寄存器共（14）个

（4）ax,bx,cx,dx（大小为一个字） 进一步细分为ah,al,bh,bl,ch,cl,dh,dl （大小为1个字节），其中cx还用于loop循环递减寄存器

（2）cs,ip,指令寄存器，cs为指令段地址，ip为指令偏移地址。代码段

（1）ds,[address], ds为内存数据段地址，偏移地址,通过这种方式访问内存中的数据

（2）ss,sp，内存栈寄存器，ss为栈的段地址，sp为栈的偏移地址，也是**栈顶**的指针寄存器。Ss sp定义好了后栈指针会自动标记当前数据地址。

（1）bp,未知指针寄存器，和sp类似 bx与bp区别 bx默认ds数据段 bp默认ss堆栈

（2）si,di,变址寄存器

（1）es 附加段地址寄存器 与ss,ds,cs同等效果 es往往和ds一起使用，做为内存段地址使用

（1）psw 未知标志寄存器

汇编语言程式

mov指令为赋值指令

add指令为+=指令

sub指令为-=指令

Mov ax,2000

Mov bx,3000

Add ax,bx 意思是ax+bx的值存入ax中

Sub ax,bx 意思是ax-bx的值存入ax中

改变指令寄存器的指针cs:ip的当前值

Jmp 指令为改变指令寄存器，段地址cs+ip 或者直接修改ip

jmp 2ae3:3 改变cs:ip的值

jmp 3:0b16 改变cs:ip的值。

jmp ax == mov ip,ax 跳转到某一合法寄存器

通过ds加偏移位置方法访问内存

mov bx,1000

mov ds,bx，ds为关键字 ，定义ds段地址，再通过[address]直接读取内存数据

mov al,[0] 将10000H中的数据读到al中，取一个字节

mov bx,[0] 将10000H中的数据读到bx中，取两个字节

写入内存栈，读出内存栈

mov ax,1000

mov ss, ax

mov sp,10

push ax ;ax的值入栈

push 100 ;100入栈

pop ax ;栈顶的值取出，并赋值给ax

pop bx ;栈顶的值取出，并赋值给bx

这样就规划出了栈的空间，pop指令为后进先出，要防止栈顶栈底越界问题。

基本的汇编程序应该包含

伪指令和代码段两部分，一个有意义的程序，应该至少包含一个代码段

Assume cs:abc,ds:data,ss:stack ;ds,ss为非必要程序

;非必要开始

Data segment

Dw 0123h,0456h,0789h,0abch,0defh

Data ends

Stack segment

Dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Stack ends

;非必要结束

Abc segment

Mov ax,2

Add ax,ax

Add ax,ax

Mov ax,4c00h

Int 21h

Abc ends

End

Inc bx

Inc bx

加1指定

dec减1

Mov ax,2

Mov cx,11

S: add ax,ax

Loop s

循环执行ax相加,cx减1，如果cx为零时，退出循环

Mov ax,0ffffh

这句话表明汇编语言中数据不能以字母开头，如果是，前面要加0.

在段前缀失效的情况下[0]，要加明确的段前缀ds:[0],只有这种写法才不会造成编译器错误

Dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h

Dw 定义一个字

Db 定义一个字节

Dd 定义一个双字

Assume cs:code

Code segment

Dw 0123h,0456h,0789h,0abch,0defh,0fedh,0cbah,0987h

Start:mov bx,0

......

Code ends

End start

通过start记录程序起始位置

‘......’汇编程序中，用单引号指明数据是以字符形式给出的。编译器会

把它们转化成ASCII码

Ds:data

Data segment

Db ‘abcde’ ;用db的方式就可以直接在内存段中写入ASCII码。

Data ends

Assume cs:codesg,ds:datasg

Datasg segment

Db ‘basdf’

Db ‘asdlfkas;dfjk’

Datasg ends

And 是与运算，二进制中，1和任意数字（0或者是1）进行与运算结果都

是保持原数字不变

And 用法 and dest, src and 目标字节，原字节。

Or 用法 or dest, src or 目标字节，原字节。用法和and相同d

小字转大写，and 1101 1111

大写转小写， or 0010 0000

巧妙运用0和1的与运算，得出相应的结果

ds不可以直接输入值，必须由其它寄存器例如ax转入，代码如下

Mov ax,datasg ax的段地址和偏移地址等于ds:datasg中定义的值

Mov ds,ax

Mov bx,0 ;用bx做偏移地址累加寄存器

Mov cx,5 ;循环5次

S:mov al,[bx] ;s定义循环体开始,al，ax寄存器，低位地址值设置为0

Bx+idata 一个特殊的数来寻址

Mov ax,[bx+200] / mov ax,[200+bx]

