第六章　子程序设计

学习目的

- 掌握汇编语言中子程序的基本结构、掌握子程序与主程序之间的参数传递方法。
- 能够正确的在程序设计中使用子程序。

6.1.1 子程序的定义

- PROCNAME PROC [NEAR/FAR]
- ...
- RET
- ...
- PROCNAME ENDP

6.1.1 子程序的定义

- ＰＲＯＣＮＡＭＥ：子程序名称字段，必选字段，用于标识子程序第一条指令的段内偏移量或逻辑地址。

- 在关键字ＰＲＯＣ和ＥＮＤＰ之间的内容为子程序的主体部分，由指令、伪指令序列构成。

子程序类型

- 子程序有ＮＥＡＲ和ＦＡＲ两种类型
- ＮＥＡＲ类型的子程序名称对应其段内偏移量；ＦＡＲ类型的子程序名称对应其完整逻辑地址.
- 子程序与标号的类型具有相同解释，因为子程序调用是一种特殊的程序流程转移，即带返回功能的流程转移。
- ＮＥＡＲ类型是子程序的隐含类型.

子程序类型

- ＮＥＡＲ类型的子程序只能在其所在的代码段内被调用，即调用指令和子程序本身必须在同一代码段。

- ＦＡＲ类型的子程序可以在任何一个代码段中被调用，即调用指令不必和子程序在同一代码段。

6．1．2　子程序的调用与返回

- （1）子程序调用

- 一般格式：
- ＣＡＬＬ　　子程序名

- 转移指令通常引用标号作为目标地址的符号地址；类似的，ＣＡＬＬ指令引用子程序名作为符号地址。

6．1．2　子程序的调用与返回

- 执行ＣＡＬＬ指令时，ＣＳ：ＩＰ指向它后面一条指令，该地址称为返回地址（返回点），子程序执行完毕以后一定要返回到这个地址继续执行.

- ＣＡＬＬ指令在把程序流程转向目标地址（子程序的首地址）之前，会把返回地址首先压入堆栈保存，为子程序返回主程序提供必要条件.

6．1．2　子程序的调用与返回

- CALL指令功能：保存返回地址到堆栈，把程序流程转移到子程序的入口地址。

- 标志位影响：无

子程序调用方式

- 子程序调用指令和无条件转移指令相似，按照它获取目标地址的方式，可以分为：

- 段内直接调用
- 段内间接调用
- 段间直接调用
- 段间间接调用

1）段内直接调用

- 格式:
- CALL　　PROC_NAME
- CALL　NEAR PTR PROC_NAME

- 此调用方式仅将子程序名称作为一个符号偏移量来引用.

2）段内间接调用

- 段内间接调用仍然仅引用子程序在代码段内的偏移量，但所引用的偏移量存放在某一个１６位寄存器或字内存单元中.

- 格式：（例子）
- ＣＡＬＬ　　ＢＸ

- ＣＡＬＬ　　２０Ｈ［ＢＸ＋ＳＩ］

3）段间直接调用

- 格式:
- CALL　　PROC_NAME
- CALL　FAR PTR PROC_NAME

- 此调用方式将子程序名称作为符号地址引用，与段内直接调用不同，这里引用的是完整逻辑地址。

4）段间间接调用

- 此调用方式引用子程序的完整逻辑地址，并且假定子程序的入口地址保存在指定的双字内存单元中，低地址保存偏移量，高地址保存段基值.

- 格式：（例子）
- CALL DWORD PTR DISP[BX+SI]

- 假定子程序的入口地址保存在一个双字单元中，EA＝DISP＋（BX）＋（SI）.

（2）子程序的返回

- 在任何子程序中，最后一条被执行的指令一定是返回指令。
- 功能：从堆栈中恢复返回地址，把程序流程返回到主程序。
- 标志位影响：子程序返回指令不会影响标志位

1）段内返回指令

- 使用段内调用指令调用的子程序，必须使用段内返回指令才能正确返回主程序。

- 格式1：RET （RETN）

- 功能：从堆栈中出栈一个字恢复到IP

1）段内返回指令

- 格式2：RET　N　　（RETN　N）

- N必须使用偶数

- 功能：
- 从堆栈中出栈一个字，恢复到IP中，
- 然后执行 SP<=（SP）+ N

1）段内返回指令

- （SP）+ N 这一个操作从栈顶直接出栈 N / 2 个字，这一功能主要用于清除主程序通过堆栈传递的入口参数。

2）段间返回指令

- 使用段间调用指令调用的子程序必须使用段间返回指令才能正确返回主程序。

- 格式1：RET　（RETF）

- 功能：从堆栈中出栈两个字，先出栈的字用于恢复IP，后出栈的字用于恢复CS。

2）段间返回指令

- 格式2：RET　N　　（RETF N）
- N 必须是偶数。
- 功能：
- 从堆栈中出栈两个字，分别恢复到IP和CS，
- 然后执行 SP<=（SP）+ N

