微机原理与接口技术

§5 MCS-51汇编程序设计

主讲人：佘青山

Homepage: https://auto.hdu.edu.cn/2019/0403/c3803a93084/page.htm
Email: qsshe@hdu.edu.cn
Mob：13758167196
Office: 第二教研楼南楼308室

2024年10月24日

指令是规定计算机（或微控制器）完成某种操作的指示和命令。根据特定任务要求，运用指令集编写的指令序列称为程序。微控制器执行不同的程序就可以完成不同的任务。针对不同的问题，应用微控制器的指令系统把解决该问题的步骤用指令有序地描述出来，就是程序设计。8051程序设计中常用的语言有汇编语言和C51高级语言。

本章内容分为20个教学单元，主要包括8051微控制器指令系统的寻址方式、5大功能指令与功能、典型指令的应用，以及**汇编语言程序设计基础、结构化程序设计、子程序设计方法以及设计实例**。

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. 基本程序设计
7. 子程序设计概述
8. 子程序设计举例

1. 编程语言

计算机的工作过程就是执行程序的过程，程序是完成某一运算或功能的若干指令的有序集合。

用于程序设计的语言可分为三种：**机器语言**、**汇编语言**和**高级语言**。

□机器语言

计算机能够识别的、用二进制代码表示的语言。

□汇编语言

用助记符表示的编程语言称为汇编语言，用汇编语言编写的程序称为汇编语言程序，或称为源程序。

□高级语言

高级语言是面向过程和问题的程序设计语言，是一种接近人们自然语言和常用数学表达的计算机语言。

2. 编程语言的特点

□ 机器语言的特点

程序不通用、不易读、易出错、难以维护，目前几乎不用机器语言编写程序。

□ 汇编语言的特点

1）**指令与机器码一 一对应**，程序效率高，占用存储空间小，运行速度快，且能反映计算机的实际运行情况，所以用汇编语言能编写出最优化的程序。

2）**汇编语言能直接管理和控制硬件设备**，如访问存储器、I/O接口，处理中断等。

3）汇编语言是“面向机器”的语言，编程比高级语言困难。

4）汇编语言通用性差，面向具体的机器，不同的微控制器具有不同的指令系统，不能通用。

高级语言的特点

1）近似于人们日常用语的语言，直观、易学、易懂，通用性强，易于移植到不同类型的机器中去。如BASIC、PASCAL、C、C++等。

2）计算机不能直接识别和执行高级语言，需要用编译程序将高级语言转换成机器语言。

3）高级语言不受具体机器的限制，而且使用了许多数学公式和习惯用语，从而简化了程序设计的过程。

8051微控制器的高级编程语言，称为C51

5.1.2 汇编语言编程风格

完成同一个任务所使用的方法或程序不是唯一的。程序设计的质量将直接影响到计算机系统的工作效率、运行可靠性。良好的编程风格对于实现高质量的程序具有重要意义。

汇编程序设计时，应关注以下几点。

1. 注释

- 注释是程序设计中的重要内容之一，汇编指令的抽象特性更要重视注释的作用。
- 注释内容用“；”与助记符指令隔离，注释内容长度不限，换行时，头部仍要标注“；”。

2. 标号的使用

- 标号由不多于8个ASCII字符组成，第一个字符必须是字母，不能使用汇编语言已定义的符号，如助记符，寄存器名等。
- 同一个标号在一个独立的程序中只能定义一次。
- 标号通常代表地址，标号名的选取应能够表达所代表的目的地址。

3. 子程序的使用

- 将大而复杂的任务划分为若干个小而简单的任务，这些小任务通过子程序的形式完成。
- 每个子程序最好都有对应的注释块，在注释块中说明子程序的出入口参数、功能等。

4. 堆栈的使用

- 用于寄存器、SFR以及内存数据的保护和恢复。

5. 伪指令的使用

- 起始汇编伪指令和结束汇编伪指令不可缺少。
- 充分利用其他伪指令，如赋值、定义字节、定义字等，增加程序的可维护性和可读性。

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. **汇编程序设计中的伪指令**
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. 基本程序设计
7. 子程序设计概述
8. 子程序设计举例

伪指令又称汇编程序控制译码指令，在汇编时不产生机器指令代码，不影响程序的执行，仅指明在机器汇编源程序时，需要执行的一些特殊操作，如指定程序或数据存放的起始地址，给一些连续存放的数据确定单元以及指示汇编结束等。

1. 起始汇编 ORG（Origin）

格式： ORG nn

口 功能：

- 给出程序存放的起始地址，nn是16位二进制数。
- ORG指令给程序起始地址或数据块起始地址赋值。
- 应放在每段源程序或数据块的开始。

1. 起始汇编 ORG

在一个源程序中可以多次使用，以规定不同程序段或数据块的起始位置，所规定的地址必须从小到大，并且不允许重叠。

例：

```
        ORG   0000H
MAIN:   MOV   SP，#6FH
        . . .
        LCALL SUB1
        . . .
        ORG   1000H
SUB1:   MOV   A，#74H
        . . .
        RET
```

主程序MAIN存放的起始地址是0000H；子程序SUB1的起始地址是1000H。

2. 赋值 EQU（Equal）

格式：字符名称 EQU 数据或表达式

□ 功能：把数据或表达式赋值给字符名称

例：

```
DATA1   EQU  22H
ADDR1   EQU  2000H
```

给标号DATA1和ADDR1赋值22H和2000H

➢ 注意

- EQU语句给字符名称赋值后，在整个程序中该字符名称的值就固定不能更改了。
- EQU定义的字符必须**先定义后使用**。

3. 定义字节 DB（Define Byte）

格式：标号: DB 字节常数或字符串

口 功能：将常数或字符串存入从标号开始的连续存储单元中。

例： ORG 2000H

TABLE： DB 73H,04,100,32,00,-2,"ABC"；

表示： TABLE 标号的起始地址为2000H；

数据73H,04H,64H,20H,00H, FEH,41H,42H,43H

依次存入 2000H 开始的ROM中。

TABLE:	2000H	73H
	2001H	04H
	2002H	64H
	2003H	20H
	2004H	00H
	2005H	FEH
	2006H	41H
	2007H	42H
	2008H	43H
	2009H	
	200AH	
	200BH	

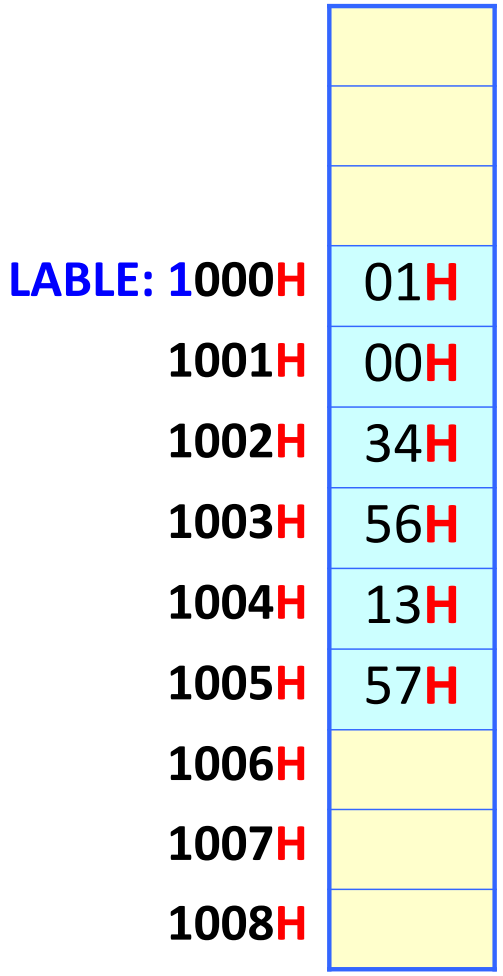
4. 定义字 DW（Define Word）

- **格式：标号： DW 字或字串**
- **功能：把字或字串存入由标号开始的连续存储单元中，且把字的高字节数存入低地址单元，低字节数存入高地址单元。**

例： ORG 1000H

LABLE： DW 100H，3456H，1357H，……

表示： 从 LABLE 地址（1000H）开始，按顺序存入 01H,00H,34H,56H，13H,57H,……



- **注意：**
- DB和DW定义的数表，其个数不得超过80个，若数据的数目较多时,可以使用多个定义命令。
- 通常用DB来定义数据，用DW来定义地址。

5. 位地址赋值 BIT

- 格式：字符名称 BIT 位地址

  位地址可以是绝对地址，也可以是符号地址。

- 功能：用于给字符名称赋予位地址。

例：
```
LED1  BIT   P3.1   ；LED1赋值为P3.1，在整个程序中，LED1即为P3.1。
      SETB  LED1   ；设置P3.1为高电平
```

