微机原理与系统设计笔记2 | 8086CPU结构与功能

- 打算整理汇编语言与接口微机这方面的学习记录。本部分讲解8086CPU的结构和基本功能以及特性。
- 参考资料
    - 西电《微机原理与系统设计》周佳社
    - 西交《微机原理与接口技术》
    - 课本《汇编语言与接口技术》王让定
    - 小甲鱼《汇编语言》

---

1. 微处理器的外部结构

1.1 引脚

外部结构就是封装出来的输入输出引脚。8086/8088有40个引脚。

> - 8086片内片外的数据总线都是16位
> - 8088片内16位，而片外8位

第 2 章我们介绍微处理器的结构时已经说明，微处理器的外部结构表现为数量有限的输入输出引脚，这些引脚构成了微处理级总线。微处理器通过微处理级总线和其他逻辑电路连接组成主机板系统，形成系统级总线，简称系统总线。

8086 微处理器采用 40 条引脚的双列直插式封装。为了减少引脚，采用分时复用的地址/数据总线，因而部分引脚具有两种功能。8086 的引脚如图 5.5 所示。8086 CPU 的 40 条引脚中，引脚 1 和引脚 20（GND）为接地端；引脚 40（V_{CC}）为电源输入端，采用的电源电压为+5V±10%。

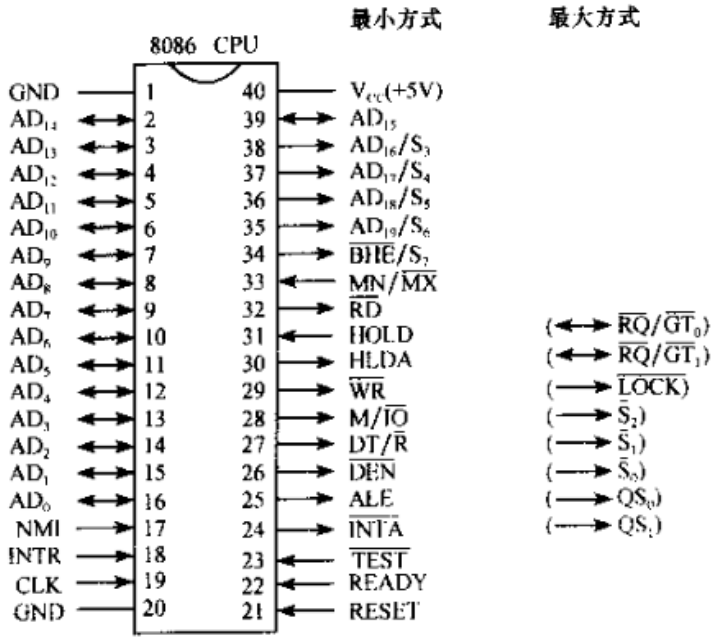


图 5.5　8086 引脚

如此前绪论所说，部分引脚专用，部分引脚复用，复用部分需要掌握其时序。

CPU的这些引脚功能：

1. 与存储器之间交换信息
2. 与I/O设备（接口）之间交换信息
3. 输入输出必要的信息

1.2 控制引脚

举个例子讲解一下这些引脚：

- 当CPU向外部（存储器/IO接口）写数据时，上图的WR引脚应为有效（低电平）

  > RD是读信号。

- 而区分操作存储器和I/O接口的是M/IO引脚
- 在此基础上可以设计逻辑电路来控制更具体的事情，比如WR和M/IO连接一个或门，就能产生IOW信号等。

1.3 地址引脚

A. 寻址空间

8086CPU有20条地址线A_{19}~A_{16}、A_{15}~A_0，可以寻址2^{20}字节的空间，也就是1M空间，地址空间大小为1MB。

B. IO端口的概念

- 操作系统设计过程中，外设部分经常提及端口，这里正经回忆一下。
- I/O接口是连接CPU与I/O设备的控制电路，在I/O接口中，有一个I/O端口寄存器，用于与CPU之间的数据交换，计算机也为其分配一个地址（端口地址），CPU也是依据这个地址与端口打交道的。
- 外设的状态存储在接口电路中的端口寄存器，如果CPU没有与接口通信，则端口寄存器保持高阻态，不向外/内输出、输入。
- 当地址总线上的地址是某接口端口寄存器的地址（选中）时，其中信息通过数据总线流入CPU。

