Отчёт по лабораторной работе №7

Дисциплина: Архитектура Компьютера

Курилко-Рюмин Евгений

Содержание

# 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

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

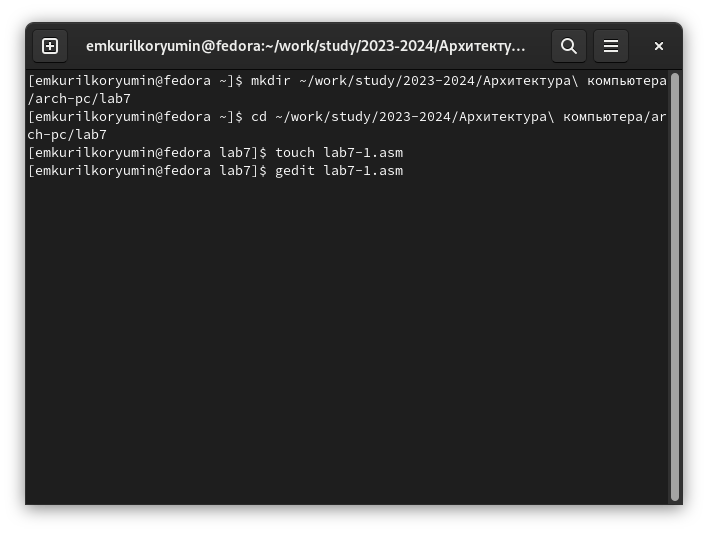
# 3 Теоретическое введение

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.1).



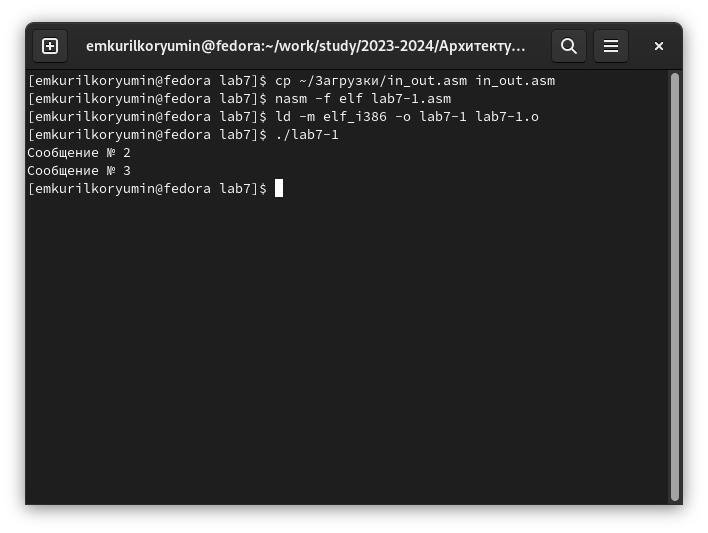
Работа с директориями и создание файла

Открываю созданный файл lab7-1.asm, вставляю в него следующую программу: (рис.2).