6. 结束汇编 END

- 格式：END
- 功能：表示程序的结束

程序的结尾必须要有END语句，而且只能有一条。

7. 定义存储器空间 DS（Define Storage）

- 格式：标号： DS nn

- 功能：通知汇编程序，在目标代码中，从标号地址开始，保留出nn个存储单元。

例： BASE: DS 100H

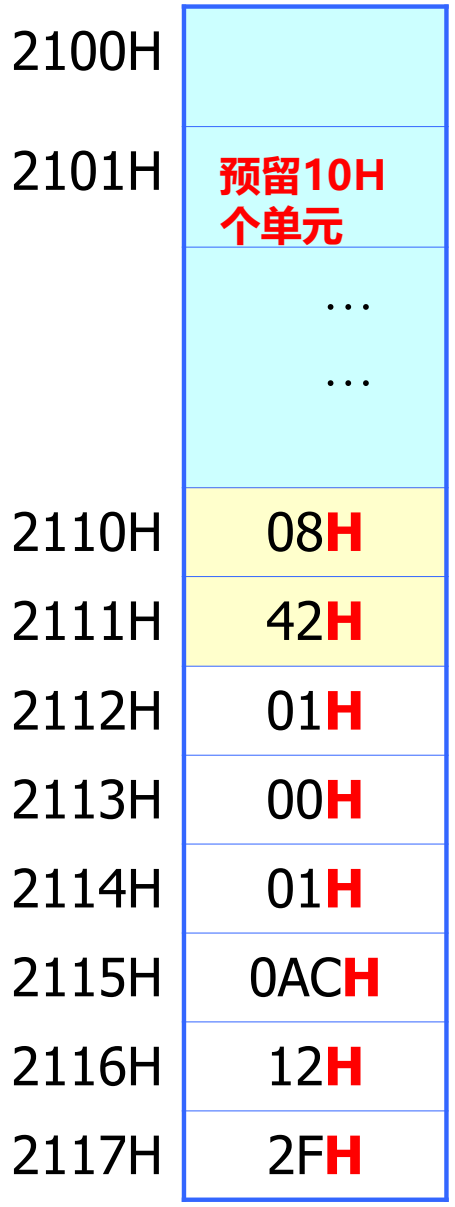
从标号BASE开始，保留出100H个存储单元，以备源程序另用。

- 注意：

- 对于8051微控制器，DB、DW、DS等伪指令是应用于ROM，而不能用于RAM。

8. 伪指令应用举例

```
        ORG  2100H
BUF:    DS   10H
        DB   08H,42H;
        DW   100H,1ACH,122FH;
```



说明:

- 从2100H至210FH为预留的缓冲区空间
- (2110H)=08H　(2111H)=42H
- 2112H单元起依次存放01H、00H、01H、ACH、00H、12H、2FH

单选题 1分

设置

汇编语言源程序中的伪指令，在汇编时不产生机器码。

A √

B ×

提交

单选题 1分

设置

END表示程序指令执行到此结束。

A √

B ×

提交

单选题 1分

设置

NOP不会使微控制器产生任何操作，因此属于伪指令。

A √

B ×

提交

单选题 1分

设置

在8051微控制器中，伪指令ORG XXXX(16位地址)的功能是_____。

A 用于定义字节

B 用于定义字

C 用于定义汇编程序的起始地址

D 用于定义某特定位的标识符

提交

第5章 内容提要

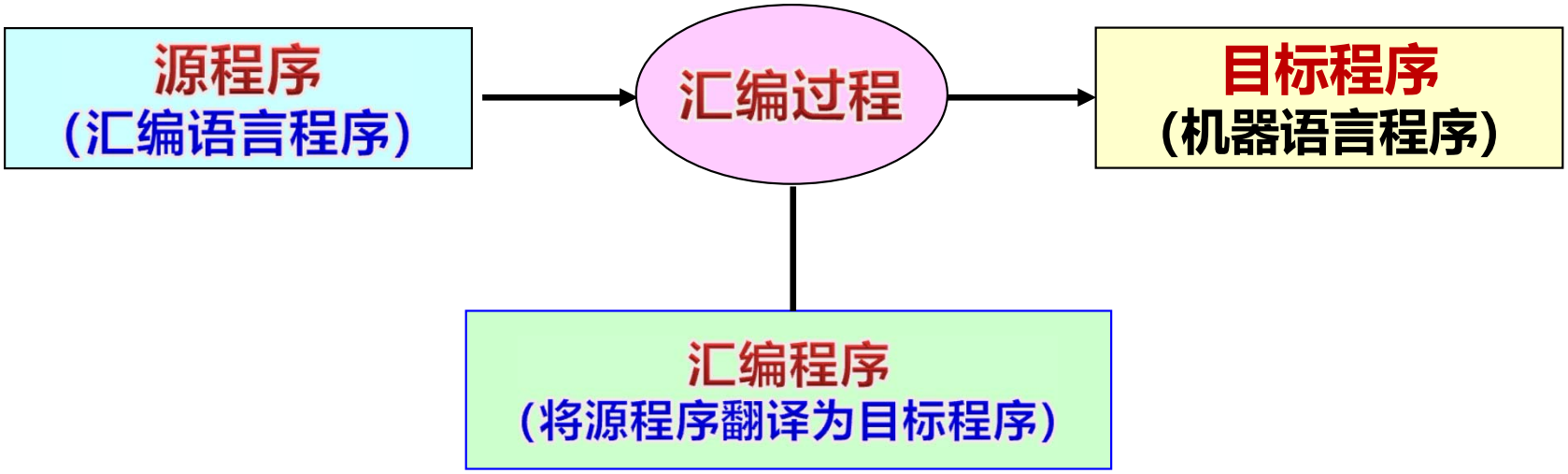
1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. **汇编与调试过程**
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. 基本程序设计
7. 子程序设计概述
8. 子程序设计举例

1. 汇编程序的编辑

设计程序时，首先应用某个编辑软件完成源程序的编写，扩展名为.ASM。

2. 汇编程序的汇编

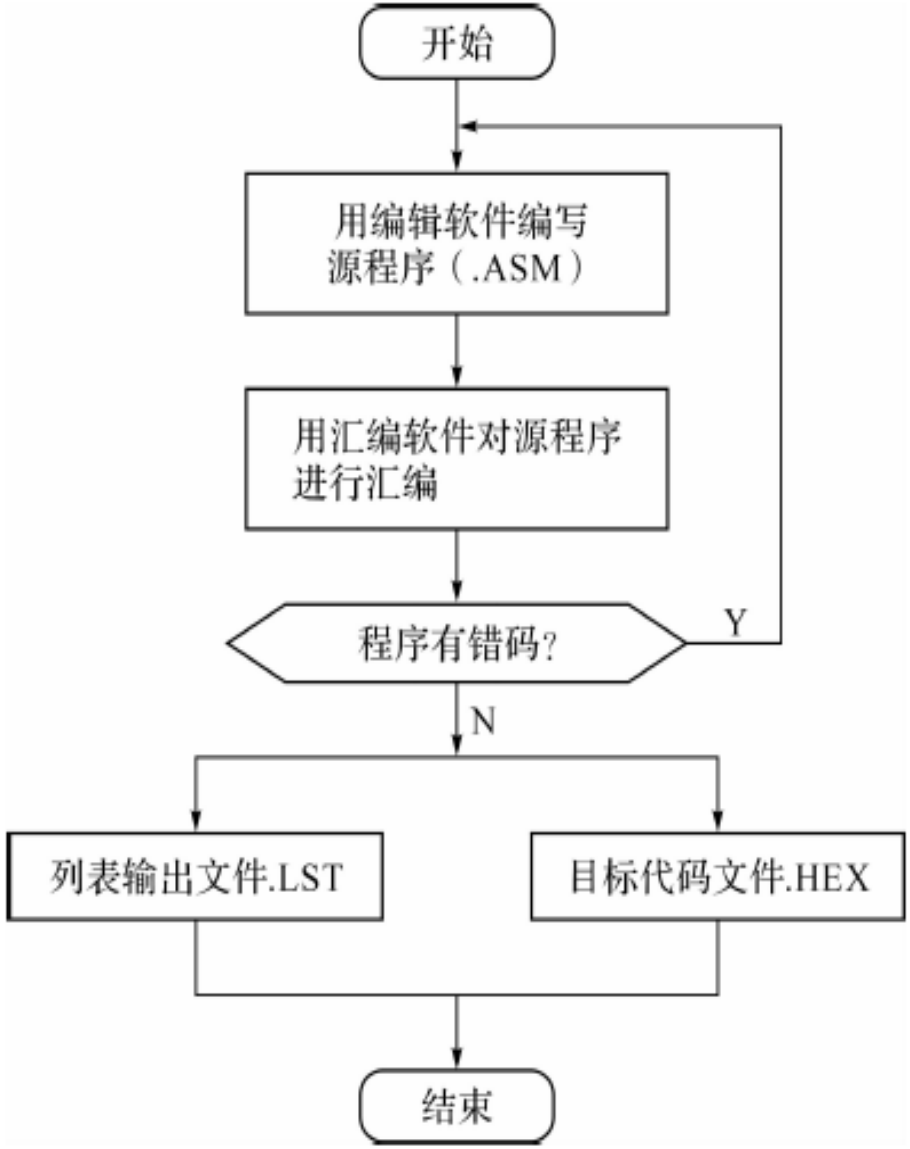
将汇编程序转换成机器程序（目标程序）的过程，称汇编。有人工汇编和机器汇编两种方式。



机器汇编过程

3. 汇编语言程序的设计过程

- 编写好源程序后进行汇编操作。如果源程序有语法错误，汇编程序会列出出错个数及错误语句所在行。
- 再返回编辑状态、修改源程序，再进行汇编操作，如此往复，直到无错误为止，即汇编通过。
- 汇编通过后会形成两个文件，即列表程序文件(.LST)和目标程序文件(.HEX) 。列表程序为汇编后的程序清单；目标程序文件是可执行文件。



4. 汇编程序的调试过程

对于汇编通过的程序即可进行调试，可以在通用计算机环境下的模拟调试或在仿真器、目标系统上，进行在线实时仿真调试。

- 模拟仿真调试：将目标程序文件在模拟调试软件环境中，模拟程序运行状态的调试。难以对外围电路进行调试。
- 实时目标仿真调试：通过串行口将汇编好的目标程序文件传送到实时在线仿真器中，，实时仿真器通过仿真头与目标系统相连。仿真器为目标系统提供了一个可单步、可设断点运行、可修改、可观察运行状态。
- 脱机运行：经实时目标仿真调试通过的系统程序，通过程序写入器写入到目标系统的ROM中，进行脱机试运行。如果运行良好，则系统程序设计调试完毕，否则需要重新修改源程序，反复进行以上操作，直到成功。

