汇编语言程序设计

Assembly Language Programming

主讲：徐娟

计算机与信息学院　计算机系 分布式控制研究所

E-mail: xujuan@hfut.edu.cn,

Mobile: 18055100485

在掌握基本的汇编语言程序设计方法之后，进一步学习如何提高编程效率的各种实用方法：

- ✓高级语言特性
- ✓宏结构
- ✓模块化

5.1 高级语言特性

MASM 6.0引入高级语言的程序设计特性

- 条件控制伪指令

  .IF .ELSE .ENDIF

- 循环控制伪指令

  .WHILE .ENDW .REPEAT .UNTIL

- 过程声明和过程调用伪指令

  .PROTO .INVOKE

5.2 宏结构程序设计

宏汇编

重复汇编

条件汇编

——统称宏结构

宏（Macro）是汇编语言的一个特点，它是与子程序类似又独具特色的另一种简化源程序的方法

宏汇编

宏——具有宏名的一段汇编语句序列

——宏定义时书写

宏指令——这段汇编语句序列的缩写，一次书写，多次调用

——宏调用时书写

宏展开——宏指令处用这段宏代替的过程

——宏汇编时实现

宏的参数功能强大，颇具特色

配合宏，还有宏操作符和有关伪指令

宏定义

- 宏定义
  参数可以是任意内容
  形式参数表：可以有多个，用“，”分开
  同样注意要保护和恢复现场

```
宏名    macro［形参表］
        宏定义体
        endm
```

```
mainbegin    MACRO       ;;定义名为mainbegin的宏，无参数
             mov ax,@data            ;;宏定义体
             mov ds,ax
             ENDM                    ;;宏定义结束

mainend      MACRO retnum            ;;带有形参retnum
             mov al,retnum           ;;宏定义中使用参数
             mov ah,4ch
             int 21h
             ENDM
```

宏注释符

宏调用

宏名　［实参表］

```
start:    mainbegin                ;宏调用，建立DS内容
          dispmsg string           ;宏调用，显示字符串
          mainend 0                ;宏调用，返回DOS
          end start
```

- 实参表中的实参与形参表中的形参在位置上一一对应
  - 若实参数>形参数，则多余的实参无效;
  - 若实参数<形参数，则多余的形参作“空(NUL)”处理;
- 对宏指令必须先定义后调用。宏定义通常放在源程序的开头。
  宏调用的实质是在汇编过程中进行宏展开
- 宏展开的具体过程是：当汇编程序扫描源程序遇到已有定义的宏调用时，即用相应的宏定义体取代源程序的宏指令，同时用位置匹配的实参对形参进行取代

宏展开

宏展开——在汇编时，用宏定义体的代码序列替代宏指令的过程。

- 展开后必须语法正确

```
start:      mainbegin           ;宏指令
 1          mov ax,@data        ;宏展开
 1          mov ds,ax

            mainend 0           ;宏指令
 1          mov al,0            ;宏展开
 1          mov ah,4ch
 1          int 21h
```

宏的参数

宏的参数使用非常灵活

宏定义时，

- 可以无参数，例如mainbegin；可以带有一个参数，例如mainend；也可以具有多个参数；
- 参数可以是常数、变量、存储单元、指令（操作码）或它们的一部分，也可以是表达式；
- 宏定义体可以是任何合法的汇编语句，既可以是硬指令序列，又可以是伪指令序列；

宏的参数

例5. 5a

```
; 宏定义
shlext      macro shloprand,shlnum
            push cx
            mov cl,shlnum
            shl shloprand,cl
            pop cx
            endm
; 宏指令
            shlext ax,6
; 宏展开
 1          push cx
 1          mov cl,06
 1          shl ax,cl
 1          pop cx
```

宏操作符

例5. 6

```
；宏定义
dstring     macro string
            db ’&string&’,0dh,0ah,’$’
            endm
；宏调用
dstring     < This is a example. >
dstring     < 0 !< Number !< 10 >
；宏展开
 1          db ’This is a example.’, 0dh,0ah,’$’
 1          db ’0 < Number < 10’, 0dh,0ah, ’$’
```

