指令系统

笔记源文件：Markdown，长图，PDF，HTML

1. 指令的一般格式

1.1. 概述

1️⃣指令所需要实现的功能

1. 指明要实现何种操作(操作码)
2. 指明参与操作的数据类型(操作码)，以及该数据在哪(地址码)
3. 指明接下来执行什么指令(地址码)

2️⃣一般格式：`<操作码OP_Code> <地址码Addr_Code>`

3️⃣操作码：提供以下信息

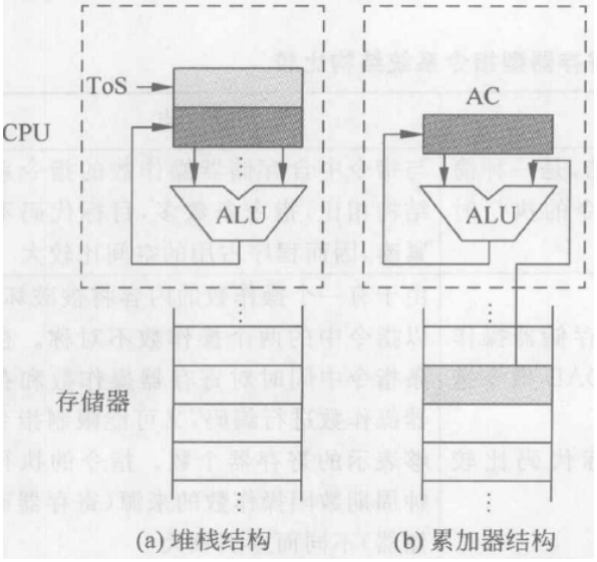
1. 指令的操作类型：是加减乘除，还是其余逻辑运算，指明一个
2. 操作数类型：
    - 存储设备(Reg/MM/IO)中存放的是纯数据，纯数据可以解释为不同类型/字长/进制
    - 如何解释，就是操作码说了算

4️⃣地址码

1. 涉及：IO设备地址—主存单元地址—寄存器地址
2. 三者对应：CPU结构—指令系统结构—地址码形式

1.2. CPU结构

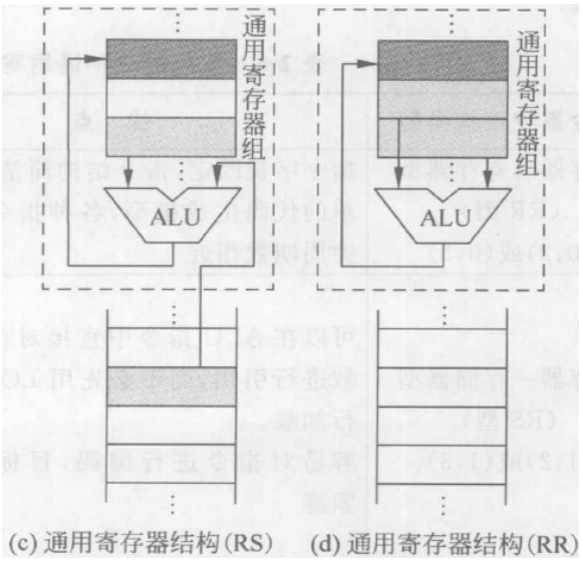
1.2.1. 古早结构



灰色代表源操作数，给色代表目的操作数

	栈式结构	累加器结构
源操作数	栈顶**(隐式)**+栈次顶**(隐式)**	累加器AC**(隐式)**+存储器**(显式，即指令中明确指定)**
目的操作数	栈顶	累加器AC
优点	指令较短，程序空间小	指令较短，程序空间小
缺点	无法随意访问堆栈	只有AC用于暂存中间结果，将频繁访问AC

1.2.2. 通用寄存器结构



(c) 通用寄存器结构(RS)　(d) 通用寄存器结构(RR)

灰色代表源操作数，给色代表目的操作数

	RR	RS	SS
源操作数(全部显式)	寄存器+寄存器	寄存器+存储器	存储器+存储器
目的操作数	寄存器	寄存器	寄存器
优点	指令字长固定，结构简单	目标代码紧凑	目标代码最紧凑
缺点	指令条数多	两操作数不对称	两操作数最不对称