4. 汇编程序的调试过程

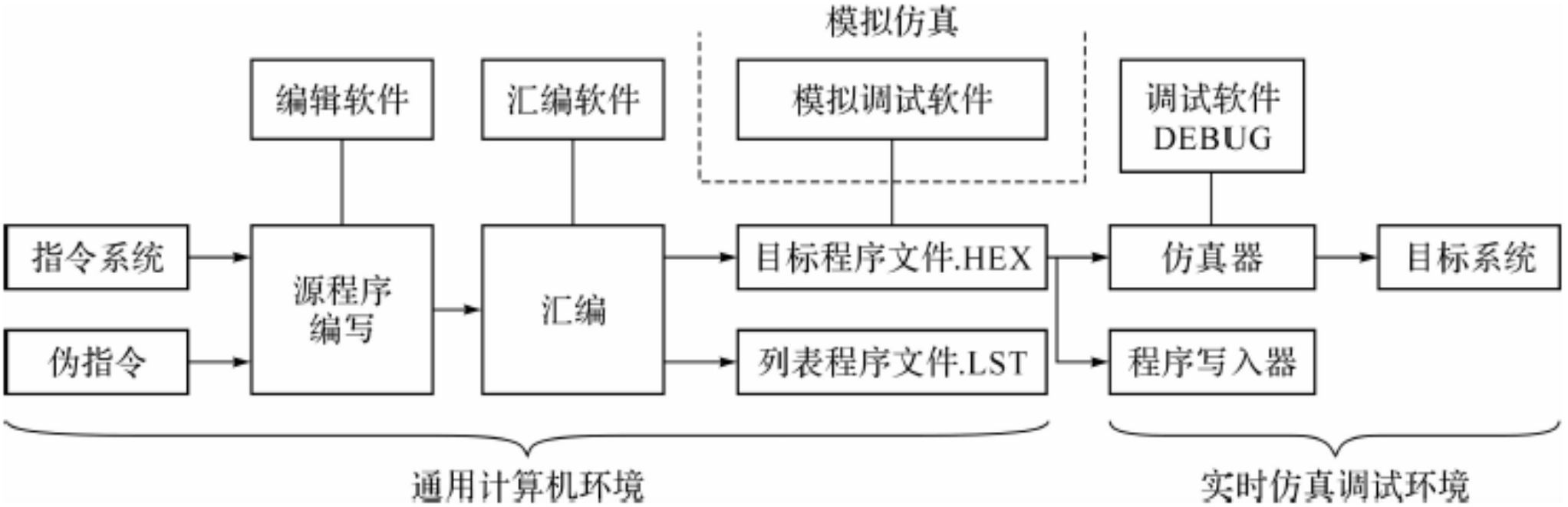


图 3-4　程序编辑、汇编运行调试的全过程

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. **汇编语言程序设计概述**
5. 程序设计的结构化
6. 基本程序设计
7. 子程序设计概述
8. 子程序设计举例

1. 评价程序质量的标准

- 程序的执行时间，程序长度；
- 程序的逻辑性、可读性；
- 程序的兼容性、可扩展性；
- 程序的可靠性。

2. 采用汇编语言的优点

- 占用的内存单元和CPU资源少；
- 程序简短，执行速度快；
- 可直接调动微控制器的硬件资源，有效地利用微控制器的专有特性；
- 能准确地掌握指令的执行时间，适用于实时控制系统。

3. 汇编语言程序设计的步骤

- **模块划分：**根据设计系统的功能需求，进行功能模块的划分，把一个大而复杂的功能划分为若干个相对独立的功能模块。
- **模块功能分析：**尽可能将一个功能设计为一个子程序；仔细分析每个子程序的功能与具体实现方法，确定并画出子程序的流程图。
- **子程序功能和资源确定：**确定子程序名、调用条件、出入口参数等，以及程序中使用的工作寄存器、内存单元和其它硬件资源。
- **子程序编写调试：**按照各子程序流程图，分别编写源程序并进行汇编、调试和运行，直至实现各子程序的预期功能。
- **总程序构建：**有机整合各子程序构成系统总程序，并进行系统总体程序的分析调试，直至实现系统全部功能。

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. **程序设计的结构化**
6. 基本程序设计
7. 子程序设计概述
8. 子程序设计举例

结构化程序设计方法是程序设计的常用方法。结构化程序设计是对用到的**控制结构类程序**作适当的限制，特别是限制转向语句的使用，从而控制程序的复杂性，使程序易读易理解，减少逻辑错误。

根据结构化程序设计的观点，任何复杂的程序都可由**顺序结构**、**分支结构**和**循环结构**三种基本结构组合而成。

- **结构化程序设计的优点：**控制程序的复杂性，使程序易读易理解，减少逻辑错误。
- **结构化程序设计的特点：**程序结构简单清晰、易读/写、调试方便、生成周期短及可靠高。
- **程序设计的结构类型：**顺序结构、分支结构、循环结构

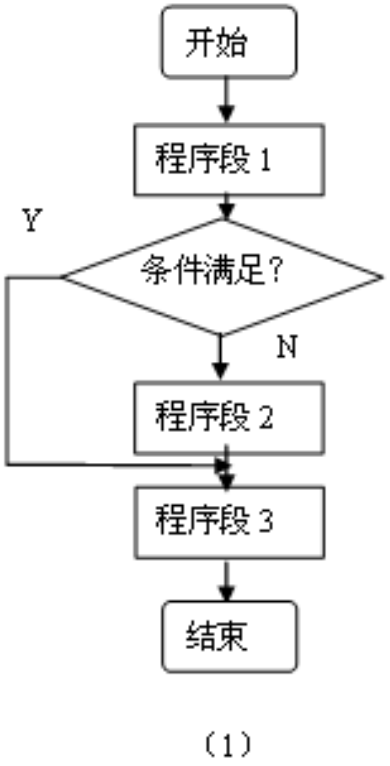
1. 顺序结构

按照逻辑操作顺序，从某一条指令开始逐条顺序执行，直至某一条指令为止。**具有一定功能的顺序程序是构成复杂程序的基础。**

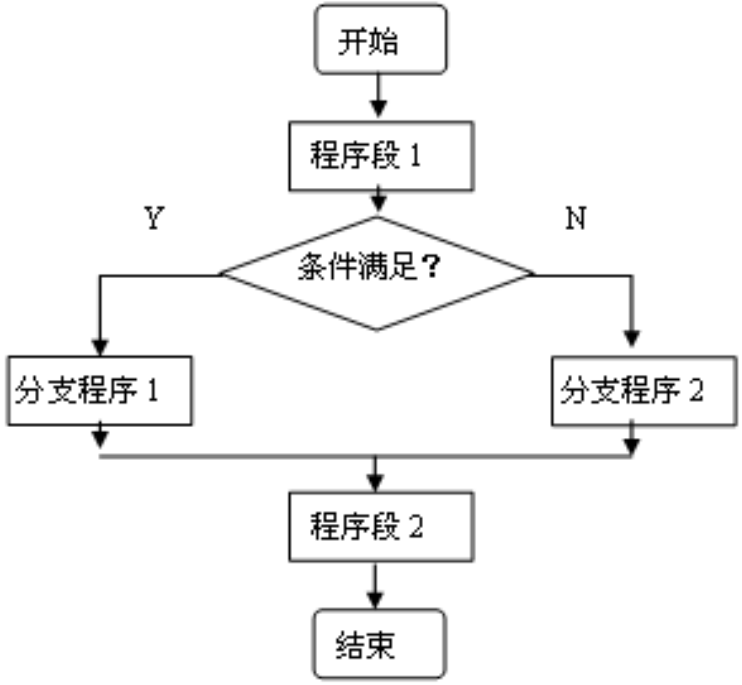
2. 分支结构

包含条件判断指令，程序执行流程中需要做出逻辑判断，并根据判断结果选择合适的执行路径。

分支结构有**单分支结构、多分支结构。**

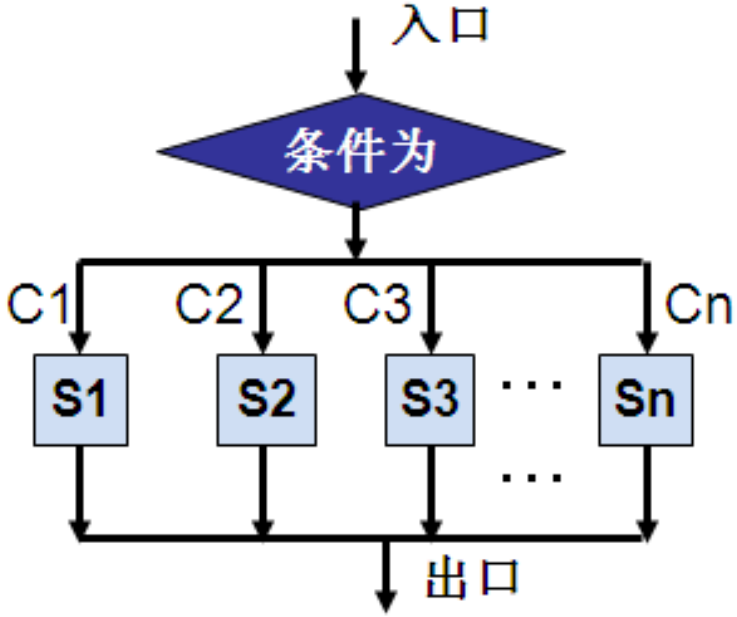


(1)



(2)

单分支结构



多分支结构

2. 分支结构

(1) 单分支结构

通常用条件转移指令来实现程序的分支。相关指令有:

- **位条件转移指令**，如：**JC、JNC、JB、JNB**和**JBC**等;
- **条件转移指令**，如：**JZ、JNZ、DJNZ**等。

(2) 多分支结构

当程序通过判别，有两个以上的不同走向时，称为多分支结构。 对于8051微控制器，可实现多分支选择的相关指令有:

