

Федеральное государственное бюджетное образовательное учреждение
высшего образования



**«Московский государственный технический университет
имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ
(ИУ6)

О Т Ч Е Т
по лабораторной работе №3

**Название домашнего задания: Программирование ветвлений и
циклов**

Дисциплина: Машинно-зависимые языки и основы компиляции

Студент гр. ИУ6-416

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

(И.О. Фамилия)

Москва, 2025

СОДЕРЖАНИЕ

Задание	3
Ход работы.....	4
Схема алгоритма	4
Реализация программы	5
Тестирование программы.....	13
Тест 1	13
Тест 2	13
Тест 3	13
Тест 4	14
Контрольные вопросы	15
Вопрос 1	15
Вопрос 2	15
Вопрос 3	16
Вопрос 4	16
Вывод.....	18
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	19

Задание

Вычислить целочисленное выражение (представлено на рисунке 1).

$$f = \begin{cases} \frac{a^3}{d} - x & \text{если } c > 10 \\ 3 & \text{иначе} \end{cases}$$

Рисунок 1 – Целочисленное выражение

Ход работы

Схема алгоритма

Построим схему алгоритма программы, решающей эту задачу. Схема алгоритма представлена на рисунке 2.

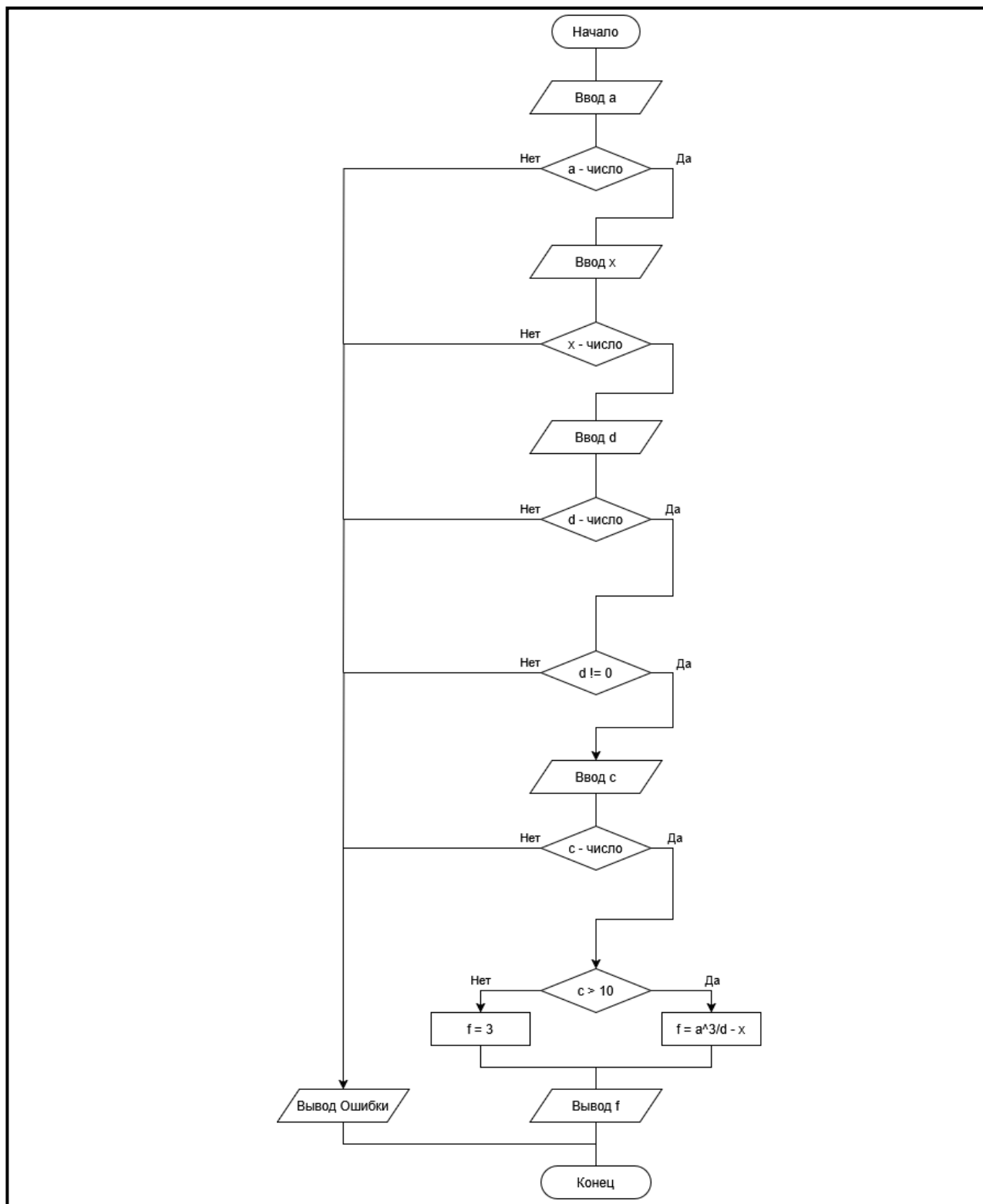


Рисунок 2 – Схема алгоритма

Реализация программы

На листинге 1 представлена программа, решающая задачу. Чтобы программа работала корректно, необходимо вводить числа без других символов, числа должны находиться в интервале от -30000 до 30000.

Листинг 1 – Программа, решающая задачу

```
section .data
    enter_msg db "Enter a: ", 0
    len_enter_msg equ $-enter_msg

    incor_number_msg db "Error: Incorrect number", 10
    len_incor_number_msg equ $-incor_number_msg

    div_zero_msg db "Error: divide by zero", 10
    len_div_zero_msg equ $-div_zero_msg

    res_msg db "Result: f = "
    len_res_msg equ $-res_msg

    a dd 0
    d dd 0
    x dd 0
    c dd 0
section .bss
    inbuf db 10
    outbuf db 10
