Chapter2.指令系统

2.1 概述

指令系统=机器指令集合(硬件语言，机器语言)

→反应计算机基本功能+决定了硬件的所有功能和结构+影响软件结构

2.2 指令系统分类

2.2.1. CICS

1️⃣特征：

1. 软件硬化：指令实现的功能复杂，例如把排序算法用CPU架构实现
2. 向上向后兼容：旧软件在新机器运行，新机器包含老机器所有指令
3. 支持高级语言

2️⃣缺点：大量不经常使用的指令导致计算机硬件非常复杂

2.2.2.RISC

特点

1. 优选使用频率高的简单指令
2. 指令长度固定，指令格式种类少，寻址方式种类少
3. 只有存取数/指令才访问寄存器(CISC种任何一条指令都可访问存储器)
4. 通用寄存器相当多等等
5. CPU采用流水线结构，CPI低
6. 控制单元以硬布线逻辑为主
7. x

PS：现代处理器一般兼容RISC+CICS

2.3 指令系统的功能与设计

2.3.1. 指令系统设计原则

1️⃣CICS强调完备性，功能完整

2️⃣规整性：对称性(寻址方式统一)，匀齐性(对不同数据类型处理方式一样)，一致性(指令数据格式统一)

3️⃣高效性：尽量多的功能给硬件实现(CICS)，降低每条指令的执行时间(RISC)

2.3.2. 数据

1️⃣数据类型：整型/布尔/字符(简单)，文件/图/队列(复杂)

2️⃣数据表示：硬件能够识别，指令能够操作的数据结构，如正数，布尔，浮点，字符串，栈

3 操作数：机器指令的数据，能够被硬件直接识别和处理，包含地址/数字/字符/逻辑数(逻辑运算的数)

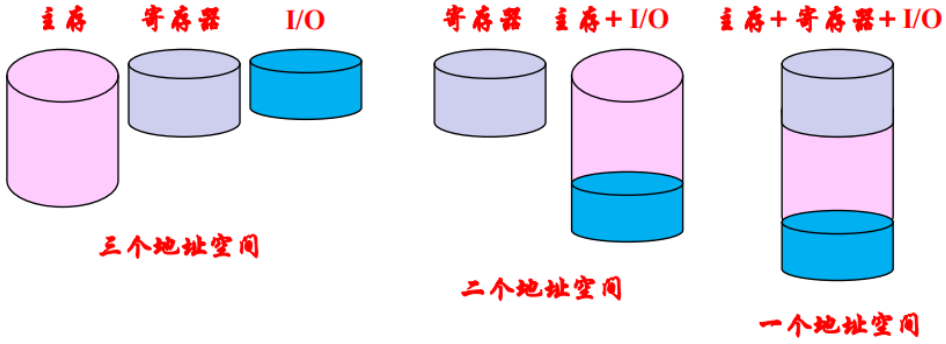
2.3.3. 地址空间

2.3.3.1. 单元编址方式

1 独立编址：主存，寄存器，IO的地址的开头都是0，如8086(左)

2 混合地址：如ARM(种)

3 统一编制：寄存器从0开始，主存从64开始，IO从128开始



2.3.3.2. 主存编址方式

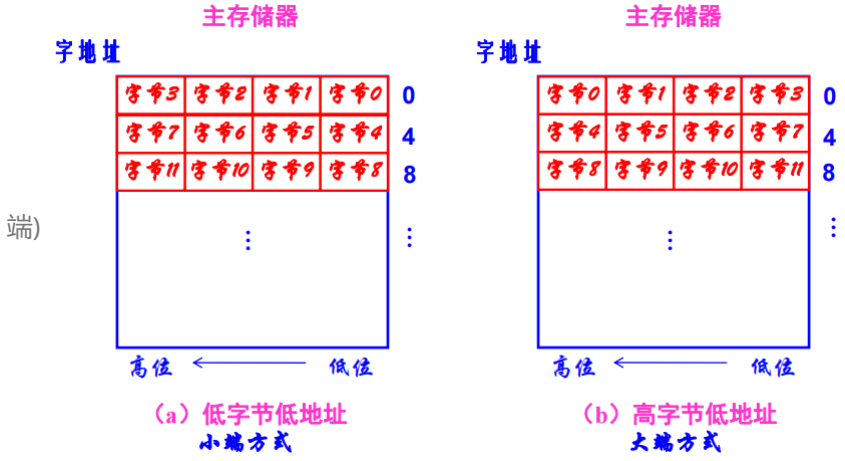
1 按字编址：最小编址&访问单位为存贮字长(通常=机器字长)，主存容量=存储字数x存储字长，例如128Mx32位，**格式必须是这样**

