# 第3章 指令系统

3.1 指令保留字与寻址方式

3.2 基本指令系统

3.3 指令集与机器码对应表

3.4 GUN汇编器的基本语法

CPU是微机的运算和控制核心，其功能主要解释指令、执行指令和处理数据。一个CPU能识别哪些指令是由CPU的设计者给出，指令系统一般使用英文简写描述。基本掌握任何一种CPU的指令系统，当遇到新的CPU时就不会感到陌生，其本质不变。学习指令系统的基本方法是：理解寻址方式、记住几个简单指令、利用汇编语言编程练习。本章给出Arm Cortex-M的基本指令系统及汇编语言基本语法，下一章将给出编程框架及实践环境之后，开始进行实际的编程实践，通过实践，理解巩固这些基本指令。但本章可以通过汇编环境了解指令对应的机器码，直观的基本理解助记符与机器指令的对应关系。

# 3.1 指令保留字与寻址方式

## 3.1.1 指令保留字简表

CPU的功能是从外部设备获得数据，通过处理，再把处理结果送到CPU的外部世界。设计一个CPU，首先需要设计一套可以执行特定功能的操作命令，这种操作命令称为指令。CPU所能执行的各种指令的集合，称为该CPU的指令系统。一条CPU指令可以包含指令代码及操作数，编写汇编程序时，指令代码用英文简写或缩写的助记符表达，这个助记符称为指令保留字。

本节给出Arm Cortex-M微处理器的基本指令简表，主要接触一下表中简写字及含义，以便快速了解指令功能，为指令系统学习做个铺垫，也为复习时收拢知识。

表 3-1　基本指令保留字简表

| 类型 | | 保留字 | 含义 |
|---|---|---|---|
| 数据传送类 | | LDR、LDRH、LDRB、LDRSB、LDRSH、LDM | 取数指令（寄存器←存储器） |
| | | STR、STRH、STRB、STM | 存数指令（存储器←寄存器） |
| | | MOV、MOVS、MVN | 寄存器间数据传送指令 |
| | | PUSH、POP | 栈操作指令（进栈、出栈） |
| | | ADR | 生成与 PC 指针相关的地址 |
| 数据操作类 | 算术运算类 | ADC、ADD、RSB、SBC、SUB、MUL | 算术运算（加、减、乘） |
| | | CMN、CMP | 比较指令 |
| | 逻辑运算类 | AND、ORR、EOR、BIC | 逻辑运算（按位与、或、异或、位段清零） |
| | 移位类 | ASR、LSL、LSR、ROR | 算术右移、逻辑左移、逻辑右移、循环右移 |
| | 位操作类 | TST | 测试位指令 |
| | 数据序转类 | REV、REVSH、REVH | 反转字节序 |
| | 扩展类 | SXTB、SXTH、UXTB、UXTH-40 | 无符号扩展字节、有符号扩展字节 |
| 跳转控制类 | | B、BL、BX、BLX | 跳转指令 |
| 其他指令 | | BKPT、CPSIE、CPSID、DMB、DSB、ISB、MRS、MSR、NOP、SEV、SVC、WFE、WFI | |

例如：LDR 含义是“存储器中内容加载到寄存器中”，LD 是 Load 的缩写，R 是 Register 的缩写

# 3.1.2 寻址方式

指令（Instruction）是对数据的操作，通常把指令中所要操作的数据称为操作数（Operand）。操作数可能来自：寄存器、指令代码、存储单元。而确定指令中所需操作数来自哪里的各种方法称为寻址方式（Addressing mode）。不同计算机，支持的寻址方式不相同，Arm Cortex-M支持立即数寻址方式、寄存器直接寻址方式、直接地址寻址方式、寄存器加偏移间接寻址方式等。

## 1. 立即数寻址方式（Immediate addressing mode）

操作数直接通过指令给出，数据包含在指令中，随着指令一起被汇编成机器码，存储于程序空间中。用“#”作为立即数的前导标识符，例如：

```
MOV  R0,#0xFF    //立即数0xFF装入R0寄存器
SUB  R1,R0,#1    //R1←R0-1
```

