汇编语言

1. 基础知识

1.1机器语言

·机器语言：机器指令的集合

机器指令：一台机器可以正确执行的命令

1.2汇编语言的产生

1.3汇编语言的组成

·汇编语言有以下3类指令组成：

(1.汇编指令：有对应机器码，机器码的助记符

(2.伪指令：没有对应机器码，由编译器执行，计算机并不执行

(3.其他符号：没有对应机器码，如+，-，\*，/等，由编译器识别

1.4存储器

1.5指令和数据

1.6存储单元

·最小信息单元：1bit

·存储单元：1Byte = 8bit

1.7 CPU对存储器的读写

1.8地址总线

·每个CPU都有许多管脚（与总线相连）

·3种总线的宽度标志了这个CPU的不同方面的性能

(1.地址总线 => 寻址能力

(2.数据总线 => 与其他器件进行数据传送时一次性数据传送量

(3.控制总线 => 对系统中其他器件控制能力

1.9数据总线

1.10控制总线

1.11内存地址空间

·系统中所有存储器中的存储单元都处于一个统一的逻辑存储器中，这个逻辑存储器即是我们所说的内存地址空间

1.12主板

1.13接口卡

1.14各类存储器芯片

1.15内存地址空间

1. 寄存器

·在CPU中：

(1.运算器：进行信息处理

(2.寄存器：进行信息存储

(3.控制器：控制各种器件进行工作

(4.内部总线：连接各种器件，在它们之间进行数据的传送

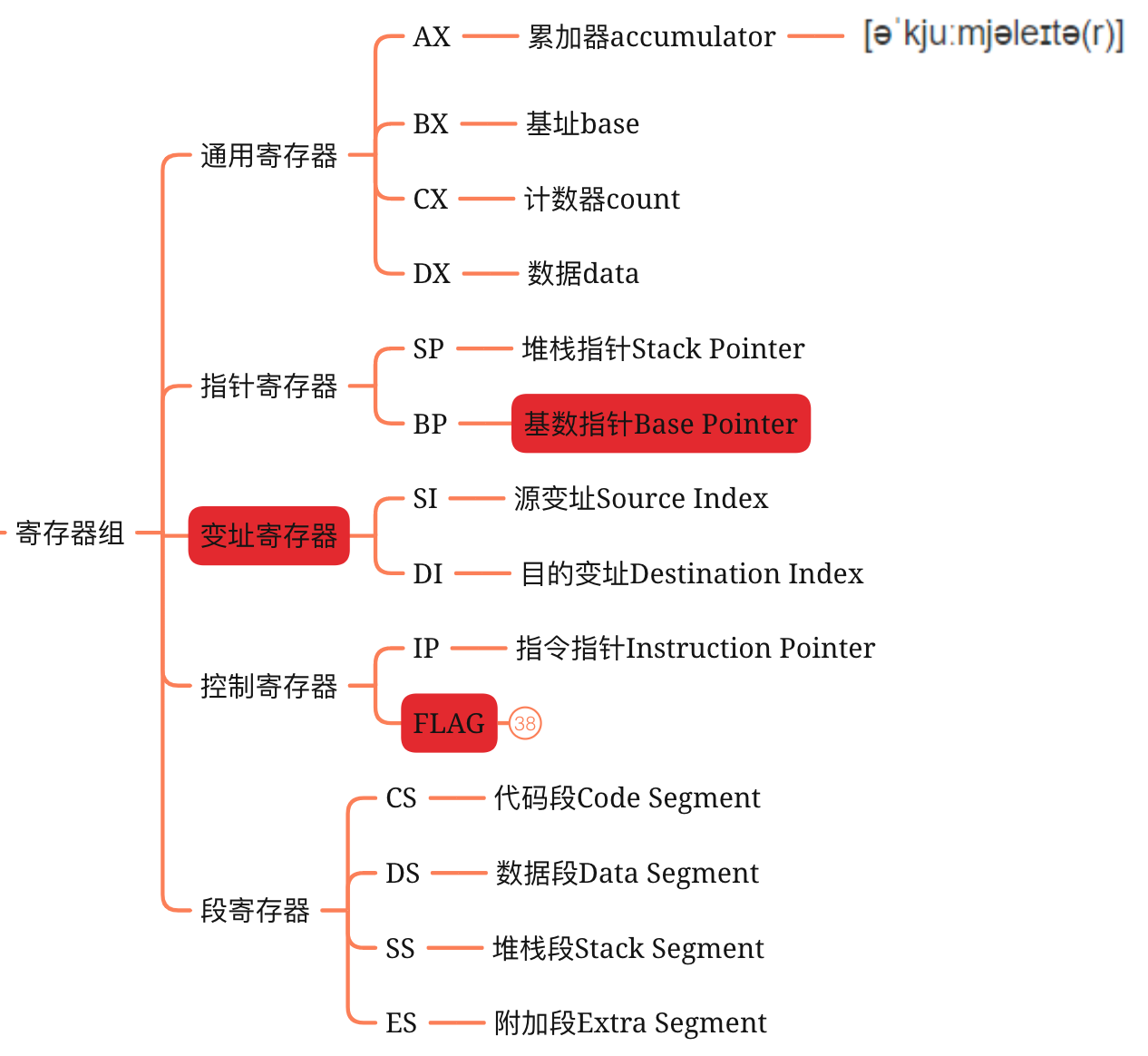


图1·寄存器组名称及分类

2.1通用寄存器

2.2字在寄存器中的存储

2.3几条汇编指令

·add ax,bx 其中ax和bx都为8226H

最高位的1不能在ax中保存

·add al,93H 其中al=C5H

进位值不能在8位寄存器中保存

·在进行数据传送或运算时，两个操作对象的位数应当是一致的

错误指令

mov ax,bl ·mov bh,ax ·mov al,20000 ·add al,100H

2.4物理地址

2.5 16位结构的CPU

2.6 8086CPU给出物理地址的方法

2.7段地址\*16+偏移地址=物理地址的本质含义

2.8段的概念

2.9段寄存器

·16位地址的寻址能力为64KB，所以一个段的最大长度为64KB

2.10 CS和IP

·FFFF：0000是8086PC机开机后执行的第一条指令

2.11修改CS，IP的指令

·传送指令：能修改除了CS，IP的大部分寄存器的值，如mov指令

·转移指令：能够改变CS，IP的内容的指令

·8086CPU的工作过程：

(1.从CS：IP指向的内存单元读取指令，读取的指令进入指令缓冲器；

(2.IP指向下一条指令；

(3.执行指令；（回到步骤一，重复）

2.12 代码段

1. 寄存器（内存访问）

3.1内存中字的存储

高地址内存单元中存放字型数据的高位字节，低地址内存单元中存放字型数据的低位字节

·地址单元

·地址字单元

3.2 DS和[address]

3.3字的传送

3.4 mov,add,sub指令

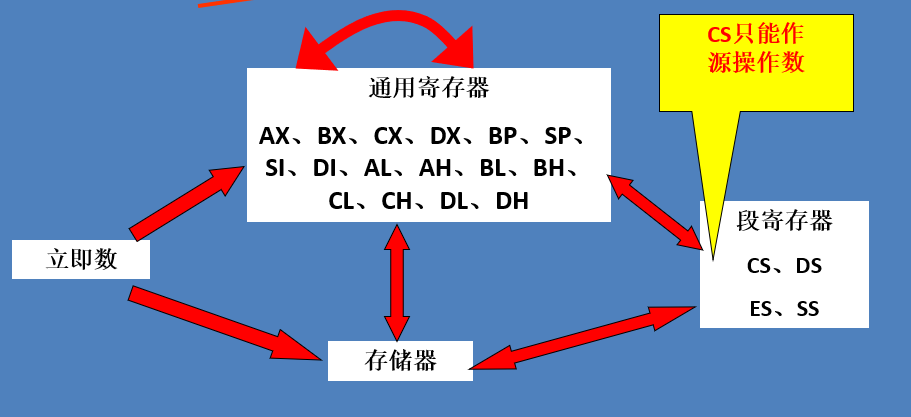


图2.数据传送指令mov操作对象

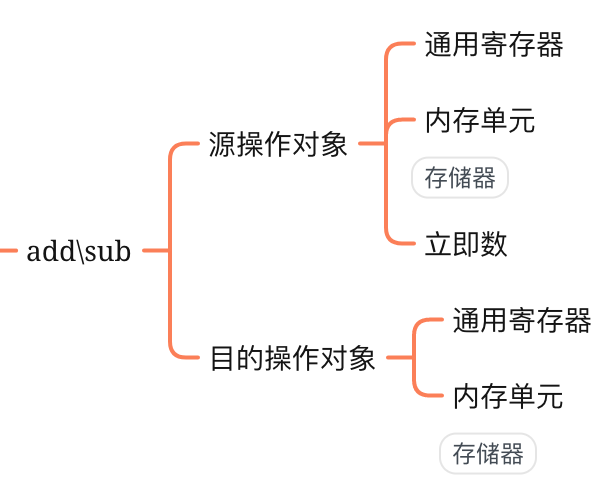


图3.add,sub算数运算指令操作对象

3.5数据段

