南京大学
NANJING UNIVERSITY

指令系统

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

指令系统

- **指令格式设计**
- 指令系统设计
- 程序的机器级表示
- 指令系统实例

指令格式设计

- **指令系统处在软/硬件交界面，能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（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

回顾：**冯.诺依曼结构机器对指令规定**：

- 用二进制表示，和数据一起存放在主存中
- 由两部分组成：操作码和操作数（或其地址码）
  - **操作码**：定义操作类型
  - **操作数**：表示操作的源和目的

指令格式设计——指令地址码的个数

问题：一条指令必须明显或隐含包含的信息有哪些？

- **操作码**：指定操作类型，如加、减、乘、除、传送等
- **源操作数或其地址**：一个或多个源操作数所在的地址，如主(虚)存地址、寄存器编号、I/O端口、指令给出
- **结果的地址**：产生的结果存放何处（目的操作数），如存储单元地址、寄存器编号、I/O端口
- **下条指令地址**：下条指令存放何处，通常隐含在程序计数器PC中，当改变顺序时由指令给出

指令格式设计——指令地址码的个数

一条指令中应该有几个地址码字段？

零地址指令

(1) 无需操作数　如：空操作 / 停机等

(2) 所需操作数为默认的　如：堆栈 / 累加器等

形式：

OP

一地址指令

其地址既是操作数的地址，也是结果的地址

(1) 单目运算：如：取反 / 取负等

(2) 双目运算：另一操作数为默认的　如：累加器等

形式：

OP	A1

二地址指令（最常用）

分别存放双目运算中两个操作数，并将其中一个地址作为结果的地址。

形式：

OP	A1	A2

三地址指令（RISC风格）

分别作为双目运算中两个源操作数的地址和一个结果的地址。

形式：

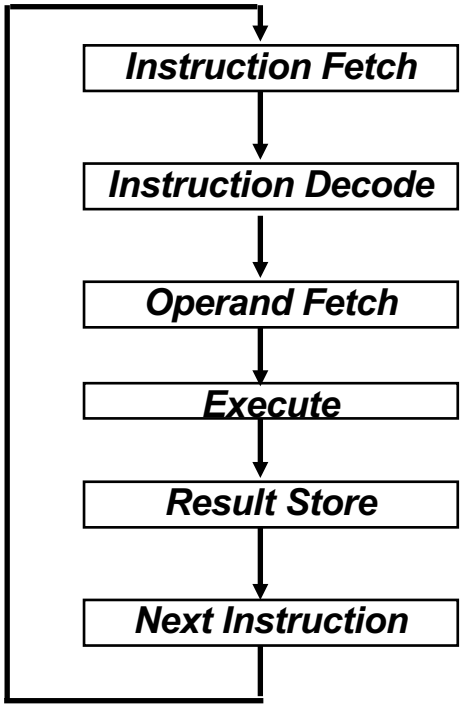
OP	A1	A2	A3

多地址指令

用于成批数据处理的指令，如: 向量 / 矩阵等运算的SIMD指令。

指令格式设计

从指令执行周期看指令设计涉及的问题



- Instruction Fetch：从存储器取指令　指令地址、指令长度（定长/变长）
- Instruction Decode：对指令译码，以确定将要做什么操作　指令格式、操作码编码、操作数类型
- Operand Fetch：计算操作数地址并取操作数　地址码、寻址方式、操作数格式和存放方式
- Execute：进行相应计算，并得到标志位　操作类型、标志或条件码
- Result Store：将计算结果保存到目的地　结果数据位置（目的操作数）
- Next Instruction：计算下条指令地址（通常和取指令同时进行）　下条指令地址（顺序 / 转移）

指令格式设计——指令格式设计原则

指令格式的选择应遵循的几条基本原则

- 应尽量短
- 要有足够的操作码位数
- 指令编码必须有唯一的解释，否则是不合法的序列
- 指令字长应是字节的整数倍
- 合理地选择地址字段的个数
- 指令尽量规整

指令系统

- 指令格式设计
- **指令系统设计**
- 程序的机器级表示
- 指令系统实例

指令系统设计

指令系统设计，必须遵循的基本原则

- **完备性或完整性**：应能足够编制任何可计算程序
- **兼容性**：高档机的指令系统应兼容以前低档机的指令系统
- **均匀性**：运算指令应能对多种类型的数据进行处理
- **可扩充性**：操作码字段要预留一定的编码空间，以便扩充

指令系统设计——基本设计问题

与指令集设计相关的重要方面

- **操作码的全部组成**：操作码个数 / 种类 / 复杂度
  LD/ST/INC/BRN 四种指令已足够编制任何可计算程序，但程序会很长
- **数据类型**：对哪几种数据类型完成操作
- **指令格式**：指令长度 / 地址码个数 / 各字段长度
- **通用寄存器**：个数 / 功能 / 长度
- **寻址方式**：操作数地址的指定方式
- **下条指令的地址如何确定**：顺序，PC+1；条件转移；无条件转移；……

一般通过对操作码进行不同的编码来定义不同的含义，操作码相同时，再由功能码定义不同的含义!

指令系统设计——操作数类型

操作数是指令处理的对象，与高级语言数据类型对应，基本类型有哪些？

- **指针或地址**：被看成无符号整数，用来参加运算以确定主(虚)存地址
- **数值数据**：
  - 定点数(整数)：一般用二进制补码表示
  - 浮点数(实数)：大多数机器采用IEEE 754标准
  - 十进制数：用NBCD码（8421码）表示
- **位、位串、字符和字符串**：
  - 位和位串：标志、控制和状态等信息
  - 字符和字符串：表示文本等

指令系统设计——操作数类型

- **IA-32指令系统**
  - **基本类型**：字节、字(16位)、双字(32位)、四字(64位)
  - **整数**：
    - ✓ 16位、32位、64位三种2-补码表示的整数
    - ✓ 18位压缩8421 BCD码表示的十进制整数
  - **无符号整数**：8、16或32位
  - **浮点数**：IEEE 754（80位扩展精度浮点数寄存器）
  - **近指针**：32位段内偏移（有效地址）

- **MIPS指令系统**
  - **基本类型**：字节、半字(16位)、字(32位)、四字(64位)
  - **整数**： 16位、32位、64位三种2-补码表示的整数
  - **无符号整数**：16、32位
  - **浮点数**：IEEE 754（32位/64位浮点数寄存器）

指令系统设计——寻址方式

- **什么是“寻址方式”**
  - 指令或操作数地址的指定方式。即：根据地址找到指令或操作数的方法。
- **地址码编码**由操作数的寻址方式决定，其编码原则：
  - 指令地址码尽量短 → **目标代码短，省空间**
  - 操作数存放位置灵活，空间应尽量大 → **利于编译器优化产生高效代码**
  - 地址计算过程尽量简单 → **指令执行快**

通常寻址方式特指“操作数的寻址”

- **指令的寻址----简单**
  - 正常：PC增值
  - 跳转（jump / branch / call / return）：同操作数的寻址
- **操作数的寻址----复杂（想象一下高级语言程序中操作数情况多复杂）**
  - 操作数来源：寄存器 / 外设端口 / 主(虚)存 / 栈顶
  - 操作数结构：位 / 字节 / 半字 / 字 / 双字 / 一维表 / 二维表 /...

