第三章　AVR 单片机指令与汇编系统

传统的 8 位单片机（如最典型的 8051 结构的单片机）大都采用复杂指令 CISC(Complex Instruction Set Computer) 系统体系。由于 CISC 结构存在指令系统不等长，指令数多，CPU 利用效率低，执行速度慢等缺陷，已不能满足和适应设计高档电子产品和嵌入式系统应用的需要。

作为 8 位的 AVR 单片机来讲，除了其具备比较完善和功能强大的硬件结构和组成外，其更重要的是它的内核和指令系统为先进的 RISC 体系结构，采用了大型快速存取寄存器组（32 个通用工作寄存器）、快速的单周期指令系统以及单级流水线等先进技术。因此，AVR 内核核指令系统的显著特点有：

1. 16/32 位定长指令

AVR 的一个指令字为 16 位或 32 位，其中大部分的指令为 16 位。采用定长指令，不仅使取指操作简单，提高了取指令的速度；同时也降低了在取指操作过程中的错误，提高了系统的可靠性。

2. 流水线操作

AVR 采用流水线技术，在前一条指令执行的时候，就取出现行的指令，然后以一个周期执行指令。大大提高了 CPU 的运行速度。

3. 大型快速存取寄存器组

传统的基于累加器的结构单片机（如 8051），需要大量的程序代码来完成和实现在累加器和存储器之间的数据传送。而在 AVR 单片机中，采用 32 个通用工作寄存器构成大型快速存取寄存器组，用 32 个通用工作寄存器代替了累加器（相当有 32 个累加器），从而避免了传统结构中累加器和存储器之间数据传送造成的瓶颈现象。

由于 AVR 单片机采用 RISC 结构，使得它具有高达 1MIPS / MHz 的高速运行处理能力。同时也能更好地适合采用高级语言（例如 C 语言、BASIC 语言）来编写系统程序，高效地开发出目标代码，以加快产品进入市场的时间和简化系统的设计、开发、维护和支持。

3.1　ATmega16 指令综述

指令是 CPU 用于控制各功能部件完成某一指定动作或操作的指示和命令。指令不同，CPU 和各个功能部件完成的动作也不一样，指令的功能也不同。程序员根据系统的要求，选用不同功能指令的有序组合就构成的程序。CPU 执行不同的程序，就能完成不同的任务。

CPU 指令的的集合或全体称为指令系统。指令系统是 CPU 的重要性能指标，也是学习以及使用单片机的重要内容。由于 CPU 结构的不同，每一种 CPU 的指令和功能也不同，因此学习 AVR，也必须了解它的指令结构、功能和特点。只有在此基础上，才能更清楚的了解 AVR 的硬件使用，编写出好的系统程序。

AVR 单片机指令系统是 RISC 结构的精简指令集，是一种简明、易掌握、效率高的指令系统。ATmega16 单片机完全兼容 AVR 的指令系统，具有高性能的数据处理能力，能对位、半字节、字节和双字节数据进行各种操作，包括算术和逻辑运算、数据传送、布尔处理、控制转移和硬件乘法等操作。

ATmega16 共有 131 条指令，按功能可分为五大类，它们是：

- 算术和逻辑运算指令（28 条）；
- 比较和跳转指令（36 条）；
- 数据传送指令（35 条）；
- 位操作和位测试指令（28 条）；
- MCU 控制指令（4 条）。

在附录 A 中列出了 ATmega16 全部的 131 条指令，包括字节数、功能、对标志位的影响以及执行周期数等。

3.1.1 指令格式以及三种表示方式

指令格式是指指令码的结构形式。通常，指令可分为操作码和操作数两部分。其中操作码部分比较简单，操作数部分则比较复杂，而且随 CPU 类型的不同，寻址方式的不同有较大的变化。

AVR 的指令的一般格式为：

操作码	第 1 操作数或操作数地址	第 2 操作数或操作数地址

其中，**操作码**用于指示 CPU 执行何种操作，是加法操作还是减法操作，是数据传送还是数据移位等。**第 1 操作数或操作数地址**用于表示参与操作的第 1 个操作数，或该操作数在内存的地址，同时该地址也将作为操作结果存放的地址。**第 2 操作数或操作数地址**（如果有的话）用于表示参与操作的第 2 个操作数，或该操作数在内存的地址。需要注意的是，在 AVR 的指令中，有相当一部分只有操作码，或只有操作码和第 1 操作数或操作数地址，前者在操作码中隐含了操作数或操作数的地址。

指令的表示方式是指采用何种形式描述指令，也是人们用于编写和阅读程序的基础。通常指令采用二进制、十六进制和助记符三种表示方式。

指令的二进制表示形式是一种可以直接为 CPU 识别和执行的形式，故称为指令的机器码或汇编语言的目标代码，下载到 AVR 中的代码必须是可执行的目标代码。但二进制形式的代码具有难读、难写、难记忆和难修改等缺点，因此人们通常不用它来编写程序。

指令的十六进制形式是二进制形式的变型，只是将二进制代码 4 位一组用十六进制的形式描述。十六进制的形式虽然比二进制形式读写方便些，但还是不易被人们识别和修改，所以通常也不被用于编写程序，只是在某些场合，如调试程序、修改调整个别指令代码时作为输入程序的辅助手段。

指令的助记符形式又称为指令的汇编形式或汇编语句，是一种用英文单词或缩写字母以及数字来表征指令功能的形式。这种形式不仅容易为人们识别和读写，也方便记忆和交流，因此也是人们用于进行程序设计的一种常用的形式。在附录 A 中列出了 ATmega16 全部的指令，就是采用指令的助记符形式描述的。

由于 CPU 可以直接识别和执行的指令形式必须是二进制表示形式的，因此不管使用十六进制的形式还是汇编形式构成的程序，都需要通过人工或机器把它们翻译成二进制机器码的形式，才能下载到芯片中被 CPU 执行。

现在决大多数单片机都提供相应的，能够在 PC 上工作的的开发平台，其最基本的功能就是提供用户编写汇编代码的源程序，并能将汇编源程序翻译成二进制的机器码，生成可下载的目标代码文件。

3.1.2 AVR 指令系统中使用的符号

ATmega16 指令系统中所有的 131 条指令如附录 A 所列,给出了全部指令的汇编助记符、操作数、操作说明、相应的操作、操作数的范围、对标志位的影响、以及指令的执行周期。在指令表中，除了操作码采用了助记符表示外，还在指令的操作数的描述说明中采用了一些符号代码，下面对所使用符号的意义进行简单的说明。

1．状态寄存器与标志位

SREG： 8 位状态寄存器，其中每一位的定义为：

C： 进位标志位

Z： 结果为零标志位

N： 结果为负数标志位

V： 2 的补码溢出标志位

S： N⊕V，用于符号测试的标志位

H： 操作中产生半进位的标志位

T： 用于和 BLD、BST 指令进行位数据交换的位

I： 全局中断触发/禁止标志位

2．寄存器和操作码

Rd： 目的（或源）寄存器，取值为 R0～R31 或 R16～R31（取决于指令）。

Rr： 源寄存器，取值为 R0～R31。

A： I/O 寄存器，取值为 0～63 或 0～31（取决于指令）。

b： I/O 寄存器中的指定位，常数（0～7）。

s： 状态寄存器 SREG 中的指定位，常数（0～7）。

K： 立即数，常数（0～255）。

k： 地址常数，取值范围取决于指令。

q： 地址偏移量常数（0～63）。

X、Y、Z：地址指针寄存器（X=R27:R26；Y=R29:R28；Z=R31:R30）。

3．堆栈

STACK： 作为返回地址和压栈寄存器的堆栈

SP： 堆栈 STACK 的指针

3.1.3 AVR 指令的寻址方式和寻址空间

指令的一个重要组成部分是操作数。指令给出参与运算数据的方式称为寻址方式。CPU

执行指令时，首先要根据地址获取参加操作的操作数，然后才能对操作数进行操作，操作的结果还要根据地址保存在相应的储存器或寄存器中。因此 CPU 执行程序实际上是不断的寻找操作数并进行操作的过程。通常，指令的寻址方式有多种，寻址方式越多，指令的功能也就越强。

AVR 单片机指令操作数的寻址方式有以下几种：单寄存器直接寻址、双寄存器直接寻址、I/O 寄存器直接寻址、数据存储器直接寻址、数据存储器间接寻址、带后增量的数据存储器间接寻址、带预减量的数据存储器间接寻址、带位移的数据存储器间接寻址、程序存储器取常量寻址、程序存储器空间直接寻址、程序存储器空间间接寻址、程序相对寻址等。

1. 单寄存器直接寻址（图 3-1）

由指令指定一个寄存器的内容作为操作数，在指令中给出寄存器的直接地址，这种寻址方式称为单寄存器直接寻址。单寄存器寻址的地址范围限制为通用工作寄存器组中的 32 个寄存器 R0～R31，或后 16 个寄存器 R16～R31（取决于不同指令）。

例：INC Rd；操作：Rd←Rd+1。

INC R5；将寄存器 R5 内容加 1 回放。

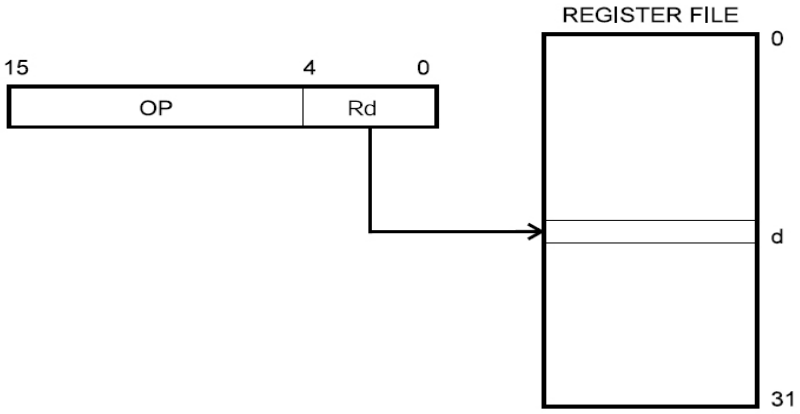

图 3-1 单寄存器直接寻址

2. 双寄存器直接寻址（图 3-2）

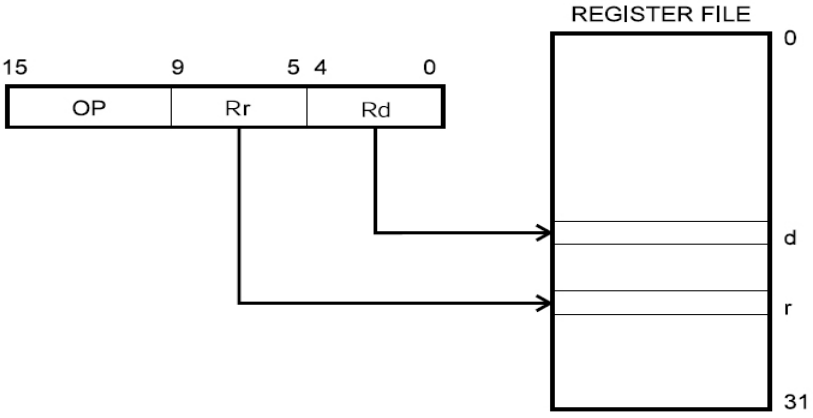

图 3-2 双寄存器直接寻址

双寄存器直接寻址方式与单寄存器直接寻址方式相似，它是将指令指出的两个寄存器 Rd 和 Rr 的内容作为操作数，而结果存放在 Rd 寄存器中。指令中同时给出两个寄存器的直接地址，这种寻址方式称为双寄存器直接寻址。双寄存器寻址的地址范围限制为通用工作寄存器组中的 32 个寄存器 R0～R31，或后 16 个寄存器 R16～R31，或后 8 个寄存器 R16～R23（取决于不同指令）。

例：ADD Rd，Rr；操作：Rd←Rd+Rr。

ADD R0，R1；将 R0 和 R1 寄存器内容相加，结果回放 R0。

3．I/O 寄存器直接寻址（图 3-3）

由指令指定一个 I/O 寄存器的内容作为操作数。在指令中直接给出 I/O 寄存器的地址，这种寻址方式称为 I/O 寄存器直接寻址。I/O 寄存器直接寻址的地址使用 I/O 寄存器空间的地址$00～$3F，共 64 个，取值为 0～63 或 0～31（取决于指令）。

例：IN Rd，P；操作：Rd←P。

IN R5，$3E；读 I/O 空间地址为$3E 寄存器（SPH）的内容，放入寄存器 R5。

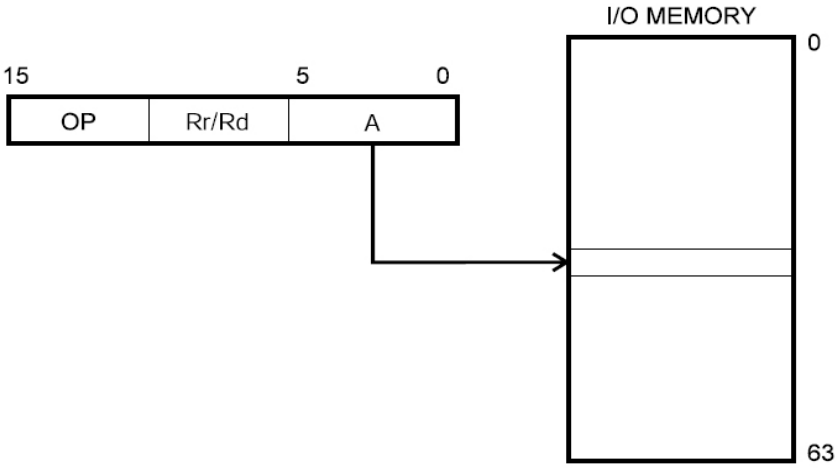

图 3-3 I/O 寄存器直接寻址

4．数据存储器空间直接寻址（图 3-4）

数据存储器空间直接寻址方式用于直接 CPU 从 SRAM 存储器中存取数据。数据存储器空间直接寻址为双字节指令，在指令的低字节中指出一个 16 位的 SRAM 地址。