·就算在开头加上了assume ds：data，代码段中也不能少了mov ax,data，mov ds,ax

assume只是给程序员看的

3.6栈

3.7 CPU提供的栈机制

·任意时刻，SS:SP指向栈顶元素

8086CPU中，入栈时，栈顶从高地址向低地址方向增长

3.8栈顶超界的问题

·8086CPU不保证我们对栈的操作不会超界，需要自己操心

3.9 push,pop指令

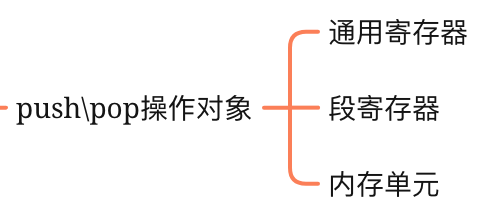


图4·push\pop操作对象

3.10栈段

·将10000H～1FFFFH当作栈段，初始栈段是空的，此时SS=1000H，SP=0000H

栈空，相当于栈中唯一元素出栈，SP=FFFEH+2=0000H；

·一段内存，可以既是代码的存储空间，又是数据的存储空间，还可以是栈空间，也可以什么都不是。关键在于CPU中寄存器的设置，即CS,IP,SS,SP,DS的指向

1. 第一个程序

4.1一个源程序从写出到执行的过程

4.2源程序

4.3编辑源程序（源程序文件）

4.4编译（目标文件）

4.5连接（可执行文件）

4.6以简化的方式进行编译和连接

4.7 1.exe的执行

4.8 谁将可执行文件中的程序装载进入内存并使它运行

·汇编程序从写出到执行的过程：

编程Edit—>1.asm—>编译masm—>1.obj—>连接link—>1.exe—>加载command—>内存中的程序—>运行CPU

4.9程序执行过程的跟踪

·debug（调试）将程序从可执行文件加载入内存后，cx存放的是程序的长度

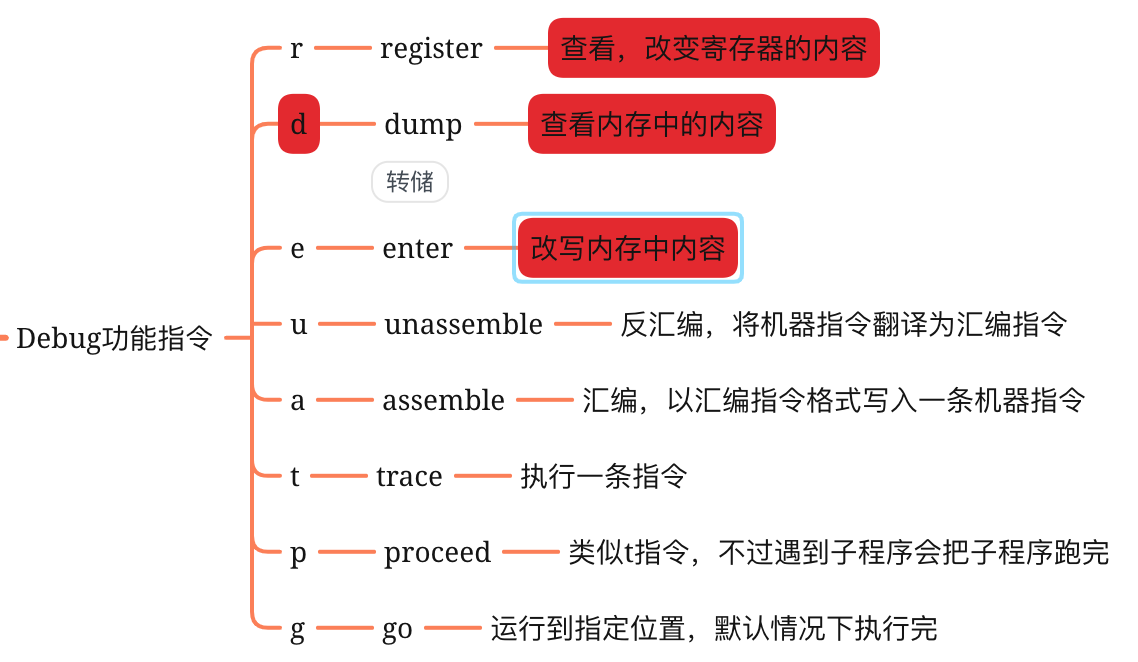


图5·debug功能指令

1. [BX]和loop指令

·(ax)表示ax中的内容

5.1 [BX]

5.2 LOOP指令

5.3在Debug中跟踪用loop指令实现的循环程序

·汇编程序中，数据不能以字母开头，所以A000H在汇编程序中应该表示为“0A000H”

5.4 Debug和汇编编译器masm对指令的不同处理

·eg: mov ax,2000H mov ds,ax

Debug中mov al,[0]与 masm中有很大区别

masm中需mov al,ds:[0]

即没有显示给出[ ]前的段寄存器，则编译器masm将把指令中的[idata]解释为idata

5.5 loop和[bx]

5.6段前缀

5.7一段安全的空间

·DOS方式下，一般情况，0:200～0:2ff空间中没有系统或其他程序的数据或代码

5.8段前缀的使用

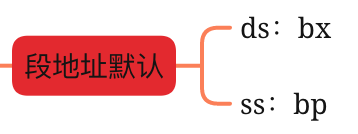


图6.段地址默认

六、包含多个段的程序

6.1在代码段中使用数据

·dw 定义字型数据 ·dd 定义双字型数据

·若要CPU从何处开始执行程序，只要在源程序中使用“end 标号”指明就可以了

eg：end start指明程序入口在标号start处

6.2在代码段中使用栈

6.3将数据、代码、栈放入不同的段

·段名相当于一个立即数

mov ds,data是错误的

1. 更灵活的定位内存地址的方法

7.1 and和or指令

7.2 关于ASCII码

7.3以字符形式给出的数据

7.4大小写转换问题

·大写字母与小写字母ASCII码的唯一区别第5位（0～7）为0

大写字母+20H=小写字母

7.5 [bx+idata]