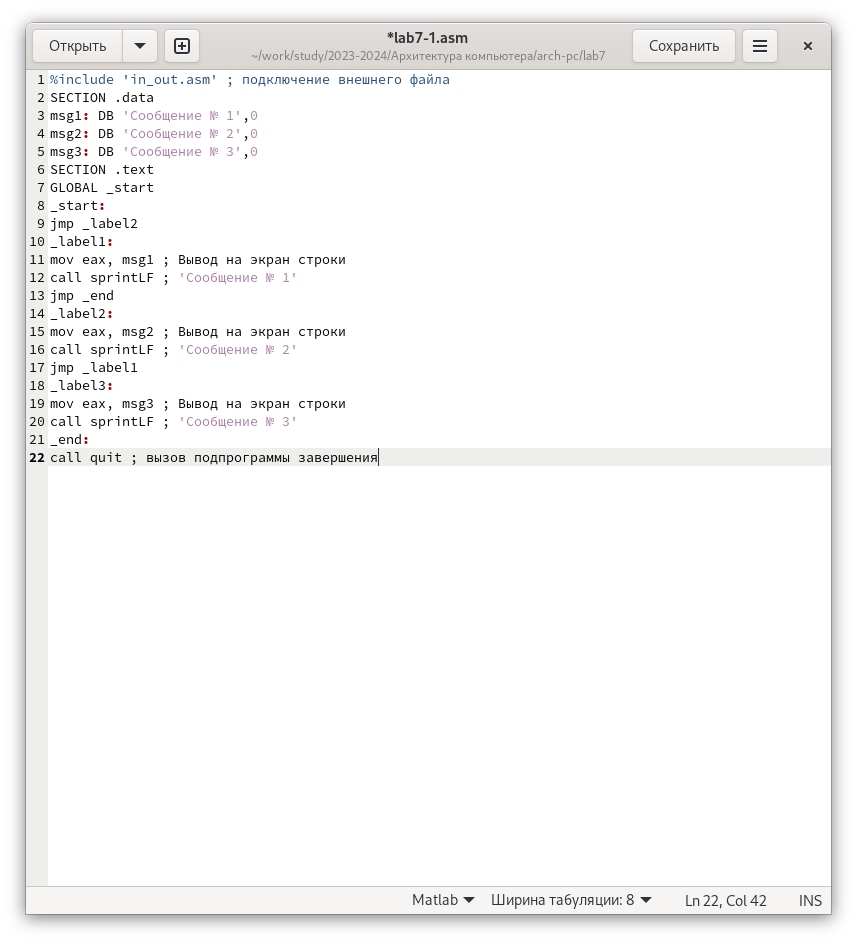
Редактирование файла

Копирую файл in\_out.asm с помощью утилиты cp в этот каталог,так как 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.3).



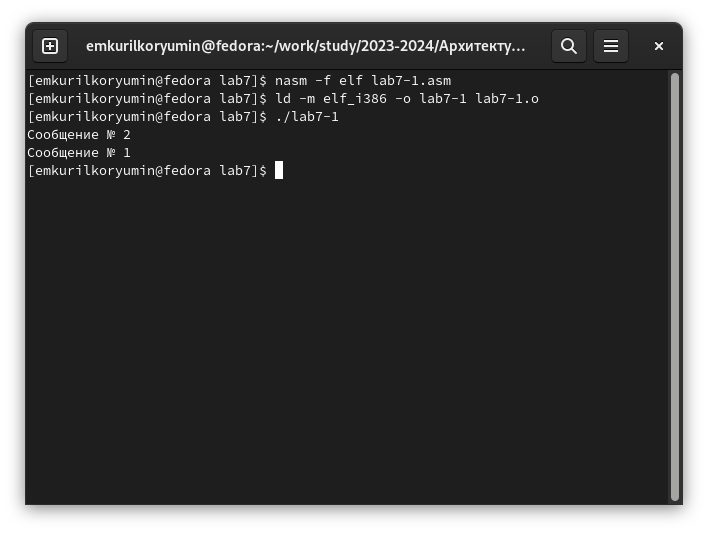
Подготовка и исполнение файла

Добавляю в текст метки jmp\_label1 jmp\_end (рис.4).



