

Отчёт по лабораторной работе №7

Дисциплина: Архитектура Компьютера

Егор Витальевич Кузьмин

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Работа с директориями и создание файла	8
4.2	Редактирование файла	8
4.3	Подготовка и исполнение файла	9
4.4	Редактирование файла	9
4.5	Трансляция, компоновка, запуск исполняемого файла	9
4.6	Создание, редактирование файла	10
4.7	Компиляция, обработка и запуск исполняемого файла	10
4.8	Редактирование файла	11
4.9	Компиляция, обработка и запуск исполняемого	11
4.10	Получение файла	12
4.11	Открытие файла в mcedit	12
4.12	Редактирование файла, создание листинга	13
4.13	Открытие листинга	13
4.14	Создание и редактирование файла	14
4.15	Компиляция, обработка и запуск исполняемого файла	14
4.16	Создание и редактирование файла	15
4.17	Компиляция, обработка и запуск исполняемого файла	15

1 Цель работы

Целью данной работы является изучение команд условного и безусловного переходов, приобретение практического опыта в написании программ с использованием переходов, знакомство с назначением и структурой файла листинга

2 Задание

- 0. Общее ознакомление с командами условного и безусловного переходов.
- 1. Реализация переходов в NASM.
- 2. Изучение структуры файла листинга.
- 3. Выполнение заданий для самостоятельной работы

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. 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. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания. Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. Листинг (в

рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию. Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);

исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается)

4 Выполнение лабораторной работы

4.1) Символьные и численные данные в NASM.

С помощью утилиты `mkdir` создаю директорию `lab07` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab7-1.asm`. (рис. 4.1).

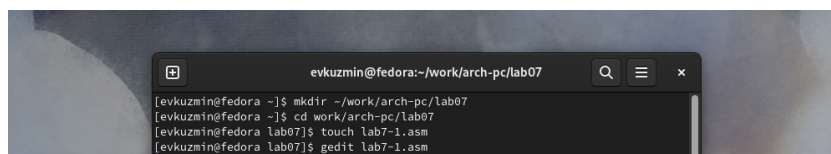


Рис. 4.1: Работа с директориями и создание файла

Открываю созданный файл `lab7-1.asm`, вставляю в него следующую программу: (рис. 4.2).

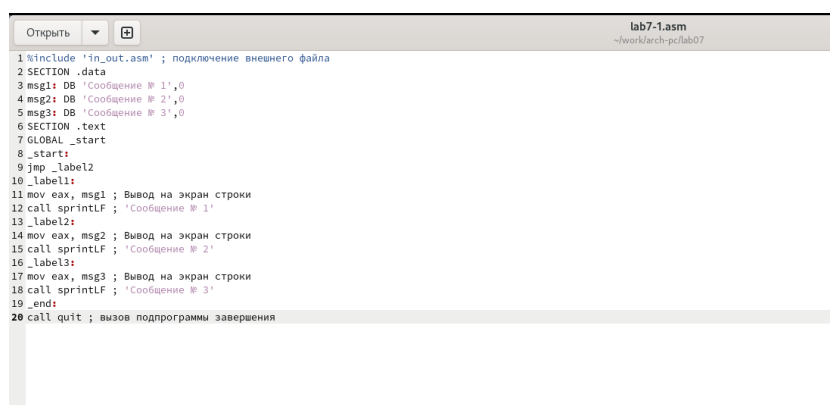


Рис. 4.2: Редактирование файла

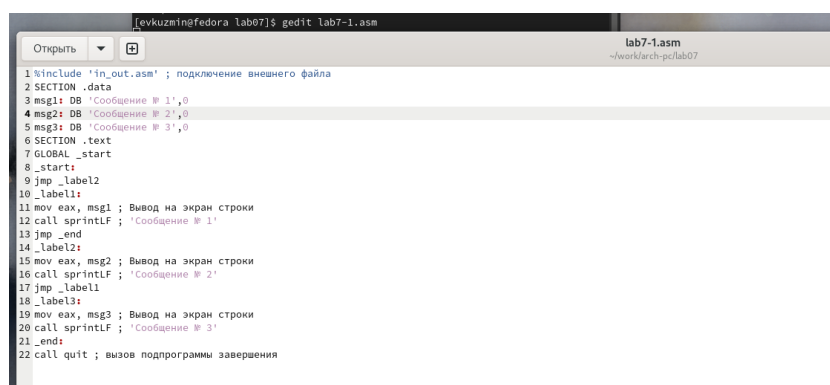
Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, ибо он