> 总线竞争：多个设备端口同时激活。

端口如下图右下角部分所示：

- 数据输入端口
- 命令端口
- 状态输入端口

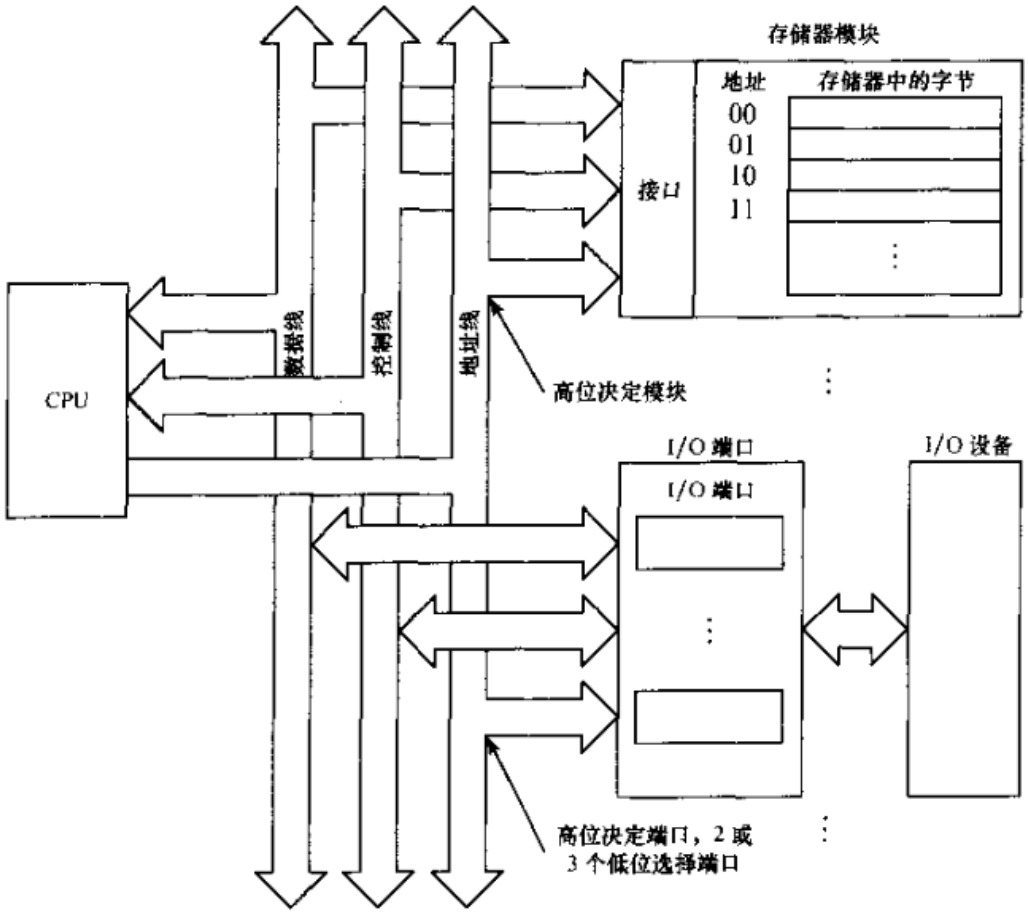


图 2.1　微处理器的外部结构

> 具体而言，外设芯片是有手册的，当我们操作显卡，就要查VGA相关的手册，鼠标和键盘就有另一个芯片来管理。
>
> 手册上会详细说明端口的作用，在程序中用指令向端口赋值即可。

C. 统一编址与独立编址

上面提到，一个IO端口至少占用一个IO地址----IO端口地址。根据端口地址和存储器是否一起编址，有：

- 统一编址
  - 缺点：浪费了存储器的空间。
  - 优点：IO驱动程序编写方便，灵活。
  - 例：51系列。
- 独立编制
  - 优点：节约存储器的空间。
  - 缺点：要多记忆操作IO的指令，IO编程不灵活。
  - 例：8086。
  - x86的IO访存指令为 in/out