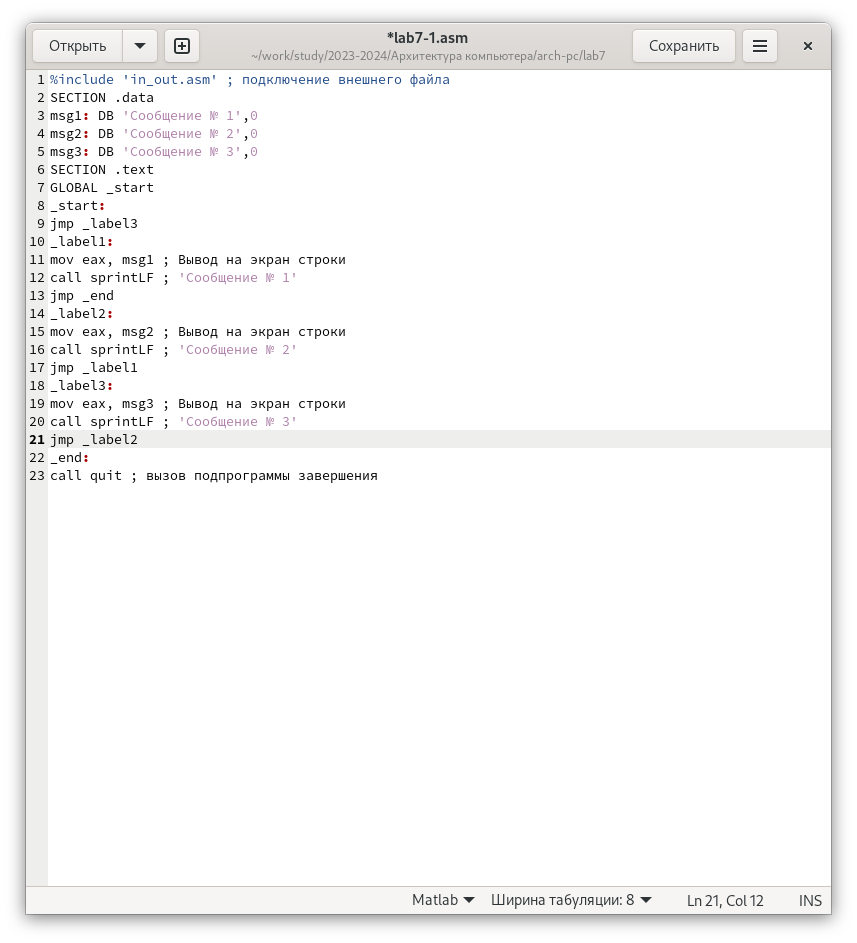
Редактирование файла

Создаю новый исполняемый файл программы и запускаю его.Инструкция jmp позволяет осуществлять переходы не только вперед но и назад. (рис.5).



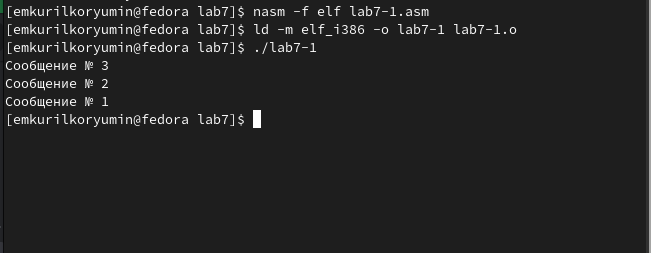
Запуск исполняемого файла

Изменяю метки jmp в программе, чтобы выводились сообщения в порядке 3,2,1 (рис.6).



Создание и редактирование файла

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. Видим, что все работает так, как нужно. (рис.7).



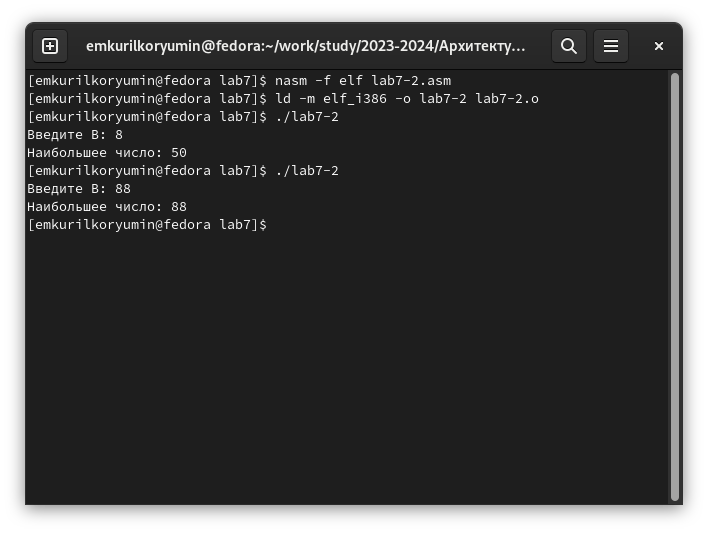
Компиляция и запуск исполняемого файла

Создаю файл lab7-2.asm. Редактирую его, вводя предлагаемую программу. (рис.8).



Редактирование файла

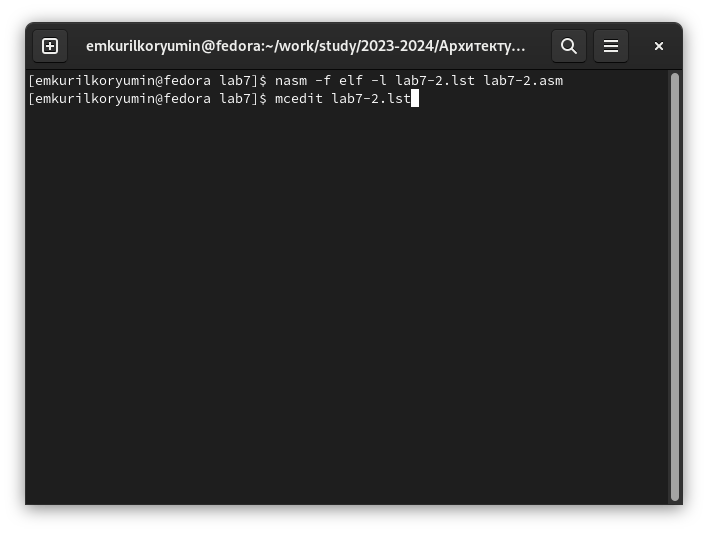
Создаю исполняемый файл и проверяю его работу для разных значений В. (рис.9).