指令系统设计——寻址方式

- **寻址方式的确定**

  （1）没有专门的寻址方式位（由操作码确定寻址方式）

  如：MIPS指令，一条指令中最多仅有一个主(虚)存地址，且仅有一到两种寻址方式，Load/store型机器指令属于这种情况。

  （2）有专门的寻址方式位

  如：X86指令，一条指令中有多个操作数，且寻址方式各不相同，需要各自说明寻址方式，因此每个操作数有专门的寻址方式位。

- **有效地址的含义**

  操作数所在存储单元的地址（可能是逻辑地址或物理地址），可通过指令的寻址方式和地址码计算得到

- **基本寻址方式**

  立即 / 直接 / 间接 / 寄存器 / 寄存器间接 / 偏移（变址 / 相对 / 基址）/ 其他（如堆栈）

- **基本寻址方式的算法及优缺点（见下页）**

指令系统设计——寻址方式

假设：A=地址字段值，R=寄存器编号，
EA=有效地址，(X)=X中的内容

方式	算法	主要优点	主要缺点
立即	操作数=A	指令执行速度快	操作数幅值有限
直接	EA=A	有效地址计算简单	地址范围有限
间接	EA=(A)	有效地址范围大	多次存储器访问
寄存器	操作数=(R)	指令执行快，指令短	地址范围有限
寄间接	EA=(R)	地址范围大	额外存储器访问
偏移	EA=A+(R)	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

问题：以上各种寻址方式下，操作数在寄存器中还是在存储器中？有没有可能在磁盘中？什么情况下，所取数据在磁盘中？

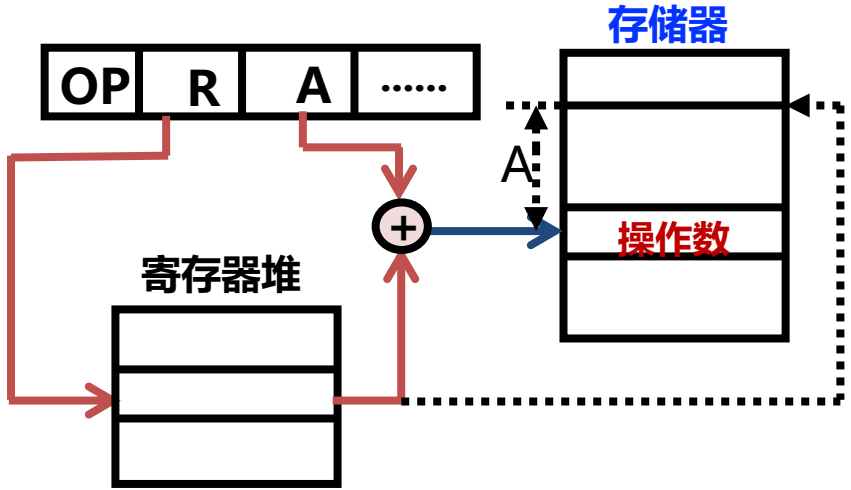
只有当操作数在存储器中时，才有可能“缺页”，此时操作数在磁盘中！

偏移方式：将直接方式和寄存器间接方式结合起来，有 相对 / 基址 / 变址 三种（见后面几页！）

指令系统设计——寻址方式

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**基址A处**位移量为**(I)**的单元

指令系统设计——寻址方式

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）
  - 可用来实现程序(公共子程序)的浮动 或 指定转移目标地址
  - 注意：当前PC的值可以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）
  - 可用来实现多道程序重定位 或 过程调用中参数的访问
- **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
  - 可为循环重复操作提供一种高效机制，如实现对线性表的方便操作

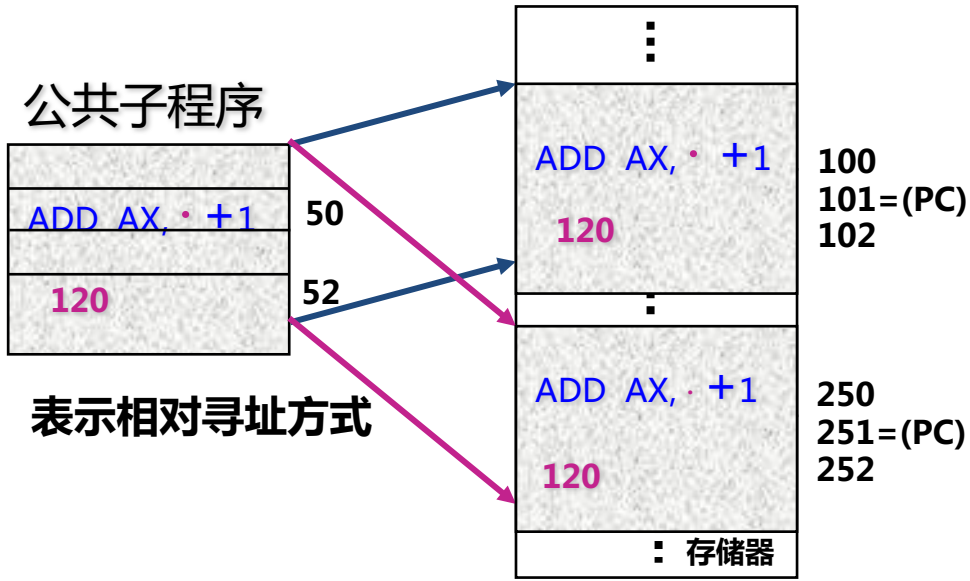
指令系统设计——寻址方式

相对寻址实现公共子程序的浮动

假定每条指令占一个单元

有效地址EA=(PC)+1

即：操作数在当前指令随后的一条指令后！



“ ● ” 表示相对寻址方式

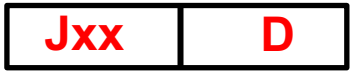
子程序内地址关系相对独立，与用户程序的地址无关，不管浮动到哪里，总是实现AX和120相加。

通常计算有效地址时，PC已指向下条指令!