  举个例子

  某个I/O端口的地址为2000H，则访问如果要输出数据到该端口，汇编语言应该这样写：

```
mov AL,01H
;这里是要输出的数据量，仅是一个参考
;该课程中这里要点一个灯
mov DX,2000H
mov DX,AL
;注意这里跟存储器的并不同，没有中括号
```

2. 微处理器的内部结构

2.1 作用

微处理器是组成计算机的核心部件，它具有下列运算和控制功能：

(1) 进行算术和逻辑运算。

(2) 具有接收存储器与 I/O 接口来的数据和发送数据给存储器与 I/O 接口的能力。

(3) 可以暂存少量数据。

(4) 能对指令进行寄存、译码并执行指令所规定的操作。

(5) 能提供整个系统所需的定时和控制信号。

(6) 可响应 I/O 设备发出的中断请求。

典型的 CPU 内部结构如图 2.2 所示。

2.2 结构

不论CPU型号如何，其内部基本都有以下结构：

典型的 CPU 内部结构如图 2.2 所示。

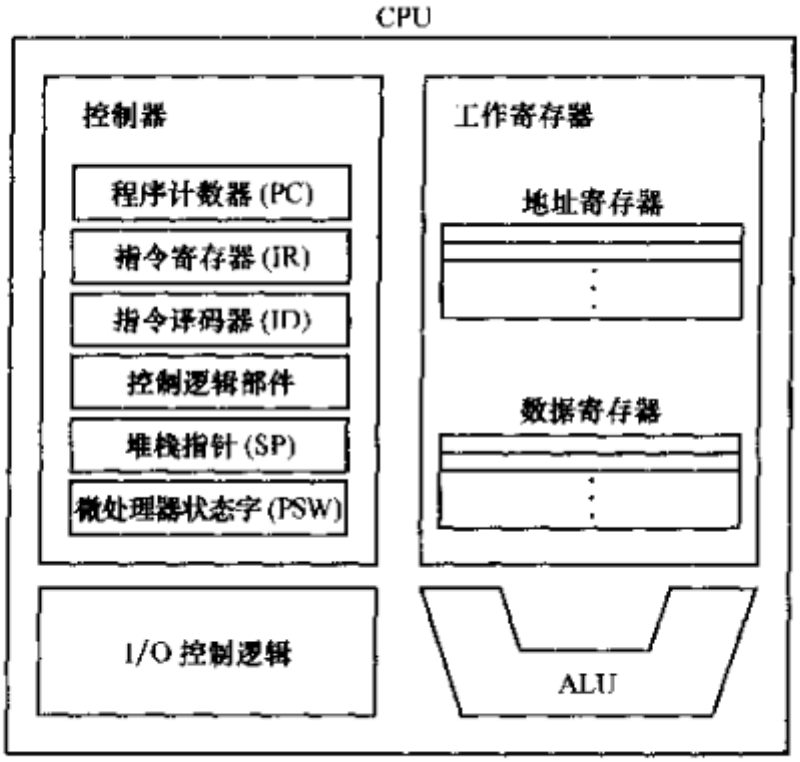


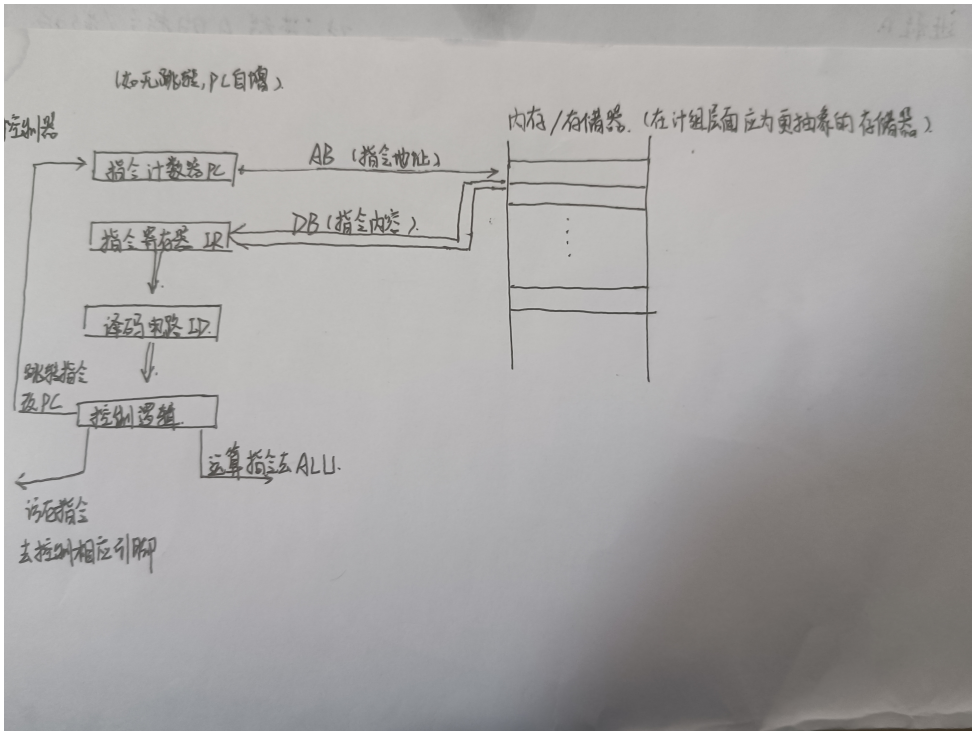
图 2.2　微处理器的内部结构

1. ALU：必须，算术逻辑运算
2. 地址寄存器可以充当数据寄存器

   数据寄存器不能充当地址寄存器
3. 控制器：负责取指令，放在指令寄存器中，译码
4. I/O控制逻辑：与外部I/O打交道，使得CPU可以响应I/O设备发出的中断请求。

2.3 控制器讲解

这部分要与其实跟计组没多大差别了，要结合CPU"取指执行"的思想来理解。



> 学到这里突然想复习 流水线 Verilog 什么的了。
>
> 因为上面的图还是很笼统，怎么判断、怎么控制时序都还没有涉及。先忍住，把接口部分学完。