2 按字节编址：最小编址&访问单位为8位

1. 访问速度更快，但是不灵活
2. 主存数据既能以字节为单位访问，也能以字为单位访问

3 字节地址与字地址

1. 基于两种编址方式的地址，字地址不连续而字节地址连续
2. 字地址中字节地址有两种编址顺序：低字节低地址(小端，X86)，高字节低地址(大端)



3. 存放边界问题：边界可对其与不对其

4 边界问题：边界对齐方式&边界不对齐方式

在以下例子中，我们按顺序定义如下变量int1→char1→char2→int2→半字数据1→半字数据2

1. 边界对齐方式：

①int1(地址0)→char1(4)→char2(5)→int2(8)→半字数据(12)→半字数据2(14)

②也就是说，字只能放在字起始地址，半字可以放在字起始和中间，字节可以放在字的起始/0.25/0.5/0.75处

③缺点会浪费空间，但是好处是性能会高一些()

2. 边界不对齐方式：

①int1(地址0)→char1(4)→char2(5)→int2(6,8)→半字数据(12)→半字数据2(14)

②缺点显然是效率低，取int2需要取两个字，屏蔽不相关的，再拼接

2.3.4. 指令设计的操作类型

1. 数据传送指令
2. 数据运算指令(算数/逻辑/移位/位操作)
3. 程序控制指令(转移/调用/返回/陷阱aka中断调用)
4. IO指令
5. 其它指令

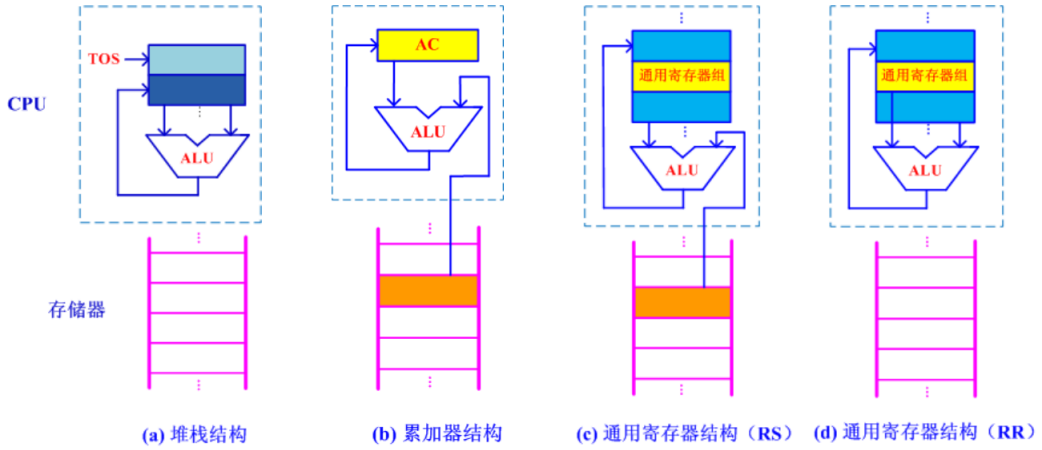
2.4. 指令格式

```
[操作码(指令功能)][地址码(操作对象的地址)]
```

