## 第四章课后习题

- **第八题解：**

  1. 指令格式中，OP 字段占 4 位（bit 15～12），所以最多有$2^4 = 16$条指令；由题中给出的指令格式可知，寄存器占用三个二进制位，所以最多有$2^3 = 8$个通用寄存器。

     主址内存空间大小为$128\text{KB}$，机器字长为16位，采用按字编址，1个字占用2个字节。所以共有$\frac{128\times1024}{2} = 2^{16}$个地址，范围为$0 \sim 2^{16}-1$，所以$MAR$和$MDR$都需要至少16位。

     综上，最多可有**16条指令**，最多有**8个通用寄存器**，存储器地址寄存器和存储器数据寄存器**均至少需要16位**。

  2. 由题知，$转移目标地址 = PC + R[Rn]$，PC 范围：0x0000 ~ 0xFFFF，寄存器中内容为16位二进制补码，对应范围是$-2^{16} \sim 2^{16}-1$。直接将对应范围相加可能出现**范围溢出**，应将得到的结果模$2^{16}$。

     所以，**目标地址范围为0x0000~0xFFFF**。

  3. 已知汇编语句为“add(R4), (R5)+”，根据定义的寻址方式及其含义，可以得到该汇编语句对应的机器码为：

     $$0010\ 001\ 100\ 010\ 101$$

     化为**十六进制表示是0x2315**，执行后**R5寄存器存放的地址变为0x5679**，**M[0x5678]=0x68AC**。

- **第九题解：**

  $$A\_upper20\_adjusted = A\_upper20 + (A\_lower12 \gg 11)$$

  即：**A_upper20_adjusted等于A_upper20加上A_lower12的最高位（bit 11）的值**。

- **第十六题解：**

  1. **错误1**：`addi t0, zero, 0` 覆盖了传入的参数，应用另一个寄存器计数。
  2. **错误2**：`beq t1, zero, loop` 条件跳转逻辑反了，应该是 `beq t1, zero, exit`。
  3. **错误3**：缺少对初始个数 `t0` 的检查，会复制超过 t0 个数据或直到 0。
  4. **错误4**：返回值 `mv a0, t0` 在循环内且 `t0` 值错误，应放在循环外，并返回实际非 0 数据个数。

  修改后代码如下：

```
addi  t2, zero, 0        ; 计数器，记录非0数据的个数
loop:
    lw    t1, 0(a0)          ; 取数据
    sw    t1, 0(a1)          ; 存储数据
    addi  a0, a0, 4
    addi  a1, a1, 4
    beq   t1, zero, exit     ; 遇到0，退出循环
    addi  t2, t2, 1          ; 非0，计数+1
    blt   t2, t0, loop       ; 如果计数 < 初始t0，继续循环
exit:
    mv    a0, t2             ; 返回非0数据的个数
```

- **第十九题解：**

  1. 编址单位 1 字节，数组元素 4 字节。
  2. 左移 2 位 = 乘以 4。
  3. R 型：add；I 型：slli, lw, addi；B 型：bne；J 型：j。
  4. t0=5, s6=22。
  5. jal x0, offset；操作码 1101111。
  6. exit=40040，B 型立即数 24 字节，40016+24=40040。
  7. loop=40000，J 型立即数 -12（单位 2 字节），40024 + (−12)×2 = 40000。

- **第二十题解：**

  汇编代码如下：

```
sum_array:
    addi sp, sp, -32
    sw ra, 28(sp)
    sw s0, 24(sp)
    sw s1, 20(sp)
    sw s2, 16(sp)
    sw s3, 12(sp)

    mv s0, a0
    mv s1, a1
    li s2, 0
    la t0, sum
    lw s3, 0(t0)

loop:
    bge s2, s1, end_loop
    mv a0, s1
    addi a1, s2, 1

    # 保存可能被破坏的临时寄存器
    sw t0, 8(sp)
    call compare
```

```
    lw t0, 8(sp)

    beq a0, zero, skip_add

    # 计算 array[i] 地址并加载值
    slli t1, s2, 2
    add t1, s0, t1
    lw t2, 0(t1)
    add s3, s3, t2

skip_add:
    addi s2, s2, 1
    j loop

end_loop:
    # 将更新后的 sum 存回全局变量和 t0
    la t0, sum
    sw s3, 0(t0)
    mv t0, s3
    mv a0, s3

    # 恢复寄存器
    lw s3, 12(sp)
    lw s2, 16(sp)
    lw s1, 20(sp)
    lw s0, 24(sp)
    lw ra, 28(sp)
    addi sp, sp, 32
    ret

compare:
    sgt a0, a0, a1
    ret
```

1. sum_array 调用前：假设初始 sp = 0x1000
2. sum_array 调用后（进入函数时）：sp = 0x1000 - 32 = 0xFE0
3. 调用 compare 时的栈状态：在compare 调用期间，sp 保持不变（0xFE0），因为 compare 是叶子函数，不需要额外的栈空间。
4. sum_array 返回前：恢复所有保存的寄存器后，sp = 0xFE0 + 32 = 0x1000，回到调用前的状态。