第5章

Thumb 指令

在 ARM 体系结构中，ARM 指令是 32 位的，具有很高的执行效率。但是对于嵌入式设备而言，其存储空间极其有限，由于每条 ARM 指令都要占用 4 字节，对存储空间的要求较高。为了压缩代码的存储，增加代码存储密度，ARM 公司设计了 16 位的 Thumb 指令。因此在程序状态寄存器中有一个 T 标志位，用来标志处理器运行的指令类型。当处理器在执行 ARM 程序段时，称 ARM 处理器处于 ARM 工作状态；当处理器在执行 Thumb 程序段时，称 ARM 处理器处于 Thumb 工作状态。

严格来讲，Thumb 指令集是 ARM 指令集的一个子集，所有的 Thumb 指令都有对应的 ARM 指令，在应用程序的编写过程中，只要遵循一定调用的规则，Thumb 子程序和 ARM 子程序就可以互相调用。

编程时，可根据任务需求和存储资源合理地选择应用哪种类型的指令：若对系统的性能有较高要求，应使用 32 位的存储系统和 ARM 指令集；若对系统的成本及功耗有较高要求，则应使用 16 位的存储系统和 Thumb 指令集。本章将对 Thumb 指令做详细介绍。

5.1 Thumb 数据处理指令

Thumb 数据处理操作包括数据传送、算术运算、逻辑运算、移位操作和比较操作等。具体的 Thumb 数据处理指令如表 5-1 所示。

表 5-1 Thumb 数据处理指令

指令类型	汇编语法格式	操作
数据传送指令	MOV Rd,#expr	Rd←expr，Rd 为 R0～R7
	MOV Rd,Rm	Rd←Rm，Rd、Rm 均可为 R0～R7
数据非传送指令	MVN Rd,Rm	Rd←(～Rm)，Rd、Rm 均为 R0～R7
数据取负指令	NEG Rd,Rm	Rd←(－Rm)，Rd、Rm 均为 R0～R7

续表

指 令 类 型	汇编语法格式	操　　作
加法运算指令	ADD Rd,Rn,Rm	Rd←Rn+Rm,Rd、Rn、Rm 为 R0～R7
	ADD Rd,Rn,#expr3	Rd←Rn+expr3,Rd、Rn 均为 R0～R7
	ADD Rd,#expr8	Rd←Rd+expr8,Rd 为 R0～R7
	ADD Rd,Rm	Rd←Rd+Rm,Rd、Rm 均可为 R0～R15
SP/PC 加法运算指令	ADD Rd,Rp,#expr	Rd←SP+expr 或 PC+expr,Rd 为 R0～R7
SP 加法运算指令	ADD SP,#expr	SP←SP+expr
减法运算指令	SUB Rd,Rn,Rm	Rd←Rn−Rm,Rd、Rn、Rm 均为 R0～R7
	SUB Rd,Rn,#expr3	Rd←Rn−expr3,Rd、Rn 均为 R0～R7
	SUB Rd,#expr8	Rd←Rn−expr8,Rd 为 R0～R7
SP 减法运算指令	SUB SP,#expr	SP←SP−expr
带进位加法指令	ADC Rd,Rm	Rd←Rd+Rm+Carry,Rd、Rm 为 R0～R7
带进位减法指令	SBC Rd,Rm	Rd←Rd−Rm−(NOT)Carry,Rd、Rm 为 R0～R7
乘法运算指令	MUL Rd,Rm	Rd←Rd * Rm,Rd、Rm 为 R0～R7
逻辑“与”操作指令	AND Rd,Rm	Rd←Rd & Rm,Rd、Rm 为 R0～R7
逻辑“或”操作指令	ORR Rd,Rm	Rd←Rd | Rm,Rd、Rm 为 R0～R7
逻辑“异或”操作指令	EOR Rd,Rm	Rd←Rd ^ Rm,Rd、Rm 为 R0～R7
位清除指令	BIC Rd,Rm	Rd←Rd &(～Rm),Rd、Rm 为 R0～R7
算术右移指令	ASR Rd,Rs	Rd←Rd 算术右移 Rs 位,Rd、Rs 为 R0～R7
	ASR Rd,Rm,#expr	Rd←Rm 算术右移 expr 位,Rd、Rm 为 R0～R7
逻辑左移指令	LSL Rd, Rs	Rd←Rd≪Rs, Rd、Rs 为 R0～R7
	LSL Rd, Rm, #expr	Rd←Rm≪expr, Rd、Rm 为 R0～R7
逻辑右移指令	LSR Rd, Rs	Rd←Rd≫Rs, Rd、Rs 为 R0～R7
	LSR Rd, Rm, #expr	Rd←Rm≫expr, Rd、Rm 为 R0～R7
循环右移指令	ROR Rd, Rs	Rd←Rd 循环右移 Rs 位,Rd、Rs 为 R0～R7
比较指令	CMP Rn, Rm	标识状态←Rn−Rm, Rn、Rm 均可为 R0～R7
	CMP Rn, #expr	标识状态←Rn−expr, Rn 为 R0～R7
负数比较指令	CMN Rn, Rm	标识状态←Rn+Rm, Rn、Rm 为 R0～R7
位测试指令	TST Rn, Rm	标识状态←Rn & Rm, Rn、Rm 为 R0～R7

5.1.1　寄存器移位指令

Thumb 指令将移位操作作为独立的指令进行操作，可以对寄存器内容直接进行移位操作。

这类指令的汇编语法格式为：

opcode Rd,Rs,#shift_immed_5

Thumb 寄存器移位指令 16 位编码格式如下：

15	14	13	12　11	10　　6	5　　3	2　　0
0	0	0	opcode	shift_immed_5	Rs	Rd

其中:

Rd:目标寄存器;

