

Отчёт по лабораторной работе №7

Дисциплина: Архитектура Компьютера

Курилко-Рюмин Евгений

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Работа с директориями и создание файла	8
4.2	Редактирование файла	9
4.3	Подготовка и исполнение файла	10
4.4	Редактирование файла	11
4.5	Запуск исполняемого файла	12
4.6	Создание и редактирование файла	13
4.7	Компиляция и запуск исполняемого файла	13
4.8	Редактирование файла	14
4.9	Компиляция и запуск исполняемого	15
4.10	Получение файла	16
4.11	Открытие файла в mcedit	16
4.12	Редактирование файла и создание листинга	17
4.13	Открытие листинга	18
4.14	Создание и редактирование файла	19
4.15	Компиляция, обработка и запуск исполняемого файла	20
4.16	Создание и редактирование файла	21
4.17	Компиляция, обработка и запуск исполняемого файла	21

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:

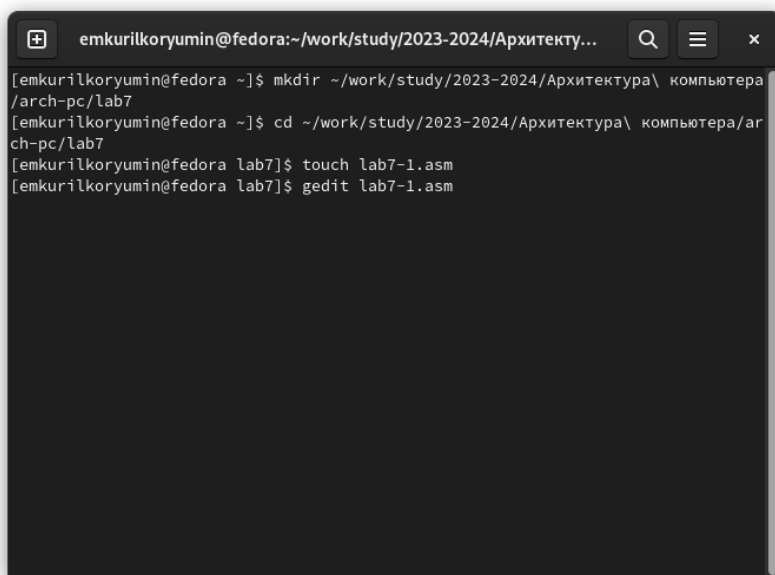
- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную 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.

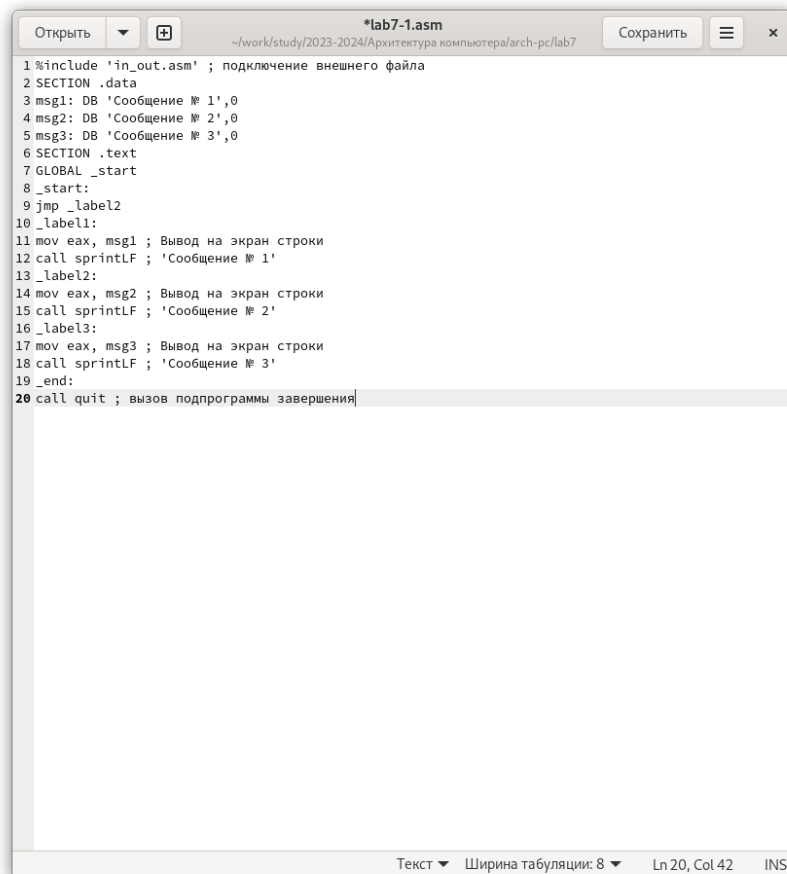
С помощью утилиты `mkdir` создаю директорию `lab07` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab7-1.asm`. (рис.1).



```
emkurilkoryumin@fedora:~/work/study/2023-2024/Архитекту...
[emkurilkoryumin@fedora ~]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера
/arch-pc/lab7
[emkurilkoryumin@fedora ~]$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab7
[emkurilkoryumin@fedora lab7]$ touch lab7-1.asm
[emkurilkoryumin@fedora lab7]$ gedit lab7-1.asm
```

