191220154 张涵之 第 4 章作业

2. 简单回答下列问题。
   4) 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。
   9) 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。

6. 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 $16 - 2 \times 6 = 4$ 位可以用来区分指令，用去 k2，余下 $2^4 - k2$ 种组合
   一地址指令有 $16 - 6 = 10$ 位可区分指令，用去 k1，余$(2^4 - k2) \times 2^{10-4} - k1$ 种组合
   无地址指令全部 16 位可区分指令，用去 k0，余$(2^{10} - 2^6 \times k2 - k1) \times 2^{16-10} - k0$ 种
   则有 $2^{16} - 2^{12} \times k2 - 2^6 \times k1 - k0 \geq 0$，解得 $k1 \leq 2^{10} - 2^6 \times k2 - k0 / 2^6$
   综上所述，单地址指令最多有 $2^{10} - 2^6 \cdot k2 - 2^{-6} \cdot k0$ 条。

7. 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。

8. 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 $2^4 = 16$ 条指令；Rs 和 Rd 字段均为 3 位，则最多有 $2^3 = 8$ 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。

9. 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。

12.
```
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
```

14.
```
        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$(9 \times 2500 + 7) \times 2500 + 4 = 56267504$ 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]。

15. 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
```

16. 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:
```

17. 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。

19. 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
    7) 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）