

Отчет по лабораторной работе №6

Архитектура компьютера и операционные системы

Александр Дмитриевич Собко

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файлы листинга	13
5	Задание для самостоятельной работы	14
6	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Рисунок1	8
4.2	Рисунок2	9
4.3	Рисунок3	10
4.4	Рисунок4	10
4.5	Рисунок5	11
4.6	Рисунок6	12
4.7	Рисунок7	12
4.8	Рисунок8	13
4.9	Рисунок9	13
5.1	Рисунок10	14
5.2	Рисунок11	15
5.3	Рисунок12	15
5.4	Рисунок13	15
5.5	Рисунок14	16
5.6	Рисунок15	17
5.7	Рисунок16	17

Список таблиц

1 Цель работы

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

2 Задание

Изменить пару программ с условиями и написать 2 свои

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре

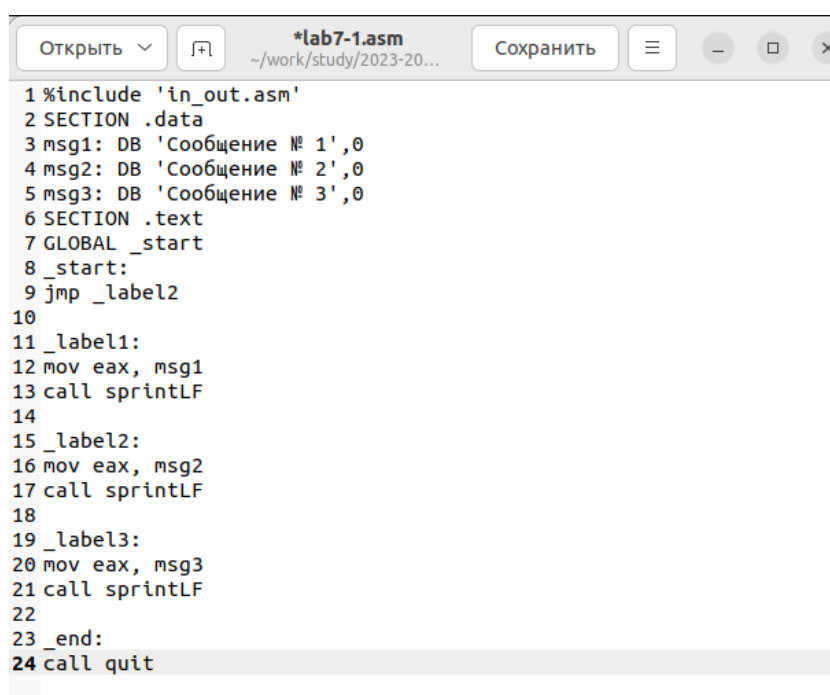
Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

1. Создайте каталог для программ лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm
2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл lab7-1.asm текст программы из листинга 7.1.



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label2
10
11 _label1:
12 mov eax, msg1
13 call sprintf
14
15 _label2:
16 mov eax, msg2
17 call sprintf
18
19 _label3:
20 mov eax, msg3
21 call sprintf
22
23 _end:
24 call quit
```

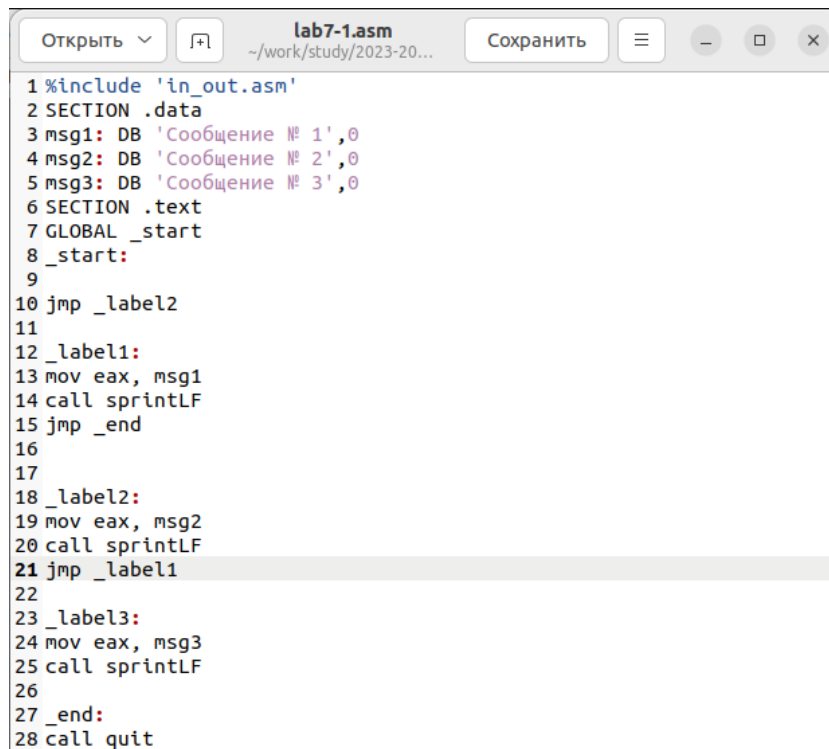
Рис. 4.1: Рисунок1

Создайте исполняемый файл и запустите его.