例：LDS Rd，K；操作：Rd←（K）。

LDS R18，$100；读地址为$100 的 SRAM 中内容，传送到 R18 中。

指令中 16 位 SRAM 的地址字长度限定了 SRAM 的地址空间为 64K 字节，该地址空间实际包含了 32 个通用寄存器和 64 个 I/O 寄存器。因此，也可使用数据存储器空间直接寻址的方式读取通用寄存器或 I/O 寄存器中的内容，此时应使用这些寄存器在 SRAM 空间的映射地址，而且其效率也比使用寄存器直接寻址的方式要低。原因在于数据存储器空间直接寻址的指令为双字节指令，指令周期为 2 个系统时钟。

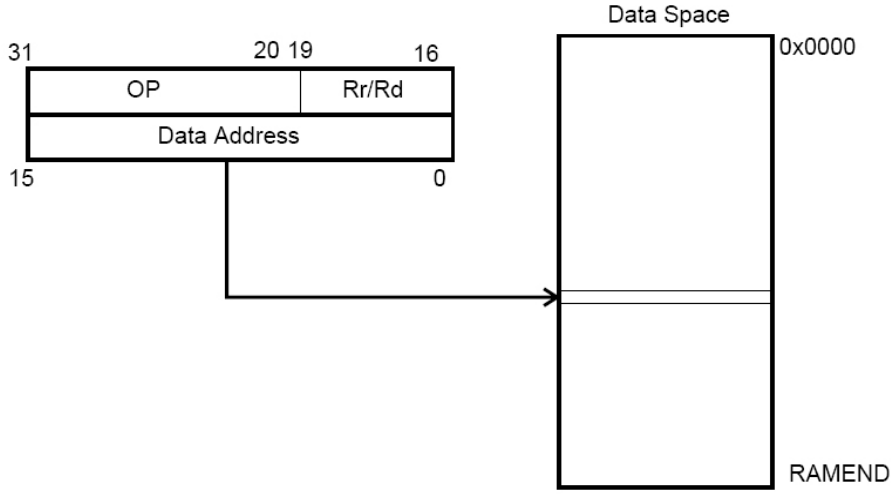

图 3-4 数据存储器空间直接寻址

5．数据存储器空间的寄存器间接寻址（图 3-5）

由指令指定某一个 16 位寄存器的内容作为操作数在 SRAM 中的地址，该寻址方式称为数据存储器空间的寄存器间接寻址。AVR 单片机中使用 16 位寄存器 X、Y 或 Z 作为规定的地址指针寄存器，因此操作数的 SRAM 地址在间址寄存器 X、Y 或 Z 中。

例：LD Rd，Y；操作：Rd ←（Y），把以 Y 为指针的 SRAM 的内容送 Rd。

LD R16，Y；设 Y=$0567，即把 SRAM 地址为$0567 的内容传送到 R16 中。

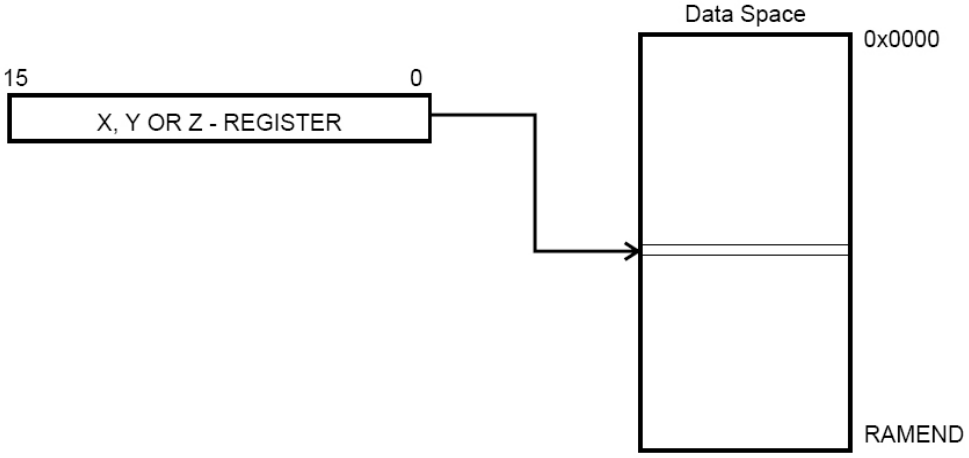

图 3-5 数据存储器空间的寄存器间接寻址

6．带后增量的数据存储器空间的寄存器间接寻址（图 3-6）

这种寻址方式类似于数据存储器空间的寄存器间接寻址方式，间址寄存器 X、Y、Z 中的内容仍为操作数在 SRAM 空间的地址，但指令在间接寻址操作后，再自动把间址寄存器中的内容加 1。这种寻址方式特别适用于访问矩阵、查表等应用。

例：LD Rd，Y+；操作：Rd←（Y），Y=Y+1，先把以 Y 为指针的 SRAM 的内容送 Rd，再把 Y 增 1。

LD R16，Y+；设原 Y=$0567，指令把 SRAM 地址为$0567 的内容传送到 R16 中，再将 Y 的值加 1，操作完成后 Y=$0568。

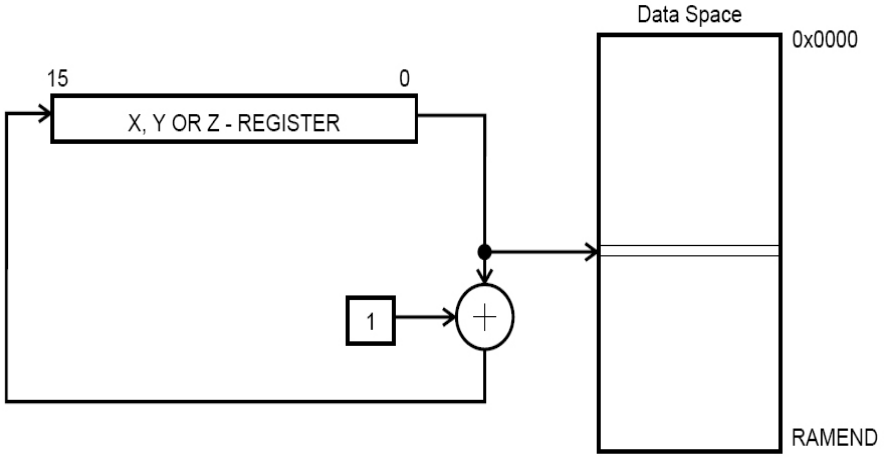

图 3-6 带后增量的数据存储器空间的寄存器间接寻址

7．带预减量的数据存储器空间寄存器间接寻址（图 3-7）

这种寻址方式类似于数据存储器空间的寄存器间接寻址方式，间址寄存器 X、Y、Z 中的内容仍为操作数在 SRAM 空间的地址，但指令在间接寻址操作之前，先自动将间址寄存器中的内容减 1，然后把减 1 后的内容作为操作数在 SRAM 空间的地址。这种寻址方式也特别适用于访问矩阵、查表等应用。

例：LD Rd，-Y；操作：Y=Y-1，Rd←（Y），先把 Y 减 1，再把以 Y 为指针的 SRAM 的内容送 Rd。

LD R16，-Y；设原 Y=$0567，指令即先把 Y 减 1，Y=$0566，再把 SRAM 地址为$0566 的内容传送到 R16 中。

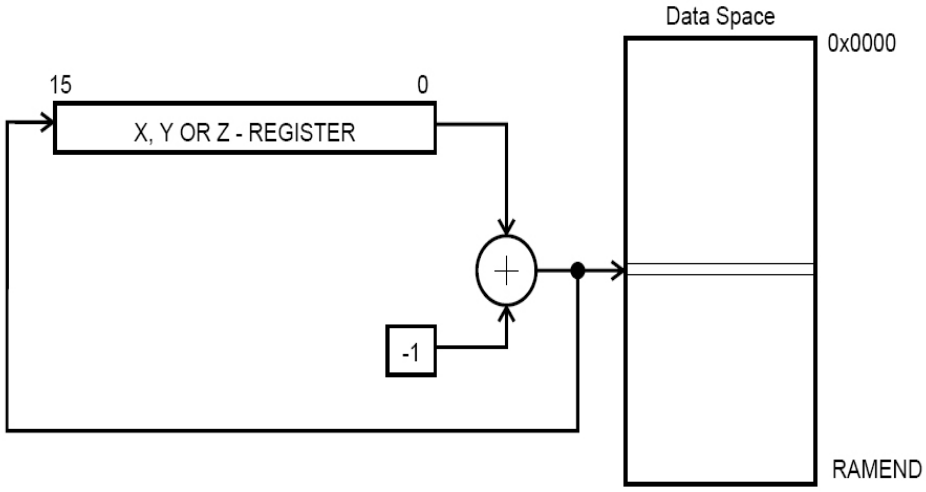

图 3-7 带预减量的数据存储器空间寄存器间接寻址

8．带位移的数据存储器空间寄存器间接寻址（图 3-8）

带位移的数据存储器空间寄存器间接寻址方式是：由间址寄存器（Y 或 Z）中的内容和指令字中给出的地址偏移量共同决定操作数在 SRAM 空间的地址，偏移量的范围为 0～63。

例：LDD Rd，Y+q；操作：Rd←（Y+q），其中 0≤q≤63，即把以 Y+q 为地址的 SRAM 的内容送 Rd，而 Y 寄存器的内容不变。

LDD R16，Y+31；设 Y=$0567，把 SRAM 地址为$0598 的内容传送到 R16 中，Y 寄存器的内容不变。

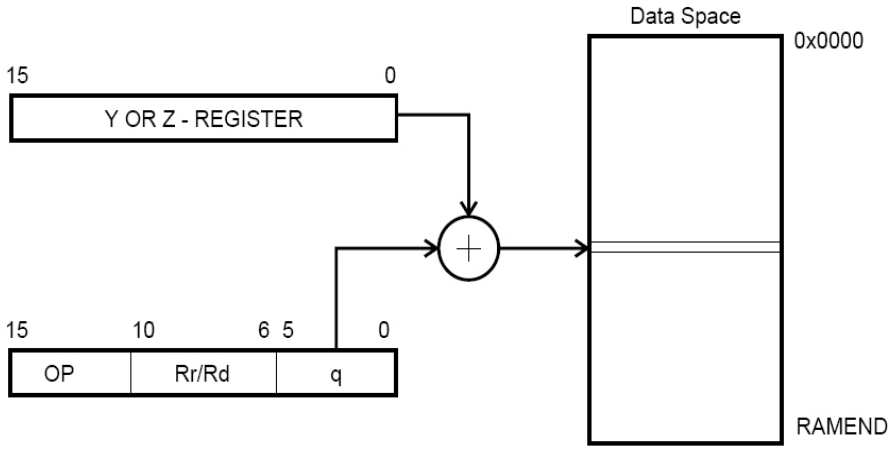

图 3-8 带位移的数据存储器空间寄存器间接寻址

9．程序存储器空间取常量寻址（图 3-9）

程序存储器空间取常量寻址主要从程序存储器 Flash 中读取常量，此种寻址方式只用于指令 LPM。程序存储器中常量字节的地址由地址寄存器 Z 的内容确定。Z 寄存器的高 15 位用于选择字地址（程序存储器的存储单元为字），而 Z 寄存器的最低位 Z（d0）用于确定字地址的高/低字节。若 d0=0，则选择字的低字节；d0=1，则选择字的高字节。

例：LPM；操作：R0 ←（Z），即把以 Z 为指针的程序存储器的内容送 R0。

若 Z=$0100，即把地址为$0080 的程序存储器的低字节内容送 R0。

若 Z=$0101，即把地址为$0080 的程序存储器的高字节内容送 R0。

例：LPM R16，Z；操作：R16←（Z），即把以 Z 为指针的程序存储器的内容送 R16。

若 Z=$0100，即把地址为$0080 的程序存储器的低字节内容送 R16。

若 Z=$0101，即把地址为$0080 的程序存储器的高字节内容送 R16。

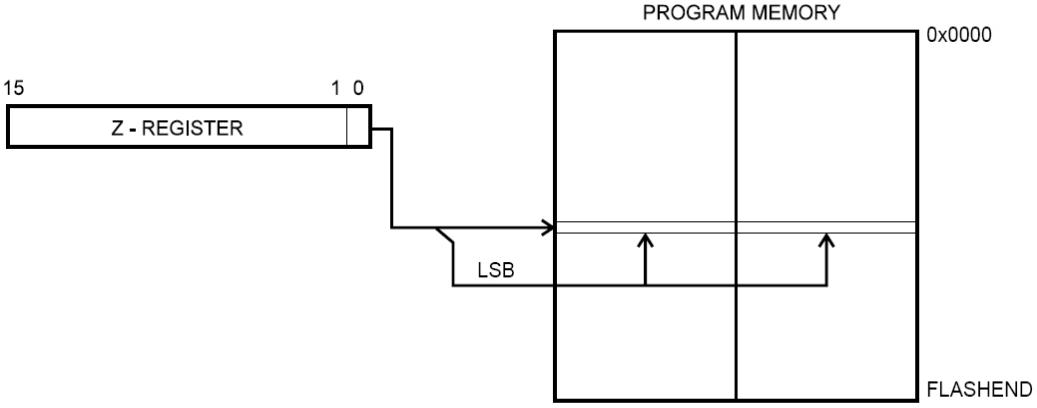

图 3-9 程序存储器空间取常量寻址

10．带后增量的程序存储器空间取常量寻址（图 3-10）

带后增量的程序存储器空间取常量寻址主要从程序存储器 Flash 中取常量，此种寻址方式只用于指令 LPM Rd，Z+。程序存储器中常量字节的地址由地址寄存器 Z 的内容确定。Z 寄存器的高 15 位用于选择字地址（程序存储器的存储单元为字），而 Z 寄存器的最低位 Z