Компиляция и запуск исполняемого

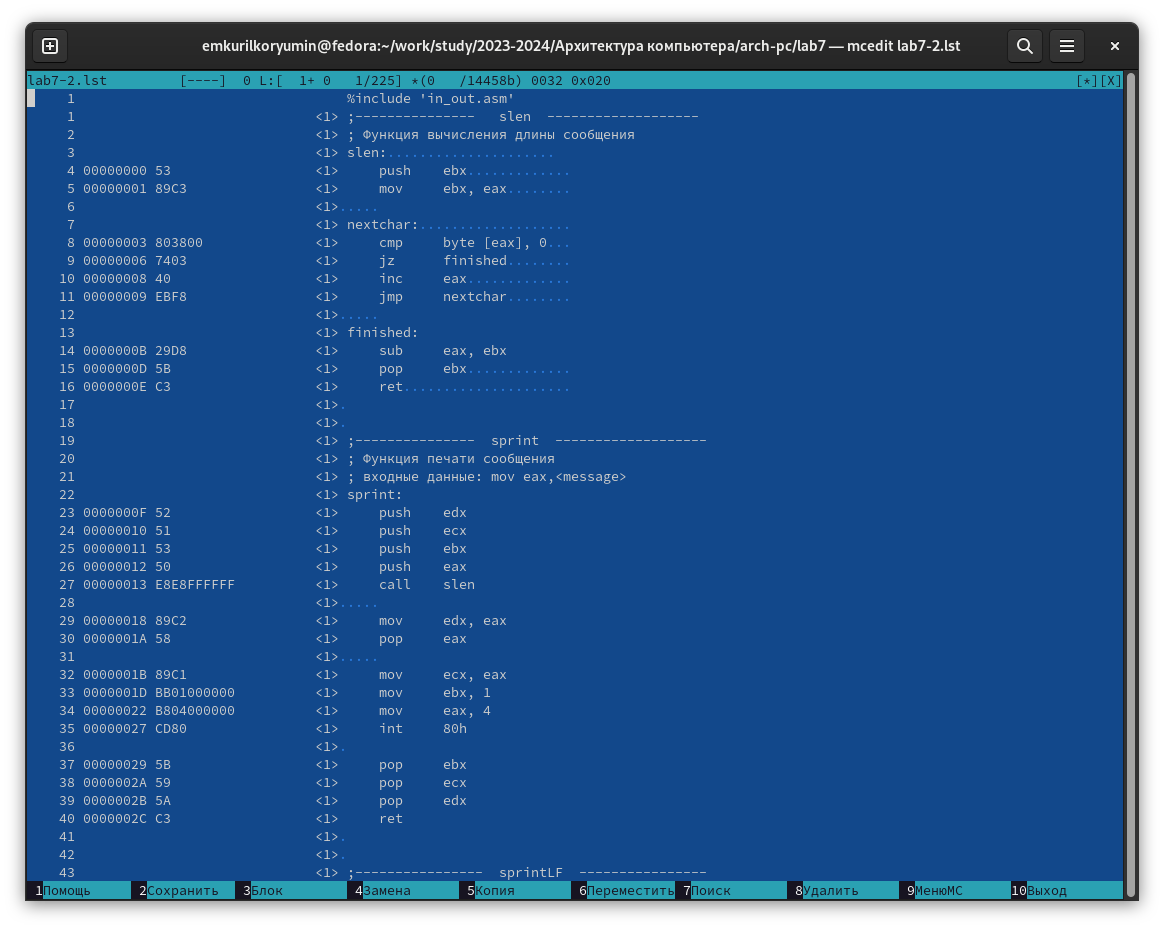
4.2) Изучение структуры файла листинга.

Получаю файл листинга для программы lab7-2, указав ключ -l и введя имя листинга в командной строке. (рис.10).



