

船用微型计算机及接口技术

第四章知识整理

前言

个人整理总结，仅作为复习参考使用，并不权威，如有错误纰漏，欢迎指正，请联系
2124436512@qq.com 或 y0r4h21@whut.edu.cn。此文档完全免费
这章真的很……无聊？

第一节 汇编语言源程序（这节课做了解）

汇编语言源程序结构

1.汇编语言源程序与汇编程序

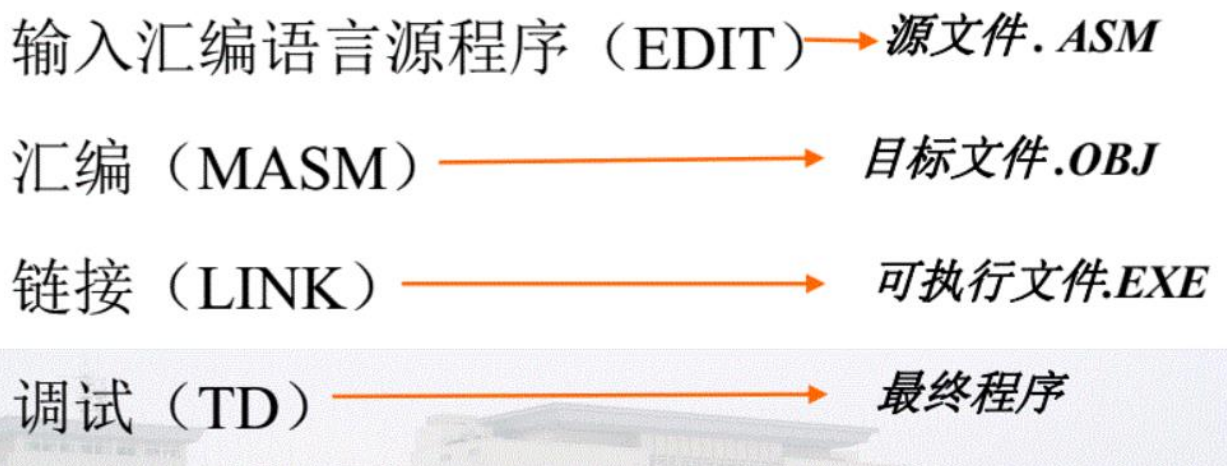
汇编语言源程序和汇编程序是完全不一样的东西。

汇编语言源程序就是我们使用各种助记符编写的程序。

汇编程序是源程序的编译程序系统，它就是将汇编语言源程序翻译为电脑真正能看懂的机器语言目标程序的一个系统。



2.汇编语言程序设计与执行过程



汇编语言语句类型及格式

汇编语言语句有两类：指令性语句、指示性语句

指令性语句是 CPU 执行的语句，能够生成目标代码。

指示性语句是 CPU 不执行，而由汇编程序执行的语句，不生成目标代码。

简单来说，指令性语句就是前面所学的那些用助记符编写的语句，最后会被翻译成由 0 和 1 组成的机器语言。

指示性语句不会被翻译为由 0 和 1 组成的机器语言，它一般是那些告诉 CPU 某些信息的语句。

指令性语句：

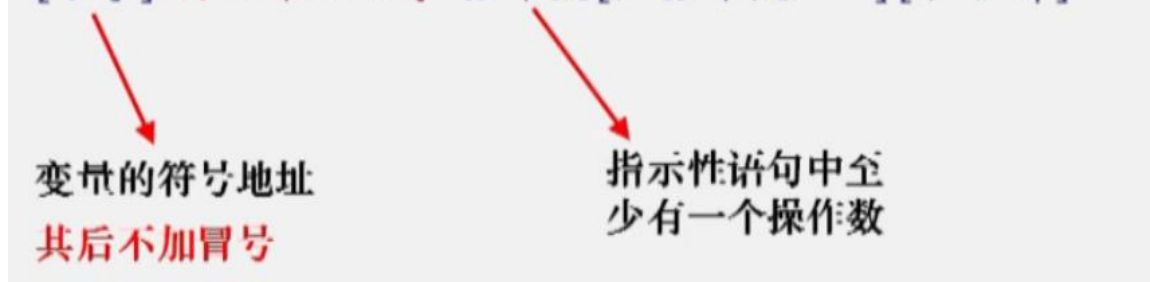
[标号：] [前缀] **助记符** [操作数]，[操作数][； 注释]