内存单元段地址在ds中，偏移地址在[bx+200]中，其中方括号

意思表示就是相加的结果

同样意思的几种写法

Mov ax,[200+bx]

Mov ax,200[bx]

Mov ax,[bx].200 虽然写的时候不一定全用得上，但是读

别人代码的时候应该会用得上。

有类似于c语言中的数组的味道

Ds+基址+变址方式指定地址

Mov ax,[bx+si]

Mov ax,[bx][si]

以上两种写法效果相同

在汇编语言代码中，如果数字后边不写h，代表本身就是16进制

数的话，会按照10进制计算，结果转换为16进制。

assume cs:codesg

codesg segment

start: mov ax,2000h

 mov ds,ax

 mov bx,1000h

 mov si,0

 mov ax,[bx+si]

 inc si

 mov cx,[bx+si]

 inc si

 mov di,si

 mov ax,[bx+di]

 mov ax,4c00h

 int 21h

codesg ends

end start

P9.asm

[bx+si+idata]

Mov ax,[bx+si+idata]

将一个内存单元的值放入ax,这个单元的长度为一个字，不是字节

段地址在ds中

CPU运行模式

一、读取指令

1. 从cs:ip开始 代码段段地址：偏移地址
2. 地址加法器 确认数据命令存储的物理地址
3. 地址总线 负责找到地址
4. 内存按cs:ip取出三字节或者两字节的16进制数
5. 取出三字节的指令A10E00
6. 数据总线 负责回传数据
7. 数据放入缓冲寄存器
8. CPU读取指令mov ax,ds:[000e]

二、获取数据

1. 直接寻址即直接操作内存中的数据，把数据存在ax中
2. 地址加法器地址总线
3. 取ds:000e中的数据
4. 数据总线
5. 直接把ds000e中的值存入ax

