

8086 Assembly

CH0 Other

4.5 画图说明下列语句所分配的存储空间及初始化的数据值。

(1) `BYTE_VAR DB 'BYTE',12,-12H,3 DUP(0,?,2 DUP(1,2),?)`

(2) `WORD_VAR DW 5 DUP(0,1,2),?,-5,'BY','TE',256H`

(2)

内存单元	数据值	内存单元	数据值	内存单元	数据值
20000/1BH	00H	20010/2BH	02H	20020/3BH	FBH
20001/1CH	00H	20011/2CH	00H	20021/3CH	FFH
20002/1DH	01H	20012/2DH	00H	20022/3DH	59H
20003/1EH	00H	20013/2EH	00H	20023/3EH	42H
20004/1FH	02H	20014/2FH	01H	20024/3FH	45H
20005/20H	00H	20015/30H	00H	20025/40H	54H
20006/21H	00H	20016/31H	02H	20026/41H	56H
20007/22H	00H	20017/32H	00H	20027/42H	02H
20008/23H	01H	20018/33H	00H		
20009/24H	00H	20019/34H	00H		
2000A/25H	02H	2001A/35H	01H		
2000B/26H	00H	2001B/36H	00H		
2000C/27H	00H	2001C/37H	02H		
2000D/28H	00H	2001D/38H	00H		
2000E/29H	01H	2001E/39H	??		
2000F/2AH	00H	2001F/3AH	??		

1. 一个db中两个dup，取length取的是谁
2. 8086未定义堆栈段时堆栈位置，以及是否需要像处理DS一样处理SS和SP
3. assume的作用是什么（尤其是多段的时候）
4. 8086汇编中，段地址指的是16位还是20位
5. `mov ax, buffer \ mov ax, [buffer] \ mov ax, offset buffer \ mov ax, [offset buffer]`
6. 单操作数+Mem（或者两个Mem或一Mem加一IMM的）的一定要表明数据类型，如DEC【DI】

CH1 汇编机器指令

一、数据转移指令

1. MOV, PUSH 和 POP

- push 和POP操作只能对dw进行操作（16bit，两个字节，一个字）
- 为什么 `mov ax, [buffer+1] ; mov ax, [0004h]` 不能使用 `mov ax, [offset buffer+1] ; mov ax, 0004h`
- PUSH、POP（以及似乎只有PUSH，POP，MOV可以操作段寄存器）
 - PUSH/POP REG
 - PUSH/POP MEM
 - PUSH/POP SEGREG（但不能是CS）
 - 不允许立即数操作，也不允许立即数寻址

2. XLAT

- 换码指令（查表转换指令）：`XLAT ; al ← ds:[bx+al]`（BX: 字节表格(长度不超过256)首地址的偏移地址）

3. IN, OUT

- 只限于使用EAX、AX或AL
- 端口号在0~255之间，则端口号直接写在指令中 `IN AL, PORT`
- 端口号大于255，则端口号通过DX寄存器间接寻址，即端口号应先放入DX中 `IN AL, DX`

4. 串处理指令

- 涉及到寄存器时只能使用累加器
- 目标操作数一定是ES:[DI]，源操作数一定是[SI]（SI的段前缀可以不是DS）
- MOVS：必须是 `MOVS ES: type ptr [DI], DS:[SI]`，其中只有SI的段超越前缀可更改。而目标操作数DI、源操作数SI和DI的段前缀ES都不可变
- STOS：把AL或AX数据传送到目的地址DST `STOS ES:BYTE PTR[DI]`

