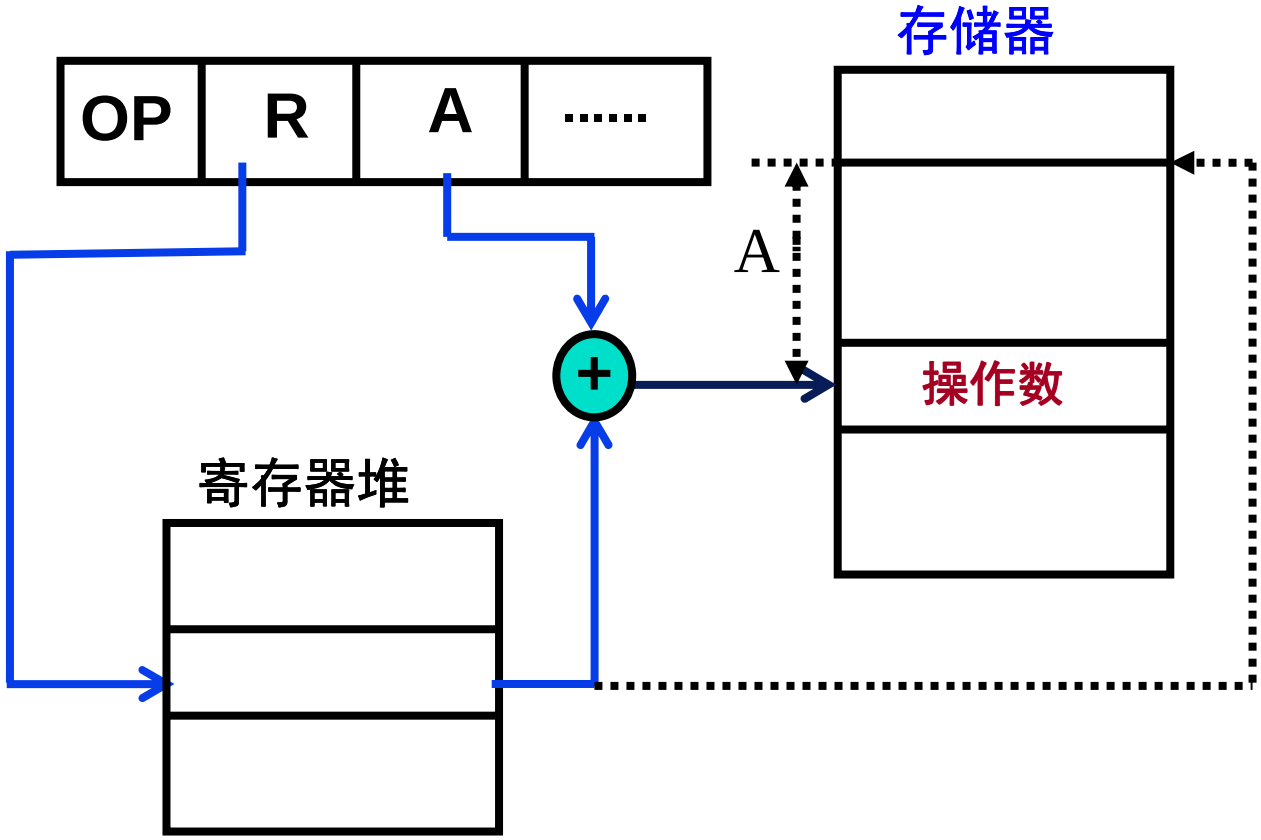
Lecture 18: Instruction Set II

偏移寻址方式

指令中给出的地址码A称为形式地址



偏移寻址：EA=A+(R)　R可以明显给出，也可以隐含给出

R可以为PC、基址寄存器B、变址寄存器I

- 相对寻址：　EA=A+(PC)　　相对于当前指令处位移量为A的单元
- 基址寻址：　EA=A+(B)　　相对于基址(B)处位移量为A的单元
- 变址寻址：　EA=A+(I)　　相对于首址A处位移量为(I)的单元