以上程序执行过程中，都会把指令缓冲寄存器中的后两字节当作偏移地址

一个程序的执行，都是从cs:ip单一入口开始执行的。举一返三的进行分支判断，循环判断等等，程序的入口一定要先学会书写，学习一种语文的开始就是找到程序的cs:ip

首先取指令呢，要先找到指令的物理地址

三字节指令取出来，不用按字节倒序，两字节数据取出来，需要按字节进行倒序

三个字节怎样通过16位总线传输呢，还是一个字一个字的传输

指令执行过程中呢，CPU知道需要取数据了，就会自己去取

我们现在学的是汇编语言，每一条指令都和CPU直接关联，有些指令不能执行，是因为设计电路的时候就没有让它执行，在机器设置指令集的时候就没有给它。

汇编中，可以用mov al,’a’来把字符串赋值给寄存器

指令要处理的数据在寄存器中，在汇编指令中，给出相应的寄存器名

指令要处理的数据在内存中，要给出段地址和偏移地址

内存中的数据可以通过debug命令 -d 查看，-e直接修改内存中的值

数据在汇编语文中共有三种，1、立即数(mov ax,1中的1就是立即数) 2、寄存器（ax,bx,cx,dx） 3、内存中（段地址：偏移地址）

Mov word ptr ds:[0],1 这个指令看不出来ds0是字还是字节，用ptr声明后就知道是一个字了

Mov byte ptr 也是同样的道理，只不过表示的是字节

Acsii码用一个字节表示一个英文，可以用byte ptr方式寻找物理地址，然后替换

Div 指令是除尘指令 被除数默认放在ax或dx和ax中，除数，8位或者16位，在寄存器

或内存单元中。

Dup指令设置重复的内存空间

Data segment

Db 3 dup (0) 定义三个0

Db 3 dup (0,1,2) 定义三次0，1，2

Db 80 dup (0) 定义80个字节

与以下代码等价

Dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0

Dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0

Dw 0,0,0,0,0,0,0,0,0,0,0,0,0,0

Jmp short s

Add ax,1

S:inc ax 跳过上一行直接走s

按转移行为

段内转移，只修改ip 例jmp ax

段间转移，同时修改cs,ip例 jmp 1000:0

按ip修改范围

段内短转移，ip范围-128-127

段内近转移，ip范围-32768-32767

按转移指令

无条件Jmp,

有条件jcxz 转移看cx寄存器z值是否为0

循环 loop

过程

中断

Offset + 标号 s: ss:都可以取得标号的偏移地址

Codesg segment

Statr: mov ax,offset start; 相当于mov ax,0

S: mov ax,offset s; 相当于mov ax,3

Codeseg ends

End start

通过offset的方式，自动获取偏移地址，减少出错的概率

S0: nop

nop

这两条指令为空指令

Jmp指令，修改cs或者ip

Jmp short 标号 段内短转移

Jmp near ptr 标号 段内近转移

Jmp 实现原理是相对转移和跳转

assume cs:codesg

codesg segment

start: mov ax,1

 ;jmp short s ;short 表示只在-128-127之间跳转

 jmp far ptr s ;far ptr 表示在cs:ip间转移，远转移

 jmp near ptr s ;只在ip间跳转

 db 128 dup(0)

s: mov ax, 0ffffh

codesg ends

end start

;本程序存在错误,跳转范围越界

jmp word ptr ds:[0] ;段内转移

 mov word ptr ds:[2],0

 jmp dword ptr ds:[0] ;两个字转移

Jcxz 标号

当cx==0的时候，跳转到标号处

Ok:mov ax,0 前边的ok就是标号

Loop 标号

Cx=cx-1

Cx==0跳出

Cx!=0 循环 相对位移不容易发生错误

Call 和 ret 指令

主程序和子程序

Mov ax,0

Call s

Mov ax,4c00h

Int 21h

S:add ax,1

Ret

其实质还是一个流程转移

Call far ptr s

S:add ax,1

Ret

是段转移

默认是call near ptr 标号 是短转移

Call ax call 16位寄存器

相当于 push

Call word ptr 内存单元地址

Mov sp,10h

Call word ptr ds:[0]

Call dword ptr 内存单元地址

从栈中取值，没有call也行 也就是说 call 和ret是栈操作

Ret == pop ip

Retf == pop ip

Pop cs

Mul 乘法指令

Div 寄存器

Div 内存单元

除法指令

乘法和除法类似

8位乘法

Mul bx

Ax=al\*bl

16位乘法

Mul word ptr [bx+si+8]

100\*10

Mov al,100

Mov bl,10

Mul bl

结果

Ax=1000(03e8h)