будет использоваться в дальнейшем. Выполняю компиляцию, компоновку файла и запускаю его. Мы видим, что использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения (рис. 4.3).

```
[evkuzmin@fedora lab07]$ cp ~/Загрузки/in_out.asm in_out.asm
[evkuzmin@fedora lab07]$ nasm -f elf lab7-1.asm
[evkuzmin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[evkuzmin@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4.3: Подготовка и исполнение файла

Добавляю в текст метки `jmp _label1 jmp_end` (рис. 4.4).



```
[evkuzmin@fedora lab07]$ gedit lab7-1.asm
lab7-1.asm
~/work/arch-pc/lab07

1 %include 'in_out.asm' ; подключение внешнего файла
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 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 _end:
22 call quit ; вызов подпрограммы завершения
```

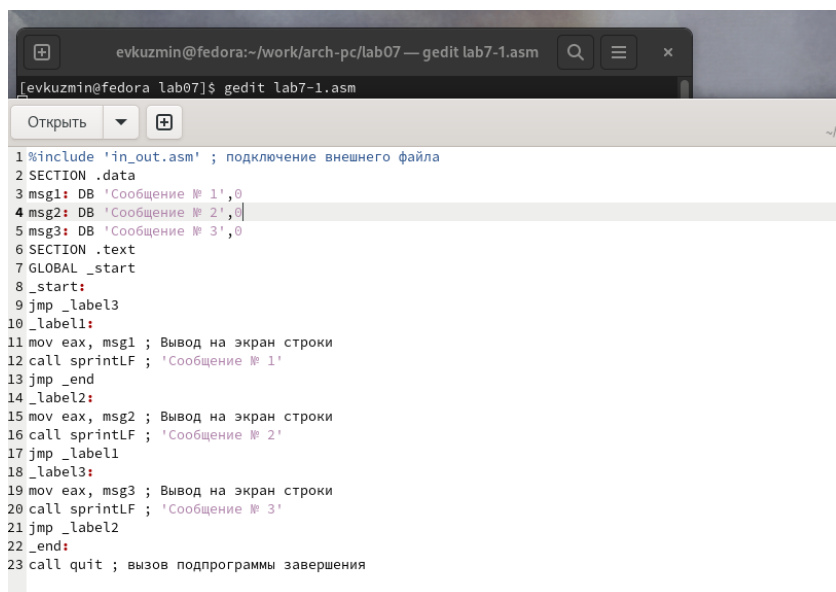
Рис. 4.4: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его. Соответственно, инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. (рис. 4.5).

```
[evkuzmin@fedora lab07]$ nasm -f elf lab7-1.asm
[evkuzmin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[evkuzmin@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 4.5: Трансляция, компоновка, запуск исполняемого файла

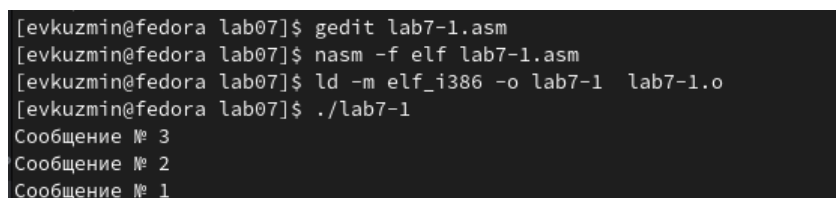
Изменяю метки `jmp` в программе, чтобы выводились сообщения в порядке 3,2,1 (рис. 4.6).



```
1 %include 'in_out.asm' ; подключение внешнего файла
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 _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
```

