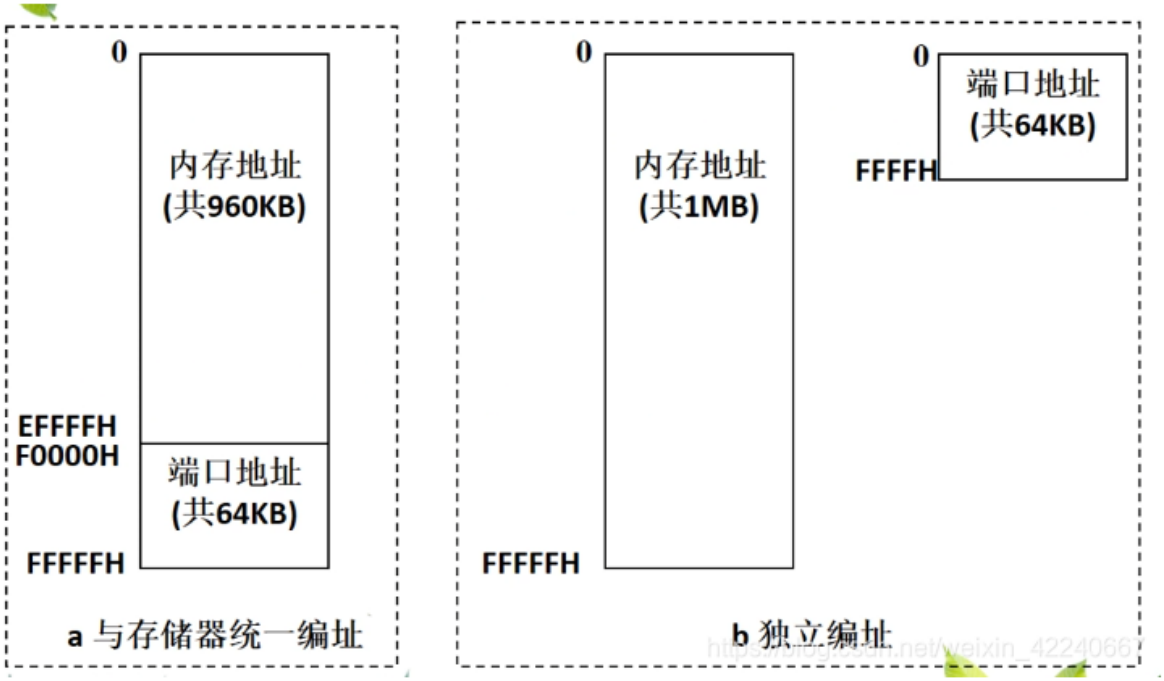
第5节 端口读写

1. 编址方式



如上图，如果是统一编址，20位地址线的寻址空间中，端口地址就占用了64KB，主存只有960KB了，好处就是

访问空间的指令统一，方便，缺点就是主存变小了。

然后是独立编址，8086cpu采用的就是独立编址方式，内存条和各种IO设备采用两套不同的指令和地址空间。

2. 主存地址空间划分

首先8086处理器对内存的地址空间划分如上，下面三段是开机启动的时候bios程序载入的空间，然后通过bios吧ibm dos系统加载到主存0x0000地址开始的地方开始执行.

用户程序，也就是编译好的汇编，有dos系统的加载器加载到内存中去，然后运行。所有运行模式都是实模式。

所以8086cpu的运行机制大致如下：

开机加电，bios通过硬件机制加载到BIOS空间，开始运行bios程序；

bios加载bootloader，开始运行bootloader；

bootloader为IBM DOS系统分配内存，然后加载到主存物理地址为0的地方，开始运行dos系统；

dos系统初始化一些相关的内存资源信息，设备信息，中断信息，为后面程序运行做铺垫；

然后由dos系统启动一段程序，包括加载，分配物理地址空间，最后CS:IP指向程序，开始运行程序。

重要的事情：操作系统由bootloader分配资源，程序由操作系统分配资源

实模式运行的缺点：假如这个程序瞎改dos系统所在物理地址的内容，会直接死机

3. 关于显示器显示

正常来说，需要通过IO端口，将数据写入显示器显存，然后显示出字符，但是种种原因，我们不这么做，而是先将数据搬运到显示器缓存区，然后显示器会定期从缓存区拿数据（通过硬件实现），进而显示出要显示的内容。