STOSB ； 字节串存储： $ES:[DI] \leftarrow AL$ ； $DI \leftarrow DI \pm 1$

STOSW ； 字串存储： $ES:[DI] \leftarrow AX$ ； $DI \leftarrow DI \pm 2$

- LODS：把数据从SRC传送到寄存器AX或AL `LODS DS:BYTE PTR[SI]`

LODSB ； 字节串读取： $AL \leftarrow DS:[SI]$ ； $SI \leftarrow SI \pm 1$

LODSW ； 字串读取： $AX \leftarrow DS:[SI]$ ； $SI \leftarrow SI \pm 2$

- INS和OUTS用的是DX寄存器表示端口号： `INS ES:BYTE PTR[DI], DX`， `OUTS DX, DS:BYTE PTR[SI]`
- CMPS：DST-SRC
- SCAS：DST-AX

5. Load Segment 指令（LDS、LES、LCS、LSS）

- 指令源操作数只能用存储器寻址方式
- 低地址中的16位数据装入指针寄存器，高地址中的16位数据装入段寄存器

二、数据处理指令

- **BCD?**

1. 加减法

- **OF、CF：溢出与进位** [计算机判断溢出的原理？ - 北极的回答 - 知乎](#)
 - 若CF=1，对无符号数而言发生了溢出
 - 若OF=1，对带符号数而言发生了溢出
 - 一旦发生溢出，结果就不正确了
- NEG：理解方式：用**零减去操作数**，然后结果返回操作数
 - 对操作数所能表示的最小负数(例若操作数是8位则为-128)求补，原操作数不变，但OF被置1
 - 当操作数为0时，CF清0；对非0操作数求补后，CF置1

2. 乘法 IMUL、MUL、IDIV和DIV



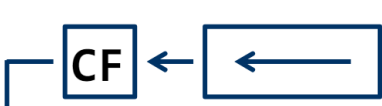

- MUL和IMUL
 - 乘积的高一半为0，则CF、OF均为0，否则CF、OF均为1：**这样可以检查结果是字节、字或双字**
 - 乘积的高一半是低一半的符号扩展，则CF、OF均为0，否则CF、OF均为1。其实质和MUL情况下一样，主要用于判断结果是字节、字或双字
- DIV和IDIV：源操作数不能是立即数

3. 逻辑指令

- 逻辑运算：CF和OF清0，**影响SF、ZF及PF**，AF不定

4. 移位指令

- DST可以是8位、16位或32位的寄存器或存储器操作数
- CF存最后移出去的那一位
- ZF, SF, PF按具体情况设置（循环移位不改）
- OF（CNT为1时，最高位变化则置1）

名称	格式	功能	标志
循环左移	ROL DST, CNT		(1) CF中总是最后移进的位; (2) 不影响ZF、SF、PF; (3) 当CNT=1时, 移位使DST符号位改变, 则OF置1, 否则清0
循环右移	ROR DST, CNT		
带进位循环左移	RCL DST, CNT		
带进位循环右移	RCR DST, CNT		

三、跳转指令

1. JMP和JCC

- jmp指令中对于位移量和偏移量的区分
 - 区分??? https://blog.csdn.net/weixin_41890599/article/details/99556319
 - JCC: 条件转移指令只能使用段内直接寻址方式
 - 无条件转移 (JMP) 和无条件子程序调用 (CALL) 可用四种方式的任何一种
- 条件转移只能段内短转移short (为什么) 还有JCXZ
 - 条件转移指令只能使用段内直接寻址方式

2. CALL和RET

- call命令不能短转移 (short)
- ret imm8 中, imm8表示额外返回的字节数 (不是字数)

CH2 寻址方式

一、数据寻址: 7种寻址

https://www.cnblogs.com/Hardworking666/p/17374792.html#4_74

1. 基址变址

- 只要指令寻址时使用了BP, 计算物理地址时约定段是SS段

- 指令寻址时使用了除BP以外的其它寄存器，计算物理地址时约定段为DS段
- 寄存器间接寻址 和 寄存器相对寻址 中，可以是基址BX，BP；也可以是变址SI，DI