偏移寻址方式

- 相对寻址

  指令地址码给出一个偏移量(带符号数)，基准地址隐含由PC给出。

  即：EA=(PC)+A　　（ex. MIPS's instruction：Beq）

  注意：当前PC的值可能是正在执行指令的地址或下条指令的地址

- 基址寻址

  指令地址码给出一个偏移量，基准地址明显或隐含由基址寄存器B给出。
  即：EA=(B)+A　（ex. MIPS's instructions：lw / sw）

- 变址寻址

  指令地址码给出一个基准地址，而偏移量(无符号数)明显或隐含由变址寄存器I给出。即：EA=(I)+A

Addressing Modes（寻址方式的汇编表示）

Addressing mode	Example	Meaning
Immediate	Add R4,3	R4 ← R4+3
Register	Add R4,R3	R4 ← R4+R3
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Indexed	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,–(R2)	R2 ← R2– d; R1← R1+Mem[R2]
Scaled(乘比例因子)	Add R1,100(R2)[R3]	R1 ← R1+Mem[100+R2+R3*d]

为简化表示，“←”右部的Ri表示寄存器Ri中的内容

上述形式是一种示意性表示，不同系列处理器的汇编表示形式不同！

Instruction Format(指令格式)

- 操作码的编码有两种方式
  - Fixed Length Opcodes (定长操作码法)
  - Expanding Opcodes (扩展操作码编法)
- instructions size
  - 代码长度更重要时：采用变长指令字、变长操作码
  - 性能更重要时：采用定长指令字、定长操作码

为什么？

变长指令字和操作码使机器代码更紧凑；定长指令字和操作码便于快速访问和译码。学了CPU设计就更明白了。

定长操作码编码Fixed Length Opcodes

- 基本思想

  指令的操作码部分采用固定长度的编码

  如：假设操作码固定为6位，则系统最多可表示64种指令

- 特点

  译码方便，但有信息冗余

- 举例

  IBM360/370采用:

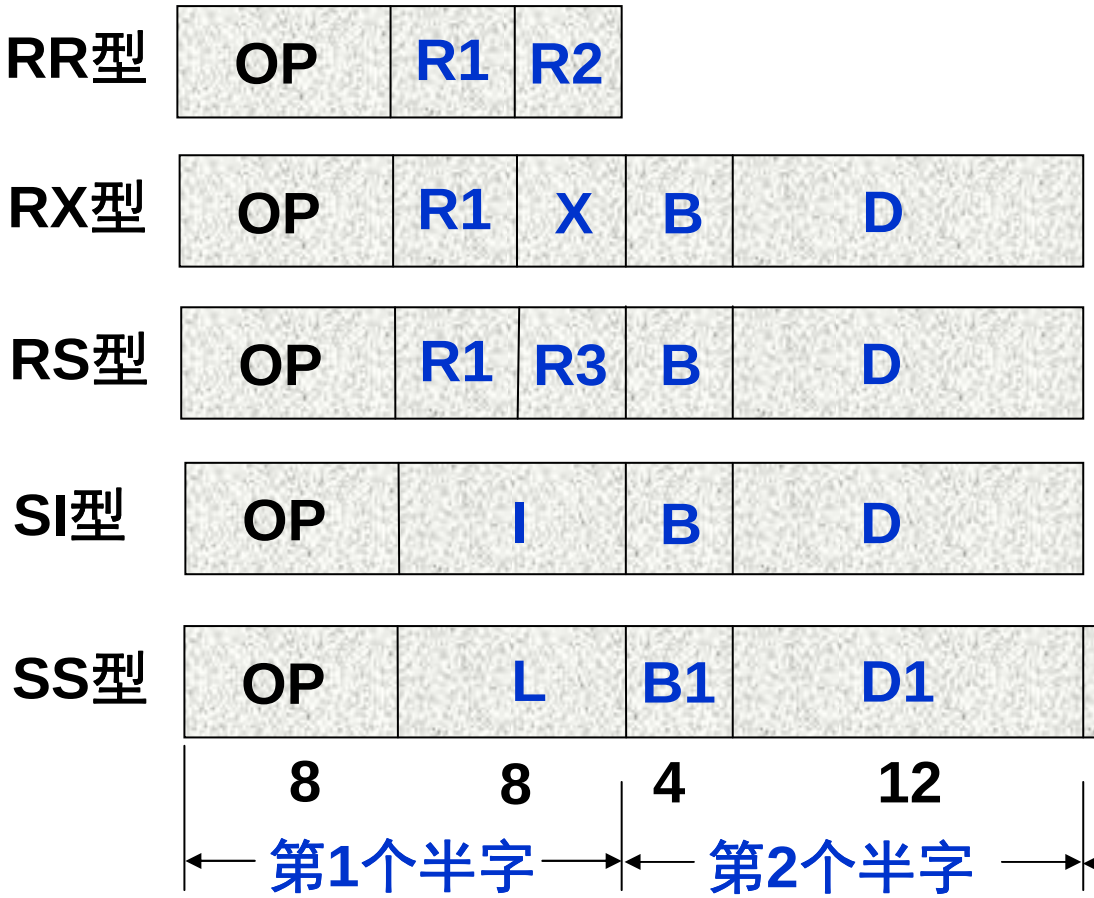
  ８位定长操作码，最多可有256条指令

  只提供了183条指令，有73种编码为冗余信息

  机器字长32位，按字节编址

  有16个32位通用寄存器，基址器B和变址器X可用其中任意一个

IBM370指令格式



Ri：寄存器

X：变址器

Bi：基址器

Di：位移量

I：立即数

L：数的长度

RR：寄存器 - 寄存器

RX：寄存器 - 变址存储器

RS：寄存器 – 基址存储器

SS：基址存储器 - 基址存储器

SI：基址存储器 - 立即数

扩展操作码编码Expanding Opcodes

基本思想

将操作码的编码长度分成几种固定长的格式。被大多数指令集采用。
PDP-11是典型的变长操作码机器。

种类

等长扩展法：4-8-12；3-6-9；…… / 不等长扩展法

PDP-11中典型指令格式

采用专门的寻址方式字段

S、D：3位指定寻址方式，3位为寄存器编号

R：8个通用寄存器之一

FR：4个浮点寄存器之一

X：位移

Methods of Testing Condition (条件测试方式)

- 条件转移指令通常根据Condition Codes (条件码 / 状态位 / 标志位)转移

  通过执行算术指令或显式地由比较和测试指令来设置

  ex: sub r1, r2, r3 ;r2和r3相减, 结果在r1中，并生成标志位ZF、CF等

  bz label ;标志位ZF=1时，转移到label处执行；否则顺序执行

- 常用的标志有四种（哪四种？）：

  NF(SF) - negative VF(OF) - overflow CF - carry ZF - zero

- 标志位可存放在标志(Flag)寄存器（条件码CC寄存器 / 状态Status寄存器 / 程序状态字PSW寄存器）中

  也可由指定的通用寄存器来存放状态位

  不同处理器对标志位的处理不同

  Ex: cmp r1, r2, r3 ;比较r2和r3, 标志位存储在r1中

  bgt r1, label ;判断r1是否大于0，是则转移到label处

- 可以将两条指令合成一条指令，即：计算并转移

  Ex: bgt r1, r2, label ;如果r1>r2，则转移到label处执行；否则顺序执行

指令设计风格 -- 按操作数位置指定风格来分

Accumulator: (earliest machines) 累加器型

特点：其中一个操作数（源操作数1）和目的操作数总在累加器中

1 address	add A	acc ← acc + mem[A]
1(+x) address	add x A	acc ← acc + mem[A + x]

Stack: (e.g. HP calculator, Java virtual machines) 堆栈型

特点：总是将栈顶两个操作数进行运算，指令无需指定操作数地址

0 address	add	tos ← tos + next

General Purpose Register: (e.g. IA-32, Motorola 68xxx) 通用寄存器型

特点：操作数可以是寄存器或存储器数据（即A、B和C可以是寄存器或存储单元）

2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)

Load/Store: (e.g. SPARC, MIPS, PowerPC) 装入/存储型

特点：运算操作数只能是寄存器数据，只有load/store能访问存储器

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

Comparing Instructions

Comparison:

Bytes per instruction? Number of Instructions? Cycles per instruction?