对于其他端口，也有类似的运行机制，一般不会直接通过IO端口，将数据写入IO设备，而是先写入缓存区，然后再通过IO端口或者硬件实现来写入IO设备

4. 显示器显示的例子

直接定址表（模拟数组）

为了更方便搬运数据，这里给出直接定址表的概念，模拟c语言中的数组，使得搬运操作时更容易

前面数据标号采用 标号+冒号+数据的形式，标号仅仅代表首地址，用起来还是不方便，然后这里

定义了一种新的标号，直接标号+数据的形式，去掉冒号，比如

```
data segment
    a dw 0，1，2，3，4
    b db a,b,c,d,e
data ends
```

使用方式如下

```
code segment
main:
    mov si,0
    mov di,0
    mov ax,a[si]
    mov bl,b[di]
    inc si
    inc di
code ends
```

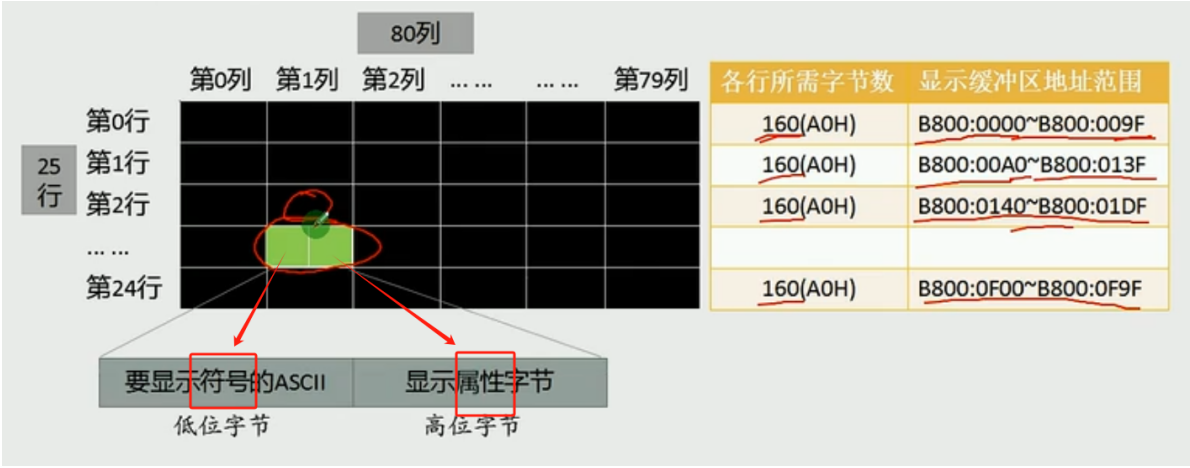
上面a[si]，b[si]完全等价与c语言中的数组，对于a，相当于 short a[5]，对于b相当于char b[5],

对于这种数据标号a和b，不仅仅表示地址，还包含了数据大小信息。（汇编编译器完成的）

关于显存缓存区

8086cpu为显示器显存设置了一段固定内存作为缓存区，定时从缓存区拿数据显示在屏幕上，缓存区

的物理地址如下图所示



缓存区物理地址的基址B800，偏移的范围从0000到0F9F，一共25x80=2000个单元，25x80x2=4000个字节。

两个字节定义了一个显示的字符，低字节表示字符，高字节表示属性，我们后面显示字符的时候并不关心属性。

向缓存区拷贝数据

有了直接定址表，我们接下来将data段的字符搬运到显存的缓存区，以此来显示字符

搬运思路：

data段定义需要写入的字符，比如hello，world

```
data segment
    a db 'hello world! '
data ends
```