指令系统设计——寻址方式

相对寻址实现相对转移

举例：双字节定长指令字，其中转移指令的第一字节是操作码Jxx，第二字节是位移量D，用补码表示，则转移目标指令**相对于转移指令的范围**为多少？ **-128～+127 ？ 不一定！**



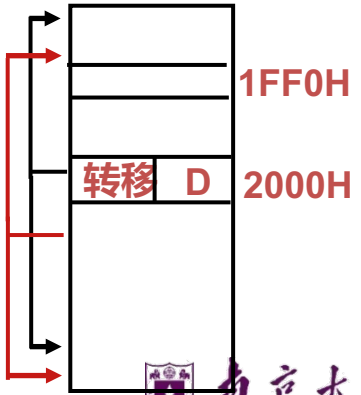
- 若转移指令地址为2000H，转移目标地址为1FF0H，总是在取指令同时对PC增量，则转移指令第二字节位移量为多少？ **不知道！**
- **只有确定了是按字还是字节编址、位移量D是指指令条数还是单元数，才能确定目标地址范围**（**目标地址范围不等于位移量D的表示范围！**）。
- 当按字节编址且D为单元数时，**转移目标地址= (PC)+2+D**

1FF0H = 2000H+2+D

D=1FF0H–2002H=EEH（–18）

跳转范围：-126～128单元

- 63～64条指令



举例：MIPS指令“beq $1, $2, 25”的转移目标地址为(PC)+4+4*25，**这里的25是指令条数而不是单元数**，MIPS采用定长指令字，按字节编址，所有指令的长度都是32位（4字节）。

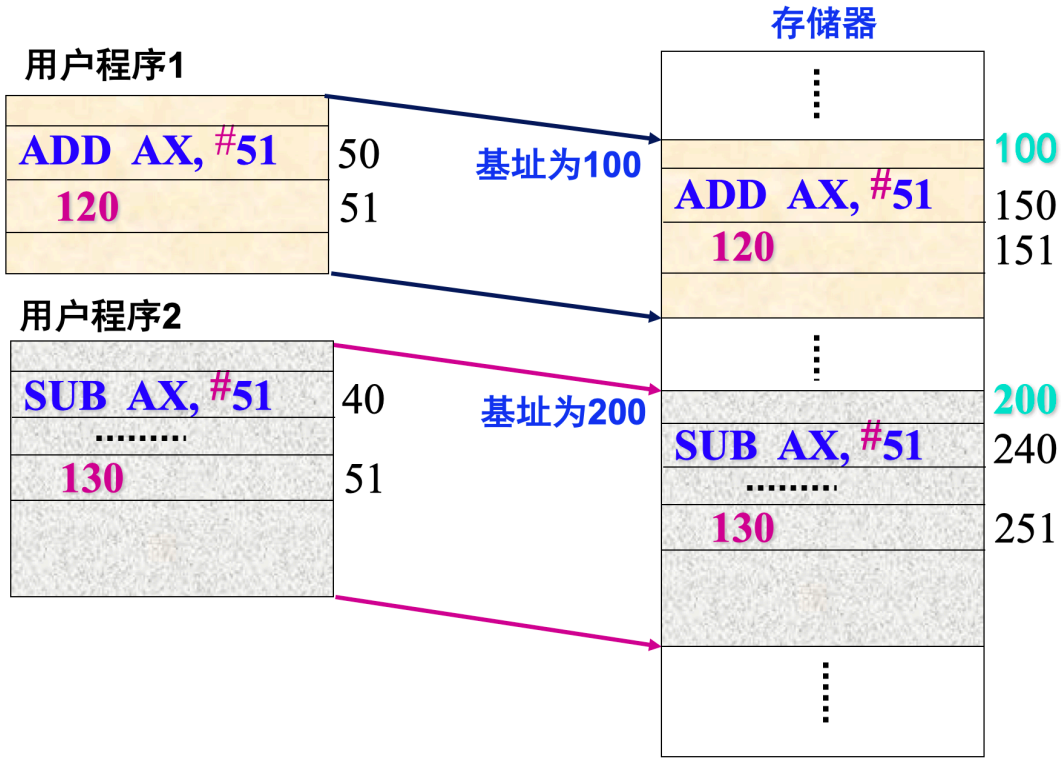
指令系统设计——寻址方式

基址寻址实现程序重定位

“ # ” 表示基址寻址方式

有效地址EA=基址值+51

用户程序装入系统后有一个基址，虽然**偏移量都为51，但因基址不同，故操作数不同**。



指令系统设计——寻址方式

变址寻址实现线性表元素的存取

- **自动变址**

  指令中的地址码A给定数组首址，变址器I每次自动加/减数组元素的长度x。

  EA=(I)+A

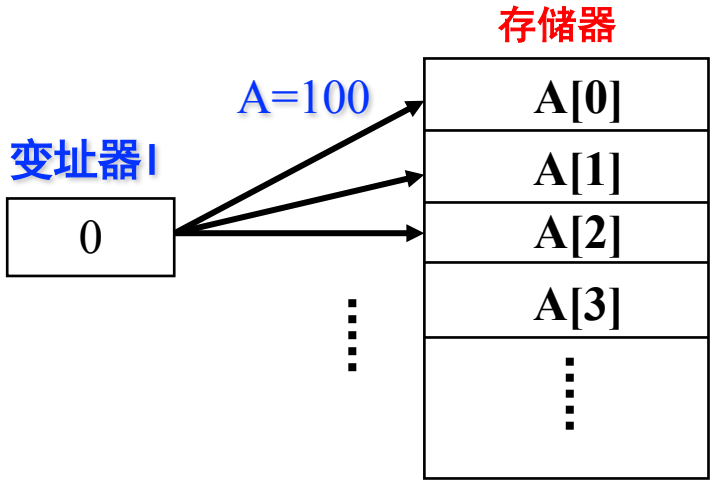
  I=(I) ± x

- **例如，X86中的串操作指令**

  - 对于“for (i=0;i<N;i++) ....” ，即地址从低→高增长：加
  - 对于“for (i=N-1;i>=0;i--) ....”,即地址从高→低增长：减
  - 可提供对线性表的方便访问

一般RISC机器不提供自动变址寻址，并将变址和基址寻址统一成一种偏移寻址方式

假定一维数组A从内存100号单元开始



若每个元素为一个字节，则 I=(I) ± 1

若每个元素为4个字节，则 I=(I) ± 4

指令系统设计——操作类型

- **算术和逻辑运算指令**：加、减、乘、除、比较、与、或、取反等
- **移位指令**：算术移位、逻辑移位、循环移位、半字交换等
- **传送指令**：传送、读取、写等
- **串指令**：串传送、串比较、检索、传送转换等
- **顺序控制指令**：条件转移、无条件转移、跳步、调用、返回等
- **CPU控制指令**：停机、开中断、关中断、系统模式切换等
- **输入输出指令**：CPU与外部设备交换数据或传输控制命令及状态信息