section .text
    global _start
```

Продолжение Листинга 1

```
_start:
    ; Input a, a^3
    mov eax, 4
    mov ebx, 1
    mov ecx, enter_msg
    mov edx, len_enter_msg
    int 80h

    mov eax, 3
    mov ebx, 0
    mov ecx, inbuf
    mov edx, 10
    int 80h

    mov esi, inbuf
    call check_digits
    cmp ebx, 1
    je exit_program

    mov esi, inbuf
    call StrToInt
    mov ebx, eax
    mul ebx
    mul ebx
    mov [a], eax

    ; Input x
    mov byte[enter_msg + 6], 'x'
```

Продолжение Листинга 1

```
    mov eax, 4
    mov ebx, 1
    mov ecx, enter_msg
    mov edx, len_enter_msg
    int 80h

    mov eax, 3
    mov ebx, 0
    mov ecx, inbuf
    mov edx, 10
    int 80h

    mov esi, inbuf
    call check_digits
    cmp ebx, 1
    je exit_program

    mov esi, inbuf
    call StrToInt
    mov [x], eax

; Input d
    mov byte[enter_msg + 6], 'd'
    mov eax, 4
    mov ebx, 1
    mov ecx, enter_msg
    mov edx, len_enter_msg
    int 80h
```

Продолжение Листинга 1

```
    mov eax, 3
    mov ebx, 0
    mov ecx, inbuf
    mov edx, 10
    int 80h

    mov esi, inbuf
    call check_digits
    cmp ebx, 1
    je exit_program

    mov esi, inbuf
    call StrToInt
    cmp eax, 0
    je divide_by_zero
    mov [d], eax

; Input c
    mov byte[enter_msg + 6], 'c'
    mov eax, 4
    mov ebx, 1
    mov ecx, enter_msg
    mov edx, len_enter_msg
    int 80h

    mov eax, 3
    mov ebx, 0
    mov ecx, inbuf
```


Продолжение листинга 1

```
    mov edx, 10
    int 80h

    mov esi, inbuf
    call check_digits
    cmp ebx, 1
    je exit_program

    mov eax, 4
    mov ebx, 1
    mov ecx, res_msg
    mov edx, len_res_msg
    int 80h

    mov esi, inbuf
    call StrToInt
    cmp eax, 10
    jg count_h

    mov eax, 3
    mov esi, outbuf
    call IntToStr
    push eax
    mov eax, 4
    mov ebx, 1
    mov ecx, outbuf
    pop edx
    int 80h
```