（d0）用于确定字地址的高/低字节。若 d0=0，则选择字的低字节；d0=1，则选择字的高字节。寻址操作后，Z 寄存器的内容加 1。

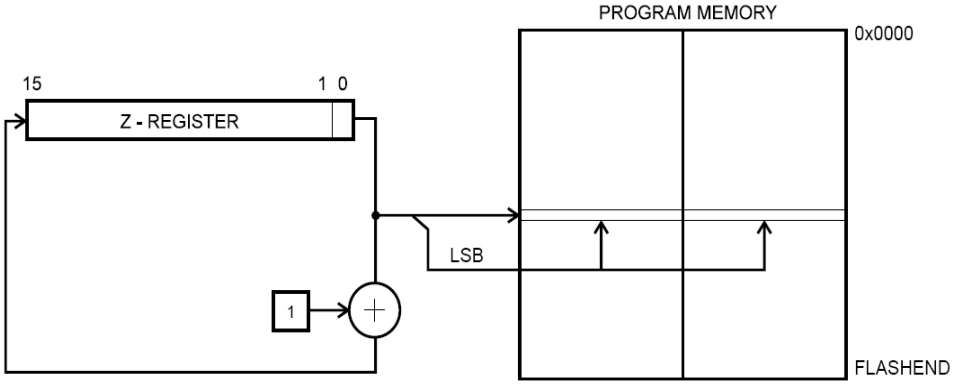

图 3-10 带后增量的程序存储器空间取常量寻址

例：LPM R16，Z+；操作：R16←（Z）；Z←Z+1，即把以 Z 为指针的程序存储器的内容送 R16，然后 Z 的内容加 1。

若 Z=$0100,即把地址为$0080 的程序存储器的低字节内容送 R16,完成后 Z=$0101。

若 Z=$0101,即把地址为$0080 的程序存储器的高字节内容送 R16,完成后 Z=$0102。

11．程序存储器空间写数据寻址（图 3-9）

程序存储器空间写数据寻址主要用于可进行在系统自编程的 AVR 单片机，此种寻址方式只用于指令 SPM。该指令将寄存器 R1 和 R0 中的内容组成一个字 R1:R0，然后写入由 Z 寄存器的内容作为地址（Z 寄存器的最低位必须为 0）的程序存储器单元中。（注意：实际是写入到 Flash 的页缓冲区中）。

例：SPM；操作：（Z）←R1:R0，把 R1:R0 内容写入以 Z 为指针的程序存储器单元。

12．程序存储器空间直接寻址（图 3-11）

程序存储器空间直接寻址方式用于程序的无条件跳转指令 JMP、CALL。指令中含有一个 16 位的操作数，指令将操作数存入程序计数器 PC 中，作为下一条要执行指令在程序存储器空间的地址。JMP 类指令和 CALL 类指令的寻址方式相同，但 CALL 类的指令还包括了返回地址的压进堆栈和堆栈指针寄存器 SP 内容减 2 的操作。

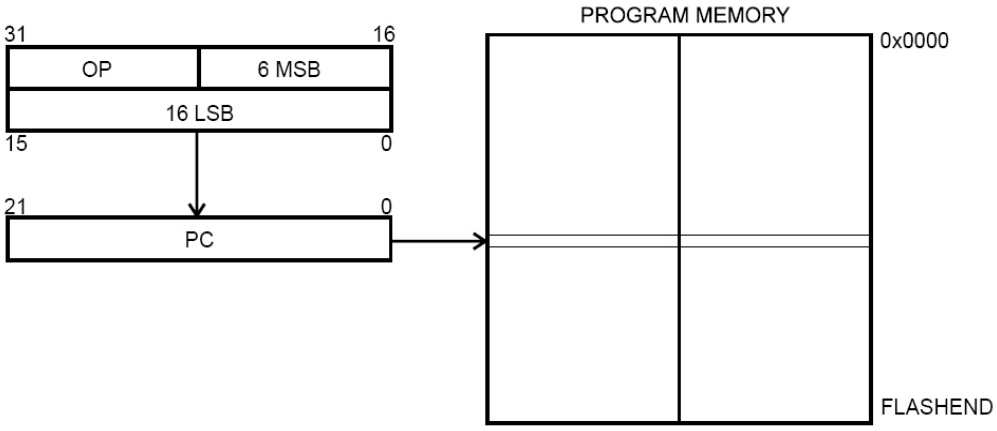

图 3-11 程序存储器空间直接寻址

例：JMP $0100；操作：PC←$0100。程序计数器 PC 的值设置为$0100，接下来执行程序存储器$0100 单元的指令代码。

例：CALL $0100；操作：STACK←PC+2；SP←SP-2；PC←$0100。先将程序计数器 PC 的当前值加 2 后压进堆栈（CALL 指令为 2 个字长），堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0100，接下来执行程序存储器$0100 单元的指令代码。

13．程序存储器空间 Z 寄存器间接寻址（图 3-12）

程序存储器空间间接寻址方式是使用 Z 寄存器存放下一步要执行指令代码程序地址，程序转到 Z 寄存器内容所指定程序存储器的地址处继续执行，即用寄存器 Z 的内容代替 PC 的值。此寻址方式用于 IJMP、ICALL 指令。

例：IJMP；操作：PC←Z，即把 Z 的内容送程序计数器 PC。若 Z=$0100，即把$0100 送程序计数器 PC，接下来执行程序存储器$0100 单元的指令代码。

例：ICALL；操作：STACK←PC+1；SP←SP-2；PC←Z。若 Z=$0100，先将程序计数器 PC 的当前值加 1 后压进堆栈，堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0100，接下来执行程序存储器$0100 单元的指令代码。

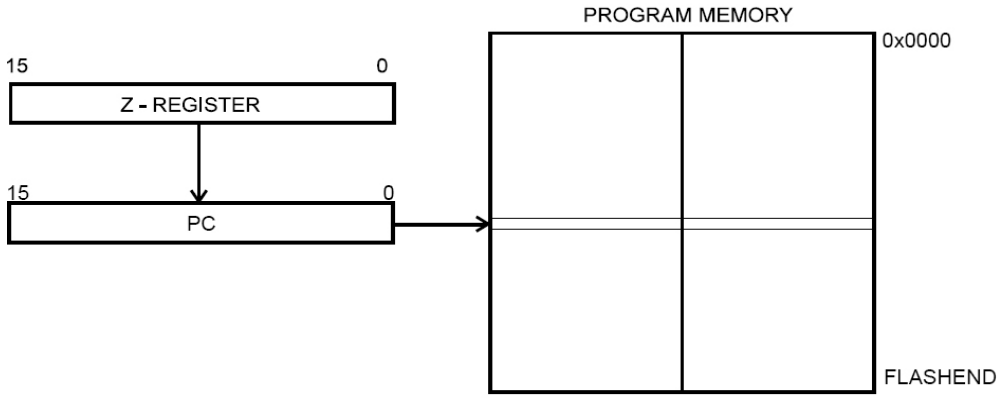

图 3-12 程序存储器空间 Z 寄存器间接寻址

14．程序存储器空间相对寻址（图 3-13）

在程序存储器空间相对寻址方式中，在指令中包含一个相对偏移量 k，指令执行时，首先将当前程序计数器 PC 值加 1 后再与偏移量 k 相加，作为程序下一条要执行指令的地址。此寻址方式用于 RJMP、RCALL 指令。

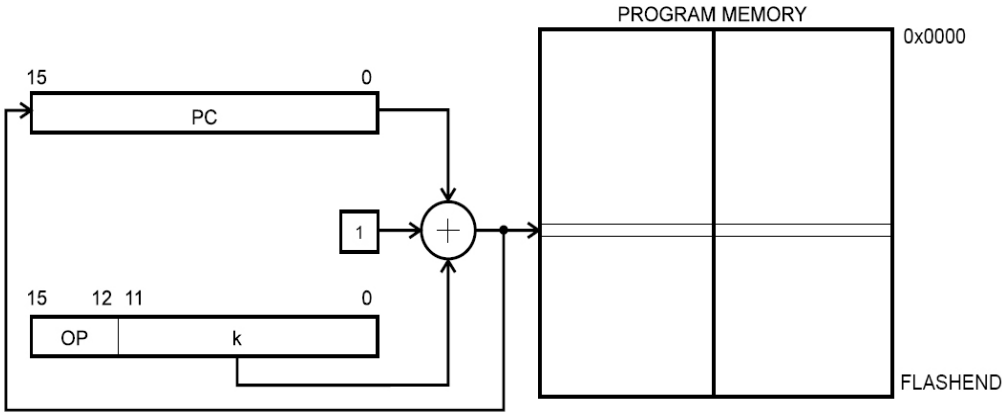

图 3-13 程序存储器空间相对寻址

例：RJMP $0100；操作：PC←PC+1+$0100。若当前指令地址为$0200（PC=$0200），即把$0301 送程序计数器 PC，接下来执行程序存储器$0301 单元的指令代码。

例：RCALL $0100；操作：STACK←PC+1；SP←SP-2；PC←PC+1+$0100。若当前指令地址为$0200（PC=$0200），先将程序计数器 PC 的当前值加 1 后压进堆栈，堆栈指针计数器 SP 内容减 2，然后 PC 的值为$0301，接下来执行程序存储器$0301 单元的指令代码。

15. 数据存储器空间堆栈寄存器 SP 间接寻址（图 3-14）

数据存储器空间堆栈寄存器 SP 间接寻址，是将 16 位的堆栈寄存器 SP 的内容作为操作数在 SRAM 空间的地址，此寻址方式用于 PUSH、POP 指令。

例：PUSH R0；操作：STACK←R0；SP←SP-1。若当前 SP=$10FF，先把寄存器 R0 的内容送到 SRAM 的$10FF 单元，再将 SP 内容减 1，即 SP=$10FE。

例：POP R1；操作：SP←SP+1；R1←STACK。若当前 SP=$10FE，先将 SP 内容加 1，再把 SRAM 的$10FF 单元内容送到寄存器 R1，此时 SP=$10FF。

此外，在 CPU 响应中断和执行 CALL 一类的子程序调用指令（SP = SP-2），以及执行中断返回 IRET 和子程序返回 RET 一类的子程序返回指令中（SP = SP+2），都隐含着使用堆栈寄存器 SP 间接寻址的方式。

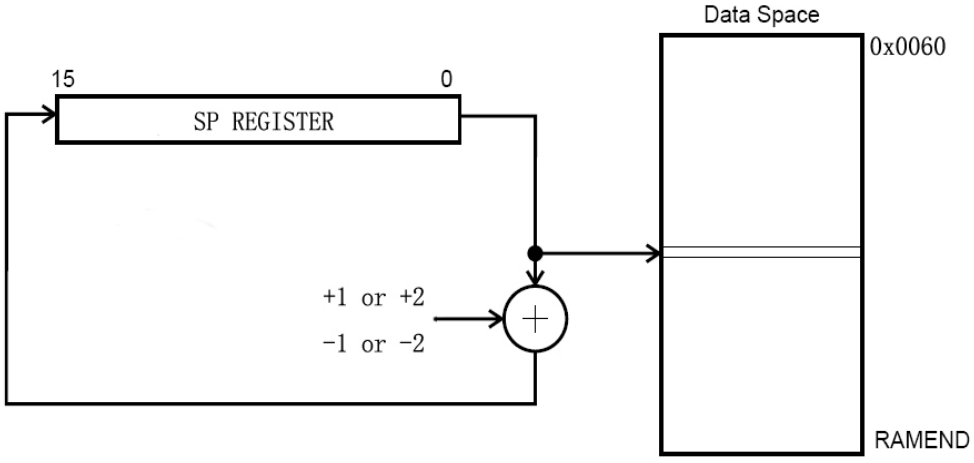

图 3-14 数据存储器空间堆栈寄存器 SP 间接寻址

3.1.4 AVR 指令操作结果对标志位的影响

在AVR的指令中，一类指令执行后要影响到状态寄存器中某些标志位的状态，即不论指令执行前标志位状态如何，指令执行后总是根据执行结果的定义形成新的状态标志。另一类指令在执行后不会影响标志位，标志位原来什么状态，指令执行后标志状态还是保持不变。

由于AVR的CPU响应中断时，硬件不保护状态寄存器，所以在编写中断服务处理程序时，应注意将状态寄存器进行保护（如用PUSH指令压入到堆栈），在中断返回前还要恢复状态寄存器在进入中断前时的标志状态（如用POP指令从堆栈中弹出）。

3.2　算术和逻辑指令

AVR 的算术运算指令有加、减、乘法、取反、取补、比较指令、增量和减量指令。逻辑运算指令有与、或和异或指令等。

3.2.1　加法指令

1．不带进位位加法

ADD Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器不带进位 C 标志相加，结果送目的寄存器 Rd。

操作：Rd←Rd+Rr　　PC←PC+1　　机器码：0000 11rd dddd rrrr

对标志位的影响：H S V N Z C

2．带进位位加法

ADC Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器和 C 标志的内容相加，结果送目的寄存器 Rd。

操作：Rd←Rd+Rr+C　PC←PC+1　　机器码：0001 11rd dddd rrrr

对标志位的影响：H S V N Z C

3．字加立即数

ADIW Rdl,K　　dl 为：24、26、28、30，0≤K≤63

说明：寄存器对（一个字）同立即数（0～63）相加，结果放到寄存器对。

操作：Rdh:Rdl←Rdh:Rdl+K　PC←PC+1　　机器码：1001 0110 KKdd KKKK

对标志位的影响：S V N Z C

注意：dl 只能取 24、26、28、30，即仅用于最后 4 个寄存器对。K 为 6 位二进制无符号数（0～63）。

4．增 1 指令

INC Rd　　0≤d≤31

说明：寄存器 Rd 的内容加 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 INC 指令允许在多倍字长计算中用作循环计数。当对无符号数操作时，仅有 BREQ（相等跳转）和 BRNE（不为零跳转）指令有效。当对二进制补码值操作时，所有的带符号跳转指令都有效。

操作：Rd←Rd+1　　PC←PC+1　　机器码：1001 010d dddd 0011

对标志位的影响：S V N Z

3.2.2　减法指令

1．不带进位位减法

SUB Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器相减结果送目的寄存器 Rd 中。

操作：Rd←Rd-Rr　　PC←PC+1　　机器码：0001 10rd dddd rrrr

对标志位的影响：H S V N Z C

2．减立即数（字节）

SUBI Rd，K　　16≤d≤31，0≤K≤255

说明：一个寄存器和常数相减，结果送目的寄存器 Rd。该指令工作于寄存器 R16～R31 之间，非常适合 X、Y 和 Z 指针的操作。

操作：Rd←Rd-K　　PC←PC+1　　机器码：0101 KKKK dddd KKKK

对标志位的影响：H S V N Z C

3．带进位位减法

SBC Rd，Rr　　0≤d≤31，0≤r≤31

说明：两个寄存器带着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：Rd←Rd-Rr-C　PC←PC+1　　机器码：0000 10rd dddd rrrr