7.6 用[bx+idata]的方式进行数组的处理

7.7 SI和DI

·用16位寄存器进行内存单元之间的数据传递，一次性复制2个字节

7.8 [bx+si]和[bx+di]

7.9 [bx+si+idata]和[bx+di+idata]

1. 数据处理的两个基本问题

8.1 bx,si,di和bp

8.2机器指令处理的数据在什么地方

8.3汇编语言中数据位置的表达

8.4寻址方式

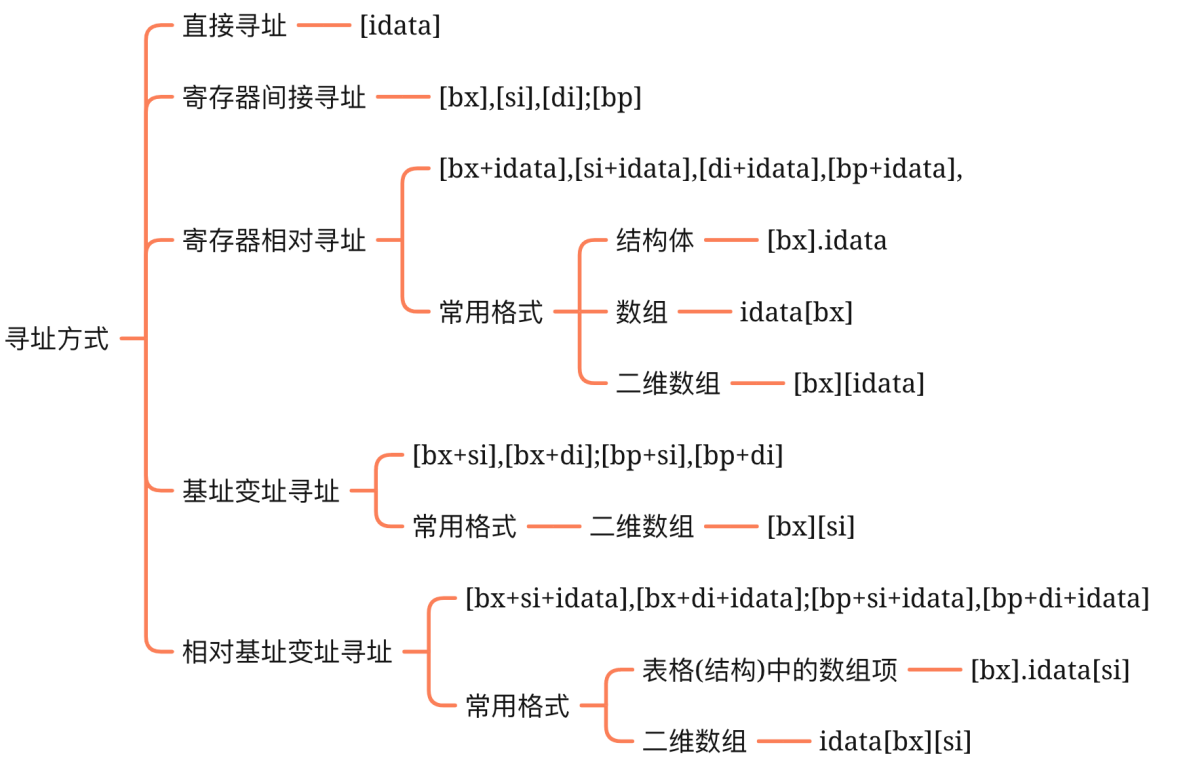


图7·寻址方式小结

8.5指令要处理的数据有多长

3种方式

(1.寄存器名指明

(2.X ptr：在没有寄存器名存在的情况下，指明内存单元的长度

(3.指令默认：比如push指令默认字操作

8.6寻址方式的综合运用

8.7 div指令

8.8伪指令dd

8.9 dup

·和db,dw,dd等数据定义伪指令配合使用，用来进行数据的重复

比如db 3 dup (‘abc’,’ABC’)定义了18个字节

1. 转移指令的原理

可以修改IP，或同时修改CS和IP的指令统称位转移指令

9.1操作符offset

9.2 jmp指令

9.3 依据位移进行转移的jmp指令

·CPU在执行jmp指令的时候并不需要转移的目的地址

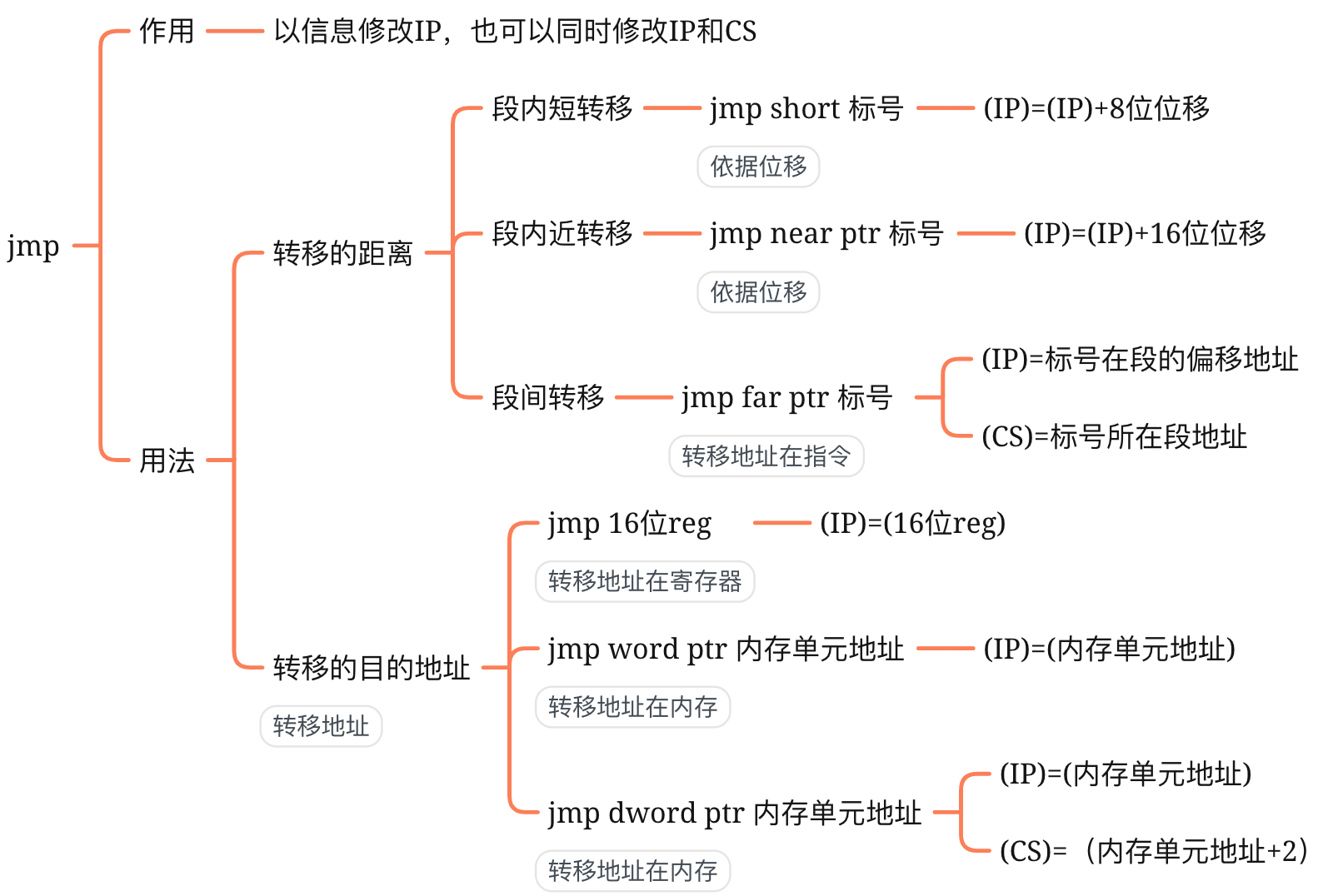


图8·jmp指令用法

9.4转移的目的地址在指令中的jmp指令