```
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ nasm -f elf lab7-1.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$
```

Рис. 4.2: Рисунок2

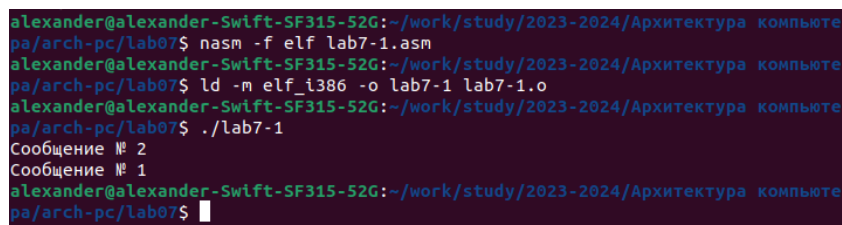
Таким образом, использование инструкции `jmp label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы 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`). Измените текст программы в соответствии с листингом 7.2



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 jmp _label2
11
12 _label1:
13 mov eax, msg1
14 call sprintLF
15 jmp _end
16
17
18 _label2:
19 mov eax, msg2
20 call sprintLF
21 jmp _label1
22
23 _label3:
24 mov eax, msg3
25 call sprintLF
26
27 _end:
28 call quit
```

Рис. 4.3: Рисунок3

Создайте исполняемый файл и проверьте его работу

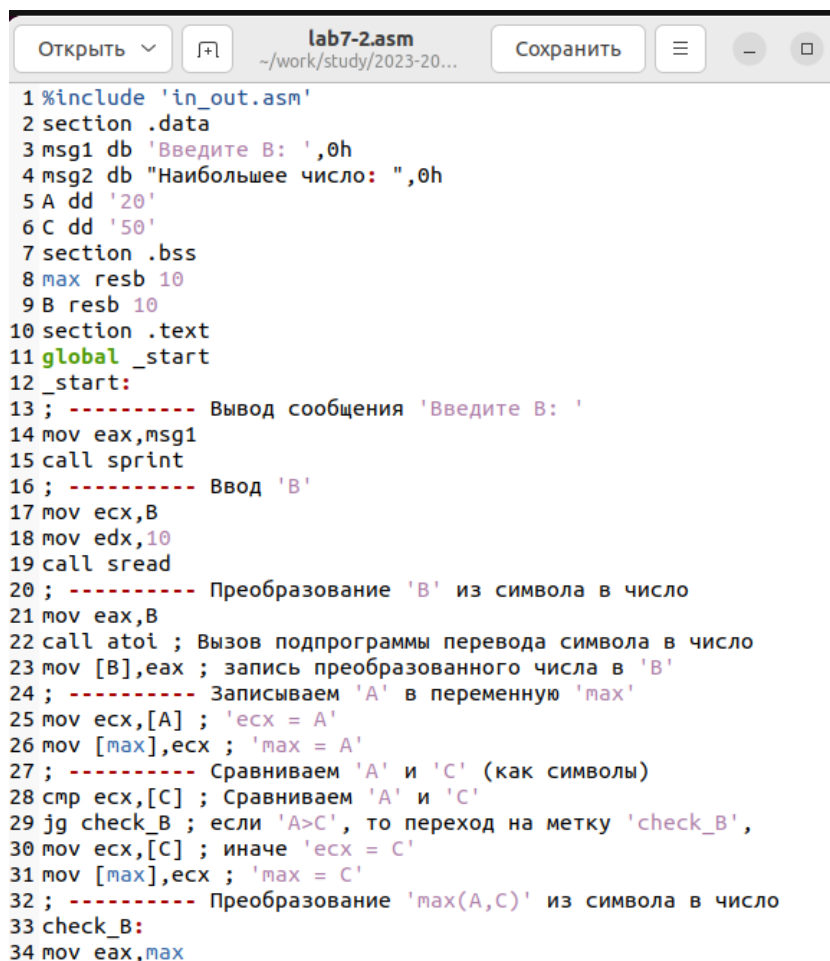


```
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab07$ nasm -f elf lab7-1.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab07$
```

