Hw7

T1

1.

```
; 寄存器分配情况
; R0 用以判断和输出
; R1 暂存 x
; R2 暂存 y
; R3 暂存 z
; R4 暂存 i
; R5 暂存 -z

            .ORIG x3000
MAIN        ; 主函数
INIT        INPUT[R1]       ; x 初始化（包括输入值部分）
            AND R2, R2, #0
            ADD R2, R2, #2  ; y 初始化
            ADD R3, R2, R2
            ADD R3, R3, R2
            ADD R3, R3, #1  ; z 初始化

            AND R4, R4, #0  ; i 初始化
            NOT R5, R3
            ADD R5, R5, #1  ; -z
LOOP        ADD R0, R4, R5
            BRzp DONE       ; 判断到 i >= z 时退出
            ADD R2, R4, #3  ; y = i + 3

            ADD R4, R4, #0
            BRn IF_TRUE

IF_ELSE     ADD R1, R1, #1
            BRnzp PRINT_x

IF_TRUE     ADD R1, R1, #-1
PRINT_x     OUTPUT[R1]
            LD R0, NEWLINE
            OUT

            ADD R4, R4, #1
            BRnzp LOOP

DONE        LD R0, SPACE    ; 暂认为 OUTPUT 有对 R0 进行保护
            OUTPUT[R1]
            OUT
            OUTPUT[R2]
            OUT
            OUTPUT[R3]
            OUT
            HALT

CONSTANT    ; 常量表
NEWLINE     .FILL   #10
```

```
SPACE       .FILL   #32
            .END
```

2. **Without changing its semantics（不改变语义）：**
   - **定义：对于任何给定的输入，优化后的程序产生的“可观测行为”（Observable Behavior）必须与原程序完全一致。**
   - **可观测行为**：
     - 所有输出内容和顺序必须保持一致；
     - 程序的最终状态（关键变量的最终值）对于外部观察者来说必须保持一致；
     - 不能引入新的错误。

> 暂时认为优化不是 -Og 的优化（不保证中间过程一致）。

```
; 优化代码
; 寄存器分配情况
; R0 用以判断和输出
; R1 暂存 x
; R2 暂存 y
; R3 暂存 z
; R4 暂存 i
; R5 暂存 -z

            .ORIG x3000
MAIN        ; 主函数
INIT        INPUT[R1]       ; x 初始化（包括输入值部分）
            AND R3, R3, #0
            ADD R3, R3, #7  ; z 初始化

            AND R4, R4, #0  ; i 初始化
            NOT R5, R3
            ADD R5, R5, #1  ; -z
LOOP        ADD R0, R4, R5
            BRzp DONE       ; 判断到 i >= z 时退出
            ADD R1, R1, #1  ; x++

            OUTPUT[R1]
            LD R0, NEWLINE
            OUT

            ADD R4, R4, #1
            BRnzp LOOP

DONE        ADD R2, R4, #2  ; y = i - 1 + 3（未进行常量传播和常量折叠的优化是因为初始化要一条指令的开销）
            LD R0, SPACE    ; 暂认为 OUTPUT 有对 R0 进行保护
            OUTPUT[R1]
            OUT
            OUTPUT[R2]
            OUT
            OUTPUT[R3]
            OUT
            HALT

CONSTANT    ; 常量表
NEWLINE     .FILL   #10
```

```
42  SPACE       .FILL   #32
43              .END
```

3.
   - **寄存器分配**：将变量直接放在寄存器里，而非反复读写内存；
   - **常量传播**：将已知变量视为常量；
   - **常量折叠**：在编译阶段算出常量值，不在运行阶段进行；
   - **死代码消除**：删除不会运行的代码；
   - **无效存储消除**：删除被重新赋值前未被读取的存储操作；
   - **循环展开**：消除跳转指令开销（在循环次数较小时）。

T2

1.
   - 1518；
   - 20；
   - 711。
2.
   - **A B + C D - ***
   - **A B && C D ! && ||**
   - **A B C D E ^ - * F / + G * H -**
3. $n + m - \sum_{i=1}^{m} opt_i = 1$

T3

1.
   - 要入栈的信息：
     - 函数参数；
     - 返回地址；
     - 帧指针；
     - 局部变量；
     - 受保护的寄存器。
   - 不需要，有些临时寄存器是 **caller-saved**，如果是非临时寄存器（**callee-saved**），只有在子程序被调用才需要保护；
2.
   - 因为每次递归都会在栈上创建新的栈帧。如果递归层级太深，会耗尽分配给栈的有限内存空间。
   - **解决原因**：
     - 迭代使用循环结构，复用同一块内存区域，而申请的变量内存是有限且较少的；
     - 手动栈在堆上申请空间，堆的内存显著大于栈的内存。

T4

1.
   - **“42 42”**；
   - **“5 42”**。
2.
   - 前者是浅拷贝，后者是深拷贝。
   - **浅拷贝**：
     - 优点：
       1. 速度快；
       2. 省内存，不需要额外申请。
     - 缺点：
       1. 与原数据共用内存，一旦修改，也会在原数据内体现；
       2. 如果原数据被销毁了，会导致 **b** 成为悬空指针。
   - **深拷贝**：
     - 优点：

        1. 数据独立，修改新的数据不会对原始数据产生影响；
        2. 生命周期灵活，在 **free(b)** 之前，即使 **a** 所在函数结束了， **b** 依然存在在堆上。
    - 缺点：
        1. 拷贝花费的时间更长；
        2. 拷贝会额外占用内存；
        3. 管理更麻烦，需要管理员手动调用 **free(b)** 释放内存，以防内存泄露。
3. - **b** 可以指向其他数组；
   - **a** 不能被重新赋值，因为数组名是常量左值。

T5

1. $4 \times 4 \times 4 \times (20 + 20) = 2560$
2. $20 \times 16 + 4 \times 4 \times 4 \times 2 = 448$

T6

- 26213；
- - 内存对齐；
  - 小端序（低位字节存放在低地址）。
偏移量 (Offset)	原始成员	写入的字符	十六进制 (Hex)	说明
0	c[0]	**'a'**	0x61	正常写入
1	c[1]	**'b'**	0x62	正常写入
2	c[2]	**'c'**	0x63	正常写入
3	Padding	**'d'**	0x64	**覆盖了填充字节**
4	i (Byte 0)	**'e'**	0x65	**覆盖了 i 的最低字节**
5	i (Byte 1)	**'f'**	0x66	**覆盖了 i 的第2个字节**
6	i (Byte 2)	**'\0'**	0x00	**覆盖了 i 的第3个字节**
7	i (Byte 3)	(未写入)	0x00	保持原值 (假设初始化为0)
  - 于是 **i = 0x00006665 = 26213**