9.5转移地址在寄存器中的jmp指令

9.6转移地址在内存中的jmp指令

9.7 jcxz指令

·所有有条件转移指令都是短转移，在对应机器码中包含转移的位移，而不是目的地址

9.8 loop指令

·所有循环指令都是短转移；

9.9 根据位移进行转移的意义

9.10编译器对转移位移超界的检测

·越界会报错

·jmp 2000:0100是在debug中使用的汇编指令，汇编编译器并不认识

1. CLAA和RET指令

10.1 ret和retf

10.2 call指令

·call指令不能实现短转移；

10.3依据位移进行的call指令

10.4转移的目的地址在指令中的call指令

10.5 转移地址在寄存器中的call指令

10.6转移地址在内存中的call指令

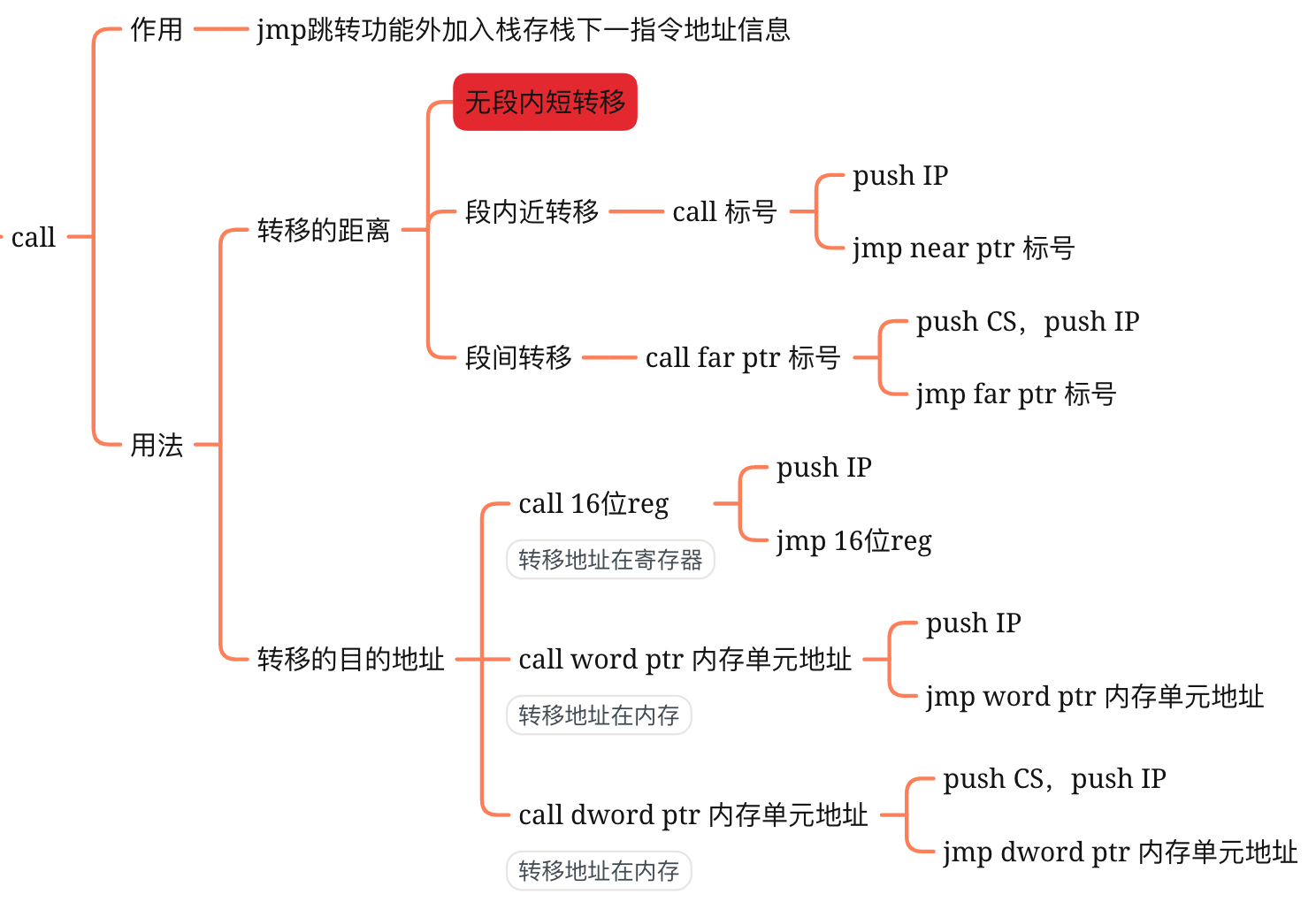


图9·call指令用法

10.7 call和ret的配合使用

10.8 mul指令

10.9模块化程序设计

10.10参数和结果传递的问题

10.11 批量数据的传递

10.12 寄存器冲突的问题

1. 标志寄存器

·8086CPU的flag寄存器

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  |  | OF | DF | IF | TF | SF | ZF |  | AF |  | PF |  | CF |