演示：通过开发环境查看机器码（用程序Exam3-1）

【练习3-1】利用AHL-GEC-IDE开发工具寻找指令的机器码（工程Exam3_1）：

```
//-------------------------------------------------------------
//主函数，一般情况下可以认为程序从此开始运行（实际上有启
main:
     MOV   R0,#0xFF
 800d840:    f04f 00ff    mov.w    r0, #255     ; 0xff
E:\13-20-BOOK\18-20-PeoplePostPress\MicrocomputerPri
     SUB    R1,R0,#1
 800d844:    f1a0 0101    sub.w    r1, r0, #1
E:\13-20-BOOK\18-20-PeoplePostPress\MicrocomputerPri
```

说明：16位机器码与32位指令机器码有差异

## 2. 寄存器直接寻址方式（Register direct addressing mode）

操作数来自于寄存器。例如：**MOV R1,R2 //R1←R2**

## 3. 直接地址寻址方式（Direct address addressing mode）

操作数来自于存储单元，指令中直接给出存储单元地址。例如：

**LDR R1,label //从label处连续读取4字节至寄存器R1中**
**LDRH R1,label //从label处连续读取2字节至寄存器R1中**
**LDRB R1,label //从label处读取1字节至寄存器R1中**

## 4. 寄存器加偏移间接寻址方式（Register plus offset indirect addressing mode）

操作数来自于存储单元，指令中通过寄存器及偏移量给出存储单元的地址，例如：

**LDR R3, [PC, #100] //从地址（PC + 100）处读取4字节到R3中**
**LDR R3,[R4] //以R4中内容为地址，读取4个字节到R3中**

# 3.2 基本指令系统

本节按照数据传送、数据操作、跳转控制、其它等四类给出Arm Cortex-M系列微处理器的基本指令系统。指令格式中的“{}”表示其中为可选项，如LDRH Rt, [Rn {, #imm}]，表示有：“LDRH Rt, [Rn ]”、“LDRH Rt, [Rn , #imm]”两种指令格式，指令中的“[ ]”表示其中内容为地址，“//”表示注释。

## 3.2.1 数据传送类指令

数据传送类指令的功能就是将数据从一个地方复制到另一个地方 。有两种情况，一是取存储器地址空间中的数传送到寄存器中，二是将寄存器中的数传送到另一寄存器或存储器地址空间中。基本数据传送类基本指令有16条。

# 1．取数指令

存储器（RAM或Flash）由地址进行表征，把存储器中内容加载（load）到CPU内部寄存器中的指令被称为取数指令

（1）~（6），辅以寻址方式形成10条具体指令，学习要点：LD是load的缩写

| 编号 | 指令 | 说明 |
|---|---|---|
| (1) | LDR Rt, [<Rn \| SP> {, #imm}] | 从{SP/Rn+ #imm}地址处，取字到 Rt，imm=0,4,8,……,1020 |
| | LDR Rt,[Rn, Rm] | 从地址 Rn+Rm 处读取字到 Rt |
| | LDR Rt, label | 从 label 指定的存储器单元取数至寄存器，label 必须在当前指令的-4~4KB 范围内，且应 4 字节对齐 |
| (2) | LDRH Rt, [Rn {, #imm}] | 从{Rn+ #imm}地址处，取半字到 Rt 中，imm=0,2,4,……,62 |
| | LDRH Rt,[Rn, Rm] | 从地址 Rn+Rm 处读取半字到 Rt |
| (3) | LDRB Rt, [Rn {, #imm}] | 从{Rn+ #imm}地址处，取字节到 Rt 中，imm=0~31 |
| | LDRB Rt,[Rn, Rm] | 从地址 Rn+Rm 处读取字节到 Rt |
| (4) | LDRSH Rt,[Rn, Rm] | 从地址 Rn+ Rm 处读取半字至 Rt，并带符号扩展至 32 位 |
| (5) | LDRSB Rt,[Rn, Rm] | 从地址 Rn+ Rm 处读取字节至 Rt，并带符号扩展至 32 位 |
| (6) | LDM Rn{!}, reglist | 从 Rn 处读取多个字加载到 reglist 列表寄存器中，每读一个字后 Rn 自增一次 |