Rs:源操作数寄存器;

shift_immed_5:寄存器移位的数值,取值范围为 0～31;

opcode:指令操作码,其编码类型如表 5-2 所示。

表 5-2 Thumb 寄存器移位指令操作码编码类型

opcode	Thumb 汇编语法格式		实现的操作
0 0	LSL	Rd,Rs,＃shift_immed_5	将 Rs 逻辑左移 shift_immed_5 位,并将结果传送到 Rd 中
0 1	LSR	Rd,Rs,＃shift_immed_5	将 Rs 逻辑右移 shift_immed_5 位,并将结果传送到 Rd 中
1 0	ASR	Rd,Rs,＃shift_immed_5	将 Rs 算术右移 shift_immed_5 位,并将结果传送到 Rd 中

注意事项:

- Rd、Rs 必须在 R0～R7 中选取。
- 指令执行结果影响 N、Z、C 标志位。

【例 5-1】 指令功能解析。

```
LSR R2, R5, ＃10   ; 将 R5 逻辑右移 10 位,并将结果传送到 R2 中,并影响 N、Z、C 标志位
LSL R0, R3, ＃8    ; 将 R3 逻辑左移 8 位,并将结果传送到 R0 中,并影响 N、Z、C 标志位
ASR R1, R2, ＃3    ; 将 R2 算术右移 3 位,并将结果传送到 R1 中,并影响 N、Z、C 标志位
```

5.1.2 低位寄存器算术运算指令

低位寄存器算术运算指令所操作的寄存器选取范围必须在 R0～R7 中选取。

1. 加法与减法运算指令(对 R0～R7 操作)

这类指令的汇编语法格式为:

```
opcode  Rd,  Rs,  Rm
opcode  Rd,  Rs,  ＃immed_3
```

Thumb 加法减法运算指令 16 位编码格式如下:

15	14	13	12	11	10	9	8 6	5 3	2 0
0	0	0	1	1	I	opcode	Rm/immed_3	Rs	Rd

其中:

Rd:目标寄存器;

Rs:源操作数寄存器;

Rm:第二操作数寄存器;

immed_3:第二操作数为立即数;

I：决定第二操作数是寄存器还是立即数；

opcode：指令操作码，其编码类型如表 5-3 所示。

表 5-3　Thumb 加法与减法运算指令操作码编码类型

opcode	I	Thumb 汇编语法格式	实现的操作
0	0	ADD　Rd,Rs,Rm	Rd←Rs+Rm
0	1	ADD　Rd,Rs,#immed_3	Rd←Rs+immed_3
1	0	SUB　Rd,Rs,Rm	Rd←Rn−Rm
1	1	SUB　Rd,Rs,#immed_3	Rd←Rn−immed_3

注意事项：

- Rd、Rs、Rm 必须在 R0～R7 中选取。
- 指令执行结果影响 N、Z、C、V 标志位。

【例 5-2】 指令功能解析。

```
ADD R0, R3, R2      ; R0←R3 + R2,指令执行结果影响 N、Z、C、V 标志位
SUB R1, R2, #4      ; R1←R2 − 4,指令执行结果影响 N、Z、C、V 标志位
```

2. MOV、CMP、ADD 与 SUB 指令（对 R0～R7 操作）

这类指令的汇编语法格式为：

```
opcode Rd, # immed_8
```

Thumb MOV、CMP、ADD 与 SUB 指令 16 位编码格式如下：

15	14	13	12　11	10　8	7　0
0	0	0	opcode	Rd	immed_8

其中：

immed_8：寄存器移位的数值；

Rd：目标寄存器；

opcode：指令操作码，其编码类型如表 5-4 所示。

表 5-4　Thumb MOV、CMP、ADD 与 SUB 指令操作码编码类型

opcode	Thumb 汇编语法格式	实现的操作
0　0	MOV　Rd,# immed_8	Rd←immed_8
0　1	CMP　Rd,# immed_8	标识状态←Rd−immed_8
1　0	ADD　Rd,# immed_8	Rd←Rd+immed_8
1　1	SUB　Rd,# immed_8	Rd←Rn−immed_8

注意事项：

- Rd 必须在 R0～R7 中选取。
- MOV 指令执行结果影响 N、Z 标志位，CMP、ADD、SUB 指令执行结果影响 N、Z、C、V 标志位。

【例 5-3】 指令功能解析。

```
MOV   R1, #50       ; R0←50,指令执行结果影响标志位
CMP   R2, #0x30     ; 根据 R2 - 0x30 的结果来影响标志位
ADD   R1, #0xAA     ; R1←R1 + 0xAA,指令执行结果影响标志位
SUB   R6, #0xAA     ; R6←R6 - 0xAA,指令执行结果影响标志位
```

5.1.3 ALU 操作指令

这类指令的汇编语法格式为:

```
opcode Rd, Rs
```

对应的 16 位 Thumb 指令编码格式如下:

15	14	13	12	11	10	9 ~ 6	5 ~ 3	2 ~ 0
0	0	0	0	0	0	opcode	Rs	Rd

其中:

Rd:目标寄存器;

Rs:源操作数寄存器;

opcode:指令操作码,其编码类型如表 5-5 所示。

表 5-5 Thumb ALU 操作指令操作码编码类型

opcode	Thumb 汇编语法格式	实现的操作
0 0 0 0	AND Rd,Rs	Rd←Rd & Rs
0 0 0 1	EOR Rd,Rs	Rd←Rd ^ Rs
0 0 1 0	LSL Rd,Rs	Rd←Rd << Rs
0 0 1 1	LSR Rd,Rs	Rd←Rd >> Rs
0 1 0 0	ASR Rd,Rs	Rd←Rd 算术右移 Rs 位
0 1 0 1	ADC Rd,Rs	Rd←Rd+Rs+Carry
0 1 1 0	SBC Rd,Rs	Rd←Rd-Rs-(NOT)Carry
0 1 1 1	ROR Rd,Rs	Rd←Rd 循环右移 Rs 位
1 0 0 0	TST Rd,Rs	标识状态←Rd & Rs
1 0 0 1	NEG Rd,Rs	Rd←(-Rs)
1 0 1 0	CMP Rd,Rs	标识状态←Rn-Rs
1 0 1 1	CMN Rd,Rs	标识状态←Rn+Rs
1 1 0 0	ORR Rd,Rs	Rd←Rd | Rs
1 1 0 1	MUL Rd,Rs	Rd←Rd * Rs
1 1 1 0	BIC Rd,Rs	Rd←Rd & (~Rs)
1 1 1 1	MVN Rd,Rs	Rd←(~Rs)