Рис. 4.6: Создание, редактирование файла

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. Видим, что все работает так, как нужно. (рис. 4.7).



```
[evkuzmin@fedora lab07]$ gedit lab7-1.asm
[evkuzmin@fedora lab07]$ nasm -f elf lab7-1.asm
[evkuzmin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[evkuzmin@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 4.7: Компиляция, обработка и запуск исполняемого файла

Создаю файл `lab7-2.asm`. Редактирую его, вводя предлагаемую программу. (рис. 4.8).

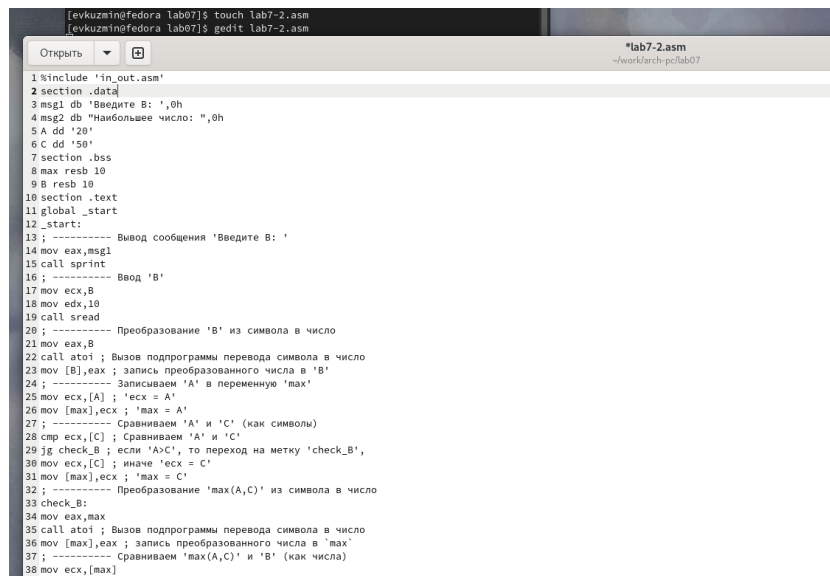


Рис. 4.8: Редактирование файла

Создаю исполняемый файл и проверяю его работу для разных значений B. (рис. 4.9).

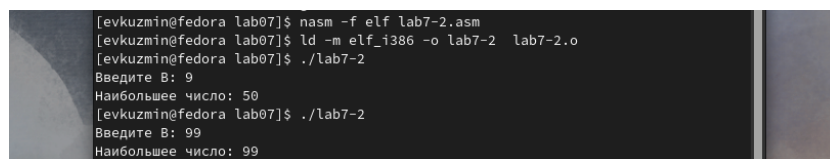


Рис. 4.9: Компиляция, обработка и запуск исполняемого

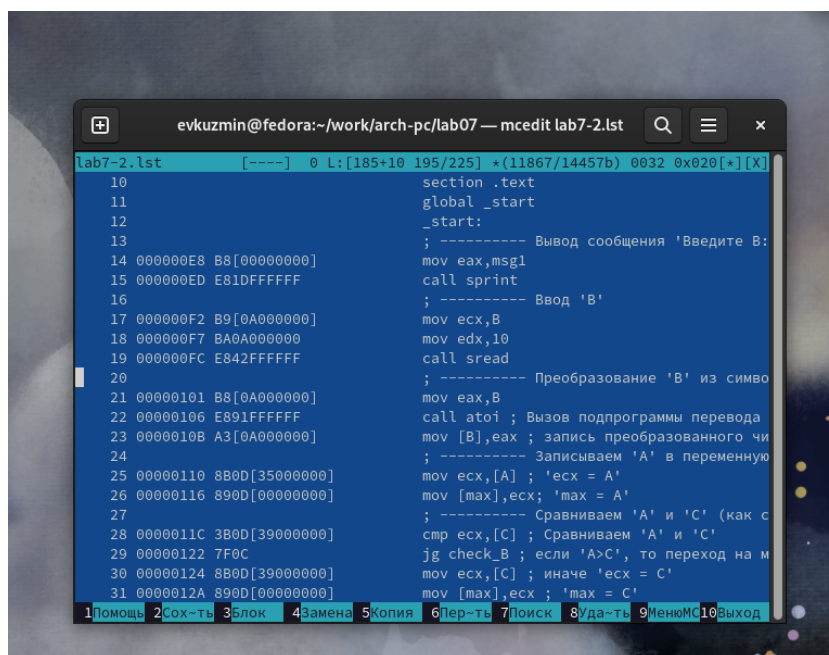
4.2) Изучение структуры файла листинга.

Получаю файл листинга для программы lab7-2, указав ключ -l и введя имя листинга в командной строке. (рис. 4.10).