2. 地址场结构

1 格式：注意$A1,A2,A3$是通用寄存器，涉及的专用寄存器如累加器AC

类型	`OP_Code`	`A1`	`A2`	`A3`
三地址指令	操作码	**源**操作数1地址	**源**操作数2地址	**目的**操作数地址

类型	OP_Code	A1	A2	A3
二地址指令	操作码	**源**操作数+**目的**操作数地址	**源**操作数地址	N/A
一地址指令	操作码	**源**操作数地址	N/A	N/A
零地址指令	操作码	N/A	N/A	N/A

2️⃣功能

类型	功能	特点
三地址指令	$A3 \leftarrow (A1)\ OP\ (A2)$	两个源操作数内容不变；指令字长太长
二地址指令	$A1 \leftarrow (A1)\ OP\ (A2)$	砍掉一个地址码字段后寻址范围增大，扩展性增强
一地址指令	$AC \leftarrow (AC)\ OP\ (A)$	隐式默认一个操作数在特定位置，如AC
零地址指令	堆顶 $\leftarrow$ (堆顶) OP (堆次顶)	出左边所说，还可用于停机/空操作/隐式单操作数

3. 操作码扩展技术

3.1. 定长/变长操作码

操作码类型	本质特征	特点	举例
定长操作码	操作码长度不变，位置不变	硬件设计方便，译码简单	MIPS32
变长操作码	操作码长度可变，位置可变	控制器设计复杂，译码复杂，可扩展	IA-32

3.2. 扩展技术

1️⃣核心思想：地址码较少时，操作码占用更多位

2️⃣等长扩展：典型如操作码按照4位/8位/12位依次扩展

3️⃣两种4-8-12扩展：

指令格式及操作码编码				说明
OP_Code	A1	A2	A3	
0000 0001 … 1110				4位操作码的三地址指令共15条
OP_Code		A1	A2	
1111 0000 1111 0001 … … 1111 1110				8位操作码的二地址指令共15条
OP_Code			A	
1111 1111 0000 1111 1111 0001 … … … 1111 1111 1110				12位操作码的一地址指令共15条
OP_Code				
1111 1111 1111 0000 1111 1111 1111 0001 … … … … 1111 1111 1111 1111				16位操作码的零地址指令共16条

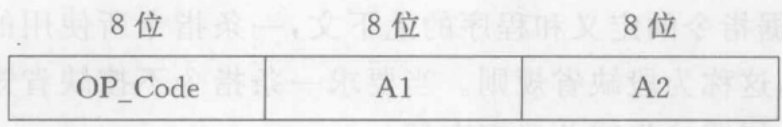
(a) 等长15/15/15/16扩展法

指令格式及操作码编码				说明
OP_Code	A1	A2	A3	
0000 0001 … 0111				4位操作码的三地址指令共8条
OP_Code		A1	A2	
1000 0000 1000 0001 … … 1111 0111				8位操作码的二地址指令共64条
OP_Code			A	
1000 1000 0000 1000 1000 0001 … … … 1111 1111 0111				12位操作码的一地址指令共512条
OP_Code				
1000 1000 1000 0000 1000 1000 1000 0001 … … … … 1000 1000 1000 1111				16位操作码的零地址指令共4096条

(b) 等长8/64/512/4096扩展法

3.3. 例题

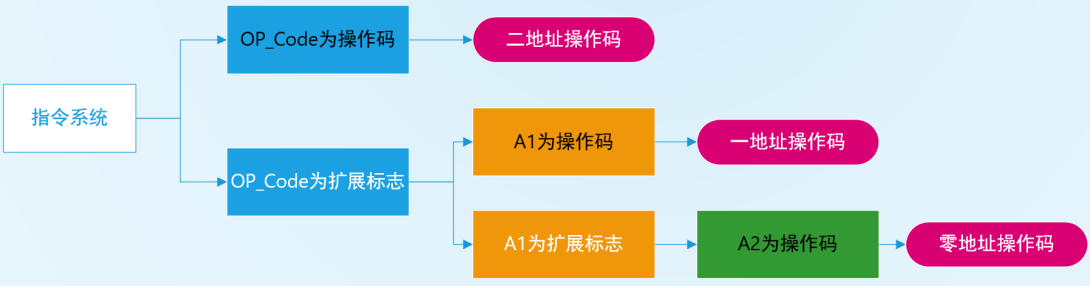
1 条件



1. 某机器二地址格式为：`<OP_Code_8位><A1_8位><A2_8位>`
2. 该机器只有三种指令：二地址/一地址/零地址
3. 扩展前操作码(基本操作码)为8位
4. 目前已设计出：m条二地址指令，n条零地址指令