对标志位的影响：H S V N Z C

4．带进位位减立即数（字节）

SBCI Rd，K　　16≤d≤31，0≤K≤255

说明：寄存器和立即数带着 C 标志相减，结果放到目的寄存器 Rd 中。

操作：Rd←Rd-K-C　PC←PC+1　　机器码：0100 KKKK dddd KKKK

对标志位的影响：H S V N Z C

5．减立即数（字）

SBIW Rdl，K　　dl 为 24、26、28、30，0≤K≤63

说明：寄存器对（字）与立即数 0～63 相减，结果放入寄存器对。

操作：Rdh:Rdl←Rdh:Rdl-K　PC←PC+1　机器码：1001 0111 KKdd KKKK

注意：dl 只能取 24、26、28、30，即仅用于最后 4 个寄存器对。K 为 6 位二进制无符号数（0～63）。

对标志位的影响：S V N Z C

6．减 1 指令

DEC Rd　　0≤d≤31

说明：寄存器 Rd 的内容减 1，结果送目的寄存器 Rd 中。该操作不改变 SREG 中的 C 标志，所以 DEC 指令允许在多倍字长计算中用作循环计数。当对无符号值操作时，仅有 BREQ（不相等跳转）和 BRNE（不为零跳转）指令有效。当对二进制补码值操作时，所有的带符号跳转指令都有效。

操作：Rd←Rd-1　PC←PC+1　　机器码：1001 010d dddd 1010

对标志位的影响：S V N Z

3.2.3　取反码指令

COM Rd　　0≤d≤31

说明：该指令完成对寄存器 Rd 的二进制反码操作。

操作：Rd←$FF-Rd　PC←PC+1　　机器码：1001 010d dddd 0000

对标志位的影响：S N Z V(0) C(1)

3.2.4 取补码指令

NEG Rd　　0≤d≤31

说明：寄存器 Rd 的内容转换成二进制补码值。

操作：Rd←$00-Rd　　PC←PC+1　　机器码：1001 010d dddd 0001

对标志位的影响：H S V N Z C

3.2.5 比较指令

1．寄存器比较

CP Rd，Rr　　0≤d≤31，0≤r≤31

说明：该指令完成两个寄存器 Rd 和 Rr 相比较操作，而寄存器的内容不改变，该指令后能使用所有条件跳转指令。

操作：Rd-Rr　　PC←PC+1　　机器码：0001 01rd dddd rrrr

对标志位的影响：H S V N Z C

2．带进位比较

CPC Rd，Rr　0≤d≤31，　0≤r≤31

说明：该指令完成寄存器 Rd 的值和寄存器 Rr 加 C 相比较操作，而寄存器的内容不改变，该指令后能使用所有条件跳转指令。

操作：Rd-Rr-C　PC←PC+1　　机器码：0000 01rd dddd rrrr

对标志位的影响：H S V N Z C

3．与立即数（字节）比较

CPI Rd，K　　16≤d≤31，0≤K≤255

说明：该指令完成寄存器 Rd 和常数的比较操作，寄存器的内容不改变，该指令后能使用所有条件跳转指令。

操作：Rd-K　　PC←PC+1　　机器码：0011 KKKK dddd KKKK

对标志位的影响：H S V N Z C

3.2.6 逻辑与指令

1．寄存器逻辑与

AND Rd，Rr　　0≤d≤31，0≤d≤31

说明：寄存器 Rd 和寄存器 Rr 的内容逻辑与，结果送目的寄存器 Rd。

应用：清 0 某位，用 0 去同该位逻辑与；保留某位值，用 1 去逻辑与；代替硬件与门。

操作 Rd←Rd・Rr　　PC←PC+1　　机器码：0010 00rd dddd rrrr

对标志位的影响：S V(0) N Z

2．与立即数（字节）

ANDI Rd，K　　16≤d≤31，0≤K≤255

说明：寄存器 Rd 的内容与常数逻辑与，结果送目的寄存器 Rd。

应用：清 0 某位时，用 0 去同该位逻辑与；保留某位的值，用 1 去逻辑与；代替硬件

与门。

操作：Rd←Rd • K　　PC←PC+1　　机器码：0111 KKKK dddd KKKK

对标志位的影响：S V(0) N Z

3．寄存器位清零

CBR Rd，K　　16≤d≤31，0≤K≤255

说明：清除寄存器 Rd 中的指定位，利用寄存器 Rd 的内容与常数表征码 K 的补码相与，其结果放在寄存器 Rd 中。

操作：Rd←Rd •（$FF-K）　PC←PC+1　　机器码：0111 ~~KKKK~~ dddd ~~KKKK~~（~~KKKK~~为KKKK的补码）

对标志位的影响：S V(0) N Z

4．测试寄存器为零或负

TST Rd　　0≤d≤31

说明：测试寄存器为零或负，实现寄存器内容自己同自己的逻辑与操作，而寄存器内容不改变。

操作：Rd←Rd • Rd　　PC←PC+1　　机器码：0010 00dd dddd dddd

对标志位的影响：S V(0) N Z

3.2.7　逻辑或指令

1．寄存器逻辑或

OR Rd，Rr　　0≤d≤31，　0≤r≤31

应用：置数，使某位为 1，用 1 去或；保留，用 0 去逻辑或；代替硬件或门。

说明：完成寄存器 Rd 与寄存器 Rr 的内容逻辑或操作，结果送目的寄存器 Rd 中。

操作：Rd←Rd∨Rr　　PC←PC+1　　机器码：0010 10rd dddd rrrr

对标志位的影响：S V(0) N Z

2．或立即数(字节)

ORI Rd，K　　16≤d≤31，　0≤K≤255

说明：完成寄存器 Rd 的内容与常量 K 逻辑或操作，结果送目的寄存器 Rd 中。

操作：Rd←Rd∨K　　PC←PC+1　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S V(0) N Z

3．置寄存器位

SBR Rd，K　　16≤d≤31，　0≤K≤255

说明：用于对寄存器 Rd 中指定位置位。完成寄存器 Rd 和常数表征码 K 之间的逻辑或操作，结果送目的寄存器 Rd。

操作：Rd←Rd∨K　　PC←PC+1　　机器码：0110 KKKK dddd KKKK

对标志位的影响：S V(0) N Z

4．置寄存器为$FF

SER Rd　　16≤d≤31

说明：直接装入$FF 到寄存器 Rd。

操作：Rd←$FF　　PC←PC+1　　机器码：1110 1111 dddd 1111

对标志位的影响：无

3.2.8 逻辑异或指令

1．寄存器异或

EOR Rd，Rr　　0≤d≤31，　0≤r≤31

说明：完成寄存器 Rd 和寄存器 Rr 的内容逻辑异或操作，结果送目的寄存器 Rd。

操作：Rd←Rd⊕Rr　　PC←PC+1　　机器码：0010 01rd dddd rrrr

对标志位的影响：S V(0) N Z

2．寄存器清零

CLR Rd　　0≤d≤31

说明：寄存器清零。该指令采用寄存器 Rd 与自己的内容相异或实现的寄存器的所有位都被清零。

操作：Rd←Rd⊕Rd　　PC←PC+1　　机器码：0010 01dd dddd dddd

对标志位的影响：S(0) V(0) N(0) Z(1)

3.2.9 乘法指令

1．无符号数乘法

MUL Rd，Rr　　0≤d≤31，　0≤r≤31

说明：该指令完成将寄存器 Rd 和寄存器 Rr 的内容作为两个无符号 8 位数的乘法操作，结果为 16 位的无符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。如果操作数为寄存器 R1 或 R0，则结果会将原操作数覆盖。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：1001 11rd dddd rrrr

对标志位的影响：Z C

2．有符号数乘法

MULS Rd，Rr　　16≤d≤31，　16≤r≤31

说明：该指令完成将寄存器 Rd 和寄存器 Rr 的内容作为两个 8 位有符号数的乘法操作，结果为 16 位的有符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16～R31。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：0000 0010 dddd rrrr

对标志位的影响：Z C

3．有符号数与无符号数乘法

MULSU Rd，Rr　　16≤d≤23，　16≤r≤23

说明：该指令完成将寄存器 Rd（8 位，有符号数）和寄存器 Rr（8 位，无符号数）的内容相乘操作，结果为 16 位的有符号数，保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16～R23。

操作：R1:R0=Rd×Rr　　PC←PC+1　　机器码：0000 0011 0ddd 0rrr

对标志位的影响：Z C

4．无符号定点小数乘法

FMUL Rd，Rr　　16≤d≤23，　16≤r≤23

说明：该指令完成将寄存器 Rd（8 位无符号数）和寄存器 Rr（8 位无符号数）的内容相乘操作，结果为 16 位的无符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，

R0 为低 8 位。源操作数为寄存器 R16～R23。

操作：R1:R0=Rd×Rr （unsigned（1.15）=unsigened（1.7）×unsigened（1.7））

PC←PC+1 机器码：0000 0011 0ddd 1rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2).（q1+q2)）的结果。

对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMUL 指令的执行周期与 MUL 指令相同，但比 MUL 指令增加了左移操作。

被乘数 Rd 和乘数 Rr 是两个包含无符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位无符号定点小数，其小数点固定在第 15 位和第 14 位之间。

5．有符号定点小数乘法

FMULS Rd，Rr 16≤d≤23， 16≤r≤23

说明：该指令完成寄存器 Rd（8 位带符号数）和寄存器 Rr（8 位带符号数）的内容相乘操作，结果为 16 位的带符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16～R23。

操作：R1:R0=Rd×Rr （signed（1.15）=sigened（1.7）×sigened（1.7））

PC←PC+1 机器码：0000 0011 1ddd 0rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2).（q1+q2)）的结果。

对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMULS 指令的执行周期与 MULS 指令相同，但比 MULS 指令增加了左移操作。

被乘数 Rd 和乘数 Rr 是两个包含带符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位带符号的定点小数，其小数点固定在第 15 位和第 14 位之间。

6．有符号定点小数和无符号定点小数乘法

FMULSU Rd，Rr 16≤d≤23， 16≤r≤23

说明：该指令完成寄存器 Rd（8 位带符号数）和寄存器 Rr（8 位无符号数）的内容相乘操作，结果为 16 位的带符号数，并将结果左移一位后保存在 R1:R0 中，R1 为高 8 位，R0 为低 8 位。源操作数为寄存器 R16～R23。

操作：R1:R0=Rd×Rr （signed（1.15）=sigened（1.7）×unsigened（1.7））

PC←PC+1 机器码：0000 0011 1ddd 1rrr

对标志位的影响：Z C

注：（n.q）表示一个小数点左边有 n 个二进制数位、小数点右边有 q 个二进制数位的小数。以（n1.q1）和（n2.q2）为格式的两个小数相乘，产生格式为（(n1+n2).（q1+q2)）的结果。对于要有效保留小数位的处理应用，输入的数据通常采用（1.7）的格式，产生的结果为（2.14）格式。因此将结果左移一位，以使高字节的格式与输入的相一致。FMULSU 指令的执行周期与 MULSU 指令相同，但比 MULSU 指令增加了左移操作。

被乘数 Rd 为一个包含带符号定点小数的寄存器，乘数 Rr 是一个包含无符号定点小数的寄存器，小数点固定在第 7 位和第 6 位之间。结果为 16 位带符号的定点小数，其小数点固定在第 15 位和第 14 位之间。

3.3 跳 转 指 令

3.3.1 无条件跳转指令

1．相对跳转

RJMP k　　-2048≤k≤2047

说明：相对跳转到 PC-2048～PC+2047 字范围内的地址，在汇编程序中，用目的地址的标号替代相对跳转字 k。

操作：PC←（PC+1）+ k　　机器码：1100 kkkk kkkk kkkk

对标志位的影响：无

汇编语言中，只要使用欲转向的标号即可。

例：

```
        RJMP ABC
             ……
             ……
   ABC:      ……
```

2．间接跳转

IJMP

说明：间接跳转到 Z 指针寄存器指向的 16 位地址。Z 指针寄存器是 16 位宽，允许在当前程序存储器空间 64K 字（128K 字节）内跳转。

IJMP 间接跳转优点：跳转范围大；缺点：作为子程序模块，移植时需修改跳转地址，所以一般在子程序中不要使用。

操作：PC←Z（15～0）　　机器码：1001 0100 0000 1001

对标志位的影响：无

3．直接跳转

JMP k　　0≤k≤4194303

说明：直接跳转到 k 地址处，在汇编程序中，用目的地址的标号替代跳转字 k。

操作：PC←k　　机器码：1001 010k kkkk 110k kkkk kkkk kkkk kkkk

对标志位的影响：无

汇编语言中，只要使用欲转向的标号即可。

例：

```
    JMP ABC
    ……
ABC：……
```

3.3.2 条件跳转指令

条件跳转指令是依照某种特定的条件跳转的指令，条件满足则跳转；条件不满足时则顺序执行下面的指令。

1．测试条件符合跳转指令

（1）状态寄存器中位为“1”跳转

BRBS s,k　　0≤s≤7，-64≤k≤63

说明：执行该指令时，PC 先加 1，再测试 SREG 的 s 位，如果该位被置位，则跳转 k 个字，k 为 7 位带符号数，最多可向前跳 63 个字，向后跳 64 个字；否则顺序执行。在汇编程序中，用目的地址的标号替代相对跳转字 k。

操作：If SREG（s）=1，then PC←（PC+1）+k，else PC←PC+1

机器码：1111 00kk kkkk ksss

对标志位的影响：无

（2）状态寄存器中位为“0”跳转

BRBC s,k　　0≤s≤7，-64≤k≤63

说明：执行该指令时，PC 先加 1，再测试 SREG 的 s 位，如果该位被清零，则跳转 k 个字，k 为 7 位带符号数，最多可向前跳 63 个字，向后跳 64 个字；否则顺序执行。在汇编程序中，用目的地址的标号替代相对跳转字 k。

操作：If SREG（s）=0，then PC←（PC+1）+k，else PC←PC+1

机器码：1111 01kk kkkk ksss

对标志位的影响：无

（3）相等跳转

BREQ k　　-64≤k≤63

说明：条件相对跳转，测试零标志位 Z，如果 Z 位被置位，则相对 PC 值跳转 k 个字。如果在执行 CP、CPI、SUB 或 SUBI 指令后，立即执行该指令，且当寄存器 Rd 中数与寄存器 Rr 中数相等时，将发生跳转。这条指令相当于指令 BRBS 1，k。