指示性语句的操作数个数可以是 0,1,2,3 个

指示性语句格式：

[名字] **伪指令助记符** 操作数[, 操作数, ...][； 注释]



“标号”和“名字”不太一样。“标号”后面有冒号，用在指令性语句前。“名字”后不加冒号，用在指示性语句前。它俩均由英文字母、数字及专用字符组成，长度不能超过 31 个字符且不能用数字来开头

汇编语言语句中的操作数（相对重要一些）

前面说 8086 指令系统中，操作数基本分为三种：立即数、寄存器、存储器。

严谨的说，操作数分为：寄存器、存储器单元、常量（这个其实就是立即数）、变量或标号、表达式。（所以下面这个题答案为“错误”）


10. 汇编语言中的操作数只能是立即数、寄存器和存储器操作数。

☐ 正确

☒ 错误


常量有数字常量与字符串常量

数字常量

字符串常量  用单引号引起的字符或字符串

例： 'A'

MOV AL, 'A'

例： 'ABCD'  汇编时被译成对应的ASCII码41H, 42H, 43H, 44H

变量代表内存中的数据区，在程序中视为存储器操作数。

变量具有以下属性：

- 1.段值——变量所在段的段地址
- 2.偏移量——变量单元地址与段首地址之间的位移量
- 3.类型——字节型、字型、双字型

表达式有以下几种：算术运算、逻辑运算、关系运算、取值运算、属性运算、其它运算。

算术运算和逻辑运算符

算术运算符

+, -, *, /, MOD

逻辑运算符

AND, OR, NOT, XOR

例：

MOV AL, 8 AND 4

MOV AL, 8+4-1

取值运算符用于分析存储器操作数的属性。其中常用的有 OFFSET 与 SEG。

OFFSET 用于取其后续变量或标号的偏移地址。

SEG 用于取其后续变量或标号的段地址。

也就是说。LEA BX,DATA 和 MOV BX,OFFSET DATA 是等价的。但是由于没有其它用于取段地址的指令，所以 SEG 比较特殊。

还有 TYPE、LENGTH、SIZE 运算符，功能如图



属性运算符用于指定其后存储器操作数的类型，运算符为 PTR。

举例：

MOV BYTE PTR[BX],12H 与 MOV WORD PTR[BX],12H

前者指定[BX]对应单元操作数为字节型，所以直接在[BX]对应单元存入了 12H

后者则指定其为字型，所以存入的其实是 0012H，并且[BX]对应位置存入 12H，[BX+1]对应位置存入 00H。

其实我们到目前为止用了很多的 “[]” 也是一个运算符，它里面的内容为操作数的偏移地址。

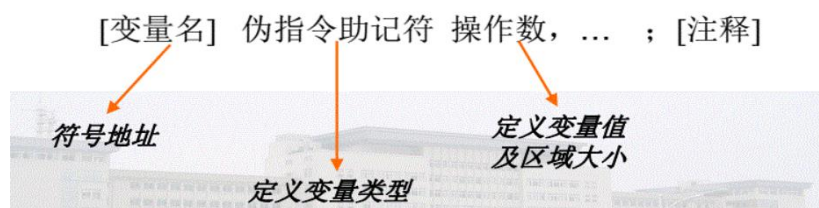
还有段重设运算符，形式为“段寄存器名：[]”，比如“ES：[BX]”，用于修改默认的段基地址（例如在使用间址寄存器时进行段重设操作）。

第二节 伪指令

伪指令在程序里也常常出现，为尽量减少负担，接下来直接结合例子总结各伪指令。

数据定义伪指令

数据定义伪指令用于定义数据区中变量的类型和大小。其格式如图



此处的“操作数”可以是常数、表达式、运算符

伪指令助记符有以下几种，如图

DB	定义的变量为字节型
DW	定义的变量为字类型（双字节）
DD	定义的变量为双字型（4字节）
DQ	定义的变量为4字型（8字节）
DT	定义的变量为10字节型