2 计算：一地址指令数h的最大值

1. 画出扩展的图谱



2. m条二地址指令：
   - OP_Code有m个用作操作码，2^8-m个用作扩展标志
   - 前16位用于零地址/一地址，共有$(2^8-m)\times 2^8$种可能
3. n条零地址指令：
   - 前16位中一种扩展标志$\overset{对应}{\longleftrightarrow}$A2中$2^8$种操作码$\overset{对应}{\longleftrightarrow}2^8$种零地址操作码
   - 所以前16位中共抽离了$[\frac{n}{2^8}]$种可能作为零地址操作的标志，剩下的就是用作一地址的了
4. 所以$h=(2^8-m)\times 2^8-[\frac{n}{2^8}]$

4. 寻址方式

4.1. 两种寻址

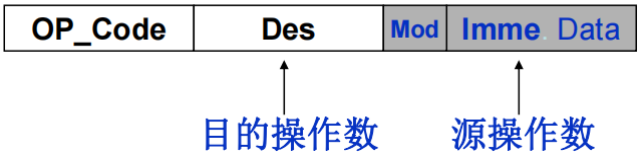
1️⃣指令寻址

1. 目的：确定下个执行的指令，所在主存单元的地址
2. 方式：顺序PC+1，跳跃JMP

2️⃣数据寻址

1. 格式：`<操作码> <寻址方式码> <形式地址1><形式地址2>.....<形式地址n>`
2. 目的：形式地址(符号地址) $\xrightarrow{\text{寻址方式}}$ 有效地址(逻辑地址)
3. 种类：立即寻址、寄存器寻址、存储器寻址、堆栈寻址

4.2. 立即寻址：EA=Imme Data

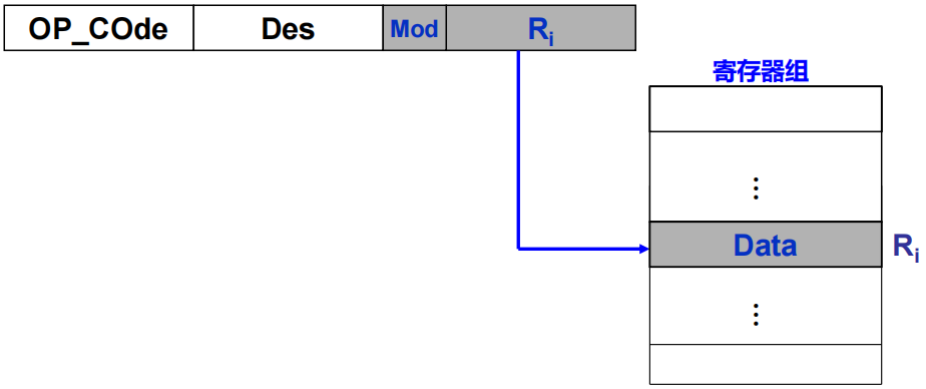


Des/目的操作数 $\xleftarrow{\text{赋值}}$ Imme Data/源操作数

```
mov eax,100
```

4.3. 寄存器寻址：EA=Ri

1️⃣寄存器直接寻址



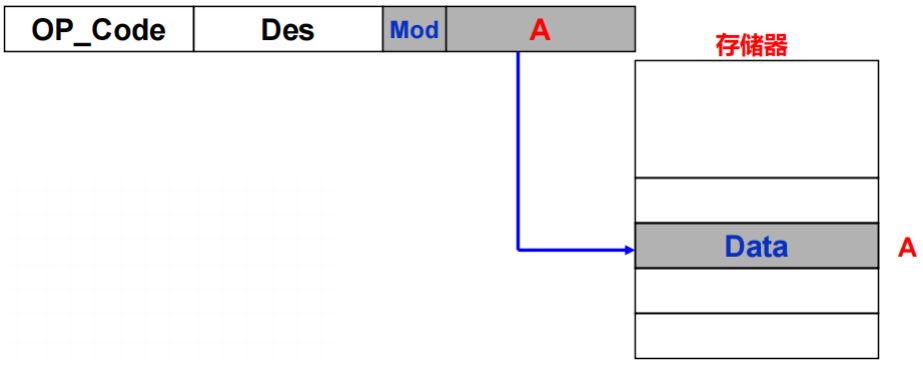
- R_i/通用寄存器号 $\xrightarrow{\text{找到}}$ Data/通用寄存器组/源操作数
- Des/目的操作数 $\xleftarrow{\text{赋值}}$ Data/通用寄存器组/源操作数

```
mov eax ebx
```