操作：If Rd=Rr（Z=1），then PC←（PC+1）+k；else PC←PC+1

机器码：1111 00kk kkkk k001

对标志位的影响：无

（4）不相等跳转

BRNE k　　-64≤k≤63

说明：条件相对跳转，测试零标志位 Z，如果 Z 位被清零，则相对 PC 值跳转 k 个字。这条指令相当于指令 BRBC 1，k

操作：If Rd≠Rr（Z=0），then PC←（PC+1）+k；else PC←PC+1

机器码：1111 01kk kkkk k001

对标志位的影响：无

（5）进位标志位 C 为“1”跳转

BRCS k　　-64≤k≤63

说明：条件相对跳转，测试进位标志 C，如果 C 位被置位，则相对 PC 值跳转 k 个字。这条指令相当于指令 BRBS 0，k。

操作：If C=1 then PC←（PC+1）+k ；else PC←PC+1

机器码：1111 00kk kkkk k000

对标志位的影响：无

（6）进位标志位 C 为“0”跳转

BRCC k　　-64≤k≤63

说明：条件相对跳转，测试进位标志 C，如果 C 位被清除，则相对 PC 值跳转 k 个字。这条指令相当于指令 BRBC 0，k。

操作：If C=0 then PC←（PC+1）+k　else PC←PC+1

机器码：1111 01kk kkkk k000

对标志位的影响：无

（7）大于或等于跳转（对无符号数）

BRSH k　　-64≤k≤63

说明：条件相对跳转，测试进位标志 C，如果 C 位被清零，则相对 PC 值跳转 k 个字。如果在执行 CP、CPI、SUB 或 SUBI 指令后，立即执行该指令，且当在寄存器 Rd 中无符号二进制数大于或等于寄存器 Rr 中无符号二进制数时,将发生跳转。该指令相当于指令 BRBS 0，k。

操作：If Rd≥Rr（C=0） then PC←（PC+1）+k else PC←PC+1

机器码：1111 01kk kkkk k000

对标志位的影响：无

（8）小于跳转（对无符号数）

BRLO k　　-64≤k≤63

说明：条件相对跳转，测试进位标志 C，如果 C 位被置位，则相对 PC 值跳转 k 个字。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中无符号二进制数小于在寄存器 Rr 中无符号二进制数时，将发生跳转。该指令相当于指令 BRBS 0，k。

操作：If Rd<Rr（C=1） then PC←（PC+1）+k　else PC←PC+1

机器码：1111 00kk kkkk k000

对标志位的影响：无

（9）结果为负跳转

BRMI k　　-64≤k≤63

说明：条件相对跳转，测试负号标志 N，如果 N 位被置位，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBS 2，k。

操作：If N=1 then　PC←（PC+1）+k　else PC←PC+1

机器码：1111 00kk kkkk k010

对标志位的影响：无

（10）结果为正跳转

BRPL k　　-64≤k≤63

说明：条件相对跳转，测试负号标志 N，如果 N 位被清零，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBC 2，k

操作：If N=0 then PC←（PC+1）+k；　else PC←PC+1

机器码：1111 01kk kkkk k010

对标志位的影响：无

（11）大于或等于跳转（带符号数）

BRGE k　　-64≤k≤63

说明：条件相对跳转，测试符号标志 S，如果 S 位被清零，则相对 PC 值跳转 k 个字。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中带符号二进制数大于或等于寄存器 Rr 中带符号二进制数时,将发生跳转。该指令相当于指令 BRBC 4，k。

操作：If Rd≥Rr （N⊕V=0） then PC←（PC+1）+k，else PC←PC+1

机器码：1111 01kk kkkk k100

对标志位的影响：无

（12）小于跳转（带符号数）

BRLT k　　-64≤k≤63

说明：条件相对跳转，测试符号标志 S，如果 S 位被置位，则相对 PC 值跳转 k 个字。如果在执行 CP、CPI、SUB 或 SUBI 指令后立即执行该指令，且当在寄存器 Rd 中带符号二进制数小于在寄存器 Rr 中带符号二进制数时，将发生跳转。该指令相当于指令 BRBS 4，k。

操作：If Rd＜Rf （N⊕V＝1） then PC←（PC＋1）＋k else PC←PC＋1

机器码：1111 00kk kkkk k100

对标志位的影响：无

（13）半进位标志为“1”跳转

BRHS k　　-64≤k≤63

说明：条件相对跳转，测试半进位标志 H，如果 H 位被置位，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBS 5，k。

操作：If H=1 then PC←（PC+1）+k，else PC←PC+1

机器码：1111 00kk kkkk k101

对标志位的影响：无

（14）半进位标志为“0”跳转

BRHC k　　-64≤k≤63

说明：条件相对跳转，测试半进位标志 H，如果 H 位被清零，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBC 5，k。

操作：If H=0 then PC←（PC+1）+k， else PC←PC+1

机器码：1111 01kk kkkk k101

对标志位的影响：无

（15）T 标志为“1”跳转

BRTS k　　-64≤k≤63

说明：条件相对跳转，测试标志位 T，如果标志位 T 被置位，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBS 6，k。

操作：If T=1 then PC←（PC+1）+k else PC←PC+1

机器码：1111 00kk kkkk k110

对标志位的影响：无

（16）T 标志为“0”跳转

BRTC k　　-64≤k≤63

说明：条件相对跳转，测试 T 标志位，如果标志位 T 被清零，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBC 6，k。

操作：If T=0 then PC←（PC+1）+k else PC←PC+1

机器码：1111 01kk kkkk k110

对标志位的影响：无

（17）溢出标志为“1”跳转

BRVS k　　-64≤k≤63

说明：条件相对跳转，测试溢出标志 V，如果 V 位被置位，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBS 3，k。

操作：If V=1 then PC←（PC+1）+k，else PC←PC+1

机器码：1111 00kk kkkk k011

对标志位的影响：无

（18）溢出标志为“0”跳转

BRVC k　　-64≤k≤63

说明：条件相对跳转，测试溢出标志 V，如果 V 位被清零，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBC 3，k。

操作：If V=0 then PC←（PC+1）+k，else PC←PC+1

机器码：1111 01kk kkkk k011

对标志位的影响：无

（19）中断标志为“1”跳转

BRIE k　　-64≤k≤63

说明：条件相对跳转，测试全局中断允许标志 I，如果 I 位被置位，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBS 7，k。

操作：If I=1 then PC←（PC+1）+k，else PC←PC+1

机器码：1111 00kk kkkk k111

对标志位的影响：无

（20）中断标志为“0”跳转

BRID k　　-64≤k≤63

说明：条件相对跳转，测试全局中断允许标志 I，如果 I 位被清零，则相对 PC 值跳转 k 个字。该指令相当于指令 BRBC 7，k。

操作：If I=0 then PC←（PC+1）+k, else PC←PC+1

机器码：1111 01kk kkkk k111

对标志位的影响：无

2．测试条件符合跳行跳转指令

（1）相等跳行

CPSE Rd，Rr　　0≤d≤31，　0≤r≤31

说明：该指令完成两个寄存器 Rd 和 Rr 的比较，若 Rd=Rr，则跳一行执行指令。

操作：If Rd=Rr then PC←PC+2（or 3） else PC←PC+1

机器码：0001 00rd dddd rrrr

对标志位的影响：无

（2）寄存器位为“0”跳行

SBRC Rr，b　　0≤r≤31，　0≤b≤7

说明：该指令测试寄存器第 b 位，如果该位被清零，则跳一行执行指令。

操作：If Rd（b）=0 then PC←PC+2（or 3） else PC←PC+1

机器码：1111 110r rrrr 0bbb

对标志位的影响：无

（3）寄存器位为“1”跳行

SBRS Rr，b　　0≤r≤31， 0≤b≤7

说明：该指令测试寄存器第 b 位，如果该位被置位，则跳下一行执行指令。

操作：If Rr（b）=1 then PC←PC+2（or 3），else PC←PC+1

机器码：1111 111r rrrr 0bbb

对标志位的影响：无

（4）I/O 寄存器位为“0”跳行

SBIC P，b　　0≤P≤31， 0≤b≤7

说明：该指令测试 I/O 寄存器第 b 位，如果该位被清零，则跳一行执行指令。该指令只在低 32 个 I/O 寄存器内操作，地址为 I/O 寄存器空间的 0～31。

操作：If P（b）=0 then PC←PC+2（or 3），else PC←PC+1

机器码：1001 1001 PPPP Pbbb

对标志位的影响：无

（5）I/O 寄存器位为“1”跳行

SBIS P，b　　0≤P≤31，0≤b≤7

说明：该指令测试 I/O 寄存器第 b 位，如果该位被置位，则跳一行执行指令。该指令只在低 32 个 I/O 寄存器内操作，地址为 I/O 寄存器空间的 0～31。

操作：If P（b）=1 then PC←PC+2（or3），else PC←PC+1

机器码：1001 1011 PPPP Pbbb

对标志位的影响：无

3.3.3　子程序调用和返回指令

在程序设计中通常把具有一定功能模块的公用程序段定义为子程序。为了实观调用子程序的功能，指令系统中都有调用子程序指令。调用子程序指令与跳转指令的区别如下：执行调用子程序时把下一条指令地址 PC 值保留到堆栈中，即断点保护，然后把子程序的起始地址置入 PC，子程序执行完毕返回时，将断点由堆栈中弹出到 PC，然后从断点处继续执行原程序；而跳转指令既不保护断点也不返回原程序。在每个子程序中都必须有返回指令，返回指令的功能就是把调用前压入堆栈的断点弹出置入 PC，恢复执行调用子程序前的原程序。

在一个程序中，子程序中还会调用别的子程序，这称为子程序嵌套。每次调用子程序时必须将下条指令地址保存起来，返回时，按后进先出原则依次取出相应的 PC 值。堆栈就是按后进先出规则存取数据的，调用指令和返回指令具有自动保存和恢复 PC 内容的功能，即自动进栈，自动出栈。

1．相对调用

RCALL k　　-2048≤k≤2047

说明：将 PC+1 后的值（RCALL 指令后的下一条指令地址）压入堆栈，然后调用在当前 PC 前或后 k+1 处地址的子程序。

操作：STACK←PC+1，SP←SP-2，PC←（PC+1）+k 机器码：1101 kkkk kkkk kkkk

对标志位的影响：无

2．间接调用

ICALL

说明：间接调用由 Z 寄存器中（16 位指针寄存器）指向的子程序。地址指针寄存器 Z 为 16 位，允许调用在当前程序存储空间 64K 字（128K 字节）内的子程序。

操作：STACK←PC+1，SP←SP-2，PC←Z　　机器码：1001 0101 0000 1001

对标志位的影响：无

3．直接调用

CALL k　　0≤k≤65535

说明：将 PC+2 后的值（CALL 指令后的下一条指令地址）压入堆栈，然后直接调用 k 处地址的子程序。

操作：STACK←PC+2，SP←SP-2，PC←k

机器码：1001 010k kkkk 111k kkkk kkkk kkkk kkkk

对标志位的影响：无

4．从子程序返回

RET

说明：从子程序返回，返回地址从堆栈中弹出。

操作：SP←SP+2，PC←STACK　　机器码：1001 0101 0000 1000

对标志位的影响：无

5．从中断程序返回

RETI

说明：从中断程序中返回，返回地址从堆栈中弹出，且全局中断标志被置位。

操作：SP←SP+2，PC←STACK　　机器码：1001 0101 0001 1000

对标志位的影响：I（1）

注意：（1）主程序应跳过中断向量区，防止给修改补充中断程序带来麻烦。

（2）应在中断向量区中不用的中断入口地址写上 RETI——中断返回指令，有抗干扰作用。

3.4　数据传送指令

数据传送指令是在编程中使用最频繁的一类指令，数据传送指令是否灵活快速对程序的执行速度产生很大影响。数据传送指令执行操作是寄存器与寄存器、寄存器与数据存储器 SRAM、寄存器与 I/O 端口之间的数据传送，另外还有从程序存储器直接取数指令 LPM（ELPM）以及 PUSH 压栈和 POP 出栈的堆栈指令。