Получение файла

Открываю полученный файл листинга в mcedit (рис.11).

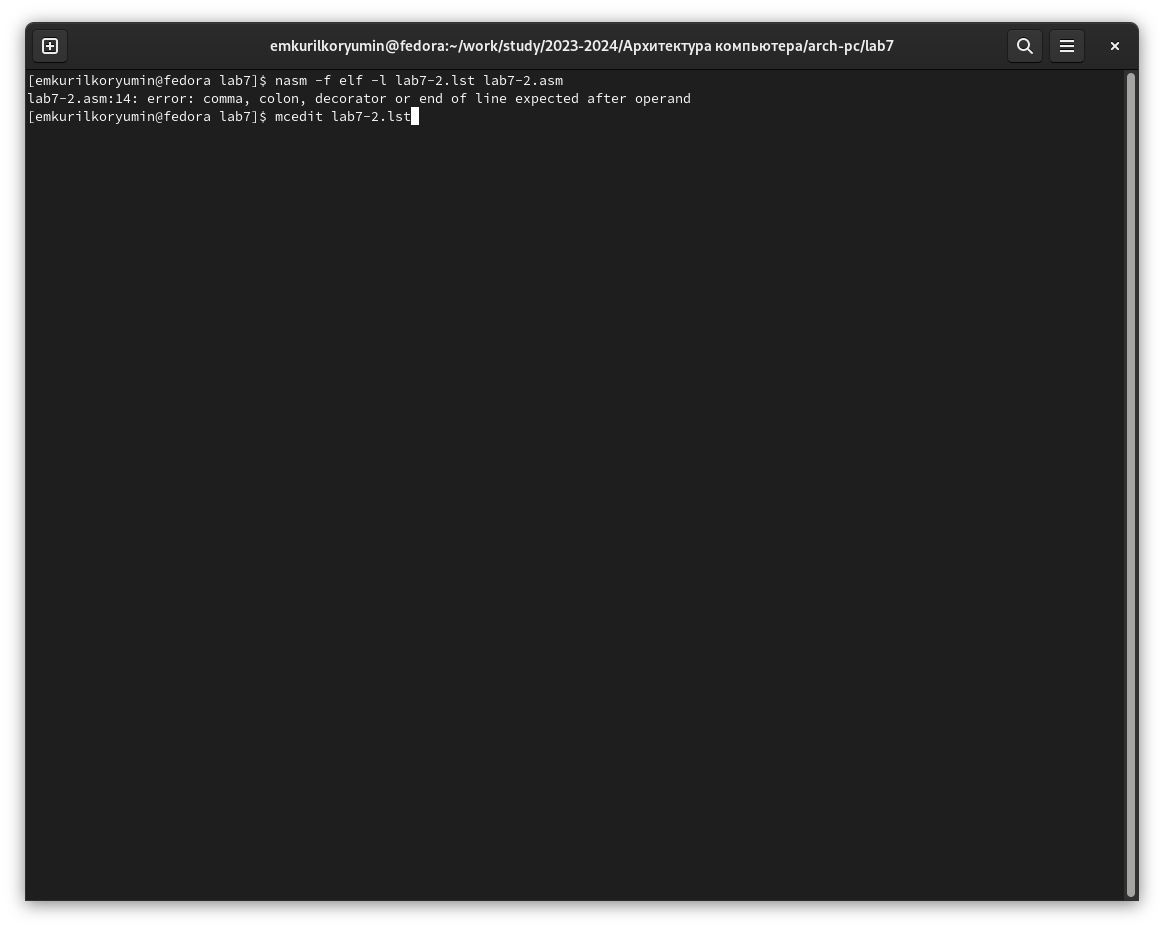


Открытие файла в mcedit

Объяснение строк:

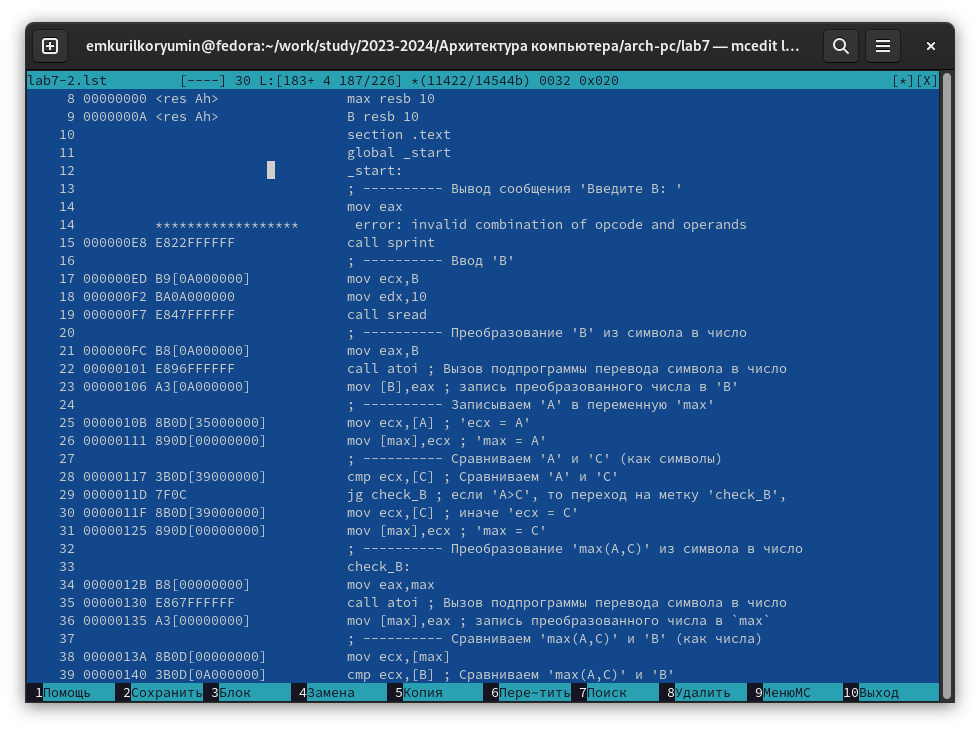
Инструкция mov ecx,B используется, чтобы положить адрес вводимой строки B в регистр ecx. call sread - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры. call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число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

Открываю заново файл для редактирования и убираю один из операндов в инструкции двумя операндами. Заново создаю листинг. (рис.12).



Редактирование файла и создание листинга

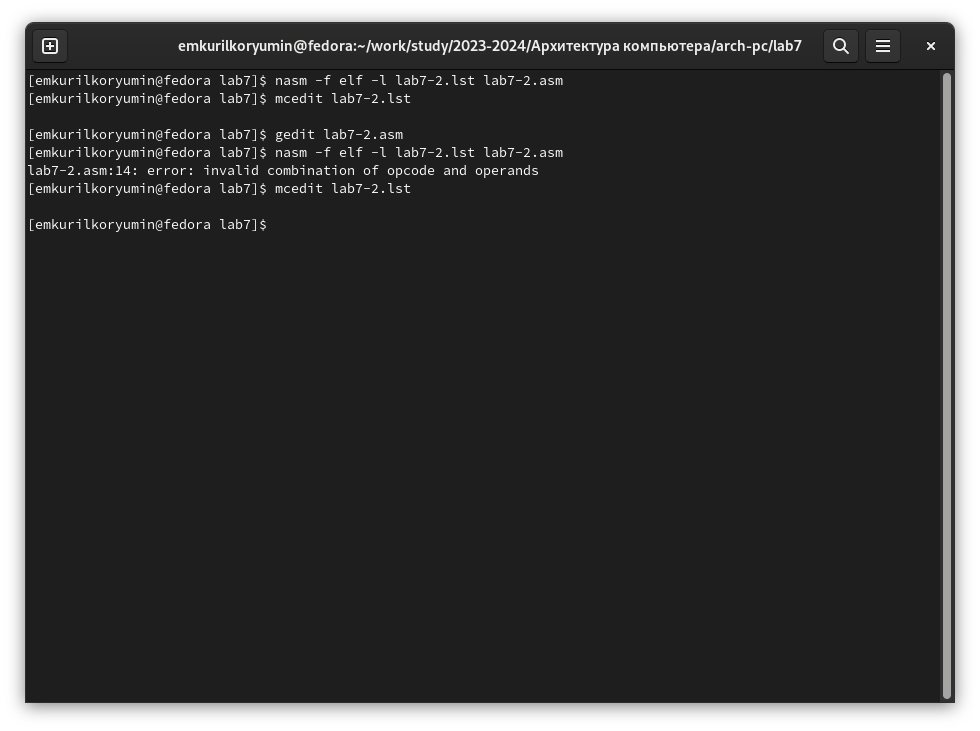
Мы видим ошибку, но файл листинга сойдаётся. Открываю его. Также на месте строки находится сообщение об ошибке. (рис.13).