模块化程序设计

调用过程结果结果存在哪里？

用寄存器 最常用

Dw 2字节

Dd 4字节

用内存单元批量传递参数

用栈传递参数 不理解

寄存器的冲突问题及解决方法

Db ‘asdfasdfasdf’,0

读到是0的时候结束，和while相似

有冲突的寄存器，避免使用，但是会很麻烦

必须看到子程序源码

不要使用会产生冲突的寄存器，也是相当麻烦的。

我们希望

调用子程序不关心

编写子程序也不关心

不发生寄存器冲突

在子程序开始，把将要用到的寄存器都保存起来，子程序结束时再恢复起来。

这是一个非常重要的技术。

感觉栈就和剪切板是一样的功能，只不过剪切板只支持一个入栈和出栈

1. 标志寄存器

标志（Flag）寄存器 psw/flags 也叫程序状态字 16位，是按位起作用的。

Pushf

Popf

使用一条指令的时候，要知道指令的全部功能，其中包括对执行结果对标志寄存器的哪些位造成影响

Sf符号标志位，负数补码取反，正数与原数相同

符号标志位是结果的重要标志。没有这个位，结果将没有正负之分。当作有符号数计算时，sf有意义，当作无符号数时，sf无意义。

Cf 进位标志，借位标志。Cf用来判断是否进位，结果是否正确。

Of 溢出标志，超出机器表达数的范围，则为真。用来判断结果是否正确。

Of ov nv 溢出

Df dn up 方向

Sf ng pl 符号

Zf zr nz 零值

Pf pe po 奇偶

Cf cy nc 进位

1. 带进借位的加减法

Adc 操作对象1，操作对象2

Adc ax,bx == ax=ax+bx+cf

Cf 最多进一位，所以只要有进位，在前面加1就对了。

128位数据相加 需要8个字单元 16 个字节

sbb带借位减法指令。

1. cmp比较指令

Cmp 操作对象一，操作对象二

相当于是减法指令，不保存结果，但对标志寄存器产生影响。

cmp或者其它影响寄存器的指令，进行条件转移

Jxxx 标号

Je/jz

Jne/jnz

Js

Jns

Jo

Jno

Jp

Jnp

Jb/jnae/jc

Jnb/jae/jnc

Jna/jbe

Ja/jnbe

Jnb/jae/jnc

Jl/jnge

Jnl/jge

Jle/jng

Jnle/jg

1. df标志和串传送

df方向标志位，加1或者减1

串传送指令

Movsb 以字节为单位传送

movsw以字为单位传送 这样一个命令就是复制粘贴命令。

Cld 设置df位为0

Std 设置 df位为1

Rep 指令

根据cx值，重复执行的后面指令

Rep movsb

59、中断和外部设备操作

60、移位指令

左移Shl opr,cnt

右移 shr opr,cnt

Rol opr,cnt

Ror opr,cnt

Sal opr,cnt

Sar opr,cnt

Rcl opr,cnt

Rcr opr,cnt

S,sh-shift l-left r-right a-arithmetic r,ro-rotate c-carry

Mov cl,3

Shl al,cl cl不能用立即数，除非是1

结果

Al=100001000b

Cf=0

1. 操作显存数据

屏幕上的内容=显存中的数据

8086共有128k显存空间

显示器共有25行，80列组成。

每行160个字节

低位放ascii，高位为显示属性

Bl r g b i r g b

闪烁 背景 高亮 前景

1. 关于标号

代码段中数据也可以用标号

A:db 1,2,3,4,5,6,7,8

B:dw 0

冒号可以去掉

Mov si offset a

Mov bx offset b

它们同时描述内存地址和单元长度

直接用标号可以免声明字还是字节

地址标号只能在代码段中使用

将标号当作数据来使用

A db 1,2,3,4,5,65,67,78

B dw 0

C dw a,b == c dw offset a, offset b

c保存的是地址值，可以间接的访问a和b

C dd a,b == c dw offset a, seg a, offset b, seg b

seg为操作符，取段地址

63、数据的直接定址表

先将byte的高位低位分开，再显示对应的数码字符

0+30h=‘0’的ascii值

10+37h=’a’ 的ascii值

建立一张表，直接找到，最简单 table db ‘0123456789abcdef’

根据对应关系，查找对应数据

优点，算法清晰简洁，加快运算速度，使程序易于扩充

考虑容错性。直接定址表可以存子程序。方便子程序的调用。

64、代码的直接定址表

写4个子程序，然后调用子程序。程序入口保存在一个表里。功能号\*2就能够找到功能地址表中的偏移。

清屏 设置前景色 设置背景色 设置滚屏，其实就是对显存的操作。

优势：便于扩充，结构清晰，

1. 中断及其处理

分内中断和外中断

cpu内部和cpu外部

可编程中断控制器

8259a pic

内中断

除尘错误 0

单步执行 1

执行into 溢出中断4

Int 指令 n

Int n n 为立即数，中断

13 10 ‘$’回车换行

lea取标号地址

Int 可以完成的事情很多

？中断处理程序在哪里

中断向量表中，知道中断号，就可以查到中断入口地址

8086最多有 0-255个中断 从0000:0000-0000:03fe ip:cs ip=n\*4 cs=n\*4+2

Div bh, 除以0触发中断处理程序

中断过程由cpu硬件自动完成，一定要深刻理解.用中断类型码，找到中断向量，并用它设置cs和ip

66、中断处理程序的编制方法

1，取得中断类型码

2，pushf

3，tf=0, if=0

4，push cs

5，push ip

6，ip=n\*4 cs = n\*4+2 cs入口地址 ip n号中断处理程序

cpu随时都可以检测到中断信息，所以，中断处理程序必须常驻内存。

中断处理程序的入口地址，必须存储在中断向量表中0000：0000-0000：03fe

除法中断处理程序

1. 保护现场
2. 屏幕打印
3. 返回dos

要求重新找处理程序

事实，需要向操作系统申请

简化方案，找到一块别的程序不必用到的内存区

8086支持256个中断。中断向量表中不用的空间

确定：就这样做了，用中断向量表中不用的来保存 0000：0200

编写程序，

安装程序，程序送入内存

把程序存储在中断向量表中

cld传输方向为正

Rep movsb 串传送命令

Offset do0end-offset do0

将数据写到代码区，直接输出到显示缓冲区

写到中断向量表

把自己的程序安装到内存指定区域

通过运行相关除法程序，证明，程序已经在内存中运行了。中断是cpu直接接手的，8086一共有256个中断，中断运行模式为，1，保存现场，2，执行中断程序，3，回到程序中继续执行。