用的最多的就是 DB 和 DW。

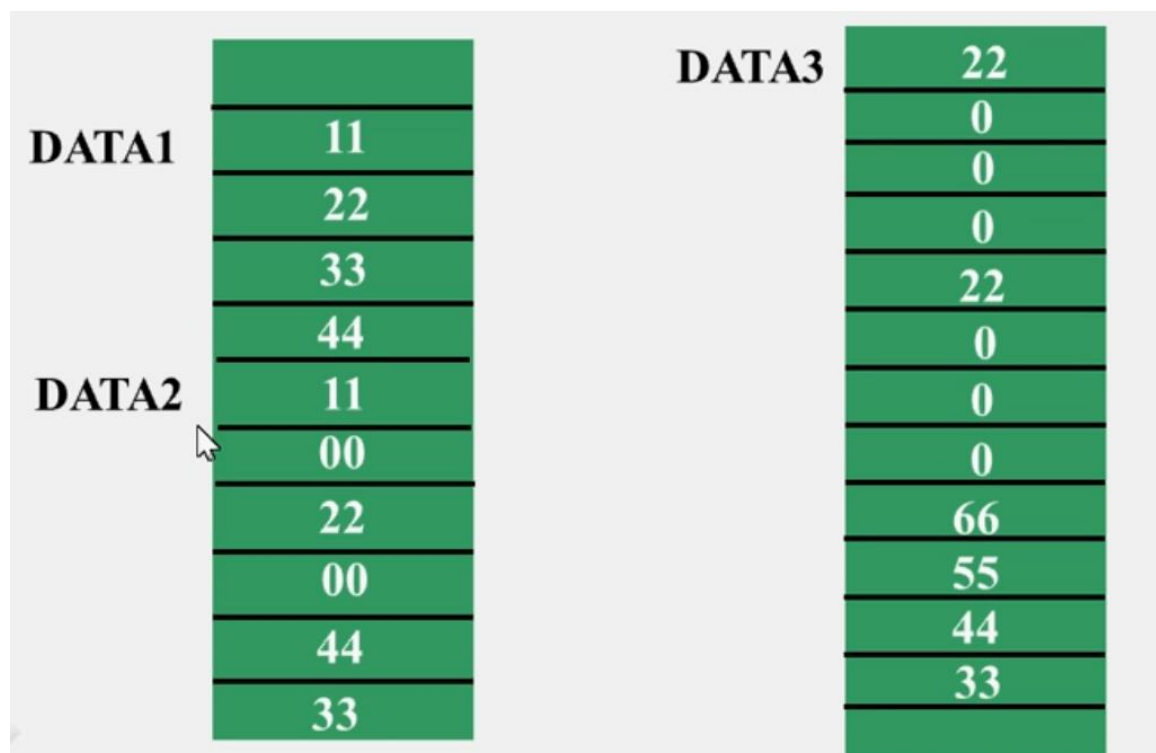
举例如下（这三条语句是连续的）：

```
DATA1 DB 11H,22H,33H,44H
```

```
DATA2 DW 11H,22H,3344H
```

```
DATA3 DD 11H*2,22H,33445566H
```

其中，DATA1\2\3 就是“变量名”，DB\DW\DD 就是伪指令助记符（表明类型）
它们在内存中的存放形式如图



解释：DATA1 指定为字节型，所以其变量值都是以一个单元为单位。

DATA2 指定为字型，其变量值以两个单元为单位，虽然写的是 11H\22H 这种，但其实存入的是 0011H\0022H，没特定给出的都默认填入 0。

DATA3 指定为双字型，以四个单元为单位，11H*2 就是十六进制的乘法，和十进制的运算方法是一样的，结果是 22H，最后存入 00000022H。

备注：定义字符串必须要用 DB，例如

例：



当需要为一个数据区的各单元设置相同初值时，会使用重复操作符。

格式：

[变量名] 伪指令助记符 n DUP (初值, ...)

例如：M1 DW 20 DUP(0)

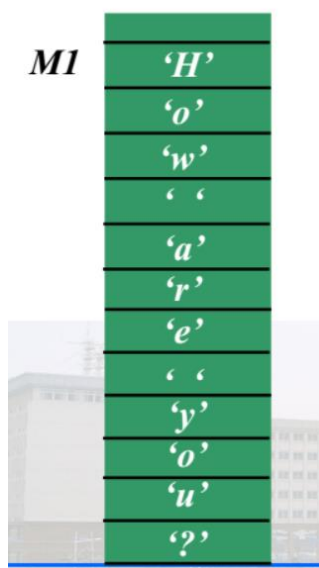
M1 为变量名，DW 为伪指令助记符，n 为重复次数，DUP(0)规定这 20 个 2 单元的值都是 0。（DW 代表定义的变量为字型，即两个单元为一个单位）

如果不需要设置初值，也可以采用“？”符号，例如：

M1 DW 20 DUP(?)

