РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: Архитектура компьютера

Студент: Терентьев Максим

Группа: НКАбд-05-25

МОСКВА

2025 г.

ЫI

Содержание

1. Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2. Порядок выполнения лабораторной работы

2.1. Реализация переходов в NASM

Создадим каталог для программ лабораторной работы № 7, перейдём в него и создадим файл lab7-1.asm:

```
msterentjev@rudn:~$ mkdir ~/work/arch-pc/lab07
msterentjev@rudn:~$ cd ~/work/arch-pc/lab07
msterentjev@rudn:~/work/arch-pc/lab07$ touch lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ls
lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 1. Создание каталога для lab7-1.asm

Введём в файл lab7-1.asm текст программы из листинга 7.1:

Открыть — *lab7-1.asm — ~/work/arch-pc/lab07

```
%include    'in_out.asm'    ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
    mov eax, msg1    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 1'

_label2:
    mov eax, msg2    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 2'

_label3:
    mov eax, msg3    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 3'

_end:
    call quit        ; вызов подпрограммы завершения
```

Рис. 2. Текст программы lab7-1.asm

Теперь сделаем исполняемый файл и запустим его:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 3. Работа программы lab7-1.asm

Таким образом, использование инструкции jmp _label2 меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки _label2, пропустив вывод первого сообщения. Инструкция jmp позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию jmp с меткой _label1 (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию jmp с меткой _end (т.е. переход к инструкции call quit). Изменим текст программы в соответствии с листингом 7.2:

Открыть *lab7-1.asm ~/work/arch-pc/lab07

```
%include    'in_out.asm'    ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
    mov eax, msg1    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 1'
    jmp _end

_label2:
    mov eax, msg2    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 2'
    jmp _label1

_label3:
    mov eax, msg3    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 3'

_end:
    call quit        ; вызов подпрограммы завершения
```

Рис. 4. Измененный текст программы lab7-1.asm в соответствии с листингом 7.2

Создадим исполняемый файл и проверим работу измененной программы:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 5. Работа измененной программы lab7-1.asm

Теперь изменим тест программы так, что бы выводилось сообщение 3, потом 2 потом 1 добавив или изменив инструкции jmp:

Открыть lab7-1.asm ~/work/arch-pc/lab07

```
%include    'in_out.asm'    ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:
    jmp _label3      ; Сначала прыгаем сразу к 3-му сообщению

_label1:
    mov eax, msg1    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 1'
    jmp _end         ; После 1-го сообщения – выход

_label2:
    mov eax, msg2    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 2'
    jmp _label1      ; После 2-го сообщения прыгаем к 1-му

_label3:
    mov eax, msg3    ; Вывод на экран строки
    call sprintLF    ; 'Сообщение № 3'
    jmp _label2      ; После 3-го сообщения прыгаем ко 2-му