指令系统设计——操作码编码

- **操作码的编码有两种方式**
  - 定长操作码编码
  - 扩展操作码编码
- **编码长度**
  - 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

变长指令字和变长操作码使机器代码更紧凑；定长指令字和定长操作码便于快速访问和译码。

问题：是否可以有定长指令字、变长操作码？定长操作码、变长指令字呢？

指令长度是否可变与操作码长度是否可变没有绝对关系，但通常是：“定长操作码不一定是定长指令字”、“变长操作码一般是变长指令字”。

指令系统设计——操作码编码

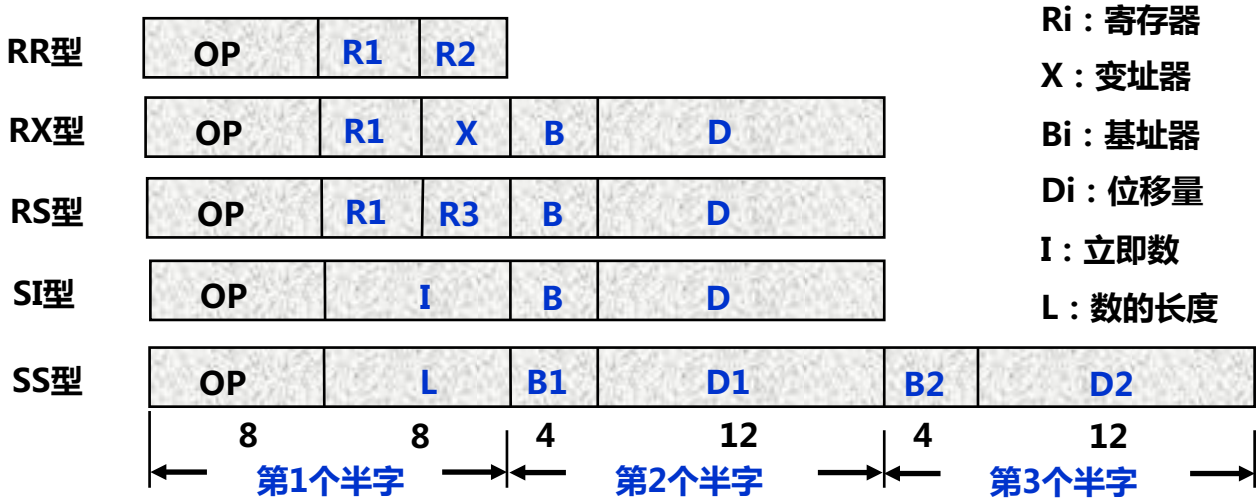
定长操作码编码

- **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
  - 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

问题：通用寄存器编号有几位？B和X的编号占几位？

指令系统设计——操作码编码

IBM 370指令格式



格式：
定长操作码、
变长指令字

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

指令系统设计——操作码编码

扩展（变长）操作码编码

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

举例：

设某指令系统指令字是16位，每个地址码为6位。若二地址指令15条，一地址指令34条，则剩下零地址指令最多有多少条？

解: 操作码按短到长进行扩展编码

二地址指令: (0000 ~ 1110)

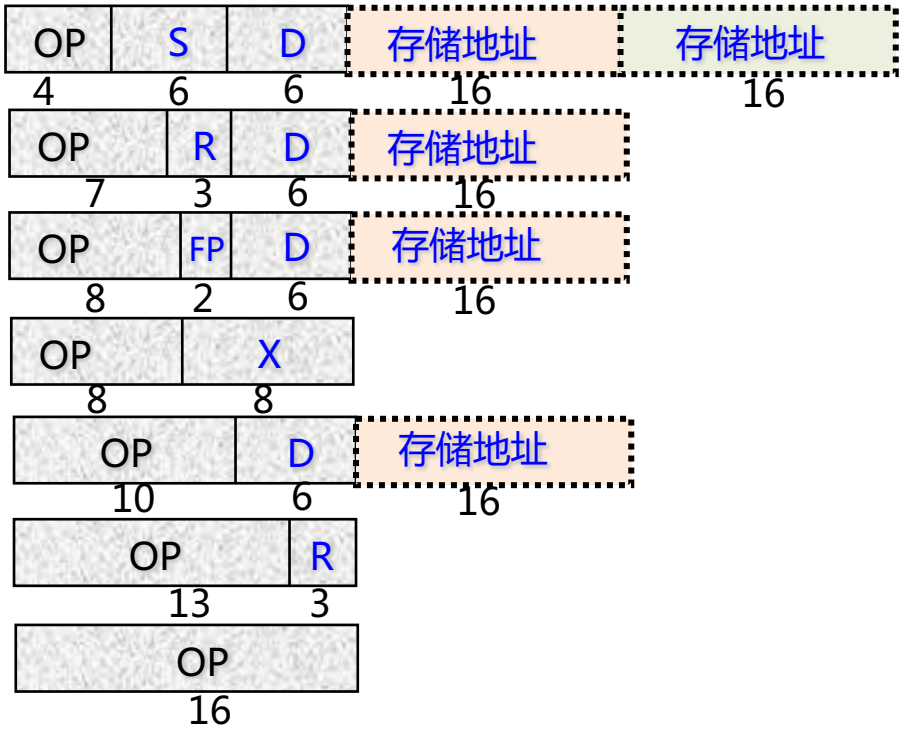
一地址指令: **11110** (00000 ~ 11111); **11111** (00000 ~ 00001)

零地址指令: **11111** (00010 ~ 11111) (000000 ~ 111111)

故零地址指令最多有 $30x2^6=15x2^7$ 种

指令系统设计——操作码编码

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

格式：变长操作码、变长指令字

指令系统设计——标志信息的生成与使用

- **条件转移指令通常根据Condition Codes (条件码 CC/ 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CC

