作业：习题 2(4)、2(9)、3、6、7、8、9、12、14、15、16、17、19

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. 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。

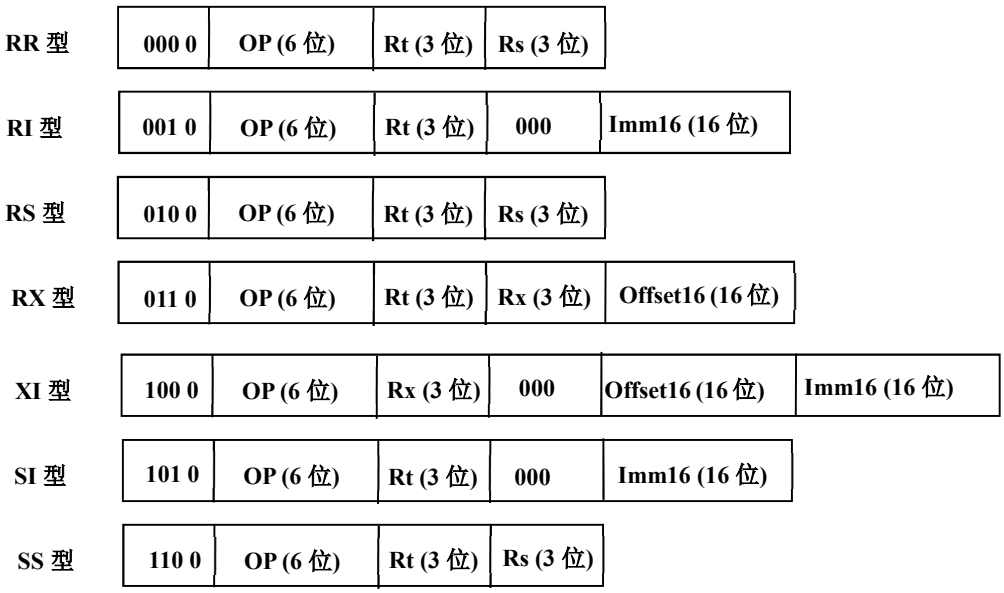


图 1　第一种指令格式示例

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。

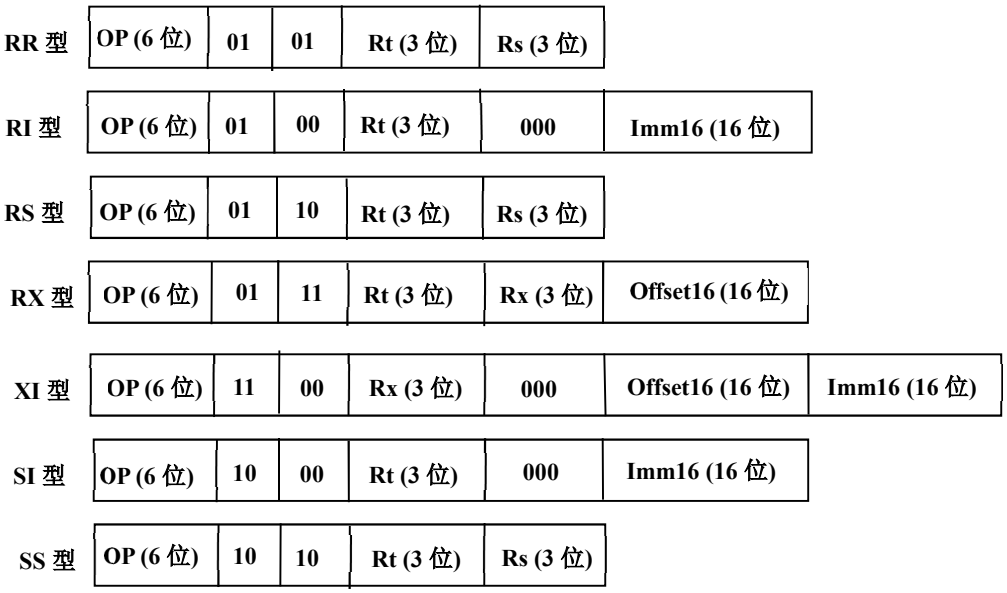


图 2　第二种指令格式示例

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；

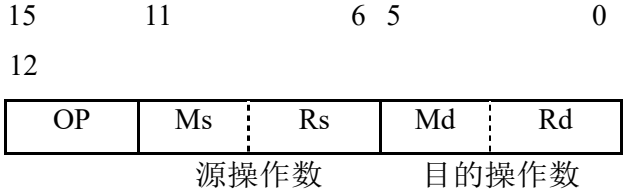
XI 型功能为 M[R[Rx]+Offset]←M[R[Rx]+Offset] op Imm16，取指令访存 3 次，取第一个源操作数访存 1 次，写结果访存 1 次，共访存 5 次；