Рис. 4.1: Работа с директориями и создание файла

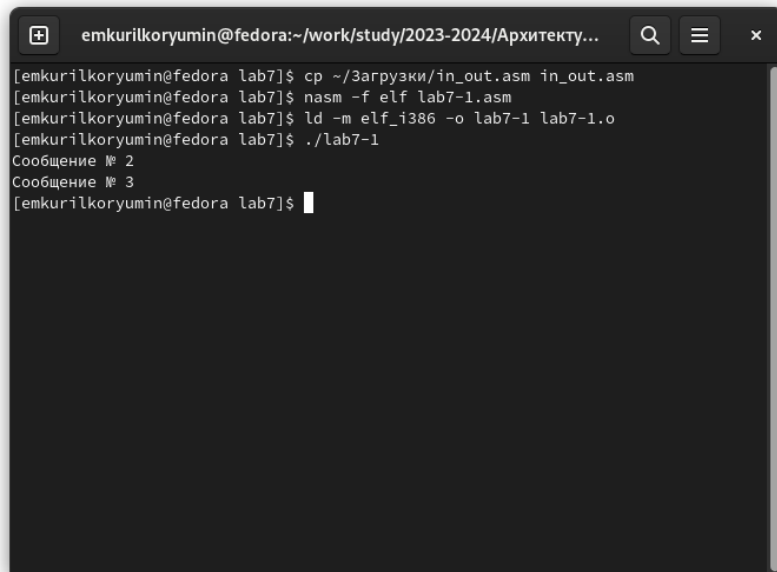
Открываю созданный файл `lab7-1.asm`, вставляю в него следующую программу: (рис.2).



```
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 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение № 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintf ; 'Сообщение № 3'
19 _end:
20 call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Редактирование файла

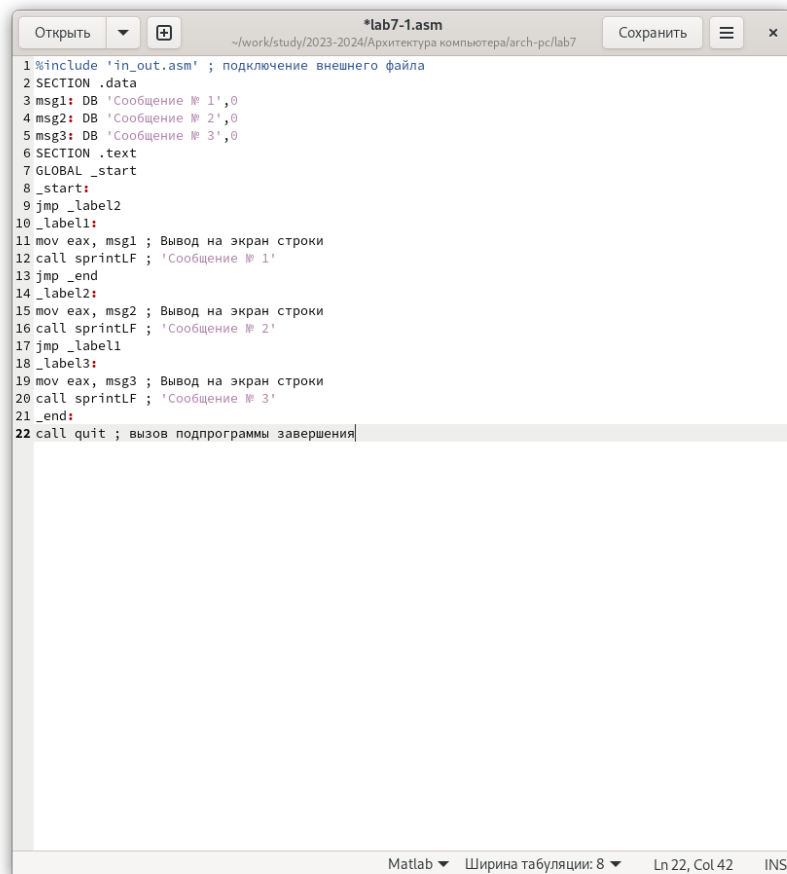