67、单步中断

程序正常执行：取指令，改变cs:ip，执行指令，取指令....

-t命令

就是cpu单步中断功能 tf标志为1的时候，就在单步中断方式下工作

寄存器Tf,单步执行

寄存器if，中断标志，if=1可屏蔽中断请示，0，关闭中断

中断不响应的情况

执行完向ss寄存器传送数据后，不会发生中断

原因是ss,sp联合设置栈，所以不允许中断的发生。

1. int n 引起的中断

n为立即数，为中断类型码

Int 21 程序结束, int 0 除法错误

In 指令的功能和 call 一样，都是调用一段子程序

用int 调用子程序 这种程序简称中断例程

Iret == pop cs pop ip popf

Int 7ch 执行7ch中断

中断处理程序的常规步骤

1. 保存用到的寄存器
2. 处理中断
3. 恢复用到的寄存器
4. 用iret指令返回

69、bios 和dos 中断处理

Bios 容量8k 地址从fe000h开始

主要内容

1. 硬件系统的检测和初始化
2. 外部中断和内部中断的中断例程
3. 用于对硬件设备进行i/o操作的中断例程
4. 其它和硬件系统相关的中断例程

系统bios rom地址空间，网卡bios rom地址空间（网卡bios），显卡bios rom 地址空间(ram显存，显卡bios)，显存地址空间，主存贮器地址空间（内存条）

使用bios功能调用，程序员不用了解硬件操作细节，直接使用指令调协参数，并用中断调用bios例程，即可完成工作。使用bios，方便编程，能写出简洁，可读性好，易于移植的程序。

打印特殊文字在屏幕上

还有哪些bios中断，怎么用？

百度bios中断手册 一共8种中断 10h 13h 14h 15h 16h

除了bios中断，还有dos中断，由操作系统提供

20h 21h 2ah 取系统日期 ah=2ah cx=year dh month dl day al weak

dos中断其实是依据bios中断的，但dos中断提供更高级的功能。程序可以通过dos中断，bios中断之外，还可以直接操作外设，不需要中断。

Bios和dos所提供的中断例程包含了许多子程序，这些子程序实现了程序员在编程时经常用到的功能。

和硬件相关的dos中断例程，一般都是调用的bios中断例程。

Int 21h dos中断例程的使用

4ch号功能：程序返回

Mov ah,4ch

Mov al,0

Int 21h

09h号功能

Mov ah,9

Int 21h

bios和dos中断例程的安装过程

1,cpu加电，初始化cs=0ffffh,ip=0

自动从ffff:0单元开始执行程序。

Ffff:0处有一条跳转指令，cpu执行该指令后，转去执行bios中的硬件系统检测和初始化程序。

1. 初始化程序将建立bios所支持的中断向量，即将bios提供的中断例程登记在中断向量表中，1k的空间。
2. 硬件系统检测和初始化完成后，调用int19h进行操作系统的引导。从此将计算机交由操作系统控制。
3. dos启动后，除完成其它工作外，还将它所提供的中断例程装入内存，并建立相应的中断向量。

70、端口的读写

Out in 就是在读写端口

In al,61h 设备控制器

Out 42h,al 输出到端口

cpu可以读写3个地方的数据，1，cpu内部寄存器，2，内存单元3，端口

端口包括，各种接口卡，网卡，显卡，主板上的接口芯片，其它芯片

各种芯片工作时，都有一些寄存器由cpu读写，从cpu角度，将各寄存器当端口，并统一编址，cpu用统一方法与各种设备通信。也就是说cpu和这些芯片通信的时候，都是由这些芯片内部的寄存器的读写来做的，比如显卡中的显存。

在8086中，程序外设被统一编址为端口的64k地址空间（0-65535）。访问端口地址或者端口号，就可以访问端口对应的外设。

读取内存与寄存器的指令

Mov, add, push ...