调用与返回的搭配

- 主程序中的调用指令和子程序中的返回指令必须正确的配合使用才能保证正确的调用和返回过程。

- 段内调用必须和段内返回搭配，段间调用必须和段间返回搭配。

调用与返回的搭配

- 在语法上RET形式表示返回指令，既可以解释为RETN，也可解释为RETF。
- 具体如何解释由汇编程序根据该子程序的类型来决定。
- 若在程序中明确使用RETN或RETF形式的返回指令，那么一定注意搭配关系。

6.2 子程序的设计要求

- 1. 具有通用性和独立性

- 一般在程序设计中，具有一定独立性和通用性，且被反复使用的功能模块适合于组织为子程序的形式。

- 使用子程序可以提高代码重用率、编程效率、简化程序结构。

6.2 子程序的设计要求

- 适合于编制为子程序的功能模块：

- （1）字处理软件中：在字符串中查找某一个给定的字符；比较两个字符串是否相等；合并两个字符串；在字符串指定位置插入一个新的字符；删除字符串指定位置的字符。

- （2）图形生成软件中：在屏幕指定位置显示一个点；在屏幕指定位置显示一条直线；在屏幕指定位置显示一个多边形。

6.2 子程序的设计要求

- 2. 选用适当的方法实现主程序和子程序间的参数传递

- 主程序和子程序之间一般会涉及到参数的传递过程，主程序需要为子程序提供入口参数，子程序则需要向主程序返回出口参数。

子程序的参数传递

- 在高级语言程序设计中只需考虑子程序入口参数和出口参数的构成，具体的参数传递方法由编译系统实现。

- 在汇编语言程序设计中，必须考虑参数传递方法，一般分为三类：使用寄存器、使用堆栈、使用数据段中参数传递区域。

子程序的参数传递

- 在具有调用关系的主程序和子程序之间必须保持相同的参数传递方式，并且必须保持对所传递参数的一致解释。

- 在程序设计中，一定要保持主程序和子程序之间对参数传递的**约定**（由程序员来确定）。

6.2 子程序的设计要求

- 3. 保护主程序的执行现场

- 执行现场主要是指主程序执行期间，CPU中各个寄存器和标志位的状态。

- 主程序在调用子程序那一时刻的执行现场应该受到保护，因为子程序的执行一般会破坏原有的执行现场。

现场保护

- 若在子程序调用前后执行现场发生了改变，且主程序并不知道这种改变，则主程序的执行可能发生不可预期的错误。

现场保护

- 保护执行现场一般使用堆栈来实现，执行子程序前，先将子程序所使用的所有寄存器压入堆栈保存。
- 子程序执行完毕以后，从堆栈恢复主程序的执行现场。
- 为降低现场保护的复杂性，现场保护的相关代码一般设计在子程序中。

