

汇编语言源程序

主要内容

1

汇编程序与汇编语言源程序

2

汇编语言语句类型与格式

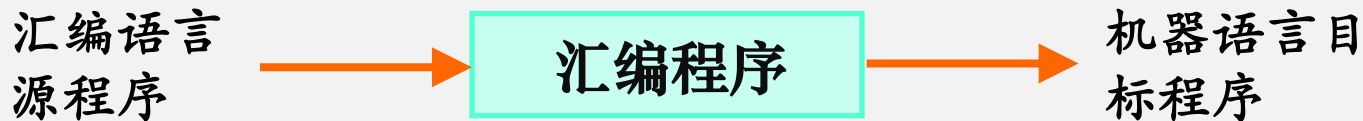
3

汇编语言语句中的操作数

1. 汇编语言源程序与汇编程序

■ 汇编语言源程序 —————> 用助记符编写

■ 汇编程序 —————> 源程序的编译程序

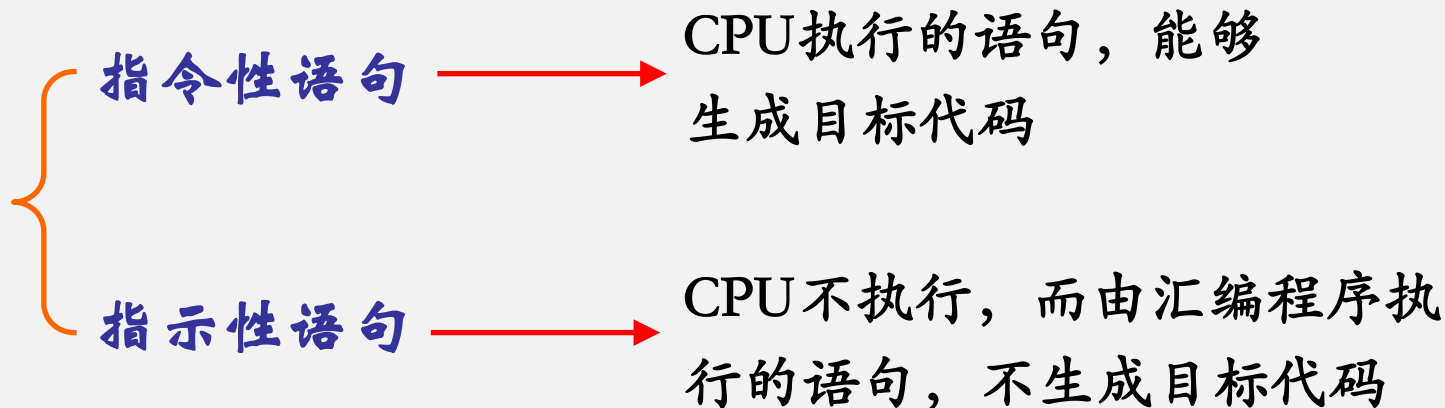


汇编语言程序设计与执行过程

- 输入汇编语言源程序 (EDIT) → 源文件 .ASM
- 汇编 (MASM) → 目标文件 .OBJ
- 链接 (LINK) → 可执行文件.EXE
- 调试 (TD) → 最终程序

2. 汇编语言语句语句类型和格式

汇编语言语句类型：



汇编语言语句类型和格式

汇编语言语句格式：

指令性语句：

[标号:] [前缀] 助记符 [操作数], [操作数] [; 注释]



指令的符号地址

标号后要有冒号



操作码


汇编语言语句类型和格式

指示性语句格式：

[名字] 伪指令助记符 操作数[, 操作数, …][; 注释]



变量的符号地址
其后不加冒号



指示性语句中至少有一个操作数

3. 汇编语言语句中的操作数



寄存器

存储器单元

常量

变量或标号

表达式

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：

- MOV AL, 'A'

- 例：

- 定义字符串：'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，程序中视为存储器操作数
- 变量的属性：

