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？

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、偏移（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？

转移（跳转）指令执行后，CPU将跳转到目标指令地址中执行，无需返回。

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，然后再跳转到目标指令（被调用过程第一条指令）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返回到调用指令处继续往后执行。

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）。

- 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？

- **因为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，因此取出两个字节的指令后，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y，则根据转移目标地址（220）=PC（260）+相对位移量（y），可知y=220-260=-40，用补码表示为1101 1000B=D8H。**
- **（注：如果PC增量在最后做，即取出一个字节的操作码之后， PC仍为258；但在取第二个字节的位移量前，PC变为259，且此时就开始进行目标地址计算，那么结果就为-39，也算正确。）**

220	OP
◦ ◦ ◦	
◦ ◦ ◦	
258	Jump（操作码）
259	y（算出来为D8）
260	OPnext
◦ ◦ ◦	

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

- 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？
- **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。**

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。

添3个0是为了补足位数，以使指令长度为16的倍数。

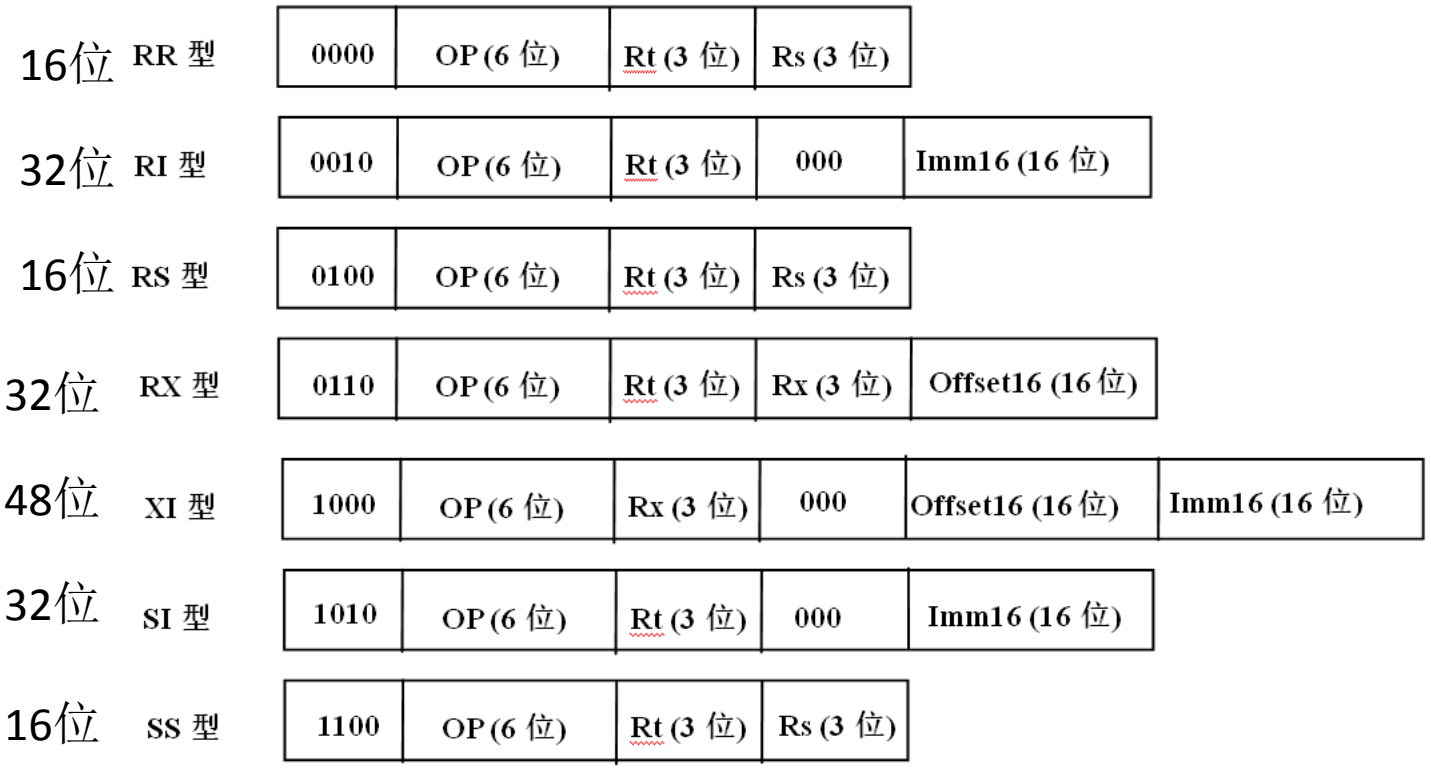


图 1　第一种指令格式示例

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。

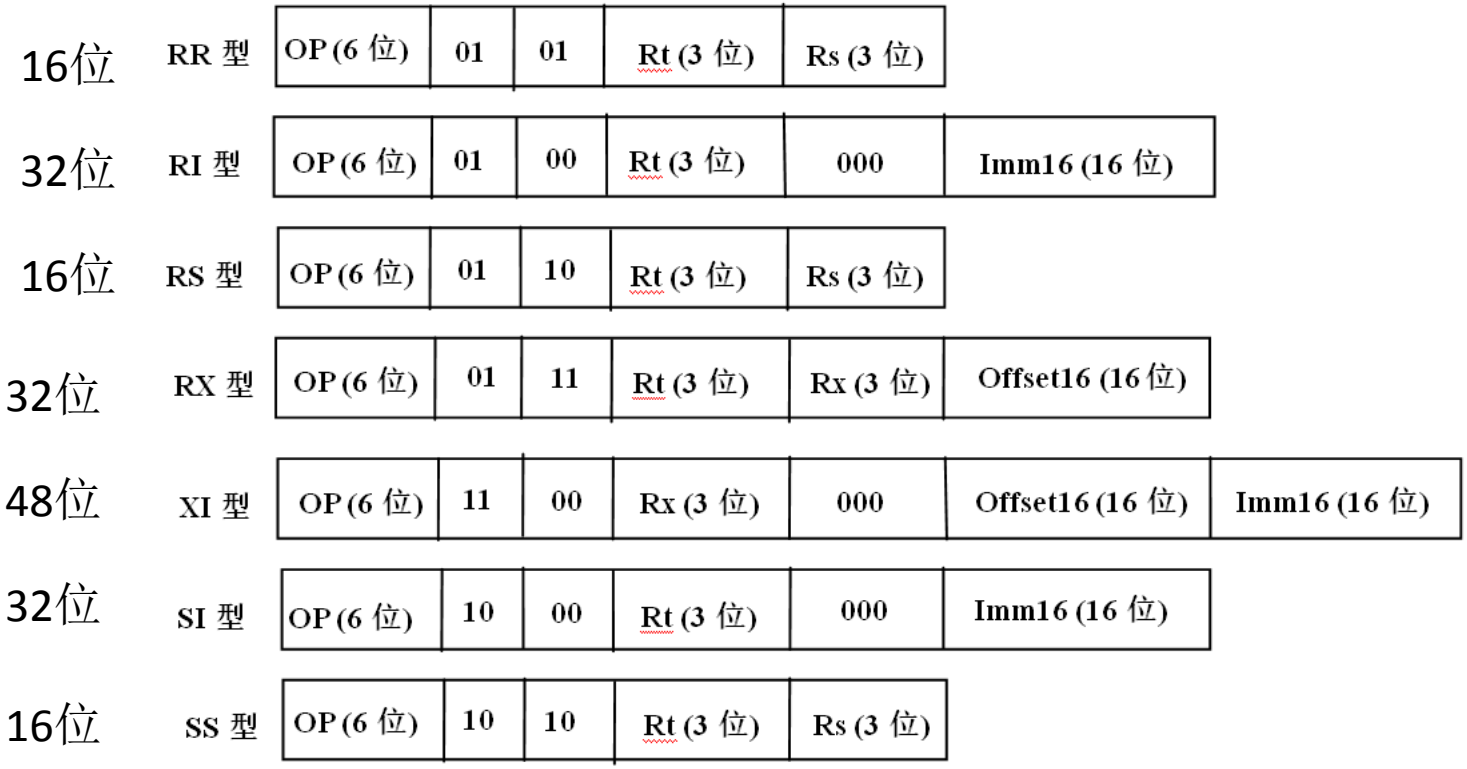
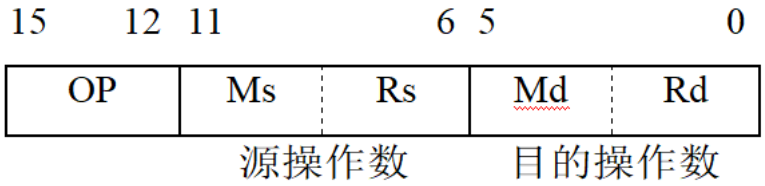


图 2　第二种指令格式示例

- 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。

- 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。

- 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。寻址方式定义如表所示。（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）

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]

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？

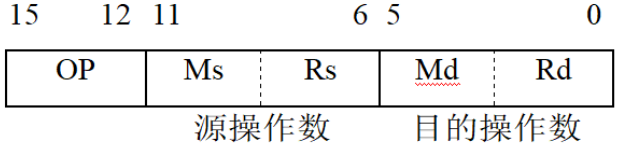
操作码字段占4位，所以最多有16条指令；

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（16位）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）转移指令的目标地址范围是多少？