上图并不完整，如果从存储器中取出的是数据data而不是指令 instructions ，则直接放入数据寄存器或者指定寄存器（也不属于控制器的范畴了）

2.4 堆栈

> 来自计操的补充：
>
> - 说 堆 特指 堆
> - 说 堆栈 指的是 栈

- 堆栈在存储器空间中，大小和位置都是编程自定义的。
- 8086中堆栈必须按字操作
- 堆栈操作的代表性指令是

```
push ax
pop ax
;如果是8086的按字操作规定，目的寄存器就不能是AL
```

- 堆栈基址寄存器:sp，初始置向栈底+1(也就是栈底再向下的一个存储单元)，这个位置是程序设定的。
- 假如执行以下操作：

```
push ax;第1步
push bx;第2步
```

  第1步中，sp先-2空出一个字，然后AX分高低八位分别存入这个空字的高低存储单元。

  第2步重复第1步操作。

  > 如果要pop堆栈中的值，则是上述逆过程，先取出栈顶的值，再sp+2。

- 堆栈溢出：

- 当一直push，向堆栈区域增加数据，超过堆栈分配空间（超过最高栈顶），则溢出
- 当一直pop，超过栈底，则溢出。
- 堆栈溢出会造成系统crush。

3. 8086/8088CPU 内部结构

2.3节讲解的是处理器的工作思路，或是说 “取指执行”的计算机思想。下面介绍8086CPU的内部结构（仍然符合2.3节的大致思路）。

微处理器的功能结构如图 2.3 所示，它主要包含两个独立的逻辑单元：执行单元 EU（execution unit）和总线接口单元 BIU（bus interface unit）。ALU 的数据总线（16 位）、队列总线用于 EU 内部、EU 与 BIU 之间的通信。

