2025 HW1 Answer

Problem 1 （20分）

```
slli x5, x28, 2      # x5 = i*4 (A[i]地址偏移)
add x5, x10, x5      # x5 = &A[i]
lw x5, 0(x5)         # x5 = A[i]
-------------------------------------------------
slli x6, x29, 2      # x6 = j*4 (A[j]地址偏移)
add x6, x10, x6      # x6 = &A[j]
lw x6, 0(x6)         # x6 = A[j]
-------------------------------------------------
addi x6, x6, -5      # x6 = A[j] - 5
sub x5, x5, x6       # x5 = A[i] - (A[j]-5)
-------------------------------------------------
sw x5, 32(x11)       # B[8] = x5 (地址偏移32=8 * 4)
```

注意，其中**x5 x6**可以使用其他寄存器。

评分标准

- 用分割线分成了四部分，一部分**5分。**

Problem 2 （25分）

a) （10分）

0000 0000 0001 0000 1000 0000 1011 0011

0000000 00001 000001 000 00001 0110011

	funct7	rs2	rs1	funct3	rd	op
bit码	0000000	00001	00001	000	00001	0110011
含义	ADD	x1	x1	ADD	x1	R-format

0000000	rs2	rs1	000	rd	0110011	ADD

```
ADD x1,x1,x1
```

评分标准

- funct7/funct3/op 分析正确 **5分**
- rs2/rs1/rd 分析正确 **5分**

b) （15分）

opcode = 0x3 = 0000011 funct3 = 0x0 = 000 推出 LB 指令

imm[11:0]	rs1	000	rd	0000011	LB

rs1 = 27 = 11011 = x27

rd = 3 = 00011 = x3

imm = -5 = 1111 1111 1011

合并为

```
1111 1111 1011 1101 1000 0001 1000 0011
```

因此指令为

```
lb x3 -5(x27)
```

十六进制码为

```
0xFFBD8183
```

评分标准

- imm 转换正确 **4分**
- 指令正确 **8分**
- 十六进制码正确 **3分**

Problem 3 （25分）

a) （10分）

x0 是只读寄存器，永远是0

初始条件:

- x6 = 10（循环计数器），x5 = 0（累加器）。

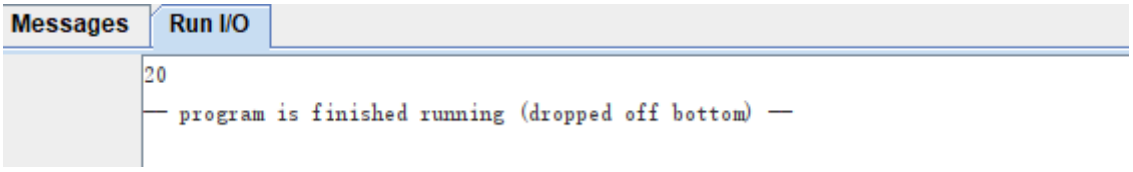
循环逻辑:

- `beq x6, x0, DONE`：若x6=0，跳出循环。
- 每轮循环执行：
  - `x6 = x6 - 1` → 共执行10次。
  - `x5 = x5 + 2` → 每次加2，总计增加 `10 × 2 = 20`。

结果：x5 = 20。

模拟器模拟结果

```
10    .text
11    addi x6, x0, 10
12    LOOP: beq x6, x0, DONE
13    addi x6, x6, -1
14    addi x5, x5, 2
15    jal x0, LOOP
16    DONE:
17    li a7 1
18    mv a0 x5
19    ecall
20
```



评分标准

- 结果正确**10分**，否则没有分数

b) （15分）

```
int i = 10, A = 0;
while (i != 0) {
    i--;
    A += 2;
}
```

评分标准

- 省略 int i = 10, A = 0 **扣5分**
- 其余代码部分 **10分**

Problem 4 （30分）

```
f:
    addi sp, sp, -16     # 分配栈空间
    sw ra, 12(sp)        # 保存返回地址
------------------------------------------------------------
    # 第一次调用g(a, b)
    jal ra, g            # g(a, b)，结果在a0
------------------------------------------------------------
    # 计算c + d
    add a1, a2, a3       # a1 = c + d
------------------------------------------------------------
```

```
    # 第二次调用g(g(a,b), c+d)
    jal ra, g           # 结果保存在a0, 直接返回即可
-----------------------------------------------------------
    # 恢复寄存器并返回
    lw ra, 12(sp)
    addi sp, sp, 16
    jr ra
```

默认参数`a,b,c,d`通过寄存器`a0,a1,a2,a3`传入`f`

调用`g(a, b)`的时候，也假设`a,b`通过`a0,a1`传入`g`

1. **第一次调用`g(a, b)`**：
   - 参数通过`a0`和`a1`传递（RISC-V调用约定）。
   - `call g`保存返回地址到`ra`，跳转到`g`。
   - 结果`g(a, b)`保存在`a0`，需用`s0`保存（防止被后续调用覆盖）。
2. **计算`c + d`**：
   - `add a1, a2, a3` → `a1 = c + d`。
3. **第二次调用`g(g(a,b), c+d)`**：
   - 参数`g(a,b)`（保存在`s0`）通过`a0`传递，`c+d`通过`a1`传递。
   - `call g`再次调用函数。
4. **栈操作**：
   - 保存`ra`和`s0`到栈中，确保嵌套调用后能正确返回。

评分标准

- ```
  addi sp, sp, -8    # 分配栈空间
  sw ra, 4(sp)       # 保存返回地址
  lw ra, 4(sp)
  addi sp, sp, 8
  ```

  这种也是可以的，不扣分
- 其他标注冗余操作的代码，多一处**扣2分**
-

  ```
  mv a0, a0           # a0 = a (冗余操作，实际可省略)
  mv a1, a1           # a1 = b (冗余操作，实际可省略)
  mv s0, a0           # 保存结果到s0 (冗余操作，实际可省略)
  mv a0, s0           # a0 = g(a, b) (冗余操作，实际可省略)
  ```

- 代码分成5部分，每部分正确**6分**
  ```