- **散转指令： JMP @A+DPTR**

  根据A的内容选择对应的分支程序,可达256个分支。

- **比较转移指令：如 CJNE 指令4条**

  比较两个数的大小，必然存在大于、等于、小于三种情况，因此可实现三个程序分支。

3. 循环结构

(1) 循环结构由初始化、循环体、循环控制和结束四部分组成

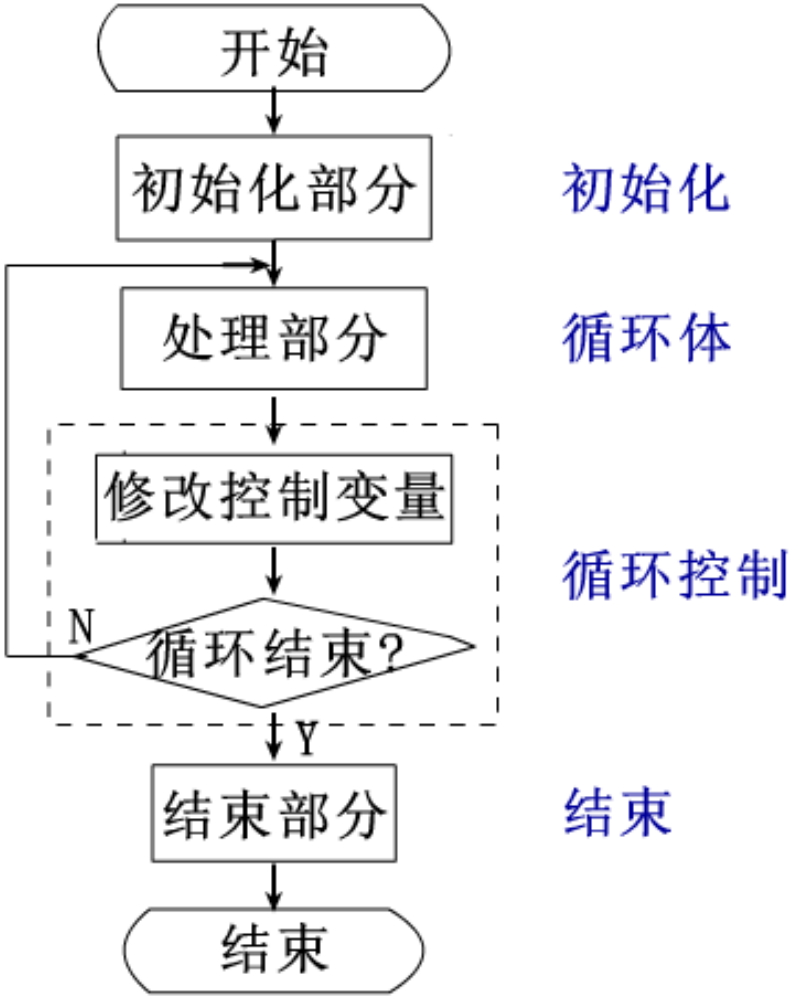
- **初始化部分**：程序在进入循环之前，应对循环过程的工作单元，如循环次数、起始地址等变量设置初值，为循环做准备。有些情况下还要保护现场。
- **循环体**：是循环结构程序核心部分，完成实际的处理工作，需反复循环执行。
- **循环控制**：循环程序的控制部分，通过循环变量和结束条件进行控制。在循环体执行过程中，要不断修改循环变量和地址指针等有关参数，当符合结束条件时，结束程序的执行。
- **循环结束**：对循环程序执行的结果进行分析、处理和保存。如初始化部分进行了保护现场，则在这里需恢复现场。

3. 循环结构

(2) 循环控制方式

循环控制的实现方法主要有**计数控制法**和**条件控制法**。即通过修改循环变量或判断循环条件，实现对循环的判断和控制。

- **计数循环结构**：循环次数已知(初始化中已设定其初值)，由循环次数决定循环体的执行次数。常用DJNZ的2条指令进行控制。计数循环结构一般采用**先处理后判断**的流程。



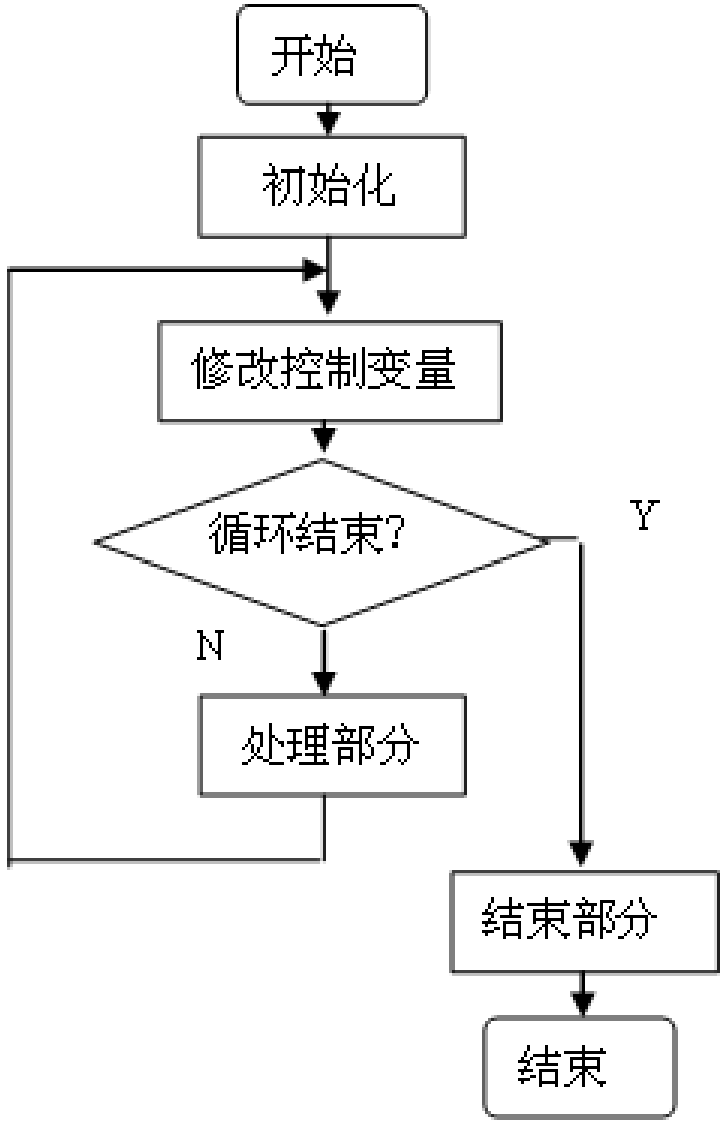
3. 循环结构

(2) 循环控制方式

- **条件循环结构**：根据循环结束的条件，决定是否继续循环程序的执行。

结束条件可以是搜索到某个参数（比如空格的ASCII码20H），也可以是发生某种状态（如电平变化）等，什么时侯结束循环是不可预知的。

常用比较转移指令或条件转移指令，来进行条件控制。一般采用先判断后处理的流程。



3. 循环结构

(3) 循环程序设计的注意点

- 在进入程序之前，应合理设置循环初始变量。
- 循环体只能执行有限次，如果无限执行的话，则会造成**死循环**，应避免这种情况的发生。
- 不能破坏或修改循环体，不能从循环体外直接跳转到循环体内。
- 在**多重循环结构**中，要求嵌套是从外层向内层一层层进入，从内层向外层一层层退出，不能在外层循环中用跳转指令直接转到内层循环体内。

单选题 1分

设置

DJNZ和CJNE指令通常放在循环体末尾并向后跳转，分别采用计数和条件判断的方式完成循环控制。

A √

B ×

提交

单选题 1分

设置

已知某汇编语言的循环体：

```
LOOP:          ......
               ......
LOOP_END:   DJNZ R0, LOOP
LOOP_AFTER: ......
```