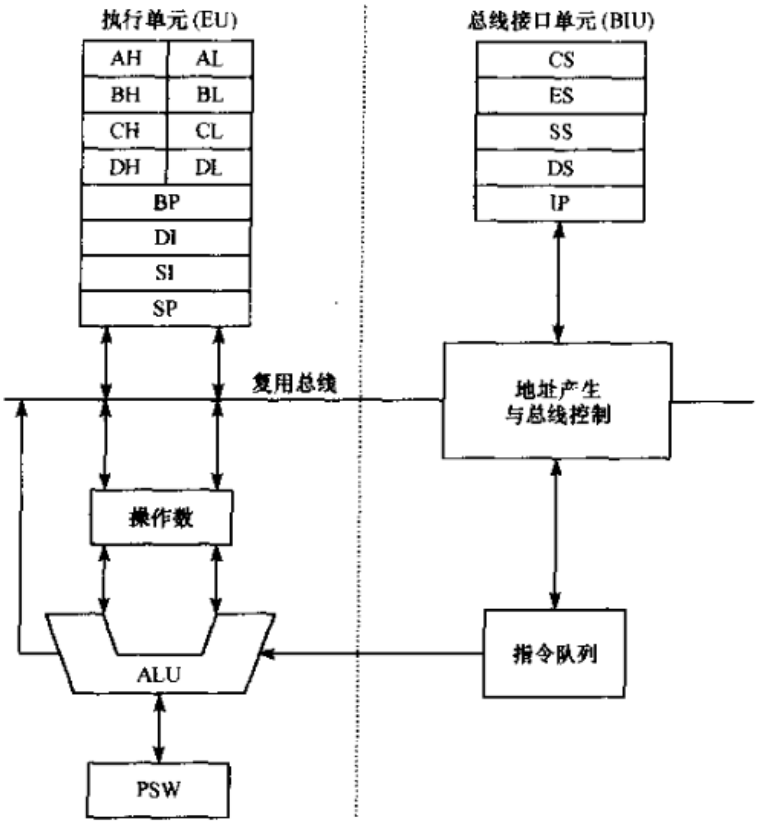


图 2.3 微处理器的功能结构

3.1 BIU

- 有一个问题，地址总线20位，意味着可以寻址1MB地址空间；而CPU内部寄存器只有16位。

  如何**用16位寄存器存放20位的地址信息**呢？
- 8086的设计是：讲存储器分为逻辑段，一个寄存器负责寻址段，一个寄存器负责寻址段内空间，也就是一个段内最多64KB（2^{16}byte）空间。
- 各种段基址寄存器以及指令寄存器就在上图BIU右上角
- 这里的转化的具体过程就是在上图的地址产生与总线控制单元进行的。
- 转化公式为：

  CS:IP(CS左移4位+IP)，也就是

  (CS << 4) + IP
- BIU右下角的指令队列充当的是2.3节的IR指令寄存器的角色，8086中有6个字节，8088中4个字节。

> 这是两个CPU内部结构唯一的区别。

总结，BIU负责外部存储器取出指令、取出数据，并将取出的指令放入指令队列，对应 “取指”。数据通过ALU总线直接送入EU。

3.2 EU

总结放在前面：负责从指令队列中获取指令，对该指令译码并执行，对应“执行”。

这里可以看出，指令队列的存在，可以使得两个部分的性能都得到提升。

EU 和 BIU 可以独立、并行执行，但相互之间会有协作。当指令队列中还没有指令时，EU 处于等待状态，当 EU 执行指令需要访问存储器或 I/O 端口时，BIU 应尽快完成存取数据的操作，这一过程可以用图 2.4 表示。

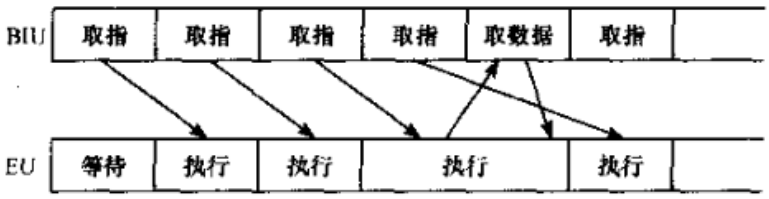


图 2.4 EU 和 BIU 单元的并行执行过程

而外部总线在上图过程中始终处于忙状态，总线的使用率也上升。

4. 8086的寄存器组织

8086内部**共有14个16位寄存器**:

- 通用寄存器 （EU左上角）
  - 数据寄存器 4个
    - AX,BX,CX,DX
    - 各个寄存器又有特殊功能，但是给我的印象不深：
      - AX—累加器（特殊功能）| AH AL
      - BX—基址寄存器（特）—段内的–DS段的
      - CX—计数器
      - DX—数据寄存器（IO）
  - 它们又都可以分为XH和XL。
- 地址指针寄存器与变址寄存器
  - 地址指针寄存器
    - SP:堆栈指针寄存器
    - BP:地址指针寄存器