```
[evkuzmin@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[evkuzmin@fedora lab07]$ mcedit lab7-2.lst
```

Рис. 4.10: Получение файла

Открываю полученный файл листинга в mcedit (рис. 4.11).



```
lab7-2.lst  [----]  0 L:[185+10 195/225] *(11867/14457b) 0032 0x020[*][X]
10                                     section .text
11                                     global _start
12                                     _start:
13                                     ; ----- Вывод сообщения 'Введите B:
14 000000E8 B8[00000000]               mov eax,msg1
15 000000ED E81DFFFFFF               call sprint
16                                     ; ----- Ввод 'B'
17 000000F2 B9[0A000000]               mov ecx,B
18 000000F7 BA0A000000               mov edx,10
19 000000FC E842FFFFFF               call sread
20                                     ; ----- Преобразование 'B' из симво
21 00000101 B8[0A000000]               mov eax,B
22 00000106 E891FFFFFF               call atoi ; Вызов подпрограммы перевода
23 0000010B A3[0A000000]               mov [B],eax ; запись преобразованного чи
24                                     ; ----- Записываем 'A' в переменную
25 00000110 8B0D[35000000]             mov ecx,[A] ; 'ecx = A'
26 00000116 890D[00000000]             mov [max],ecx ; 'max = A'
27                                     ; ----- Сравниваем 'A' и 'C' (как с
28 0000011C 3B0D[39000000]             cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 00000122 7F0C                       jg check_B ; если 'A>C', то переход на м
30 00000124 8B0D[39000000]             mov ecx,[C] ; иначе 'ecx = C'
31 0000012A 890D[00000000]             mov [max],ecx ; 'max = C'
```

Рис. 4.11: Открытие файла в mcedit

Объяснение строк:

Инструкция `mov ecx,B` используется, чтобы положить адрес вводимой строки `B` в регистр `ecx`. `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.

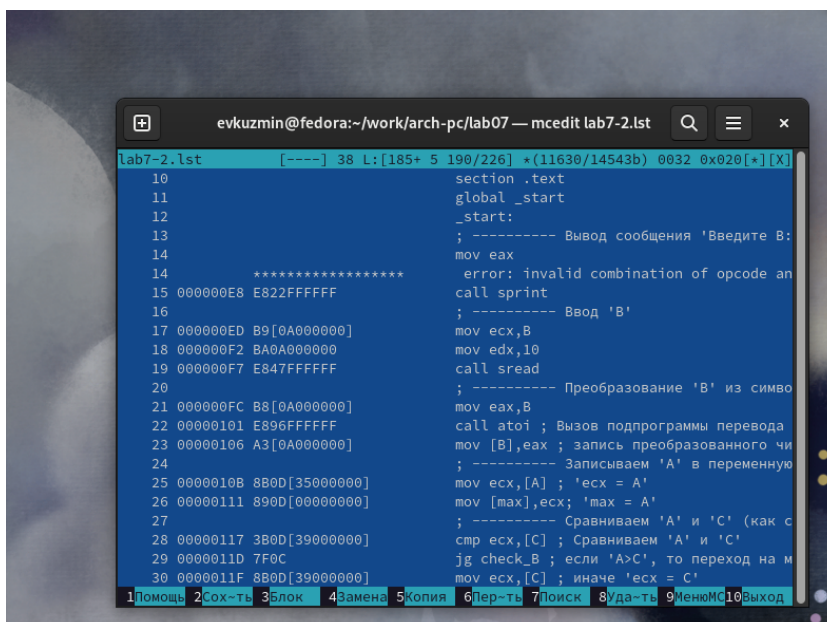
Открываю заново файл для редактирования и убираю один из операндов в инструкции двумя операндами. Заново создаю листинг. (рис. 4.12).