1. 若寄存器间接寻址或相对寻址是 `mov ax, 80h[di]`，则默认段基址是ES还是DS

✓ 指针及变址寄存器（4个，16位）

◆ 堆栈指针寄存器：SP

- 存放当前堆栈段栈顶偏移量
- 总是与SS堆栈段寄存器配合存取堆栈中的数据

◆ 基址指针寄存器：BP

- 存放地址的偏移量（或数据）
- 若存放偏移量时，缺省情况与SS配合

◆ 变址寄存器：SI

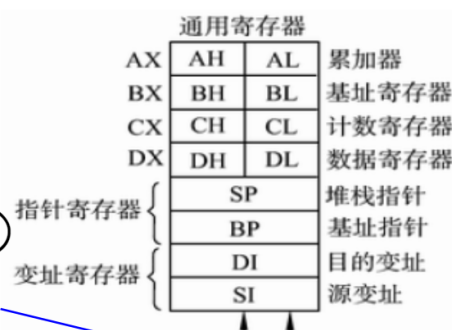
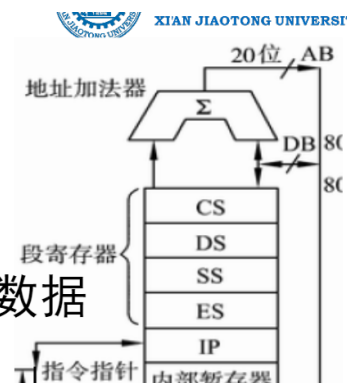
- 存放串数据的源地址偏移量（或数据）
- 若存放偏移量时，缺省情况与DS配合

◆ 变址寄存器：DI

- 存放串数据的目的地地址偏移量（或数据）
- 若存放偏移量时，缺省情况与ES配合

■ 注：ESP、EBP、ESI、EDI（32位）只有386以上使用

实模式使用SP、BP、SI、DI，保护模式使用ESP、EBP、ESI、EDI



`MOVSB [DI], [SI]`
`MOVSB ES:[DI], DS:[SI]`

段寄存器名如何指定？

如果是对一般数据的寻址, 4个段寄存器都可以用

段基址和偏移量的约定情况

段寄存器名：偏移地址

数据操作

操作类型	约定寄存器	允许指定的寄存器	偏移量
1. 指令	CS	无	IP
2. 堆栈操作	SS	无	SP
3. 普通变量	DS	ES、SS、CS	EA
4. 字符串指令的源串地址	DS	ES、SS、CS	SI
5. 字符串指令的目标串地址	ES	无	DI
6. BP用作基址寄存器	SS	DS、ES、CS	EA

指令中操作数地址书写时：EA=DS:EA；[SI]=DS:[SI]，[DI]=ES:[DI]；
如果EA中有BP，EA=SS:EA，如[BP+X]=SS:[BP+X]

	16位寻址	32位寻址
位移量	0, 8位, 16位	0, 8位, 32位
基址寄存器	BX, BP	任何32位通用寄存器 (包括ESP)
变址寄存器	SI, DI	除ESP以外的32位通用 寄存器
比例因子	无	1, 2, 4, 8

$$EA = \text{基址} + (\text{变址} * \text{比例因子}) + \text{位移量}$$

2. 关于EA与label

◆ 操作数地址可由符号地址表示

MOV AX, DS:[2000H]

MOV AH, VALUE

MOV AH, [VALUE]

VALUE的属性应该为地址

- 两者等效，这时VALUE是操作数的符号地址，但必须在程序中提前定义属性和数值

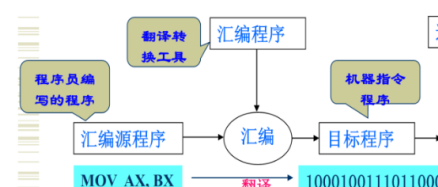
◆ 使用变量时，要注意变量的属性

VALUE DB 10

✗ MOV AX, VALUE

warning A4031: Operand types must match

✓ MOV AX, WORD PTR VALUE



- ◆ 实际上在汇编语言源程序中一般所看到的直接寻址方式都是用符号表示的，只有在DEBUG环境下，才有[2000H]这样的表示，存储器中二进制代码 反汇编 的结果

理解时，注意区分：

什么由汇编工具汇编时处理、如何处理？什么由CPU执行程序时处理、如何处理？

二、指令寻址

- short, near ptr
- fat ptr, dword ptr
- https://blog.csdn.net/weixin_41890599/article/details/99556319