读取端口的指令

In cpu从端口读数据

Out cpu往端口写数据

例： in al,60h ;从60号端口读入一个字节

在in和out指令中，只能使用ax或al来存放端口中读入的数据或者要发送到端口中的数据。访问8位端口用al，访问16位端口用ax.

Mov dx,3f8h

In al,dx 端口读取

Out dx,al 端口写入

想要使用某个端口，必须知道对应端口的端口号。

71、操作cmos ram芯片

即bios芯片

包含一个时钟，有一个128个存储单元的ram存储器

ram中包括，内部时钟，系统配置信息，相关的程序

Cmos ram 芯片依靠电池供电，关机后其内部实时时钟仍可正常工作，ram中的信息不丢失

该芯片有两个端口 端口地址为70h 71h ，cpu通过这两个端口读写cmos ram

例：在cmos中提取时间信息

时间信息用bcd码存贮。

Mov al,8 ;取出地址是8的月份

Out 70h,al

In al, 71h 提取月份到al中

72、外设连接与中断

Cpu直接读取三个地方数据

1. cpu内部寄存器
2. 内存单元
3. 端口（各种接口卡，网卡，显卡，主板上的接口芯片，其它芯片）

端口是统一编址的，包含在64k的范围之内。

In,out命令直接与接口通讯。

需要中断的参与，处理外设的输入。

中断分为，内中断和外中断

外中断是指由外部设备的事件引起的中断。

可屏蔽中断

是否响应，要看标志寄存器if位的设置，if为1，可中断，if为0，忽略可屏蔽中断

来自cpu外部，中断类型码是通过数据总线送往cpu

相比内中断，中断类型码在cpu内部产生。

1. 取中断类型码n
2. 标志寄存器入栈，if=0 tf=0 if为0的原因是进入中断处理程序后，禁止其它的可屏蔽中断，如果在中断处理程序中需要处理可屏蔽中断，可以设置if=1
3. Cs,ip入栈
4. Ip=n\*4 cs= n\*4+2 根据cs:ip转去执行中断处理程序

不可屏蔽中断

cpu必须立即响应的外中断，8086中不可屏蔽中断类型码为2

突然断电了，计算机并不是什么都不做，而是会以紧急中断的方式执行一段代码。

1. 标志寄存器入栈 if=0 tf=0
2. Cs ip 入栈
3. Ip=8, cs=0ah

8086cpu提供的设置if的指令

Sti 用于设置if=1

Cli 用于设置 if=0

73、PC机键盘的处理过程

键盘的输入处理过程

1. 键盘输入
2. 引发9号中断，是一个外中断
3. 执行int9的中断例程

每个键相当于一个开关，有一个芯片对键盘上每一个键进行扫描。

按下键的时候，该芯片就会产生一个扫描码，说明了按下的键盘位置。

扫描码被送往主板上的相关接口芯片的寄存器中，该寄存器的端口地址为60h

松开键也会产生相应操作，存入60h端口中

扫描码为一个字节的编码，按下时，产生通码，第7位为0，松开时，产生断码，第7位为1

例g键通码为22h,断码为a2h,通码+80h==断码

与acsii码并不同，并不是相邻的。和当时的电路设计相关，此处不做详细讲解。

键盘输入到达60h端口时，相关芯片就会向cpu发出中断类型为9的可屏蔽中断。

cpu检测到该中断信息后，如果if==1,则响应中断，引发中断过程，转去执行int9中断例程

输入的字符键值如何保存

有bios键盘缓冲区，可以存15个键盘输入。高位存扫描码，低位存字符码acsii码。

如何处理控制键和切换键

有一个键盘状态字节。0040：17 76543210

执行时，先读出60h中的扫描码。根据扫描码分情况对待。如果是扫描码（ascii），把它存入内存中的bios键盘缓冲区。如果是控制键、切换键，转为状态字节，写入内存中的存储状态字节。对键盘系统进行相关控制，如向芯片发出应答指令。