Копирую файл `in_out.asm` с помощью утилиты `cp` в этот каталог, так как он будет использоваться в дальнейшем. Выполняю компиляцию, компоновку файла и запускаю его. Мы видим, что использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения (рис.3).

A terminal window with a dark background and light text. The window title is "emkurilkoryumin@fedora:~/work/study/2023-2024/Архитекту...". The terminal shows the following commands and output:

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

Рис. 4.3: Подготовка и исполнение файла

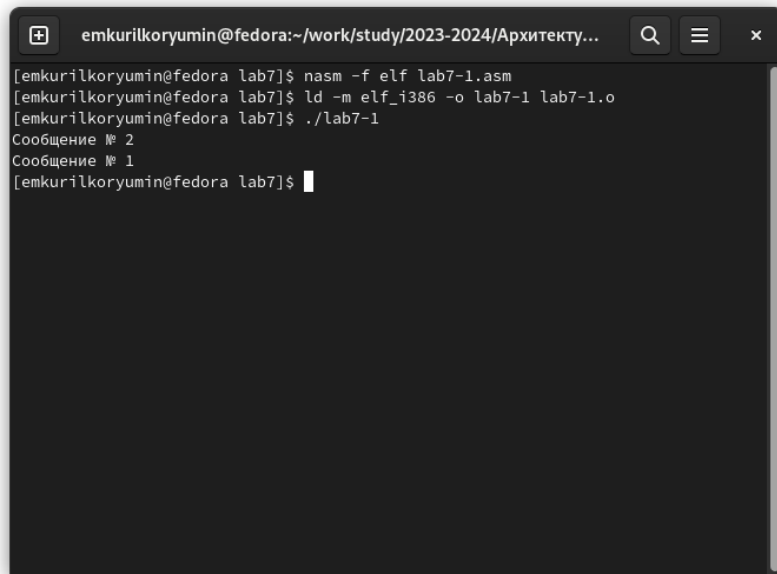
Добавляю в текст метки `jmp_label1 jmp_end` (рис.4).