```
[evkuzmin@fedora lab07]$ gedit lab7-2.asm
[evkuzmin@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:14: error: invalid combination of opcode and operands
[evkuzmin@fedora lab07]$ mcedit lab7-2.lst
```

Рис. 4.12: Редактирование файла, создание листинга

Мы видим ошибку, но файл листинга сойдётся. Открываю его. Также на месте строки находится сообщение об ошибке. (рис. 4.13).

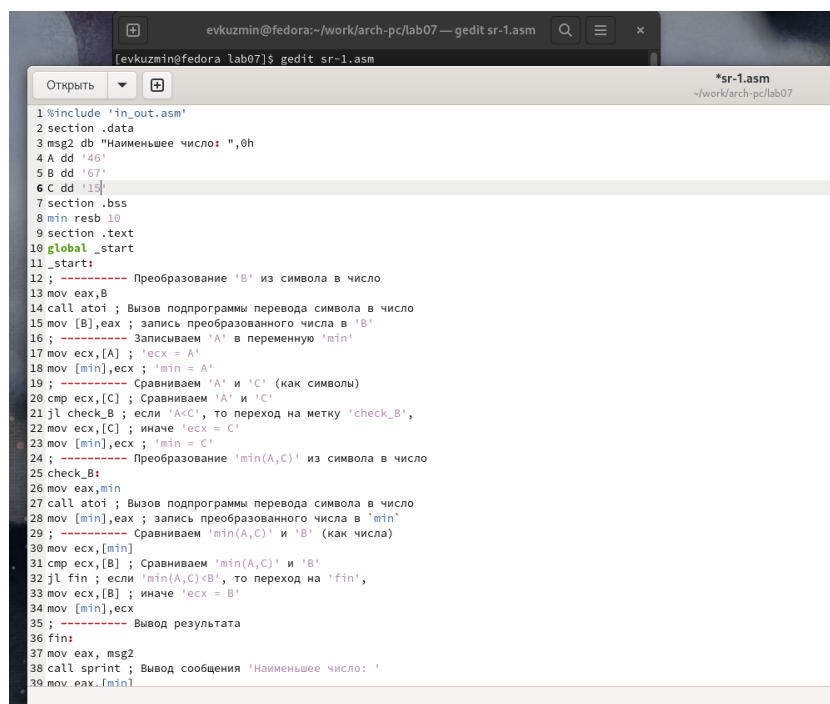


```
lab7-2.lst  [----] 38 L: [185+ 5 190/226] *(11630/14543b) 0032 0x020[*][X]
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B:
14 mov eax
14 ***** error: invalid combination of opcode and operands
15 000000E8 E822FFFFFF call sprint
16 ; ----- Ввод 'B'
17 000000ED B9[0A000000] mov ecx,B
18 000000F2 BA0A000000 mov edx,10
19 000000F7 E847FFFFFF call sread
20 ; ----- Преобразование 'B' из символа
21 000000FC B8[0A000000] mov eax,B
22 00000101 E896FFFFFF call atoi ; Вызов подпрограммы перевода
23 00000106 A3[0A000000] mov [B],eax ; запись преобразованного числа
24 ; ----- Записываем 'A' в переменную
25 0000010B 8B0D[35000000] mov ecx,[A] ; 'ecx = A'
26 00000111 890D[00000000] mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как строки)
28 00000117 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 0000011D 7F0C jg check_B ; если 'A>C', то переход на m
30 0000011F 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
```

Рис. 4.13: Открытие листинга

4.3) Выполнение заданий для самостоятельной работы

Создаю файл sr-1.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для определения наименьшего числа из 3-х, предложенных в варианте 7, полученным мною при выполнении прошлой лабораторной работы (рис. 4.14)



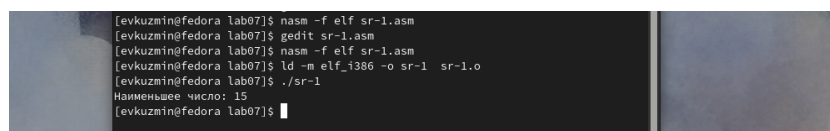
```

