第 7 章 指令系统

作业：习题 2(4)、2(9)、3、6、7、8、10、11、13、15、17

2.（4）哪些寻址方式下的操作数在寄存器中？哪些寻址方式下的操作数在存储器中？

【分析解答】

寄存器直接寻址的操作数在寄存器中。

寄存器间接、直接、间接、基址、变址、相对这几种寻址方式的操作数都在存储器中。

2.（9）转移跳转和调用指令的区别是什么？返回指令是否需要有地址码字段？

【分析解答】

转移（跳转）指令执行后，CPU 将跳转到目标指令地址中执行，而调用指令执行后，其返回地址（即调用指令的下条指令的地址）会保存到栈中或特定的寄存器中，然后跳转到被调用过程第一条指令（目标指令）处执行，因此，被调用过程执行结束时会执行一条返回指令，返回指令将取出返回地址并置入 PC，从而使 CPU 返回到调用指令处执行。如果返回地址存放在栈中或特定的寄存器中，则返回指令中可以不需要地址码；如果返回地址存放在某个通用寄存器中，则返回指令中需要给出通用寄存器编号（地址码）。

3 假定某计算机中有一条转移指令，采用相对寻址方式，共占两个字节，第一字节是操作码，第二字节是相对位移量（用补码表示），CPU 每次从内存只能取一个字节。假设执行到某转移指令时 PC 的内容为 258，执行该转移指令后要求转移到 220 开始的一段程序执行，则该转移指令第二字节的内容应该是多少？

【分析解答】

因为该计算机的 CPU 每次从内存只能取一个字节，因而它采用字节编址方式。执行到该转移指令时 PC 的内容为 258，因此取出第一字节的操作码后，PC 的内容为 259，取出第二字节的位移量后，PC 的内容为 260。在执行该转移指令计算转移目标地址时，PC 应该已经是 260 了。假定位移量为 x，则根据转移目标地址（220）= PC（260）+相对位移量（x），可知 x=220−260=−40，用补码表示为 1101 1000B=D8H。

6.某计算机指令系统采用定长指令字格式，指令字长 16 位，每个操作数的地址码长 6 位。指令分二地址、单地址和零地址三类。若二地址指令有 k2 条，无地址指令有 k0 条，则单地址指令最多有多少条？

【分析解答】

假设单地址指令有 k1 条，则 $((16-k2)\times 2^6-k1)\times 2^6=k0$，所以 $k1=(16-k2)\times 2^6-k0/2^6$

7.某计算机字长 16 位，存储器存取宽度为 16 位，即每次从存储器取出 16 位。CPU 中有 8 个 16 位通用寄存器。现为该机设计指令系统，要求指令长度为字长的整数倍，至多支持 64 种不同操作，每个操作数都支持 4 种寻址方式：立即（I）、寄存器直接（R）、寄存器间接（S）和变址（X）寻址方式。存储器地址位数和立即数均为 16 位，任何一个通用寄存器都可作变址寄存器，支持以下 7 种二地址指令格式（R、I、S、X 代表上述 4 种寻址方式）：RR 型、RI 型、RS 型、RX 型、XI 型、SI 型、SS 型。请设计该指令系统的 7 种指令格式，给出每种格式的指令长度、各字段所占位数和含义，并说明每种格式指令的功能以及需要的访存次数？

【分析解答】

因为至多有 64 种操作，所以操作码字段只需要 6 位；有 8 个通用寄存器，所以寄存器编号至少占 3 位；寻址方式有 4 种，所以寻址方式位至少占 2 位；直接地址和立即数都是 16 位；任何通用寄存器都可作变址寄存器，所以指令中要明显指定变址寄存器，其编号占 3 位；指令总位数是 16 的倍数。此外，指令格式应尽量规整，指令长度应尽量短。按照上述这些要求设计出的指令格式可以有很多种。