```
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. Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. (рис.5).

A terminal window with a dark background and light text. The title bar shows the user 'emkurilkoryumin' on a 'fedora' machine, in the directory '~/work/study/2023-2024/Архитекту...'. The terminal contains the following commands and output:

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

Рис. 4.5: Запуск исполняемого файла

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

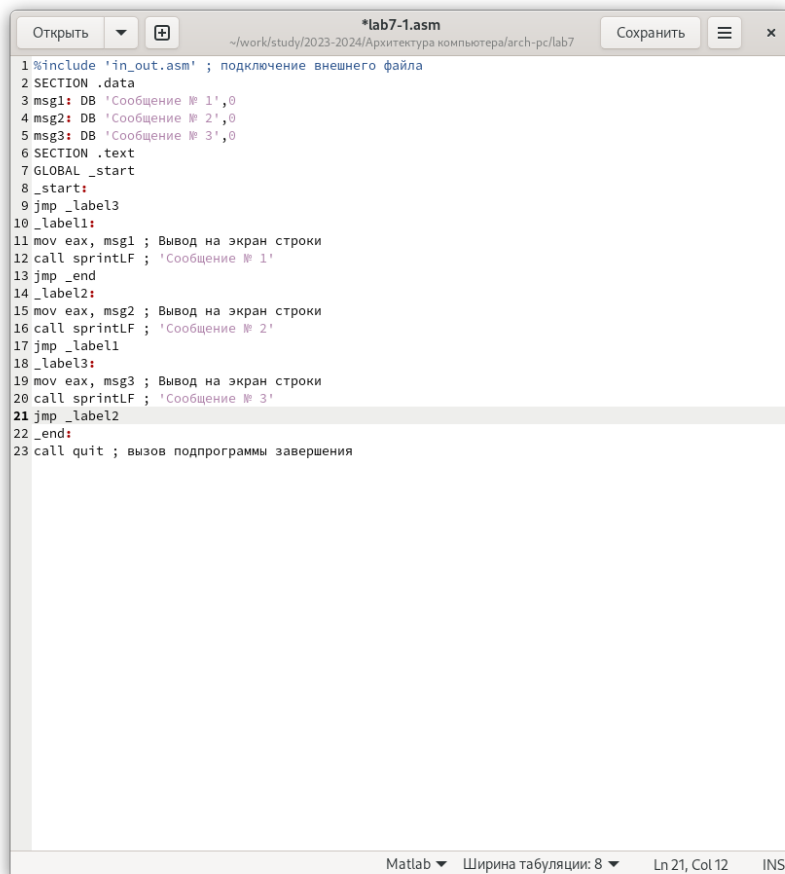


Рис. 4.6: Создание и редактирование файла

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

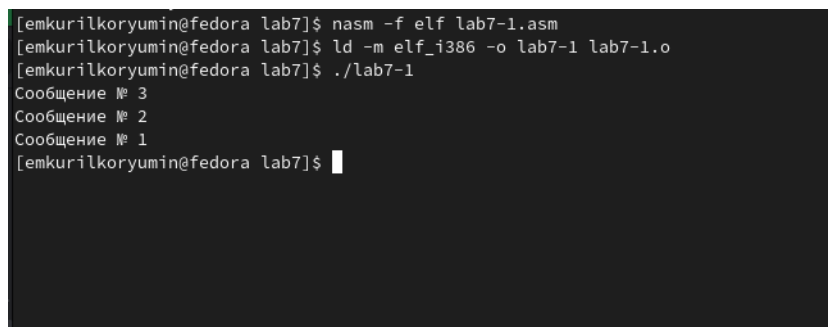


Рис. 4.7: Компиляция и запуск исполняемого файла

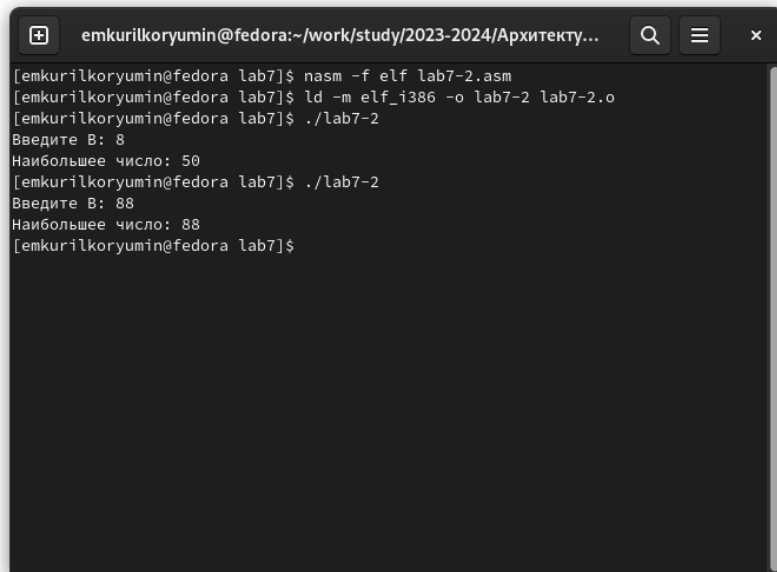
Создаю файл lab7-2.asm. Редактирую его, вводя предлагаемую программу.
(рис.8).



```
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
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
```