# 1. 取数指令的说明

- 当用LDR将一个立即数或符号常量存储到寄存器时，若立即数或符号常量的值大于或等于256时，必须在该立即数或符号常量前增加“=”前缀；
- LDM Rn{!},reglist 指令
  - Rn表示存储器单元起始地址的寄存器；
  - reglist可包含一个或多个寄存器，若包含多个寄存器必须以“，”分隔，外面用“{}”标识；
  - “!”是一个可选的回写后缀， reglist列表中包含了Rn寄存器时不要回写后缀，否则须带回写后缀“!”。带后缀时，在数据传送完毕之后，将最后的地址写回到Rn=Rn+4×(n-1)，n为reglist中寄存器的个数。
  - Rn不能为R15，reglist可以为R0~R15任意组合；
  - Rn寄存器中的值必须字对齐。

# 2. 存数指令

**将CPU内部寄存器中的内容存储（store）至存储器中的指令被称为存数指令（7）~（10），辅以寻址方式形成7条具体指令，学习要点：ST是store的缩写**

| 编号 | 指令 | 说明 |
|---|---|---|
| (7) | STR Rt, [<Rn \| SP> {, #imm}] | 把 Rt 中的字存储到地址 SP/Rn+#imm，imm=0,4,8,……,1020 |
| | STR Rt, [Rn, Rm] | 把 Rt 中的字存储到地址 Rn+ Rm 处 |
| (8) | STRH Rt, [Rn {, #imm}] | 把 Rt 中的低半字存储到地址 SP/Rn+#imm，imm=0,2,4,……,62 |
| | STRH Rt, [Rn, Rm] | 把 Rt 中的低半字存储到地址 Rn+ Rm 处 |
| (9) | STRB Rt, [Rn {, #imm}] | 把 Rt 中的低字节 SP/Rn+#imm，imm=0~31 |
| | STRB Rt, [Rn, Rm] | 把 Rt 中的低字节存储到地址 Rn+ Rm 处 |
| (10) | STM Rn!, reglist | 存储多个字到 Rn 处，每存一个字后 Rn 自增一次 |

## 2. 存数指令的说明

- Rt、Rn和Rm必须为R0～R7之一
- STM Rn!, reglist指令
  - 将reglist列表寄存器内容以字存储至Rn寄存器中的存储单元地址。
  - 以4字节访问存储器地址单元，访问地址从Rn寄存器指定的地址值到Rn+4×(n-1)，n为reglist中寄存器的个数。
  - 按寄存器编号递增顺序访问，最低编号使用最低地址空间，最高编号使用最高地址空间。
  - 若reglist列表中包含了Rn寄存器，则Rn寄存器必须位于列表首位。如果列表中不包含Rn，则将位于Rn+4×n地址回写到Rn寄存器中。

# 3. 寄存器间数据传送指令

MOV指令用于CPU内部寄存器之间的数据传送，（11）~（13），共3条指令

| 编号 | 指令 | 说明 |
|---|---|---|
| (11) | MOV Rd, Rm | Rd←Rm，Rd只可以是R0~R7 |
| (12) | MOVS Rd, #imm | Rd←#imm，立即数范围是0x00~0xFF。MOV指令亦可用这种格式 |
| (13) | MVN Rd, Rm | 将寄存器Rm中数据取反，传送给寄存器Rd |

MOV指令实现跳转功能的方式：

1)MOV指令的目标寄存器Rd为程序计数器

2)传送值的第0位为0，执行MOV指令后就进入程序计数器，从而实现跳转

注意：这种跳转方式不常用

# 4. 栈操作指令

| 编号 | 指令 | 说明 |
| --- | --- | --- |
| (14) | PUSH reglist | 进栈指令，SP 递减 4，reglist 为 R0-R7，LR |
| (15) | POP reglist | 出栈指令，SP 递增 4，reglist 为 R0-R7，PC |

注意：

1)若POP指令的reglist列表中包含了PC寄存器，在POP指令执行完成后会跳转到PC所指地址处。

```
#程序片段实例
PUSH {R0,R4-R7}   //将R0,R4,R5,R6,R7寄存器值入栈
PUSH {R2,LR}      //将R2,LR寄存器值入栈
POP {R0,R6,PC}    //出栈值到R0,R6,PC中，同时程序跳转至PC所指向的地址开始执行
```

# 5．生成与指针PC相关地址指令

| 编号 | 指令 | 说明 |
| --- | --- | --- |
| (16) | ADR Rd, label | 生成与PC相关地址，将label相对于当前指令的偏移地址值与PC相加或者相减（label有前后，即负、正）写入Rd中 |