2.4.1. 指令字长

数值上=操作码长度+地址码1长度+地址码2长度+……+地址码n长度，可等长(=机器字长)也可变长

2.4.2. 四种 CPU 结构



1️⃣堆栈结构：CPU只有一个寄存器叫TOS(栈顶)， 运行示例S=AB+CD如下

```
规则：从左往右，遇到数据就把它压入栈，遇到运算符就把数据取出来
A  B  *  C  D  *  +          栈底→ A              备注：数据压栈
↑
A  B  *  C  D  *  +          栈底→ A  B           备注：数据压栈
   ↑
A  B  *  C  D  *  +          栈底→ AB             备注：A,B出栈送ALU，运算结果AB回栈
↑
A  B  *  C  D  *  +          栈底→ AB C           备注：数据压栈
   ↑
A  B  *  C  D  *  +          栈底→ AB C  D        备注：数据压栈
      ↑
A  B  *  C  D  *  +          栈底→ AB CD          备注：C,D出栈送ALU，运算结果CD回栈
         ↑
A  B  *  C  D  *  +          栈底→S               备注：AB,CD出栈送ALU，运算结果AB+CD回栈
            ↑
```

2️⃣累加器结构：运行模式与示例为

1.把第一个数从主存取出送AC，再把第二个数送给ALU
2.ALU和AC的计算结果送给ALU

```
MOV [100],AC   ;先把100从主存中取出到AC
ADD [101]      ;101送ALU，默认与AC中的100进行累加，结果送ALU
```

3️⃣通用寄存器结构(RS/RR)：X86支持这两种

```
ADD AX [100]   ;RS执行操作，一个数据来源于内存，一个来源于通用寄存器
ADD AX BX      ;RS执行操作，两个数据来源
```

2.4.3. 地址码字段

```
1.四地址码:[操作码][A1][A2][A3][PC] ;执行操作A3<-(A1)OP(A2),PC为下条指令的地址
2.三地址码:[操作码][A1][A2][A3]     ;执行操作A3<-(A1)OP(A2)
3.二地址码:[操作码][A1][A2]          ;执行操作A1<-(A1)OP(A2),就是8086算数操作
4.一地址码:[操作码][A1]              ;执行操作AC<-(AC)OP(A1),就是8086的累加器
5.O地址码:[操作码]                   ;操作数送堆栈栈顶
```

2.4.4. 操作码字段

1 每个指令与操作码一一映射

2 分为定长操作码(操作码位数相同，在指令字中的位置固定)，变长操作码(二者反之)

3 **操作码拓展技术**：对定长指令，可空出部分址码字段，来增加操作码的位数，在实际操作中(下图)保留1111为扩展标志

<table>
<tr><th colspan="4">指令格式及操作码编码</th><th>说明</th></tr>
<tr><td>OP_Code</td><td>A1</td><td>A2</td><td>A3</td><td rowspan="2">4位操作码的三地址指令
15条</td></tr>
<tr><td>0000
0001
…
1110</td><td>A1</td><td>A2</td><td>A3</td></tr>
<tr><td colspan="2">OP_Code</td><td>A1</td><td>A2</td><td rowspan="2">8位操作码的二地址指令
15条</td></tr>
<tr><td>1111 0000
1111 0001
… …
1111 1110</td><td colspan="2">A1</td><td>A2</td></tr>
<tr><td colspan="3">OP_Code</td><td>A</td><td rowspan="2">12位操作码的一地址指令
15条</td></tr>
<tr><td>1111 1111 0000
1111 1111 0001
… … …
1111 1111 1110</td><td colspan="3">A</td></tr>
<tr><td colspan="4">OP_Code</td><td rowspan="2">16位操作码的零地址指令
16条</td></tr>
<tr><td>1111 1111 1111 0000
1111 1111 1111 0001
… … … …
1111 1111 1111 1111</td><td colspan="4">OP_Code</td></tr>
</table>

2.4.5. MIPS32指令格式(32位定长指令)

1 R-指令：OP_Cods全为0的时候表示是一个算术逻辑运算，Func表示算术逻辑类型，Shamt移位用，执行运算Rs(OP)Rt→Rd

2 :I-指令：执行操作Rs(OP)Rt→Rt

3 J-指令：从当前位置跳转到Addr26给出的位置

(a) R-型指令

(b) I-型指令

2.5 寻址方式

形式地址(符号地址) $\xrightarrow{\text{寻址方式}}$ 有效地址(逻辑地址)