2️⃣寄存器间接寻址：这实质上是一种存储器寻址，见下

4.4. 存储器(主存)寻址

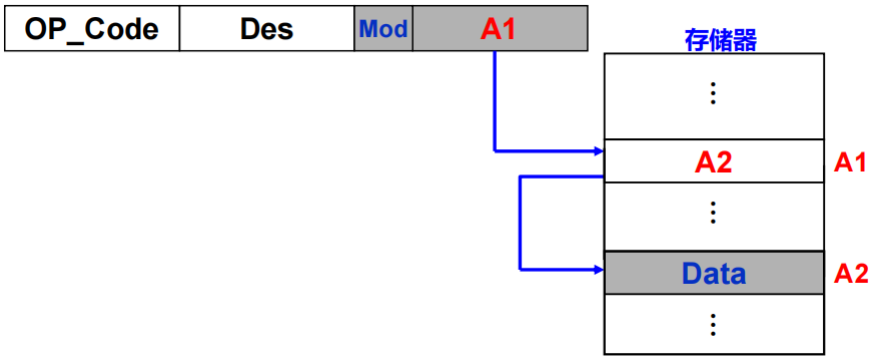
1️⃣直接寻址：EA=A



- A/主存单元号 $\xrightarrow{找到}$ Data/主存单元/源操作数
- Des/目的操作数 $\xleftarrow{赋值}$ Data/主存单元/源操作数

```
mov eax [100]
```

2️⃣一次存储器间接寻址：EA=(A1)

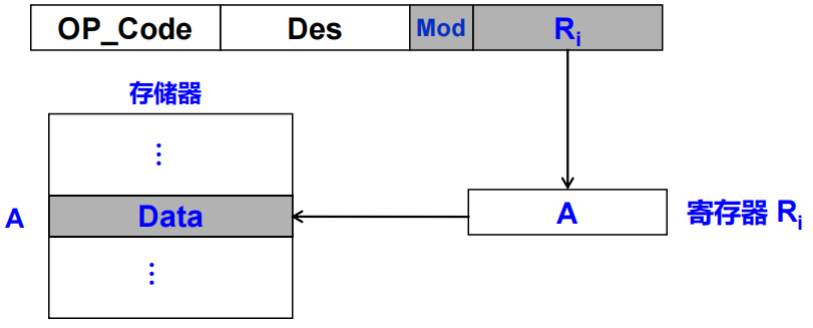


- A1/主存单元号 $\xrightarrow{找到}$ A2/主存中/另一个主存单元号 $\xrightarrow{找到}$ Data/主存中/源操作数
- Des/目的操作数 $\xleftarrow{赋值}$ Data/主存中/源操作数

3️⃣二次存储器间接寻址：EA=((A1))

1. 寻址范围：
    - A1主存单元：主存字长为m，可寻址2^m个单元
    - A2及以后主存单元：主存字长为m，但首尾为间址标志位，可再寻址2^{m-1}个单元
    - 一次可间址2^m个单元，n次可间址$2^m \times (n-1)2^{m-1}$个单元
2. 间址过程：不断在主存单元跳，跳到主存单元的间址标志位为0时，停止间址取出Data
3. 严禁套娃：否则执行速度巨慢

4️⃣寄存器间接寻址：EA=(Ri)