**1)ADR是将PC值加上一个偏移量得到的地址写进目标寄存器中。**
**2)若利用ADR指令将生成的目标地址用于跳转指令BX或BLX时，则必须确保该地址最后一位为1.**
**3) Rd必须为R0～R7，label的值必须字对齐且在当前PC值的1020字节以内**

# 3.2.2 数据操作类指令

数据操作主要指算术运算、逻辑运算、移位等

## 1. 算术运算类指令

算术类指令有加、减、乘、比较等

| 编号 | 指令 | 说明 |
|---|---|---|
| (17) | ADC {Rd, } Rn, Rm | 带进位加法。Rd←Rn+Rm+C，影响 N、Z、C 和 V 标志位 |
| (18) | ADD {Rd } Rn, <Rm \| #imm> | 加法。Rd←Rn+Rm，影响 N、Z、C 和 V 标志位 |
| (19) | RSB {Rd, } Rn, #0 | Rd←0-Rn，影响 N、Z、C 和 V 标志位（部分汇编环境不支持） |
| (20) | SBC {Rd, }Rn, Rm | 带借位减法。Rd←Rn-Rm-C，影响 N、Z、C 和 V 标志位 |
| (21) | SUB {Rd } Rn, <Rm \| #imm> | 常规减法。Rd←Rn-Rm/ #imm，影响 N、Z、C 和 V 标志位 |
| (22) | MUL Rd, Rn Rm | 常规乘法。Rd←Rn*Rm，同时更新 N、Z 状态标志，不影响 C、V 状态标志。该指令所得结果与操作数是否为无符号、有符号数无关。Rd、Rn、Rm 寄存器必须为 R0～R7，且 Rd 与 Rm 须一致 |
| (23) | CMN Rn, Rm | 加比较指令。Rn+Rm，更新 N、Z、C 和 V 标志，但不保存所得结果。Rn、Rm 寄存器必须为 R0～R7 |
| (24) | CMP Rn, #imm<br>CMP Rn, Rm | （减）比较指令。Rn-Rm/#imm，更新 N、Z、C 和 V 标志，但不保存所得结果。Rn、Rm 寄存器为 R0～R7，立即数 imm 范围 0～255 |

加、减指令对操作数的限制条件，见书本表3-8

# 加、减指令对操作数的限制条件

| 指令 | Rd | Rn | Rm | imm | 限制条件 |
|---|---|---|---|---|---|
| ADC | R0 ~ R7 | R0 ~ R7 | R0 ~ R7 | - | Rd和Rn必须相同 |
| ADD | R0 ~ R15 | R0 ~ R15 | R0 ~ PC | - | Rd和Rn必须相同；<br>Rn和Rm不能同时指定为PC寄存器； |
| | R0 ~ R7 | SP或PC | - | 0 ~ 1020 | 立即数必须为4的整数倍 |
| | SP | SP | - | 0 ~ 508 | 立即数必须为4的整数倍 |
| ADD | R0 ~ R7 | R0 ~ R7 | - | 0 ~ 7 | - |
| | R0 ~ R7 | R0 ~ R7 | - | 0 ~ 255 | Rd和Rn必须相同 |
| | R0 ~ R7 | R0 ~ R7 | R0 ~ R7 | - | - |
| RSB | R0 ~ R7 | R0 ~ R7 | - | - | - |
| SBC | R0 ~ R7 | R0 ~ R7 | R0 ~ R7 | - | Rd和Rn必须相同 |
| SUB | SP | SP | - | 0 ~ 508 | 立即数必须为4的整数倍 |
| SUB | R0 ~ R7 | R0 ~ R7 | - | 0 ~ 7 | - |
| | R0 ~ R7 | R0 ~ R7 | - | 0 ~ 255 | Rd和Rn必须相同 |
| | R0 ~ R7 | R0 ~ R7 | R0 ~ R7 | - | - |

# 2. 逻辑运算类指令

逻辑运算指与、或、异或、位段清零，（25）~（28）

| 编号 | 指令 | 说明 | 举例 |
|---|---|---|---|
| (25) | AND {Rd, } Rn, Rm | 按位与 | AND R2, R2, R1 |
| (26) | ORR {Rd, } Rn, Rm | 按位或 | ORR R2, R2, R5 |
| (27) | EOR {Rd, } Rn, Rm | 按位异或 | EOR R7, R7, R6 |
| (28) | BIC {Rd, } Rn, Rm | 位段清零 | BIC R0, R0, R1 |