段 值 —— 变量所在段的段地址

偏移量 —— 变量单元地址与段首地址之间的位移量

类 型 —— 字节型、字型和双字型

表达式



算术运算

逻辑运算

关系运算

*取值运算和属性运算

其它运算

取值运算符

■ 用于分析存储器操作数的属性

■ 获取变量的属性值

OFFSET → 取得其后变量或标号的偏移地址
SEG → 取得其后变量或标号的段地址

```
MOV AX, SEG DATA  
MOV DS, AX  
MOV BX, OFFSET DATA
```

取变量的段地址并送给DS

取变量的偏移地址送给BX

等价于 **LEA BX, DATA**

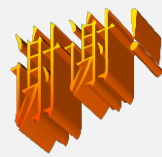
属性运算符

- 用于指定其后存储器操作数的类型
- 运算符：**PTR**
- 例：

■ **MOV BYTR** **PTR**[BX], 12H



指定存储器操作数
[BX]为“**字节型**”



数据定义伪指令

伪指令

- 由汇编程序执行的“指令系统”

- 作用：

- 定义变量；
- 分配存储区
- 定义逻辑段；
- 指示程序开始和结束；
- 定义过程等。

帮助计算机理
解助记符指令
编写的汇编语
言源程序

常用伪指令



数据定义伪指令

符号定义伪指令

段定义伪指令

结束伪指令

过程定义伪指令

宏命令伪指令

一、数据定义伪指令

- 用于定义数据区中变量的类型及其所占内存空间大小
- 格式:

[变量名] 伪指令助记符 操作数, ... ; [注释]

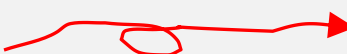

符号地址

定义变量类型

变量值, 可以是常数, 表达式或字符串。其大小不能超过伪指令助记符所限定的范围

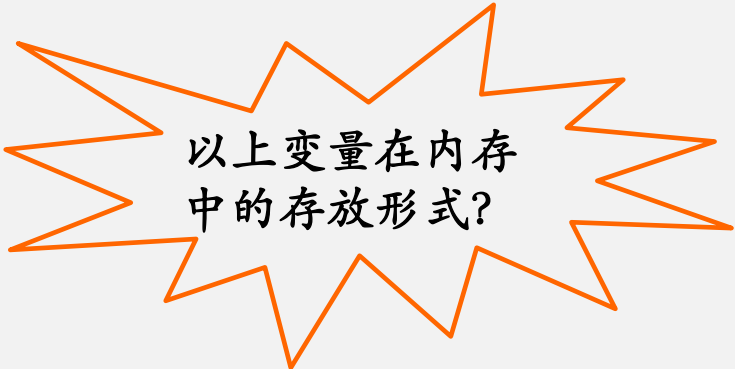
变量的类型及其操作数的个数决定了该变量所在内存空间的大小

1. 数据定义伪指令助记符

- DB (Define Byte) :  指向的每一个操作数占1个字节单元
 - 定义的变量为字节型
- DW (Define Word) :  指向的每一个操作数占1个字单元
 - 定义的变量为字类型
- DD (Define Double Word) :
 - 定义的变量为双字型
- DQ (Define Quadword) :
 - 定义的变量为4字型
- DT (Define Tenbytes) :
 - 定义的变量为10字节型

数据定义伪指令例

- **DATA1 DB 11H, 22H, 33H, 44H**
- **DATA2 DW 11H, 22H, 3344H**
- **DATA3 DD 11H*2, 22H, 33445566H**



以上变量在内存
中的存放形式?