_end:
    call quit        ; вызов подпрограммы завершения
```

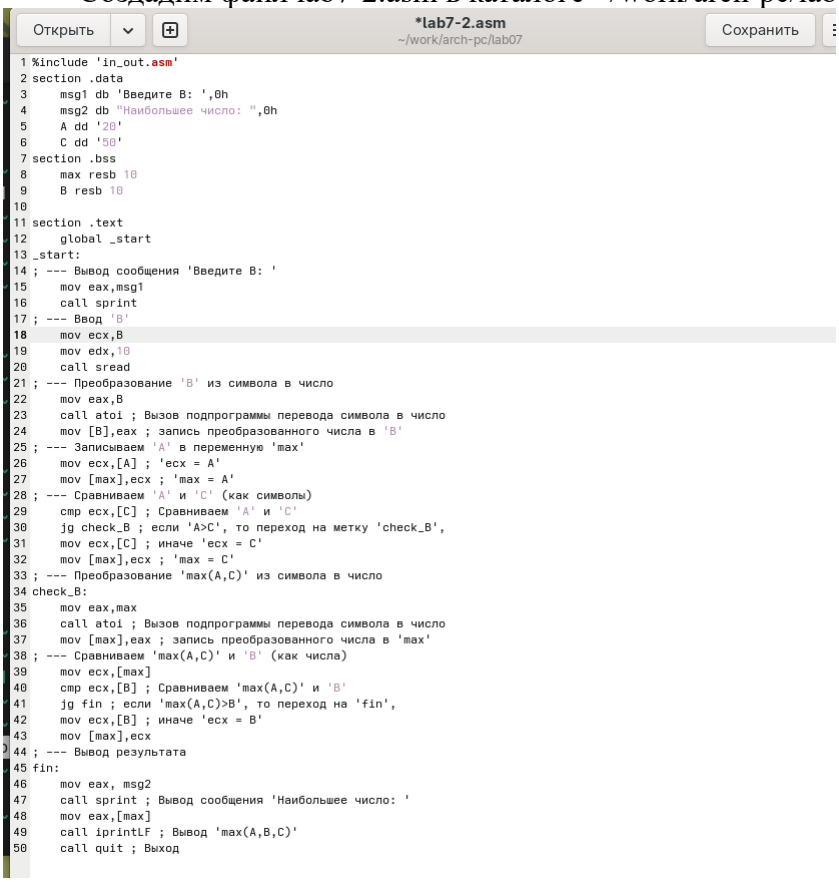
Рис. 6. Измененный код lab7-1.asm в соответствии с условием

Проверим вывод исп. файла сделанного из этого кода:

```
msterentjev@rudn:~/work/arch-pc/lab07$ gedit lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 7. Работа измененного в соотв. с условием файла lab7-1.asm

Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 и вставим в него код из листинга 7.3:




```
%include 'in_out.asm'
section .data
    msg1 db 'Введите B: ',0h
    msg2 db "Наибольшее число: ",0h
    A dd '20'
    C dd '50'
section .bss
    max resb 10
    B resb 10

section .text
    global _start
_start:
; --- Вывод сообщения 'Введите B: '
    mov eax,msg1
    call sprint
; --- Ввод 'B'
    mov ecx,B
    mov edx,10
    call sread
; --- Преобразование 'B' из символа в число
    mov eax,B
    call atoi ; Вызов подпрограммы перевода символа в число
    mov [B],eax ; запись преобразованного числа в 'B'
; --- Записываем 'A' в переменную 'max'
    mov ecx,[A] ; 'ecx = A'
    mov [max],ecx ; 'max = A'
; --- Сравниваем 'A' и 'C' (как символы)
    cmp ecx,[C] ; Сравниваем 'A' и 'C'
    jg check_B ; если 'A>C', то переход на метку 'check_B',
    mov ecx,[C] ; иначе 'ecx = C'
    mov [max],ecx ; 'max = C'
; --- Преобразование 'max(A,C)' из символа в число
check_B:
    mov eax,max
    call atoi ; Вызов подпрограммы перевода символа в число
    mov [max],eax ; запись преобразованного числа в 'max'
; --- Сравниваем 'max(A,C)' и 'B' (как числа)
    mov ecx,[max]
    cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
    jg fin ; если 'max(A,C)>B', то переход на 'fin',
    mov ecx,[B] ; иначе 'ecx = B'
    mov [max],ecx
; --- Вывод результата
fin:
    mov eax, msg2
    call sprint ; Вывод сообщения 'Наибольшее число: '
    mov eax,[max]
    call iprintLF ; Вывод 'max(A,B,C)'
    call quit ; Выход
```