C语言中：与运算（&）、或运算（|）、异或运算（^）
0&0=0，0&1=0，1&0=0，1&1=1，即：两位同时为“1”，结果才为“1”，否则为0
0|0=0， 0|1=1， 1|0=1， 1|1=1，即：若有一个为1，其值为1
0^0=0，1^0=1，0^1=1，1^1=0（同为0，异为1），即为：不进位加法

## 3. 移位类指令

ASR、LSL、LSR和ROR指令，（29）~（32），实际多用于乘除运算

## 4. 位测试指令

(33) TST Rn, Rm //为测试寄存器Rn某位为0或1，将Rn寄存器某位置1，其余位清零，根据结果，更新N、Z状态标志

## 5. 数据序转指令

了解，（34）~（36），用于大小端转换

## 6. 扩展类指令

了解，（37）~（40），8位、16位扩展位32位

# 3.2.3 跳转控制类指令

| 编号 | 指令 | 跳转范围 | 说明 |
|---|---|---|---|
| (41) | B{cond}　label | -256B~+254B | 转移到 Label 处对应的地址处。可以带（或不带）条件，所带条件见表 3-15，如：BEQ 表示标志位 Z=1 时转移 |
| (42) | BL　label | -16MB~+16MB | 转移到 Label 处对应的地址，并且把转移前的下条指令地址保存到 LR，并置寄存器 LR 的 Bit[0]为 1，保证了随后执行 POP {PC}或 BX 指令时成功返回分支 |
| (43) | BX　Rm | 任意 | 转移到由寄存器 Rm 给出的地址，寄存器 Rm 的 Bit[0]必须为 1，否则会导致硬件故障 |
| (44) | BLX　Rm | 任意 | 转移到由寄存器 Rm 给出的地址，并且把转移前的下条指令地址保存到 LR。寄存器 Rm 的 Bit[0]必须为 1，否则会导致硬件故障 |

# 3.2.3 跳转控制类指令

1)构成循环的主要指令
2)B指令只在前256至后254字节地址范围内跳转
3)BL用于±16MB以内相对调用子程序
4)B指令的可带条件

```
BEQ  label    //条件转移，标志位Z=1时转移到label
BL   func     //调用子程序 funC，把转移前的下条指令地址保存到LR
BX   LR       //返回到函数调用处
```

# 3.2.3 跳转控制类指令

| 条件后缀 | 标志位 | 含义 | 条件后缀 | 标志位 | 含义 |
|---|---|---|---|---|---|
| EQ | Z=1 | 相等 | HI | C=1并且Z=0 | 无符号数大于 |
| NE | Z=0 | 不相等 | LS | C=1或Z=1 | 无符号数小于或等于 |
| CS或者HS | C=1 | 无符号数大于或等于 | GE | N=V | 带符号数大于或等于 |
| CC或者LO | C=0 | 无符号数小于 | LT | N!=V | 带符号数小于 |
| MI | N=1 | 负数 | GT | Z=0并且N=V | 带符号数大于 |
| PL | N=0 | 正数或零 | LE | Z=1并且N!=V | 带符号数小于或等于 |
| VS | V=1 | 溢出 | AL | 任何情况 | 无条件执行 |
| VC | V=0 | 未溢出 | | | |

# 3.2.4 其它基本指令（了解）

| 类型 | 编号 | 指令 | 说明 |
|---|---|---|---|
| 断点指令 | (45) | BKPT #imm | 如果调试被使能，则进入调试状态（停机）。或者如果调试监视器异常被使能，则调用一个调试异常，否则调用一个错误异常。处理器忽视立即数 imm，立即数范围 0~255，表示断点调试的信息。不影响 N、Z、C、V 状态标志 |
| 中断指令 | (46) | CPSIE i | 除了 NMI，使能总中断，不影响 N、Z、C、V 标志 |
| | (47) | CPSID i | 除了 NMI，禁止总中断，不影响 N、Z、C、V 标志 |
| 屏蔽指令 | (48) | DMB | 数据内存屏蔽（与流水线、MPU 和 cache 等有关） |
| | (49) | DSB | 数据同步屏蔽（与流水线、MPU 和 cache 等有关） |
| | (50) | ISB | 指令同步屏蔽（与流水线、MPU 等有关） |