  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时转到label处执行；否则顺序执行

- **常用的标志（条件码）有四种（哪四种？）**

  SF – 符号 OF – 溢出 CF – 进位/借位　ZF – zero

  借位如何生成？$CF=C_{out}\oplus sub$

- **标志可存**标志寄存器/条件码寄存器/状态寄存器/程序状态字寄存器

  也可由指定的通用寄存器来存放状态位

  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

对于带符号和无符号运算，标志生成方式有没有不同？

没有，因为加法电路不知道是无符号数还是带符号整数！

bgt的条件？

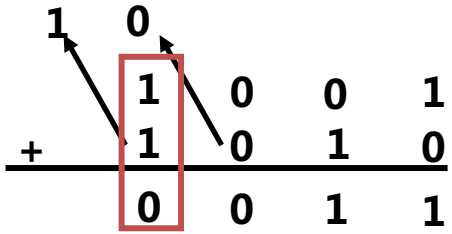
无符号数：ZF=0^CF=0

带符号整数：ZF=0^SF≡OF

指令系统设计——标志信息的生成与使用

例1: -7- 6 = -7 + (-6) = +3

9 - 6 = 3

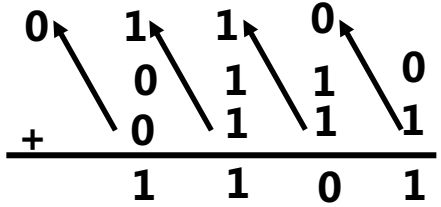


OF=1、ZF=0

SF=0、借位CF=0

6 - (-7) = 6 + 7 = -3

6 - 9 = 13



OF=1、ZF=0

SF=1、借位CF=1

做减法以比较大小，规则：

Unsigned: CF=0时，大于

Signed：OF=SF时，大于

指令系统设计——标志信息的生成与使用

IA-32中的部分条件转移指令

分三类：

(1)根据单个标志的值转移

(2)按无符号整数比较转移

(3)按带符号整数比较转移

序号	指令	转移条件	说明
1	jc label	CF=1	有进位/借位
2	jnc label	CF=0	无进位/借位
3	je/jz label	ZF=1	相等/等于零
4	jne/jnz label	ZF=0	不相等/不等于零
5	js label	SF=1	是负数
6	jns label	SF=0	是非负数
7	jo label	OF=1	有溢出
8	jno label	OF=0	无溢出
9	ja/jnbe label	CF=0 AND ZF=0	无符号整数 A＞B
10	jae/jnb label	CF=0 OR ZF=1	无符号整数 A≥B
11	jb/jnae label	CF=1 AND ZF=0	无符号整数 A＜B
12	jbe/jna label	CF=1 OR ZF=1	无符号整数 A≤B
13	jg/jnle label	SF=OF AND ZF=0	带符号整数 A＞B
14	jge/jnl label	SF=OF OR ZF=1	带符号整数 A≥B
15	jl/jnge label	SF≠OF AND ZF=	带符号整数 A＜B
16	jle/jng label	SF≠OF OR ZF=1	带符号整数 A≤B

指令系统设计——指令系统设计风格

- **累加器型指令系统 (早期机器)**

  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]

- **堆栈型指令系统 (e.g. HP calculator, Java virtual machines)**

  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

- **通用寄存器型指令系统 (e.g. IA-32, Motorola 68xxx)**

  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)

- **装入/存储型指令系统 (e.g. SPARC, MIPS, PowerPC)**

  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

按操作数位置指定风格来分

指令系统设计——指令系统设计风格

- **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**

变长操作码 / 变长指令字 / 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。

指令系统设计——指令系统设计风格

- **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 Cocks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**。

指令系统设计——指令系统设计风格

° 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

Top 10 80x86 Instructions

（简单指令占主要部分，使用频率高！）

指令系统设计——指令系统设计风格

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

x86因为“兼容”的需要，保留了CISC的风格，同时也借鉴了RISC思想

指令系统设计——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

Mod	Reg/OP	R/M
7 6	5 4 3	2 1 0

SS	Index	Base
7 6	5 4 3	2 1 0

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

指令系统设计——指令系统举例（Pentium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)

