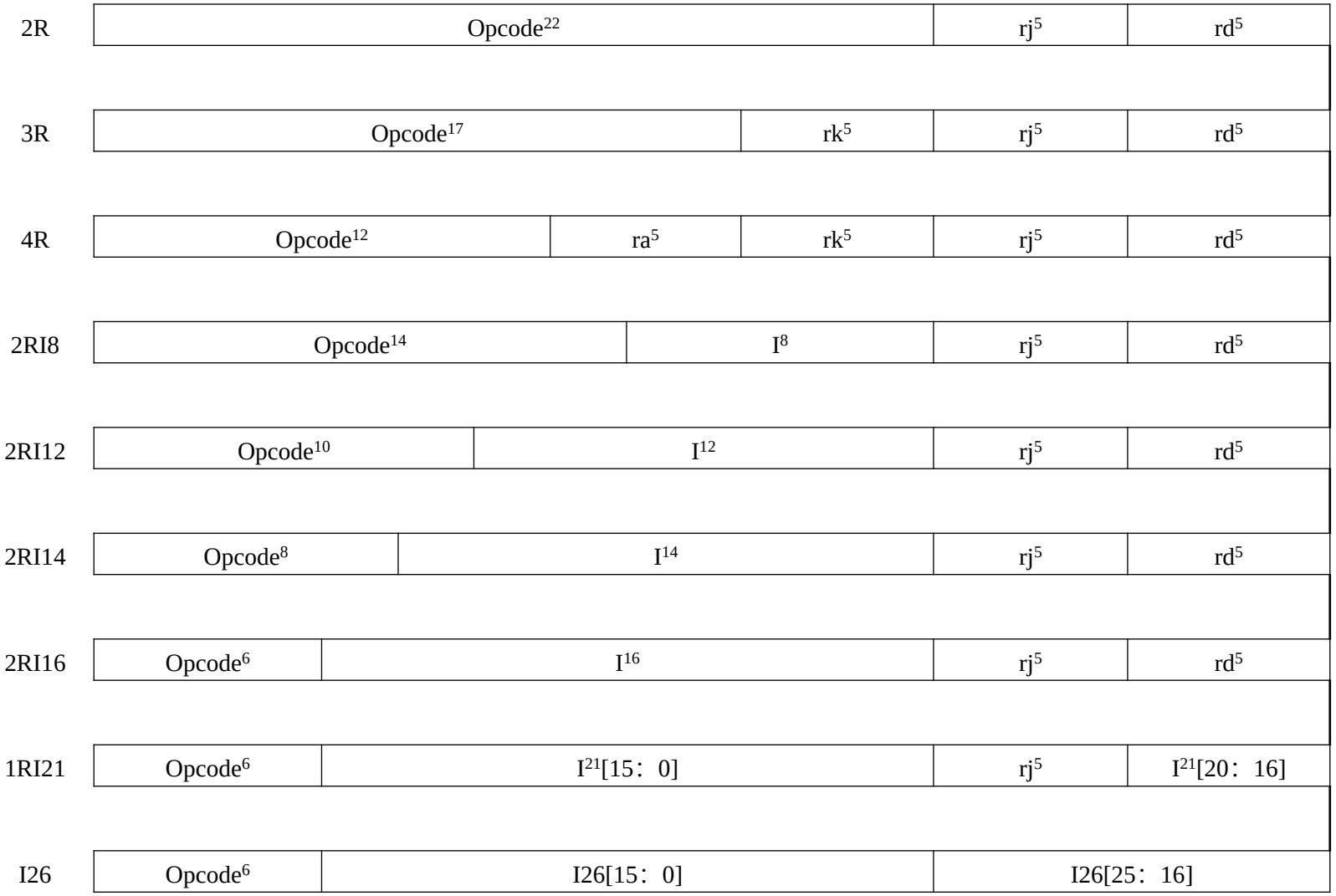
LoongArch指令系统

- **LoongArch是中国龙芯中科自行研发的指令集架构，其目的在于构建起自主可控的信息技术体系，此架构在设计过程中充分考量了国产软硬件生态的兼容性以及优化性，有简洁、高效且可扩展等特性。**
- **LoongArch指令集体系结构依据功能被划分成基础指令集以及多个扩展指令集，其中用于教学一般是LoongArch 32位精简版架构，也就是LA32龙芯基础指令集。**