- **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）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？
- **“add (R4),(R5)+” ——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；——对应的机器码为0010 001 100 010 101B，用十六进制表示为2315H。功能为：M[R5]←M[R[R4]] + M[R[R5]]，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**



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]？

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（全1加1之后变成全0，保持正确性）。

综上所述，PC所加的高20位应该为offset[31:12]+offset[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。

- **一个数x乘以7，相当于8x-x，**
- **8x可以通过将x左移3位来实现**
- **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。**
- **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**

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      #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
        #“jal x0, loop”的伪指令
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”。

- 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？

- **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00010 000H置于寄存器t1**
- **addi t1, t1, 4095 #将00010000H与FFFFFFFFH相加，送t1**
- **and t1, t0, t1 #将t0和t1的内容进行“与”运算，送t1**

```
  0000 0000 0000 0001 0000 0000 0000 0000
+ 1111 1111 1111 1111 1111 1111 1111 1111
  0000 0000 0000 0000 1111 1111 1111 1111
```

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
```

```
here:   bne  t0, t2, skip
        jal x0, there
skip:
……
there:  addi  t1, a0, 4
```

在RV32I指令系统中，分支指令beq是B-型指令，其偏移量offset为imm[12:1]乘2，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，当offset的范围为0000 0000 0000 0B ~0111 1111 1111 0B时，beq指令向后（正）跳，最多向后跳1023条指令；当offset的范围为1000 0000 0000 0B ~1111 1111 1111 0B时，beq指令向前（负）跳，最多向前跳1024条指令。

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。

here和skip之间仅相差两条指令，因而bne指令中的立即数不会超过可表示范围；而无条件跳转指令jal x0, there中可以表示20位的立即数，跳转范围为当前指令的前262 144到后262 143条指令），只要there处离jal指令在262 143条指令范围内，就可以直接跳转到there。