Рис. 8. Текст программы lab7-2.asm

Сделаем исполняемый файл и проверим работу программы для разных значений B:

```
msterentjev@rudn:~/work/arch-pc/lab07$ gedit lab7-2.asm
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 192
Наибольшее число: 192
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 1032253550
Наибольшее число: 1032253550
msterentjev@rudn:~/work/arch-pc/lab07$
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 4
Наибольшее число: 50
msterentjev@rudn:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 45
Наибольшее число: 50
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 9. Работа lab7-2.asm для разных значений B

Обратим внимание, в данном примере переменные A и C сравниваются как символы, а переменная B и максимум из A и C как числа (для этого используется функция atoi преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные.

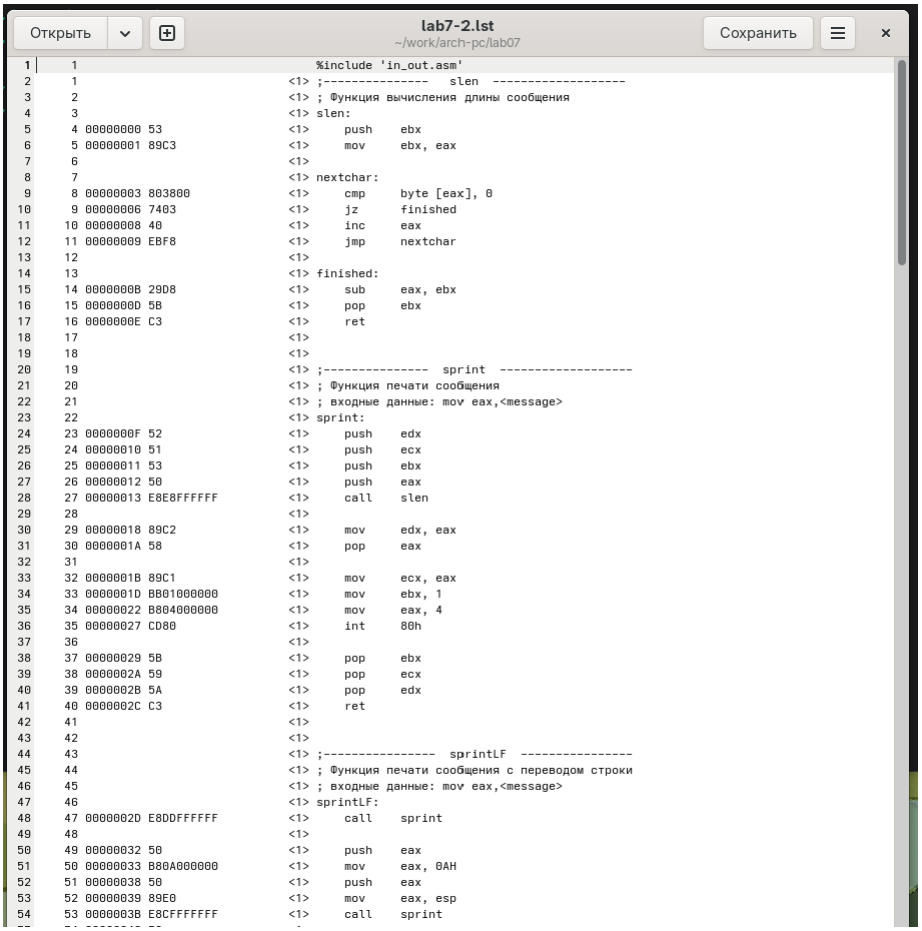
2.2. Изучение структуры файлы листинга

Создадим файл листинга для программы из файла lab7-2.asm:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst  lab7-2.o
```