在循环体内插入与C语言continue和break类似的功能的条件分支语句，分支目标应该分别是

A LOOP_END，LOOP_END

B LOOP_END，LOOP_AFTER

C LOOP_AFTER，LOOP_END

D LOOP_AFTER，LOOP_AFTER

提交

单选题 1分

设置

使用开发环境调试程序时，对源程序进行汇编的目的是

A 将源程序转换成目标程序

B 将目标程序转换成源程序

C 将机器语言转换成汇编语言

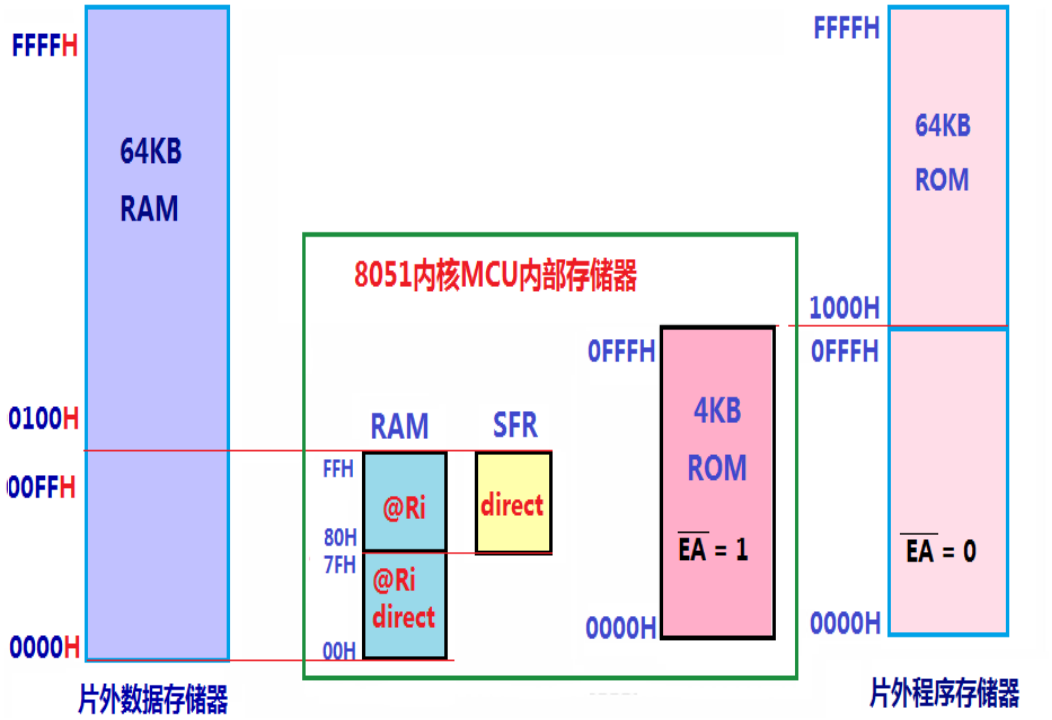
D 将高级语言转换成机器语言

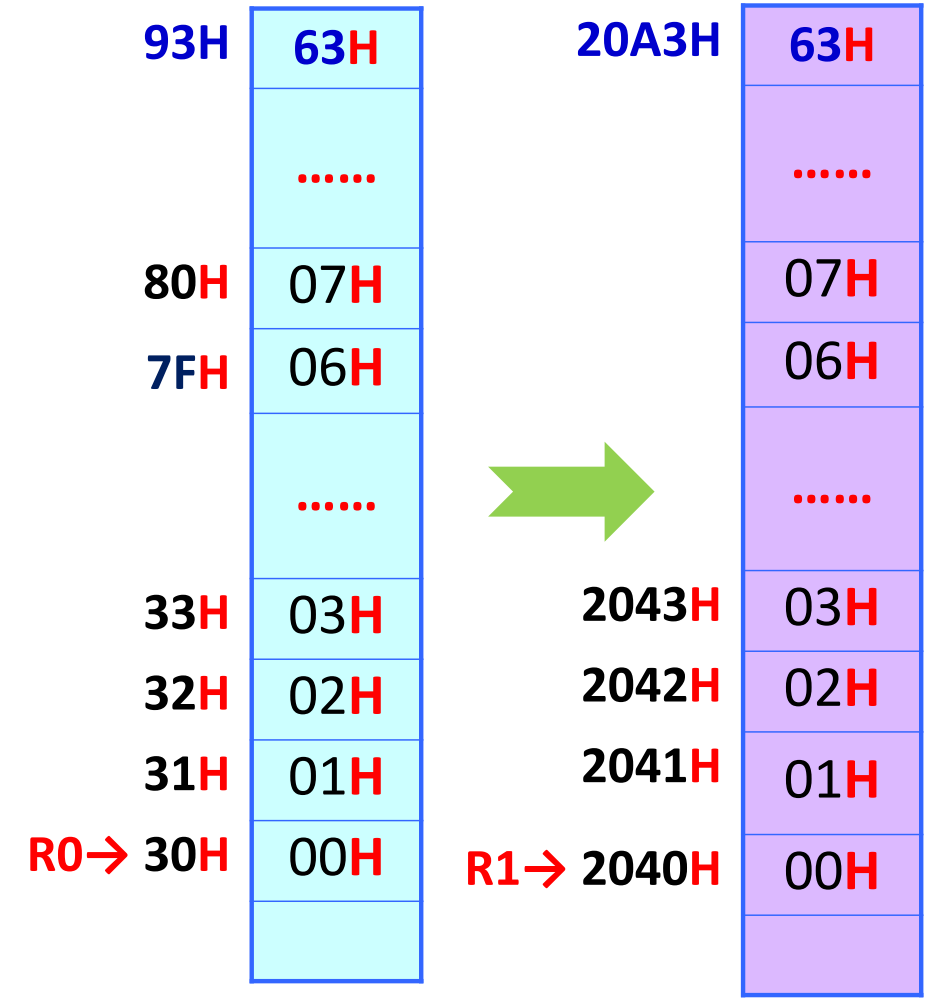
提交

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. **基本程序设计**
7. 子程序设计概述
8. 子程序设计举例

设以R0做为源数据所在内部RAM地址的指针（30H），R1做为目标外部RAM的地址指针(假设外部RAM低8位地址为40，高位地址为20H)，R2做为循环计数控制变量。编写循环程序实现数据传送。

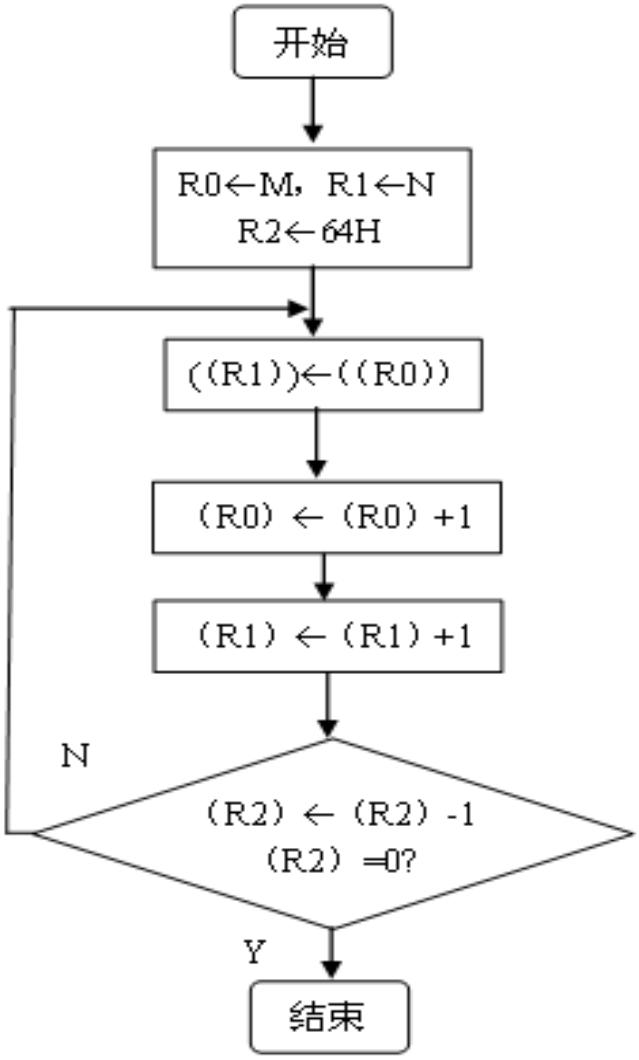




例3-24： 数据块传送程序。设在以M为起始地址的内部RAM中存有100个单字节数，试编一程序，把这100个数传送到以N为起始地址的外部RAM中。（假设外部RAM的高位地址为20H）

解： 以R0作为源数据所在内部RAM地址的指针，R1作为目标外部RAM的地址指针，R2作为循环计数控制变量。编写循环程序实现数据传送。

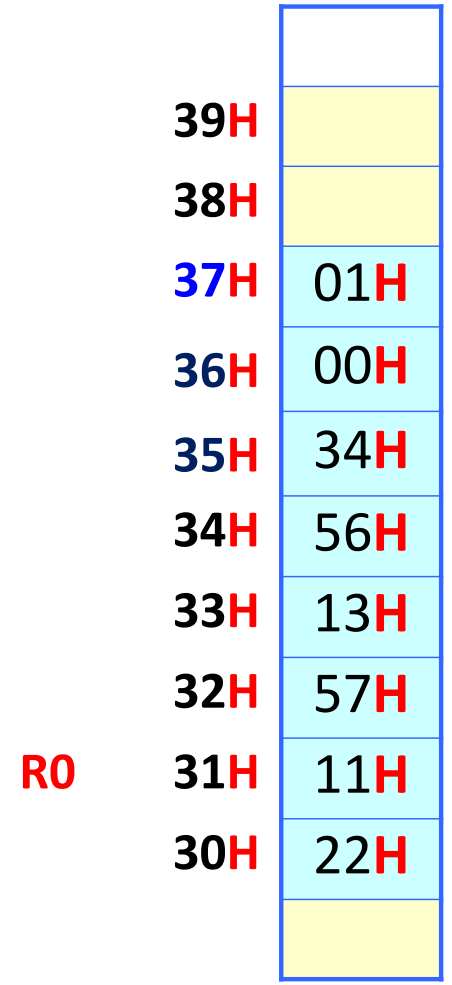
```
        RAM_HIG  EQU  20H

        ORG     0000H
        JMP     START

        ORG     0100H
START:  MOV     P2,#RAM_HIG     ;目标外部RAM高8位地址
        MOV     R0, #M          ;内部RAM起始地址
        MOV     R1, #N          ;目标外部RAM低8位地址
        MOV     R2, #64H        ;循环计数器
LP:     MOV     A, @R0
        MOVX    @R1, A
        INC     R0
        INC     R1
        DJNZ    R2, LP
        SJMP    $
        END
```



例3-25：试求内部RAM 30H～37H单元中8个无符号数的算术和，2字节结果存入38H，39H单元中。

解：8个无符号数累加，相加次数由R7控制；相加过程的**进位位**（累加和的高字节内容）累计在**R3**中。

```
        ORG    0000H
        JMP    START
        ORG    0100H
START:  MOV    R3, #0
        MOV    R7, #7H
        MOV    R0, #31H
        MOV    A, 30H
LOOP:   ADD    A, @R0
        JNC    NEXT
        INC    R3
NEXT:   INC    R0
        DJNZ   R7, LOOP
        MOV    39H, R3
        MOV    38H, A
        SJMP   $
        END
```



例3-26： 设有一个双字节的二进制数存放在内部RAM的50H（高字节）及51H（低字节）单元中，要求将其算术左移一位（即原数各位均向左移1位，最低位移入0）后仍存放在原单元。（自学）