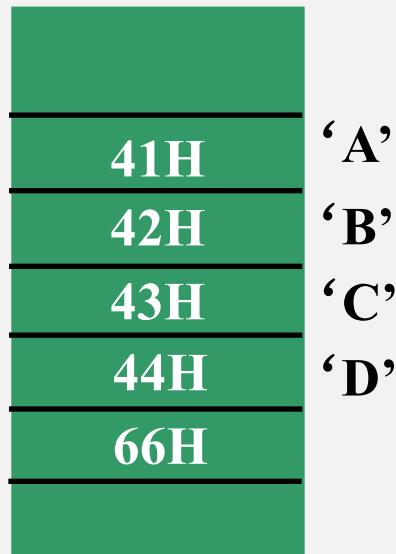
数据定义伪指令例_____变量在内存中的分布

DATA1	
	11
	22
	33
DATA2	44
	11
	00
	22
	00
	44
	33

DATA3	22
	0
	0
	0
	22
	0
	0
	0
	66
	55
	44
	33

数据定义伪指令的几点说明

- 数据定义伪指令决定所定义变量的类型；
- 定义字符串必须用DB伪指令
- 例：
 - DATA1 DB 'ABCD', 66H



2. 重复操作符

- 当同样的操作数重复多次时，可以使用重复操作符。

- 作用：

- 为一个数据区的各单元设置相同的初值

- 格式：

[变量名] 伪指令助记符 **n** DUP (初值 [,初值,...])

- 例：

- M1 DB 10 DUP (0)

重复次数

重复的内容

常用于声明一个数据区

3. “？”的作用

- 表示随机值，用于预留存储空间

- 例：

- MEM1 DB 34H, 'A', ?

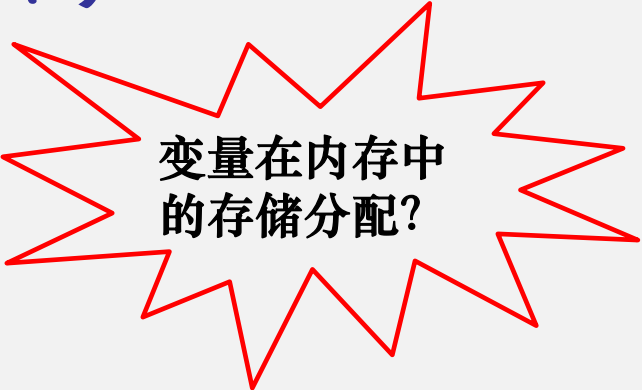
随机数
占1个字节单元

- DW 20 DUP (?)

预留40个字节单元，每单元为随机值

数据定义伪指令例

- M1 DB 'How are you?'
- M2 DW 3 DUP(11H), 3344H
- DB 4 DUP (?)
- M3 DB 3 DUP (22H, 11H, ?)



变量在内存中的存储分配?

数据定义伪指令例

M1

48H	'H'
6FH	'o'
77H	'w'
20H	' '
61H	'a'
72H	'r'
65H	'e'
20H	' '
79H	'y'
6FH	'o'
75H	'u'
3FH	'?'

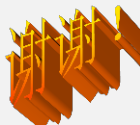
M2

11H
00H
11H
00H
11H
00H
44H
33H
XX
XX
XX
XX

M3

22H
11H
XX
22H
11H
XX
22H
11H
XX

随机数



调整偏移量伪指令

- 规定程序或变量在逻辑段中的起始地址

- 格式:

- ORG 表达式

计算值为
非负常数

- 例:

DATA SEGMENT

ORG 1200H

BUFF DB 1,2

DATA ENDS

变量BUFF的偏
移地址=1200H

符号与段定义相关伪指令

一、符号定义伪指令

- 将表达式的值赋给一个名字。当源程序中需多次引用某一表达式时，可以利用EQU伪指令，用一个符号代替表达式，以便于程序维护。

- 格式：

- 符号名 EQU 表达式

EQU说明的表达式不占用内存空间

- 操作：

- 用符号名取代后边的表达式，不可重新定义

- 例：

- `CONSTANT EQU 100`

二、段定义伪指令

■ 在汇编语言源程序中定义逻辑段

- 说明逻辑段的起始和结束
- 说明不同程序模块中同类逻辑段之间的联系形态

■ 格式:

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

逻辑段的
段基地址

说明逻辑段
的起点

装入内存时各逻辑段的组合方式

链接时不同程序模块中的同类逻辑段将被装入连续存储区

段定义伪指令例

DATA SEGMENT

MEM1 DB 11H, 22H

MEM2 DB 'Hello! '

MEM3 DW 2 DUP (?)