| | | | |
|---|---|---|---|
| 特殊寄存器操作指令 | (51) | MRS Rd, spec_reg1 | 加载特殊功能寄存器值到通用寄存器。若当前执行模式不为特权模式，除 APSR 寄存器外，读其余所有寄存器值为 0 |
| | (52) | MSR spe_reg, Rn | 存储通用寄存器的值到特殊功能寄存器。Rd 不允许为 SP 或 PC 寄存器，若当前执行模式不为特权模式，除 APSR 外，任何试图修改寄存器操作均被忽视。影响 N、Z、C、V 标志 |
| 空操作 | (53) | NOP | 空操作，但无法保证能够延迟时间，处理器可能在执行阶段之前就将此指令从线程中移除。不影响 N、Z、C、V 标志 |
| 发送事件指令 | (54) | SEV | 发送事件指令。在多处理器系统中，向所有处理器发送一个事件，也可置位本地寄存器。不影响 N、Z、C、V 标志 |
| 操作系统服务调用指令 | (55) | SVC #imm | 操作系统服务调用，带立即数调用代码。SVC 指令触发 SVC 异常。处理器忽视立即数 imm，若需要，该值可通过异常处理程序重新取回，以确定哪些服务正在请求。执行 SVC 指令期间，当前任务优先级高于等于 SVC 指令调用处理程序时，将产生一个错误。不影响 N、Z、C、V 标志 |
| 休眠指令 | (56) | WFE | 休眠并且在发生事件时被唤醒。不影响 N、Z、C、V 标志 |
| | (57) | WFI | 休眠并且在发生中断时被唤醒。不影响 N、Z、C、V 标志 |

# 3.3 指令集与机器码对应表（了解，资料性质）

表中给出的是16位机器码，实际汇编器可能给出32位机器码。
通过开发环境，编译，可以获得指令的机器码

# 3.4 GUN汇编器的基本语法

## 3.4.1 汇编语言概述

能够在计算机内部直接执行的指令序列是二进制描述的机器码，显示时通常为十六进制。最初，人们就用它书写程序，这就是第一代计算机语言，也就是机器语言。后来，人们为了方便记忆，用助记符号来表示机器指令，形成了汇编语言，它是介于机器语言与高级语言之间的计算机语言，被人们称为第二代计算机语言。

用汇编语言写成的程序不能直接放入计算机内部存储器中去执行，必须先转为机器语言。把用汇编语言写成的源程序“翻译”成机器语言的工具叫汇编程序或汇编器（Assembler。

汇编语言源程序的书写格式有一定规则，为了能够正确地产生目标代码以及方便汇编语言的编写，汇编器还提供了一些在汇编时使用的命令、操作符号。由于汇编器提供的指令，仅是辅助把汇编语言源程序“翻译”成机器码工作，并不产生运行阶段执行的机器码，被称为伪指令（Pseudo Instruction）。如，伪指令告诉汇编器：从哪里开始汇编，到何处结束等相关信息，它们包含在汇编源程序中，否则汇编器就难以汇编好源程序以生成正确目标代码。

# 3.4.2 GUN汇编书写格式

汇编语言源程序以行为单位进行设计，每一行最多可以包含以下四个部分：
标号： 操作码 操作数 注释

## 1. 标号（Label）

标号表示地址位置

## 2. 操作码（Opcodes）

操作码可以是指令和伪指令，指令就是3.2节介绍的，伪指令将在3.4.3节中介绍

# 3. 操作数（Operands）

操作数可以是地址、标号或指令码定义的常数，也可以是由表3-18所给出的伪运算符构成的表达式。

表3-18 Arm-GUN汇编器识别的伪运算符

| 运算符 | 功能 | 类型 | 实例 | |
|---|---|---|---|---|
| + | 加法 | 二元 | mov r3,#30+40 | 等价于 mov r3,#70 |
| - | 减法 | 二元 | mov r3,#40-30 | 等价于 mov r3,#10 |
| * | 乘法 | 二元 | mov r3,#5*4 | 等价于 mov r3,#20 |
| / | 除法 | 二元 | mov r3,#20/4 | 等价于 mov r3,#5 |
| % | 取模 | 二元 | mov r3,#20%7 | 等价于 mov r3,#6 |
| \|\| | 逻辑或 | 二元 | mov r3,#1\|\|0 | 等价于 mov r3,#1 |