Продолжение Листинга 1

```
    jmp exit_program
count_h:
    mov eax, [a]
    cdq
    mov ebx, [d]
    div ebx
    mov ebx, [x]
    sub eax, ebx
    mov esi, outbuf
    call IntToStr
    push eax
    mov eax, 4
    mov ebx, 1
    mov ecx, outbuf
    pop edx
    int 80h
exit_program:
    mov eax, 1
    xor ebx, ebx
    int 80h

check_digits:
    xor ebx, ebx
    xor ecx, ecx
    cmp byte[esi], '-'
    je .next_char
    jmp .check_loop
```

Продолжение Листинга 1

```
.check_loop:
    cmp byte[esi], 10
    je .end_check
    cmp byte[esi], '0'
    jl .fail
    cmp byte[esi], '9'
    jg .fail
    jmp .next_char

.next_char:
    inc esi
    jmp .check_loop

.fail:
    mov eax, 4
    mov ebx, 1
    mov ecx, incor_number_msg
    mov edx, len_incor_number_msg
    int 80h
    xor ebx, ebx
    mov ebx, 1
    ret

.end_check:
    ret

divide_by_zero:
    mov eax, 4
```

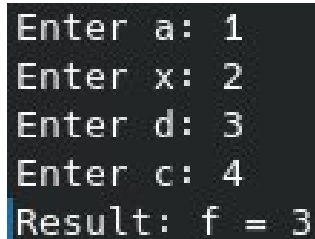
Продолжение Листинга 1

```
    mov ebx, 1
    mov ecx, div_zero_msg
    mov edx, len_div_zero_msg
    int 80h
    jmp exit_program
print_res:
    mov eax, 4
    mov ebx, 1
    mov ecx, res_msg
    mov edx, len_res_msg
    int 80h
    ret
#include "/home/user/MDLandCB/lib.asm"
```

Тестирование программы

Тест 1

Запустим программу, введем корректные данные, при которых программа выведет 3. Для этого введем несколько целых чисел, при чем c будет меньше 10. Ввод данных и результат работы программы представлены на рисунке 3.

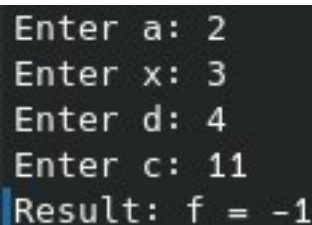
A screenshot of a terminal window with a dark background and light gray text. It shows the input of four variables: 'a' (1), 'x' (2), 'd' (3), and 'c' (4). The final output line is 'Result: f = 3'.

```
Enter a: 1
Enter x: 2
Enter d: 3
Enter c: 4
Result: f = 3
```

Рисунок 3 – Результат работы программы при $c < 10$

Тест 2

Запустим программу, введем $a = 2$, $x = 3$, $d = 4$, $c = 11$. Ожидается, что программа выведет -1. Ввод данных и результат работы программы представлены на рисунке 4.

A screenshot of a terminal window with a dark background and light gray text. It shows the input of four variables: 'a' (2), 'x' (3), 'd' (4), and 'c' (11). The final output line is 'Result: f = -1'.

```
Enter a: 2
Enter x: 3
Enter d: 4
Enter c: 11
Result: f = -1
```

Рисунок 4 – Результат работы программы при $c > 10$

Тест 3

Запустим программу, введем некорректные данные. Пусть $d = 0$. Ожидается, что при вводе $d = 0$, программа выведет ошибку и завершится. Ввод данных и результат работы программы представлены на рисунке 5.

```
Enter a: 1
Enter x: 2
Enter d: 0
Error: divide by zero
```

Рисунок 5 – Результат работы программы при $d = 0$

Тест 4

Запустим программу, введем некорректные данные. Пусть $a = g$ (не число). Ожидается, что при вводе $a = g$, программа выведет ошибку и завершится. Ввод данных и результат работы программы представлены на рисунке 6.

```
Enter a: g
Error: Incorrect number
```

Рисунок 6 – Результат работы программы при $a = g$

Из всех тестов следует, что программа работает корректно, при вводе некорректных данных выводится ошибка.

Контрольные вопросы

Вопрос 1

Какие машинные команды используют при программировании ветвлений и циклов?