使用附加段定位到缓存区，也就是mov es，B800H

由于上述2000个单元表示25行，80列的屏幕，我们打算把hello world写入中间第13行，从35列开始显示，

由于每行80个单元，所以第13行35列对应就是第12x80+35=995单元，也就是995x2=1990个字节的位置。

16进制为776h

ok，明确将a[0]写入B800H[7c6h]，然后依次搬运剩下的字符即可，一共需要搬运12个字符

全部的代码如下

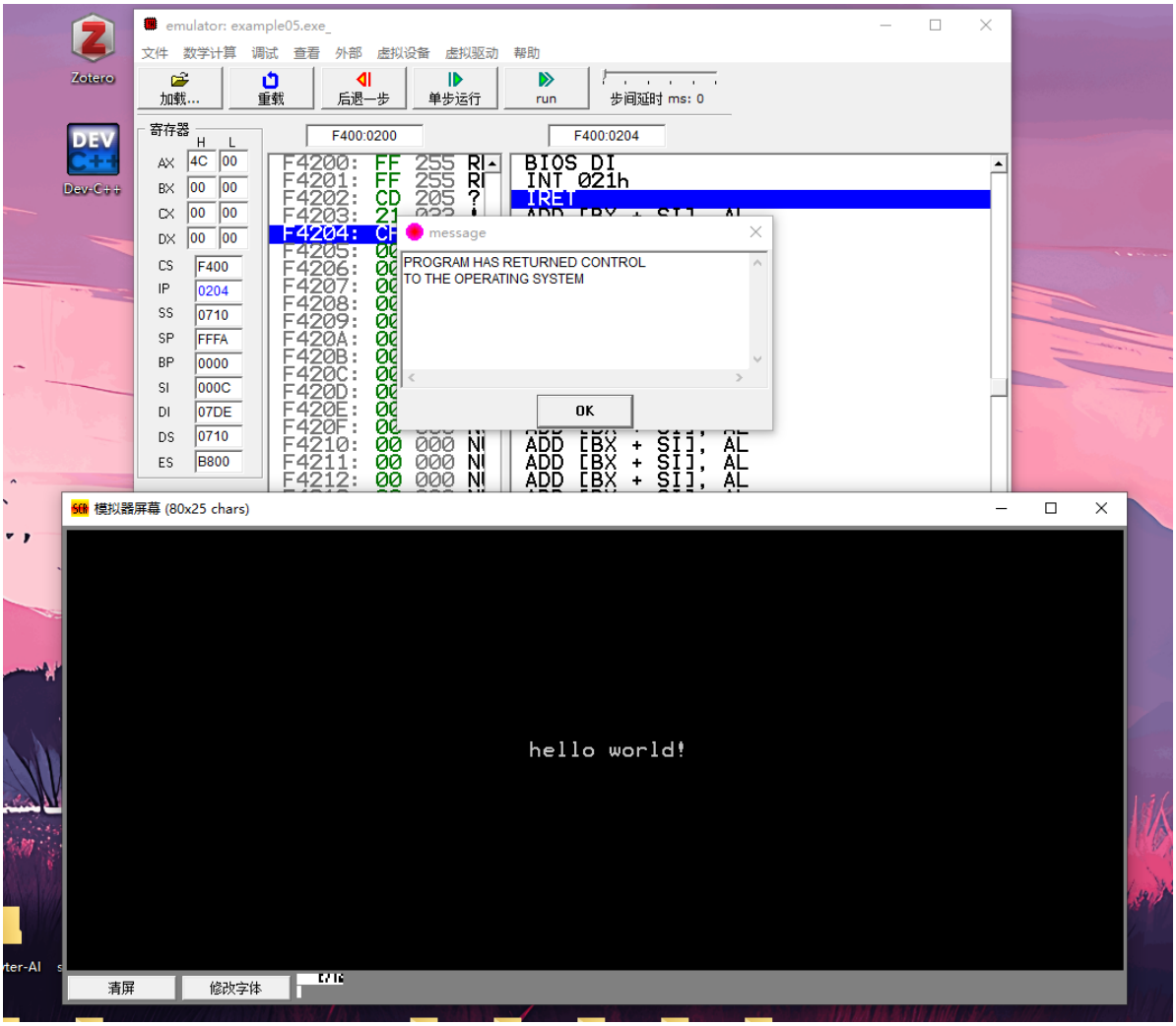
example05.asm

```
assume cs:code, ds:data

data segment
    a db 'hello world!'
data ends

code segment
main:
    mov ax, data
    mov ds, ax
    mov ax, 0b800h
```