注意事项:

- Rd、Rs 必须在 R0~R7 中选取。
- AND、EOR、ORR、MUL、BIC、MVN 指令执行结果影响 N、Z 标志位。
- LSL、LSR、ASR、ROR 指令执行结果影响 N、Z、C 标志位。
- ADC、SBC、TST、NEG、CMP、CMN 指令执行结果影响 N、Z、C、V 标志位。

【例 5-4】　指令功能解析。

```
EOR    R3, R4      ; R3←R3 EOR R4,指令执行结果影响标志位
ROR    R1, R0      ; R1←R1 循环右移 R0 位,指令执行结果影响标志位
MUL    R0, R7      ; R0←R7 * R0,指令执行结果影响标志位
NEG    R5, R3      ; R5←(－R3),指令执行结果影响标志位
CMP    R2, R6      ; 根据 R2 － R6 来影响标志位
```

5.1.4　带高位寄存器操作的 Thumb 指令

带高位寄存器操作的 Thumb 指令所操作的寄存器可以在 R8～R15 中选取。这类指令的汇编语法格式为：

```
opcode    Rd, Hs
opcode    Hd, Rs
opcode    Hd, Hs
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10	9　8	7	6	5　3	2　0
0	0	0	0	0	0	opcode	H1	H2	Rs/Hs	Rd/Hd

其中：

Rd：目标寄存器，寄存器在 R0～R7 中选取；

Hd：目标寄存器；寄存器在 R8～R15 中选取；

Rs：源操作数寄存器；寄存器在 R0～R7 中选取；

Hs：源操作数寄存器；寄存器在 R8～R15 中选取；

opcode：指令操作码，其编码类型如表 5-6 所示。

表 5-6　带高位寄存器操作的 Thumb 指令操作码编码类型

opcode	H1	H2	Thumb 汇编语法格式	实现的操作
0　0	0	1	ADD　Rd,Hs	Rd←Rd＋Hs
0　0	1	0	ADD　Hd,Rs	Hd←Hd＋Rs
0　0	1	1	ADD　Hd,Hs	Hd←Hd＋Hs
0　1	0	1	CMP　Rd,Hs	影响标识状态←Rd－Hs
0　1	1	0	CMP　Hd,Rs	影响标识状态←Hd－Rs
0　1	1	1	CMP　Hd,Hs	影响标识状态←Hd－Hs
1　0	0	1	MOV　Rd,Hs	Rd←Hs
1　0	1	0	MOV　Hd,Rs	Hd←Rs
1　0	1	1	MOV　Hd,Hs	Hd←Hs

注意事项：

- 带高位寄存器的 Thumb 中，ADD、MOV 指令不影响标志位。
- CMP 指令执行结果影响 N、Z、C、V 标志位。

【例 5-5】 指令功能解析。

```
ADD   PC, R1      ; PC←PC + R1,指令执行结果不影响标志位
MOV   R13, R14    ; R13←R14,指令执行结果不影响标志位
CMP   R4, R12     ; 根据 R4 - R12 来影响标志位
CMP   PC, R12     ; 根据 PC - R12 来影响标志位
```

5.1.5 带 SP/PC 的算术运算指令

带 SP/PC 的算术运算指令的操作数中包含 SP 或 PC,分为两种类型:SP/PC 加法运算指令、SP 加法/减法运算指令。

1. SP/PC 加法运算指令

这类指令的汇编语法格式为:

```
ADD   Rd,   PC,#immed_8 × 4
ADD   Rd,   SP,#immed_8 × 4
```

对应的 16 位 Thumb 指令编码格式如下:

15	14	13	12	11	10 … 8	7 … 0
1	0	1	0	X	Rd	immed_8

其中:

immed_8:偏移量,它是一个无符号的立即数表达式,取值范围是 0～255;

Rd:目标寄存器;

X:区别 SP、PC 操作,其编码类型如表 5-7 所示。

表 5-7 Thumb SP/PC 加法运算指令操作码编码类型

X	Thumb 汇编语法格式	实现的操作
0	ADD Rd,PC,#immed_8 × 4	Rd←PC+ immed_8 × 4
1	ADD Rd,SP,#immed_8 × 4	Rd←SP+ immed_8 × 4

注意事项:

- 寄存器 Rd 必须在 R0～R7 中选取;
- 指令执行结果不影响标志位。

【例 5-6】 指令功能解析。

```
ADD   R1,   PC,#0x40   ; R1←PC + 0x40,指令执行结果不影响标志位
ADD   R1,   SP,#576    ; R1←SP +576,指令执行结果不影响标志位
```

2. SP 加法/减法运算指令

这类指令的汇编语法格式为:

```
ADD   SP,#immed_7 × 4
SUB   SP,#immed_7 × 4
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10	9	8	7	6	0
1	0	1	1	0	0	0	0	S	immed_7	

其中：

SP：目标寄存器；

immed_7：偏移量，它是一个无符号的立即数表达式，取值范围是 0～127；

S：用于区别 ADD、SUB 操作，其编码类型如表 5-8 所示。

表 5-8　Thumb SP 加法/减法运算指令操作码编码类型

S	Thumb 汇编语法格式	实现的操作
0	ADD SP，＃immed_7 × 4	SP←PC＋immed_7 × 4
1	SUB SP，＃immed_7 × 4	SP←SP－immed_7 × 4