表1·FLAG标志位

记忆：CPAZS（TI）DO “裁判爱郑爽对哦” 0，2，6，7，10，11；

11.1 ZF标志

11.2 PF标志

11.3 SF标志

11.4 CF标志

11.5 OF标志



图10·重要标志位

·其中左边为1，右边为0；比如DF：减少为1，增加为0；

11.6 adc指令

11.7 sbb指令

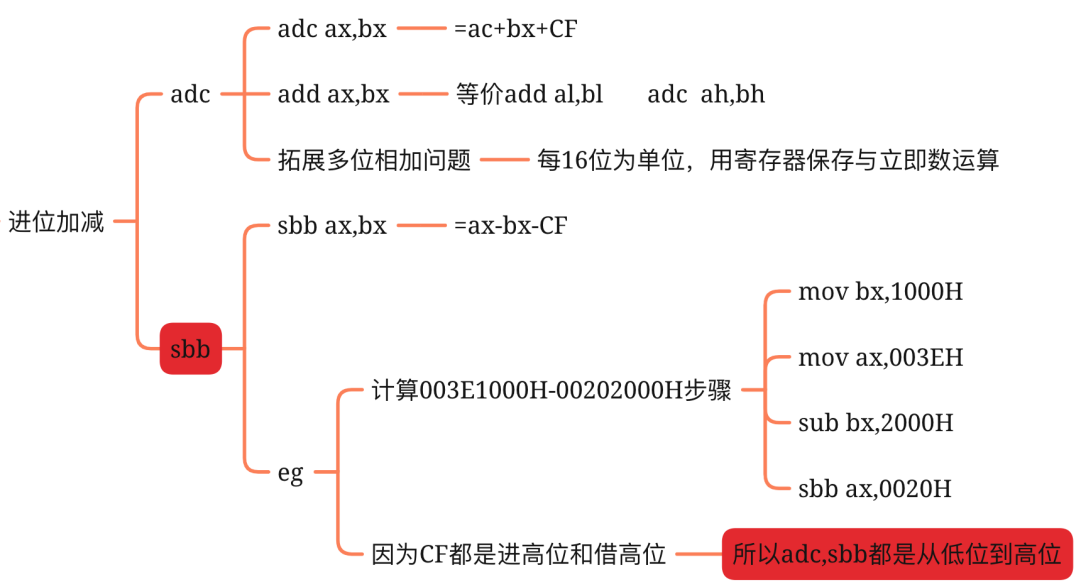
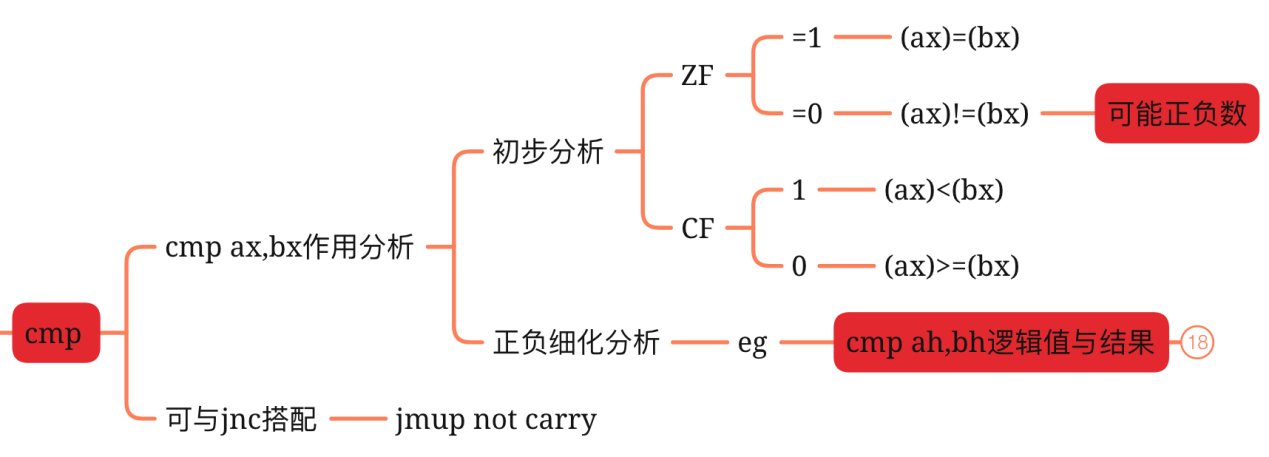