Рис. 10. Создания файла листинга в формате .lst

Теперь откроем его в текстовом редакторе, я буду использовать gedit:




```
1     1                                %include 'in_out.asm'
2     1                            <1> ;---------------  slen  --------------------
3     2                            <1> ; Функция вычисления длины сообщения
4     3                            <1> slen:
5     4 00000000 53                <1>     push    ebx
6     5 00000001 89C3              <1>     mov     ebx, eax
7     6                            <1>
8     7                            <1> nextchar:
9     8 00000003 803800            <1>     cmp     byte [eax], 0
10    9 00000006 7403              <1>     jz      finished
11   10 00000008 40                <1>     inc     eax
12   11 00000009 EBF8              <1>     jmp     nextchar
13   12                            <1>
14   13                            <1> finished:
15   14 0000000B 29D8              <1>     sub     eax, ebx
16   15 0000000D 5B                <1>     pop     ebx
17   16 0000000E C3                <1>     ret
18   17                            <1>
19   18                            <1>
20   19                            <1> ;---------------  sprint  --------------------
21   20                            <1> ; Функция печати сообщения
22   21                            <1> ; входные данные: mov eax,<message>
23   22                            <1> sprint:
24   23 0000000F 52                <1>     push    edx
25   24 00000010 51                <1>     push    ecx
26   25 00000011 53                <1>     push    ebx
27   26 00000012 50                <1>     push    eax
28   27 00000013 E8E8FFFFFF        <1>     call    slen
29   28                            <1>
30   29 00000018 89C2              <1>     mov     edx, eax
31   30 0000001A 58                <1>     pop     eax
32   31                            <1>
33   32 0000001B 89C1              <1>     mov     ecx, eax
34   33 0000001D BB01000000        <1>     mov     ebx, 1
35   34 00000022 B804000000        <1>     mov     eax, 4
36   35 00000027 CD80              <1>     int     80h
37   36                            <1>
38   37 00000029 5B                <1>     pop     ebx
39   38 0000002A 59                <1>     pop     ecx
40   39 0000002B 5A                <1>     pop     edx
41   40 0000002C C3                <1>     ret
42   41                            <1>
43   42                            <1>
44   43                            <1> ;---------------  sprintLF  ----------------
45   44                            <1> ; Функция печати сообщения с переводом строки
46   45                            <1> ; входные данные: mov eax,<message>
47   46                            <1> sprintLF:
48   47 0000002D E8DDFFFFFF        <1>     call    sprint
49   48                            <1>
50   49 00000032 50                <1>     push    eax
51   50 00000033 B80A000000        <1>     mov     eax, 0AH
52   51 00000038 50                <1>     push    eax
53   52 00000039 89E0              <1>     mov     eax, esp
54   53 0000003B E8CFFFFFFF        <1>     call    sprint
```

Рис. 11. Часть текста в файле листинга lab7-2.lst

Опишу подробно данные 3 строки кода:

```
90     89 0000006D 52                <1>     push    edx
91     90 0000006E 83F800            <1>     cmp     eax, 0
92     91 00000071 75EA              <1>     jnz     divideLoop
```

Рис. 12. Строчки 90-92 из lab7-2.lst

Первые 2 столбца это номера строк. Третий столбец это адрес в оперативной памяти, куда будут загружены эти команды (в шестнадцатеричной системе). Четвертый столбец это машинный код команды в строке.

Строка 90(89)**:** push edx

- Что делает: Кладет значение из регистра edx в стек.
- Откуда там значение: Чуть выше (строка 87) было деление на 10. В edx попал остаток от деления (то есть последняя цифра числа). В строке 88 к нему прибавили 48 (код '0' в ASCII), превратив цифру в символ.
- Зачем в стек: Алгоритм получает цифры в обратном порядке (с конца: единицы, десятки, сотни). А выводить на экран нужно слева направо. Стек работает по принципу LIFO (последним зашел — первым вышел). Мы складываем цифры в стек, чтобы потом достать их в правильном порядке для печати.

Строка 91(90): cmp eax, 0

- Что делает: Сравнивает значение регистра eax с нулём.
- Контекст: В регистре eax хранится частное после деления на 10 (оставшаяся часть числа).
- Пример: Было 123, поделили на 10 -> остаток 3 (ушел в стек), в eax осталось 12.
- Смысл: Мы проверяем, «закончилось» ли число. Если в eax остался ноль, значит, мы обработали все цифры.