所有的传送指令的操作对标志位均无影响。

3.4.1 直接寻址数据传送指令

1．工作寄存器间传送数据

MOV Rd，Rr　　0≤d≤31，0≤r≤31

说明：该指令将一个寄存器内容传送到另一个寄存器中，源寄存器 Rr 的内容不改变，而目的寄存器 Rd 复制了 Rr 的内容。

操作：Rd←Rr　　PC←PC+1　　机器码：0010 11rd dddd rrrr

2．SRAM 数据直接送寄存器

LDS Rd，k　　0≤d≤31，　0≤k≤65535

说明：把 SRAM 中一个存储单元的内容（字节）装入到寄存器，其中 k 为该存储单元的 16 位地址。

操作：Rd←（k）　　PC←PC+2　　机器码：1001 000d dddd 0000 kkkk kkkk kkkk kkkk

3．寄存器数据直接送 SRAM

STS k，Rr　　0≤r≤31，　0≤k≤65535

说明：将寄存器的内容直接存储到 SRAM 中，其中 k 为存储单元的 16 位地址。

操作：（k）←Rr　　PC←PC+1　　机器码：1001 001d dddd 0000 kkkk kkkk kkkk kkkk

4．立即数送寄存器

LDI Rd，K　　16≤d≤31，　0≤K≤255

说明：装入一个 8 位立即数到寄存器 R16～R31 中。

操作：Rd←K　　PC←PC+1　　机器码：1110 KKKK dddd KKKK

3.4.2 间接寻址数据传送指令

1．使用 X 指针寄存器间接寻址传送数据

（1）使用地址指针寄存器 X 间接寻址将 SRAM 内容装入到指定寄存器

①LD Rd，X　　0≤d≤31；将指针为 X 的 SRAM 中的数送寄存器，指针不变。

操作：Rd←（X）　　PC←PC+1　　机器码：1001 000d dddd 1100

②LD Rd，X+　　0≤d≤31；先将指针为 X 的 SRAM 中的数送寄存器，X 指针加 1。

操作：Rd←（X），X←X+1 PC←PC+1　　机器码：1001 000d dddd 1101

③LD Rd，-X　0≤d≤31；X 指针减 1，将指针为 X 的 SRAM 中的数送寄存器。

操作：X←X-1，Rd←（X）　PC←PC+1　　机器码：1001 000d dddd 1110

（2）使用地址指针寄存器 X 间接寻址将寄存器内容存储到 SRAM

①ST X，Rr　　0≤d≤31；将寄存器内容送 X 为指针的 SRAM 中，X 指针不改变。

操作：（X）←Rr　　PC←PC+1　　机器码：1001 001r rrrr 1100

②ST X，Rr　　0≤d≤31；先将寄存器内容送 X 为指针的 SRAM 中，后 X 指针加 1。

操作：（X）←Rr，X←X+1　PC←PC+1　　机器码：1001 001r rrrr 1101

③ST -X，Rr　　0≤d≤31；先将 X 指针减 1，然后将寄存器内容送 X 为指针的 SRAM 中。

操作：X←X-1，（X）←Rr　PC←PC+1　　机器码：1001 001r rrrr 1110

2．使用 Y 指针寄存器间接寻址传送数据

（1）使用地址指针寄存器 Y 间接寻址将 SRAM 中的内容装入寄存器

①LD Rd，Y　　0≤d≤31；将指针为 Y 的 SRAM 中的数送寄存器，Y 指针不变。

操作：Rd←（Y）　　PC←PC+1　　机器码：1000 000d dddd 1000

②LD Rd，Y+　　0≤d≤31；先将指针为 Y 的 SRAM 中的数送寄存器，然后 Y 指针加 1。

操作：Rd←（Y），Y←Y+1　PC←PC+1　　机器码：1001 000d dddd 1001

③LD Rd，-Y　　0≤d≤31；先将 Y 指针减 1，将指针为 Y 的 SRAM 中的数送寄存器。

操作：Y←Y-1，Rd←（Y）　PC←PC+1　　机器码：1001 000d dddd 1010

④LDD Rd，Y+q　　0≤d≤31，0≤q≤63；将指针为 Y+q 的 SRAM 中的数送寄存器，而 Y 指针不改变。

操作：Rd←（Y+q）　　PC←PC+1　　机器码：10q0 qq0d dddd 1qqq

（2）使用地址指针寄存器 Y 间接寻址将寄存器内容存储到 SRAM

①ST Y，Rr　　0≤d≤31；将寄存器内容送 Y 为指针的 SRAM 中，Y 指针不改变。

操作：（Y）←Rr　　PC←PC+1　　机器码：1000 001r rrrr 1000

②ST Y+，Rr　　0≤d≤31；先将寄存器内容送 Y 为指针的 SRAM 中，然后 Y 指针加 1。

操作：（Y）←Rr，Y←Y+1　PC←PC+1　　机器码：1001 001r rrrr 1001

③ST －Y，Rr　　0≤d≤31；先将 Y 指针减 1，然后将寄存器内容送 Y 为指针的 SRAM 中。

操作：Y←Y-1，（Y）←Rr　PC←PC+1　　机器码：1001 001r rrrr 1010

④STD Y+q，Rr　　0≤d≤31，0≤q≤63；将寄存器内容送 Y+q 为指针的 SRAM 中。

操作：（Y+q）←Rr　　PC←PC+1　　机器码：10q0 qq1r rrrr 1qqqq

3．使用 Z 指针寄存器间接寻址传送数据

（1）使用地址指针寄存器 Z 间接寻址将 SRAM 中的内容装入到指定寄存器

①LD Rd，Z　　0≤d≤31；将指针为 Z 的 SRAM 中的数送寄存器，Z 指针不变。

操作：Rd←（Z）　　PC←PC+1　　机器码：1000 000d dddd 0000

②LD Rd，Z+　　0≤d≤31；先将指针为 Z 的 SRAM 中的数送寄存器，然后 Z 指针加 1。

操作：Rd←（Z），Z←Z+1　PC←PC+1　　机器码：1001 000d dddd 0001

③LD Rd，-Z　　0≤d≤31；先将 Z 指针减 1，然后将指针为 Z 的 SRAM 中的数送寄存器。

操作：Z←Z-1，Rd←（Z）　PC←PC+1　　机器码：1001 000d dddd 0010

④LDD Rd，Z+q　　0≤d≤31，0≤q≤63；将指针为 Z+q 的 SRAM 中的数送寄存器，而 Z 指针不改变。

操作：Rd←（Z+q）　　PC←PC+1　　机器码：10q0 qq0d dddd 0qqq

（2）使用地址指针寄存器 Z 间接寻址将寄存器内容存储到 SRAM

①ST Z，Rr　　0≤d≤31；将寄存器内容送 Z 为指针的 SRAM 中，Z 指针不改变。

操作：（Z）←Rr　　PC←PC+1　　机器码：1000 001r rrrr 0000

②ST Z+，Rr　　0≤d≤31；先将寄存器内容送 Z 为指针的 SRAM 中，然后 Z 指针加 1。

操作：（Z）←Rr，Z←Z+1　　PC←PC+1　　机器码：1001 001r rrrr 0001

③ST -Z，Rr　　0≤d≤31；先将 Z 指针减 1，然后将寄存器内容送 Z 为指针的 SRAM 中。

操作：Z←Z-1，（Z）←Rr　PC←PC+1　　机器码：1001 001r rrrr 0010

④STD Z+q，Rr　　0≤d≤31，0≤q≤63；将寄存器内容送 Z+q 为指针的 SRAM 中。

操作：(Z+q) ←Rr　　PC←PC+1　　机器码：10q0 qq1r rrrr 0qqq

以上 22 条指令操作之后，X、Y、Z 指针寄存器要么不改变，要么是加 1 或减 1。使用 X、Y、Z 指针寄存器的这些特性特别适用于访问矩阵表和堆栈指针等。

3.4.3　从程序存储器中取数装入寄存器指令

1．从程序存储器中取数装入寄存器 R0

LPM

说明：将 Z 指向的程序存储器空间的一个字节装入寄存器 R0。

操作：R0←（Z）　　PC←PC+1　　机器码：1001 0101 1100 1000

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

2．从程序存储器中取数装入寄存器

LPM Rd，Z　0≤d≤31

说明：将 Z 指向的程序存储器空间的一个字节装入寄存器 Rd。

操作：Rd←（Z）　　PC←PC+1　　机器码：1001 000d dddd 0100

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

3．带后增量的从程序存储器中取数装入寄存器 Rd

LPM Rd，Z+

说明：将 Z 指向的程序存储器空间的一个字节装入 Rd，然后 Z 指针加 1。

操作：Rd←（Z），Z←Z+1　PC←PC+1　　机器码：1001 000d dddd 0101

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，16 位地址指针寄存器 Z 的高 15 位为程序存储器的字地址，最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。

4．从程序存储器中取数装入寄存器 R0(扩展)

ELPM

说明：将 RAMPZ:Z 指向的程序存储器空间的一个字节装入寄存器 R0。

操作：R0←（RAMPZ:Z）　　PC←PC+1　　机器码：1001 0101 1101 1000

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，PAMPZ 寄存器的最低位，加上 16 位地址指针寄存器 Z 的高 15 位，组成 16 位的程序存储器字寻址地址，而 Z 寄存器的最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址整个 128K 字节（64K 字）的程序存储器空间。

5．从程序存储器中取数装入寄存器（扩展）

ELPM Rd，Z　0≤d≤31

说明：将 RAMPZ:Z 指向的程序存储器空间的一个字节装入寄存器 Rd。

操作：Rd←（RAMPZ:Z）　　PC←PC+1　　机器码：1001 000d dddd 0110

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，PAMPZ 寄存器的最低位，加上 16 位地址指针寄存器 Z 的高 15 位，组成 16 位的程序存储器字寻址地址，而 Z

寄存器的最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址整个 128K 字节（64K 字）的程序存储器空间。

6．带后增量的从程序存储器中取数装入寄存器 Rd（扩展）

LPM Rd，Z+

说明：将 RAMPZ:Z 指向的程序存储器空间的一个字节装入 Rd，然后 RAMPZ:Z 指针加 1。

操作：Rd←（RAMPZ:Z），　RAMPZ:Z←RAMPZ:Z+1　PC←PC+1

机器码：1001 000d dddd 0111

注释：由于程序存储器的地址是以字（双字节）为单位的，因此，PAMPZ 寄存器的最低位，加上 16 位地址指针寄存器 Z 的高 15 位，组成 16 位的程序存储器字寻址地址，而 Z 寄存器的最低位 LSB 为“0”时，指字的低字节；为“1”时，指字的高字节。该指令能寻址整个 128K 字节（64K 字）的程序存储器空间。

3.4.4　写程序存储器指令

SPM

说明：将寄存器对 R1:R0 的内容（16 位字）写入 Z 指向的程序存储器空间。

操作：（Z）←R1:R0　　PC←PC+1　　机器码：1001 0101 1110 1000

注释：该指令用于具有在应用自编程性能的 AVR 单片机，应用这一特性，单片机系统程序可以在运行中更改程序存储器中的程序，实现动态修改系统程序的功能。由于程序存储器的地址是以字（双字节）为单位的，因此，寄存器 R1:R0 的内容组成一个 16 位的字，其中 R1 为字的高位字节，R0 为字的低位字节。该指令能寻址程序存储器空间为一个 64K 字节的页（32k 字）。具体应用见相关章节的介绍。

3.4.5　I/O 口数据传送

1．I/O 口数据装入寄存器

IN Rd，P　　0≤d≤31，　0≤P≤63

说明：将 I/O 空间（口、定时器、配置寄存器等）的数据传送到寄存器区中的寄存器 Rd 中。

操作：Rd←P　　PC←PC+1　　机器码：1011 0PPd dddd PPPP

2．寄存器数据送 I/O 口

OUT P，Rr　　0≤r≤31，　0≤P≤63

说明：将寄存器区中 Rr 的数据传送到 I/O 空间（口、定时器、配置寄存器等）。

操作：P←Rr　　PC←PC+1　　机器码：1011 1PPr rrrr PPPP

3.4.6　堆栈操作指令

AVR 单片机的特殊功能寄存器中有一个 16 位的堆栈指针寄存器 SP，它指出栈顶的位置，在指令系统中有两条用于数据传送的栈操作指令。

1．进栈指令

PUSH Rr　　0≤d≤31

说明：该指令存储寄存器 Rr 的内容到堆栈。

操作 STACK←Rr，SP←SP-1　　PC←PC+1　　机器码：1001 001d dddd 1111

2．出栈指令

POP Rd　　0≤d≤31

说明：该指令将堆栈中的字节装入到寄存器 Rd 中。

操作：SP←SP+1，Rd←STACK　　PC←PC+1　　机器码：1001 000d dddd 1111

3.5　位操作和位测试指令

AVR 单片机指令系统中有四分之一的指令为位操作和位测试指令，这些指令的灵活应用极大地提高了系统的逻辑控制和处理能力。

3.5.1　带进位逻辑操作指令

1．寄存器逻辑左移

LSL Rd　　0≤d≤31

说明：寄存器 Rd 中所有位左移 1 位，第 0 位被清零，第 7 位移到 SREG 中的 C 标志。该指令完成一个无符号数乘 2 的操作。

操作：C←$b_7b_6b_5b_4b_3b_2b_1b_0$←0　　PC←PC+1　　机器码：0000 11dd dddd dddd

对标志位的影响：H S V N Z C

2．寄存器逻辑右移

LSR Rd　　0≤d≤31

说明：寄存器 Rd 中所有位右移 1 位，第 7 位被清零，第 0 位移到 SREG 中的 C 标志。该指令完成一个无符号数除 2 的操作，C 标志被用于结果舍入。

操作：0→$b_7b_6b_5b_4b_3b_2b_1b_0$→C　　PC←PC+1　　机器码：1001 010d dddd 0110

对标志位的影响：S V N（0） Z C

3．带进位位的寄存器逻辑循环左移

ROL Rd　　0≤d≤31

说明：寄存器 Rd 的所有位左移 1 位，C 标志被移到 Rd 的第 0 位，Rd 的第 7 位移到 C 标志。

操作：C←$b_7b_6b_5b_4b_3b_2b_1b_0$←C　　PC←PC+1　　机器码：0001 11dd dddd dddd（参见ADC指令）

对标志位的影响：H S V N Z C

4．带进位位的寄存器逻辑循环右移

ROR Rd　　0≤d≤31

说明：寄存器 Rd 的所有位右移 1 位，C 标志被移到 Rd 的第 7 位，Rd 的第 0 位移到 C 标志。

操作：C→$b_7b_6b_5b_4b_3b_2b_1b_0$→C　　PC←PC+1　　机器码：1001 010d dddd 0111

对标志位的影响：S V N Z C

5．寄存器算术右移

ASR Rd　　0≤d≤31

说明：寄存器 Rd 中的所有位右移 1 位，而第 7 位保持原逻辑值，第 0 位被装入 SREG 的 C 标志位。这个操作实现 2 的补码值除 2，而不改变符号，进位标志用于结果的舍入。

操作：$b_7 \rightarrow b_7b_6b_5b_4b_3b_2b_1b_0 \rightarrow C$　PC←PC+1　　机器码：1001 010d dddd 0101

对标志位的影响：S V N Z C

6．寄存器半字节交换

SWAP Rd　　0≤d≤31

说明：寄存器中的高半字节和低半字节交换。

操作：$b_7b_6b_5b_4 \leftrightarrow b_3b_2b_1b_0$ PC←PC+1　　机器码：1001 010d dddd 0010

对标志位的影响：无

3.5.2　位变量传送指令

1．寄存器中的位存储到 SREG 中的 T 标志

BST Rr，b　　0≤d≤31，0≤b≤7

说明：把寄存器 Rr 中的位 b 存储到 SREG 状态寄存器中的 T 标志位中。

操作：T←Rr（b）　　PC←PC+1　　机器码：1111 101d dddd 0bbb

对标志位的影响：T

2．SREG 中的 T 标志位值装入寄存器 Rd 中的某一位

BLD Rd，d　　0≤d≤31，0≤b≤7

说明：复制 SREG 状态寄存器的 T 标志到寄存器 Rd 中的位 b。

操作：Rd（b）←T　　PC←PC+1　　机器码：1111 100d dddd 0bbb

对标志位的影响：无

3.5.3　位变量修改指令

1．状态寄存器 SREG 的指定位置“1”

BSET s　　0≤s≤7

说明：置“1”状态寄存器 SREG 的某一标志位。

操作：SREG（s）←1　　PC←PC+1　　机器码：1001 0100 0sss 1000

对标志位的影响：I T H S V N Z C

2．状态寄存器 SREG 的指定位清“0”

BCLR s　　0≤s≤7

说明：清零 SREG 状态寄存器 SREG 中的一个标志位。

操作：SREG（s）←0　　PC←PC+1　　机器码：1001 0100 1sss 1000

对标志位的影响：I T H S V N Z C

3．I/O 寄存器的指定位置“1”