图11.进位加减

11.8 cmp指令



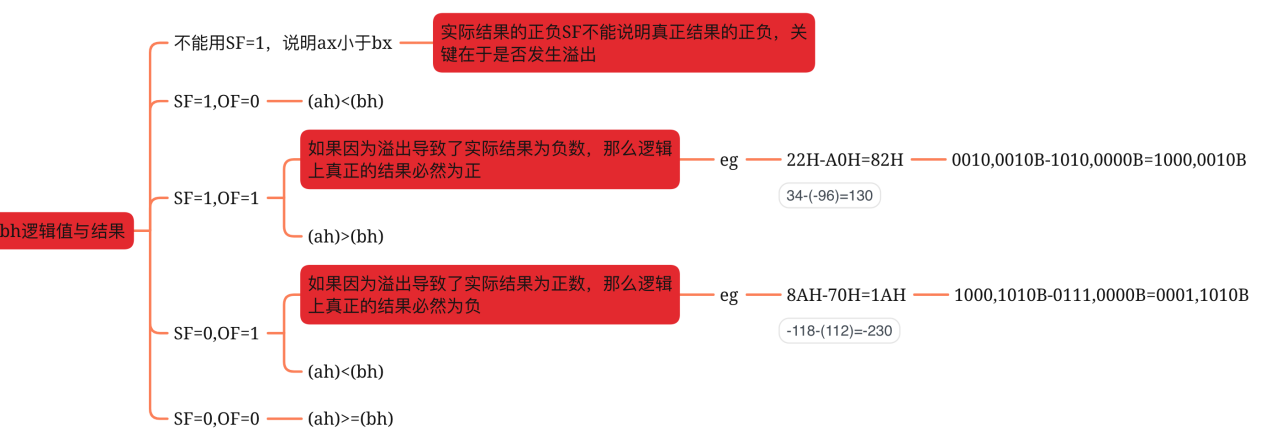


图12·cmp指令

11.9检测比较结果的条件转移指令

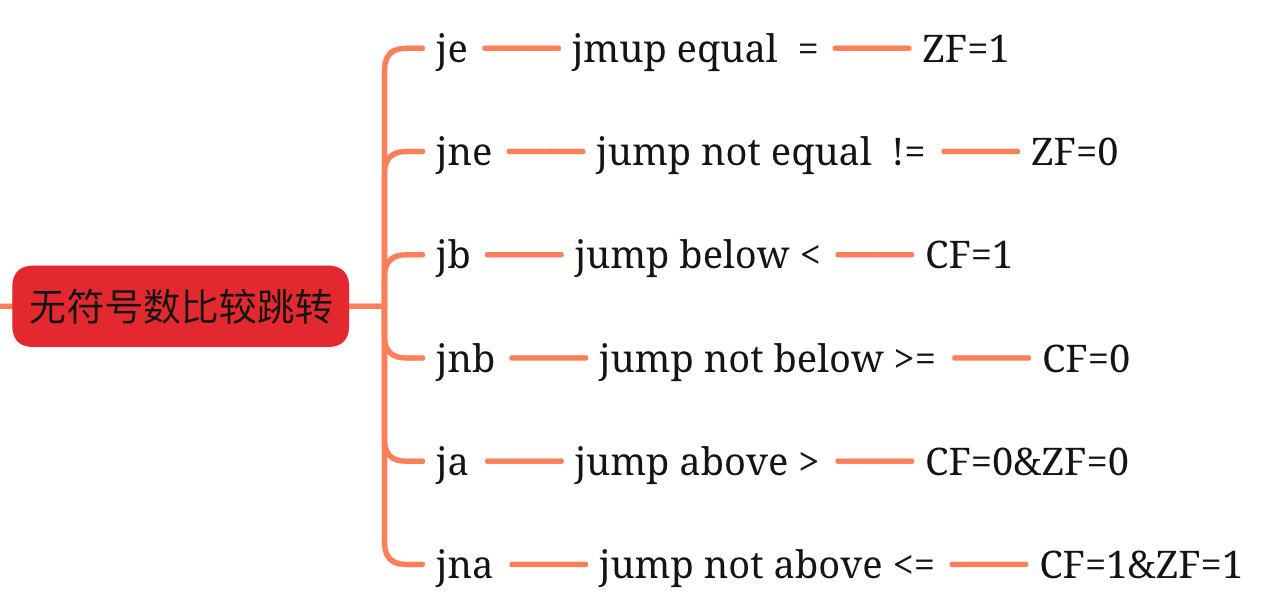
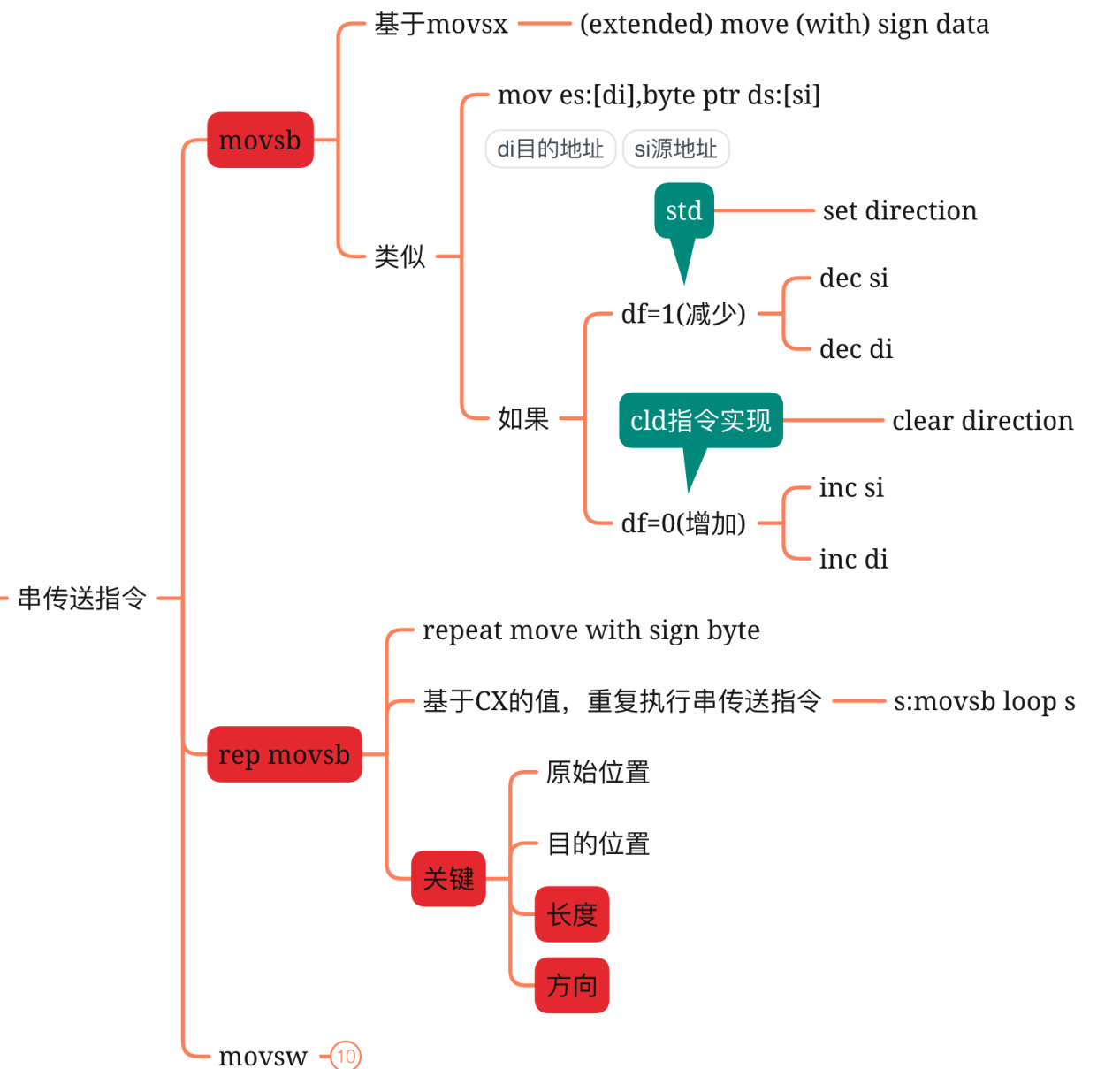


图13.条件跳转

11.10 DF标志和串传送指令



图·14串传送指令

11.11 pushf和popf

11.12 标志寄存器在debug中的表示

1. 内中断

·任何一个通用的CPU都具备一种能力，可以在执行完当前正在执行的指令之后，检测到从CPU外部发送过来的或内部产生的一种特殊信息，并且可以立即对接受到的信息进行处理