Строка 92(91): jnz divideLoop

- Что делает: (Jump if Not Zero) — Прыжок, если не ноль.
- Логика: Смотрит на результат сравнения из предыдущей строки (строка 90).
    - Если в eax НЕ ноль (число еще не закончилось), программа прыгает назад на метку divideLoop (строка 83), чтобы отделить следующую цифру.
    - Если в eax ноль (число закончилось), прыжок не выполняется, и программа идет дальше (к коду, который будет выводить цифры из стека на экран).

Откроем файл с программой lab7-2.asm и в любой инструкции с двумя операндами и удалим один операнд. Я удалю B из этой строчки(выделена серым):

```
14 ; --- Вывод сообщения 'Введите B: '
15     mov eax,msg1
16     call sprint
17 ; --- Ввод 'B'
18     mov ecx,B
19     mov edx,10
20     call sread
```

Рис. 13. Часть кода lab7-2.asm, откуда мы уберем второй операнд

Теперь выполним трансляцию с получением файла листинга:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:18: error: invalid combination of opcode and operands
```

Рис. 14. Ошибка при трансляции

Консоль выдала ошибку, но мы посмотрим изменился ли файл lab7-2.lst:

```
190     15 000000E8 B8[00000000]                mov eax,msg1
191     16 000000ED E81DFFFFFF                  call sprint
192     17                                  ; --- Ввод 'B'
193     18                                      mov ecx
194     18          ******************       error: invalid combination of opcode and operands
195     19 000000F2 BA0A000000                  mov edx,10
196     20 000000F7 E847FFFFFF                  call sread
197     21                                  ; --- Преобразование 'B' из символа в число
198     22 000000FC B8[0A000000]                mov eax,B
```

Рис. 15. Изменения в lab7-2.lst

Можем заметить, что у нас появилась эта ошибка в файле листинга и пометилась звездочками.

3. Задание для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы №6. Создайте исполняемый файл и проверьте его работу.

Мой вариант – 11, значит значения переменных a, b и c будут 21, 28 и 34 соответственно.
Создам файл labobus7-1.asm в который вставлю получившийся код:

Открыть *labobus7-1.asm ~/work/arch-pc/lab07 Сохранить

```
%include 'in_out.asm'
section .data
    msg2 db "Наименьшее число: ",0h
    A    db '21',0
    B    db '28',0
    C    db '34',0
section .bss
    min  resb 10
section .text
    global _start
_start:
    ; 1. Берем число 'A' и пока считаем его минимумом
    mov eax, A
    call atoi       ; Переводим строку '21' в число
    mov [min], eax  ; Записываем его в переменную min
    ; 2. Сравниваем 'B' с текущим минимумом
    mov eax, B
    call atoi       ; Переводим строку '28' в число

    mov ecx, [min]  ; В ecx лежит текущий минимум (21)
    cmp eax, ecx    ; Сравниваем B (eax) и min (ecx)

    jge check_C     ; Если B >= min, то ничего не меняем и прыгаем к проверке C
                    ; (jge = Jump if Greater or Equal)

    mov [min], eax  ; Иначе (если B < min), обновляем минимум

check_C:
    ; 3. Сравниваем 'C' с текущим минимумом
    mov eax, C
    call atoi       ; Переводим строку '34' в число

    mov ecx, [min]  ; В ecx лежит текущий минимум
    cmp eax, ecx    ; Сравниваем C и min

    jge fin         ; Если C >= min, прыгаем на выход

    mov [min], eax  ; Иначе обновляем минимум

fin:
    ; Вывод результата
    mov eax, msg2
    call sprint     ; Печатаем текст "Наименьшее число: "

    mov eax, [min]
    call iprintLF   ; Печатаем само число (должно быть 21)