SBI P，b　　0≤P≤31，　0≤b≤7

说明：对 P 指定的 I/O 寄存器的指定位置“1”。该指令只在低 32 个 I/O 寄存器内操作，I/O 寄存器地址为 0～31。

操作：I/O（P，b）←1　　PC←PC+1　　机器码：1001 1010 PPPP Pbbb

对标志位的影响：无

4．I/O 寄存器的指定位置“0”

CBI P，b　　0≤P≤31，0≤b≤7

说明：清零指定 I/O 寄存器中的指定位。该指令只用在低 32 个 I/O 寄存器上操作，I/O 寄存器地址为 0～31。

操作：I/O（P，b）←0　　PC←PC+1　　机器码：1001 1000 PPPP Pbbb

对标志位的影响：无

5．置进位位

SEC　　置位 SREG 状态寄存器中的进位标志 C

操作：C←1　　PC←PC+1　　机器码：1001 0100 0000 1000

对标志位的影响：C（1）

6．清进位位

CLC　　清零 SREG 状态寄存器中的进位标志 C

操作：C←0　　PC←PC+1　　机器码：1001 0100 1000 1000

对标志位的影响：C（0）

7．置位负标志位

SEN　　置位 SREG 状态寄存器中的负数标志 N

操作：N←1　　PC←PC+1　　机器码：1001 0100 0010 1000

对标志位的影响：N（1）

8．清负标志位

CLN 清零 SREG 状态寄存器中的负数标志 N

操作：N←0　　PC←PC+1　　机器码：1001 0100 1010 1000

对标志位的影响：N（0）

9．置零标志位

SEZ　　置位 SREG 状态寄存器中的零标志 Z

操作：Z←1　　PC←PC+1　　机器码：1001 0100 0001 1000

对标志位的影响：Z（1）

10．清零标志位

CLZ　　清零 SREG 状态寄存器中的零标志 Z

操作：Z←0　　PC←PC+1　　机器码：1001 0100 1001 1000

对标志位的影响：Z（0）

11．触发全局中断位

SEI　　置位 SREG 状态寄存器中的全局中断标志 I

操作：I←1　　PC←PC+1　　机器码：1001 0100 0111 1000

对标志位的影响：I（1）

12．禁止全局中断位

CLI　　清除 SREG 状态寄存器中的全局中断标志 I

操作：I←0　　PC←PC+1　　机器码：1001 0100 1111 1000

对标志位的影响：I（0）

13．置 S 标志位

SES　　置位 SREG 状态寄存器中的符号标志 S

操作：　S←1　　PC←PC+1　　机器码：1001 0100 0100 1000

对标志位的影响：S（1）

14．清 S 标志位

CLS　　清零 SREG 状态寄存器中的符号标志 S

操作：S←0　　PC←PC+1　　机器码：1001 0100 1100 1000

对标志位的影响：S（0）

15．置溢出标志位

SEV　　置位 SREG 状态寄存器中的溢出标志 V

操作：V←1　　PC←PC+1　　机器码：1001 0100 0011 1000

对标志位的影响：V（1）

16．清溢出标志位

CLV　　清零 SREG 状态寄存器中的溢出标志 V

操作：V←0　　PC←PC+1　　机器码：1001 0100 1011 1000

对标志位的影响：V（0）

17．置 T 标志位

SET　　置位 SREG 状态寄存器中的 T 标志

操作：T←1　　PC←PC+1　　机器码：1001 0100 0110 1000

对标志位的影响：T（1）

18．清 T 标志位

CLT　　清零 SREG 状态寄存器中的 T 标志

操作：T←0　　PC←PC+1　　机器码：1001 0100 1110 1000

对标志位的影响：T（0）

19．置半进位标志

SHE　　置位 SREG 状态寄存器中的半进位标志 H

操作：H←1　　PC←PC+1　　机器码：1001 0100 0101 1000

对标志位的影响：H（1）

20．清半进位标志

CLH 清零 SREG 状态寄存器中的半进位标志 H

操作：H←0　　PC←PC+1　　机器码：1001 0100 1101 1000

对标志位的影响：H（0）

3.6　MCU 控制指令

MCU 控制指令有 3 条，主要用于控制 MCU 的运行方式以及清零看门狗定时器。

1．空操作指令

NOP

说明：该指令完成一个单周期空操作。

操作：无　　PC←PC+1　　机器码：0000 0000 0000 0000

对标志位的影响：无

应用：延时等待、产生方波。抗干扰处理：在空的程序存储器单元中写上空操作，空操作指令最后加一跳转指令，转到$000H。

2．进入休眠方式指令

SLEEP

说明：该指令使 MCU 进入休眠方式运行。休眠模式由 MCU 控制寄存器定义。当 MCU 在休眠状态下由一个中断被唤醒时，在中断程序执行后，紧跟在休眠指令后的指令将被　执行。

操作：PC←PC+1　　MCU 进入由 MCU 控制寄存器定义的休眠方式运行

机器码：1001 0101 1000 1000

应用：省电，尤其对绿色、便携式仪器特别有用。

对标志位的影响：无

3．清零看门狗计数器

WDR

说明：该指令清零看门狗定时器。在允许使用看门狗定时器情况下，系统程序在正常运行中必须在 WD 预定比例器给出限定时间内执行一次该指令，以防止看门狗定时器溢出，造成系统复位。参见看门狗定时器硬件部分。

操作：清零看门狗定时器　　PC←PC+1　　机器码：1001 0101 1010 1000

对标志位的影响：无

应用：抗干扰、延时、提高系统的稳定性。

3.7　AVR 汇编语言系统

用汇编语言编写的程序称为汇编语言程序，或称源程序。显然汇编语言源程序比二进制的机器语言更容易学习和掌握。但是，单片机不能直接识别和执行汇编语言程序，因此需要使用一个专用的软件系统，将汇编语言的源程序“翻译”成二进制的机器语言程序——目标程序（执行代码）。这个专用软件系统就是汇编语言编译软件。

ATMEL 公司提供免费的 AVR 开发平台—AVR Studio 集成开发环境（IDE），其中就包括 AVR Assembler 汇编编译器（HTTP://www.atmel.com）。

用汇编语言编写程序必须按汇编工具软件的规定进行，否则会发生错误。汇编语言系统除了允许用户使用规定的汇编指令，还定义了一些用于编写汇编程序的伪指令，以及使用表达式等，以方便用户编写汇编程序。

3.7.1　汇编语言语句格式

汇编语言源程序是由一系列汇编语句组成的。AVR 的汇编语句的标准格式有以下 4 种：

（1）[标号:] 伪指令 [操作数][;注释]

（2）[标号:] 指令 [操作数][;注释]

（3）[;注释]

（4）空行

注意，在汇编语句中是不使用中括号的，格式中的中括号内容表示其可以缺省。

- 标号

标号是语句地址的标记符号，用于引导对该语句的访问和定位。使用标号的目的是为了跳转和分支指令及在程序存储器、数据存储器SRAM以及E^2PROM中定义变量名。有关标号的一些规定如下：

（1）标号一般由 ASCII 字符组成，第一个字符为字母；

（2）同一标号在一个独立的程序中只能定义一次；

（3）不能使用汇编语言中已定义的符号（保留字），如指令字、寄存器名、伪指令字等。

- 伪指令

在汇编语言程序中可以使用一些伪指令。伪指令不属于 CPU 指令集，编译时并不产生实际的目标机器操作代码，只是用于在汇编程序中对地址、寄存器、数据、常量等进行定义说明，以及对编译过程进行某种控制等。AVR 的指令系统不包括伪指令，伪指令通常由汇编编译系统给出。

- 指令

指令是汇编程序中主要的部分，汇编程序中使用 AVR 指令集中给出的全部指令。

- 操作数

操作数是指令操作时所需要的数据或地址。汇编程序完全支持指令系统所定义的操作数格式。但指令系统采用的操作数格式通常为数字形式，在编写程序时使用起来不太方便，因此，在编译器的支持下，可以使用多种形式的操作数，如数字、标识符、表达式等。

- 注释

注释部分仅用于对程序和语句进行说明，帮助程序设计人员阅读、理解和修改程序。只要有“;”符号，后面即为注释内容，注释内容长度不限，注释内容换行时，开头部分还要使用符号“;”。编译系统对注释内容不予理会，不产生任何代码。

- 其它字符

汇编语句中，“:”用于标号之后；空格用于指令字和操作数的分隔；指令有两个操作数时用“,”分隔两个操作数；“;”用于注释之前；“[ ]”中的内容表示可选项。

3.7.2 汇编器伪指令

AVR 的汇编器提供一些伪指令。伪指令并不直接转换生成操作执行代码。它只是用于指示编译生成目标程序的起始地址或常数表格的起始地址，或为工作寄存器定义符号名称，定义外设口地址，SRAM 工作区，规定编译器的工作内容等。因此伪指令有间接产生机器码、控制机器代码空间的分配、对编译器工作过程进行控制等。AVR 汇编系统使用的伪指令共 18 条，如表 3.6 中给出。

表 3-6 伪指令表

序 号	伪 指 令	说 明	序 号	伪 指 令	说 明
1	BYTE	在 RAM 中定义预留存储单元	10	EXIT	退出文件

2	CSEG	声明代码段	11	INCLUDE	包含指定的文件
3	DB	定义字节常数	12	MACRO	宏定义开始
4	DEF	定义寄存器符号名	13	ENDMACRO	宏定义结束
5	DEVICE	指定为何器件生成汇编代码	14	LISTMAC	列表宏表达式
6	DSEG	声明数据段	15	LIST	列表文件生成允许器
7	DW	定义字常数	16	NOLIST	关闭列表文件生成
8	EQU	定义标识符常量	17	ORG	设置程序起始位置
9	ESEG	声名E^2PROM段	18	SET	赋值给标识符

1．BYTE—为变量预留字节型存储单元

BYTE 伪指令是从指定的地址开始，在 SRAM 中预留若干字节的存储空间备用。备用存储空间以字节计算，个数由 BYTE 伪指令的参数或表达式的值确定。BYTE 伪指令前应使用一个标号，该标号既作为变量的名称，又作为符号地址，以标记备用存储空间在 SRAM 中的起始位置。该伪指令有一个参数，表示保留存储空间的字节数，但并不能给这些 RAM 单元赋值。BYTE 伪指令仅能用在数据段内（见伪指令 CSEG，DSEG 和 ESEG)。

语法：LABEL：.BYTE 表达式

示例：

```
.DSEG                       ;RAM 数据段(SRAM)
var1:  .BYTE 1              ;保留 1 个字节的存储单元，用 var1 标识
table: .BYTE tab_size       ;保留 tab_size 个字节的存储空间

.CSEG                       ;代码段开始(Flash)
     ldi r30，low(var1)     ;将保留存储单元 var1 起始地址的低 8 位装入 Z
     ldi r31，high(var1)    ;将保留存储单元 var1 起始地址的高 8 位装入 Z
     ld r1，Z               ;将保留存储单元的内容读到寄存器 R1
```

2．CSEG—声明代码段（Flash）

CSEG 伪指令用于声明代码段的起始（在 Flash 中)，既 CSEG 之后是要写入程序存储器中的指令代码、常数表格等。一个汇编程序可包含几个代码段，程序中缺省的段为代码段，这些代码段在编译过程中被连接成一个代码段。每个代码段内部都有自己的字定位计数器。可使用 ORG 伪指令定义该字定位计数器的初始值，作为代码段在程序存储器中的起始位置。CSEG 伪指令不带参数。在代码段中不能使用 BYTE 伪指令。

语法：.CSEG

3．DB—在程序存储器或E^2PROM存储器中定义字节常数

DB伪指令是从程序存储器或E^2PROM存储器的某个地址单元开始，存入一组规定的 8 位二进制常数（字节常数)。DB伪指令只能出现在代码段或E^2PROM段中。DB伪指令前应使用一个标号，以标记所定义的字节常数区域的起始位置。DB伪指令为一个表达式列表，表达式列表由多个表达式组成，但至少要含有一个表达式，表达式之间用逗号分隔。每个表达式值的范围必须在-128～255 之间。如果表达式的值是负数，则用 8 位 2 的补码表示，存入程序存储器或E^2PROM存储器中。如果DB伪指令用在代码段，并且表达式表中多于一个表达式，则以两个字节组合成一个字放在程序存储器中。如果表达式的个数是奇数，不管下一

行汇编代码是否仍是DB伪指令，最后一个表达式的值将单独以字的格式放在程序存储器中。

语法：LABEL：.DB 表达式表

4．DEF—定义寄存器符号名

DEF 伪指令给寄存器定义一个替代的符号名。在后序程序中可以使用定义的符号名来表示被定义的寄存器。可以给一个寄存器定义多个符号名。符号名在后面程序中可以重新定义指定。编译时，凡遇到符号名都以相应被定义的寄存器替代。

语法：.DEF 符号名 = 寄存器

5．DEVICE—指定为何器件生成汇编代码

DEVICE伪指令告知汇编器为何器件编译产生执行代码。如果在程序中使用该伪指令指定了器件型号，那么在编译过程中，若存在指定器件所不支持的指令，编译器则给出一个警告。如果代码段或E^2PROM段所使用的存储器空间大于指定器件本身所能提供的存储器容量，编译器也会给出警告。如果不使用DEVICE伪指令，则假定器件支持所有的指令，也不限制存储器容量的大小。

语法：.DEVICE Atmega16

6．DSEG—声明数据段（SRAM）

DSEG 伪指令声明数据段的起始。一个汇编程序文件可以包含几个数据段，这些数据段在汇编过程中被连接成一个数据段。在数据段中，通常是仅由 BYTE 伪指令（和标号）组成。每个数据段内部都有自己的字节定位计数器。可使用 ORG 伪指令定义该字节定位计数器的初始值，作为数据段在 SRAM 中的起始位置。DSEG 伪指令不带参数。

语法：.DSEG

7．DW—在程序存储器或E^2PROM存储器中定义字常数

DW伪指令是从程序存储器或E^2PROM存储器的某个地址单元开始，存入一组规定的 16 位二进制常数（字常数）。DW伪指令只能出现在代码段或E^2PROM段。DW伪指令前应使用一个标号，以标记所定义的字常数区域的起始位置。DW伪指令为一个表达式列表，表达式列表由多个表达式组成，但至少要含有一个表达式，表达式之间用逗号分隔。每个表达式值的范围必须在-32768～65535 之间。如果表达式的值是负数，则用 16 位 2 的补码表示。