Открытие листинга

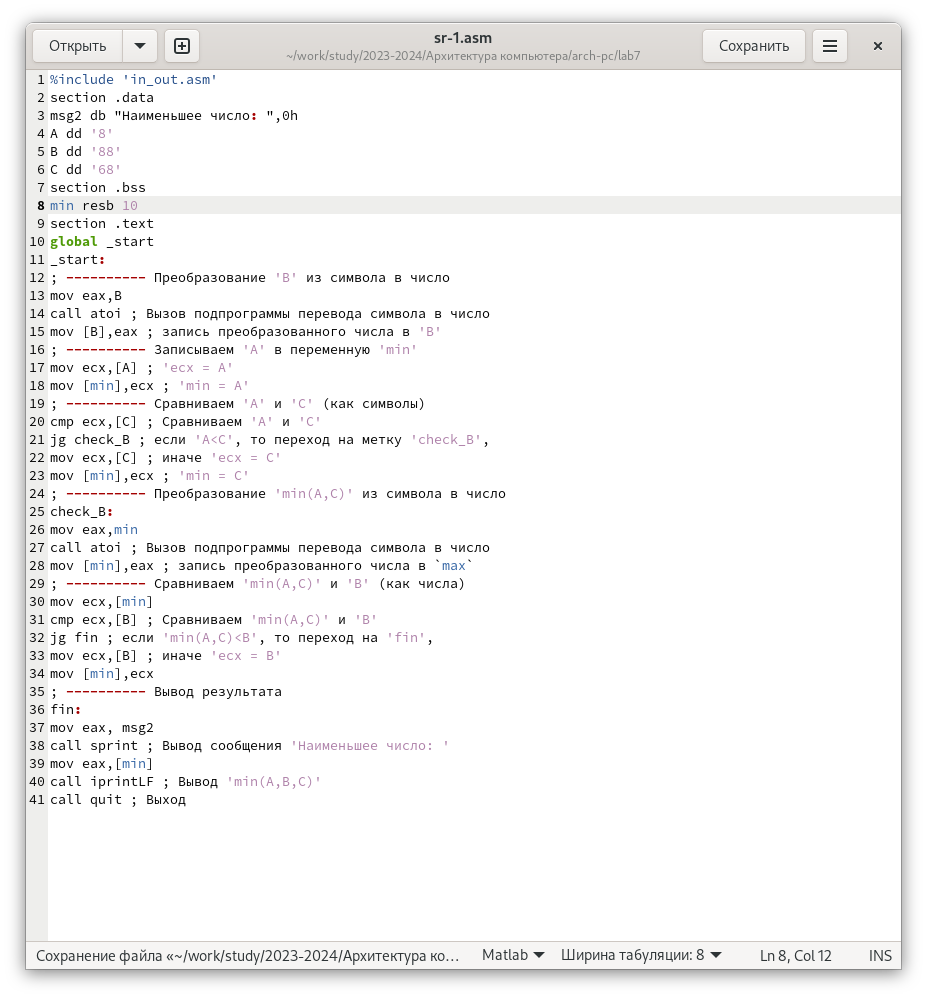
4.3) Выполнение заданий для самостоятельной работы

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 4, полученным в предыдушей лабораторной работы (рис.14)