内存键盘缓冲区，高位放的扫描码，低位放的ascii码

74、定制键盘输入处理 需要读懂得每一行代码

1，键盘产生扫描码

2，扫描码送入60h端口

3，引发9号中断

以上3条由硬件完成

1. cpu执行9号中断，处理键盘输入

编程任务

在屏幕中间显示a-z，按下esc后，改变颜色

策略，忽略硬件处理过程，利用原有的bios中int9。

改变中断向量表中的int9中断例程的入口地址改为自编的中断处理程序的入口地址。

要模拟原中断例程的调用

1. 改写中断例程的方法

以int9为例，其它中断例程类似

在dos下，按f1可以改变当前屏幕颜色，并让程序常驻内存

1. 改变屏幕颜色
2. F1改变功能，其它键照常
3. 原int9中断例程入口地址保存

Push es:[9\*4]

Pop es:[200h]

Push es:[9\*4+2]

Push es:es:[202h]

1. 新int9中断例程的安装

Call dword ptr cs:[200h]

76、用中断响应外设

硬件中断 int 9h

bios中断 int 16h

Dos中断 int 21h

对键盘输入的处理的Int9中断和int16h中断

Ah=扫描码 al=ascii码

中断逻辑中，CPU有中断检测功能，检测到键盘中断后，调用int9中断例程。

按a ，扫描码1e ,ascii码 61 ，放入内存键盘缓冲区，结合键盘状态字节，得出最后结果，通过数据总线，逐步清空键盘缓冲区，进入例程。

键盘缓冲区，共16字，用的是环形队列，先进先出。最多存15个字，从数据结构中可以得出这个结论。

77、应用：字符串的输入处理

在输入的时候显示字符

在回车后，结束输入

用退格键，删除已输入的字符

需要处理字符入栈，出栈，显示栈中的字符

Al 字符入栈 ah==0，

字符出栈 ah==1

显示栈中的字符 ah ==2

Dh,dl ==字符串在屏幕上的显示位置

Ds:si 指向字符串的存储空间，字符串以0为结尾符。

Table dw charpush,charpop,charshow

定址表

78、硬盘的读写

磁头，磁道，扇区，0柱面，1柱面，2柱面

磁盘操作，int 13h，在bios中已经给我们提供了。

Ah==2 2表示读扇区

Al==读取的扇区数

Ch==磁道号 ，cl==扇区号

Dh==磁头号 对于软盘即面号，一个面用一个磁头读写

Dl==驱动器号 软驱从0开始，0软驱A ，1 软驱B

硬盘从80h开始，80h==硬盘c 81h==硬盘d

Es:bx指向接收从扇区读入数据的内存区

返回参数：

操作成功，ah==0 al==读入的扇区数

操作失败，ah==出错代码

Dos中断对磁盘的操作 int 21h

目录控制功能，磁盘管理功能，文件操作功能，文件分块操作，学习操作系统时候讲过，还有记录操作功能，记录分块操作

比在c语言中的文件操作功能要强大得多。

1. 让计算机唱歌

外部设备如何被控制的

微机原理与接口技术，

可编程中断控制器8259a

可编程定时/计数芯片 8253

可编程的并行接口 8255a

可编程的串行接口芯片 16550

模/数和数/模转换器

......

从根本上是通过in out命令直接操作端口。

8253芯片的设置

Mov al,0b6h

......

8255控制扬声器的开关

翻译乐谱

音符与频率的对应表

制作数字化乐谱。

debug命令的使用方法

-r 查看寄存器状态

-rip -rax -rbx 直接修改寄存器的值

-d 查看内存中数据情况

-d 1000:1000 查看相应段地址和偏移地址的内存情况

-e 1000:1000 设置内存中的值，通过空格来同时设置多个内存值

-e 1000:1000 12 34 56 ab 3f用这种方式直接设置内存大小

-u 除了-r之外，还可以用-u查看所有机器指令，-u命令会不断指针下移，通过-r命令可以使之回到cs:ip位置 -u 也可以通过指定内存物理地址的方式来翻译命令

-t 单步执行输入的机器指令

-p 与-t类似，也是单步执行，但遇到结束或者中断也执行

-g 直接执行所有程序，直到程序结束，还可以在g命令后边加cs:ip，设置断点，直接执行到这一步停止。

-g 001e 这种方式可以执行到相关行，省略了cs段地址

-a 输入汇编指令，也可以通过指定地址的方式输入指令

-q 退出debug命令行