以下是采用二地址指令格式的两种指令格式设计方案，RI、XI 和 SI 三种指令格式中添了 3 个 0，是为了补足位数，以使指令长度为 16 的倍数。这两种方案得到的 RR、RS 和 SS 型指令都是 16 位，RI、RX 和 SI 型指令都是 32 位，XI 型指令是 48 位。

指令格式示例 1： 如图 1 所示，用专门的“类型”字段（最左 3 位）说明不同指令类型，这样无需对两个操作数的寻址方式分别进行说明。7 种指令类型只要 3 位编码即可，之后的一位总是 0，这一位在需要扩充指令操作类型时可作为 OP 中新的编码位。

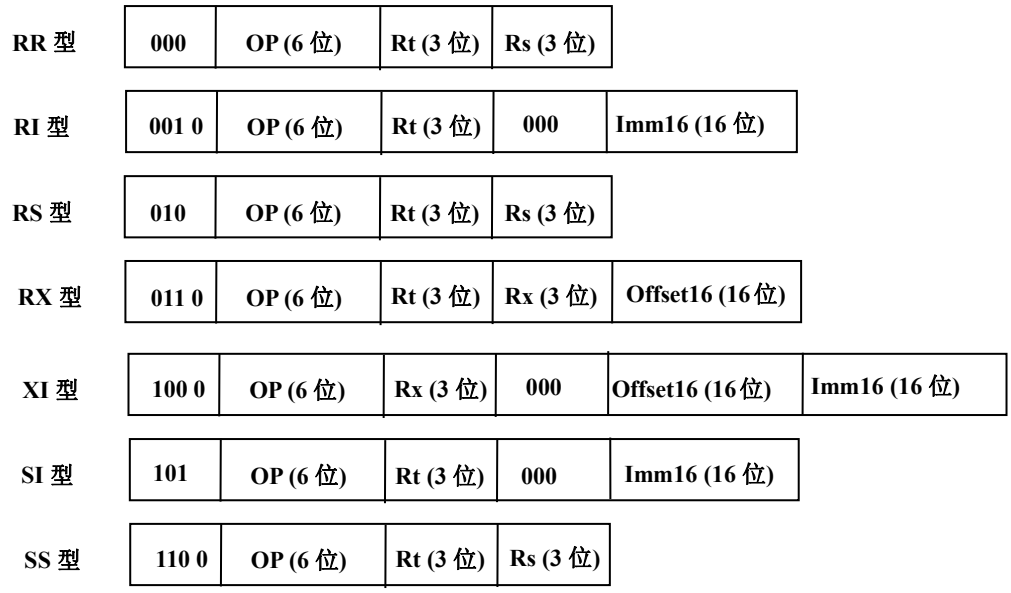


图 1　第一种指令格式示例

指令格式示例 2： 如图 2 所示，用专门的“寻址方式”字段分别说明两个操作数的寻址方式。其定义为 00-立即；01-寄直；10-寄间；11-变址。这种格式相当于用 4 位编码来说明指令格式，比第一种指令格式多用了一位编码，因此可扩展性没有第一种指令格式好。