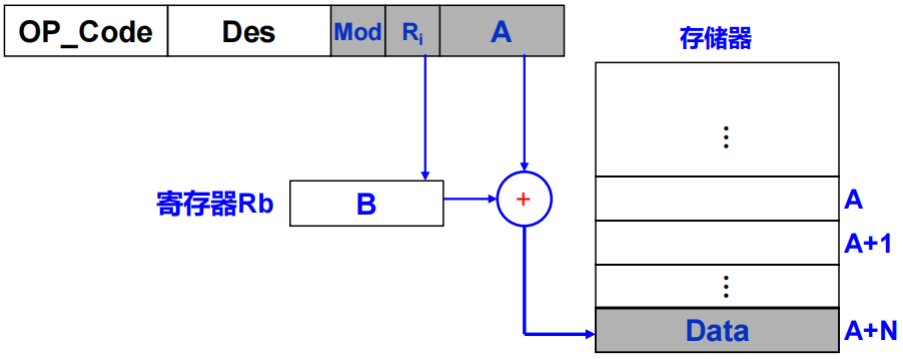


- R_i/通用寄存器号 $\xrightarrow{找到}$ A/通用寄存器 $\xrightarrow{找到}$ Data/存储器/源操作数

- Des/目的操作数 $\xleftarrow{\text{赋值}}$ Data/存储器/源操作数

```
mov eax [ebx]
```

5️⃣偏移寻址：EA=B(基地址)+A(偏移量)

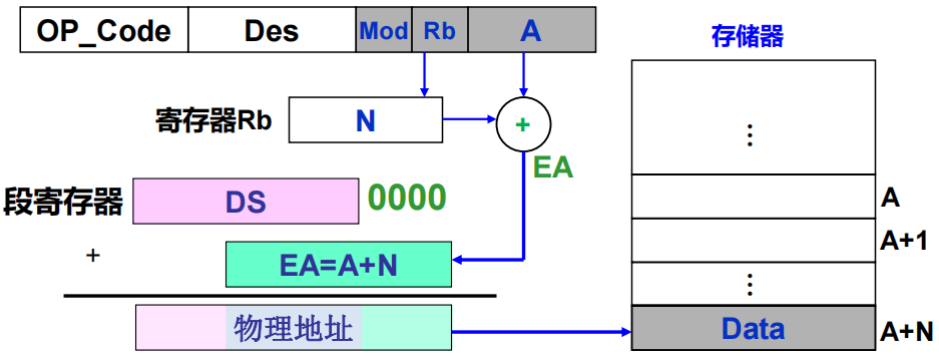


1. 寻址过程：
   - R_i/通用寄存器号 $\xrightarrow{\text{找到}}$ B/通用寄存器/基地址 $\xrightarrow[\text{加上}A\text{/偏移量}]{\text{找到}}$ Data/主存单元/源操作数
   - Des/目的操作数 $\xleftarrow{\text{赋值}}$ Data/主存单元/源操作数
2. 类型

类型	操作数EA	偏移量	基地址	特性	用途
相对寻址	(PC)+A	A	(PC)	用专门PC指针，指令只给出A	处理跳转
变址寻址1	(Rx)+A	(Rx)	A	引用专用变址寄存器，为默认	处理数组
变址寻址2	(Ri)+A	(Ri)	A	引用通用变址寄存器，需指定	同上
基址寻址	(Rb)+A	A	(Rb)	引用基址寄存器	逻辑→物理地址

3. 寻址范围：由偏移量决定
   - 相对寻址：A共k位时，$\mathrm{PC}-2^{k-1}\rightarrow\mathrm{PC}+2^{k-1}-1$
   - 变址寻址：(Rx)为n位时，寻址大小为2^n字
   - 基址寻址：A共k位时，$\mathrm{Rb}-2^{k-1}\rightarrow\mathrm{Rb}+2^{k-1}-1$
4. 补充：基址寄存器设定基址后，不可改变