- **Code sequence for C = A + B for four classes of instruction sets:**

Stack	Accumulator	Register (register- memory)	Register (load - store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C, R1	Add R3,R1,R2
Pop C			Store C,R3

指令条数较少

复杂表达式时，累加器型风格指令条数变多，因为所有运算都要用累加器，使得程序中多出许多移入 / 移出累加器的指令！

75年开始，寄存器型占主导地位 (Java Virtual Machine 采用Stack型)

- 寄存器速度快，使用大量通用寄存器可减少访存操作
- 表达式编译时与顺序无关（相对于Stack）

Examples of Register Usage

每条典型ALU指令中的存储器地址个数

每条典型ALU指令中的最多操作数个数

		Examples
0	3	SPARC, MIPS, Precision Architecture, Power PC
1	2	Intel 80x86, Motorola 68000
2	2	VAX (also has 3-operand formats)
3	3	VAX (also has 2-operand formats)

In VAX(CISC): ADDL (R9), (R10), (R11) 一条指令！

; mem[R9] ←mem[R10] + mem[R11]

In MIPS(RISC):

```
lw   R1, (R10)     # R1 ← mem[R10]
lw   R2, (R11)     # R2 ← mem[R11]
add  R3, R1, R2    # R3 ← R1+R2
sw   R3, (R9)      # mem[R9] ← R3
```

四条指令！

哪一种风格更好呢？学了第6和第7章后会有更深的体会！

指令设计风格 – 按指令格式的复杂度来分

按指令格式的复杂度来分，有两种类型计算机：

复杂指令集计算机CISC (Complex Instruction Set Computer)

精简指令集计算机RISC (Reduce Instruction Set Computer)

早期CISC设计风格的主要特点

(1) 指令系统复杂

指令多 / 寻址方式多 / 指令格式多

(2) 指令周期长

绝大多数指令需要多个时钟周期才能完成

(3) 各种指令都能访问存储器

除了专门的存储器读写指令外，运算指令也能访问存储器

(4) 采用微程序控制

(5) 有专用寄存器

(6) 难以进行编译优化来生成高效目标代码

例如，VAX-11/780小型机

16种寻址方式；9种数据格式；303条指令；

一条指令包括1～2个字节的操作码和下续N个操作数说明符。

一个说明符的长度达1 ～10个字节。

复杂指令集计算机CISC

- CISC的缺陷
  - 日趋庞大的指令系统不但使计算机的研制周期变长，而且难以保证设计的正确性，难以调试和维护，并且因指令操作复杂而增加机器周期，从而降低了系统性能。
- 1975年IBM公司开始研究指令系统的合理性问题，John Cocke提出精简指令系统计算机 RISC (Reduce Instruction Set Computer)。
- 对CISC进行测试，发现一个事实：
  - 在程序中各种指令出现的频率悬殊很大，最常使用的是一些简单指令，这些指令占程序的80%，但只占指令系统的20%。而且在微程序控制的计算机中，占指令总数20%的复杂指令占用了控制存储器容量的80%。
- 1982年美国加州伯克利大学的RISC Ⅰ，斯坦福大学的MIPS，IBM公司的IBM801相继宣告完成，这些机器被称为第一代RISC机。

John Cocke & RISC

"We knew we wanted a computer with a simple architecture and a set of simple instructions that could be executed in a single machine cycle—making the resulting machine significantly more efficient than possible with other, more complex computer designs."