    2023-02-20！！！注意SP和BP的区别：ss栈段寄存器，sp栈顶指针寄存器sp会随着带有堆栈操作的指令(比如PUSH、CALL、INT、RETF)产生变化，而BP不会，所以在带参数的子过程中用BP来获取参数和访问设在堆栈里面的临时变量。

bp默认的栈寻址寄存器

```
> BP与BX在做地址指针时的区别：
>
> ```assembly
> mov BX,002H
```

```
> mov BP,002H
> mov AL,34H
> mov [BX],AL;1
> mov [BP],AL;2
> ```
>
> 1处AL值**默认**放到了数据段的BX偏移处，2处AL值**默认**放到了堆栈段的BP偏移处。
>
> 如果要使它们不默认，可以将上面代码表示地址的中括号内加上它们的目的段基址寄存器如：
>
> ```assembly
> mov SS:[BX],AL
> ;此时BX表示堆栈偏移
> mov DS:[BP],AL
> ;此时BP表示数据段偏移
> ```
```

- 变址寄存器
  - SI：源变址寄存器
  - DI：目的变址寄存器
  - **找到的都是DS段的地址**
  - 变址寄存器中“变”的概念来自于8086对字符串的处理。

    > 具体涉及8086指令系统中的字符串操作指令。

    如将字符串搬运到存储器另一个位置，源字符串的位置需要定义在DS:SI，目的字符串的位置定义在DS:DI。

    > DS:SI==(DS<<4)+SI

    当使用movsb或movsw（无操作数指令），自动从源字符串搬运到目的字符串。（两者的区别是按字节搬运和按字搬运）

    在这个字符串操作过程中，DI和SI是在**自动增加**的，所以其名字中的“变”不言而喻。

    当DI和SI像BP一样进行普通数据段操作时，不会自己增加。
- 段寄存器
  - 8086汇编程序结构分为若干逻辑段，汇编后放到存储器的不同段。
  - CS：代码段基址寄存器
  - DS：数据段基址寄存器
  - ES：附加数据段基址寄存器
  - SS：堆栈段基址寄存器
  - 在代码段开始时，赋值DS ES SS，使其符合自己安排的位置，而CS由操作系统安排。

    所以不会出现`mov CS,AX`

    > 突然感觉这个原理有点古董。
- 控制寄存器
  - IP：指令指针寄存器，相当于2.3节的程序计数器PC

    代码段的偏移地址
  - PSW：处理器状态字寄存器，设置9个状态位。

    6个状态位表达ALU运算后的程序状态。

控制标志：控制CPU的运行状态

- DF方向控制，在字符串操作中，DF==0，变址寄存器SI DI自增；DF==1，SI DI自减

  即控制SI DI的变化方向

- IF中断允许标志，IF=1时，CPU可以响应可屏蔽中断请求（也就是外部中断）；IF=0时，CPU不响应中断请求。

  > 中断是操作系统中很重要的概念，开中断和闭中断的指令为`sti cli`

- TF陷阱标志/单步标志：TF=1时，CPU处于单步执行方式，每次执行一条指令自动执行一次特定的内部中断，具体应用就是Debug。

- 汇编语言中的`pushF`表示将PSW标志寄存器压栈，`popF`表示将栈顶出给PSW

5. 8086存储器和IO组织地址空间

5.1 地址空间

- 地址线：A19-A16，A15-A10，A9-A0
- 8086给存储器编址20根地址线，IO16根地址线（A0~A15）
- 在早期IBM pc机中，给IO分配A9-A0地址线来寻址1KB空间，
  - 前512B：为主板上的IO分配地址（000H-1FFH）
  - 后512B：给插件板上的IO分配地址（200H-3FFH）