1 %include 'in_out.asm'
2 section .data
3 msg2 db "Наименьшее число: ",0h
4 A dd '46'
5 B dd '67'
6 C dd '15'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Преобразование 'B' из символа в число
13 mov eax,B
14 call atoi ; Вызов подпрограммы перевода символа в число
15 mov [B],eax ; запись преобразованного числа в 'B'
16 ; ----- Записываем 'A' в переменную 'min'
17 mov ecx,[A] ; 'ecx = A'
18 mov [min],ecx ; 'min = A'
19 ; ----- Сравниваем 'A' и 'C' (как символы)
20 cmp ecx,[C] ; Сравниваем 'A' и 'C'
21 jl check_B ; если 'A < C', то переход на метку 'check_B',
22 mov ecx,[C] ; иначе 'ecx = C'
23 mov [min],ecx ; 'min = C'
24 ; ----- Преобразование 'min(A,C)' из символа в число
25 check_B:
26 mov eax,min
27 call atoi ; Вызов подпрограммы перевода символа в число
28 mov [min],eax ; запись преобразованного числа в 'min'
29 ; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
30 mov ecx,[min]
31 cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
32 jl fin ; если 'min(A,C) < B', то переход на 'fin',
33 mov ecx,[B] ; иначе 'ecx = B'
34 mov [min],ecx
35 ; ----- Вывод результата
36 fin:
37 mov eax,msg2
38 call printf ; Вывод сообщения 'Наименьшее число: '
39 mov eax,[min]

```

Рис. 4.14: Создание и редактирование файла

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис. 4.16)



```

[evkuzmin@fedora lab07]$ nasm -f elf sr-1.asm
[evkuzmin@fedora lab07]$ gedit sr-1.asm
[evkuzmin@fedora lab07]$ nasm -f elf sr-1.asm
[evkuzmin@fedora lab07]$ ld -m elf_i386 -o sr-1 sr-1.o
[evkuzmin@fedora lab07]$ ./sr-1
Наименьшее число: 15
[evkuzmin@fedora lab07]$

```

Рис. 4.15: Компиляция, обработка и запуск исполняемого файла

Создаю файл sr-2.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для своего 7-го варианта: $f = 6a$, если $x=a$ и $f = x+a$, если $x \neq a$ (рис. 4.16)

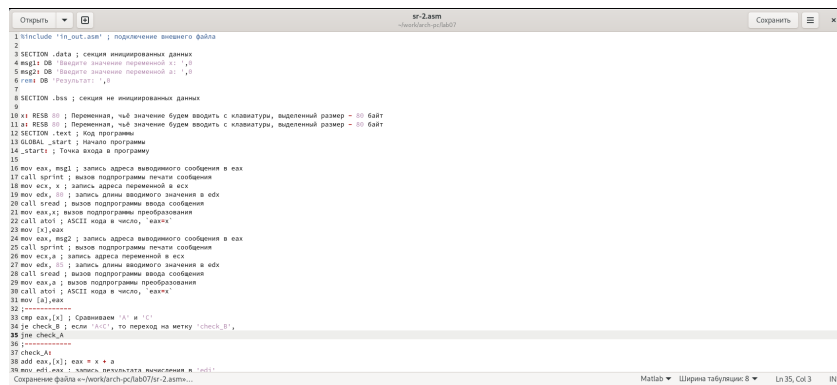


Рис. 4.16: Создание и редактирование файла

Компилирую, обрабатываю и конце концов запускаю исполняемый файл. Ввожу предложенные значения, и, сделав проверку, понимаю, что программа работает верно(рис. 4.17)

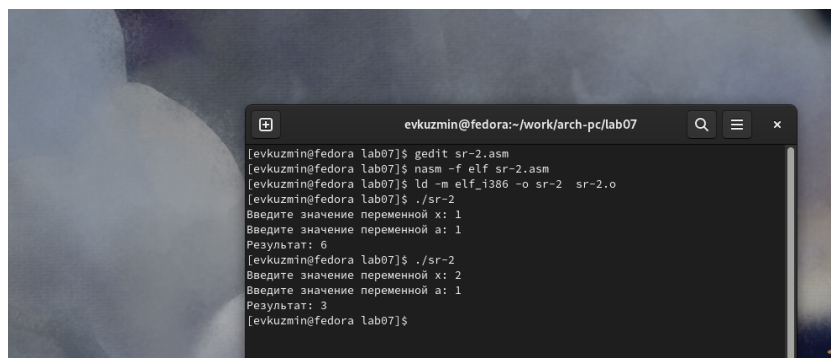


Рис. 4.17: Компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для определения наименьшего числа из 3-х, предложенных в варианте 7.