LoongArch体系结构中的寄存器

寄存器号	助记符	Saver
r0	zero	常量寄存器，值恒为1
r1	ra	函数返回地址（Return Address）
r2	tp	用于支持TLS（Thread-Local Storage）
r3	sp	栈指针（Stack Pointer）
r4-r11	a0-a7 v0/v1=a0/a1	参数寄存器（Augument） V0-V1做函数返回值
r12-r20	t0-t8	临时寄存器
r21	reserved	保留寄存器
r22	fp	帧指针（Frame pointer）
r23-r31	s0-s8	保存寄存器（saved）

LoongArch指令格式



LoongArch寻址方式

寻址方式	指令示例	格式说明
寄存器寻址	**ADD R1， R2， R3**	**regs[R3] = regs[R1] + regs[R2]**
立即数寻址	**ADD R1， R2， #2**	**regs[R1] = regs[R2] + 2**
基址 + 立即数偏移寻址	**LD R1， R2， #100**	**regs[R1] = mem[regs[R2] +100]**
基址 + 寄存器偏移寻址	**LD R1， R2， R3**	**regs[R1] = mem[reg[R2] + regs[R3]]**
相对寻址	**BL #100**	**PC = mem[PC + 100]**

LoongArch典型指令

- 第一类为运算指令，包括算数运算、逻辑运算以及移位指令等。其中算数运算包括加减乘除以及模运算，并且根据指令后是否含u分别表示把运算的操作数看作无符号还是有符号数。逻辑类指令则主要是指与或非以及异或运算；而移位指令则指算数移位或逻辑移位，其中逻辑移位有左右移，而算术移位则只有右移。
- 第二类为数据传送类，也称为访存指令，负责对存储器的读写。根据操作的数据字长分为字节（b）、16位的半字（h）和32位的字（w）；由于读出的数据需要扩展后放到32位的寄存器里，故分为做零扩展的无符号数读指令和做符号扩展的读指令，前者通过带u来区别；而写操作因为要写入内存没有扩展问题而没有上述问题。

LoongArch典型指令

- 第三类是转移指令，用于控制程序的流向。主要分为条件转移指令和无条件转移类指令，过程调用和过程返回等类型。转移条件和转移目标地址是转移指令的两个要素， 两者的组合构成了不同的转移指令：条件转移要判断条件再决定是否转移， 无条件转移则无须判断条件；
- 第四类是特殊指令，用于操作系统的特定用途。这里主要展示的是系统调用指令**syscall**和**break**指令，前者可以让用户通过此入口进入到系统核心态完成系统开放的功能，后者则可以随时中断程序的执行。

本章小结

- 指令系统的基本概念
- 指令系统结构的分类
  - 堆栈型、累加器型、通用寄存器型
  - 根据ALU操作数来源，通用寄存器型包括：寄存器-存储器型和寄存器-寄存器型
- 寻址方式
  - 操作数的寻址
  - 偏移寻址与立即寻址的范围

本章小结

- 操作数的分类和大小
- 指令系统设计的设计和优化
  - 基本原则（完整性、规整性、正交性、高效率、兼容性）
  - 控制指令、操作码的优化、指令集编码
- 指令系统的发展和优化
- **MIPS指令集**
- **ARM、RISC-V、LoongArch指令集**

第二章作业

- 王志英教材，P62，T12，T15（张春元教材，P71， T12，T15)

- 阅读以下论文，结合第二章的课程内容完成有关现代指令系统发展的报告（3000字以内）：

[1] Charles E. Gimarc and Veljko M. Milutinovic. A Survey of RISC Processors and Computers of the Mid-1980s, Computer, 1987.

[2] 舒燕君, 郑翔宇, 徐成华等. 面向 LoongArch 边界检查访存指令的 GCC 优化实现, 计算机研究与发展，2025.

第二章作业

- 和第一章作业一起交（11月12日下午3点-5点）
- 需要手写
- 地点：综合楼514