传递注释符

转义注释符

宏操作符

- 替换操作符&　将参数与其他字符分开

```
MYCAL MACRO HOW,DATA,TIMES;
    MOV CL, TIMES
    SH&HOW DATA,CL
ENDM
```

```
MYCAL L,DL,3
```

⇩

```
    MOV CL, 3
    SHL DL,CL
```

```
MYCAL R,AL,2
```

⇩

```
MOV CL,2
SHR AL,CL
```

一条宏指令实现左移和右移

宏操作符

- 定界符< >，　括起字符串

```
BUF MACRO DATA
    DB   DATA
ENDM
```

```
BUF 1,2,3
```

↓

```
DB 1
```

```
BUF <1,2,3>
```

↓

```
DB 1,2,3
```

宏操作符

- %——强迫后面表达式先计算

```
MYCAL MACRO N1,N2;
      MOV AL,N1*10
      MOV N2,AL
ENDM
```

```
MYCAL 2-1,DL
```

```
MOV AL,2-1*10
MOV DL,AL
```

```
MYCAL %2-1,DL
```

```
MOV AL,1*10
MOV DL,AL
```

宏操作符

- ;;——宏注释符，用于表示在宏定义中的注释。采用这个符号的注释，在宏展开时不出现
- &——替换操作符，用于将参数与其他字符分开。如果参数紧接在其他字符之前或之后，或者参数出现在带引号的字符串中，就必须使用该伪操作符
- < >——字符串传递操作符，用于括起字符串。在宏调用中，如果传递的字符串实参数含有逗号、空格等间隔符号，则必须用这对操作符，以保证字符串的完整
- !——转义操作符，用于指示其后的一个字符作为一般字符，不含特殊意义
- %——表达式操作符，用在宏调用中，表示将后跟的一个表达式的值作为实参，而不是将表达式本身作为参数

宏伪指令

- 局部标号伪指令

  LOCAL　　标号列表

  - 宏定义体采用了标号，应使用LOCAL加以说明
  - 它必须是宏定义MACRO语句之后的第一条语句
  - 由LOCAL定义的标号由??0000~ FFFF替代（??字符开头）

- 宏定义删除伪指令

  PURGE 宏名表

  不需要某个宏定义时，可以把它删除

- 宏定义退出伪指令

  EXITM

  伪指令EXITM表示结束当前宏调用的展开

宏伪指令

例5.7

```
; 宏定义
absol   macro oprd
        local next
        cmp   oprd,0
        jge next
        neg   oprd
next:
        endm
```

单独占一行

```
; 宏调用
        absol word ptr [bx]
        absol bx
; 宏展开
 1      cmp word ptr [bx],0
 1      jge ??0000
 1      neg word ptr [bx]
 1      ??0000:
 1      cmp bx,0
 1      jge ??0001
 1      neg bx
 1      ??0001:
```

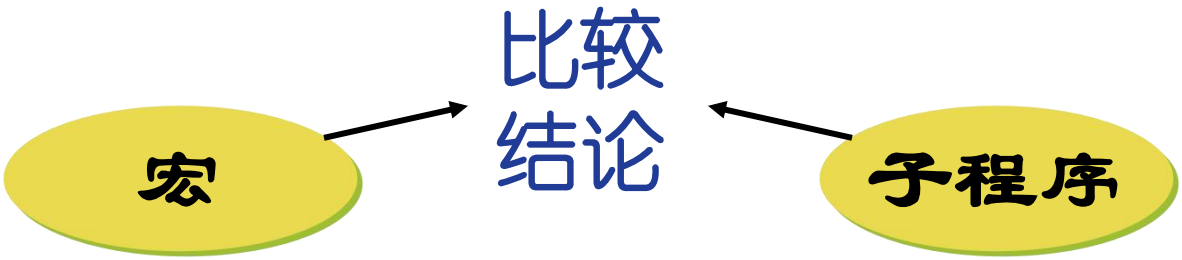
比较

宏

- 仅是源程序级的简化：宏调用在汇编时进行程序语句的展开，不需要返回；不减小目标程序，执行速度没有改变
- 通过形参、实参结合实现参数传递，简捷直观、灵活多变

子程序

- 还是目标程序级的简化：子程序调用在执行时由CALL指令转向、RET指令返回；形成的目标代码较短，执行速度减慢
- 需要利用寄存器、存储单元或堆栈等传递参数



- 宏与子程序具有各自的特点，程序员应该根据具体问题选择使用那种方法

- 通常，当程序段较短或要求较快执行时，应选用宏；当程序段较长或为减小目标代码时，要选用子程序

5.2.2 重复汇编

- 重复汇编指在汇编过程中，重复展开一段（基本）相同的语句
- 重复汇编没有名字，不能被调用
- 重复汇编常用在宏定义体中，也可以在一般汇编语句中使用
- 重复汇编伪指令有三个：