指令系统设计——指令系统举例（Pentium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

指令系统设计——指令系统举例（Pentium寻址方式）

存储器操作数的寻址方式

```
int x ;
float a[100];
short b[4][4];
char c;
double d[10];
```

a[i]的地址如何计算？

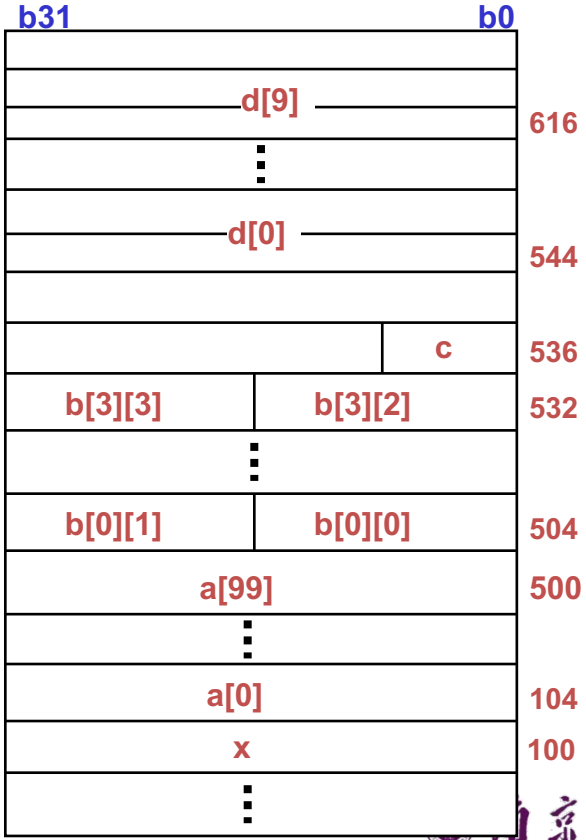
104+i×4；i=99时，104+99×4=500

b[i][j]的地址如何计算？

504+i×8+j×2；i=3、j=2时，504+24+4=532

d[i]的地址如何计算？

544+i×8；i=9时，544+9×8=616



指令系统设计——指令系统举例（Pentium寻址方式）

存储器操作数的寻址方式

```
int x ;
float a[100];
short b[4][4];
char c;
double d[10];
```

- **各变量应采用什么寻址方式？**

x、c：位移 / 基址

a[i]：104+i×4，比例变址+位移

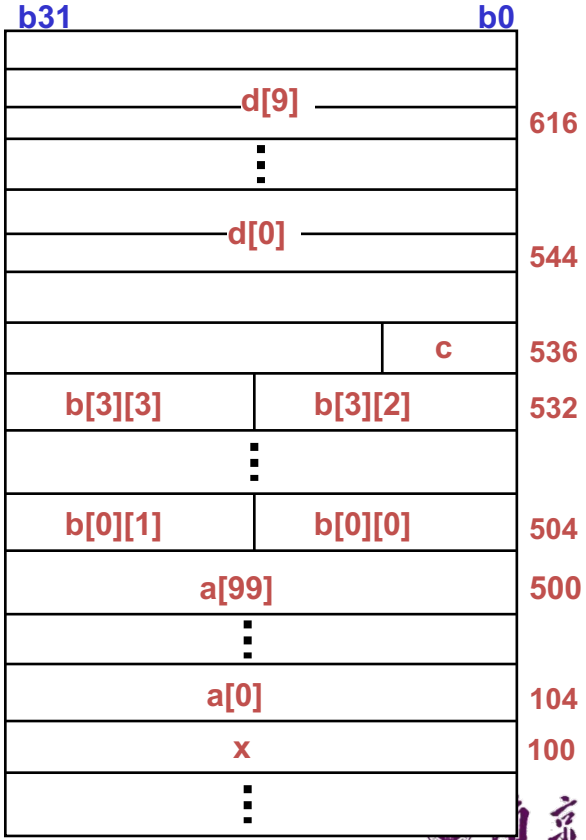
d[i]：544+i×8，比例变址+位移

b[i][j]： 504+i×8+j×2，基址+比例变址+位移

- **将b[i][j]取到AX中的指令可以是：**

“movw 504(%ebp,%esi,2), %ax”

其中， i×8在EBP中，j在ESI中，2为比例因子



指令系统设计——指令系统举例（MMX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，AVX等指令集**

  SSE（Streaming SIMD extensions）

指令系统设计——异常和中断处理机制

- **程序执行过程中CPU会遇到一些特殊情况，使正在执行的程序被“中断”**
  - CPU中止原来正在执行的程序，转到处理异常情况或特殊事件的程序去执行，结束后再返回到原被中止的程序处（**断点**）继续执行。
- **程序执行被 “中断” 的事件有两类**
  - **内部异常**：在CPU执行某指令时内部发生的意外事件或特殊事件

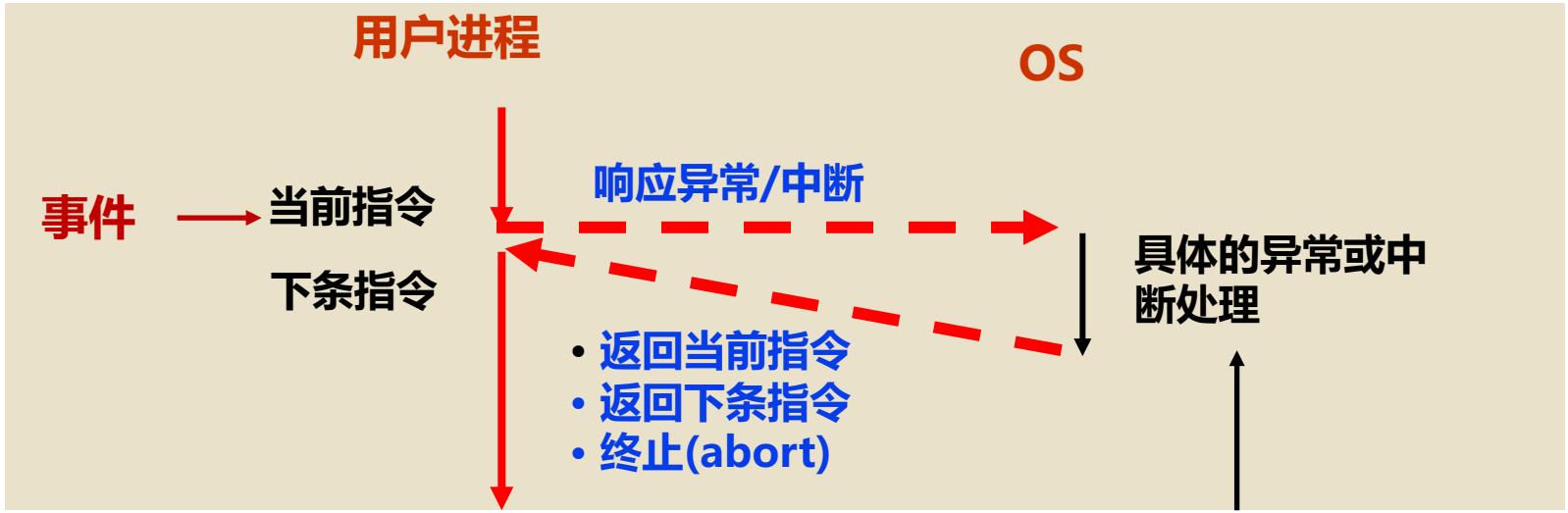
    故障(fault)：执行某条指令时发生的异常事件，如溢出、缺页、越界、越权、越级、非法指令、除数为0、堆/栈溢出、访问超时等。

    自陷(trap)：执行预先设置的指令，如断点、单步、系统调用等。

    终止(abort)：指令执行过程中出现了硬件故障，如访存校验错等。
  - **外部中断**：在CPU外部发生的特殊事件，通过“中断请求”信号向CPU请求处理。如实时钟、控制台、打印机缺纸、外设准备好、采样计时到、DMA传输结束等。
- **异常/中断处理分两个阶段**
  - 检测和响应：由硬件完成
  - 具体的处理过程由软件（操作系统）执行程序完成

指令系统设计——异常和中断处理机制

- 发生异常(exception)和中断(interrupt)事件后，系统将进入OS内核态对相应事件进行处理，即改变处理器状态（用户态→内核态）



用户进程的正常控制流中插入了一段内核控制路径

指令系统

- 指令格式设计
- 指令系统设计
- **程序的机器级表示**
- 指令系统实例

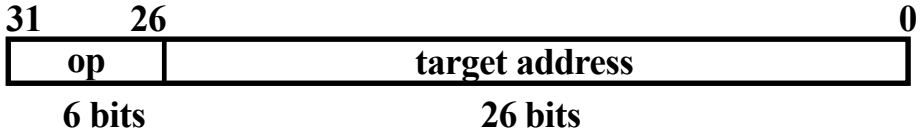
程序的机器级表示——MIPS汇编语言和机器语言

- MIPS中，所有指令都是32位宽，须按字地址对齐，字地址为4的倍数！
- **MIPS有三种指令格式**
  - **R-Type**

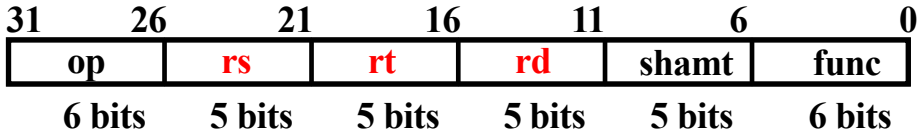
    两个操作数和结果都在寄存器的运算指令。如：sub rd, rs, rt
  - **I-Type**
    - 运算指令：一个寄存器、一个立即数。如：ori rt, rs, imm16
    - LOAD和STORE指令。如：lw rt, rs, imm16
    - 条件分支指令。如：beq rs, rt, imm16
  - **J-Type**