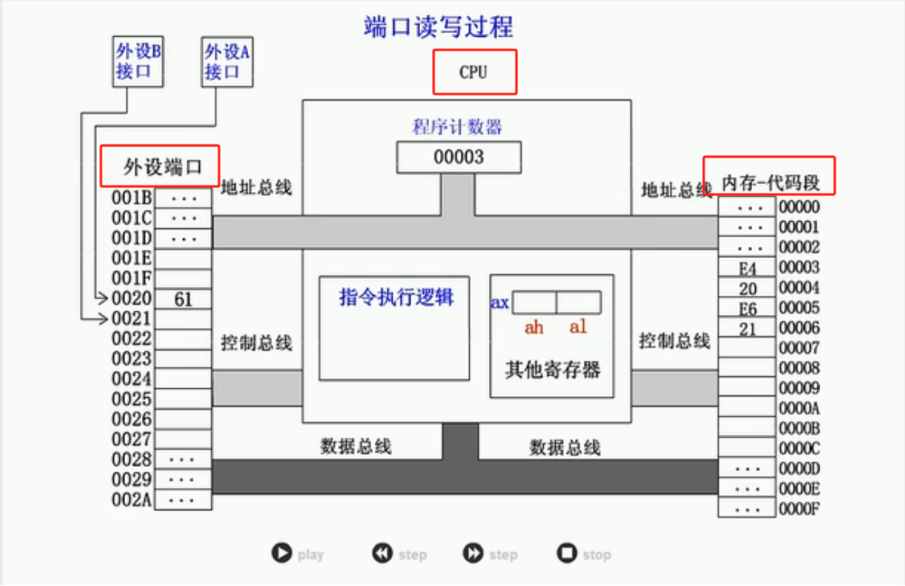
```
    mov es, ax
    mov si, 0
    mov di, 7c6h
    mov cx, 12
copy:
    mov al, a[si]     ; 使用数据标号模拟数组a
    mov es:[di], al   ; 使用es段定位目标数据段
    inc si            ; 数组a的地方，每次移动一个字节
    add di,2          ; 拷贝到目标的低字节处，每次移动2个字节
    loop copy
    mov ax, 4C00h
    int 21h
end main
code ends
```



5. IO端口读写的例子

端口的读写过程演示

```
in al, 20h
out 21h, al
```



I/O端口分配

I/O地址	分配说明	I/O地址	分配说明
00-1f	8237A DMA控制器1	170-177	IDE硬盘控制器1
20-3f	8259A 可编程中断控制器1	1f0-1f7	IDE硬盘控制器2
40-5f	8253/8254可编程中断计数器	278-27f	并行打印机端口2
60-6f	8255A可编程外设接口电路	2f8-2ff	串行控制器2
70-71	访问CMOS RAM/实时时钟RTC端口	378-38f	并行打印机端口1
80-9f	DMA页面寄存器访问端口	3b0-3bf	单色MDA显示控制器
a0-bf	8259 可编程中断控制器2	3c0-3cf	彩色CGA显示控制器
c0-df	8237A DMA控制器2	3d0-3df	彩色EGA/VGA显示控制器
f0-ff	协处理器访问端口	3f8-3ff	串行控制器1