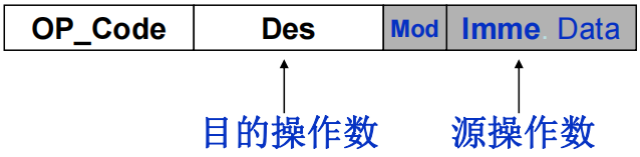
分为指令寻址(设置PC值)，数据寻址，以下介绍数据寻址

2.5.0. 寻址指令格式

```
[操作码][寻址特征1][A1][寻址特征2][A2]
```

2.5.1. 立即寻址

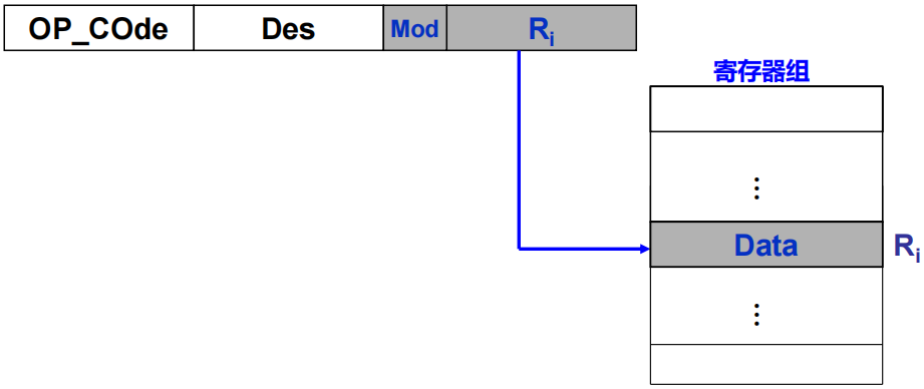
Des(操作数目的地) ⬅ 地址字段(立即数/操作数)




```
mov eax,100       ;IA-32
addi $s1,$s2,100 ;MIPS32，取$s2寄存器中的值，加上100，然后把结果存储在$s1寄存器中
```

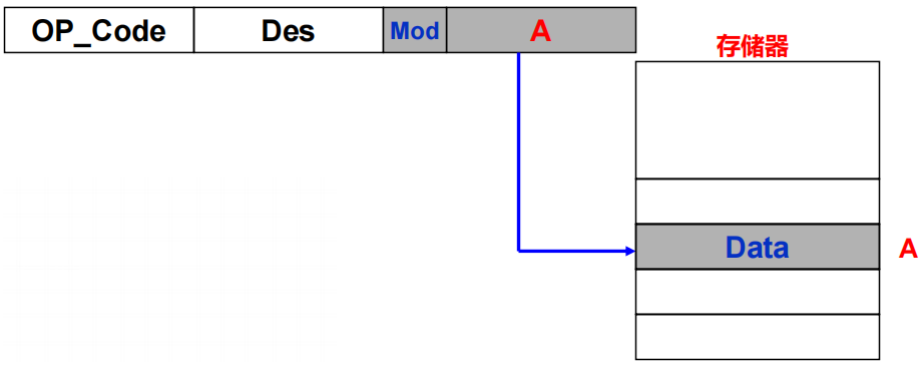
2.5.2. 寄存器寻址

Des(操作数目的地) ⬅ 寄存器(操作数所在地) ⬅ 地址字段(寄存器号)