6️⃣段寻址：EA=N+A 物理地址=DS × 16+EA

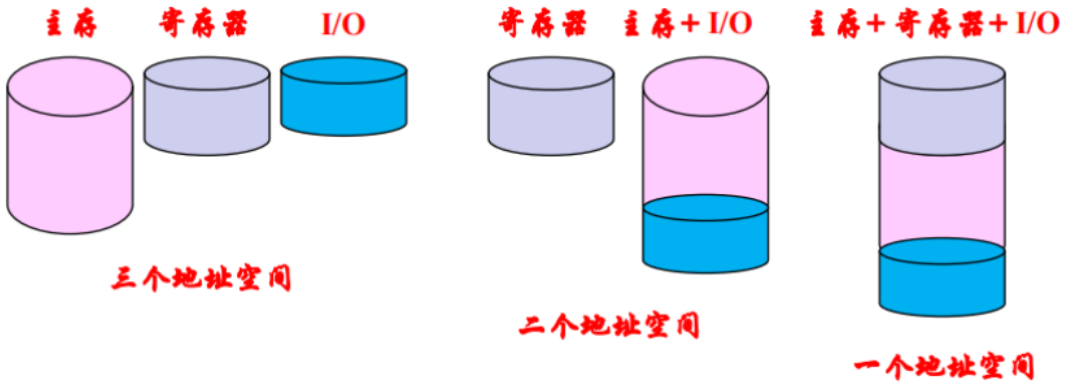


1. 核心思想：
    - 将主存化为多个逻辑段，一个程序可以横跨多个段
    - 访问操作数时，先指定在哪个段，再在对应段内偏移
2. 寻址过程
    - $\text{Rb/寄存器号} \xrightarrow{\text{找到}} \text{N/通用寄存器} \xrightarrow[\text{加上}A]{\text{找到}} \text{EA/有效地址/段内偏移}$
    - $\text{DS/段寄存器} \xrightarrow{\text{左移4位}} \text{DS} \times 16\text{/段基地址}$
    - $\text{DS} \times 16\text{/段基地址} 0 + \text{EA/有效地址/段内偏移} \xrightarrow{\text{找到}} \text{Data/主存单元/物理地址/源操作数}$
    - $\text{Des/目的操作数} \xleftarrow{\text{赋值}} \text{Data/主存单元/源操作数}$
3. 补充：整个过程硬件自动完成，对用户透明

4.5. 外设寻址：关注两种编址方式

1️⃣独立编址：主存，寄存器，IO的地址的开头都是0，如8086(左)

2️⃣统一编制：寄存器从0开始，主存从64开始，IO从128开始



4.6. 堆栈寻址

4.6.1. 关于堆栈

1️⃣两种堆栈

1. 硬堆栈/寄存器堆栈
2. 软堆栈/存储器堆栈：主存中开辟一块地，栈底固定，栈顶通过SP指针浮动

2️⃣堆栈顶/底的设定

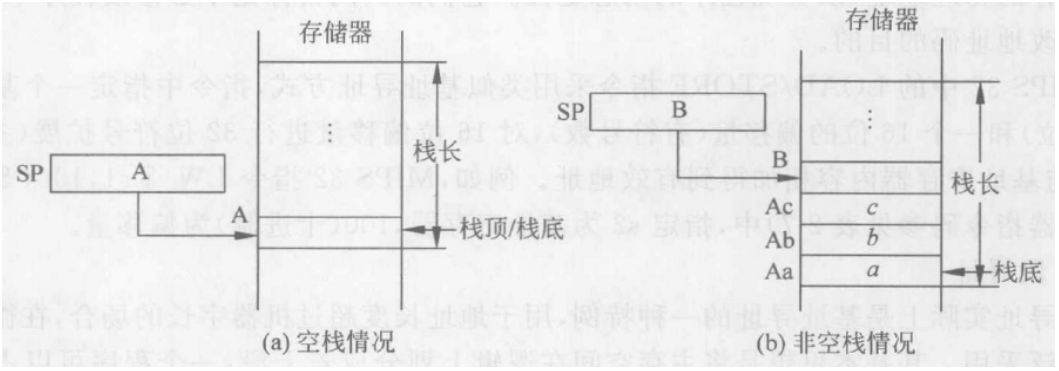
1. 栈底在低地址，栈顶在高地址，栈向上生长
2. 栈底在高地址，栈顶在低地址，栈向下生长