6.2 子程序的设计要求

- 4. 正确使用堆栈

- 在子程序设计中，最容易出现错误的是堆栈操作。

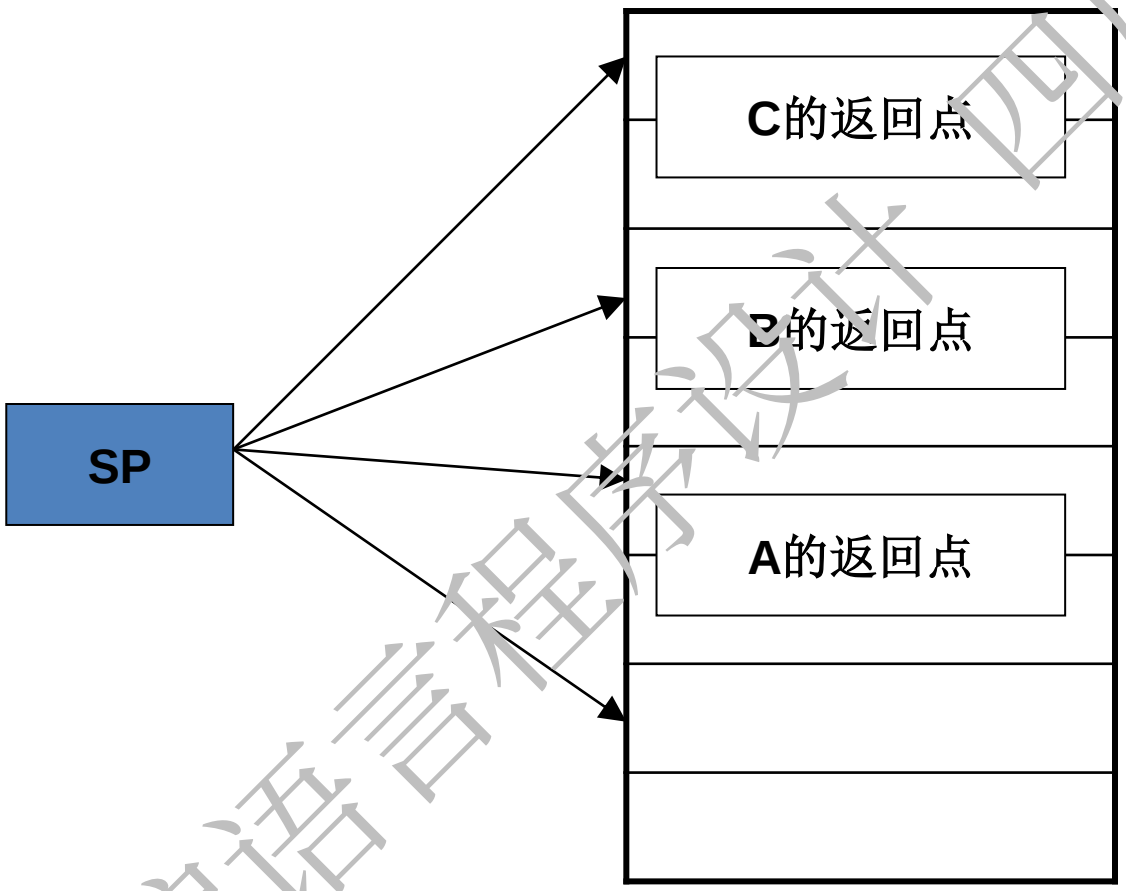
- 重要的规则：除非通过堆栈传递参数，子程序执行RET指令后，（SP）必须和主程序执行CALL指令前保持一致。

段内调用与返回的堆栈操作

- 假设A为主程序，B、C、D为子程序，调用关系为A调用B，B调用C，C调用D。

- 调用与返回时栈顶的变化如下所示：

段内调用与返回的堆栈操作



6.3 子程序与主程序间的参数传递

- 结合一个实际例子说明三种参数传递方式
- 子程序功能：实现对8位或者16位二进制数的ASCII转换，转换后得到的字符串存放在主程序指定的位置。

6.3 子程序与主程序间的参数传递

- 入口参数：
- 1）待转换的二进制数
- 2）转换位数
- 3）转换后字符串存放的首地址

- 出口参数：无

6.3 子程序与主程序间的参数传递

- C语言中该子程序（函数）定义格式：

- void Transfer_Bin（BYTE* lp_Data, int Bit_Num, char* lp_Res）

6.3.1 使用寄存器存放参数

- 主程序先将入口参数传送到约定好的寄存器中，然后使用CALL指令调用子程序。
- 子程序执行后首先从相应的寄存器中取出参数，然后再执行程序主体。
- 若存在出口参数，则子程序在返回前将出口参数按照约定保存到指定的寄存器。

6.3.1 使用寄存器存放参数

- 参数传递约定：

- 使用DX存放待转换的二进制数，使用CX存放数据的位数，使用DI存放子程序得到的字符串首地址。

寄存器传递方式

- DATA SEGMENT
- BIN8 DB 57H
- BIN16 DB 47bcH
- ASCBUF DB 20H DUP(20H)
- DATA ENDS

寄存器传递方式

- STACK1　SEGMENT　STACK
- 　　DW　　40H　DUP(0)
- STACK1　ENDS

寄存器传递方式

```
CODE SEGMENT
    ASSUME  CS:CODE,DS:DATA,SS:STACK1
MAIN:    MOV    AX,DATA
         MOV    DS,AX
         MOV    DX,0
         ;转换二进制数送DX
         MOV    DL,BIN8
         ;置位数
         MOV    CX,8
```