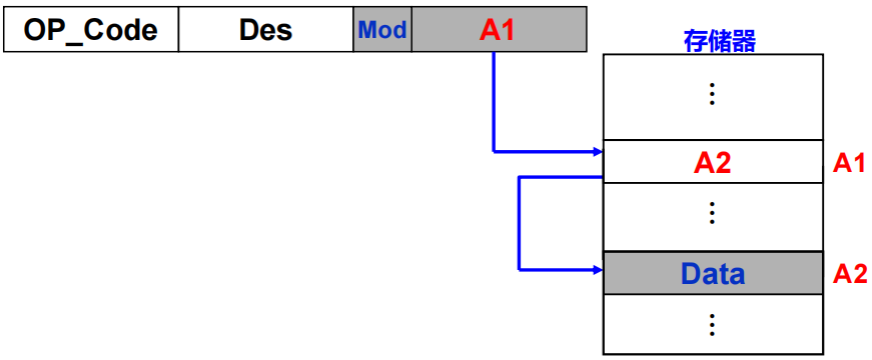
2.5.3. 存储器的直接寻址

Des(操作数目的地) ⬅ 内存(操作数所在地) ⬅ 地址字段(存储器地址)



2.5.4. 存储器的间接寻址

Des(操作数目的地)⬅️内存n(操作数所在地)⬅️.....⬅️内存2(存有内存3的地址)⬅️内存1(存有内存2的地址)⬅️地址字段(存储器地址)



1️⃣可一次间接和多次间接寻址，多次间接寻址时，将A2的最高位设置为间址标志位I，I=0时，间址结束

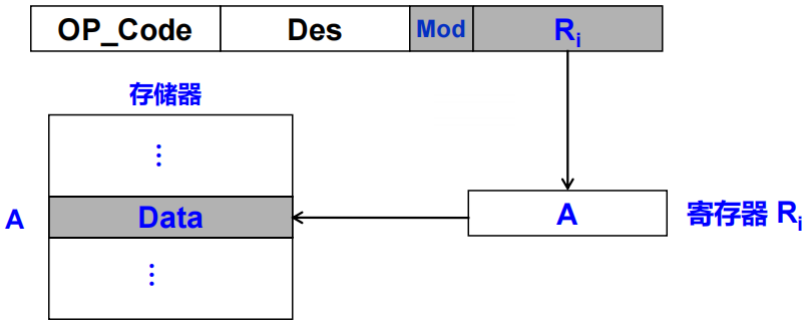
2️⃣好处是可扩大指令寻址范围，但指令执行速度慢

```
jmp word ptr[1000]
;从内存1000地址处取得另一个地址，跳转到另一个地址，PC指针也跳转过去
```

2.5.5. 寄存器间接寻址

Des(操作数目的地)⬅️存储器(操作数所在地)⬅️寄存器(存有内存单元的地址)⬅️地址字段(寄存器号)

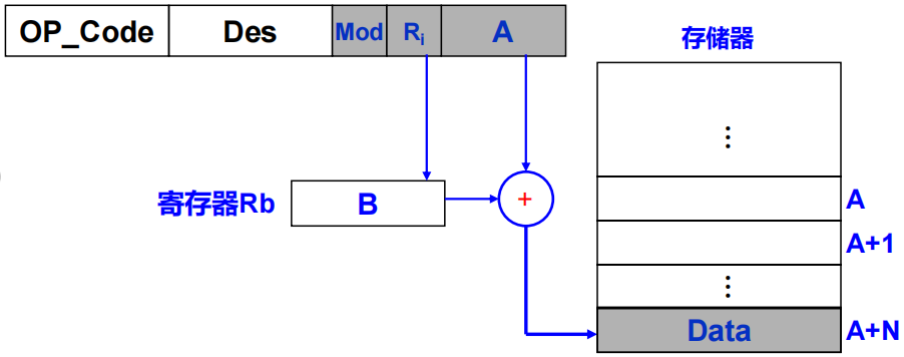
地址码字段给出某一通用寄存器的编号，该寄存器中存放的是操作数在主存单元的地址




```
mov eax [ebx] ;EA = (Ri), Operand = ((Ri))
```

2.5.6. 偏移寻址

Des(操作数目的地)⬅️内存(操作数所在地)⬅️地址字段(寄存器给出的偏移量+指定内存的位置)