Рис. 4.4: Рисунок4

- Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создайте файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Внимательно изучите текст программы из листинга 7.3 и введите в lab7-2.asm



```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '20'
6 C dd '50'
7 section .bss
8 max resb 10
9 B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
```

Рис. 4.5: Рисунок5

```

32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax, msg2
46 call sprint ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call iprintLF ; Вывод 'max(A,B,C)'
49 call quit ; Выход

```

Matlab ▾ Ширина табуляции: 8 ▾ Стр 26, Стлб 21 ▾

Рис. 4.6: Рисунок6

Создайте исполняемый файл и проверьте его работу для разных значений В. Обратите внимание, в данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

```

alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ touch lab7-2.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ open lab7-2.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ nasm -f elf lab7-1.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ nasm -f elf lab7-2.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ./lab7-2
Введите B: 7
Наибольшее число: 50
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ./lab7-2
Введите B: 52
Наибольшее число: 52
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$

```

Рис. 4.7: Рисунок7

4.2 Изучение структуры файлы листинга

4. Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создайте файл листинга для программы из файла `lab7-2.asm` `nasm -f elf -l lab7-2.lst lab7-2.asm` Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`: `mcedit lab7-2.lst` Внимательно ознакомиться с его форматом и содержимым. Подробно объяснить содержимое трёх строк файла листинга по выбору.

```
38 0000013F 8B0D[00000000]      mov ecx,[max]
39 00000145 3B0D[0A000000]      cmp ecx,[B] ; Сравниваем 'max(A,C)' и '
40 0000014B 7F0C                jg fin ; если 'max(A,C)>B', то переход
```

Рис. 4.8: Рисунок8

Строка в листинге состоит из номера строки в файле, смещении строки в программе (она больше номера строки потому что сначала в программу импортируется файл `in_out.asm`), затем следует оператор с операндами представленный в виде машинного кода, ну а затем то, как выглядит сама строчка в файле программы.

Откройте файл с программой `lab7-2.asm` и в любой инструкции с двумя операндами удалите один операнд. Выполните трансляцию с получением файла листинга: `nasm -f elf -l lab7-2.lst lab7-2.asm` Какие выходные файлы создаются в этом случае? Что добавляется в листинге?

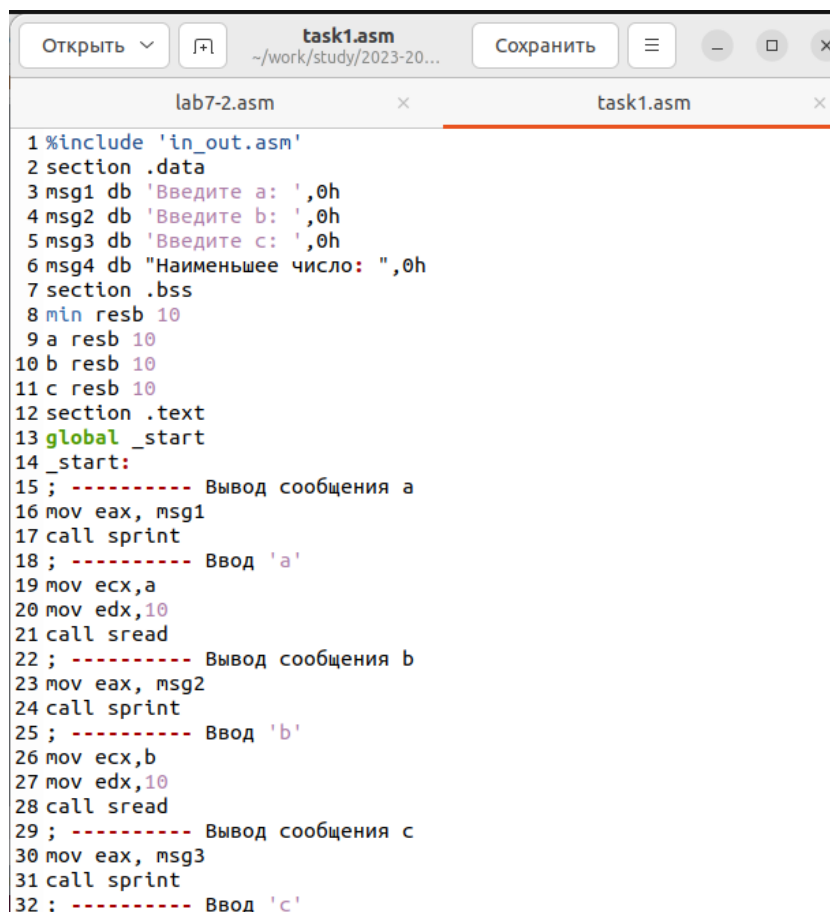
```
alexander@alexander-Swift-SF315-52G: ~/work/study/2023-2024/Архитектура_компьюте
pa/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:14: error: invalid combination of opcode and operands
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура_компьюте
```