？就代表着随机数，占一个字节单元，上面的例子就意味着给 M1 所占有 40 个单元的值都定为随机数。

如果有一语句 M1 DB 'How are you?', 那么其在内存中的形式如图。



备注：定义字符串时，空格也占一个位置，其存入的是 ASCII 码。

其实“变量名”是可以不要的，比如可以直接写一条 DB 4 DUP(?)。

符号定义伪指令

符号定义伪指令 EQU。其可将表达式的值赋给一个名字，当源程序中需要多次引用某一表达式的时候可以用 EQU 伪指令，便于程序维护。

■ 格式：
■ 符号名 EQU 表达式

例如：

CONST EQU 100

就是规定 CONST 就代表 100，之后在程序中只要出现 CONST，就代表是 100

段定义伪指令

格式如图

段名 SEGMENT [定位类型] [组合类型] ['类别']
:
段名 ENDS

说明逻辑段的起点

说明不同模块中同名段的组和连接方式

段定义伪指令其实我们已经用过很多了，就是这种 SEGMENT 和 ENDS 的组合

DSEG SEGMENT 'DATA'
DATA1 DW 0F865H
DATA2 DW 360CH
DSEG ENDS

定位类型和组合类型略

DSEG 代表段的名称，大可以用任何名字来称呼

‘DATA’ 就代表着这个段是数据段


变量在逻辑段内的位置就代表了它的偏移地址。按上面那个例子，DATA1 和 DATA2 在数据段就是低地址到高地址依次排列的。

设置段寄存器伪指令

格式：

ASSUME 段寄存器名:段名[, 段寄存器名:段名, ...]

例子如图

```
CSEG  SEGMENT 'CODE'
       ASSUME      CS:CSEG,DS:DSEG
START: MOV    AX, DSEG
      MOV    DS, AX
      LEA    SI, DATA1
      MOV    AX, DATA1
      ADD    AX, DATA2
      MOV    [2800H], AX
      HLT
CSEG  ENDS
      END    START
```

ASSUME 用以说明某个逻辑段的性质（写在代码段）

值得注意的是，代码段是每个源程序都必须要有，数据段也基本是必有的。所以 CS 和 DS 一般都会出现在 ASSUME 里。其次如果有串操作指令，那么一定要有附加段。如果有堆栈操作，那么一定要有堆栈段。

结束伪指令

没啥好说的，就是“END[标号]”，代表语句的结束。“标号”可有可无，但一般都推荐注明（哪怕不用注明）

过程定义伪指令

用于定义过程体。（相当于 C 语言里不需要输入形式参数的函数）

格式:

过程入口的
符号地址

过程名 PROC [NEAR / FAR]

⋮

RET

过程名 ENDP

过程名就相当于函数名，PROC 和 ENDP 是一组指令，代表过程的开始和结束。

“NEAR/FAR”二选一用，可写可不写，如果用 NEAR 就代表是近过程，用 FAR 就代表是远过程（粗略的说，近过程就是段内调用，远过程就是段间调用）。

定义延时子程序

```
DELAY PROC
    PUSH BX
    PUSH CX
    MOV BL, 2
NEXT:  MOV CX, 4167
W10M:  LOOP W10M
    DEC BL
    JNZ NEXT
    POP CX
    POP BX
    RET
DELAY ENDP
```

定义子程序以 PROC 起，以 ENDP 结束，并且子程序的末尾必须要有 RET 指令（即 return，用于返回断点）。就像 C 语言里定义一个函数一样

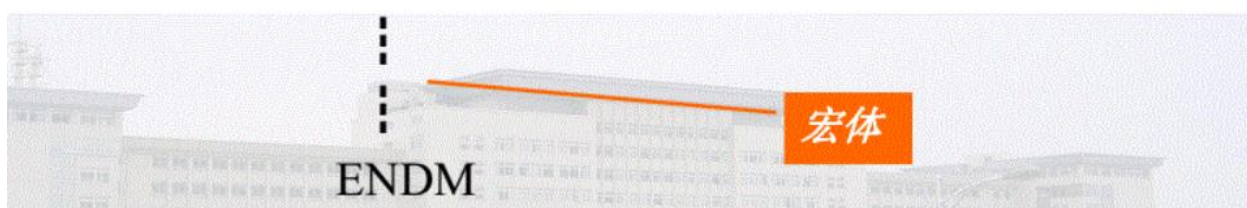
```
int function(){
    .....
    return xxx;
}
```