DATA ENDS

变量在逻辑段中的位置就代表了它的偏移地址

段名也代表逻辑段的段地址

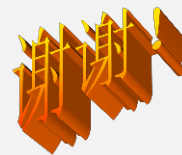
表示变量的类型

三、设定段寄存器伪指令

- 说明所定义逻辑段的性质
- 格式：
 - ASSUME 段寄存器名:段名[, 段寄存器名:段名, ...]

四、结束伪指令

- 表示源程序结束
- 格式：
 - END [标号]



其它伪指令

常用伪指令



数据定义伪指令

符号定义伪指令

段定义伪指令

结束伪指令

过程定义伪指令

宏命令伪指令

调整偏移量伪指令

过程定义伪指令

- 用于定义一个过程体
- 格式:

过程名 PROC [NEAR / FAR]

⋮

RET

过程名 ENDP

过程入口的
符号地址

过程伪指令的最后一条指令必须是RET

过程定义及调用例

■ 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

■ 调用延时子程序:

CALL DELAY

宏命令伪指令

■ 宏：

- 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

■ 当源程序中需要多次使用同一个程序段时，可以将该程序段定义为一个宏

■ 格式：

宏命令名 MACRO <形式参数>

⋮

ENDM

宏体

通过参数传递
方式引用宏

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

调整偏移量伪指令

- 规定程序或变量在逻辑段中的起始地址

默认情况下，程序或变量在逻辑段中的起始偏移地址为：0

利用ORG指令，可以改变程序或变量的起始偏移地址

调整偏移量伪指令

- 规定程序或变量在逻辑段中的起始地址

- 格式:

- ORG 表达式

计算值为
非负常数

- 例:

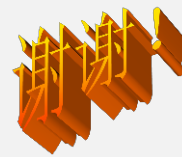
DATA SEGMENT

ORG 1200H

BUFF DB 1,2

DATA ENDS

变量BUFF的偏
移地址=1200H



系统功能调用

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**

DOS软中断

**DOS中断包括：设备管理，目录管理，文件管理，其它
用中断类型码区分**

**DOS软中断：
类型码为
21H**

DOS软中断

- 关于DOS软中断说明：
 - 包含多个子功能的功能包；
 - 各子功能用功能号区分；
 - 用软中断指令调用，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。
- 调用格式：