端口的读写指令示例

对0～255以内的端口进行读写，端口号用立即数给出

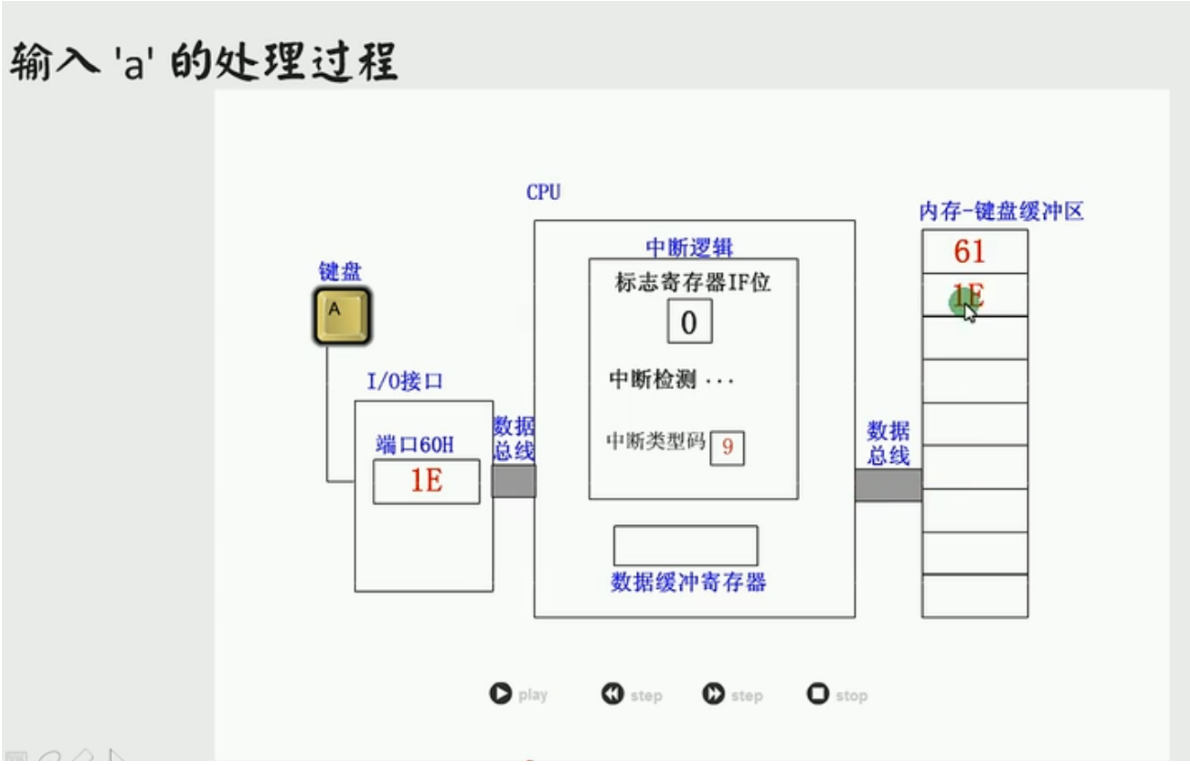
```
in al,20h       ;从20h端口读入一个字节
out 21h,al      ;往21h端口写入一个字节
```

对256～65535的端口进行读写时，端口号放在dx中：

```
mov dx,3f8h     ;将端口号3f8送入dx
in al,dx        ;从3f8h端口读入一个字节
out dx,al       ;向3f8h端口写入一个字节
```

在in和out 指令中，只能使用ax 或al 来存放从端口中读入的数据或要发送到端口中的数据。

访问8 位端口时用 al ，访问16 位端口时用ax 。



这里就不写例子了，还是搬数据，只不过这里使用 in 和 out （相对于cpu进和出）指令进行搬运，而且很多外设的操作已经封装到中断程序里面了，没必要自己从头编写。

不过需要注意的是，外设一般都会设一个内存的缓存区，你的程序通常从缓存区获取数据。

比如对于键盘输入，数据从键盘到cpu的数据缓冲寄存器再到内存中的键盘缓冲区；

对于屏幕显示，数据从内存的显存缓冲区到cpu再到屏幕。