3 栈顶的确认

1. SP指向栈顶+1处的空单元
2. SP指向栈顶处的非空单元

4.6.2. 两种堆栈寻址

1 SP指向栈顶+1处的空单元



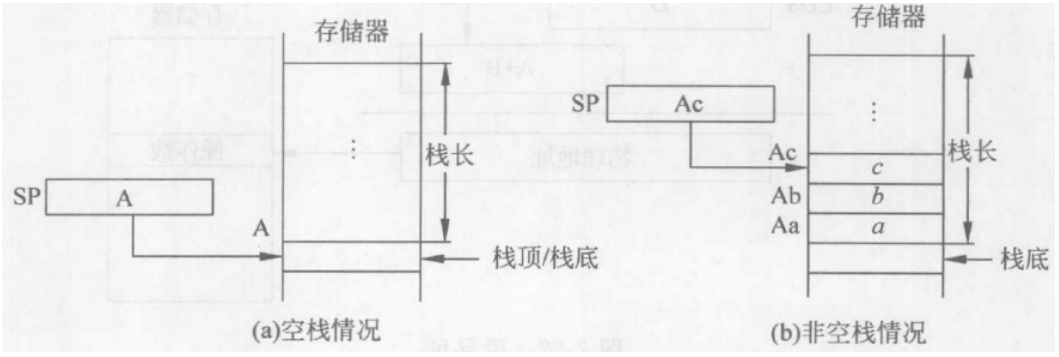
1. 空栈：栈底和栈顶为同一单元
2. 入栈

```
(SP)←数据
SP←(SP)+1(向上生长)
SP←(SP)-1(向下生长)
```

3. 出栈

```
SP←(SP)-1(向上生长)
SP←(SP)+1(向下生长)
[(SP)]清空出栈
```

2 SP指向栈顶处的非空单元



1. 空栈：栈底和栈顶为同一单元
2. 入栈

```
SP←(SP)+1(向上生长)
SP←(SP)-1(向下生长)
(SP)←数据
```

3. 出栈

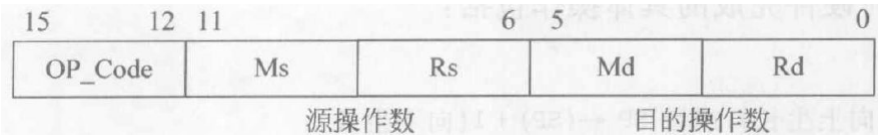
```
[(SP)]清空出栈
SP←(SP)-1(向上生长)
SP←(SP)+1(向下生长)
```

4.6. 例题

4.6.1. 条件

1️⃣机器字长16位，主存大小128KB，但是按字编址(即$64\mathrm{K} \times 16$位)

2️⃣采用字长指令格式，各字段为



3️⃣寻址方法

$\mathrm{Ms/Md}$	指令格式	助记符	含义
$000\mathrm{B}$	寄存器直接	Rn	操作数=(Rn)
$001\mathrm{B}$	寄存器间接	(Rn)	操作数=$((\mathrm{Rn}))$
$010\mathrm{B}$	寄存器间接、自增	$(\mathrm{Rn}+)$	操作数=$((\mathrm{Rn}))$, $\mathrm{Rn} \leftarrow (\mathrm{Rn})+1$
$011\mathrm{B}$	相对	$\mathrm{D}(\mathrm{Rn})$	转移目标地址=$(\mathrm{PC})+(\mathrm{Rn})$

补充：转移指令采用相对寻址，相对偏移采用补码表示

4.6.2. 第一问

❓问题

1. 该指令系统最多可有多少条指
2. 该计算机最多有多少个通用寄存器
3. 存储器地址寄存器(MAR)需要多少位，(MAR)用于接收CPU传输的地址信号
4. 存储器数据寄存器(MDR)需要多少位，(MDR)用于和CPU交互数据

👉解答

1. 假设$\mathrm{OP_Code}$全部用于设计指令，最多有$2^4 = 16$条
2. Rs, Rd是用于寄存器寻址的，故最多有$2^3 = 8$个寄存器
3. 思路如下
    - 主存128K用字编址就是64K×16位，每16位一字/一字一个存储单元
    - 主存中一共64K=2^{16}个单元，所以需要16位才能编址主存单元
    - 所以CPU传输的地址信号是16位的，也就是(MAR)是16位的
4. CPU一次性能处理16位数据(机器字长)，所以(MDR)也需要16位

4.6.3. 第二问

❓问题：转移指令的目标地址范围是多少?