寄存器传递方式

- LEA DI,ASCBUF ;置存放ASCII串首地址
- CALL BITASC ;调子程序BITASC
- MOV DX,BIN16 ;16位二进制数送DX
- MOV CX,16
- LEA DI,ASCBUF+10H ;置存放ASCII串首地址
- CALL BITASC
- MOV AH,4CH
- INT 21H ;主程序结束

寄存器传递方式

- ；代码转换子程序:BITASC

- BITASC　PROC
- 　　PUSH　AX　　;保存AX
- 　　CMP　　CX, 8　;判断转换位数
- 　　;直接转换16位数
- 　　JNE　　TRANS
- 　　MOV　　DH, DL　;8位转换数送DH

寄存器传递方式

```
TRANS:  ROL    DX, 1   ;按位转换,存放结果
        MOV    AL ,DL
        AND    AL ,1
        ADD    AL ,30H
        MOV    [DI] , AL
        INC    DI
        LOOP   TRANS
        POP    AX      ;恢复AX
        RET
BITASC  ENDP
CODE    ENDS
        END    MAIN
```

6.3.2 使用堆栈传递参数

- 主程序先按照约定顺序将入口参数压入堆栈，然后使用CALL指令调用子程序。

- 子程序执行前使用BP作为地址指针，按照约定顺序从堆栈中取出入口参数，然后执行程序主体。

6.3.2 使用堆栈传递参数

- 若存在出口参数，则使用BP作为地址指针按照约定顺序把出口参数送入堆栈，一般覆盖入口参数。

- 最后用 RET N 的形式返回主程序，把程序流程返回到主程序，同时清除掉堆栈中的入口参数。

堆栈传递方式

- 例中：主程序按照先后顺序把待转换二进制数据、数据位数、字符串首地址压入堆栈，子程序使用BP指针按照约定顺序取出参数。

- 使用堆栈传递参数时，子程序需使用BP寄存器，若不希望子程序影响该寄存器内容，则在保护执行现场时也需保护BP。

堆栈传递方式

```
CODE      SEGMENT
ASSUME    CS:CODE,DS:DATA,SS:STACK1
MAIN:      MOV     AX, DATA
           MOV     DS,  AX
           MOV     AX,  0
           MOV     AL,  BIN8
           ;待转换二进制数压栈
           PUSH    AX
           ; 待转换位数压栈
           MOV     AX,  8
           PUSH    AX
```

堆栈传递方式

```
; 存放ASCII串首地址压栈
LEA       AX,  ASCBUF
PUSH      AX
CALL      BITASC   ;调子程序
MOV       AX,  BIN16
PUSH      AX
MOV       AX,  16
PUSH      AX
```

堆栈传递方式

- ; 置存放ASCII串首址
- LEA　　AX, ASCBUF+10H
- PUSH　AX
- CALL　BITASC
- MOV　 AH, 4CH
- INT　　21H

执行CALL指令后堆栈的状态

堆栈传递方式

```
;代码转换子程序
BITASC      PROC
            ;当前SP送BP
            MOV     BP, SP
            ;保存AX, CX,DX,DI
            PUSH    AX
            PUSH    CX
            PUSH    DX
            PUSH    DI
```

堆栈传递方式

- MOV DI, [BP+2]
- MOV CX, [BP+4]
- MOV DX, [BP+6]
- CMP CX, 8
- JNE TRANS
- MOV DH, DL

堆栈传递方式

```
TRANS:    ROL   DX, 1   ;按位转换,并存放
          MOV   AL, DL
          AND   AL,1
          ADD   AL, 30H
          MOV   [DI], AL
          INC   DI
          LOOP  TRANS
          POP   DI
          POP   DX
```