  REPEAT——按参数值重复

  FOR——按参数个数重复

  FORC——按参数的字符个数重复
- 最后，用ENDM结束

重复汇编

按参数值重复

```
REPEAT    重复次数
      重复体
ENDM
```

```
X = 0
REPEAT  5
      X=X+1
      DB X
ENDM
```

```
DB 1
DB 2
DB 3
DB 4
DB 5
```

重复块

按参数个数重复

```
FOR    形参,〈实参表〉
       重复体
ENDM
```

```
FOR  X,<1,2,3,4,5>
     DB X
ENDM
```

```
DB 1
DB 2
DB 3
DB 4
DB 5
```

重复块

按参数字符个数重复

```
FORC    形参，字符串
        重复体
ENDM
```

```
FORC  X,<12345>
    DB X
ENDM
```

```
DB 1
DB 2
DB 3
DB 4
DB 5
```

条件编译

- 条件汇编——不符条件，不会编译
- 格式：

```
IFxx   表达式
        语句序列1
[ ELSE
        语句序列2 ]
ENDIF
```

- IF 表达式 == #if
- IFDEF 符号 == #ifdef
- IFNDEF 符号 == #ifndef
- ELSE == #else
- ENDIF == #endif

条件编译

例5.10

```
pdata    macro num
   IF num lt 100          ;;如果num < 100，则汇编如下语句
        db num dup (?)
   ELSE                   ;;否则，汇编如下语句
        db 100 dup (?)
   ENDIF
        endm
```

```
        pdata 12          ;宏调用①
        db 12 dup(?)      ;宏汇编结果①
```

```
        pdata 102         ;宏调用②
        db 100 dup(?)     ;宏汇编结果②
```

宏结构的作用

宏汇编、重复汇编和条件汇编

为源程序的编写提供了很多方便，

灵活运用它们可以编写出非常良好的源程序

汇编系统中有些以圆点起始的

伪指令（如.startup、.exit等）

实际上是一种宏结构

5.3 多模块（模块化）程序设计

- **多个模块：多个源文件**
  - 开发并行
  - 容易维护
- **方式：**
  - 源文件包含
  - 目标文件连接
  - 子程序库调入
- **问题**
  - 模块间组合
  - 模块间通讯

1 源文件包含

- INCLUDE common.asm
  - 将定义或者申明放在一个源文件中，可重复利用。
  - 采用插入相应文件的方式完成合并，所以被包含的文件不要有END语句。
- INCLUDE common.inc（包含文件）

5.3.1 源程序文件的包含

把源程序分放在几个文本文件中，在汇编时通过包含伪指令INCLUDE结合成一体

INCLUDE 文件名

- 可将常用的子程序形成.ASM汇编语言源文件
- 可将常用的宏定义存放在.MAC宏库文件中
- 可将常量定义、声明语句组织在.INC包含文件中
- 采用插入相应文件的方式完成合并，所以被包含的文件不要有END语句。

例5.12a

① 宏库文件 lt512a.mac
② 主程序文件 lt512a.asm
③ 子程序文件 sub512a.asm

Lt512a.mac

```
dispchar    macro char              ;显示char字符
            mov dl,char
            mov ah,2
            int 21h
            endm
dispmsg     macro message           ;显示message字符串
            mov dx,offset message
            mov ah,9
            int 21h
            endm
```

Lt512a.asm

```
        include lt512a.mac
        ...
        dispmsg msg1            ;提示输入数据
        mov bx,offset buf
        call input              ;数据输入
        cmp cx,0
        je start4               ;没有输入数据则退出
        mov count,cx
        ...                     ;显示输入的数据
        ...                     ;数据排序
        ...                     ;显示经排序后的数据
start4: .exit 0
        include sub512a.asm
        end
```

2 目标文件的连接

- 独立编译为obj文件
- 连接程序将所有目标文件连接起来，最终产生一个可执行文件
  - masm module1.asm; masm module2.asm …
  - link module1.obj + module2.obj

  需要遵循的原则：

  ① 声明共用的变量、过程等

  ② 设置好段属性，实现正确的段组合

  ③ 处理好参数传递问题

声明共用的变量、过程

- 各个模块间共用的变量、过程等要说明

  PUBLIC 标识符 [,标识符...]