-John Cocke, 1987

Top 10 80x86 Instructions

Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

- Simple instructions dominate instruction frequency

（简单指令占主要部分，使用频率高！）

RISC设计风格的主要特点

(1) 简化的指令系统

指令少 / 寻址方式少 / 指令格式少 / 指令长度一致

(2) 以RR方式工作

除Load/Store指令可访问存储器外，其余指令都只访问寄存器。

(3) 指令周期短

以流水线方式工作， 因而除Load/Store指令外，其他简单指令都只需一个或一个不到的时钟周期就可完成。

(4) 采用大量通用寄存器，以减少访存次数

(5) 采用组合逻辑电路控制，不用或少用微程序控制

(6) 采用优化的编译系统，力求有效地支持高级语言程序

MIPS是典型的RISC处理器，82年以来新的指令集大多采用RISC体系结构

Intel x86因为“兼容”的需要，保留了CISC的风格，同时也借鉴了RISC思想

指令系统举例: Address & Registers

Intel 8086	2^{20} x 8 bit bytes AX, BX, CX, DX SP, BP, SI, DI CS, SS, DS IP, Flags	acc, index, count, quot stack, stack frame, string code,stack,data segment
VAX 11	2^{32} x 8 bit bytes 16 x 32 bit GPRs	r15-- program counter r14-- stack pointer r13-- frame pointer r12-- argument pointer
MC 68000	2^{24} x 8 bit bytes 8 x 32 bit GPRs 7 x 32 bit addr reg 1 x 32 bit SP 1 x 32 bit PC	
MIPS	2^{32} x 8 bit bytes 32 x 32 bit GPRs 32 x 32 bit FPRs HI, LO, PC	HI和LO是MIPS内部的乘商寄存器

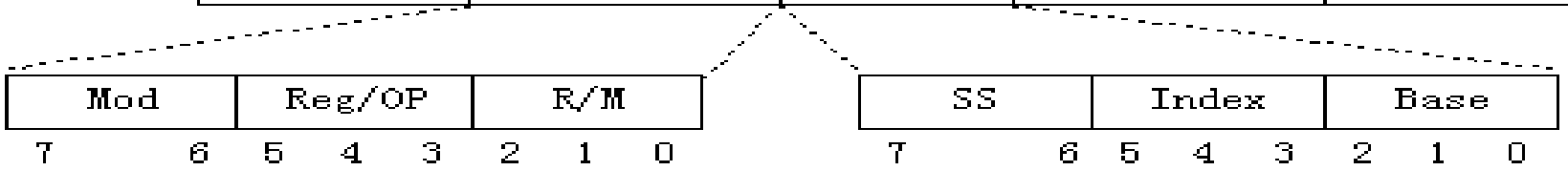
（自学）指令系统举例：Pentium指令格式

前缀：包括指令、段、操作数长度、地址长度四种类型

前缀类型：	指令前缀	段前缀	操作数长度	地址长度
字节数：	0或1	0或1	0或1	0或1

指令：含操作码、寻址方式、SIB、位移量和直接数据五部分，位移量和立即数都可是1/2/4B。SIB中基址B和变址I都可是8个GRS中任一个。SS给出比例因子。操作码：opcode; w：与机器模式（16 / 32位）一起确定寄存器位数（AL / AX / EAX）; d：操作方向；寻址方式： mod、r/m、 reg/op三个字段与w字段和机器模式一起确定操作数所在的寄存器编号或有效地址计算方式

指令段：	操作码	寻址方式	SIB	位移	直接数据
字节数：	1或2	0或1	0或1	1、2、4	立即数



变长指令字：1B~17B
变长操作码：4b / 5b / 6b / 7b / 8b /……
变长操作数：Byte / Word / DW / QW
变长寄存器：8位 / 16位 /32位

问题: 是累加器型、通用寄存器型、ld/st型？是CISC型、RISC型？

调用指令自动把返回地址压栈
专门的push/pop指令，自动修改栈指针
ALU指令在Flags中隐含生成条件码
ALU指令中的一个操作数可来自存储器
提供基址加比例索引寻址

（自学）Pentium处理器的寻址方式

操作数的来源：

立即数(立即寻址)：直接来自指令

寄存器(寄存器寻址)：来自32位 / 16位 / 8位通用寄存器

存储单元(其他寻址)：需进行地址转换

虚拟地址 => 线性地址LA (=> 内存地址)