Рис. 4.8: Редактирование файла

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

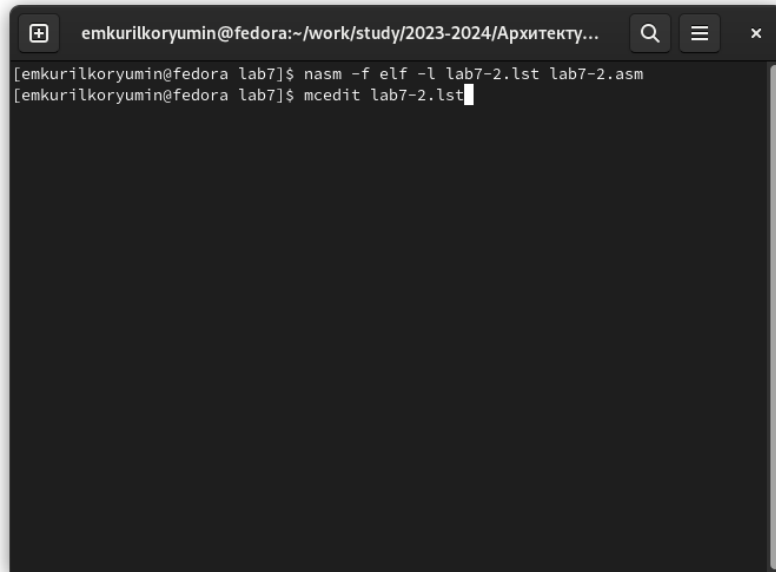
A terminal window with a dark background and light text. The title bar shows the user 'emkurilkoryumin@fedora' and the current directory '~/work/study/2023-2024/Архитекту...'. The terminal contains the following text:

```
[emkurilkoryumin@fedora lab7]$ nasm -f elf lab7-2.asm
[emkurilkoryumin@fedora lab7]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[emkurilkoryumin@fedora lab7]$ ./lab7-2
Введите B: 8
Наибольшее число: 50
[emkurilkoryumin@fedora lab7]$ ./lab7-2
Введите B: 88
Наибольшее число: 88
[emkurilkoryumin@fedora lab7]$
```

Рис. 4.9: Компиляция и запуск исполняемого

4.2) Изучение структуры файла листинга.

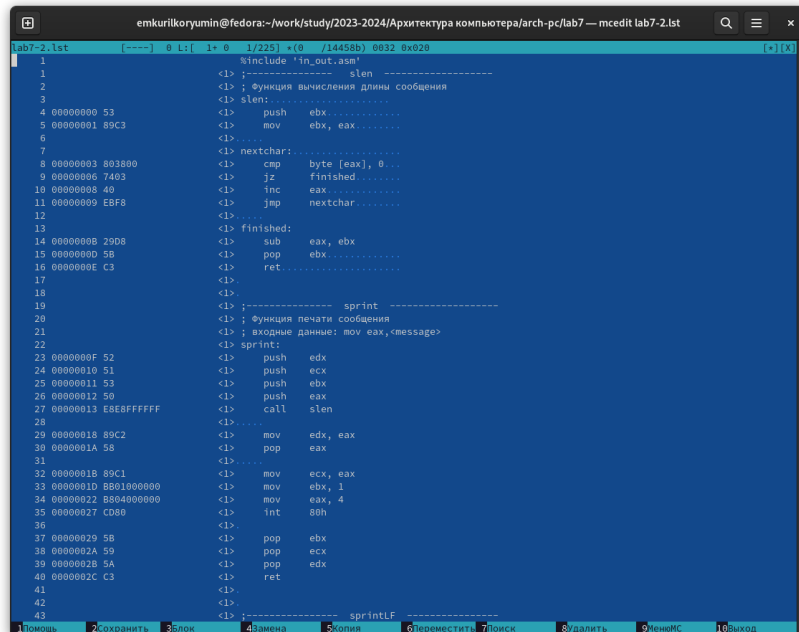
Получаю файл листинга для программы lab7-2, указав ключ -l и введя имя листинга в командной строке. (рис.10).



```
emkurilkoryumin@fedora:~/work/study/2023-2024/Архитекту...
[emkurilkoryumin@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[emkurilkoryumin@fedora lab7]$ mcedit lab7-2.lst
```