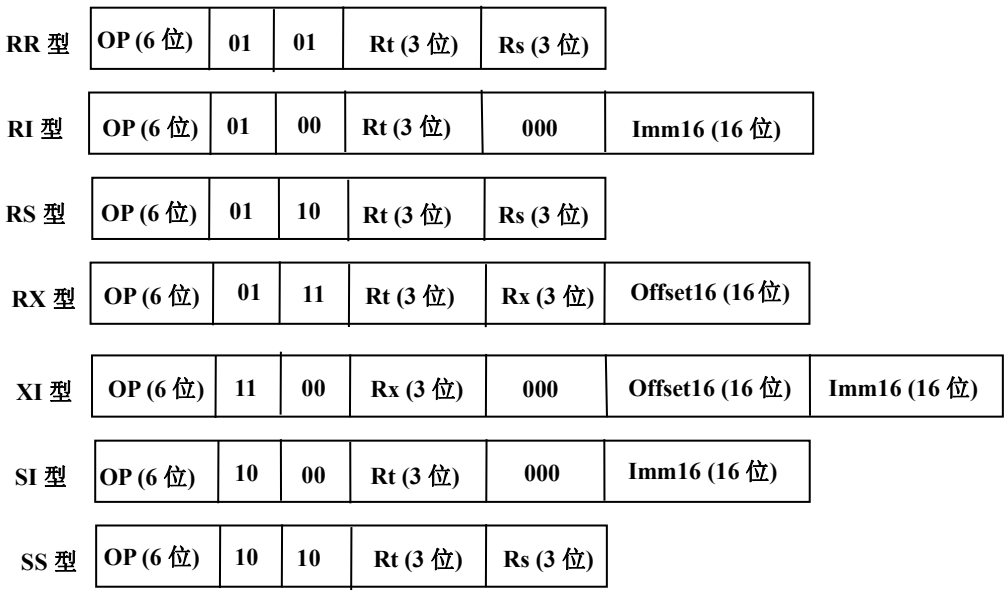


图 2　第二种指令格式示例

存储器存取宽度为 16 位，每次从存储器取出 16 位。因此，读取 16、32 和 48 位指令分别需要 1、2 和 3 次存储器访问。各类指令的功能和访存次数分别说明如下（指令功能用 RTL 表示，其中 M[x]表示存储器地址 x 中的内容，R[x]表示寄存器 x 中的内容）。

RR 型指令功能为 R[Rt]←R[Rt] op R[Rs]，取指令时访存 1 次；

RI 型指令功能为 R[Rt]←R[Rt] op Imm16，取指令时访存 2 次；

RS 型指令功能为 R[Rt]←R[Rt] op M[R[Rs]]，取指令和取第 2 个源操作数各访存 1 次，共访存 2 次；

RX 型指令功能为 R[Rt]←R[Rt] op M[R[Rx]+Offset]，取指令访存 2 次，取第 2 个源操作数访存 1 次，共访存 3 次；

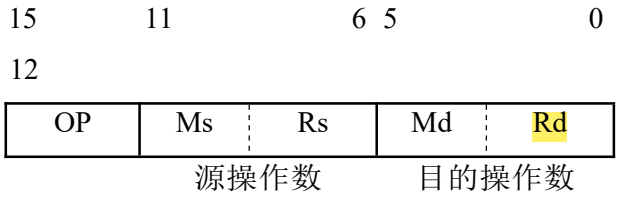
XI 型功能为 M[R[Rx]+Offset]←M[R[Rx]+Offset] op Imm16，取指令访存 3 次，取第一个源操作数访存 1 次，写结果访存 1 次，共访存 5 次；

SI 型指令功能为 M[R[Rt]]←M[R[Rt]] op Imm16，取指令访存 2 次，取第一个源操作数和写结果各访存 1 次，共访存 4 次；

SS 型功能为 M[R[Rt]]←M[R[Rt]] op M[R[Rs]]，取指令访存 1 次，取第一个源操作数、取第二

个源操作数和写结果各访存 1 次，共访存 4 次。

8.某计算机字长为 16 位，主存地址空间大小为 128 KB，按字编址。采用单字长定长指令格式，指令各字段定义如下：



转移指令采用相对寻址方式，相对偏移量用补码表示。寻址方式定义如表 7.4 所示。

表 7.4 题 8 中定义的寻址方式及其含义

Ms / Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=R[Rn]
001B	寄存器间接	(Rn)	操作数=M[R[Rn]]
010B	寄存器间接、自增	(Rn)+	操作数=M[R[Rn]], R[Rn]←R[Rn]+1
011B	相对	D(Rn)	转移目标地址=PC+R[Rn]