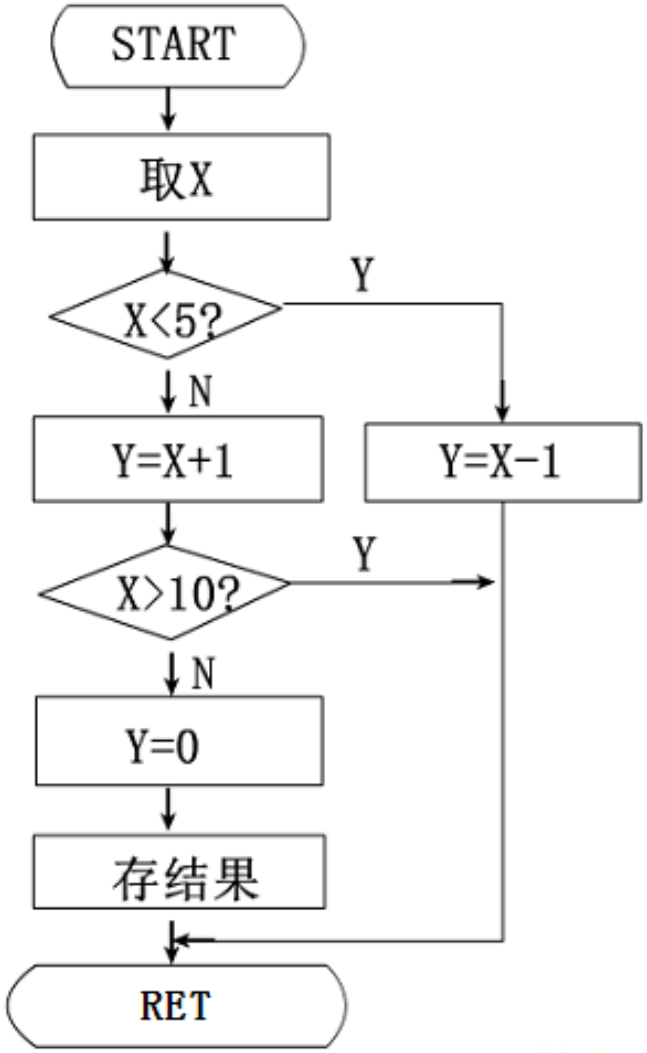
解： 16位数据左移，要求将低字节的高位移到高字节的最低位，需要采用带C的循环左移指令。先将进位标志C清零，对低字节进行循环左移，此时C的内容0进入其最低位，其最高位进入C；再对高字节进行带C的循环左移，此时C（低字节的最高位）进入高字节的最低位，从而实现16位数据的整体左移一位。

```
        ORG     0000H
        JMP     START

        ORG     0200H
STRAT:  CLR     C               ;Cy清零
        MOV     A, 51H
        RLC     A               ;低8位向左循环移1位
        MOV     51H, A
        MOV     A, 50H          ;高8位向左循环移1位
        RLC     A
        MOV     50H, A
        SJMP    $
        END
```

例3-27：设变量x存于VAR单元，试编程按下式给y赋值，并存入FUNC单元。（自学）

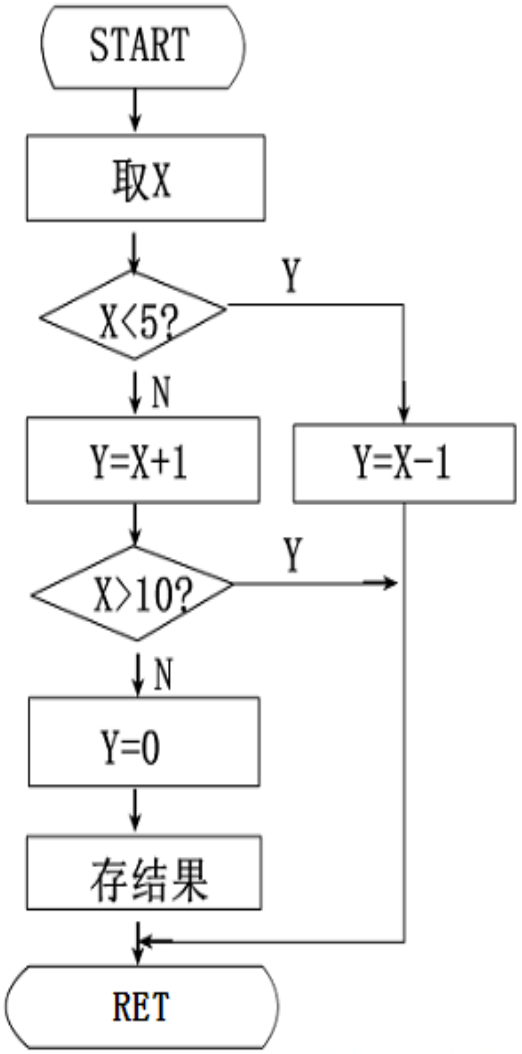
$$y=\begin{cases} x+1 & (x>10) \\ 0 & (10\geqslant x\geqslant 5) \\ x-1 & (x<5) \end{cases}$$

分析：要根据x的大小来决定y值，在判断x<5和x>10时，采用**CJNE**和**JC**以及**CJNE**和**JNC**指令进行判断，用R0暂存y的值。



程序：

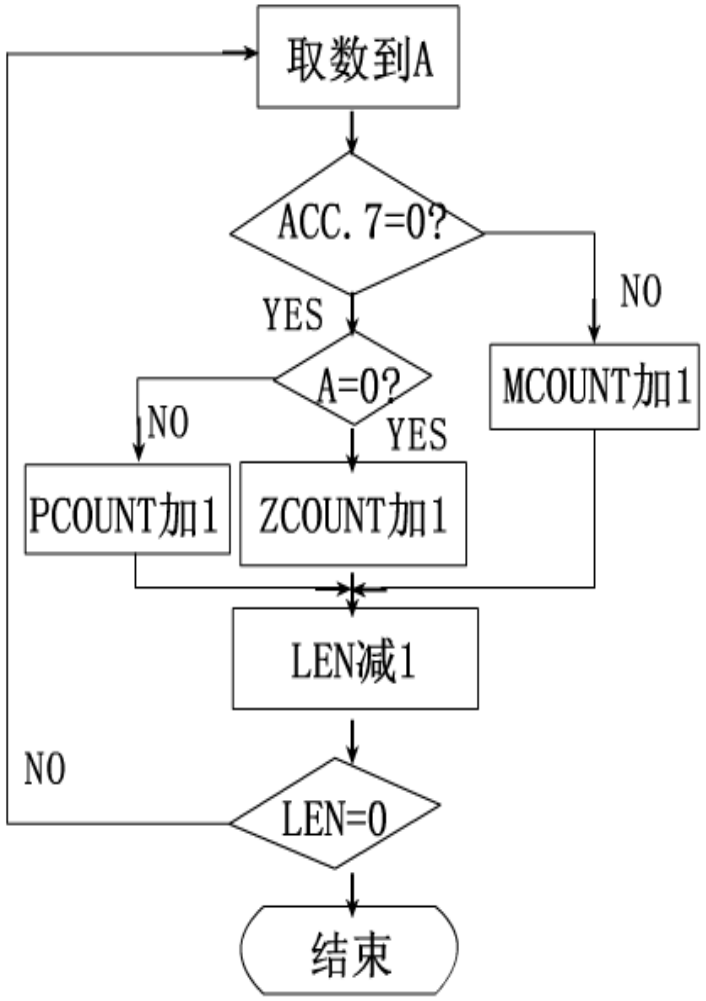
```
        x   EQU   30H
        y   EQU   31H
        ORG       0000H
        JMP       START
        ORG       1000H
START:  MOV       A, x        ;取X
        CJNE      A,#5,NEXT1  ;与5比较
NEXT1:  JC        NEXT2       ;X<5,则转NEXT2
        MOV       R0,A
        INC       R0          ;设X>10，Y=X+1
        CJNE      A,#11,NEXT3 ;与11比较
NEXT3:  JNC       NEXT4       ;X>10,则转NEXT4
        MOV       R0，#0      ;10≥x≥5,Y=0
        SJMP      NEXT4
NEXT2:  MOV       R0，A
        DEC       R0          ;X<5，Y=X-1
NEXT4:  MOV       y，R0       ;存结果
        RET
        END
```



例3-28： 在外部RAM BLOCK单元开始有一组带符号数的数据块，数据块长度存放在内存LEN单元中。试统计其中正数，负数和零的个数，并分别存入内存**PCOUNT**、**MCOUNT**和**ZCOUNT**单元中。

解： 逐一取出每个数，首先判断该数为 正数或负数或0?