注意事项：

指令执行结果不影响标志位。

【例 5-7】 指令功能解析。

```
ADD   SP,＃0x40   ；SP←SP＋0x40，指令执行结果不影响标志位
SUB   SP,＃0x20   ；SP←SP－0x20，指令执行结果不影响标志位
```

5.2　Thumb 存储器操作指令

Thumb 存储器操作指令包括无符号字节、半字和字数据的存储与加载，多寄存器的加载与存储指令和栈操作指令。对存储器操作的 Thumb 指令如表 5-9 所示。

表 5-9　对存储器操作的 Thumb 指令

汇编语法格式	说　明	操　作
LDR　Rd,[Rn,＃immed_5×4]	加载字数据	Rd←[Rn,＃immed_5×4]，Rd、Rn 为 R0～R7
LDRH　Rd,[Rn,＃immed_5×2]	加载无符号半字数据	Rd←[Rn,＃immed_5×2]，Rd、Rn 为 R0～R7
LDRB　Rd,[Rn,＃immed_5×1]	加载无符号字节数据	Rd←[Rn,＃immed_5×1]，Rd、Rn 为 R0～R7
STR　Rd,[Rn,＃immed_5×4]	存储字数据	[Rn,＃immed_5×4]←Rd，Rd、Rn 为 R0～R7
STRH　Rd,[Rn,＃immed_5×2]	存储无符号半字数据	[Rn,＃immed_5×2]←Rd，Rd、Rn 为 R0～R7
STRB　Rd,[Rn,＃immed_5×1]	加载无符号字节数据	[Rn,＃immed_5×1]←Rd，Rd、Rn 为 R0～R7
LDR　Rd,[Rn,Rm]	加载字数据	Rd←[Rn,Rm]，Rd、Rn、Rm 为 R0～R7

续表

汇编语法格式	说　明	操　作
LDRH　Rd,[Rn,Rm]	加载无符号半字数据	Rd←[Rn,Rm],Rd、Rn、Rm 为 R0～R7
LDRB　Rd,[Rn,Rm]	加载无符号字节数据	Rd←[Rn,Rm],Rd、Rn、Rm 为 R0～R7
LDRSH　Rd,[Rn,Rm]	加载有符号半字数据	Rd←[Rn,Rm],Rd、Rn、Rm 为 R0～R7
LDRSB　Rd,[Rn,Rm]	加载有符号字节数据	Rd←[Rn,Rm],Rd、Rn、Rm 为 R0～R7
STR　Rd,[Rn,Rm]	存储字数据	[Rn,Rm]←Rd,Rd、Rn、Rm 为 R0～R7
STRH　Rd,[Rn,Rm]	存储无符号半字数据	[Rn,Rm]←Rd,Rd、Rn、Rm 为 R0～R7
STRB　Rd,[Rn,Rm]	存储无符号字节数据	[Rn,Rm]←Rd,Rd、Rn、Rm 为 R0～R7
LDR　Rd,[Pc,#immed_8×4]	基于 PC 加载字数据	Rd←[PC,#immed_8×4],Rd 为 R0～R7
LDR　Rd,label	基于 PC 加载字数据	Rd←[label],Rd 为 R0～R7
LDR　Rd,[SP,#immed_8×4]	基于 SP 加载字数据	Rd←[SP,#immed_8×4],Rd 为 R0～R7
STR　Rd,[SP,#immed_8×4]	基于 SP 存储字数据	[SP,#immed_8×4]←Rd,Rd 为 R0～R7
LDMIA　Rn{!},reglist	多寄存器加载	Reglist←[Rn…],Rn 回写等(R0～R7)
STMIA　Rn{!},reglist	多寄存器存储	[Rn…]←reglist,Rn 回写等(R0～R7)
PUSH　{reglist,LR}	寄存器入栈指令	[SP…]←reglist,LR,SP 回写等(R0～R7、LR)
POP　{reglist,PC}	寄存器出栈指令	reglist,PC←[SP…],SP 回写等(R0～R7、PC)

5.2.1　字节、半字和字的加载/存储指令

1. 立即数偏移量的加载/存储指令

这类指令的汇编语法格式为：

```
LDR{B}    Rd, [Rn,    #immed_5 × 4]
STR{B}    Rd, [Rn,    #immed_5 × 4]
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10 … 6	5 … 3	2 … 0
0	1	1	B	L	immed_5	Rn	Rd

其中：

Rd：目标/源寄存器；

Rn：基址寄存器；

immed_5：偏移量，它是一个无符号的立即数表达式，取值范围是 0～31；

L：区分是 LDR 指令还是 STR 指令；

B：区分存储的是无符号的字节数据还是字数据。

根据 L、B 的编码的不同，所对应的汇编指令如表 5-10 所示。

表 5-10　立即数偏移量的加载/存储指令

L	B	Thumb 汇编语法格式	实现的操作
0	0	STR　Rd,[Rn,#immed_5 × 4]	[Rn+immed_5×4]←Rd
1	0	LDR　Rd,[Rn,#immed_5 × 4]	Rd←[Rn+immed_5×4]字数据单元
0	1	STRB　Rd,[Rn,#immed_5 × 1]	[Rn+immed_5×1]←Rd 低 8 位
1	1	LDRB　Rd,[Rn,#immed_5 × 1]	Rd←[Rn+immed_5×1]字节数据单元

注意事项：

- Rd、Rn 寄存器在 R0～R7 中选取。
- Thumb 立即数偏移量的加载/存储指令不影响标志位。

【例 5-8】 指令功能解析。

```
LDR R2,[R5,#116]  ；将内存地址 R5+116 开始的字数据单元加载到 R2 中
LDRB R0,[R3,#8]   ；将内存地址为 R3+8 的字节数据单元加载到 R0 中,高 24 位清零
STR R1,[R0,#13]   ；将 R1 存储到内存地址为 R0+13 开始的字数据单元
STRB R1,[R0,#13]  ；将 R1 的低 8 位存储到内存地址为 R0+13 的字节数据单元
```

2. 寄存器偏移量的加载/存储指令

寄存器偏移量的加载/存储指令的汇编语法格式为：