    call quit       ; Выход
```

Рис. 16. Код для 1 задания, файл labobus7-1.asm

Теперь создадим исполняемый файл и проверим работу моей программы:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf labobus7-1.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o labobus7-1 labobus7-1.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./labobus7-1
Наименьшее число: 21
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 17. Работа labobus7-1.asm

Консоль вывела 21, и действительно из 21, 28 и 34 это число наименьшее, значит программа работает корректно.

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы №6.Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.

Так как мой вариант – 11, функция и корни даны мне такие:

11 $\begin{cases} 4a, & x = 0 \\ 4a + x, & x \neq 0 \end{cases}$ (0;3) (1;2)

Рис. 18. Функция и корни для 11 варианта

Создадим файл labobus7-2.asm и вставим туда получившийся код для программы:

Открыть *labobus7-2.asm ~/work/arch-pc/lab07

```
%include 'in_out.asm'
section .data
    msg1 db 'Введите x: ',0h
    msg2 db 'Введите a: ',0h
    msg3 db 'Результат: ',0h
section .bss
    x   resb 10
    a   resb 10
    res resb 10
section .text
    global _start
_start:
    ; Ввод переменной 'x'
    mov eax, msg1
    call sprint     ; Вывод "Введите x: "
    mov ecx, x
    mov edx, 10
    call sread      ; Считываем x как строку
    mov eax, x
    call atoi       ; Преобразуем строку в число
    mov [x], eax    ; Сохраняем число в память
    ; Ввод переменной 'a'
    mov eax, msg2
    call sprint     ; Вывод "Введите a: "
    mov ecx, a
    mov edx, 10
    call sread      ; Считываем a как строку
    mov eax, a
    call atoi       ; Преобразуем строку в число
    mov [a], eax    ; Сохраняем число в память
    ; Основная логика (Сравнение)
    mov ecx, [x]    ; Кладем x в регистр
    cmp ecx, 0      ; Сравниваем x с нулем
    je is_zero      ; Если x равно 0, прыгаем на метку is_zero
    ; Ветка: Если x != 0 (Формула: 4a + x)
    mov eax, [a]    ; Берем a
    mov ebx, 4
    mul ebx         ; Умножаем a на 4 (eax = 4*a)

    add eax, [x]    ; Прибавляем x (eax = 4a + x)
    mov [res], eax  ; Сохраняем результат
    jmp print_res   ; Прыгаем на вывод (чтобы пропустить ветку для нуля)
is_zero:
    ; Ветка: Если x = 0 (Формула: 4a)
    mov eax, [a]    ; Берем a
    mov ebx, 4
    mul ebx         ; Умножаем a на 4
    mov [res], eax  ; Сохраняем результат
print_res:
    ; Вывод результата
    mov eax, msg3
    call sprint     ; Вывод "Результат: "
    mov eax, [res]
    call iprintLF   ; Вывод самого числа
    call quit       ; Выход
```

Рис. 19. Текст программы labobus7-2.asm

Теперь создадим исполняемый файл и проверим работу программы для данных мне значений:

```
msterentjev@rudn:~/work/arch-pc/lab07$ nasm -f elf labobus7-2.asm
msterentjev@rudn:~/work/arch-pc/lab07$ ld -m elf_i386 -o labobus7-2 labobus7-2.o
msterentjev@rudn:~/work/arch-pc/lab07$ ./labobus7-2
Введите x: 0
Введите a: 3
Результат: 12
msterentjev@rudn:~/work/arch-pc/lab07$ ./labobus7-2
Введите x: 1
Введите a: 2
Результат: 9
msterentjev@rudn:~/work/arch-pc/lab07$
```

Рис. 20. Работа labobus7-2.asm

Выводы

В ходе выполнения лабораторной работы я изучил команды условного и безусловного переходов и приобрел навыки написания программ с разветвленной логикой. Также я ознакомился с назначением и структурой файла листинга, научившись сопоставлять машинный код с командами ассемблера.