  ;定义标识符的模块使用

  EXTERN 标识符:类型 [,标识符:类型...]

  ;调用标识符的模块使用

- 标识符是变量名、过程名等
- 类型是byte/word/dword（变量）或near/far（过程）
- 在一个源程序中，public/extern语句可以有多条
- 各模块间的public/extern伪指令要互相配对，并且指明的类型互相一致

完整段定义

```
段名    segment     定位    组合    段字    '类别'
        ...                 ;语句序列
段名    ends
```

- 完整段定义由SEGMENT和ENDS这一对伪指令实现，
  SEGMENT伪指令定义一个逻辑段的开始，
  ENDS伪指令表示一个段的结束
- 如果不指定，则采用默认参数；但如果指定，注意要按照上列次序
- 段名对外表现为立即数，为该段的段地址

段定位（align）属性

- 指定逻辑段在主存储器中的边界，可为：

  BYTE 段开始为下一个可用的字节地址（xxxx xxxxb）

  WORD 段开始为下一个可用的偶数地址（xxxx xxx0b）

  DWORD 段开始为下一个可用的4倍数地址（xxxxxx00b）

  PARA 段开始为下一个可用的节地址（xxxx 0000b）

  PAGE 段开始为下一个可用的页地址（0000 0000b）

- 简化段定义伪指令的代码和数据段默认采用WORD定位，堆栈段默认采用PARA定位

- 完整段定义伪指令的默认定位属性是PARA，其低4位已经是0，所以默认情况下数据段的偏移地址从0开始

段组合（combine）属性

```
段名   segment    定位   组合   段字   '类别'
       ...              ;语句序列
段名   ends
```

- 指定多个逻辑段之间的关系，可为：

PRIVATE　本段与其他段没有逻辑关系，不与其他段合并，每段都有自己的段地址。这是完整段定义伪指令默认的段组合方式

PUBLIC　连接程序把本段与所有同名同类型的其他段相邻地连接在一起，然后为所有这些段指定一个共同的段地址，也就是合成一个物理段。这是简化段定义伪指令默认的段组合

STACK　本段是堆栈的一部分，连接程序将所有STACK段按照与PUBLIC段的同样方式进行合并。这是堆栈段必须具有的段组合

COMMON　把所有同名同类型的段指定一个段地址，后面覆盖前面。

段字（use）属性

```
段名   segment     定位   组合   段字   '类别'
       ...              ;语句序列
段名   ends
```

- 为支持32位段而设置的属性
- 对于16位x86 CPU来说，它默认是16位段，即USE16

而对于汇编32位x86 CPU指令时，它默认采用32位段，即USE32；但可以使用USE16指定标准的16位段

- 编写运行于实地址方式（8086工作方式）的汇编语言程序，必须采用16位段

段类别（class）属性

```
段名   segment     定位   组合   段字   '类别'
       ...         ;语句序列
段名   ends
```

- 当连接程序组织段时，将所有的同类别段相邻分配
- 段类别可以是任意名称，但必须位于单引号中
- 大多数MASM程序使用 'code'、'data'和'stack'

来分别指名代码段、数据段和堆栈段，以保持所有代码和数据的连续

模块之间的组合

- **一个模块中：段名和属性相同的两个段按先后顺序进行合并**
- **模块间：同名且类型相同的PUBLIC（或STACK）段组合成一个段**
- **组合规则：**
  - 对齐属性最好用PARA
  - 数据段可以适当组合
  - 堆栈段最好组合
  - 代码段不宜组合，通过过程调用方式进行交互