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。

9. 假定 A 是一个 32 位的地址，A_upper20 和 A_lower12 分别表示地址 A 的高 20 和低 12 位，以下 RV32I 指令用来将存放在存储器地址 A 处的机器数读入寄存器 t1 中。

```
lui   t0, A_upper20_adjusted #将 A_upper20_adjusted 的低位添加 12 个 0，送 t0
xori t0, t0, A_lower12       #将 A_lower12 高 20 位符号扩展后与 t0 异或，送 t0
lw    t1, 0(t0)              #将 t0 内容和 0 相加得到有效地址，从中取数送 t1
```

上述第一条指令中 A_upper20_adjusted 的值是如何由 A_upper20 得到的？

上述功能也能用以下两条 RV32I 指令来实现。请问以下第一条指令中的 A_upper20_adjusted 的值又是如何得到的？

```
lui   t0,   A_upper20_adjusted
lw    t1,   A_lower12(t0)      #将 A_lower12 进行符号扩展后，和 t0 的内容相加，
                               #得到有效地址，从中取数送 t1
```

【分析解答】

因为 RV32I 指令中的立即数只有 20 位或 12 位，所以一个 32 位的地址无法用一条指令直接传送到一个 32 位寄存器中。此外，RV32I 中逻辑运算指令的 12 位立即数采用的是符号扩展方式。

对于第一种实现方案，xori 指令中的立即数采用符号扩展。若 A_lower12 的最高位（看成是低 12 位数的符号位）是 0，则 A_upper20_adjusted 就等于 A_upper20，这样，A_lower12 符号扩展后高 20 位为全 0，和高 20 位 A_upper20 异或后，高 20 位还是 A_upper20；若 A_lower12 的最高位是 1，则 A_lower12 符号扩展后高 20 位为全 1，

此时，A_upper20_adjusted 应该等于 A_upper20 的各位取反。这样，A_upper20_adjusted 的每一位与 1 异或后，就等于 A_upper20。

第二种方案中取数指令 lw 的偏移量是 A 的低 12 位 A_lower12，由于取数指令 lw 在计算内存单元地址时对偏移量采用的是符号扩展，所以要使得高 20 位最终的结果为 A_upper20，必须对 A_upper20 进行以下调整：若 A_lower12 的最高位（看成是低 12 位数的符号位）是 0，则 A_upper20_adjusted = A_upper20，这样，A_lower12 符号扩展后高 20 位为全 0，和高 20 位 A_upper20 相加后，高 20 位还是 A_upper20；若 A_lower12 的最高位是 1，则 A_lower12 符号扩展后高 20 位为全 1，即 FFFFFH，此时，A_upper20_adjusted 应满足：FFFFFH+A_upper20_adjusted= A_upper20。而因为 FFFFFH+A_upper20+1 = A_upper20（最高位向前面的进位被丢弃），所以，A_upper20_adjusted = A_upper20+1。

12. 有些计算机提供了专门的指令，能从 32 位寄存器中抽取其中任意一个位串置于另一个寄存器的低位有效位上，并在高位补 0，如图 7.17 所示。RISC-V 指令系统中没有这样的指令，请写出最短的一个 RV32I 指令序列来实现这个功能，要求 i=5, j=22, 操作前后的寄存器分别为 t1 和 t2。

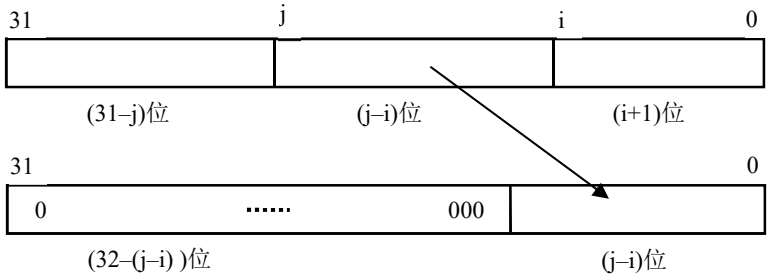


图 4.18 题 12 图

【分析解答】

可以先左移 9 位，然后右移 15 位，指令序列为：

```
sll   t2,  t1,  9        #将 t1 的内容左移 9 位，送 t2
srl   t2,  t2,  15       #将 t2 的内容右移 15 位，送 t2
```