Рис. 4.9: Рисунок9

Никакие. Выводится ошибка

5 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 № 7. Создайте исполняемый файл и проверьте его работу.



```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите a: ',0h
4 msg2 db 'Введите b: ',0h
5 msg3 db 'Введите c: ',0h
6 msg4 db "Наименьшее число: ",0h
7 section .bss
8 min resb 10
9 a resb 10
10 b resb 10
11 c resb 10
12 section .text
13 global _start
14 _start:
15 ; ----- Вывод сообщения a
16 mov eax, msg1
17 call sprint
18 ; ----- Ввод 'a'
19 mov ecx, a
20 mov edx, 10
21 call sread
22 ; ----- Вывод сообщения b
23 mov eax, msg2
24 call sprint
25 ; ----- Ввод 'b'
26 mov ecx, b
27 mov edx, 10
28 call sread
29 ; ----- Вывод сообщения c
30 mov eax, msg3
31 call sprint
32 ; ----- Ввод 'c'
```

Рис. 5.1: Рисунок10

```

31 call sprint
32 ; ----- Ввод 'с'
33 mov ecx,c
34 mov edx,10
35 call sread
36 ; ----- Преобразование 'a' из символа в число
37 mov eax,a
38 call atoi ; Вызов подпрограммы перевода символа в число
39 mov [a],eax ; запись преобразованного числа в 'a'
40 ; ----- Преобразование 'b' из символа в число
41 mov eax,b
42 call atoi ; Вызов подпрограммы перевода символа в число
43 mov [b],eax ; запись преобразованного числа в 'b'
44 ; ----- Преобразование 'с' из символа в число
45 mov eax,c
46 call atoi ; Вызов подпрограммы перевода символа в число
47 mov [c],eax ; запись преобразованного числа в 'с'
48
49
50 ; ----- Записываем 'a' в переменную 'min'
51 mov ecx,[a] ; 'ecx = a'
52 mov [min],ecx ; 'min = a'
53
54 mov ecx,[b]
55 cmp [min], ecx
56 jl cont1
57 mov [min], ecx
58 cont1:
59 mov ecx,[c]
60 cmp [min], ecx
61 jl fin
62 mov [min], ecx

```

Рис. 5.2: Рисунок11

```

61 jl fin
62 mov [min], ecx
63
64 ; ----- Вывод результата
65 fin:
66 mov eax, msg4
67 call sprint ; Вывод сообщения 'Наибольшее число: '
68 mov eax,[min]
69 call iprintLF ; Вывод 'min(A,B,C)'
70 call quit ; Выход