堆栈传递方式

```
        POP     CX
        POP     AX
        RET     6
BITASC  ENDP
CODE    ENDS
        END     MAIN
```

堆栈传递方式

- 课下练习：
- 作图说明程序执行中，堆栈的变化情况。

6.3.3 使用参数区（地址表）传递参数

- 参数区是在数据段中的一块存储区域，用于存放主程序与子程序间相互传递的参数。

- 其中的内容可能有参数的具体取值，也可能有参数的地址。

6.3.3 使用参数区传递参数

- 主程序首先按照约定的顺序把入口参数或者入口参数的地址传送到参数区中约定位置。

- 子程序按照约定顺序从参数区中取出入口参数。子程序执行完毕后，在返回主程序前，按照约定顺序把出口参数传送到参数区约定位置保存。

参数区（地址表）传递方式

```
data  segment
arg_table  dw   3  dup(?)
bin8        db   35h
bin16       dw   0ab48h
ascbuf      db   20h  dup(20h)
data  ends

stack1 segment  stack
            dw  20h  dup(?)
stack1 ends
```

参数区（地址表）传递方式

- code segment
- assume cs: code, ds: data, ss: stack1
- begin: mov ax, data
- mov ds, ax
- xor bx, bx
- mov bl, bin8
- mov arg_table, bx
- mov word ptr arg_table+2, 8

参数区（地址表）传递方式

- mov bx, offset ascbuf
- mov arg_table+4, bx
- mov bx, offset arg_table
- call   bitasc
- ……

参数区（地址表）传递方式

- bitasc proc
- 　　push ax
- 　　push cx
- 　　push dx
- 　　push di

参数区（地址表）传递方式

- mov dx, [bx]
- mov cx, [bx+2]
- mov di, [bx+4]
- cmp cx, 8
- jne trans
- ……

6.3 子程序与主程序间的参数传递

- 入口参数与出口参数传递可使用不同的方法，入口参数一般数量较多，通过堆栈、参数区传递更加适合。

- 出口参数一般比较少，可以考虑使用寄存器传递。

6.3 子程序与主程序间的参数传递

- 用高级语言实现的程序，其主程序与子程序之间的参数传递方法是由编译系统来规定的。

- 一般是采用堆栈或者参数区（地址表）来进行参数的传递。

6.4 系统服务子程序

- 除用户设计的子程序外，操作系统还提供大量的系统服务子程序，通常称为中断服务程序。

- 这类程序通常用于完成必要的输入/输出功能和一些必要的系统操作。

6.4 系统服务子程序

- 调用系统服务子程序的方式有两种：
- 一种是硬件中断，这种方式不为程序所控制；
- 另一种是软件调用的方式，由程序中的中断调用指令来完成的，程序员可以决定何时使用这种调用。

6.4 系统服务子程序

- 调用系统服务子程序一般不使用CALL指令，而是使用中断调用指令，格式如下：

- INT　中断号

INT指令的执行过程

- （1）标志寄存器、CS、IP依次压入堆栈（保存标志寄存器，延续了硬件中断调用的特征）
- （2）IF、TF标志位清0（禁止其它可屏蔽中断和单步中断）
- （3）由中断号计算中断服务子程序的入口地址（需使用中断向量表，在后面介绍），取得入口地址以后，修改CS和IP，使流程转向中断服务子程序入口。