调用子程序：CALL 过程名，比如 CALL DELAY

宏定义伪指令

宏 \longrightarrow 源程序中由汇编程序识别的具有独立功能的一段程序代码格式：

宏命令名 MACRO <形式参数>



宏定义像是 C 语言里带有形式参数的函数。

可以说过程定义 PROC 是把一段不需要输入参数的指令压缩成一段, 用一个代号代替。

宏定义 MACRO 则是把一段需要输入参数的指令压缩成一段, 用一个代号代替。

不过, MACRO 不需要 RET。

宏定义与宏调用例

■ 定义宏：

```
DADD MACRO X, Y, Z
    MOV AX, X
    ADD AX, Y
    MOV Z, AX
ENDM
```

■ 宏调用：

```
■ DADD DATA1, DATA2, SUM
```

■ 汇编后源程序中的宏展开：

```
■ MOV AX, DATA1
■ ADD AX, DATA2
■ MOV SUM, AX
```

其他伪指令

ORG 指令：用于规定段内程序代码或变量的起始偏移地址。格式为 ORG 表达式。一个例子如图。

```
DATA SEGMENT
    ORG 1200H
    BUFF DB 1,2
DATA ENDS
```

变量BUFF的偏移地址=1200H

第三节 功能调用

BIOS、DOS功能调用

■ BIOS

- 驻留在ROM中的基本输入/输出系统
 - 加电自检，装入引导，主要I/O设备处理程序及接口控制

■ DOS

- 磁盘操作系统

DOS功能/BIOS功能调用是调用系统内核子程序

DOS功能与BIOS功能均通过**中断方式调用**

BIOS中断
DOS中断

BIOS 和 DOS 的介绍略，这里只介绍 DOS 功能调用。

DOS 功能调用

调用格式：

MOV AH, 功能号

<置相应参数>

INT 21H

DOS 中断类型码固定为 21H

DOS功能调用的基本步骤

- 将调用参数装入指定的寄存器；
- 将功能号装入AH；
- 按中断类型号调用DOS中断；
- 检查返回参数是否正确。

入口参数/出口参数

功能调用前，先按照所需功能修改 AH 的值和调用参数值即可。

INT 21H 功能调用使用说明如下:

(1) 入口: AH=00H 或 AH=4CH

功能: 程序终止。

(2) 入口: AH=01H

功能: 读键盘输入到 AL 中并回显。

(3) 入口: AH=02H, DL=数据

功能: 写 DL 中的数据到显示屏。

(4) 入口: AH=08H

功能: 读键盘输入到 AL 中无回显。

(5) 入口: AH=09H, DS:DX=字符串首地址, 字符串以 '\$' 结束

功能: 显示字符串, 直到遇到 '\$' 为止。

(6) 入口: AH=0AH, DS:DX=缓冲区首地址, (DS:DX)=缓冲区最大字符数,

(DS:DX+1)=实际输入字符数, (DS:DX+2)=输入字符串起始地址。

功能: 读键盘输入的字符串到 DS:DX 指定缓冲区中并以回车结束。

针对字符串输出, 再特附上两张说明图

图 1: 输入字符串程序

```
DAT1 DB 20, ?, 20 DUP (?)
```

⋮

在数据段
中定义

```
LEA DX, DAT1
```

```
MOV AH, 0AH
```

```
INT 21H
```

图 2: 字符串输出程序显示例

```
DATA SEGMENT
```

```
MESS1 DB 'Input String:', 0DH, 0AH, '$'
```

```
DATA ENDS
```

在代码段, 执行下列指令:

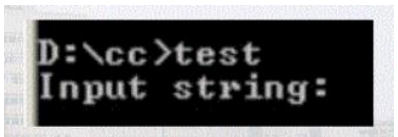
```
LEA DX, MESS1
```

```
MOV AH, 09H
```

```
INT 21H
```

便会在屏幕上窗口输出如图

考虑输出格式需要, 一般会在定义字符串时, 在后面加上回车和换行符 (这里不解



```
D:\cc>test
Input string:
```

释为什么回车和换行是两种东西)，最后加上'\$'代表字符串的结束。

第四节 汇编语言程序设计

结合习题自行理解。