```
LDR{B}  Rd,  [Rn,  Rm]
STR{B}  Rd,  [Rn,  Rm]
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10	9	8　6	5　3	2　0
0	1	0	1	B	L	0	Rm	Rn	Rd

其中：

Rd：目标/源寄存器；

Rn：基址寄存器；

Rm：偏移量寄存器；

L：区分是 LDR 指令还是 STR 指令；

B：区分存储的是无符号的字节数据还是字数据。

根据 L、B 的编码的不同，所对应的汇编指令如表 5-11 所示。

表 5-11　Thumb 寄存器偏移量的加载/存储指令

L	B	Thumb 汇编语法格式	实现的操作
0	0	STR　Rd,[Rn,Rm]	[Rn+Rm]←Rd
1	0	LDR　Rd,[Rn,Rm]	Rd←[Rn+Rm]字数据单元
0	1	STRB　Rd,[Rn,Rm]	[Rn+Rm]←Rd 低 8 位
1	1	LDRB　Rd,[Rn,Rm]	Rd←[Rn+Rm]字节数据单元

注意事项：

- Rd、Rn、Rm 寄存器在 R0～R7 中选取。
- Thumb 寄存器偏移量的加载/存储指令不影响标志位。

【例 5-9】 指令功能解析。

```
LDR R2,[R5,R0]      ;将内存地址 R5+R0 开始的字数据单元加载到 R2 中
LDRB R0,[R3,R1]     ;将内存地址为 R3+R1 的字节数据单元加载到 R0 中
STR R1,[R0,R2]      ;将 R1 存储到内存地址为 R0+R2 开始的字数据单元
STRB R1,[R0,R2]     ;将 R1 的低 8 位存储到内存地址为 R0+R2 的字节数据单元
```

3. 有符号字节/半字的加载/存储指令

有符号字节/半字的加载/存储指令的汇编语法格式为:

```
LDRH   Rd,    [Rn,  Rm]
STRH   Rd,    [Rn,  Rm]
LDRSB  Rd,    [Rn,  Rm]
LDRSH  Rd,    [Rn,  Rm]
```

对应的 16 位 Thumb 指令编码格式如下:

15	14	13	12	11	10	9	8 6	5 3	2 0
0	1	0	1	H	S	1	Rm	Rn	Rd

其中:

Rd:目标/源寄存器;

Rn:基址寄存器;

Rm:偏移量寄存器;

S:区分是对有符号数操作还是对无符号数操作;

H:区分是对有符号字节数据操作还是对半字数据操作。

根据 S、H 的编码的不同,所对应的汇编指令如表 5-12 所示。

表 5-12 Thumb 有符号字节/半字的加载/存储指令

S	H	Thumb 汇编语法格式	实现的操作
0	0	STRH Rd,[Rn,Rm]	[Rn+Rm]←Rd 的低 16 位
0	1	LDRH Rd,[Rn,Rm]	Rd←[Rn,Rm] 半字数据单元,Rd 高 16 位清零
1	0	LDRSB Rd,[Rn,Rm]	Rd←[Rn,Rm] 字节数据单元,Rd 高 24 位扩展为符号位
1	1	LDRSH Rd,[Rn,Rm]	Rd←[Rn,Rm] 半字数据单元,Rd 高 16 位扩展为符号位

注意事项:

- Rd、Rn、Rm 寄存器在 R0~R7 中选取。
- 指令执行不影响标志位。

【例 5-10】 指令功能解析。

```
LDRH R2,[R5,R0]     ;将内存地址 R5+R0 开始的半字数据单元加载到 R2 中,R2 的高 16 位清零
LDRSH R0,[R3,R1]    ;将内存地址为 R3+R1 的半字数据单元加载到 R0 中,R0 的高 16 位用
                    ;符号位扩展
STRH R1,[R0,R2]     ;将 R1 的低 16 位存储到内存地址为 R0+R2 开始的半字数据单元
LDRSB R1,[R0,R2]    ;将内存地址为 R0+R2 的字节数据单元加载到 R1 中,R1 的高 24 位用
                    ;符号位扩展
```

4. 偏移量为立即数的半字加载/存储指令

偏移量为立即数的半字加载/存储指令的汇编语法格式为：

```
LDRH Rd,    [Rn,#immed_5 × 2]
STRH Rd,    [Rn,#immed_5 × 2]
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10 … 6	5 … 3	2 … 0
0	1	0	0	L	immed_5	Rn	Rd

其中：

Rd：目标/源寄存器；

Rn：基址寄存器；

immed_5：偏移量，它是一个无符号的立即数表达式，取值范围是 0～31；

L：区分是 LDR 指令还是 STR 指令。

根据 L 的编码的不同，所对应的汇编指令如表 5-13 所示。

表 5-13　偏移量为立即数的半字操作指令

L	Thumb 汇编语法格式	实现的操作
0	STRH　Rd,[Rn,# immed_5 × 2]	[Rn+immed_5 × 2]←Rd 的低 16 位
1	LDRH　Rd,[Rn,#immed_5 × 2]	Rd←[Rn,immed_5 × 2] 半字数据单元，Rd 高 16 位清零

注意事项：

- Rd、Rn 寄存器在 R0～R7 中选取。
- 指令执行不影响标志位。

【例 5-11】　指令功能解析。

```
LDRH R2,[R5, #0x10]      ; 将内存地址 R5+0x10 开始的半字数据单元加载到 R2 中
                         ; R2 的高 16 位清零
STRH R1,[R0, #0x20]      ; 将 R1 的低 16 位存储到内存地址为 R0+0x20 开始的半字
                         ; 数据单元
```

5. PC 作为基址寄存器的字加载指令

PC 作为基址寄存器的加载指令的汇编语法格式为：