12.1内中断的产生

·情况：

(1.除法错误： 中断类型码0

(2.单步执行: 中断类型码1

(3.执行into指令: 中断类型码4

(4.执行int指令

12.2中断处理程序

12.3中断向量表

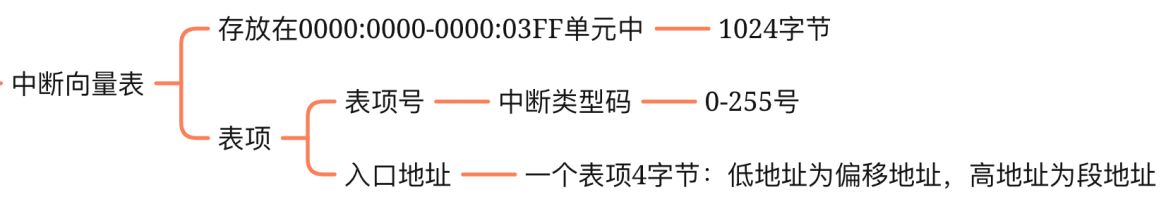


图15·中断向量表

12.4中断过程

(1.取得中断类型码

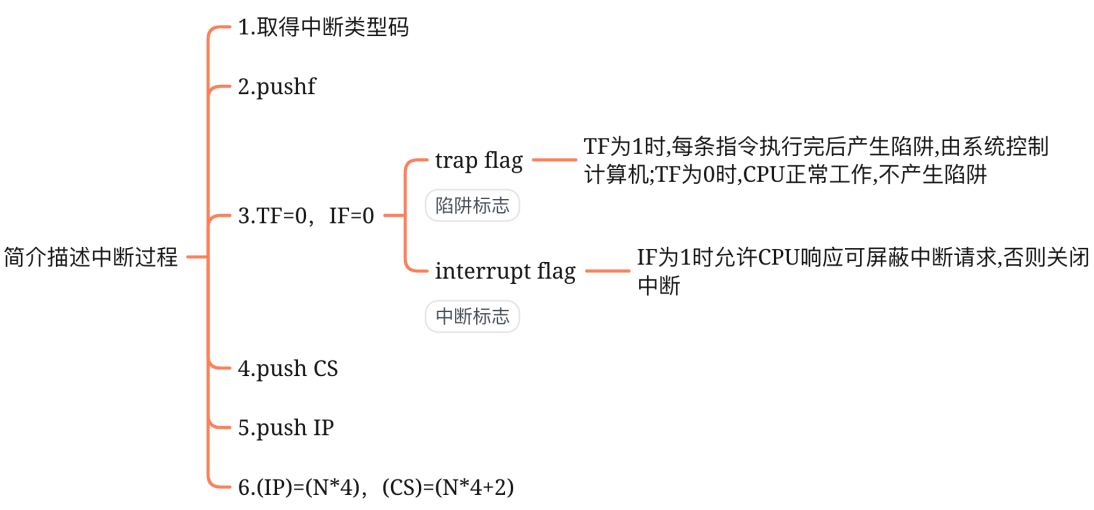
(2.标志寄存器的值入栈

(3.设置标志寄存器第8位TF和第9位的值位0

(4.CS的内容入栈

(5.IP的内容入栈

(6.从内存地址为中断类型码\*4和中断类型码\*4+2的两个字单元中读取中断处理程序的入口地址设置IP和CS



图·16简要描述中断过程

12.5中断处理程序和iret指令

·常规步骤：类似与子程序

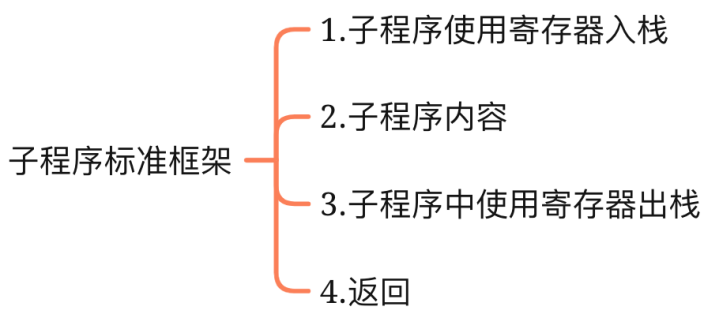


