Выполнил студент: Ошаров Александр Андреевич (БПИ248)

```
# ================================================
# Секция данных — объявляем ГЛОБАЛЬНЫЕ переменные и строки
# ================================================
.data

# Глобальная переменная: входное значение N для вычисления Фибоначчи
fib_input:    .word 10          # 32-битное целое число, инициализировано значением
10 (потом я его изменяю через ввод пользователя)

# Глобальная переменная: сюда сохраним результат
fib_result:   .word 0           # изначально 0

# Строки для вывода
msg_input:    .asciz "Кукуха требует число: "     # строка, завершённая \0
msg_result:   .asciz "Ахукук считает следующий ответ верным: "

# ================================================
# Секция кода — точка входа и функции
# ================================================
.text
.globl main                     # делаем main видимой для линковщика — точка входа
программы

# ------------------------------------------------
# main — главная функция программы
# ------------------------------------------------
main:
    # --- Шаг 1: Вывести строку "Input N: "
    li a7, 4                    # системный вызов №4 — печать строки (print_string) |
(псевдокоманда)
    la a0, msg_input            # загружаем АДРЕС строки msg_input в a0 |
(псевдокоманда)
    ecall                       # выполняем системный вызов

    # --- Шаг 2: Загрузить значение N из глобальной переменной
    #lw a0, fib_input            # I-формат: lw rd, offset(rs) — загружаем слово из
памяти [fib_input] → a0

    # --- Шаг 2: Ввести значение N с клавиатуры
    li a7, 5                    # системный вызов №5 — read_int (чтение целого числа)
| (псевдокоманда)
    ecall                       # после выполнения, введённое число будет в a0

    # --- Шаг 3: Вызвать функцию вычисления Фибоначчи
    jal ra, compute_fibonacci   # J-формат: jal rd, label — переход к метке, адрес
возврата → ra

    # --- Шаг 4: Сохранить результат в глобальную переменную
    la t1, fib_result           # Загружаем адрес переменной fib_result в регистр t1
| (псевдокоманда)
    sw a0, 0(t1)                # Сохраняем значение из a0 по адресу [t1 + 0]

    # --- Шаг 5: Вывести строку "Fibonacci result: "
    li a7, 4                    # системный вызов print_string | (псевдокоманда)
    la a0, msg_result           # загружаем адрес строки | (псевдокоманда)
    ecall

    # --- Шаг 6: Вывести само число (результат)
    li a7, 1                    # системный вызов №1 — печать целого числа
(print_int) | (псевдокоманда)
    lw a0, fib_result           # загружаем результат из памяти в a0 | I-формат (с
расширением RARS)
```

```
    ecall

    # --- Шаг 7: Завершить программу
    li a7, 10                   # системный вызов №10 — exit | (псевдокоманда)
    ecall                       # завершаем выполнение

# ================================================
# Функция: compute_fibonacci
# Вход: a0 = N (номер числа Фибоначчи)
# Выход: a0 = Fib(N)
# Использует стек для локальных переменных и сохранения регистров
# ================================================
compute_fibonacci:
    # --- Сохраняем регистры, которые будем использовать и которые обязаны сохранить
(callee-saved)
    addi sp, sp, -16            # I-формат: выделяем 16 байт в стеке (по 4 байта на
каждую ячейку)
                                # sp = sp - 16 — двигаем указатель стека вниз (стек
растёт вниз)

    sw s0, 0(sp)                # S-формат: сохраняем регистр s0 по адресу [sp + 0]
    sw s1, 4(sp)                # S-формат: сохраняем регистр s1 по адресу [sp + 4]
    sw ra, 8(sp)                # S-формат: сохраняем ra (адрес возврата) — на случай
рекурсии или вложенных вызовов
    sw s2, 12(sp)               # S-формат: дополнительно сохраняем s2 — будем
использовать как счётчик

    # --- Инициализируем локальные переменные в стеке
    # [sp + 0]  → fib_prev (предыдущее число Фибоначчи, F(n-2))
    # [sp + 4]  → fib_curr (текущее число Фибоначчи, F(n-1))
    # [sp + 8]  → (уже занято под ra)
    # [sp + 12] → s2 (счётчик, но можно и под данные)

    li t0, 0                    # псевдокоманда: загружаем 0 в t0 (F(0) = 0)
    sw t0, 0(sp)                # сохраняем F(0) как fib_prev

    li t0, 1                    # псевдокоманда: загружаем 1 в t0 (F(1) = 1)
    sw t0, 4(sp)                # сохраняем F(1) как fib_curr

    # --- Проверка граничных случаев: если N == 0 или N == 1
    beq a0, zero, return_zero   # B-формат: если a0 == 0 → переход на return_zero
    li t1, 1                    # псевдокоманда
    beq a0, t1, return_one      # B-формат: если a0 == 1 → переход на return_one

    # --- Основной цикл: вычисляем F(n) итеративно
    li s2, 2                    # инициализируем счётчик i = 2 (начинаем с F(2)) |
(псевдокоманда)
    loop_start:
        bgt s2, a0, loop_end    # B-формат: если s2 > a0 — выходим из цикла |
(псевдокоманда)

        # Загружаем fib_prev и fib_curr из стека
        lw t0, 0(sp)            # I-формат: fib_prev → t0
        lw t1, 4(sp)            # I-формат: fib_curr → t1

        # Вычисляем fib_next = fib_prev + fib_curr
        add t2, t0, t1          # R-формат: t2 = t0 + t1

        # Обновляем: fib_prev = fib_curr, fib_curr = fib_next
        sw t1, 0(sp)            # fib_prev ← fib_curr
        sw t2, 4(sp)            # fib_curr ← fib_next

        # Увеличиваем счётчик
        addi s2, s2, 1          # I-формат: s2 = s2 + 1
```

```
        j loop_start            # J-формат: безусловный переход на начало цикла |
(псевдокоманда)

    loop_end:
        lw a0, 4(sp)            # загружаем результат (fib_curr) в a0 — это
возвращаемое значение
        j func_exit             # переходим к выходу из функции | (псевдокоманда)

    return_zero:
        li a0, 0                # возвращаем 0 | (псевдокоманда)
        j func_exit             # псевдокоманда

    return_one:
        li a0, 1                # возвращаем 1 | (псевдокоманда)
        j func_exit             # псевдокоманда

    # --- Восстанавливаем сохранённые регистры и освобождаем стек
    func_exit:
        lw s0, 0(sp)            # восстанавливаем s0
        lw s1, 4(sp)            # восстанавливаем s1
        lw ra, 8(sp)            # восстанавливаем ra (адрес возврата)
        lw s2, 12(sp)           # восстанавливаем s2

        addi sp, sp, 16         # I-формат: sp = sp + 16 — освобождаем стек
(возвращаем как было)

        jr ra                   # I-формат: переход по адресу из ra — возврат в
вызывающую функцию
```