- **为正数**，则PCOUNT单元加1；
- **为负数**，则MCOUNT单元加1；
- **为零**，则ZCOUNT单元加1。



判断一个数据是正数、负数和0的方法：

1. 先判是否为0，再根据最高位是0或1，判正负。

```
        MOVX  A, @DPTR
        JZ    ZERO          ;0, 转移到ZERO
        JB    ACC.7, NEG    ;负，转移到NEG
POS:    ……                  ;正的处理
NEG:    ……                  ;负的处理
```

2. 先判是否为0，然后与80H比较判正负。小于80H为正数，反之为负数。

```
        MOVX  A, @DPTR
        JZ    ZERO          ;0, 转移到ZERO
        CJNE  A, #80H, NEXT
NEXT:   JNC   NEG           ;负，转移到NEG
POS:    ……                  ;正的处理
NEG:    ……                  ;负的处理
```

2. 先判是否为0，然后与80H比较判正负。小于80H为正数，反之为负数。

(1) n位带符号二进制整数的表示范围

$$-2^{n-1} \le X < 2^{n-1}$$

(2) 数0的原码不唯一

(3) 数0的反码不唯一

(4) 补码系统只有正零，唯一

几种编码对照表

真值	原码	反码	补码
+127	0111 1111	0111 1111	0111 1111
+126	0111 1110	0111 1110	0111 1110
…….	……	……	……
+1	0000 0001	0000 0001	0000 0001
+0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	0000 0000
-1	1000 0001	1111 1110	1111 1111
……	……	……	……
-126	1111 1110	1000 0001	1000 0010
-127	1111 1111	1000 0000	1000 0001
-128	---	---	1000 0000

程序：

```
BLOCK   EQU  2000H          ;定义数据块首址
LEN     EQU  30H            ;定义长度计数单元
PCOUNT  EQU  3lH            ;正计数单元
MCOUNT  EQU  32H            ;负计数单元
ZCOUNT  EQU  33H            ;零计数单元
        ORG     0000H
        JMP     START
        ORG     0200H
START:  MOV     DPTR,   #BLOCK ;地址指针指向数据块首址
        MOV     PCOUNT, #0
        MOV     MCOUNT, #0     ;计数单元清0
        MOV     ZCOUNT, #0
LOOP:   MOVX    A,      @DPTR  ;取一个数
        JB      ACC.7,  MCON   ;若ACC.7=1,转负数个数＋1
        JNZ     PCON           ;若(A)≠0,转正负数个数＋1
        INC     ZCOUNT         ;若(A)=0,则零的个数加1
        SJMP    NEXT
MCON:   INC     MCOUNT         ;负计数单元加1
        SJMP    NEXT
PCON:   INC     PCOUNT         ;正计数单元加1
NEXT:   INC     DPTR           ;修正地址指针,指向下一个单元
        DJNZ    LEN,    LOOP   ;未完继续
        SJMP    $
        END
```

例3-29：把内存中起始地址为BUFIN的数据串，传送到外部RAM以BUFOUT为首址的区域，直到发现“$”的ASCII码（24H）为止，数据串的长度在内存20H中。（自学）

分析：循环控制条件有2个。

首先是找到“$”的ASCII码结束循环，属条件控制，也是循环主结构；

其次是计数循环控制，即若找不到“$”的ASCII码，则由数据串的长度控制循环结束。

程序：

```
        BUFIN    EQU  30H
        BUFOUT   EQU  1000H
                 ORG  0000H
                 JMP  START
                 ORG  0100H
START:  MOV   R0, #BUFIN          ;内RAM首址
        MOV   DPTR, #BUFOUT       ;外RAM首址
LOOP:   MOV   A, @R0
        CJNE  A, #24H, LOOP2      ;判是否为"$"
        SJMP  LOOP1               ;是"$",则结束
LOOP2:  MOVX  @DPTR, A            ;不是"$",传送
        INC   R0
        INC   DPTR
        DJNZ  20H, LOOP           ;数据串未查完,继续
LOOP1:  RET
```

例3-30：已知内部RAM从BLOCK单元开始有一个无符号数的数据块，其长度在LEN单元，编程求出数据块中的最大值，并存入MAX单元。

分析：先将MAX单元清0，再把它和数据块中的数据逐一比较，若MAX中的数值大，则比较下一个，否则把数据块中的数据送入MAX；逐个比较，直到每个数都比较完毕。用R0作为数据块的地址指针。

```
MAX     EQU   20H
LEN     EQU   21H
BLOCK   EQU   30H
        ORG   0000H
        JMP   START
        ORG   0100H
START:  MOV   30H,   #10H      ;初始化内部RAM
        MOV   31H,   #15H
        MOV   32H,   #30H
        MOV   33H,   #20H
        MOV   34H,   #25H
        MOV   MAX,   #00H      ;MAX清0
        MOV   LEN,   #5        ;LEN 初始化
        MOV   R0,    #BLOCK    ;R0指向数据块的首地址
LOOP:   MOV   A,     @R0       ;从内部RAM读数据
        CLR   C                ;C清0
        SUBB  A,     MAX       ;（A）和（MAX）的数据相减，形成Cy
        JC    NEXT             ;若（A）<（MAX），比较下一个
        MOV   MAX,   @R0       ;若（A）>（MAX），则大的数送MAX
NEXT:   INC   R0               ;指向下一数据
        DJNZ  LEN,   LOOP      ;未比较完毕，继续
        SJMP  $
        END
```

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. 基本程序设计
7. **子程序设计概述**
8. 子程序设计举例

主要介绍子程序的特点、子程序的调用和嵌套、子程序的编写要点，以及8051MCU汇编子程序的参数传递，子程序的现场保护与恢复。

在程序设计中，将那些需多次应用、具有某种相同作用的程序段从整个程序中独立出来，单独编制成一个程序，尽量使其标准化，并存放于某一存储区域，需要时通过指令进行调用。这样的程序段，称为子程序。

子程序是指能被主程序或其它程序调用，在实现某种功能后自动返回到调用程序去的程序。

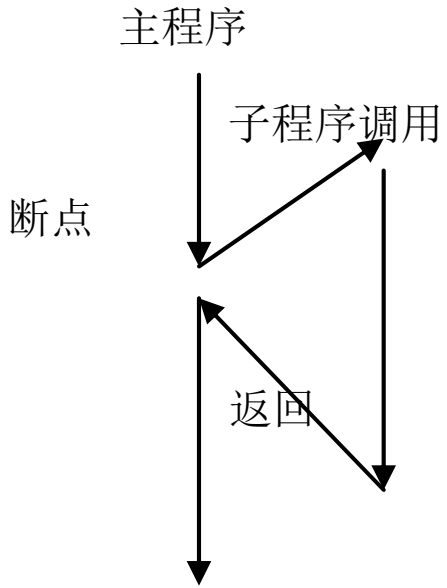
调用子程序的程序称为主程序或调用程序。

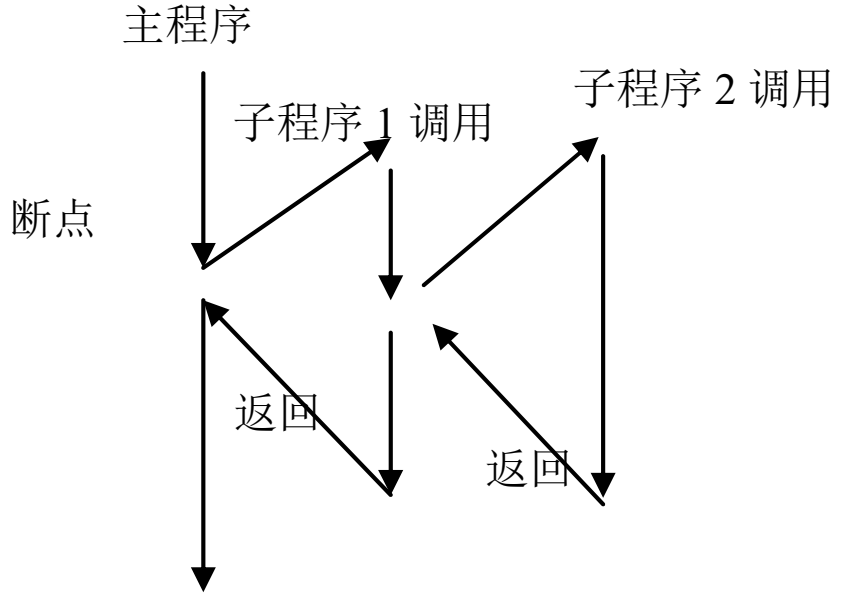
1. 子程序的优点

- 不必重复书写同样的程序，提高编程效率。
- 可使程序的逻辑结构简单，便于阅读。
- 便于程序编写、调试和修改等。
- 缩短源程序和目标程序的长度，节省程序存储器空间。
- 使程序模块化、通用化，便于交流和继承。

2. 子程序的调用和嵌套

- 通常将主程序中调用子程序指令的下一条指令的地址称为断点，子程序的第一条指令地址称为子程序首地址。
- **子程序调用：** CALL指令自动将断点地址压入堆栈保护，然后将子程序入口地址送PC，实现子程序的调用；子程序返回时，RET指令将使堆栈顶部的断点地址弹出到PC，实现子程序的返回。
- **子程序嵌套：** 在子程序的执行过程中，可能出现子程序调用其它子程序的情况，称为**子程序嵌套调用**。





3. 子程序编写要点

(1) 子程序的第1条指令必须有**标号**，该标号代表子程序的名称，也是子程序调用指令的符号地址。

(2) 子程序必须能够正确地传递参数。

- 要有入口条件用来说明调用该子程序的条件（如指明要处理的数，或存放的寄存器或内存地址等）；
- 要有出口状态，即调用子程序后的结果（结果形式或存放地址等）。

(3) 注意保护现场和恢复现场。

- 保护现场即在子程序前部将不允许被破坏的内容保护起来；
- 恢复现场即在子程序返回指令前，对保护的内容进行恢复；
- 要注意堆栈的“先进后出”操作规则，以保证现场保护和恢复的正确。

3. 子程序编写要点

(4) 要保证子程序能够正确返回。

- 首先子程序必须以RET指令结束；
- 执行RET指令时，堆栈顶部的内容是调用时保存的断点地址;
- 子程序中，对堆栈的入栈和出栈操作次数必须相同，以保证返回时堆栈指针SP的值与调用进入时一致。

(5) 子程序在功能上应具有通用性和完整性。

(6) 子程序的注释要求。

- 子程序应有功能说明，标明子程序的资源占用情况，以便调用时参考；
- 子程序应注明入口参数和出口参数，以便在调用时赋值和返回时获取结果。

4. 子程序的参数传递

当一个程序调用子程序时，通常都向子程序传递若干个数据让它来处理；当子程序处理完后，一般也向调用它的程序传递处理结果，这种在调用程序和子程序之间的信息传递称为参数传递。

- 参数传递：进行调用程序和子程序之间的信息传递
- 入口参数：调用程序向子程序传递的参数
- 出口参数：子程序向调用程序传递的参数
- 常用的参数传递方法：寄存器法和约定存储单元法