Examples

Sseg segment stack ‘stack’ … Sseg ends Dseg segment public ‘data’ … Dseg ends Cseg segment public ‘code’ … Cseg ends end	Dseg segment private ‘data’ … Dseg ends Cseg2 segment public ‘code’ … Cseg2 ends	Sseg segment stack ‘stack’ … Sseg ends Dseg segment public ‘data’ … Dseg ends Cseg3 segment public ‘code’ … Cseg3 ends

- Sseg ： 由模块1、3的Sseg组合而成，类别是stack
- Dseg ： 由模块1、3的Dseg组合而成
- Dseg ： 模块2的私有段
- 三个独立的Cseg

2 目标文件的连接

- 独立编译为obj文件
- 连接程序将所有目标文件连接起来，最终产生一个可执行文件
  - masm module1.asm; masm module2.asm …
  - link module1.obj + module2.obj

  需要遵循的原则：

  ① 声明共用的变量、过程等

  ② 设置好段属性，实现正确的段组合

  ③ 处理好参数传递问题

处理好参数传递问题

- 少量参数可用寄存器或堆栈直接传送数据本身
- 大量数据可以安排在缓冲区，用寄存器或堆栈传送数据的存储地址
- 还可利用变量传递参数，但是要采用public/extern声明为公共（全局）变量
- 这些也是子程序间的参数传递方法
- 第7章混合编程介绍了更好的堆栈传递参数方法，可以采用

模块之间的通讯

- **PUBLIC：本模块定义的，外模块想访问的变量，标号，过程，符号等。**
  - PUBLIC 公共符号1，2，3……..
- **EXTRN/EXTERN： 本模块引用的，外模块中定义的变量，标号，过程，符号等。**
  - EXTRN 外部符号1：属性1，外部符号2：属性2……

模块之间的通讯

```
EXTRN display:far
PUBLIC STRING

DATA SEGMENT
  STRING DB 'HELLO WORLD!',0DH,0AH,'$'
DATA ENDS
Code segment
        assume cs:code, ds:data
start:
        MOV AX,DATA
        MOV DS,AX
        call display
        mov ah,4ch
        int 21h
code ends
        end start
```

```
PUBLIC display
EXTRN STRING:BYTE

Code segment
    assume cs:code
display proc far
        mov ah,9
        mov DX, OFFSET STRING
        int 21h
        ret
display endp

code ends
end
```

3 子程序库的调入

- 把常用子程序写成独立的源文件，单独汇编形成OBJ文件后，存入子程序库。
- 子程序库文件（.LIB）是子程序模块的集合，其中存放着各子程序的名称、目标代码及有关定位信息
- 利用库管理工具程序（LIB.exe），把子程序目标模块加入到库中：

LIB 库文件名 + 子程序目标文件名

3 子程序库的调入

例5.12c

① 主程序文件 lt512c.asm
② 子程序文件 sub512c1.asm
③ 子程序文件 sub512c2.asm
④ 子程序文件 sub512c3.asm

Lt512c.asm

```
...                                ;宏定义
.code
extern ALdisp:near,sorting:near,input:near
                                   ;声明其他模块中的子程序
.startup
...
.exit 0
end
```

3 子程序库的调入

sub512c1.asm

```
        .model small
        .code
        public aldisp
Aldisp  proc
        ...
Aldisp  endp
        end
```

sub512c2.asm

```
        .model small
        .code
        public sorting
sorting proc
        ...
sorting endp
        end
```

sub512c3.asm

```
.model small
.code
public input
Input  proc
    ...
Input  endp
       end
```

3 子程序库的调入

① 主程序文件 lt512c.asm
② 子程序文件 sub512c1.asm
③ 子程序文件 sub512c2.asm
④ 子程序文件 sub512c3.asm

子程序库调入：

- 分别汇编得lt512c.obj, sub512c1.obj, sub512c2.obj, sub512c3.obj
- 利用lib.exe把三个子程序组合到子程序库sub512c.lib

  lib sub512c.lib+sub512c1.obj+sub512c2.obj+sub512c3.obj

- Link程序形成lt512c.exe

3 子程序库的调入

- 主程序也单独汇编形成OBJ文件;
- 主程序连接时，调入子程序库中的子程序模块，产生最终的可执行文件:
  - Libraries[.lib]:XXX.lib
- 或使用包含includelib XXX.lib