**191220154 张涵之 第4章作业**

1. 简单回答下列问题。
   1. 寄存器寻址下的操作数在寄存器中，直接寻址、间接寻址、寄存器间接寻址、偏移寻址（包括相对寻址、基址寻址、变址寻址）下的操作数在存储器中。
2. 转移指令（无论是无条件还是有条件转移）用于改变程序执行的顺序（如分支条件选择），不保存返回地址，转移后不再返回执行；调用指令用于调用子程序，需要保存返回地址，待子程序执行结束后跳转到调用的下一条指令继续执行。返回指令不必须要有地址码字段，如果指令系统约定调用子程序时返回地址按一定规律保存在栈中，或者存入特定的寄存器，则不需要地址码，去栈或寄存器中取即可。
3. 执行到该转移指令时PC的内容为258，且该转移指令占两个字节，则CPU取操作码和相对位移量之后，PC变为260。由于计算机采用相对寻址方式，且要求执行该指令后转移到220开始的一段程序执行，则PC + Offset = 260 + Offset = 220，Offset = -40，将相对位移量换算成补码，转移指令第二个字节的内容应该是11011000B。
4. 已知二地址指令有k2条，无地址指令有k0条，设单地址指令有k1条，则：

二地址指令有16 – 2 × 6 = 4位可以用来区分指令，用去k2，余下24 – k2种组合

一地址指令有16 – 6 = 10位可区分指令，用去k1，余(24 – k2) × 210-4 – k1种组合

无地址指令全部16位可区分指令，用去k0，余(210 – 26 × k2 – k1) × 216-10 – k0种

则有216 – 212 × k2 – 26 × k1 – k0 ≥ 0，解得k1 ≤ 210 – 26 × k2 – k0 / 26

综上所述，单地址指令最多有210 – 26 · k2 – 2-6 · k0条。

1. 设计指令系统的7种指令格式。由于计算机字长16位，存储器访问宽度和通用寄存器均为16位，要求指令长度为16的倍数。8个通用寄存器用3位表示，至多支持64种不同操作既操作码OP为6位，对于用到两个寄存器的指令，剩余16 – 6 – 3 – 3 = 4位用来表示指令寻址格式，立即寻址（I）为00，寄存器直接寻址（R）为01，寄存器间接寻址（S）为10，变址寻址（X）为11。对其他指令，可将某个寄存器的3位留空，在后面加上立即数或者偏移量即可。给出每种的指令长度、各字段所占位数和含义如图：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 4位 | 6位 | 3位 | 3位 | 16位 | 16位 |
| RR型 | 01 01 | OP | Rd | Rs |  |  |
| RI型 | 01 00 | OP | Rd | 000 | Imm16 |  |
| RS型 | 01 10 | OP | Rd | Rs |  |  |
| RX型 | 01 11 | OP | Rd | Rx | Offset |  |
| XI型 | 11 00 | OP | Rx | 000 | Offset | Imm16 |
| SI型 | 10 00 | OP | Rd | 000 | Imm16 |  |
| SS型 | 10 10 | OP | Rd | Rs |  |  |

其中RR型仅取指令访存1次；RI型取指令访存2次；RS型取指令访存1次，寄存器间接寻址1次，共访存2次；RX型取指令访存2次，变址寻址1次，共访存3次；XI型取指令访存3次，取源操作数即变址寻址1次，写结果1次，共访存5次；SI型取指令访存2次，取源操作数寄存器间接寻址1次，写结果1次，共访存4次；SS型取指令访存1次，取两个源操作数即寄存器间接寻址访存2次，写结果1次，共4次。

1. 回答下列问题：
   1. 操作码字段为4位，则该指令系统最多可有24 = 16条指令；Rs和Rd字段均为3位，则最多有23 = 8个通用寄存器；主存地址空间大小为128 KB，按字编址，则共是64 K个存储单元，存储器地址寄存器（MAR）至少需要16位；又因为计算机字长为16位，则存储器数据寄存器（MAR）至少也需要16位。
   2. 主存地址位数和计算机字长都是16位。转移指令采用相对寻址方式，其中PC、相对偏移量均为16位。则转移指令的目标地址范围是0000H ~ FFFFH。
   3. 各字段对应OP = 0010B，Ms = 001B，Rs = 100B，Md = 010B，Rd = 101B，则汇编语句对应机器码为0010 001 100 010 101B，换算成十六进制为2315H。

该指令的功能是R4间接寻址，R5间接寻址并自增，两数相加并存入R5寄存器间接寻址得到的地址。M[R[R5]] ← M[R[R4]] + M[R[R5]]，R[R5] ← R[R5] + 1，则代入R[R4] = 1234H，R[R5] = 5678H，M[1234H] = 5678H，M[5678H] = 1234H，得到M[5678H] ← 5678H + 1234H = 68ACH，R[R5] ← 5678H + 1 = 5679H，即执行该指令后，寄存器R5内容变为5679H，存储单元5678H内容变为68ACH。

1. 第一段：令adjuster = A\_lower12 >> 11（算数右移），A\_upper20与符号扩展后的adjuster异或得到A\_upper20\_adjusted。因为xori对A\_lower12进行符号扩展后和t0异或，如果A\_lower12最高位是0，则符号扩展后高20位全0，相当于A\_upper进行两次与全0的异或，结果不变；反之如果A\_lower12最高位是1，扩展后高20位全1，A\_upper进行两次与全1的异或，相当于取反两次，结果不变。从而对高20位进行校正。