Рис. 4.10: Получение файла

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



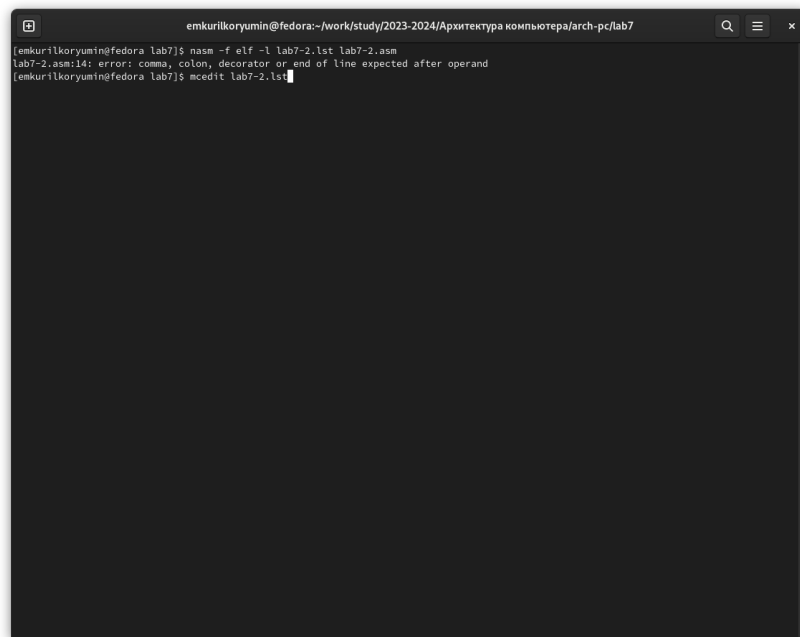
```
lab7-2.lst 0 L: 1+ 0 1/225] * (0 /14458b) 0032 0x020
1      ;include 'in_out.asm'
2      ;----- slen -----
3      ; функция вычисления длины сообщения
4      slen:
5      <1> push ebx
6      <1> mov ebx, eax
7      <1> nextchar:
8      <1> cmp byte [eax], 0
9      <1> jz finished
10     <1> inc eax
11     <1> jmp nextchar
12     <1> finished:
13     <1> sub eax, ebx
14     <1> pop ebx
15     <1> ret
16     <1>
17     <1>
18     <1>
19     ;----- sprint -----
20     ; функция печати сообщения
21     ; входные данные: mov eax, <message>
22     sprint:
23     <1> push edx
24     <1> push ecx
25     <1> push ebx
26     <1> push eax
27     <1> call slen
28     <1>
29     <1> mov edx, eax
30     <1> pop eax
31     <1>
32     <1> mov ecx, eax
33     <1> mov ebx, 1
34     <1> mov eax, 4
35     <1> int 80h
36     <1>
37     <1> pop ebx
38     <1> pop ecx
39     <1> pop edx
40     <1> ret
41     <1>
42     <1>
43     <1>
44     ;----- printf -----
45     printf:
46     <1> push ebx
47     <1> push ecx
48     <1> push edx
49     <1> push eax
50     <1> call printf
51     <1>
52     <1> mov ebx, eax
53     <1> pop eax
54     <1> pop ecx
55     <1> pop ebx
56     <1> ret
```

Рис. 4.11: Открытие файла в mcedit

Объяснение строк:

Инструкция `mov esx, B` используется, чтобы положить адрес вводимой строки `B` в регистр `esx`. `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).

A screenshot of a terminal window with a dark background. The window title is "emkurilkoryumin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab7". The terminal shows the following commands and output:

```
[emkurilkoryumin@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:14: error: comma, colon, decorator or end of line expected after operand
[emkurilkoryumin@fedora lab7]$ mcedit lab7-2.lst
```