CH3 伪指令

一、数据段定义


```
data segment
```

```
M1 DB 15, 67H, 11110000B, ?
```

```
M2 DB '15', 'AB$'
```

```
M3 DW 4*5
```

```
M4 DD 1234H
```

```
M5 DB 2 DUP(5, 'A')
```

```
M6 DW M2 ;M2的偏移量
```

```
M7 DD M2 ;M2的偏移量、段基址
```

```
data ends
```

注意：常数和字符串在存储器中的放置次序！

存储单元14713H可以对应表示为不同的逻辑地址1470:0013, 1471:0003等

1470: 0001	67	
1470: 0002	F0	
1470: 0003	?	
1470: 0004	31	← M2
1470: 0005	35	
1470: 0006	41	
1470: 0007	42	
1470: 0008	24	
1470: 0009	14	← M3
1470: 000A	00	
1470: 000B	34	← M4
1470: 000C	12	
1470: 000D	00	
1470: 000E	00	
1470: 000F	05	← M5
1470: 0010	41	
1470: 0011	05	
1470: 0012	41	
1470: 0013	04	← M6
1470: 0014	00	
1470: 0015	04	← M7
1470: 0016	00	
1470: 0017	70	
1470: 0018	14	
1470: 0019		

1. EQU和=

- EQU伪操作中的表达式名是不允许重复定义的，而“=”伪操作中的表达式名则允许重复定义

- “=”可以看作“表达式”定义
- “EQU”可以看作“等同”定义

- $k=1$
 $k=k+5$ 允许
 MOV AL, k 汇编后等同

$k \text{ EQU } 1$
 $k \text{ EQU } k+5$ 不允许
 MOV AL, 6

**“=”可以看作“表达式”定义
“EQU”可以看作“等同”定义**

- 表达式中的变量名是指变量的数值
- 表达式值汇编时一定要能确定具体数值！

2. 地址计数器

- 在汇编程序对源程序汇编过程中，使用地址计数器保存当前正在汇编的基本处理单位在存储器中的首地址（如指令的首地址，操作数首地址）
 - 内容为指令或数据首字节存储单元的偏移地址
- ORG 伪操作：用来设置当前地址计数器的值，即分配后续数据、指令的存储器开始地址
 - ORG 常数表达式 (n)

- ```

1 vectors segment
2 org 10 ; 10=000AH
3 vect1 dw 4567h ; 偏移地址值为000AH
4 org 20
5 vect2 dw 9876h ; 偏移地址值为0014H
6 vectors ends

```

- EVEN: 使下一个变量或指令地址开始于偶字节地址
  - 使得地址计数器值=2的倍数
- ALIGN 伪操作:
  - 格式: ALIGN boundary ; 其中 boundary 必须是 $2^n$

### 3. Label

- 名字项: 可以是指令标号或伪操作的变量、过程名、段名, 数字不能出现在名字的第一个字符位置
- 标号/变量 有3个属性: 段地址 偏移地址 类型
- 如何区分标号和变量

### 4. 关系运算符

- EQ、NE、LT、GT、LE、GE
- 关系运算符的两个操作数是数字或同一段内的两个存储器地址
- 计算的结果应为逻辑值
  - 结果为真, 逻辑值=0FFFFH;
  - 结果为假, 逻辑值= 0000H

### 5. 数值回送操作符

- Length+变量 回送由 dup 定义的变量的数据个数, 其他情况返回 1
- SIZE+变量: 功能 LENGTH\*TYPE
- TYPE+变量 (或标号) :
  - 变量: DB-1、DW-2、DD-4、DQ-8、DT-10
  - 标号: NEAR--1、FAR--2