三种便宜寻址的方式

1️⃣相对寻址：引用专门的程序计数器 PC，即 EA =(PC)+A，指令中只需要给出偏移量A

2️⃣变址寻址：引用一个变址寄存器 Rx(专用or通用，专用可缺省寄存器号), EA =(Rx)+A

Rx给出变址(用户设定)，A给出基址，操作数地址的变化由变址值(Rx)增减完成

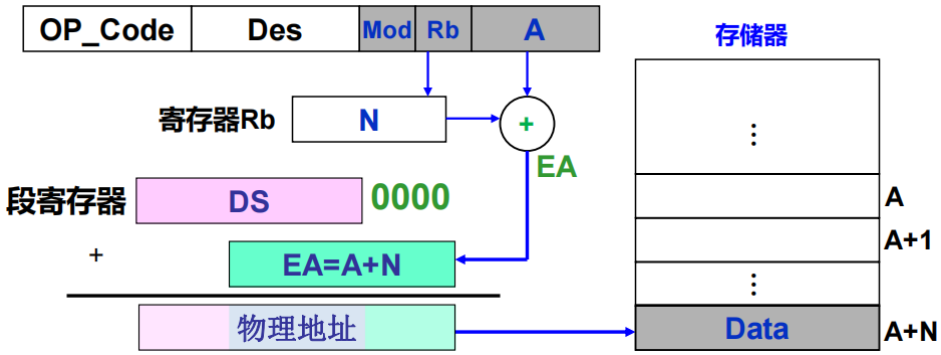
3️⃣基址寻址：引用一个基址寄存器 Rb(专用or通用，专用可缺省寄存器号), EA =(Rb)+A

Rb给出基址(程序设定)，A给出变址，操作数地址的变化由变址值(A)增减完成

2.5.7. 段寻址

适用于地址长度超过机器字长的时候，如8086

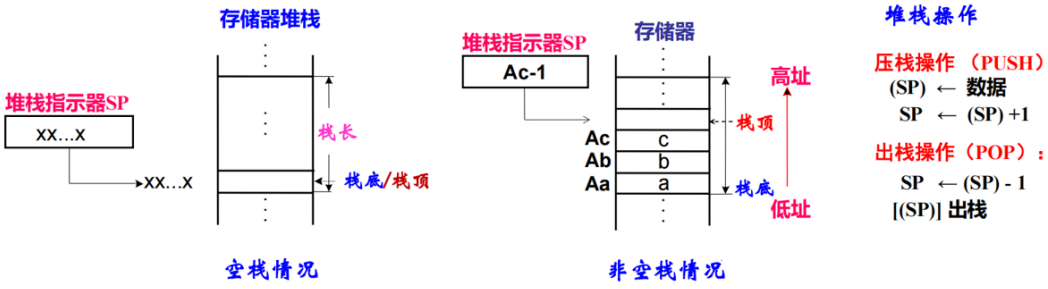
Des(操作数目的地)⬅️内存(操作数所在地)⬅️地址字段(寄存器给的偏移量+指定内存位置)⊕段寄存器



2.5.8. 堆栈寻址

1️⃣堆栈分为软堆栈(存储器中开一块区域，一端固定一端随IP指针浮动)，硬堆栈(寄存器中)

2️⃣堆栈的实现(下面是以SP指向栈顶空指针为例，其实也可以指向非空指针)



2.5.9. 复合寻址方式

例如：

1️⃣**变址间接寻址**：先变址后间接，如EA=[(Rx)+A]，先将RX的内容于A相加，然后指向一个存储器地址

2️⃣**间接变址寻址**：先间接后变址，如EA=(Rx)+(A)，先将RX指向一个存储器地址，然后在存储器上再偏移A个单位

PS：程序定位

1️⃣程序中指令/数据的逻辑地址到物理地址

2️⃣覆盖技术：程序空间>主存物理空间，在程序运行过程中逐段调入主存物理空间。覆盖技术

3️⃣定位方式：

1. **直接定位方式**：直接把主存地址编写程序，不需要转换
2. **静态定位方式**：在程序加载到主存时，一次性为指令和数据分配主存物理地址。由加载器定位，每次运行可装入不同物理空间，但每次运行时不可切换空间。覆盖技术
3. **动态定位方式**：程序执行过程中，进行逻辑地址到物理地址的转换，采用基址寻址实现