Рис. 4.12: Редактирование файла и создание листинга

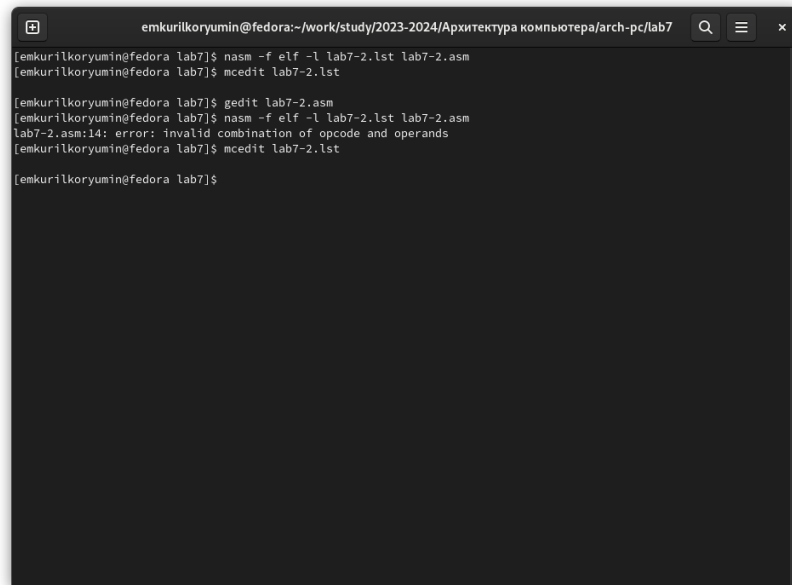
Мы видим ошибку, но файл листинга сойдётся. Открываю его. Также на месте строки находится сообщение об ошибке. (рис.13).

```
lab7-2.lst [-----] 30 L:[183+ 4 187/226] x(11422/14544b) 0032 0x020 [*)(X]
8 00000000 <res Ah> max resb 10
9 0000000A <res Ah> B resb 10
10
11 section .text
12 global _start
13 _start:
14 ; ----- Вывод сообщения 'Введите B: '
15 mov eax
16 error: invalid combination of opcode and operands
17 call sprint
18 ; ----- Ввод 'B'
19 mov ecx,B
20 mov edx,10
21 call sread
22 ; ----- Преобразование 'B' из символа в число
23 mov eax,B
24 call atoi ; Вызов подпрограммы перевода символа в число
25 mov [B],eax ; запись преобразованного числа в 'B'
26 ; ----- Записываем 'A' в переменную 'max'
27 mov ecx,[A] ; 'ecx = A'
28 mov [max],ecx ; 'max = A'
29 ; ----- Сравниваем 'A' и 'C' (как символы)
30 cmp ecx,[C] ; Сравниваем 'A' и 'C'
31 jg check_B ; если 'A>C', то переход на метку 'check_B'
32 mov ecx,[C] ; иначе 'ecx = C'
33 mov [max],ecx ; 'max = C'
34 ; ----- Преобразование 'max(A,C)' из символа в число
35 check_B:
36 mov eax,max
37 call atoi ; Вызов подпрограммы перевода символа в число
38 mov [max],eax ; запись преобразованного числа в 'max'
39 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
40 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
```

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

A terminal window with a dark background and light text. The title bar shows the user 'emkurilkoryumin' on a 'fedora' machine, in the directory '~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab7'. The terminal contains the following commands and output:

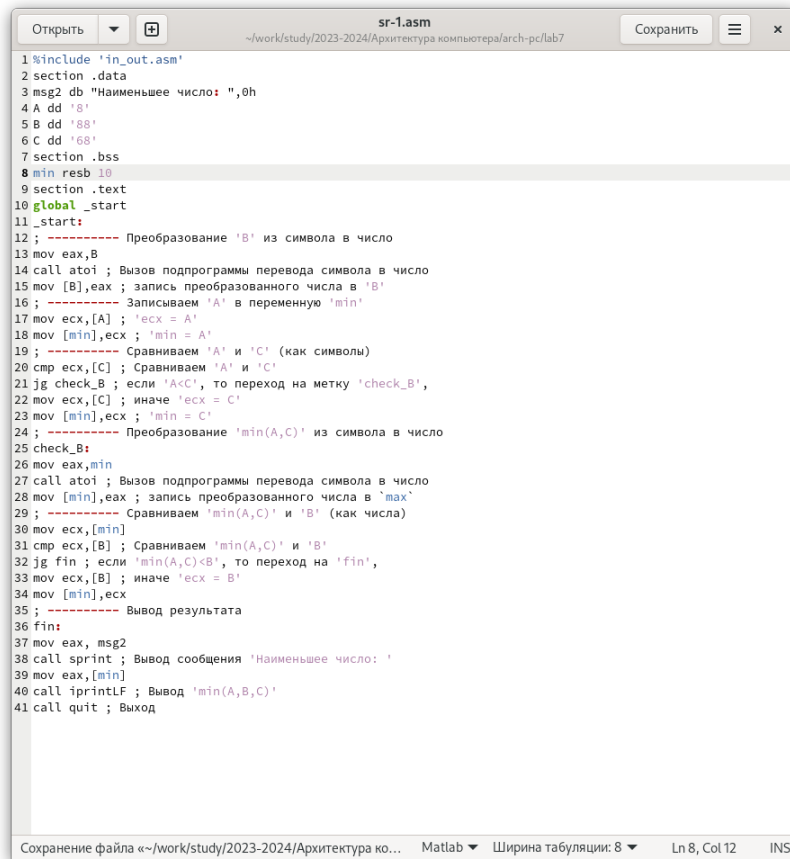
```
[emkurilkoryumin@fedora lab7]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[emkurilkoryumin@fedora lab7]$ mcedit lab7-2.lst

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

[emkurilkoryumin@fedora lab7]$
```