В NASM для ветвлений и циклов используются условные и безусловные команды перехода:

Условные переходы:

- 1) `je / jz` — переход, если равно / если ноль ($ZF = 1$);
- 2) `jne / jnz` — переход, если не равно / если не ноль ($ZF = 0$);
- 3) `jg / jnle` — переход, если больше (для знаковых чисел);
- 4) `jl / jnge` — переход, если меньше (для знаковых чисел);
- 5) `ja / jnbe` — переход, если выше (для беззнаковых чисел);
- 6) `jb / jc / jnae` — переход, если ниже / если установлен CF (для беззнаковых чисел);
- 7) `jge, jle, jae, jbe` — аналогичные проверки с учётом равенства.

Безусловный переход (`jmp`) — передача управления без проверки условий:

- 1) `loop` — уменьшает `ecx/rcx` и переходит, если не ноль;
- 2) `jecxz / jrcxz` — переходит, если `ecx/rcx` равен нулю.

Вопрос 2

Выделите в своей программе фрагмент, реализующий ветвление. Каково назначение каждой машинной команды фрагмента?

Рассматриваемый участок программы представлен на листинге 2.

Листинг 2 – Рассматриваемый участок программы.

```
mov esi, inbuf
call check_digits
cmp ebx, 1
je exit_program
```

В этом куске кода реализуется следующий алгоритм ветвления.

В `esi` заносится адрес введенной строки. Если в строке, адрес которой записан в `esi`, найдутся символы, не являющиеся цифрами или знаком «-», то в консоль будет выведено сообщение об ошибке и в регистре `ebx` запишется 1. Чтобы завершить программу, мы сравниваем `ebx` с единицей. Если значения равны, то установится флаг $ZF = 1$ и по команде `je` мы перейдем в `exit_program`, после чего программа завершится. Если флаг не будет установлен, то программа будет выполняться далее.

Вопрос 3

Чем вызвана необходимость использования команд безусловной передачи управления?

Команда `jmp` (безусловный переход) нужна в следующих случаях:

- 1) выход из условных блоков — чтобы пропустить другие ветки кода (как в примере выше, после `mov ebx, 0`);
- 2) организация циклов — например, для возврата в начало цикла;
- 3) реализация конструкции `else` — если условие не выполнилось, нужно перейти к следующей части кода.

Вопрос 4

Поясните последовательность команд, выполняющих операции ввода-вывода в вашей программе. Чем вызвана сложность преобразований данных при выполнении операций ввода-вывода?

Рассмотрим участок кода, представленный на листинге 3.

Листинг 3 – Рассматриваемый участок кода

```
mov esi, outbuf
call IntToStr
push eax
mov eax, 4
```


Продолжение Листинга 3

```
mov ebx, 1  
mov ecx, outbuf  
pop edx
```

Математические операции производились над числами в десятичной системе счисления. Чтобы корректно вывести их в терминал, необходимо преобразовать каждое число в набор символов. Для этого запишем число в регистр `eax`, а адрес переменной, в которую запишется получившаяся строка, запишем в регистр `esi`. Вызовем функцию `IntToStr` из подключаемой библиотеки, которая преобразует число в строку по адресу из регистра `esi`. Длина получившейся строки будет записана в регистре `eax`. Вызываем системную команду для вывода строки. Сохраним значение регистра `eax` в стеке, запишем 4 в `eax`, 1 в `ebx`, адрес строки-числа в `ecx`, вернем из стека значение в регистр `edx`. Вызовем команду с помощью `int 80h` и число будет выведено в консоль.

Сложность преобразований данных при вводе-выводе:

- 1) разные форматы данных — например, число нужно преобразовать в строку для вывода (или наоборот при вводе);
- 2) буферизация — ввод/вывод может требовать ручного управления буфером;
- 3) кодировки — например, символы Unicode требуют дополнительной обработки.

Вывод

В ходе выполнения домашнего задания были изучены и применены навыки работы с символами средствами NASM. Написана 32-разрядная программа, подсчитывающая количество гласных букв в словах, разделенных пробелами. Программа корректно работает для латинского алфавита.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Иванова Г.С., Ничушкина Т.Н., лабораторный практикум по программированию на ассемблере в операционной системе LINUX. М.: МГТУ имени Н.Э. Баумана, 2022. 77 с.