5.2 数据存放格式

- 三种格式：字节型、字型、双字型
- 字节型数据：

  一个字节型数据对应一个地址单元。

  汇编语言设计中，字节型数据定义在存储器中的**DS段**，具体用DB这个伪指令来定义

  > 伪指令用于汇编器如何来翻译汇编代码。

- 字型数据：

  对应两个相邻的地址单元。

  定义伪指令为DW，如将字型数据5678H放入存储器0003H和0004H位置，则78H放在字的低地址0003H，56放在字的高地址0004H。

  这里有一个对准和不对准的问题。如果字数据地址为奇地址，则称为未对准，偶地址则对准（比如上面的例子就是未对准）

  对准的数据 进行 访存指令花费时间更短。未对准会多花费一个时间周期，这与数据总线的传输机制有关。

  > 为了防止自己脑袋忘记，提示：对准情况下，高字节对应高地址，地址单元为奇，走高八位数据线（一个是线，一个是字节）

- 双字型数据：

  对应两个字，也就是4个存储单元。

  定义伪指令为DD。

在汇编语言中使用[BX /BP/ SI/ DI]，是指**寄存器所存的内容**，也就是地址；使用BX BP SI DI—指的是**寄存器本身**。

5.3 存储器的分段与物理地址的形成

- 为什么要分段
  - 已经在前面提到过了：寄存器16位而地址线20根。
- 如何分段：
  - 一个逻辑段最大64K，每个逻辑段的**起始地址**必须可以被16整除

    因此理论上讲，1MB的地址空间，可以分64K(2^{16})个逻辑段，正好是16位寄存器可以描述的。
- 物理地址
  - 这个上面3.1部分也提到过，要从段基址（段的起始地址）+段偏移的形式重新得到真实的物理地址。
  - 物理地址的唯一性：

    由于段相互有叠加（按照被16整除的判断标准），所以一个单元的逻辑地址只是可能不同。但是物理地址一定不同，物理地址是站在存储器全局为每个单元分配的门牌号。

    > 虽然段在理论上（按16整除）会重叠，但实际上，汇编源程序是自己定义的各个段，操作系统分别将其装入内存，**不会发生段覆盖的情况。**
  - 编程、调试都是逻辑地址
- 物理地址（PA）的形成

  也就是上面3.1节的转化公式。

  物理地址=段基址X16（16进制左移一位）+段内偏移地址（段内有效地址）。
- 取指令
  - CPU如何实现取指令？
  - CS:IP

    即 CS(段基地址)X16+IP(段内偏移地址)，取指令所存储的物理地址

    接着8086按照计算后的物理地址去存储器找指令取出。

    > 课程这里（11讲21分钟时）提到了无条件跳转，老师讲的是代码段间的无条件跳转。
    >
    > 据我所知，这部分比较复杂，老师估计是想强调一下cs:ip的存储器取指特性，所以具体的后续再提。
    >
    > 特性：CS和IP是用户不可写入的，CS是操作系统将代码从磁盘放入内存后初始化的，IP不可写入但是会改变，除了自增外还会跳转。
  - 为了更明确说明物理地址的形成，再举一个例子，存储器写操作:

    ```
    mov [bx],ax
    ```

    这就是将ax中的值写入ds:bx中去，如果硬要扯一下第3部分EU BIU的知识，那就是EU先将16位地址BX沿内部数据总线传送到BIU，BIU停下取指操作，配合EU去进行写存储器的操作，BIU将BX放到加法器中产生物理地址，输出20位物理地址后放到地址线，AX值放到数据线。

    > 如果对齐，一个时间周期完成；如果不对齐，两个时间周期。
  - 堆栈操作(SS:SP)，跟上述过程相似。

5.4 各类指令的地址寻址

这部分跟指令系统中的寻址方式是分不开的，指令系统会在下一部分整理。

关于各种逻辑段偏移寄存器指定与不可指定的灵活性，下面是一个小总结：

表 2.1　各类指令的地址信息

访问存储器类型	默认段	可指定的段地址	段内偏移地址
取指令代码	CS	/	IP
堆栈操作	SS	/	SP
一般数据操作	DS	CS，ES，SS	根据寻址方式确定
字符串操作源地址	DS	ES，SS，CS	SI
字符串操作目的地址	ES	/	DI
BP 用作基址寄存器时	SS	DS，ES，CS	根据寻址方式确定

这个表里总结了操作中的默认寄存器以及可以替代的寄存器，比如取指令CS必须与IP搭配，不得自己指定寄存器寻址或者段地址。