分段　　分页

指令中的信息：

(1) 段寄存器SR（隐含或显式给出）

(2) 8/16/32位偏移量A （显式给出）

(2) 基址寄存器B （明显给出，任意通用寄存器皆可）

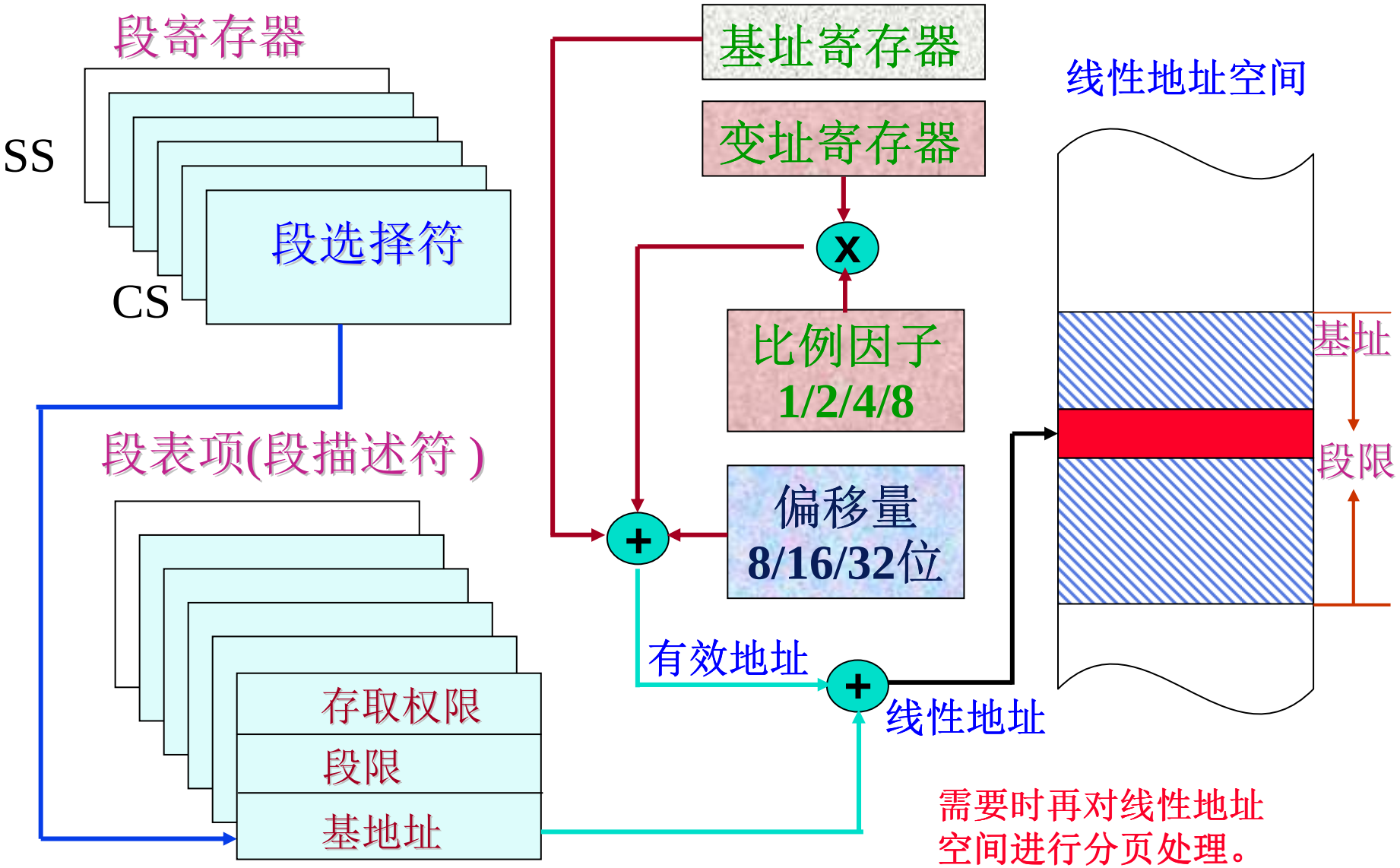
(3) 变址寄存器I （明显给出，除ESP外的任意通用寄存器皆可。）

- 有比例变址和非比例变址
- 比例变址时要乘以比例因子S (1:8位 / 2:16位 / 4:32位 / 8:64位)