```
LDR Rd,[PC,# immed_8×4]
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10 … 8	7 … 0
0	0	0	0	0	Rd	immed_8

其中：

immed_8：偏移量，它是一个 8 位的无符号的立即数表达式，取值范围是 0～255；

Rd：目标/源寄存器；

功能描述：将内存地址 PC + immed_8×4 为基地址的字数据单元加载到 Rd 中。

注意事项：

- Rd 寄存器在 R0～R7 中选取。
- 指令执行不影响标志位。

【例 5-12】 指令功能解析。

```
LDR R2,[PC, #0x10]          ;将内存地址 PC + 0x10 开始的字数据单元加载到 R2 中
LDR R0,[R15, #0x40]         ;将内存地址 PC + 0x40 开始的字数据单元加载到 R0 中
```

6. SP 作为基址寄存器的加载/存储指令

SP 作为基址寄存器的加载/存储指令的汇编语法格式为：

```
LDR    Rd,[SP,# immed_8×4]
STR    Rd,[SP,# immed_8×4]
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10 8	7 0
1	0	0	1	L	Rd	immed_8

其中：

immed_8：偏移量，它是 8 位的一个无符号的立即数表达式，取值范围是 0～255；

Rd：目标/源寄存器；

L：区分是 LDR 指令还是 STR 指令。

根据 L 的编码的不同，所对应的汇编指令如表 5-14 所示。

表 5-14 SP 作为基址寄存器的加载/存储指令

L	Thumb 汇编语法格式	实现的操作
0	STR Rd,[SP,#immed_8 × 2]	[SP+immed_8 × 4]←Rd
1	LDR Rd,[SP,#immed_8 × 2]	Rd←[SP+immed_8 × 4] 字数据单元

注意事项：

- Rd 寄存器在 R0～R7 中选取。
- 指令执行不影响标志位。

【例 5-13】 指令功能解析。

```
LDR R2,[SP, #0x10]          ;将内存地址 SP+0x10 开始的字数据单元加载到 R2 中
STR R1,[SP, #0x20]          ;将 R1 存储到内存地址为 SP+0x20 开始的字数据单元
```

5.2.2 批量加载/存储指令

1. 寄存器入栈出栈指令

ARM 处理器进行异常处理服务或子程序调用时，可使用寄存器入栈指令 PUSH 和寄存器出栈指令 POP 来保存和恢复现场信息。堆栈地址由 SP 寄存器设置，堆栈是满递减类

型。这类指令的汇编语法格式为：

```
PUSH  {  reglist  }
POP   {  reglist  }
PUSH  {  reglist, LR}
POP   {  reglist, PC}
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10	9	8	7 … 0
1	0	1	1	L	1	0	R	reglist

其中：

reglist：入栈/出栈寄存器列表；

R：程序计数器 PC 和链接寄存器 LR 是否出现在寄存器列表中，由指令编码的该位来决定；

L：区分是 PUSH 指令还是 POP 指令。

根据 L、R 的编码的不同，所对应的汇编指令如表 5-15 所示。

表 5-15　Thumb 寄存器入栈、出栈指令

L	R	Thumb 汇编语法格式	实现的操作
0	0	PUSH　{ reglist }	将 reglist 中的寄存器顺序入栈，并更新 SP
			即[SP…]←reglist，SP 回写
0	1	PUSH　{ reglist，LR }	将 reglist 中的寄存器和 LR 顺序入栈，并更新 SP 即[SP…]←reglist，LR，SP 回写
1	0	POP　{ reglist }	将堆栈中的数据顺序弹出到 reglist 中的寄存器中，并更新 SP。即 reglist←[SP…]，SP 回写
1	1	POP　{ reglist，PC }	将堆栈中的数据顺序弹出到 reglist 中的寄存器和 PC 中，并更新 SP。即 reglist，PC←[SP…]，SP 回写

注意事项：

- reglist 为低编号寄存器(R0～R7)的全部或子集，编号低的寄存器对应内存低地址单元。
- 指令执行不影响标志位。

【例 5-14】 指令功能解析。

```
PUSH  {R0-R3,R5}        ; 将寄存器 R0、R1、R2、R3、R5 压栈
POP   {R0-R3,R5}        ; 将堆栈中的数据弹出到寄存器 R0、R1、R2、R3、R5 中
PUSH  {R0-R3,R5,LR}     ; 将寄存器 R0、R1、R2、R3、R5 和 LR 顺序压栈
POP   {R0-R3,R5,PC}     ; 将堆栈中的数据弹出到寄存器 R0、R1、R2、R3、R5 和 PC 中
```

2. 多寄存器加载/存储指令

在 Thumb 多寄存器加载/存储指令中，只使用后增的形式(IA)，可以将低编号寄存器(R0～R7)的全部或子集存储到内存中，还可以将内存中的数据加载到低编号寄存器(R0～R7)的全部或子集中。这类指令的汇编语法格式为：

```
LDMIA   Rn!   {   reglist   }
STMIA   Rn!   {   reglist   }
```

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10 … 8	7 … 0
1	1	0	0	L	Rn	reglist

其中：

reglist：入栈/出栈寄存器列表；

L：区分是 LDM 指令还是 STM 指令。

根据 L 的编码的不同，所对应的汇编指令如表 5-16 所示。

表 5-16　Thumb 多寄存器加载/存储指令

L	Thumb 汇编语法格式	实现的操作
0	STMIA Rn!,　{ reglist }	将 reglist 中的寄存器以后增(IA)的方式顺序存储到以 Rn 为基地址的内存单元中，并更新 Rn。即[Rn…]←reglist，Rn 回写
1	LDMIA Rn!,　{ reglist }	将以 Rn 为基地址的内存单元数据以后增(IA)的方式顺序加载到 reglist 中的寄存器中，并更新 Rn。即 reglist←[Rn…]，Rn 回写