第二段：令adjuster = A\_lower12 >> 11（逻辑右移），A\_upper20与符号扩展后的adjuster相加得到A\_upper20\_adjusted。因为lw对A\_lower12进行符号扩展后和t0相加，如果A\_lower12最高位是0，则符号扩展后高20位全0，又之前有A\_upper20\_adjusted等于 A\_upper20，不需要校正；反之如果A\_lower12最高位是1，扩展后高20位全1，与之前A\_upper20\_adjusted = A\_upper20 + 1中的1相加，进位溢出抵消，从而实现校正。

1. sll $t2, $t1, 9 // 先左移(31 – j) = 9位

srl $t2, $t2, 15 // 再逻辑右移(31 – j) + (i + 1) = 15位

1. slli a2, a2, 2 // $a2中内容左移两位（乘4）

slli a3, a3, 2 // $a3中内容左移两位（乘4）

add t5, zero, zero // $t5初始化为0（计数器）

add t0, zero, zero // $t0初始化为0（数组1“下标）

outer: add t4, a0, t0 // $t4 = $a0 + $t0（数组1当前处理到的地址）

lw t4, 0(t4) // $t4 = ($t4)（取出数组1当前处理到的元素）

add t1, zero, zero // $t1初始化为0（数组2“下标”）

inner: add t3, a1, t1 // $t3 = $a1 + $t1（数组2当前处理到的地址）

lw t3, 0(t3) // $t3 = ($t3)（取出数组2当前处理到的元素）

bne t3, t4, skip // if $t3 ≠ $t4 goto skip（取到的两个元素不等）

addi t5, t5, 1 // $t5 = $t5 + 1（计数器 +1）

skip addi t1, t1, 4 // $t1 = $t1 + 4（数组1“下标”移动一位）

bne t1, a3, inner // if $t1 ≠ $a3 goto inner（继续内层循环）

addi t0, t0, 4 // $t0 = $t0 + 4（数组2“下标”移动一位）

bne t0, a2, outer // if $t1 ≠ $a3 goto outer（继续外层循环）

mv a0, t5 // $a0 = $t5（将计数器的值保存到$a0中）

最坏情况下从不跳转到skip，内外层循环各2500次，（由于没有提供mv的CPI，此处最后一行不加入统计），共(9 × 2500 + 7) × 2500 + 4 = 56267504个时钟周期，每个时钟周期的长度为1/2GHz = 0.5ns，则运行该段指令所需时间是28133752ns ≈ 0.028s

该过程的功能是统计两个数组中相同元素的个数，对应C语言程序：

int a[2500], b[2500];

int count;

int tempCnt = 0;

for (int i = 0; i != 2500; i++) {

for (int j = 0; j != 2500; j++) {

if (a[i] == b[j])

tempCnt += 1;

}

}

$a0, $a1分别为数组a, b的首地址，$a0为最终计数的结果count，$t5为过程中临时计数器tempCnt，$t0, $t1相当于i, j，$a2, $a3相当于两个2500用于循环条件判断，在指令中使用乘4、加4是代码优化的方法。$t3, $t4分别为当前循环取到的a[i]和b[j]。

1. 31 = 1 1111B，可以直接用andi指令中的12位立即数表示。

b = 31&a: andi t1, t0, 31

65535 = 1111 1111 1111 1111B，无法用12位立即数表示，考虑分两次用lui和addi分别将最高4位1111B（7）和低12位1111 1111 1111B（4095）存入$t1，再与$t0做and。

b = 65535&a: lui t1, 7

addi t1, t1, 4095

and t1, t0, t1

另外还考虑，b = 65535&a其实就是保留a的低16位并将高16位清零，考虑将a先左移16位，再逻辑右移16位，比上面的作法可以再减少一条指令。

b = (a << 16) >> 16 : slli t1, t0, 16

srli t1, t1, 16

1. 存在的问题：循环条件beq与程序功能不符，遇到0应该停止复制，故改为beq exit并添加jar loop语句。此外，代码没有对t0计数，且最后一行mv是错误/不必要的。

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~~ exit

addi t0, t0, 1

jar loop

~~mv a0, t0~~

exit:

1. beq是一个分支指令，如伪代码beq t0, t2, there表示如果t0和t2相等就跳转到there出执行。然而，beq指令字段除了两个寄存器，还表示了一个12位立即数，且转移目标地址为PC + SEXT[imm[12:1]<<1]，可见跳转的地址范围有限，不能覆盖系统中能表示的全部地址。从图中不能看出从here到there的距离，超出上面范围就无法表示。

here: bne t0, t2, skip

jar there

skip:

…

there: addi t1, a0, 4

如上，如果t0和t2不相等就跳转到skip，这个距离一定在bne可跳转的范围内。如果相等则自动向下执行到jar there，该指令的跳转范围足够大，一定可以无条件跳转。

1. 回答下列问题。
   1. RISC-V的编址单位是字节，数组save的每个元素占4个字节。
   2. 二进制数左移两位相当于乘以4（不溢出的前提下）。
   3. add是R型指令，slli、addi是I型指令，bne是B型指令，j是J型指令。
   4. t0的编号为5，s6的编号为22。
   5. 指令“j loop”是jal的伪指令，其操作码的2进制表示是1101111B。
   6. 标号exit的值是40024，由40012加上相对位移量6 × 2 = 24，得到40024。

imm[12|10:5] = 0 = 0|000000，imm[4:1|11] = 12 = 0110|0，imm = 0110B = 6

* 1. 标号loop的值是40000，由40020加上相对位移量-10 × 2 = -20，得到40000。

imm[20|10:1|11|19:12] = 1043967 = 1|1111110110|1|11111111，

imm = 1111 1111 1111 1111 0110B = -10（-1010B的补码）