👉解答：

1. 相对寻址：EA=(PC)+(Rn)，(Rn)用补码表示且共k位时，
   $\mathrm{PC}-2^{k-1} \rightarrow \mathrm{PC}+2^{k-1}-1$
2. 代入即可得：$-2^{k-1} \rightarrow +2^{k-1}-1$，Rn的位数等于CPU字长，代入$k=16$即可

4.6.4. 第三问

⬅️前提

1. OP_Code=0010B表示加法操作，用助记符ADD表示
2. 对于寄存器R4,R5

寄存器	寄存器编号	寄存器内容	地址单元的内容
R4	100B	(R4)=1234H	地址1234H存储内容((R4))=5678H
R5	101B	(R5)=5678H	地址5678H存储内容((R5))=1234H

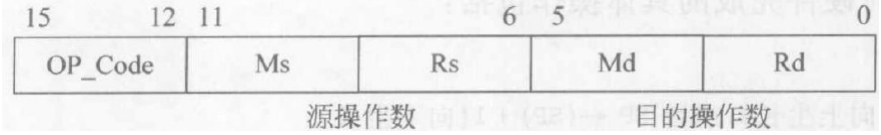
3. 给定汇编指令：ADD (R4),(R5+) 逗号前表示表示源操作数，逗号后表示目的操作数

❓问题：

1. 求出给出汇编指令的16进制机器码
2. 执行该指令后，有哪些寄存器/内存单元会改变？给出改变后的内容

👉解答：

1. 机器码：照着这个指令格式填充



   - OP_Code=0010B
   - (R4)表示寄存器间接，所以参照寻址表Ms=001B
   - (R4)对应的寄存器号是100B，所以Rs=100B
   - (R5+)表示寄存器间接寻址切自增长，所以Md=010B
   - (R5)对应的寄存器号是101B，所以Rd=101B

   合并为0010 0011 0001 0101B = 2315H

2. 寄存器/内存单元的改变
   - R4的寻址：

     源操作数=((R4))=(1234H)=5678H
   - R5的寻址：

     目的操作数=((R5))=(5678H)=1234H，并且(R5)=(R5)+1=5679H
   - 相加：

$$(5678H)单元新值=(5678H)单元旧值+(1234H)单元值=1234H+5678H=68ACH$$

$$所以(R5)\xrightarrow{变成}5679H，(5678H)\xrightarrow{变成}68ACH$$

5. CISC和RISC

5.1. CISC

1️⃣主要特征

1. 用一条功能复杂的指令，完成一串指令的功能，目的在于减少取指频率
2. 用复杂寻址&指令，来支持高级语言的高效实现
3. 向上向后兼容，新机器支持所有旧机器的指令，指令集不断扩大
4. 采用微程序控制技术设计控制器

2️⃣弊端

1. 指令和寻址太复杂，编译复杂，难以优化
2. 大多指令太复杂，几个时钟周期才能执行完
3. 微程序控制器影响速度
4. 各种指令使用的频率悬殊很大

5.2. RISC

1️⃣相较于CISC的二元对立特征

1. 指令长度固定，指令格式种类少，寻址方式种类少
2. 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)
3. 优选CISC中使用频率高的简单指令，其他扔掉
4. 控制单元以硬布线逻辑为主

2️⃣其他特征

1. 通用寄存器相当多
2. CPU采用流水线结构，CPI低
3. 采用编译优化

6. 指令系统示例：MIPS32

类型	31_26位	25_21位	20_16位	15_11位	10_6位	5_0位
R	操作码	源操作数1(rs)	源操作数2(rt)	目的操作数(rd)	移位位数	功能码
I(立即数)	操作码	源操作数1(rs)	目的操作数(rt)	立即数	立即数	立即数
J(跳转)	操作码	立即数	立即数	立即数	立即数	立即数

1️⃣ 对于R型指令

1. 31_26位全0(special类型)，需要再根据5_0位决定指令类型
2. 对于R型指令，10_6位为移位位数，当指令不涉及位移时全部置0

2️⃣ 对于I型指令：直接由5_0位决定指令类型

3️⃣ 对于J型指令：

1. `j`：无条件跳转
2. `jal`：无条件跳转并链接
3. `jr`：寄存器跳转(其实这个指令是R型的)
4. `jalr`：寄存器跳转并链接(其实这个指令是R型的)