注意事项：

- reglist 为低编号寄存器(R0～R7)的全部或子集，编号低的寄存器对应内存低地址单元。
- 指令执行不影响标志位。

【例 5-15】 指令功能解析。

```
STMIA   R4!,{R0-R3,R5}   ; 将 R0、R1、R2、R3、R5 中的数据以后增的方式顺序
                         ; 存储到以 R4 为基地址的内存单元中，并更新 R4
LDMIA   R4!,{R0-R3,R5}   ; 将以 R4 为基地址的内存单元数据以后增的方式顺序
                         ; 加载到 R0、R1、R2、R3、R5 中，并更新 R4
```

5.3　Thumb 分支指令

与 ARM 指令一样，在 Thumb 指令集中，也有相应的分支指令实现程序的跳转和子程序的调用。Thumb 分支指令可分为 B 分支指令、带链接的分支指令和带状态切换的分支指令。

5.3.1　B 分支指令

在 Thumb 指令中，B 分支指令又分为条件分支指令和无条件分支指令。

1. 条件分支指令

当指令执行的条件成立时，条件分支指令跳转到指定的地址执行程序。这是 Thumb 指令中唯一的有条件执行指令。这类指令的汇编语法格式为：

B{cond}　label

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11　　8	7　　0
1	1	0	1	cond	signed_immed_8

其中：

signed_immed_8：8 位有符号的立即数，由汇编语法格式中的 label 右移一位得到；

cond：指令执行的条件码。

根据 cond 的编码的不同，所对应的标志位及描述如表 5-17 所示。

表 5-17　Thumb 分支指令对应的标志位及描述

条件码	条件码助记符	描　　述	PSR 中的标志位
0000	EQ	相等	Z=1
0001	NE	不相等	Z=0
0010	CS	无符号大于或等于	C=1
0011	CC	无等号小于	C=0
0100	MI	负数	N=1
0101	PL	非负数	N=0
0110	VS	上溢出	V=1
0111	VC	没有上溢出	V=0
1000	HI	无符号数大于	C=1 且 Z=0
1001	LS	无符号小于或等于	C=0 或 Z=1
1010	GE	有符号数大于或等于	N=1 且 V=1 或 N=0 且 V=0
1011	LT	有符号数小于	N=1 且 V=0 或 N=0 且 V=1
1100	GT	有符号数大于	Z=0 且 N=V
1101	LE	有符号数小于/等于	Z=1 或 N!=V

【例 5-16】　指令功能解析。

```
        CMP   R0,#0x10
        BEQ   stop          ；当 R0 等于 16 时，程序跳转到 stop 标号处
        …
stop    …
        …
```

2. 无条件分支指令

这类指令的汇编语法格式为：

B　label

对应的 16 位 Thumb 指令编码格式如下：

15	14	13	12	11	10　　0
1	1	1	0	0	signed_immed_11

其中,signed_immed_11 为 11 位有符号的立即数,由汇编语法格式中的 label 右移一位得到,程序可以跳转为−2KB～2KB。

【例 5-17】 指令功能解析。

```
        B      stop      ;程序无条件跳转到 stop 标号处
        …
stop:   …
        …
```

5.3.2 带链接的分支指令

带链接的分支指令 BL 将下一条指令的地址拷贝到 R14(LR)链接寄存器中,然后跳转到指定的地址运行程序。指令的汇编语法格式为:

```
BL   label
```

对应的 16 位 Thumb 指令编码格式如下:

15	14	13	12	11	10	…	0
1	1	1	1	H	immed_11		

其中:

immed_11:11 位立即数,作为长跳转的高 11 位/低 11 位;

H:区别+/−4MB 范围的高 11 位偏移或低 11 位偏移,其对应的汇编指令如表 5-18 所示。

表 5-18 Thumb 带链接的分支指令

H	Thumb 汇编语法格式	实现的操作
0		LR←PC+高 11 位偏移量≪12
	BL label	PC←LR+低 11 位偏移量≪1
1		LR←下一条指令地址|1

由于 BL 指令通常需要大的地址范围,无法用一条 16 位的指令格式来实现,因此,Thumb 指令采用两条指令来实现此功能。具体做法是:两条指令中的 immed_11 组合成 22 位的偏移量,并有符号扩展为 32 位,使指令转移范围为−4MB～+4MB。

注意事项:

- BL 指令为无条件执行指令。
- 带链接的分支指令 BL 提供了一种在 Thumb 状态下程序间相互调用的方法,当从子程序返回时,通常使用下面的方式之一:

```
MOV   PC,LR
BX    LR
POP   {PC}
```

5.3.3　带状态切换的分支指令

带状态切换的分支指令的汇编语法格式为：

```
BX   Rs
BX   Hs
```

Thumb 寄存器移位指令 16 位编码格式如下：

15	14	13	12	11	10	9	8	7	6	5　　3	2		0
0	1	0	0	0	1	1	1	0	H	Rs/Hs	0	0	0

其中：

Rs：源操作数寄存器；寄存器在 R0～R7 中选取；

Hs：源操作数寄存器；寄存器在 R7～R15 中选取；

H：区分源操作数寄存器是高编号寄存器还是低编号寄存器，其对应的汇编指令如表 5-19 所示。

表 5-19　Thumb 带状态切换的分支指令

H2	Thumb 汇编语法格式	实现的操作
0	BX　Rs	PC←Rs，切换处理器状态
1	BX　Hs	PC←Hs，切换处理器状态

注意事项：

如果 R[1:0]=0b10，处理器切换为 ARM 状态时，指令的执行结果不可预知。

【例 5-18】　指令功能解析。

```
        ADR   R1, go_to_ARM
        BX    R1      ; 程序跳转到 go_to_ARM 标号处，处理器切换到 ARM 状态
        …
        ALIGN
        CODE32
go_to_ARM
        …
```