```

Рис. 5.3: Рисунок12

Я выбрал вариант 1

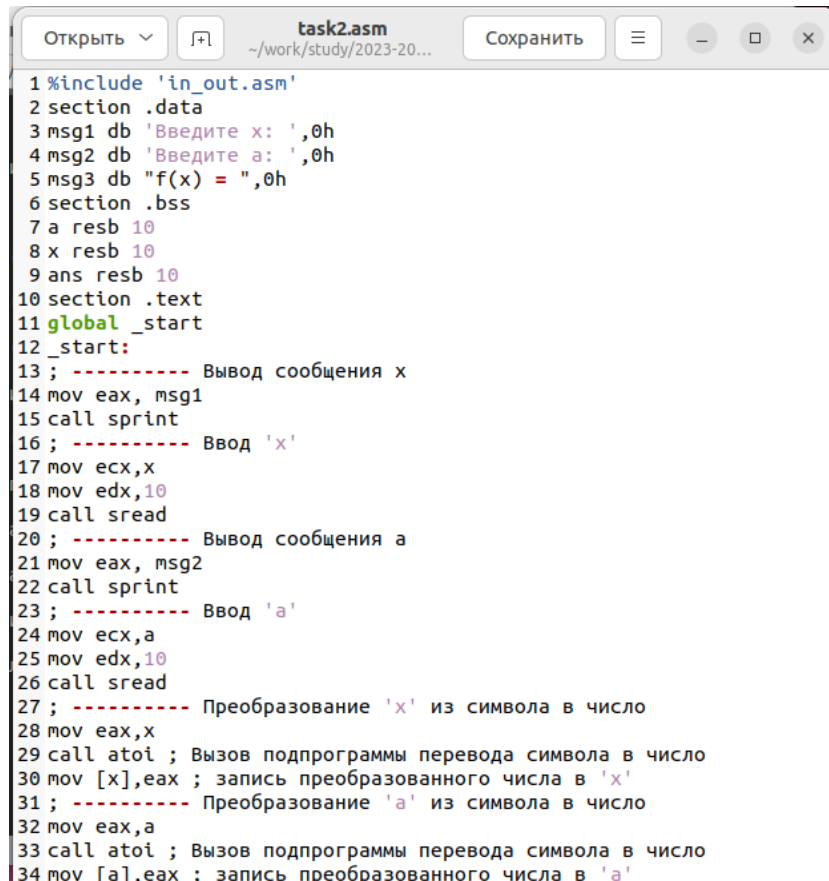
```

alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
pa/arch-pc/lab07$ ./task1
Введите a: 17
Введите b: 23
Введите c: 45
Наименьшее число: 17
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера

```

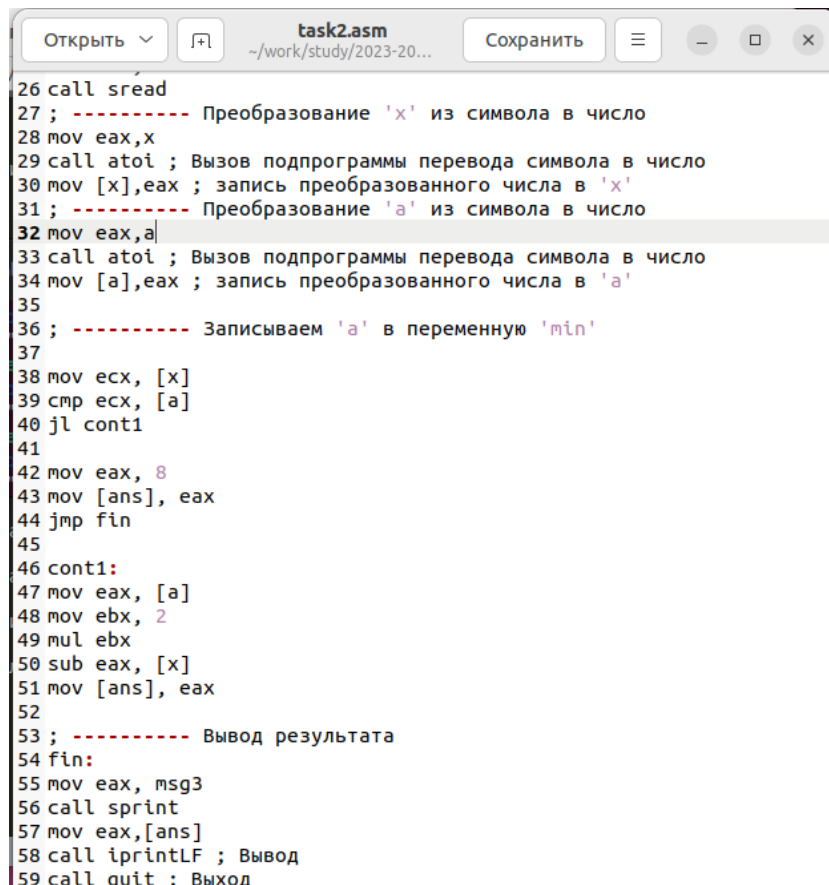
Рис. 5.4: Рисунок13

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 № 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6.



```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите x: ',0h
4 msg2 db 'Введите a: ',0h
5 msg3 db "f(x) = ",0h
6 section .bss
7 a resb 10
8 x resb 10
9 ans resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения x
14 mov eax, msg1
15 call sprint
16 ; ----- Ввод 'x'
17 mov ecx, x
18 mov edx, 10
19 call sread
20 ; ----- Вывод сообщения a
21 mov eax, msg2
22 call sprint
23 ; ----- Ввод 'a'
24 mov ecx, a
25 mov edx, 10
26 call sread
27 ; ----- Преобразование 'x' из символа в число
28 mov eax, x
29 call atoi ; Вызов подпрограммы перевода символа в число
30 mov [x], eax ; запись преобразованного числа в 'x'
31 ; ----- Преобразование 'a' из символа в число
32 mov eax, a
33 call atoi ; Вызов подпрограммы перевода символа в число
34 mov [a], eax ; запись преобразованного числа в 'a'
```

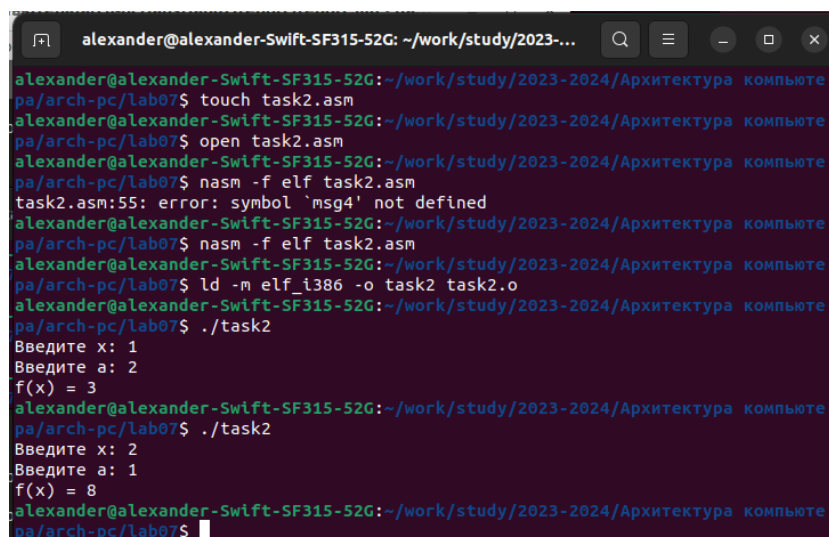
Рис. 5.5: Рисунок14



```
26 call sread
27 ; ----- Преобразование 'x' из символа в число
28 mov eax,x
29 call atoi ; Вызов подпрограммы перевода символа в число
30 mov [x],eax ; запись преобразованного числа в 'x'
31 ; ----- Преобразование 'a' из символа в число
32 mov eax,a
33 call atoi ; Вызов подпрограммы перевода символа в число
34 mov [a],eax ; запись преобразованного числа в 'a'
35
36 ; ----- Записываем 'a' в переменную 'min'
37
38 mov ecx, [x]
39 cmp ecx, [a]
40 jl cont1
41
42 mov eax, 8
43 mov [ans], eax
44 jmp fin
45
46 cont1:
47 mov eax, [a]
48 mov ebx, 2
49 mul ebx
50 sub eax, [x]
51 mov [ans], eax
52
53 ; ----- Вывод результата
54 fin:
55 mov eax, msg3
56 call sprint
57 mov eax,[ans]
58 call iprintLF ; Вывод
59 call quit ; Выход
```

Рис. 5.6: Рисунок15

Вариант 1



```
alexander@alexander-Swift-SF315-52G: ~/work/study/2023-...
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ touch task2.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ open task2.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ nasm -f elf task2.asm
task2.asm:55: error: symbol 'msg4' not defined
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ nasm -f elf task2.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ld -m elf_i386 -o task2 task2.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ./task2
Введите x: 1
Введите a: 2
f(x) = 3
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$ ./task2
Введите x: 2
Введите a: 1
f(x) = 8
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab07$
```

Рис. 5.7: Рисунок16

6 Выводы

Мы научились составлять программы с условиями

Список литературы