图17·子程序标准框架

(1.保存用到的寄存器（flag，CS，IP）

(2.处理中断

(3.恢复用到寄存器

(4.用iret指令返回

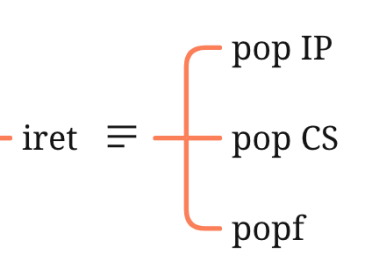


图18·iret指令

12.6除法错误中断的处理

12.7编程处理0号中断

12.8安装

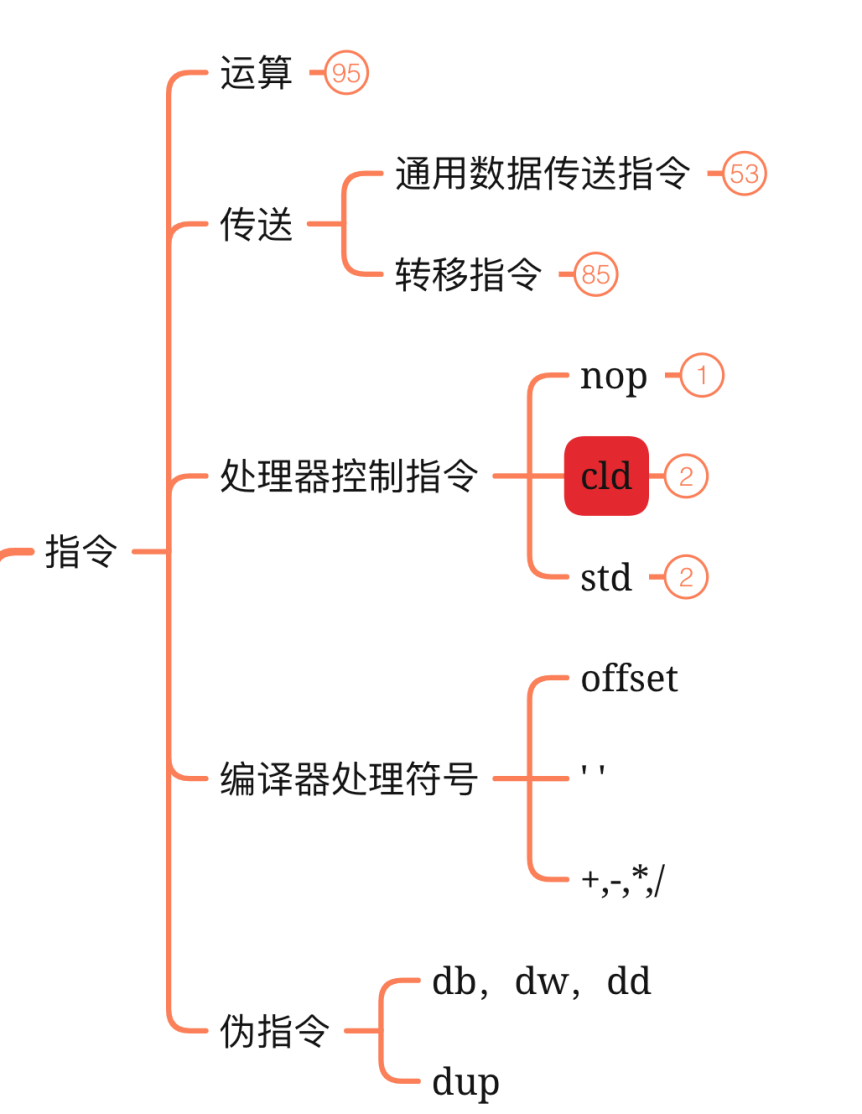
12.9 do0

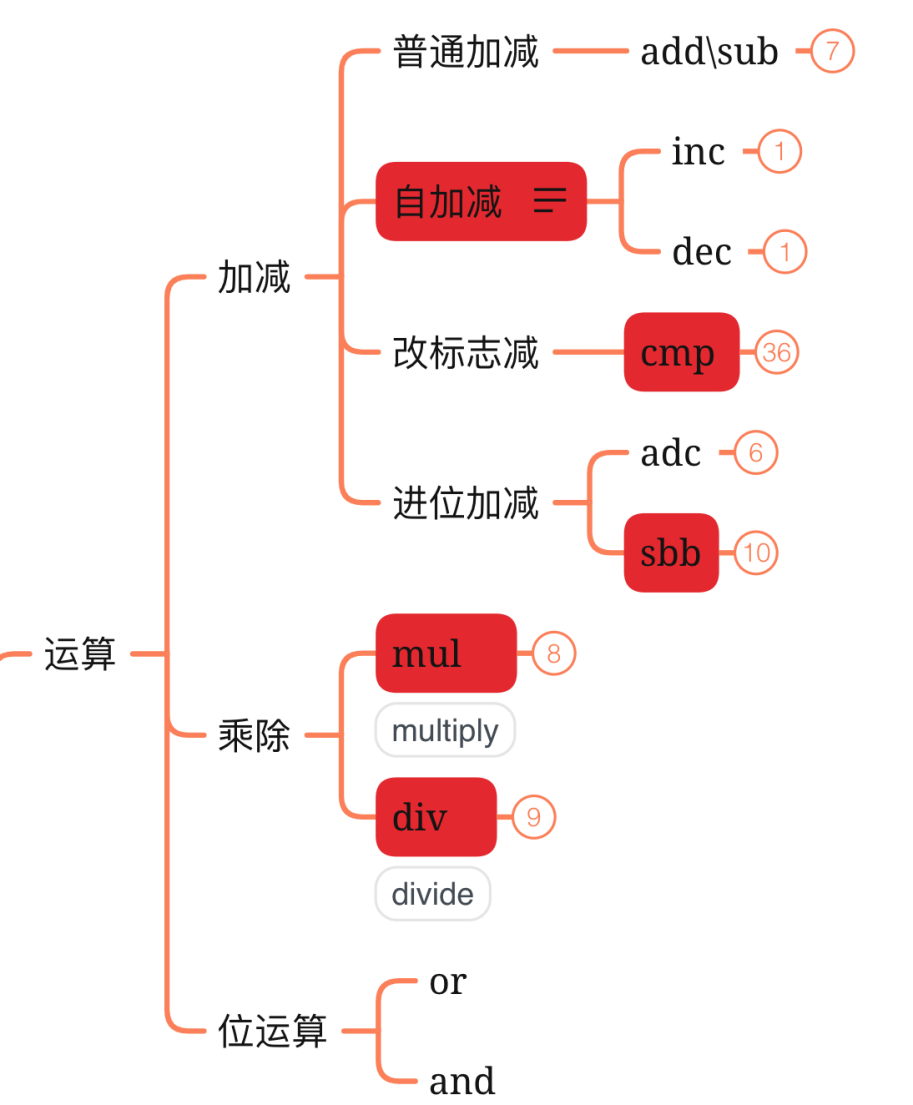
12.10设置中断向量

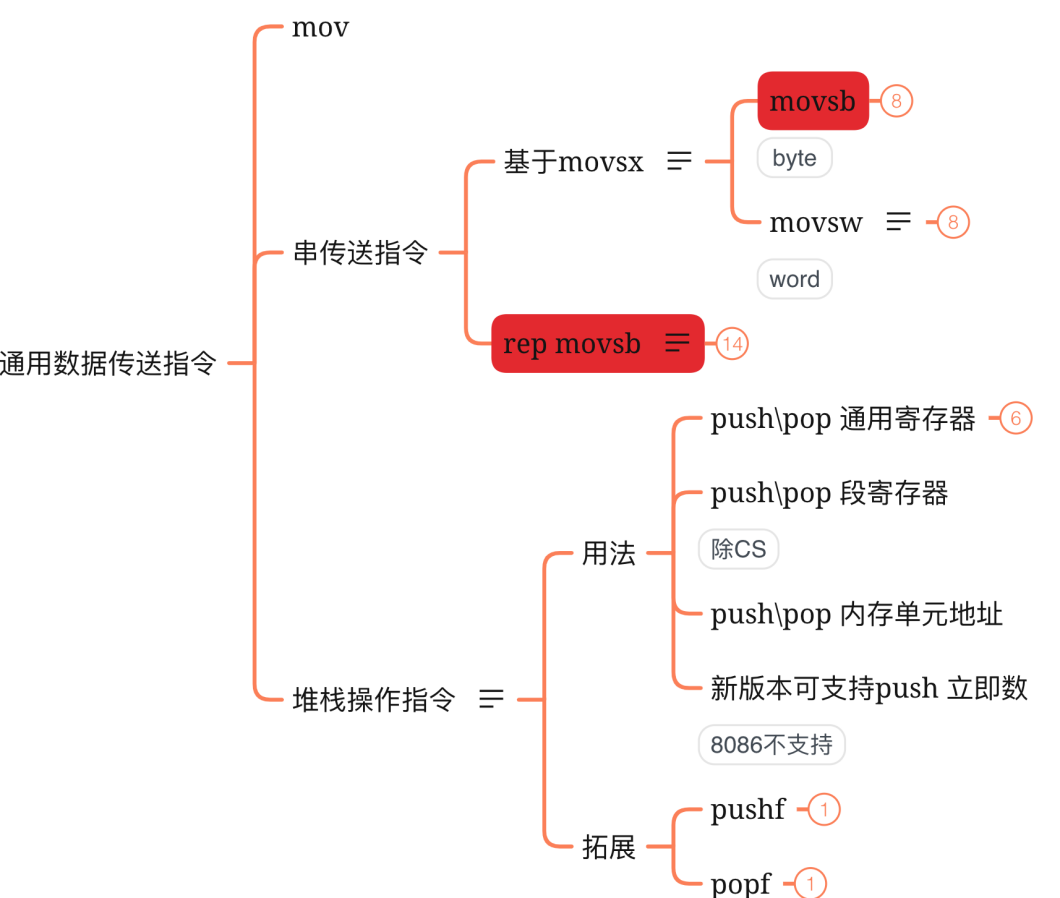
12.11单步中断

12.12响应中断的特殊情况

1. 总结







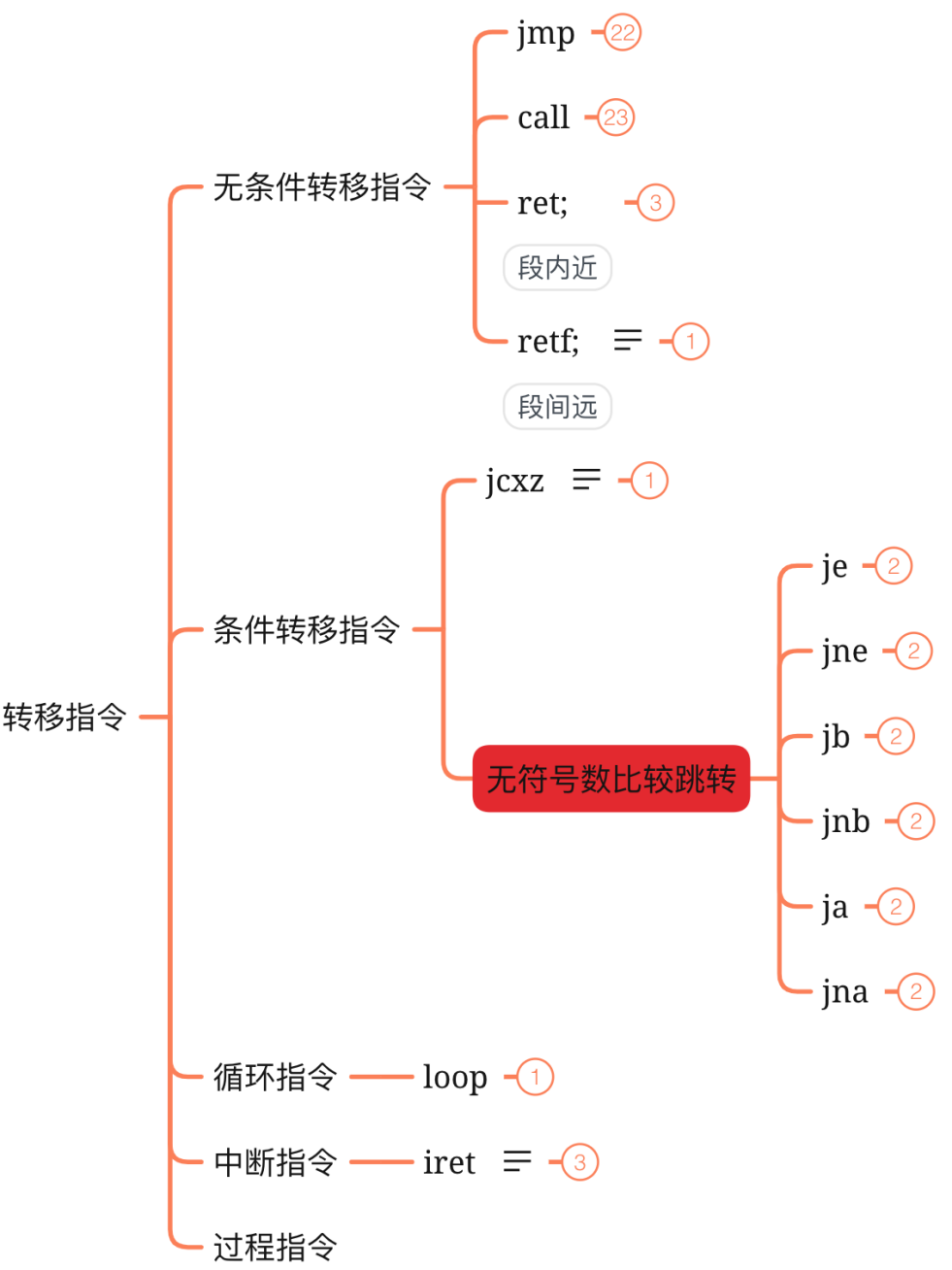


图18·指令总结