（自学） Pentium处理器寻址方式

寻址方式	算法
立即（地址码A本身为操作数）	操作数=A
寄存器（通用寄存器的内容为操作数）	操作数= (R)
偏移量（地址码A给出8/16/32位偏移量）	LA=(SR)+A
基址（地址码B给出基址器编号）	LA=(SR)+(B)
基址带偏移量（一维表访问）	LA=(SR)+(B)+A
比例变址带偏移量（一维表访问）	LA=(SR)+ (I)xS+A
基址带变址和偏移量（二维表访问）	LA=(SR)+(B)+(I) +A
基址带比例变址和偏移量（二维表访问）	LA=(SR)+(B)+(I)xS+A
相对（给出下一指令的地址，转移控制）	转移地址=(PC)+A

（自学）Pentium处理器的存储器寻址



（自学）指令系统举例: PowerPC

- RISC型 （类似于MIPS，32位定长操作码、定长指令字），主要不同在于:
  - 提供了特殊的两种变址寻址方式，可减少指令数
    - » 两个寄存器相加变址（基址寄存器和索引寄存器：间接变址寻址）

例：add $t0,$a0,$s3
lw &t1,0($t0)

lw $t1,$a0+$s3

    - » 自动变址（变址器自动+1）

例： lw &t0, 4($s3)
addi $s3,$s3,4

lwu $t0, 4($s3)

  - 引入特殊的数据块指令，可减少指令数
    - » 单条指令可传送多达32个字，并可进行存储区数据传送
    - » 提供一个特殊计数寄存器ctr，自动减1，用于循环处理

例：for (i=n; i!=0; i=i-1) { };

Loop: ……
addi $t0,$t0,-1
bne &t0, $zero, loop

Loop: ……
bc loop, ctr!=0

（自学） MMX(Microprocessor Media Extension)指令技术

- 图形/像、音/视频多媒体信息处理特点
  - 多个短整数并行操作（如8位图形像素和16位音频信号）
  - 频繁的乘-累加（如FIR滤波，矩阵运算）
- MMX的出发点
  - 使用专门指令对大量数据进行并行、复杂处理
  - 处理的数据基本单位是8b、16b、32b、64b等
- MMX指令集由Intel提出，1997年首次用于P54C Pentium处理器
  - 引入新的数据类型和通用寄存器
    - » 四种64位紧缩定点整数类型（8 x 1B、4 x 1W、2 x 2W、1 x 4W）
    - » 8个64位通用寄存器MX0～MX7（借用8个80位浮点寄存器）
  - 采用SIMD（Single Instruction Multi Data）技术
    - » 单条指令同时并行处理多个数据元素

    例如，一条指令完成图像中8个像素的并行操作
  - 引入饱和（Situration）运算

    非饱和(环绕)运算：上溢时高位数据被截去；饱和运算：上溢时结果取最大值

    例如，图像插值运算：若a点亮度F3H，b点亮度1DH，对a和b线性插值结果为：

    环绕运算：(F3H+1DH)/2=10H/2=08H　插值点的亮度比1DH还低，不合理！

    饱和运算：(F3H+1DH)/2=FFH/2=7FH　合理
- 在Intel以后的处理器中又增加了SSE、SSE2、SSE3等指令集

  SSE（Streaming SIMD extensions）

  SIMD（Single Instrution Multi Data）：单指令多数据技术

小结

- 指令由“操作码”和“地址码”两部分组成。
- 操作类型
  - 传送 / 算术 / 逻辑 / 移位 / 字符串 / 转移控制 / 调用 / 中断 / 信号同步
- 操作数类型
  - 整数（带符号、无符号、十进制）、浮点数、位、位串
- 地址码的编码要考虑：
  - 操作数的个数
  - 寻址方式：立即 / 寄存器 / 寄间 / 直接 / 间接 / 相对 / 基址 / 变址 / 堆栈
- 操作码的编码要考虑：
  - 定长操作码 / 扩展操作码
- 条件码的生成
  - 四种基本标志：NF / VF / CF / ZF
- 指令设计风格：
  - 按操作数地址指定方式来分：
    - » 累加器型 、堆栈型 、通用寄存器型、load/store型
  - 按指令格式的复杂度来分
    - » 复杂指令集计算机CISC、精简指令集计算机RISC
- 典型指令系统举例
  - Pentium / PowerPC / MMX

以下通过MIPS指令系统，介绍如何在机器语言级表示程序