这里要注意，第二条指令不能用算术右移指令 sra，因为算术右移高位添加的是符号，所以，不能保证高位一定补 0。此外，第一条指令中的目的操作数寄存器和第二条指令的源操作数寄存器都只能用 t2，而不能改成其他寄存器，否则，会破坏其他寄存器的内容！

14. 下列指令序列用来对两个数组进行处理，并产生结果存放在 a0 中，两个数组的基地址分别存放在 a0 和 a1 中，数组长度分别存放在 a2 和 a3 中。要求为以下每条 RV32I 指令加注释，并写出该过程对应的 C 语言程序段，且说明该 C 程序中的变量和寄存器之间的对应关系。假定每个数组有 2500 个字，其数组下标为 0 到 2499，该指令序列运行在一个时钟频率为 2GHz 的处理器上，add、addi 和 slli 指令的 CPI 为 1；lw 和 bne 指令的 CPI 为 2，则最坏情况下运行该段指令所需时间是多少秒？

```
        slli   a2, a2, 2
        slli   a3, a3, 2
        add    t5, zero, zero
        add    t0, zero, zero
outer:  add    t4, a0, t0
        lw     t4, 0(t4)
        add    t1, zero, zero
inner:  add    t3, a1, t1
        lw     t3, 0(t3)
        bne    t3, t4, skip
        addi   t5, t5, 1
skip:   addi   t1, t1, 4
        bne    t1, a3, inner
        addi   t0, t0, 4
        bne    t0, a2, outer
        mv     a0, t5
```

【分析解答】

```
        slli   a2, a2, 2          #a2 的内容左移 2 位，即乘 4
        slli   a3, a3, 2          #a3 的内容左移 2 位，即乘 4
        add    t5, zero, zero     #t5 初始化为 0
        add    t0, zero, zero     #t0 初始化为 0
outer:  add    t4, a0, t0         #计算数组 1 当前元素的地址
        lw     t4, 0(t4)          #数组 1 当前元素存放在 t4
        add    t1, zero, zero     #t1 初始化为 0
inner:  add    t3, a1, t1         #计算数组 2 当前元素的地址
        lw     t3, 0(t3)          #数组 2 当前元素存放在 t3
        bne    t3, t4, skip       #t3 和 t4 的内容不相等，则转 skip
        addi   t5, t5, 1          #t5 加 1
skip:   addi   t1, t1, 4          #t1 加 4
        bne    t1, a3, inner      #数组 2 未处理完，继续转 inner 执行
        addi   t0, t0, 4          #t0 加 4
```

```
        bne   t0, a2, outer              #数组 1 未处理完，继续转 outer 执行
        mv    a0, t5                     #t5 的内容作为返回结果，送 a0
```

该过程的功能是统计两个数组中相同元素的个数，多次相等则重复计数。

对应的 C 语言程序段如下：

```
int eq_cnt(int a[], int b[], int m, int n)
{    int i, j, count=0;
     for (i=0; i<m; i++; )
          for (j=0; j<n; j++)
              if (a[i]==b[j]) count=count+1;
     return count;
}
```

C 程序中的变量和寄存器之间的对应关系如下。

入口参数：a—a0、b—a1、m—a2、b—a3

返回值：a0

局部变量：i*4—t0、j*4—t1、count—t5

数组元素：a[i]—t4、b[j]—t3

该过程执行时最坏的情况是两个数组的所有元素都相等，这样，指令“addi t5, t5, 1”在每次循环中都被执行。因为 add、addi 和 slli 指令的 CPI 为 1，lw 和 bne 指令的 CPI 为 2，所以，当 m=n=2500 时，在最坏情况下所需的时钟周期总数为 {4+[4+(6+3)×2500+3]×2500}+1=56267507，时钟周期为 1/2GHz=0.5ns，因此，执行该过程的总执行时间最多为 56267507×0.5ns=28133753ns≈0.028s（28ms）。

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
```

16. 以下程序段是某个过程对应的 RV32I 指令序列，其功能为复制一个存储块数据到另一个存储块中，存储块中每个数据的类型为 int，sizeof(int)=4，源数据块和目的数据块的首地址分别存放在 a0 和 a1 中，复制的数据个数存放在 t0 中，最终作为返回参数通过寄存器 a0 返回给调用过程。假定在复制过程中遇到 0 就停止复制，最后一个 0 也需要复制，但不被计数。已知该程序段中有多个 bug，请找出它们并修改之。