```
%include 'in_out.asm' section .data msg2 db "Наименьшее число: ",0h
A dd '46' B dd '67' C dd '15' section .bss min resb 10 section .text
global _start _start: ; ----- 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 'min' mov ecx,[A] ; 'ecx = A' mov [min],ecx
```

```

; 'min = A' ; ----- Сравниваем 'A' и 'C' (как символы) cmp
ecx,[C] ; Сравниваем 'A' и 'C' jl check_B ; если 'A<C', то переход на
метку 'check_B', mov ecx,[C] ; иначе 'ecx = C' mov [min],ecx ; 'min =
C' ; ----- Преобразование 'min(A,C)' из символа в число check_B:
mov eax,min call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в 'min' ; -----
Сравниваем 'min(A,C)' и 'B' (как числа) mov ecx,[min] cmp ecx,[B] ;
Сравниваем 'min(A,C)' и 'B' jl fin ; если 'min(A,C)<B', то переход
на 'fin', mov ecx,[B] ; иначе 'ecx = B' mov [min],ecx ; -----
Вывод результата fin: mov eax, msg2 call sprint ; Вывод сообщения
'Наименьшее число: ' mov eax,[min] call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Листинг 4.2 - Программа для вычисления значения системы из варианта 7.

```

#include 'in_out.asm' ; подключение внешнего файла SECTION .data ;
секция иницированных данных msg1: DB 'Введите значение переменной x:
',0 msg2: DB 'Введите значение переменной a: ',0 rem: DB 'Результат:
',0 SECTION .bss ; секция не иницированных данных x: RESB 80 ;
Переменная, чьё значение будем вводить с клавиатуры, выделенный
размер - 80 байт a: RESB 80 ; Переменная, чьё значение будем вводить
с клавиатуры, выделенный размер - 80 байт SECTION .text ; Код программы
GLOBAL _start ; Начало программы _start: ; Точка входа в программу
mov eax, msg1 ; запись адреса выводимого сообщения в eax call sprint
; вызов подпрограммы печати сообщения mov ecx, x ; запись адреса
переменной в ecx mov edx, 80 ; запись длины вводимого значения в
edx call sread ; вызов подпрограммы ввода сообщения mov eax,x; вызов
подпрограммы преобразования call atoi ; ASCII кода в число, 'eax=x'
mov [x],eax mov eax, msg2 ; запись адреса выводимого сообщения в
eax call sprint ; вызов подпрограммы печати сообщения mov ecx,a ;
запись адреса переменной в ecx mov edx, 85 ; запись длины вводимого

```


значения в `edx` `call sread` ; вызов подпрограммы ввода сообщения `mov`
`eax,a` ; вызов подпрограммы преобразования `call atoi` ; ASCII кода в
 число, ``eax=x` mov [a],eax` ;----- `cmp eax,[x]` ; Сравниваем
 'A' и 'C' `je check_B` ; если 'A<C', то переход на метку 'check_B', `jne`
`check_A` ;----- `check_A: add eax,[x]; eax = x + a mov edi,eax` ;
 запись результата вычисления в 'edi' `jmp _end` ;----- `check_B:`
`mov ebx,6` ; запись значения 6 в регистр `ebx mul ebx; EAX=EAX*6 mov`
`edi,eax` ; запись результата вычисления в 'edi' `jmp _end` ; ---- Вывод
 результата на экран `_end: mov eax,ret` ; вызов подпрограммы печати
`call sprint` ; сообщения 'Результат: ' `mov eax,edi` ; вызов подпрограммы
 печати значения `call iprintLF` ; из 'edi' в виде символов `call quit` ;
 вызов подпрограммы завершения

5 Выводы

При выполнении лабораторной работы я изучил команды условного и безусловного переходов, приобрел практический опыт в написании программ с использованием переходов, познакомился с назначением и структурой файла листинга

Список литературы

Архитектура компьютера и ЭВМ