MOV AH, 功能号

<置相应参数>

INT 21H



入口参数/出口参数

1. 单字符输入

- 调用方法：

MOV AH, 01

INT 21H

- 输入的字符在AL中

单字符输入例

```
GET_KEY:  MOV  AH, 1
           INT  21H
           CMP  AL, 'Y'
           JZ   YES
           CMP  AL, 'N'
           JZ   NO
           JMP  GET_KEY

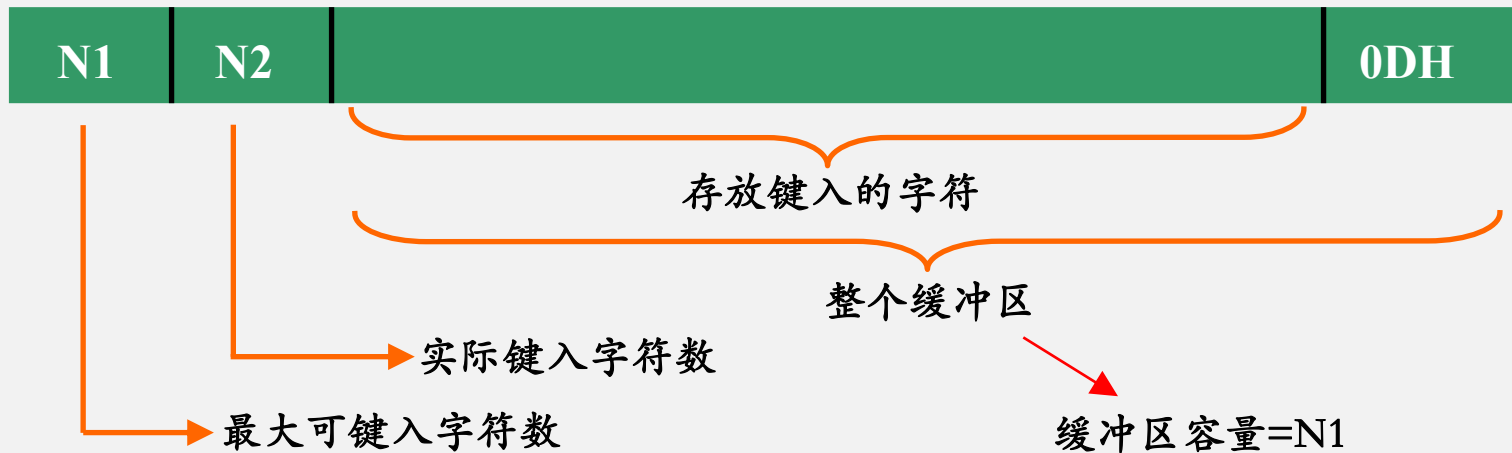
YES:      :
NO:       :
```



交互式应
答程序

2. 字符串输入

- 接收由键盘输入一串字符
- 输入的字符串存储在内存指导区域中 → 字符输入缓冲区
- 用户自定义缓冲区格式:



字符串输入

- 字符串输入功能号：
 - 10
- 缓存区须定义在数据段
- 方法：

AH ← 功能号

DS:DX ← 字符串在内存中的存放地址

INT 21H

输入字符串程序段

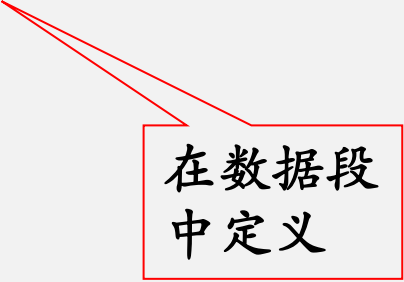
DAT1 DB 20, ? , 20 DUP (?)

⋮

LEA DX, DAT1

MOV AH, 0AH

INT 21H

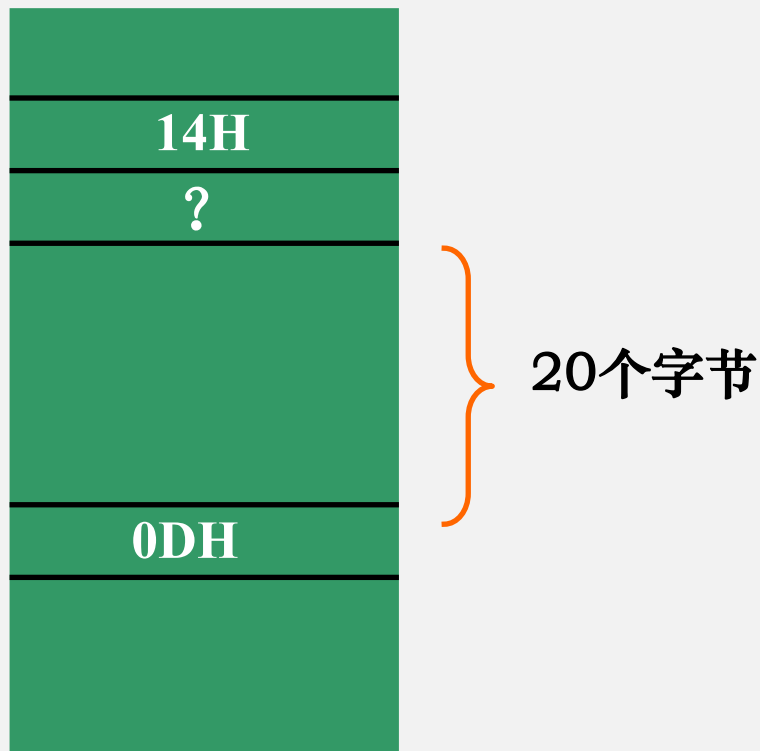


在数据段
中定义

输入缓冲区

定义后的输入缓冲区初始状态:

存储输入的字符



3. 单字符显示输出

AH ← 功能号2

DL ← 待输出字符

INT 21H

例：

MOV AH, 2

MOV DL, 41H

INT 21H

执行结果：
屏幕显示A

4. 字符串输出显示

AH ← 功能号**09H**

DS: DX ← 待输出字符串的偏移地址

INT 21H

字符串输出显示

■ 注意点：

- 被显示的字符串必须以 '\$' 结束；
- 所显示的内容不应出现非可见的ASCII码；
- 若考虑输出格式需要，在定义字符串后，加上回车符和换行符。

字符串输出显示例

DATA SEGMENT

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

DATA ENDS

CODE SEGMENT

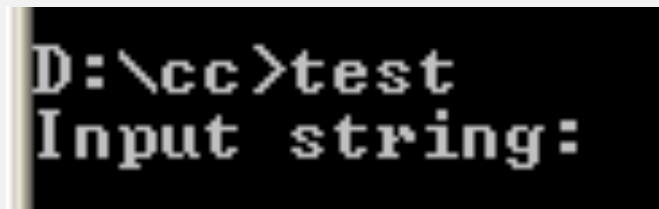
⋮

MOV AH, 09

MOV DX, OFFSET MESS1

INT 21H

⋮



D:\cc>test
Input string:

5. 返回操作系统 (DOS) 功能

- 功能号:

- 4CH

- 调用格式:

- MOV AH, 4CH
- INT 21H

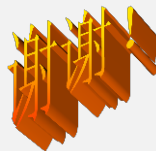
- 功能:

- 程序执行完该2条语句后能正常返回OS
- 常位于程序结尾处。

DOS功能调用小结

- 通过中断指令调用。1个中断类型码对应1个功能程序包；
- 每个程序包中的子功能通过功能号区分，调用时功能号须送AH；
- 注意不同子功能的入口/出口参数要求；

DOS和BIOS中断均可能影响AX



汇编程序设计示例

示例1：

变量在内存中的存储
(16进制表示)