语法：LABEL：.DW 表达式表

8．EQU—定义标识符常量

EQU 伪指令将表达式的值赋给一个标识符，该标识符为一个常量标识符，可以用于后面的指令表达式中，在汇编时凡遇到该标识符都以其等值表达式替代。在编写程序中，只要修改此表达式，就修改了程序中多处涉及该表达式的地方，减少了程序的修改量。但该标识符的值不能改变或重新定义。

语法：.EQU 标号 = 表达式

9．ESEG—声明E^2PROM段

ESEG伪指令声明E^2PROM段的开始。一个汇编文件可以包含几个E^2PROM段，这些E^2PROM段

在汇编编译过程中被连接成一个E²PROM段。在E²PROM段中不能使用BYTE伪指令。每个E²PROM段内部都有自己的字节定位计数器。可使用ORG伪指令定义该字节定位计数器的初始值，作为数据段在E²PROM中的起始位置。ESEG伪指令不带参数。

语法：.ESEG

10．EXIT—退出文件

EXIT 伪指令告诉汇编编译器停止汇编该文件。在正常情况下，汇编编译器的编译过程一直到文件的结束，如果中间出现 EXIT，则编译器放弃对其后边语句的编译。

如果 EXIT 出现在所包含文件中，则汇编编译器将结束对包含文件的编译，然后从本文件当前 INCLUDE 伪指令的下一行语句处开始继续编译。

语法：.EXIT

11．INCLUDE—包含指定的文件

INCLUDE 伪指令告诉汇编编译器开始从一个指定的文件中读入程序语句，并对读入的语句进行编译，直到该包含文件结束或遇到该文件中的 EXIT 伪指令，然后再从本文件当前 INCLUDE 伪指令的下一行语句处继续开始编译。在一个包含文件中，也可以使用 INCLUDE 伪指令来包含另外一个指定的文件。

语法：.INCLUDE“文件名”

12．MACRO—宏开始

MACRO 伪指令告诉汇编器一个宏程序的开始。MACRO 伪指令将宏程序名作为参数。当后面的程序中使用宏程序名，则表示在该处调用宏程序。宏有些象子程序或公式，它是一段含形式参数（亚元）的程序。一个宏程序中可带 10 个形式参数，这些形式参数在宏定义中用@0～@9 代表，参数之间用逗号分隔。当调用一个宏程序时，必须以实参替代形参。伪指令 ENDMACRO 定义宏程序的结束。

默认情况下，在汇编编译器生成的列表文件中仅给出宏的调用。如要在列表文件中给出宏的表达式，则必须使用 LISTMAC 伪指令。在列表文件的操作码域中，宏带有 a+ 的记号。

语法：.MACRO 宏名

13．ENDMACRO—宏结束

ENDMACRO 伪指令定义宏定义的结束。该伪指令并不带参数，参见 MACRO 宏定义伪指令。

语法：.ENDMACRO

示例：

```
.MACRO SUBI16                ;宏定义开始，定义一个 16 位的减法
    subi @1,low(@0)          ;减低 8 位字节
    sbci @2,high(@0)         ;带借位减高 8 位字节
.ENDMACRO                    ;宏定义结束
……
……
.CSEG                        ;代码段开始
    SUBI16 0x1234,r16,r17    ;调用宏完成 r17:r16 = r17:r16 - 0x1234
```

14．LISTMAC—列表时输出宏表达式

LISTMAC 伪指令告诉汇编编译器，在生成的列表清单文件中，显示所调用宏的内容。默认情况下，仅在列表清单文件中显示所调用的宏名和参数。

语法：.LISTMAC

15．LIST—启动生成列表清单文件功能

LIST 伪指令告诉汇编编译器在对源文件编译后，产生一个机器语言与源文件相对照的列表清单文件。正常情况下，汇编编译器在编译过程中将生成一个由汇编源代码、地址和机器操作码组成的列表文件。默认时为允许生成列表清单文件。该伪指令可以与 NOLIST 伪指令配合使用，以选择仅使某一部分的汇编源文件产生列清单表文件。

语法：.LIST

16．NOLIST—关闭列表清单文件生成功能

NOLIST 伪指令告诉汇编编译器关闭列表清单文件生成的功能。正常情况下，汇编编译器在编译过程中将生成一个由汇编源代码、地址和机器操作码组成的列表文件，默认情况下为允许生成列表清单文件。当使用该伪指令时，将禁止文件列表清单的产生。该伪指令与 LIST 伪指令配合使用，可以选择使某一部分的汇编源文件产生列表清单文件。

语法：.NOLIST

17．ORG—定义代码起始位置

ORG伪指令设置定位计数器为一个绝对数值，该数值为表达式的值，作为代码的起始位置。如果ORG伪指令出现数据段中，则设定SRAM定位计数器；如果该伪指令出现在代码段中，则设定程序存储器计数器；如果该伪指令出现在E^2PROM段中，则设定E^2PROM定位计数器。如果该伪指令前带标号（在相同的语句行），则标号的定位由ORG 的参数值定义。代码段和E^2PROM段定位计数器的默认值是零；而当汇编器启动时，SRAM定位计数器的默认值是 32（因为寄存器占有地址为 0～31）。文件中可以出现多处ORG伪指令，但后面ORG所带的地址不能小于前一个ORG所带的地址，也不能落在由前一个ORG定位的代码段空间（指同一个存储器空间）。注意，E^2PROM和SRAM定位计数器按字节计数，而程序存储器定位计数器按字计数。

语法：.ORG 表达式

18．SET—设置标识符与一个表达式值相等

与 EQU 作用类似，SET 伪指令将表达式的值赋值给一个标识符。该标识符为一个常量标识符，可以用于后面的指令表达式中，在汇编时凡遇到该标识符都以其等值表达式替代。与 EQU 不同的是，用 SET 伪指令赋值的标识符能在后面使用 SET 伪指令重新设置改变。在 SET 改变之后的代码使用新的等值表达式值，直到遇到下一个重新的 SET 定义为止。

语法：.SET 标号 = 表达式

3.7.3 表达式

在标准指令系统中，操作数通常只能使用纯数字格式，这给程序的编写带来了许多不

便。但是在编译系统的支持下，在编写汇编程序时允许使用表达式，以方便程序的编写。AVR 编译器支持的表达式是由操作数、函数和运算符组成。所有的表达式内部都是 32 位的。

1．操作数

AVR 汇编器使用的操作数有以下几种形式：

（1）用户定义的标号，该标号给出了放置标号位置的定位计数器的值。

（2）用户用 SET 伪指令定义的变量。

（3）用户用 EQU 伪指令定义的常数。

（4）整数常数，包括下列几种形式：

十进制数（默认），如：10、255；

十六进制数，如：0x0a、$0a、0xff、$ff；

二进制数，如：0b00001010、0b11111111

（5）PC：程序存储器定位计数器的当前值。

2．函数

AVR 汇编器定义了下列函数：

（1）LOW（表达式）　　返回一个表达式值的最低字节。

（2）HIGH（表达式）　　返回一个表达式值的第二个字节。

（3）BYTE2（表达式）　　与 HIGH 函数相同。

（4）BYTE3（表达式）　　返回一个表达式值的第三个字节。

（5）BYTE4（表达式）　　返回一个表达式值的第四个字节。

（6）LWRD（表达式）　　返回一个表达式值的 0～15 位。

（7）HWRD（表达式）　　返回一个表达式值的 16～31 位。

（8）PAGE（表达式）　　返回一个表达式值的 16～21 位。

（9）EXP2（表达式）　　返回（表达式值）2 次幂的值。

（10）LOG2（表达式）　　返回Log_2（表达式值）的整数部分。

3．运算符

汇编器提供的部分运算符见表 3-7。优先级数越高的运算符，其优先级也越高。表达式可以用小括号括起来，并且与括号外其他任意的表达式再组合成表达式。

表 3-7　部分运算符表

序　号	运 算 符	名　称	优 先 级	说　明
1	!	逻辑非	14	一元运算符，表达式是 0 返回 1， 表达式是 1 返回 0
2	～	逐位非	14	一元运算符，将表达式的值按位取反
3	-	负号	14	一元运算符，使表达式为算术负
4	*	乘法	13	二进制运算符，两个表达式相乘
5	/	除法	13	二进制运算符，左边表达式除以右边表达式，得整数的商值
6	+	加法	12	二进制运算符，两个表达式相加
7	—	减法	12	二进制运算符，左边表达式减去右边表达式
8	<<	左移	11	二进制运算符，左边表达式值左移右边表达式给出的次数
9	>>	右移	11	二进制运算符，左边表达式值右移右边表达式给出的次数

10	<	小于	10	二进制运算符，左边带符号表达式值小于右边带符号表达式值，则为 1，否则为 0
11	<=	小于等于	10	二进制运算符，左边带符号表达式值小于或等于右边带符号表达式值，则为 1，否则为 0
12	>	大于	10	二进制运算符，左边带符号表达式值大于右边带符号表达式值，则为 1，否则为 0
13	>=	大于等于	10	二进制运算符，左边带符号表达式值大于或等于右边带符号表达式值，则为 1，否则为 0
14	==	等于	9	二进制运算符，左边带符号表达式值等于右边带符号表达式值，则为 1，否则为 0
15	!=	不等于	9	二进制运算符，左边带符号表达式值不等于右边带符号表达式值，则为 1，否则为 0
16	&	逐位与	8	二进制运算符，两个表达式值之间逐位与
17	^	逐位异或	7	二进制运算符，两个表达式值之间逐位异或
18	|	逐位或	6	二进制运算符，两个表达式值之间逐位或
19	&&	逻辑与	5	二进制运算符，两个表达式值之间逻辑与，全非 0 则为 1，否则为 0
20	||	逻辑或	4	二进制运算符，两个表达式值之间逻辑或，非 0 则为 1，全 0 为 0

3.7.4 器件定义文件“m16def.inc”

在 AVR 的器件手册中，对寄存器进行了标称化的定义，如 32 个通用寄存器组用 R0～R31 表示，系统状态寄存器用 SREG 表示，A 口输出 I/O 寄存器用 PORTA 表示等，便于记忆、理解和使用。而当编写程序时，在指令中实际出现的应该是这些寄存器的空间地址，这就给编写程序造成了麻烦。

为了能让程序员编写程序时，不去使用那些不易记忆的寄存器地址，而是直接使用这些寄存器的标称名称，在 AVR 的开发软件平台中都含有各个 AVR 器件的标称定义文件。在 ATMEL 公司提供的 AVR 开发平台 AVR Studio 中，含有很多的“器件型号.inc”文件，这些 inc 文件，已将该器件所有的 I/O 寄存器、标志位、中断向量地址等进行了标称化的定义。这些定义的标称化符号与硬件结构的命名是相同的，这样在程序中就可直接使用标称化的符号，而不必去记住它的实际地址。如使用 PORTA 来代替 A 口 I/O 输出寄存器的地址$1b。读者可具体查看相关器件的定义文件（在安装 AVR Studio 系统目录的下一级子目录 Appnotes 中）。

我们看一下 Atmega16 器件定义文件“m16def.inc”中的部分内容：

```
;***** Specify Device
.device ATmega16

;***** I/O Register Definitions
.equ    SREG = $3f
.equ    SPH  = $3e
.equ    SPL  = $3d
.....
```

```
.equ    PORTA= $1b
.equ    DDRA = $1a
.equ    PINA = $19
……
.def    XL   = r26
.def    XH   = r27
.def    YL   = r28
.def    YH   = r29
.def    ZL   = r30
.def    ZH   = r31

.equ    RAMEND = $45F
……
```

可以看出，在器件定义文件中，大量使用汇编的伪指令 EQU、DEF 等，定义了各个寄存器标称名所对应的地址值。因此，当我们在编写 AVR 汇编程序时，在程序的开始处，需要先使用伪指令“.INCLUDE”，调用编译系统中的器件标识定义文件“****def.inc”。由于该文件已将该器件所有的 I/O 寄存器、标志位等进行了标称化的符号定义，这样在程序中就可直接使用标称化的符号，而不必去使用它的实际地址了。下面是一个标准的汇编程序的开始部分：

```
.INCLUDE "M16def.inc"                ;引用器件 I/O 标称定义文件
.DEF TEMP1 = r20                     ;定义标识符 TEMP1 代表工作寄存器 R20
……

.ORG  $0000                          ;代码段起始定位
        jmp RESET                    ;系统上电复位，跳转到主程序

.ORG  $002A                          ;代码段定位，跳过中断向量区

;程序先对器件进行初始化
;设置 ATmega16 的堆栈指针为$045F，RAMEND 在配置文件"M16def.inc"中已定义为$045F
;设置 A 口为输出方式工作
RESET:  ldi r16,high(RAMEND)         ;取 RAMEND 的高位字节
        out  SPH,r16                 ;将 RAMEND 的高位送堆栈寄存器 SP 高位字节中
        ldi  r16,low(RAMEND)         ;取 RAMEND 的低位字节
        out SPL,r16                  ;将 RAMEND 的低位送堆栈寄存器 SP 低位字节中

        ser  temp1                   ;将 temp1 即寄存器 R20 置为$FF
        out  DDRA,temp1              ;R20 值送 DDRA，A 口方向寄存器为$FF,设定为输出
        ……                           ;DDRA 在配置文件"M16def.inc"中已定义为$1A
```

以上对 AVR 单片机的指令结构和汇编系统做了介绍，在本书的第五章会结合汇编开发平台 AVR Studio 的使用，给出一些采用汇编编写的程序示例。

如果从具体应用的角度出发，并结合 AVR 单片机的特点，在产品设计开发过程中最好是使用高级语言来编写系统程序。同时，使用高级语言也便于从系统的角度描述介绍、学

习和掌握开发设计单片机系统程序的思想、方法和技巧。因此，本书将以高级语言 C 作为主要的工具（使用 CVAVR 软件平台）进行讲述。也由于篇幅的关系，对 AVR 汇编程序的使用不做更详细的学习。但了解和掌握汇编，对熟悉 AVR 的功能、系统软件的调试都非常重要。读者可进一步参考《AVR 单片机实用程序设计》一书，该书对 AVR 汇编进行了更为细致的介绍，并给出了大量的、实用的 AVR 汇编参考代码。

华东师范大学 电子科学技术系 马潮 2005.3.16