## 二、宏定义: 结尾直接ENDM, 不是NAME+ENDM

### 1. 符号1&符号2

- 文本替换操作符。宏展开时，合并前后两个符号形成一个符号
- 符号可以是操作码、操作数或是一个字符串

### 宏定义：（源程序中定义）

```

leap macro cond, lab
 j&cond lab
endm

```

### 宏调用：（源程序中调用）

```

leap z, there
...
leap nz, here

```

### 宏展开：（汇编时展开）

```

1 jz there
...
1 jnz here

```

## 2. %表达式

- 将%后面的表达式立即求值转换为数字，并在展开时用这个数取代哑元，宏调用时使用

### 宏定义：

```

MSG MACRO COUNT, STRING
 MSG&COUNT DB STRING
ENDM

ERRMSG MACRO TEXT
 CNTR=CNTR+1
 MSG %CNTR, TEXT
ENDM

```

### 宏调用：

```

...
CNTR=0
ERRMSG 'SYNTAX ERROR'
...
ERRMSG 'INVALID OPPERAND'

```

### 宏展开：

```

...
CNTR=0
2 MSG1 DB 'SYNTAX ERROR'
...
2 MSG2 DB 'INVALID OPPERAND'

```

- 如上图的 `cntr=cntr+1` 什么时候使用不会体现在代码中

## 3. :REQ 和 :=<>

- :REQ: 指定某个变元必须有。调用时必须有对应的实元，否则汇编时出错
- :=: 为宏变元提供缺省值

## 4. 重复汇编

- ```
1 array label byte
2 IRP K, <1,2,3,4,5>
3 db 'NO.&K'
4 ENDM
```

- ```
1 array label byte
2 IRPC K, 12345
3 db 'NO.&K'
4 ENDM
```

- IRP是<>内的元素，K是哑元
- IRPC是字符串内的每个字符

## CH4 IO与中断

### 一、IO相关

- 从程序设计的角度看，接口由一组寄存器组成，是完成输入输出的桥梁
- 端口地址64kB个，但是端口地址：只有两种方式给出
  - 直接寻址：PORT（00H~FFH）
  - 寄存器间接寻址：DX（0000H~FFFFH）
- 累加器：只能使用累加器：
  - AX（16位字操作）
  - AL（8位字节操作）
- 高速：直接存储器存取（DMA）方式
- 低速：中断

### 1. 程序直接控制I/O方式

- 在输入输出之前首先查询外设的状态

- ```

1  WAIT:
2      IN AL, 72H ; 读取输入口的状态
3      TEST AL, 80H ; 测试状态寄存器的最高, 80H=10000000B
4      JZ WAIT ; 若状态位=0, 则继续测试等待
5
6      MOV AL, VAR
7      OUT 70H, AL ; 输出数据

```

二、中断相关

1. 中断命令寄存器

- ```

1 IN AL, 20H
2 OR AL, 20H ; 00100000B
3 OUT 20H, AL

```

- 结束硬件中断指令：中断服务程序中，硬件中断处理结束前，应将EOI置1
- 硬件中断服务程序结束前要有这几条指令

### 2. 中断分类

- 外部中断
  - 可屏蔽外部中断
  - 不可屏蔽外部中断 (int 2)
- 内部中断 (都不可屏蔽)
  - 除零 (int 0)
  - 。 。 。

### 3. 中断时CPU响应过程

- 读取中断类型号N
- pushf
- 保存被中断程序断点 (push cs + push ip)
- 禁止硬件中断和单步中断 (IF = 0 + TF = 0)
- 设置CS: IP (CS = 0000:[4×N + 2] + IP = 0000:[4×N])

### 4. 一般中断处理程序设计格式：注意关中断和开中断时机 和 EOI

- 保存现场：寄存器入栈
- 开中断和TF (可选)
- 中断处理程序主体

- 中断结束 (EOI) (硬件中断时)
- 关中断(CLI)
- 恢复现场
- IRET

### 三、DOS系统调用

#### 1. 存取中断向量的DOS功能调用

- 设置中断向量
  - 预置: AH=25H (子功能号)
  - AL=中断类型号
  - DS:DX=中断向量 (中断程序入口地址)
- 读取中断向量
  - 预置: AH=35H (子功能号)
  - AL=中断类型号
  - 返回: ES:BX=中断向量 (中断程序入口地址)