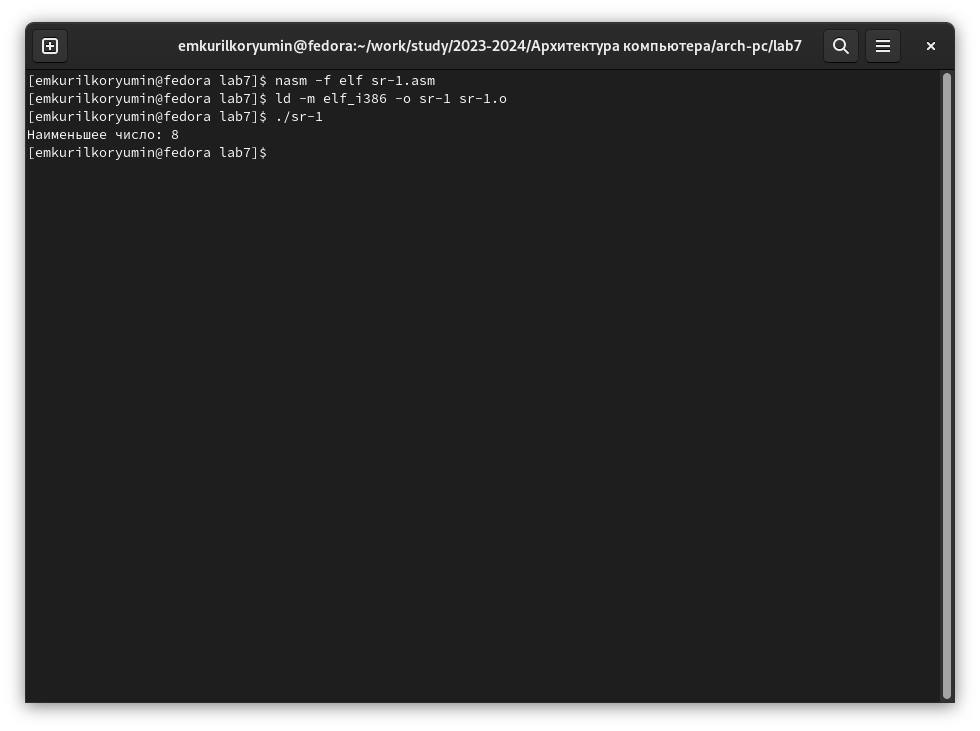
Создание и редактирование файла

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис.15)



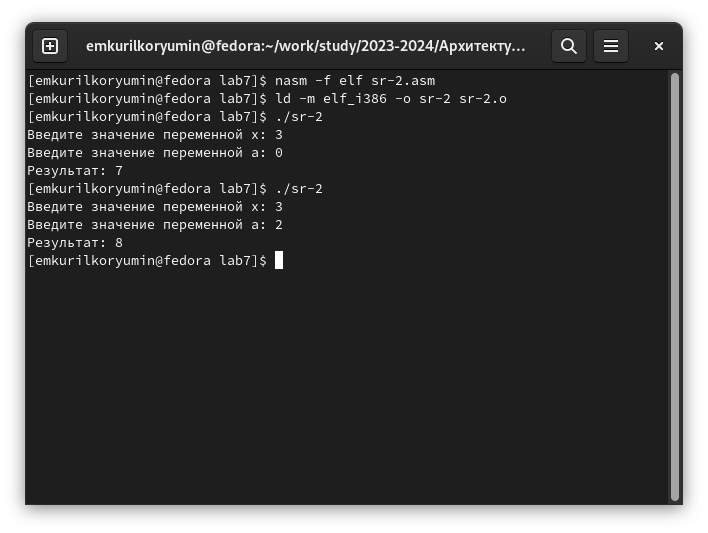
Компиляция, обработка и запуск исполняемого файла

Создаю файл sr-2.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для своего 4-го варианта: f = 2x + 1, если a=0 и f = 2x+a, если a!=0 (рис.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.17)



Компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для определения наименьшего числа из 3-х, предложенных в варианте 4.

``%include 'in\_out.asm' section .data msg2 db "Наименьшее число: ",0h A dd '8' B dd '88' C dd '68' 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 'С' (как символы) cmp ecx,[C] ; Сравниваем 'A' и 'С' 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 вmax` ; ———- 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 4.

%include ‘in\_out.asm’ ; подключение внешнего файла

SECTION .data ; секция инициированных данных msg1: DB ‘Введите значение переменной х:’,0 msg2: DB ‘Введите значение переменной a:’,0 rem: DB ‘Результат:’,0

SECTION .bss ; секция не инициированных 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 выводимиого 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 выводимиого 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,0 ; проверка что А != 0 je check\_B ; если ‘A=0’, то переход на метку ‘check\_B’, jne check\_A ;———— check\_A: mov eax,[x]; shl eax,1; add eax,[a]; eax = 2x + a mov edi,eax ; запись результата вычисления в ‘edi’ jmp \_end ;———— check\_B: mov eax,[x]; shl eax,1; add eax,1; EAX=EAX+1 mov edi,eax ; запись результата вычисления в ‘edi’ jmp \_end ; —- Вывод результата на экран \_end: mov eax,rem ; 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](https://esystem.rudn.ru/pluginfile.php/2089086/mod_resource/content/0/Лабораторная%20работа%20№6.%20Арифметические%20операции%20в%20NASM..pdf)