以下数据区在内存中是如何存储的？

DATA SEGMENT

NAMES DB 'TOM..', 20

DB 'CATE', 25

DATA ENDS

每个字符在内存中以ASCII码表示，占用1B。

10进制数

NAMES

54	'T'
4F	'O'
4D	'M'
2E	','
2E	','
14	
43	'C'
41	'A'
54	'T'
45	'E'
19	

示例2：

阅读程序段，说明该程序段的功能

在屏幕上依次
输出123ABC

```
DATA SEGMENT
    A DB '123ABC'
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
        MOV DS, AX
        LEA BX, A
        MOV CX, 6
LP:     MOV AH, 2
        MOV AL, [BX]
        XCHG AL, DL
        INC BX
        INT 21H
        LOOP LP
        MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

示例3：

DATA SEGMENT

TABLE DW 3400H, 5600H, 2300H, 4500H, 2300H, 1200H, 2344H, 3500H

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA

MOV DS, AX

MOV BX, OFFSET TABLE

MOV SI, 06H

A: JMP DWORD PTR[BX+SI+2]

.....

转移目标地址存放于内存数据
段间偏移地址=BX+SI+2 起始
的4个单元中

程序执行完标号为A的指令后: CS= (**1200H**) IP= (**2300H**)

示例4:

当程序执行到STOP时:

SI= (9)

DI= (0015H)

CX= (3)

FLAG= (0)

ZF= (0)

DATA SEGMENT

STR1 DB 'HELLO WORLD! '

STR2 DB 'HELLO WOOLD! '

COUNT DW 12

FLAG DB ?

DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA, ES: DATA

START: MOV AX, DATA

MOV DS, AX

MOV ES, AX

LEA BX, FLAG

LEA SI, STR1

LEA DI, STR2

MOV CX, COUNT

CLD

REPE CMPSB

JZ NEXT1


MOV Byte PTR[BX], 00H

JMP STOP

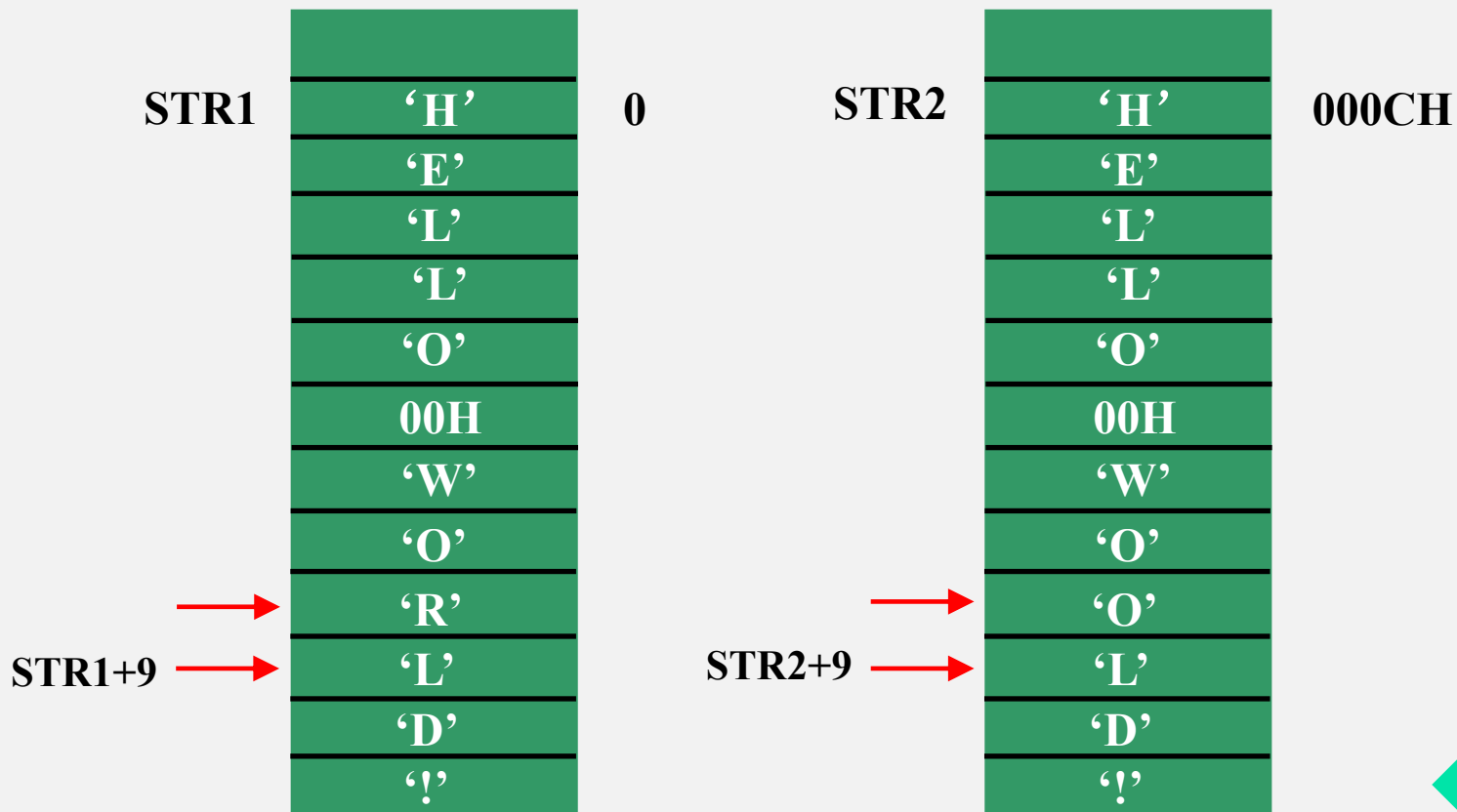
NEXT1: MOV Byte PTR[BX], 0FFH

STOP:

两字符串不相
等，结束比较
时ZF=0



实例4:



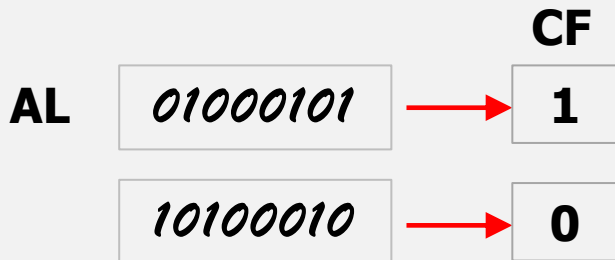
示例5：

如果从380H端口输入的是
45H，则程序执行完后：

BX= (8)

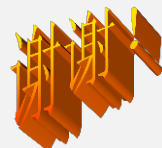
AL= (45H)

SUM0—SUM7的内容为：
(FF,0,FF,0,0,0,FF,0)