（注：M[x]表示存储器地址 x 中的内容，R[x]表示寄存器 x 中的内容）

请回答下列问题：

（1）该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器（MAR）和存储器数据寄存器（MDR）至少各需要多少位？

（2）转移指令的目标地址范围是多少？

（3）若操作码 0010B 表示加法操作（助记符为 add），寄存器 R4 和 R5 的编号分别为 100B 和 101B，R4 的内容为 1234H，R5 的内容为 5678H，地址 1234H 中的内容为 5678H，地址 5678H 中的内容为 1234H，则汇编语句“add (R4), (R5)+”（逗号前为第一源操作数，逗号后为第二源操作数和目的操作数）对应的机器码是什么（用十六进制表示）？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？

【分析解答】

（1） 因为采用单字长指令格式，操作码字段占 4 位，所以最多有 16 条指令；指令中通用寄存器编号占 3 位，所以，最多有 8 个通用寄存器；因为地址空间大小为 128KB，按字编址，故共有 64K 个存储单元，因而地址位数为 16 位，所以 MAR 至少为 16 位；因为字长为 16 位，所以 MDR 至少为 16 位。

（2）因为地址位数和字长都为 16 位，所以 PC 和通用寄存器位数均为 16 位，两个 16 位数据相加其结果也为 16 位，即转移目标地址位数为 16 位，因而能在整个地址空间转移，即目标转移地址的范围为 0000H～FFFFH。

（3）要得到汇编语句“add (R4),(R5)+”对应的机器码，只要将其对应的指令代码各个字段拼接起来即可。显然，add 对应 op 字段，为 0010B；（R4）的寻址方式字段为 001B，R4 的编号为 100B；(R5)+的寻址方式字段为 010B，R5 的编号为 101B；因此，对应的机器码为 0010 001 100 010 101B，用十六进制表示为 2315H。指令“add (R4), (R5)+”的功能为：M[R5]←M[R[R5]]+M[R[R4]]，R[R5]←R[R5]+1。已知 R[R4]=1234H， R[R5] =5678H，M[1234H]=5678H，M[5678H]=1234H，因为 1234H +5678H=68ACH，所以 5678H 单元中的内容从 1234H 改变为 68ACH，同时 R5 中的内容从 5678H 变为 5679H。

10.对于远距离的过程调用，使用伪指令“call offset”作为调用指令，它对应以下两条真实指令：

```
auipc x1, offset[31:12]+offset[11]      # R[x1]←PC+ (offset[31:12]+offset[11]) <<12
jalr x1, x1, offset[11:0]               # PC←R[x1]+offset[11:0]，R[x1]←PC+4
```

请说明为什么在 auipc 指令中高 20 位的位移量计算时 offset[31:12]需要加上 offset[11]?

【分析解答】

因为上述 jalr 指令进行加法运算时，需要对 x1 寄存器中的 PC+(offset[31:12]+offset[11]) <<12 与 offset[11:0]符号扩展结果进行相加。

若 offset[11:0]的最高位 offset[11]（看成是低 12 位数的符号位）是 0，则 PC 所加的高 20 位应该是 offset[31:12]+0；若 offset[11]是 1，则 offset[11:0]符号扩展后高 20 位为全 1，此时，PC 所加的高 20 位应为 offset[31:12]+1。综上所述，PC 所加的高 20 位应该为 offset[31:12]+offset[11]。

11.除了硬件乘法器外，还可以用移位和加法指令来实现乘法运算。在乘以较小的常数时，这种办法很有效。在不考虑溢出的情况下，假设要将 t0 的内容与 7 相乘，乘积存入 t1 中。请写出一段指令条数最少且不包括乘法指令的 RV32I 代码。