由中断号计算中断入口地址

- 中断向量表：内存中最低地址端1K字节为中断向量表，其中每2个字保存一个中断入口地址。

- 中断入口地址 =（中断号 * 4）

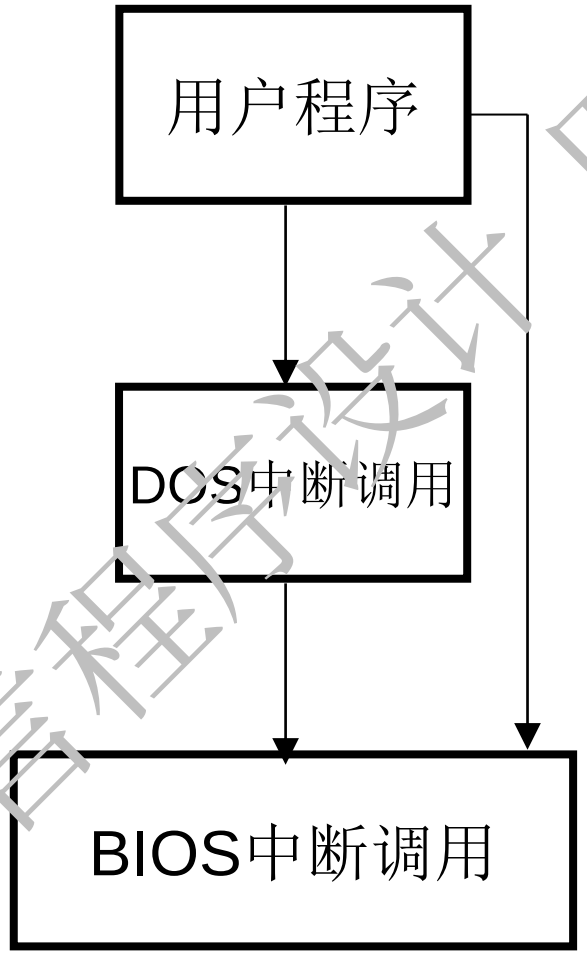
IRET指令与中断调用返回

- 与普通子程序类似，中断服务子程序执行完毕后，将程序流程返回到主程序，最后一条执行的指令是中断返回指令。

- IRET

- 功能：从堆栈中出栈三个字，依次恢复到IP、CS、标志寄存器FR。

BIOS调用与DOS调用



常用的系统调用

- INT 21H：DOS提供的服务子程序
- INT 16H：BIOS服务，处理键盘输入
- INT 10H：BIOS服务，处理显示器输出
- INT 13H：BIOS服务，处理磁盘的基本输入、输出

要求掌握的系统调用

- 输入单个字符并回显；
- 显示单个字符；
- 输入字符串并回显；
- 显示字符串

输入单个字符并回显

- 21H号调用的01H号子功能：
- 从键盘等待一个按键；

- 有按键发生时，将输入字符的ASCII码保存到AL（出口参数）；

- 并在屏幕上当前光标位置回显该字符，光标向右移动一个字符位置，自动换行、滚屏。

输入单个字符并回显

- MOV　AH，01H
- INT　　21H

显示单个字符

- 21H号调用的02H号子功能：
- DL中内容解释为ASCII码（入口参数）；
- 将DL中指定的字符在当前光标位置显示，光标向右移动一个字符位置，自动换行、滚屏。

显示单个字符

- 例：在当前光标位置显示字符A

- MOV　DL，‘A’
- MOV　AH，02H
- INT　21H

输入字符串并回显

- 21H号调用的0AH号子功能
- 入口参数：
- （DS）为字符串缓冲区所在段的段基值。
- （DX）为字符串缓冲区首字节偏移量。
- （DS：DX）约定的缓冲区最大字符数（含回车符0DH）。

- 出口参数：
- （DS：DX+1）为系统调用返回的实际字符数（不计回车符）。

输入字符串并回显

- 从键盘接收多个字符，将字符按照接收顺序由低地址到高地址保存在指定的缓冲区中。
- 在屏幕上回显字符串，光标移动到字符串末尾。
- 若输入回车键，则系统调用返回；若达到约定的最大字符数，则系统调用不再接收字符，通过扬声器发出警报。
- 返回前，系统调用将实际字符数保存到缓冲区第一个字节。

输入字符串并回显

- 例：
- data segment
- maxlen db 10
- db ?
- str1 db 10 dup(0)
- data ends

输入字符串并回显

- mov ax, data
- mov ds, ax
- ……
- lea dx, maxlen
- mov ah, 0ah
- Int 21h
- ……

显示字符串

- 21H号调用的09H号子功能
- 入口参数：
- （DS）为字符串缓冲区所在段的段基值。
- （DX）为字符串缓冲区首字节偏移量。

- 字符串末尾字符必须为 ‘$’。

显示字符串

- 显示指定缓冲区内的字符串，光标移动到字符串末尾。
- 系统调用逐个字节显示缓冲区中字符，直到检测到‘$’字符。
- 字符串必须以‘$’结尾。

显示字符串

- DATA SEGMENT
- STR1 DB ‘ HELLO! $’
- DATA ENDS
- ……
- MOV AX, DATA
- MOV DS, AX
- ……
- LEA DX, STR1
- MOV AH, 09H
- INT 21H
- ……

常用特殊字符

- 0AH：换行字符的ASCII码，显示该字符时不改变光标所在列，将光标下移一行。

- 0DH：回车字符的ASCII码，显示该字符时不改变光标所在行，将光标回到行首。