```
        addi   t0, zero, 0
loop:   lw     t1, 0(a0)
        sw     t1, 0(a1)
        addi   a0, a0, 4
        addi   a1, a1, 4
        beq    t1, zero, loop
        mv     a0, t0              #伪指令，对应 addi a0, t0, 0
```

【分析解答】

修改后的代码如下：

```
        addi   t0, zero, 0
loop:   lw     t1, 0(a0)
        sw     t1, 0(a1)
        beq    t1, zero, exit
        addi   a0, a0, 4
        addi   a1, a1, 4
        addi   t0, t0, 1
        j      loop         #伪指令，对应 jal x0, loop
exit:   mv     a0, t0
```

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
         jal x0，there
skip:    ……
there:   addi   t1, a0, 4
```

19. 某 C 语言源程序中的一个 while 语句为“while (save[i] == k) i += 1;”，若对其编译时，编译器将 i 和 k 分别分配在寄存器 s3 和 s5 中，数组 save 的基址存放在 s6 中，则生成的 RV32I 汇编代码段如下。

```
loop:    slli    t1, s3, 2          # R[t1]←R[s3]<<2，即 R[t1]=i×4
         add     t1, t1, s6         # R[t1]←R[t1]+R[s6]，即 R[t1]=address of save[i]
         lw      t0, 0(t1)          # R[t0]←M[R[t1]+0]，即 R[t0]=save[i]
         bne     t0, s5, exit       # if R[t0]≠R[s5]=k then goto exit
         addi    s3, s3, 1          # R[s3]←R[s3]+1，即 i=i+1
         j       loop               # goto loop
exit:
```

假设从 loop 处开始的指令存放在内存 40000 处，上述循环对应的 RV32I 机器码如图 4.19 所示。

	7 位	5 位	5 位	3 位	5 位	7 位
40000	0	2	19	1	6	19
40004	0	22	6	0	6	51
40008	0		6	2	5	3
40012	0	21	5	1	12	99
40016	1		19	0	19	19
40020	1043967				0	111
40024	……					

图 4.19　题 19 图

根据上述叙述，回答下列问题，要求说明理由或给出计算过程。

（1）RISC-V 的编址单位是多少？数组 save 每个元素占几个字节？

（2）为什么指令“slli t1, s3, 2”能实现 4×i 的功能？

（3）该指令序列中，哪些指令是 R-型？哪些是 I-型？哪些是 B-型？哪些是 J-型？

（4）t0 和 s6 的编号各为多少?

（5）指令“j loop”是哪条指令的伪指令？其操作码的二进位表示是什么?

（6）标号 exit 的值是多少？如何根据 40012 处的指令计算得到？

（7）标号 loop 的值是多少？如何根据 40020 处的指令计算得到？

（提示 1 043 967=1024×1024-1-512-4096）

【分析解答】

（1）RV32I 的编址单位是字节。从图 7.18 中可看出，每条指令 32 位，占 4 个地址，所以一个地址中有 8 位，因此，RV32I 的编址单位是字节。从汇编代码段可以看出，每次循环取 save 数组元素时，其下标 i 都要乘以 4，所以数组的每个元素占 4 个字节。

（2）因为这是左移指令，左移 2 位，相当于乘以 2^2=4。

（3）从 RV32I 指令系统定义可知，第 2 条为 R-型，第 1、3、5 条为 I-型，第 4 条为 B-型，第 6 条为 J 型。

（4）从图中第 3、4 两条指令可看出，t0 的编号为 5，从第 2 条指令可看出 s6 的编号为 22。

（5）指令“j loop”是指令“jal r0, loop”的伪指令，其操作码的二进位表示为 111=127-16=110 1111B。

（6）标号 exit 的值是 40024，其含义是循环结束时跳出循环后执行的首条指令的地址。由图 7.18 可知，40012 处 bne 指令的机器码为：**0000000** 10101 00101 001 **01100** 1100011，根据以下 B 型指令格式可知，其立即数字段 imm[12:1]为 0 0 000000 0110B=6。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] = 40012+6*2=40012+12=40024。

31 25	24 20	19 15	14 12	11 7	6 0	
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu

（7）标号 loop 的值为 40000，是循环入口处首条指令的地址。由图 7.18 可知，40020 处的 jal 指令的机器码为：**1 1111110110 1 11111111** 00000 1101111。根据上述 jal 指令格式可知，其立即数字段 imm[20:1]为 **1 11111111 1 1111110110**。jal 指令的转移目标地址为 PC+SEXT[imm[20:1]<<1]=40020+(-10)*2=40000 。(1 043 967=1024×1024-1-512-4096 = 1111 1110 1101 1111 1111B)