```
DATA SEGMENT
    SUM DB 8 DUP (0)
DATA ENDS
CODE SEGMENT
    ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
        MOV DS, AX
        LEA BX, SUM
        MOV DX, 380H
        IN AL, DX
        MOV CX, 8
NEXT:  ROR AL, 1
        JNC NEXT1
        MOV Byte PTR[BX], 0FFH
        JMP NEXT2
NEXT1: MOV Byte PTR[BX], 0
NEXT2: INC BX
        LOOP NEXT
        .....
```

不含CF循环右移1位

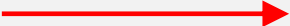


汇编程序设计小结

汇编语言程序设计一般步骤

- 根据实际问题抽象出数学模型
- 确定算法
- 画程序流程图
- 分配内存工作单元和寄存器
- 程序编码
- 调试

内容提要

- 汇编语言源程序的结构
- 汇编语言语句  变量
表达式中的运算符
- 伪指令
- DOS功能调用
 - 5个子功能的应用
- 简单汇编语言源程序的设计

注意点

- 变量的定义与应用
 - 明确所定义变量在内存中的分布
- 存储区的定义
 - 不能定义没有变量的存储区
- 完整的汇编语言源程序结构
 - 定义逻辑段，说明段的含义，初始化段寄存器
- 伪指令
 - 数据定义方式
- 字符及字符串的输入和显示输出
 - 字符输入缓冲区的定义，输出字符串的定义

注意点

■ 汇编语言程序结构：

■ 顺序结构

■ 循环结构

- 使用LOOP指令或条件转移指令实现

■ 分支结构


- 一般用条件转移指令实现

■ 子程序结构


- 子程序的定义和调用方式

例：


- ◆ 定义20B的字符输入缓冲区BUFF以及20B的字节变量DATA;
- ◆ 显示字符串输入提示信息，然后从键盘接收字符 ‘Hello, my friends!’；
- ◆ 将BUFFER中的字符串按从左到右的方向传送到DATA中。



字符串显示输出



字符串输入



串操作指令

DSEG SEGMENT

MESS DB 'Please input string:', 0DH, 0AH, '\$'

BUFF DB 20, ? , 20 DUP(?)

DATA DB 20 Dup(?)

DSEG ENDS

CSEG SEGMENT

ASSUME CS: CSEG, DS: DSEG, ES: ESEG

START: MOV AX, DSEG

MOV DS, AX

MOV ES, AX

AGAIN: LEA DX, MESS

MOV AH, 9

INT 21H

LEA DX, BUFF

MOV AH, 10

INT 21H

XOR CX, CX

MOV CL, BUFF+1

LEA DI, DATA

LEA SI, BUFF+2

CLD

REP MOVSB

MOV AH, 4CH

INT 21H

CSEG ENDS

END SATRT