4. 子程序的参数传递

主程序或调用程序和子程序之间的参数传递方法是程序员自己或和别人事先约定好的信息传递方法。常用参数传递方法有：寄存器法和约定存储单元法。

1. 寄存器法：将入口参数和出口参数存放在约定的寄存器中。

特点：数据传递速度快、编程方便、节省内存单元，是最直接、简便的参数传递方式。但是MCU的寄存器个数有限，该方法适用于传递较少的参数信息。

2. 约定存储单元法：把入口参数和出口参数都放在事先约定好的内存单元中

特点：不占用寄存器，参数个数可以较多，每个子程序要处理的数据和送出的结果都有独立的存储单元。但是该方法要用一定数量的存储单元，会增加编程中对变量定义的难度。

5. 现场的保护与恢复

主程序的断点地址（即返回地址）是调用指令自动保护的；但是对于主程序中的数据（A、PSW、Rn等），为了防止执行子程序后遭到破坏，则需要编写程序加以保护。保护的内容为主程序和子程序都要使用的寄存器、SFR、内存等。

数据保护与恢复的三种方法：

- **堆栈保护：** 子程序首先将需要保护的数据依次压入堆栈保存， 返回指令前， 反序弹出堆栈以恢复现场；
- **切换工作寄存器组保护：** 通过修改RS1、RS0，使主程序与子程序使用不同组别的工作寄存器；
- **内存保护：** 子程序首先将需要保护的内容，保存到空闲的寄存器或内存单元暂存，返回指令前，从保存处取出以恢复现场。

5. 现场的保护与恢复

主程序:

```
PROG1:  MOV     R2, #04H
PRO1:   LCALL   SUB1
        DJNZ    R2, PRO1
        RET
```

子程序:

```
SUB1:   MOV   R2, #20H
LOOP:   DJNZ  R2, LOOP
        RET
```

出现了死循环，主程序中的R2始终不会到0

由于SUB1子程序中，也要用到R2，因此必须要先保护后恢复。

5. 现场的保护与恢复

主程序：

```
PROG1: MOV    R2, #04H
PRO1:  LCALL  SUB1
       DJNZ   R2,PRO1
       RET
```

(1) 子程序（切换工作寄存器组保护）：

```
SUB1:  SETB  RS0          ;选择第1组的Rn
       MOV   R2, #20H
LOOP:  DJNZ  R2, LOOP
       CLR   RS0          ;恢复使用第0组的Rn
       RET
```

(2) 子程序（堆栈保护）：

```
SUB1:  PUSH  02H      ; R2的内容入栈保护
       MOV   R2,#20H; 子程序使用R2
LOOP:  DJNZ  R2, LOOP
       POP   02H      ; 出栈恢复R2的内容
       RET
```

(3) 子程序（内存保护）：

```
SUB1: MOV   30H, R2   ; R2的内容暂存到30H
      MOV   R2, #20H
LOOP: DJNZ  R2, LOOP
      MOV   R2,30H    ; 恢复R2内容
      RET
```

第5章 内容提要

1. 编程语言及汇编语言编程风格
2. 汇编程序设计中的伪指令
3. 汇编与调试过程
4. 汇编语言程序设计概述
5. 程序设计的结构化
6. 基本程序设计
7. 子程序设计概述
8. **子程序设计举例**

例3-34：用程序实现$c=a^2+b^2$。设a、b均小于10，a存在31H单元，B存在32H单元，并将c存入33H单元。

分析：因两次用到求平方值，所以在程序中把求平方设计为一个子程序。

主程序：

```
ORG     200H
MOV     SP, #3FH    ;设堆栈指针
MOV     A, 31H      ;取a值
LCALL   SQR         ;求a²
MOV     R1, A       ;a²暂存R1
MOV     A, 32H      ;取b值
LCALL   SQR         ;求b²
ADD     A, R1       ;求a²+ b²
MOV     33H, A      ;存入33H
SJMP    $
```

子程序SQR：

```
;*****************************
;功能：求平方值（查表法）
;入口：A存放欲求平方的数。
;出口：A存放平方值
;*****************************
SQR:    INC     A
        MOVC    A, @A+PC
        RET
TAB:    DB  0, 1, 4, 9, 16, 25,
            36, 49, 64, 81
        END
```

例3-35：编程将片内RAM20H开始的8字节16进制数据转换为对应的ASCII码串，转换结果低半字节在前，高半字节在后，存于30H开始的片内RAM中。（自学）

子程序ASCII （查表法）：

```
;功能：将16进制数转化成ASCII码（查表法）
;入口：A低4位存放单个16进制数。
;出口：A存放转化后的ASCII码。
ASCII:  ANL    A, #0FH                ；保留低4位
        MOV    DPTR, #TABLE
        MOVC   A, @A+DPTR             ；数值换码
        RET
TABLE:  DB  30H,31H,32H,33H,34H,35H,36H, 37H,38H, 39H,
            41H,42H, 43H,44H,45H,46H
        END
```

5.8 子程序设计举例

例3-35： 编程将片内RAM20H开始的8字节16进制数据转换为对应的ASCII码串，转换结果低半字节在前，高半字节在后，存于30H开始的片内RAM中。（自学）

主程序：

```
        ORG     0000H
        JMP     START
        ORG     0100H
START:  MOV     R7, #8
        MOV     R0, #20H
        MOV     R1, #30H
LOOP:   MOV     A, @R0          ; 取出一个字节16进制数
        ACALL   ASCII           ; 转换低半字节
        MOV     @R1, A          ; 存放转换结果
        INC     R1
        MOV     A, @R0          ; 重取该字节16进制数
        SWAP    A
        ACALL   ASCII           ; 转换高半字节
        MOV     @R1, A          ; 存放转换结果
        INC     R1
        INC     R0              ; 更新源指针
        DJNZ    R7, LOOP        ; 8字节转换未结束，继续
        SJMP    $
```

例3-36： 微控制器主频为12MHz，试编写软件延时程序，延时时间分别为0.1s，1s，10s的程序。

; 主频为12MHz，则机器周期为1微秒。

程序：	助记符	机器周期
DL1:	PUSH 30H	; 2
	MOV 30H, #*N*	; 1
D1:	NOP	; 1
	NOP	; 1
	DJNZ 30H, D1	; 2
	POP 30H	; 2
	RET	; 2

要求程序执行时间1ms=1000μs=2+1+(1+1+2)×*N*+2+2

则有4*N*=993μs； 取 *N*=248=0F8H，代入后可得子程序真正延时为999μs，有千分之一误差。若在RET前加入NOP，即实现了1ms延时。

其它各档延时程序均可以通过调用1ms延时子程序来实现：

(1) 0.1s子程序

```
DL100:  PUSH   30H          ;0.1s子程序: 调用DL1共100次
        MOV    30H, #100
D100:   LCALL  DL1
        DJNZ   30H, D100
        POP    30H
        RET
```

(2) 1s子程序

```
DL1S:   PUSH   30H          ;1s子程序: 调用DL100共10次
        MOV    30H, #10
D1S:    LCALL  DL100
        DJNZ   30H, D1S
        POP    30H
        RET
```

其它各档延时程序均可以通过调用1ms延时子程序来实现：

(3) 10s子程序

```
DL10S:   PUSH   30H          ;10s子程序，调用DL100ms共100次
         MOV    30H, #100
D10S:    LCALL  DL100
         DJNZ   30H, D10S
         POP    30H
         RET
```

由于各子程序的保护和恢复30H内容、循环初始化、调用和返回等都需要时间，所以存在一定的定时误差，可通过减少基准定时DL1ms的时间来补偿。更准确且不需要消耗CPU时间资源的延时，可利用MCU中的定时器/计数器来实现。

5.6 基本程序设计

请大家思考：

```
BLOCK   EQU   2000H        ;定义数据块首址
LEN     EQU   30H          ;定义长度计数单元
PCNT    EQU   31H          ;正计数单元, positive count
NCNT    EQU   32H          ;负计数单元, negative count
ZCNT    EQU   33H          ;零计数单元, zero count
        ORG   0000H
        JMP   START
        ORG   0200H
START:  MOV   DPTR,#BLOCK  ;地址指针指向数据块首址
        MOV   PCNT, #0
        MOV   NCNT, #0     ;计数单元清0
        MOV   ZCNT, #0
LOOP:   MOVX  A, @DPTR     ;取一个数
        JNB   ACC.7, L1
        INC   ____(1)____
        SJMP  NEXT
L1:     JNZ   L2
        INC   ____(2)____
        SJMP  NEXT
L2:     INC   ____(3)____
NEXT:   INC   DPTR         ;修正地址指针,指向下一个单元
        DJNZ  LEN,LOOP     ;未完继续
        SJMP  $
        END
```

阅读上一页的程序，将PCNT，NCNT，ZCNT填到正确的位置，完成数组中正数负数和零的个数统计。

(1) [填空1]

(2) [填空2]

(3) [填空3]

作答

正常使用填空题需3.0以上版本雨课堂

答案:

```
          BLOCK   EQU   2000H        ;定义数据块首址
          LEN     EQU   30H          ;定义长度计数单元
          PCNT    EQU   31H          ;正计数单元, positive count
          NCNT    EQU   32H          ;负计数单元, negative count
          ZCNT    EQU   33H          ;零计数单元, zero count
          ORG     0000H
          JMP     START
          ORG     0200H
START:    MOV     DPTR,#BLOCK        ;地址指针指向数据块首址
          MOV     PCNT, #0
          MOV     NCNT, #0           ;计数单元清0
          MOV     ZCNT, #0
LOOP:     MOVX    A, @DPTR           ;取一个数
          JNB     ACC.7,L1
          INC     NCNT               ;ACC负数
          SJMP    NEXT
L1:       JNZ     L2
          INC     ZCNT               ;ACC是0
          SJMP    NEXT
L2:       INC     PCNT               ;ACC是正数
NEXT:     INC     DPTR               ;修正地址指针,指向下一个单元
          DJNZ    LEN,LOOP           ;未完继续
          SJMP    $
          END
```

THE END

THANK YOU