注意：这里的+与指令中ADD的本质区别：+是伪指令，在汇编阶段完成，不产生机器码

# 4. 注释（Comments）

注释即说明文字，是对汇编指令的作用和功能进行解释，有助于对指令的理解，可以采用单行边注释和整行注释，建议使用“//”引导。以 “/*”开始和“*/”结束的注释保留调试时屏蔽语句行使用。

注意：整段注释、行注释、行末注释的撰写方法

# 3.4.3 GUN汇编常用伪指令

## 1. 系统预定义的段

汇编语言程序在经过汇编和链接之后，最终生成可执行文件。可执行程序是以段为单位来组织文件的，通常划分为.text、.data和.bss等段，其中，.text是只读的代码段，是程序存放的地方，一般存储在flash区；.data是可读写的数据段，而.bss则是可读写且没有初始化的数据段，启动时会清0，存储在RAM区，在链接文件中使用。

## 2. 常量的定义

常量的定义可以使用.equ或.set伪指令。

```
.equ    _NVIC_ICER,  0xE000E180    //定义常量名_NVIC_ICER=0xE000E180
LDR    R0,=_NVIC_ICER          //将常量名_NVIC_ICER的值0xE000E180放到R0中
.set  ROM_size, 128 * 1024          //定义常量ROM_size
```

# 3. 数据定义

类似高级语言可以有不同的数据类型，在汇编语言中也允许有字、半字、字节、字符串等数据类型

表3-19 数据定义伪指令

| 数据类型 | 长度 | 举例 | 备注 |
|---|---|---|---|
| .word | 字（4 字节） | .word 0x12345678 | 定义多个数据时，数据间用“,”隔开 |
| .hword | 半字（2 字节） | .hword 0x1234 | 定义多个数据时，数据间用“,”隔开 |
| .byte | 字节（1 字节） | .byte 0x12 | 定义多个数据时，数据间用“,”隔开 |
| .ascii | 字符串 | .ascii “hello\n” | 定义的字符串不以“\0”结尾，要自行添加“\0” |
| .asciz 或.string | 字符串 | .asciz “hello\n” | 定义的字符串以“\0”结尾 |

## 4. 条件伪指令

.if条件伪指令后面紧跟着一个恒定的表达式（即该表达式的值为真），并且最后要以.endif结尾。中间如果有其他条件，可以用.else填写汇编语句。

```
.ifdef 表达式     //当表达式为真时执行代码1
   代码1
.else             //否则，表示表达式为假执行代码2
代码2
.endif
```

## 5. 文件包含伪指令

类似高级语言中的文件包含一样，在汇编语言中也可以使用“.include”进行文件包含，.include是一个附加文件的链接指示命令，利用它可以把另一个源文件插入当前的源文件一起汇编，成为一个完整的源程序

# 6. 其它常用伪指令

（1）.section伪指令：用户可以通过.section伪指令来自定义一个段

格式：.section <段名>{,"<标志>"}

其中，标志可选a（允许段）、w（可写段）和x（执行段）

（2）.global伪指令可以用来定义一个全局符号

（3）.extern伪指令

格式：.extern symbol，声明symbol为外部函数

（4）.align伪指令使当前位置满足一定的对齐方式

（5）.end伪指令：.end伪指令声明汇编文件的结束

还有有限循环、宏定义和宏调用等伪指令，需要深入了解的读者，可以参见《GNU汇编语法》。

作业5：

1、给出把存储器的一个地址（label1）中的数据存储到另一个地址（label2）的指令代码。

2、给出指令代码判断r1第3位是否为0，若为0转到label1。

3、若r2=80000001H，r3=90000005H，N、Z、C和V标志位的值都为0，给出以下指令的执行后r2的结果，并指出N、Z、C和V标志位的值。

（1）ADC　r2,r3　　（2）SUB　r2,r3　　（3）LSR　r2,r3,#3

4、若sp=20002000H，r1=1234H，r2=5678H，r3=9ABCH，r4=DEF0，试说明执行下面指令后，sp=?，r5=？，r6=？并画图指出堆栈中各单元的内容。

```
push {r1-r4}
pop {r5,r6}
```

作业提交网址：见群文件

作业文件命名规则(word文档)：学号+姓名

说明：如果手写，可以拍照后插入到word文档提交