【分析解答】

一个数 x 乘以 6，相当于 8x-x，而 8x 可以通过将 x 左移 3 位来实现，这样就只要用一条左移指令和一条减法指令来实现乘 7 操作。以此类推，当乘以一个较小的常数时，只要将这个较小的常数分解成若干个 2 的幂次相加/减，就可以用若干条左移指令和一条加/减法指令来实现乘法运算。可用以下指令序列实现题目要求。

```
sll    t1,   t0,   3          #将t0的内容左移3位，送t1
sub    t1,   t1,   t0         #将t1和t0的内容相减，送t1
```

13. 以下程序段是某个过程对应的指令序列。入口参数 int a 和 int b 分别置于 a0 和 a1 中，返回参数是该过程的结果，置于 a0 中。要求为以下 RV32I 指令序列加注释，并简单说明该过程的功能。

```
        add   t0, zero, zero
loop:   beq   a1, zero, finish
        add   t0, t0, a0
        addi  a1, a1, -1
        j     loop                # “jal x0, loop”指令的伪指令
finish: addi  t0, t0, 100
        add   a0, t0, zero
```

【分析解答】

```
        add   t0, zero, zero      #将寄存器 t0 置 0
loop:   beq   a1, zero, finish    #若 a1 的值等于 0，则转 finish 处
        add   t0, t0, a0          #将 t0 和 a0 的内容相加，送 t0
        addi  a1, a1, -1          #将 a1 的值减 1
        j     loop                #无条件转到 loop 处
finish: addi  t0, t0, 100         #将 t0 的内容加 100
        add   a0, t0, zero        #将 t0 的内容送 a0
```

该过程的功能是计算“a×b+100”。

15. 假定编译器将 a 和 b 分别分配到 t0 和 t1 中，用一条 RV32I 指令或最短的 RV32I 指令序列实现以下 C 语言语句：b=31&a。如果把 31 换成 65535，即 b=65535&a，则用 RV32I 指令或指令序列如何实现？

【分析解答】

只要用一条指令“andi t1, t0, 31”就可实现 b=31&a，其中 12 位立即数为 0000 0001 1111。但是，如果把 31 换成 65535，则不能用一条指令“andi t1, t0, 65535”来实现，因为 65535 =1111 1111 1111 1111B，它不能用 12 位立即数表示。可用以下 3 条指令实现 b=65535&a。

```
lui   t1, 16          #将 0001 0000H 置于寄存器 t1
addi  t1, t1, 4095    #将 0001 0000H 与 FFFF FFFFH 相加，送 t1
and   t1, t0, t1      #将 t0 和 t1 的内容进行“与”运算，送 t1
```

17.请说明 RV32I 中 beq 指令的含义，并解释为什么汇编程序在对下列汇编语言源程序中的 beq 指令进行汇编时会遇到问题，应该如何修改该程序段？

```
here:   beq   t0, t2, there
        ……
there:  addi  t1, a0, 4
```

【分析解答】

在 RV32I 指令系统中，分支指令 beq 是 B-型指令，转移目标地址采用相对寻址方式，其偏移量为 imm[12:1]乘 2，相当于在 imm[12:1]后面添一个 0，再符号扩展为 32 位，即转移目标地址=PC+SEXT[imm[12:1]<<1]。

12 位立即数用补码表示，得到偏移量 offset，从而计算转移目标地址，因此 beq 指令执行时若条件满足，则可能跳转到当前指令前，也可能跳转到当前指令后，当 offset 的范围为 0000 0000 0000 0000 0B ~ 0111 1111 1111 1111 0B 时，beq 指令向后（正）跳；当 offset 的范围为 1000 0000 0000 0000 0B ~ 1111 1111 1111 1111 0B 时，beq 指令向前（负）跳。

当上述指令序列中 here 和 there 表示的地址相差超过上述 offset 的范围时，会因为无法得到正确的立即数而使得 beq 指令发生汇编错误。

可将上述指令序列改成以下指令序列。因为无条件跳转指令 j 的跳转范围足够大，所以可以直接从 here 跳转到 there。

```
here:    bne   t0, t2, skip
         j     there
skip:    ……
there:   addi  t1, a0, 4
```