5.4　Thumb 软中断指令

Thumb 软中断指令 SWI 与 ARM 指令集下的软中断指令相似，用于使处理器产生软件异常，使用这种机制实现在用户模式下对操作系统中特权模式的程序调用。SWI 指令的汇编语法格式为：

```
SWI     immed_8
```

SWI 指令 16 位编码格式如下：

15	14	13	12	11	10	9	8	7	0
1	1	0	1	1	1	1	1	immed_8	

其中，immed_8 是 8 位无符号的立即数，它是软中断的请求号。

注意事项：

- 进行 SWI 软中断时，处理器切换到 ARM 状态，并处于管理模式，CPSR 保存到管理模式的 SPSR 中，程序跳转到地址 0x08 处(SWI 中断向量地址)。
- 在执行 SWI 时，immed_8 被处理器忽略，但在指令编码中，它用来确定所请求的服务号。
- SWI 指令执行不影响条件标志位。

【例 5-19】 指令功能解析。

```
SWI   0x10   ；产生软中断，所请求的软中断服务号为 16
```

5.5 Thumb 指令功能码段分析

5.5.1 Thumb 与 ARM 实现功能比较

1. 实现乘法的功能段

1) 与 2^n(1,2,4,8,…)相乘

```
Thumb     ：LSL   Ra，Rb，LSL ＃n
ARM       ：MOV   Ra，Rb，LSL ＃n
```

2) 与 2^n+1(3,5,9,17,…)相乘

```
Thumb     ：LSL Rt，Rb，＃n
            ADD Ra，Rt，Rb
ARM       ：ADD Ra，Rb，Rb，LSL ＃n
```

3) 与 2^n-1(3,7,15,…) 相乘

```
Thumb     ：LSL Rt，Rb，＃n
            SUB Ra，Rt，Rb
ARM       ：RSB Ra，Rb，Rb，LSL ＃n
```

4) 与-2^n(－2，－4，－8，…)相乘

```
Thumb     ：LSL Ra，Rb，＃n
            MVN Ra，Ra
ARM       ：MOV Ra，Rb，LSL ＃n
           RSB Ra，Ra，＃0
```

5）与 -2^n+1（-1，-3，-7，-15，…）相乘

```
Thumb    ：LSL Rt，Rb，＃n
          SUB Ra，Rb，Rt
ARM      ：SUB Ra，Rb，Rb，LSL ＃n
```

2. 实现除法的功能段

下面的 ARM 代码实现除法运算，R0 用来存放被除数，R1 用来存放除数，R2 用来存放两数除法运算结果的商，R3 用来存放两数除法运算结果的余数。

```
        MOV       R0，＃0x0F      ；（4 字节）
        MOV    R1，＃0x06         ；（4 字节）
        MOV    R2，＃0x0          ；（4 字节）
Loop
        SUBS   R0，R0，R1         ；（4 字节）
        ADDGE R2，R2，＃1         ；（4 字节）
        BGE       Loop            ；（4 字节）
        ADD       R3，R0，R1      ；（4 字节）
```

上面程序代码的 7 条 ARM 指令占用的存储空间大小为

$$7\times 4\text{ 字节}=28\text{ 字节}$$

如果用 Thumb 代码来实现相同的功能，程序代码如下：

```
        MOV       R0，＃0x0F      ；（2 字节）
        MOV    R1，＃0x06         ；（2 字节）
        MOV    R2，＃0x0          ；（2 字节）
Loop
        ADD       R2，＃1         ；（2 字节）
        SUB     R0，R1            ；（2 字节）
        BGE       Loop            ；（2 字节）
        SUB     R2，＃1           ；（2 字节）
        ADD       R3，R0，R1      ；（2 字节）
```

上面程序代码的 8 条 Thumb 指令占用的存储空间大小为 8×2 字节 $=16$ 字节，可见 Thumb 代码在存储密度上高于 ARM 代码。

5.5.2　Thumb 与 ARM 性能比较

Thumb 指令具有较高的代码密度，因为 Thumb 指令的长度为 16 位，即只用 ARM 指令一半的位数来实现同样的功能。但是，要实现特定的程序功能，一般所需的 Thumb 指令的条数较 ARM 指令多。在一般的情况下，Thumb 指令与 ARM 指令的时间效率和空间效率关系为：

- Thumb 代码所需的存储空间为 ARM 代码的 60%～70%。
- Thumb 代码使用的指令数比 ARM 代码多 30%～40%。
- 若使用 32 位的存储器，ARM 代码比 Thumb 代码快约 40%。
- 若使用 16 位的存储器，Thumb 代码比 ARM 代码快 40%～50%。
- 与 ARM 代码相比较，使用 Thumb 代码，存储器的功耗会降低约 30%。

在实际应用中,一般将二者结合起来,也就是 ARM、Thumb 混合编程,充分发挥各自的优势。

思考与练习题

1. 与 32 位的 ARM 指令集相比较,16 位的 Thumb 指令集具有哪些优势?

2. Thumb 指令可分为哪几类? Thumb 指令有条件执行指令吗? 如果有,请说明哪些指令是条件执行的。

3. 分析下面的 Thumb 指令程序代码,指出程序所完成的功能。

```
.global _start
.text
.equ    num,  20
_start:
.arm
    MOV     SP, #0x400
    ADR     R0, Thumb_start + 1
    BX      R0
.thumb
Thumb_start:
    ASR     R2, R0, #31
    EOR     R0, R2
    SUB     R3, R0, R2
stop:
    B     stop
.end
```

4. 用多种方法实现将寄存器 R0 中的数据乘以 10。

5. 带链接的分支指令 BL 提供了一种在 Thumb 状态下程序间相互调用的方法,当从子程序返回时,可以采用哪几种返回方式?

6. 指出下列的 Thumb 程序代码所完成的功能:

```
ASR   R0, R1, #31
EOR   R1, R0
SUB   R1, R0
```