    无条件跳转指令。如：j target

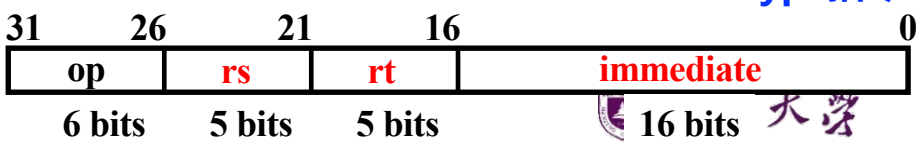
J-Type指令



R-Type指令



I-Type指令



程序的机器级表示——MIPS汇编语言和机器语言

OP：操作码

rs：第一个源操作数寄存器

rt：第二个源操作数寄存器

rd：结果寄存器

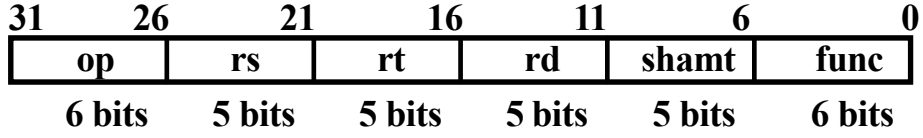
shamt：移位指令的位移量

func：R-Type指令的OP字段是特定的“000000”，具体操作由func字段给定。例如：func=“100000”时，表示“加法”运算。

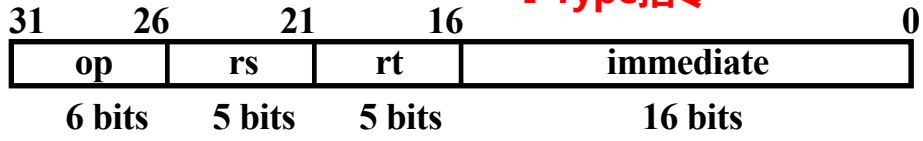
immediate：立即数或load/store指令和分支指令的偏移地址

target address：无条件转移地址的低26位。将PC高4位拼上26位直接地址，最后添2个“0”就是32位目标地址。为何最后两位要添“0”？

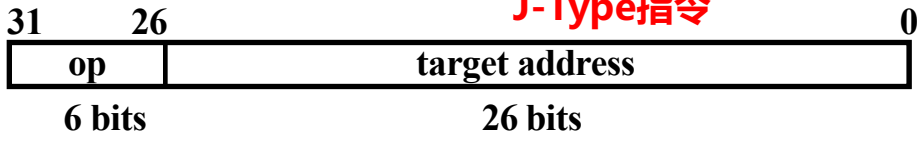
R-Type指令



I-Type指令



J-Type指令



- 操作码的不同编码定义不同的含义，操作码相同时，再由功能码定义不同的含义!
- 指令按字地址对齐，所以每条指令的地址都是4的倍数（最后两位为0）。

OP字段的含义

op(31:26)　　op=0: R型；op=2/3：J型；其余：I型

28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	sltiu	andi	ori	xori	load upper imm
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	lbu	lhu	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	lwc0	lwc1						
7(111)	swc0	swc1						

R-型指令的解码（op=0时，func字段的编码/解码表）

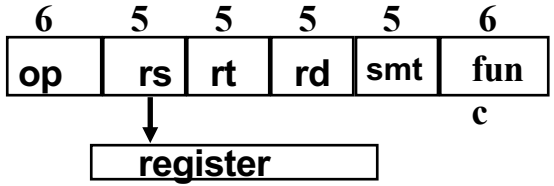
op(31:26)=000000 (R-format), funct(5:0)								
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump reg.	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	sltu				
6(110)								
7(111)								

add指令的func字段为100000B（32）
div指令的func字段为多少？
011010B（26）！

程序的机器级表示——MIPS汇编语言和机器语言

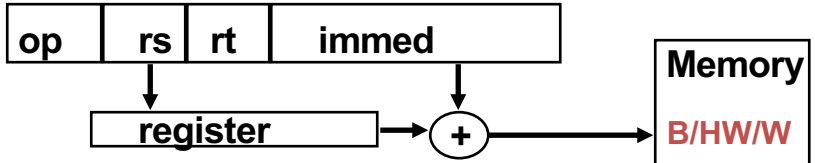
OP=000000H

R-format:
Register

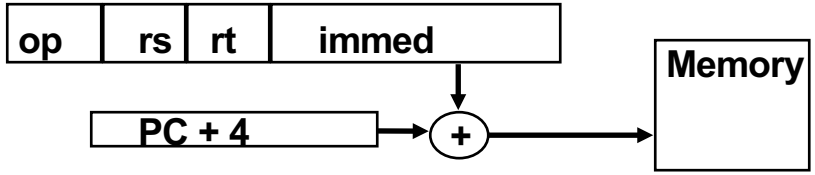


I-format:
Immediate

Base或index
基址或变址

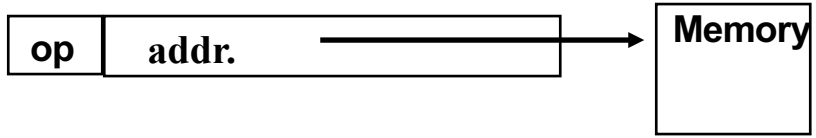


PC-relative
相对寻址



J-format: OP=000010H or 000011H

Pseudodirect
伪直接寻址



MIPS寻址方式

有专门的寻址方式字段（Mod）吗？

没有！由指令格式来确定，而指令格式由op来确定！

还记得如何确定的吗？

Byte / Half Word / Word

为什么称伪直接？还记得如何得到最终地址的吗？

最终地址=$PC_{31\sim28}$||addr.||00

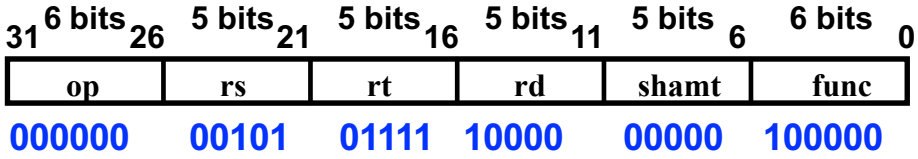
位数：4+26+2=32

汇编形式与指令的对应

- 若从存储器取来一条指令为00AF8020H，则对应的汇编形式是什么？

32位指令代码：0000 0000 1010 1111 1000 0000 0010 0000

指令的前6位为000000，根据指令解码表知，是一条R-Type指令，按照R-Type指令的格式



得到： rs=00101, rt=01111, rd=10000, shamt=00000, funct=100000

1. 根据R-Type指令解码表，知是 “add”操作（非移位操作)
2. rs、rt、rd的十进制值分别为5、15、16，从MIPS寄存器功能表知:
   rs、rt、rd分别为：$a1、$t7、$s0

故对应的汇编形式为：

add $s0 ，$a1，$t7 **功能：$a1 + $t7 → $s0**

这个过程称为“反汇编”，可用来破解他人的二进制代码（可执行程序）.

程序的机器级表示——MIPS汇编语言和机器语言

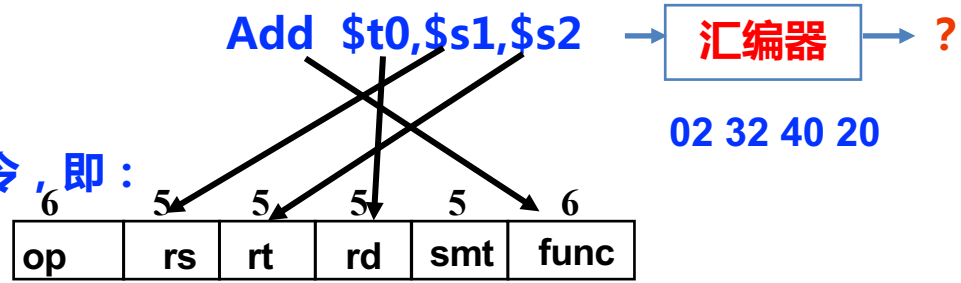
汇编形式与指令的对应

- 若MIPS汇编指令为:
  则对应的指令机器代码是什么？

从助记符表中查到Add是R型指令，即：

何为助记符？

汇编语言中的指令和寄存器等的名称。



问题：如何知道是R型指令？

根据汇编指令中的操作码助记符查表能知道是什么格式！

这个过程称为“汇编”，所有汇编源程序都必须汇编成二进制机器代码才能让机器直接执行！

十进制表示：

6	5	5	5	5	6
0	17	18	8	0	32
R-Type	$s1	$s2	$t0	No shift	Add

二进制表示：

6	5	5	5	5	6
000000	10001	10010	01000	00000	100000

0232 4020H

程序的机器级表示——MIPS汇编语言和机器语言

表 4.2　MIPS 通用寄存器

名　　称	编　　号	功　　能
zero	0	恒为 0
at	1	为汇编程序保留
v0～v1	2～3	过程调用返回值
a0～a3	4～7	过程调用参数
t0～t7	8～15	临时变量，在被调用过程无须保存
s0～s7	16～23	在被调用过程需保存
t8～t9	24～25	临时变量，在被调用过程无须保存
k0～k1	26～27	为 OS 保留
gp	28	全局指针
sp	29	栈指针
fp	30	帧指针
ra	31	过程调用返回地址

寄存器的汇编表示以$符号表示，可以使用名称（**$ a0**），也可以用使用编号 (**$ 4**)。

程序的机器级表示——MIPS汇编语言和机器语言

表 4.3　MIPS 汇编语言示例列表

类别	指令名称	汇编举例	含义	备注
算术运算	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	三个寄存器操作数
	subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3	三个寄存器操作数
存储访问	load word	lw $s1,100($s2)	$s1 = Memory[$s2+100]	从内存取一个字到寄存器
	store word	sw $s1,100($s2)	Memory[$s2+100] = $s1	从寄存器存一个字到内存
逻辑运算	and	and $s1, $s2, $s3	$s1 = $s2 & $s3	三个寄存器操作数,按位与
	or	or $s1, $s2, $s3	$s1 = $s2 | $s3	三个寄存器操作数,按位或
	nor	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	三个寄存器操作数,按位或非
	and immediate	andi $s1, $s2,100	$s1 = $s2 & 100	寄存器和常数,按位与
	or immediate	ori $s1, $s2,100	$s1 = $s2 | 100	寄存器和常数,按位或
	shift left logical	sll $s1, $s2,10	$s1 = $s2≪10	按常数对寄存器逻辑左移
	shift right logical	srl $s1, $s2,10	$s1 = $s2≫10	按常数对寄存器逻辑右移
条件分支	branch on equal	beq $s1, $s2,L	if($s1 == $s2) go to L	相等则转移
	branch on not equal	bne $s1, $s2,L	if($s1 != $s2) go to L	不相等则转移
	set on less than	slt $s1, $s2, $s3	if($s2< $s3) $s1=1; else $s1=0	小于则置寄存器为 1,否则为 0,用于后续指令判 0
	set on less than immediate	slt $s1, $s2,100	if($s2<100) $s1=1; else $s1=0	小于常数则置寄存器为 1,否则为 0,用于后续指令判 0
无条件跳转	jump	j L	go to L	直接跳转至目标地址
	jump register	jr $ra	go to $ra	过程返回
	jump and link	jal L	$ra=PC+4; go to L	过程调用

程序的机器级表示——MIPS汇编语言和机器语言

表 4.4　MIPS 机器代码示例列表

指　令	格式	指令举例						备　注
add	R	0	18	19	17	0	32	add　$s1，$s2，$s3
sub	R	0	18	19	17	0	34	sub　$s1，$s2，$s3
lw	I	35	18	17	100			lw　$s1，100($s2)
sw	I	43	18	17	100			sw　$s1，100($s2)
and	R	0	18	19	17	0	36	and　$s1，$s2，$s3
or	R	0	18	19	17	0	37	or　$s1，$s2，$s3

指　令	格式	指令举例						备　注
nor	R	0	18	19	17	0	39	nor　$s1，$s2，$s3
andi	I	12	18	17	100			andi　$s1，$s2，100
ori	I	13	18	17	100			ori　$s1，$s2，100
sll	R	0	0	18	17	10	0	sll　$s1，$s2，10
srl	R	0	0	18	17	10	2	srl　$s1，$s2，10
beq	I	4	17	18	25			beq　$s1，$s2，100
bne	I	5	17	18	25			bne　$s1，$s2，100
slt	R	0	18	19	17	0	42	slt　$s1，$s2，$s3
j	J	2	2500					j　10000
jr	R	0	31	0	0	0	8	jr　$ra
jal	J	3	2500					jal　10000
字段大小		6 位	5 位	5 位	5 位	5 位	6 位	
R-型指令	R	OP	rs	rt	rd	shamt	func	
I-型指令	I	OP	rs	rt	address			

提问

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

南京大學 NANJING UNIVERSITY