Рис. 4.14: Создание и редактирование файла

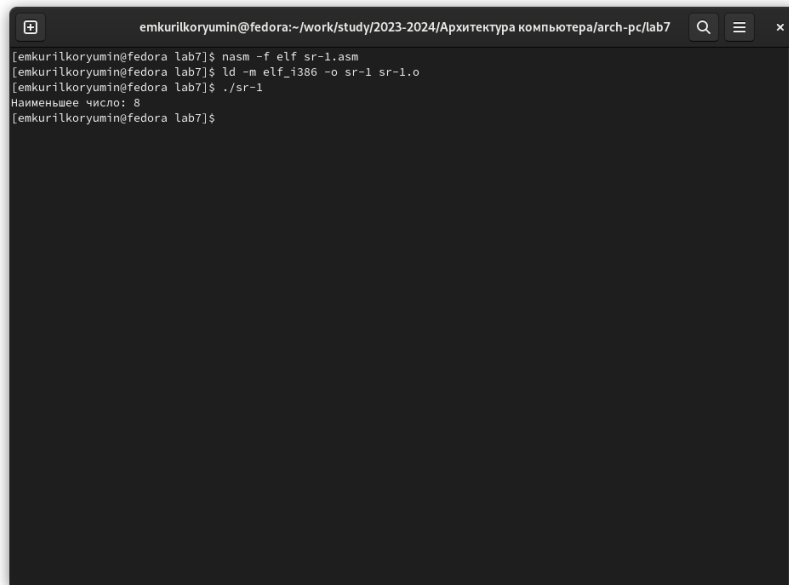
Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис.15)



```
1 %include 'in_out.asm'
2 section .data
3 msg2 db "Наименьшее число: ",0h
4 A dd '8'
5 B dd '88'
6 C dd '68'
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 jg 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 'max'
29 ; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
30 mov ecx,[min]
31 cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
32 jg 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 sprint ; Вывод сообщения 'Наименьшее число: '
39 mov eax,[min]
40 call iprintf ; Вывод 'min(A,B,C)'
41 call quit ; Выход
```

Рис. 4.15: Компиляция, обработка и запуск исполняемого файла

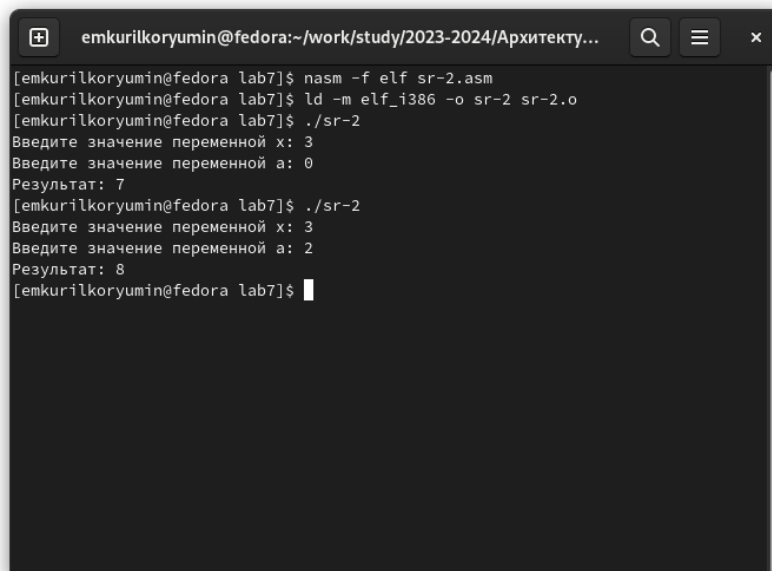
Создаю файл sr-2.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы для своего 4-го варианта: $f = 2x + 1$, если $a=0$ и $f = 2x+a$, если $a \neq 0$ (рис.16)



```
emkurilkoryumin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab7
[emkurilkoryumin@fedora lab7]$ nasm -f elf sr-1.asm
[emkurilkoryumin@fedora lab7]$ ld -m elf_i386 -o sr-1 sr-1.o
[emkurilkoryumin@fedora lab7]$ ./sr-1
Наименьшее число: 8
[emkurilkoryumin@fedora lab7]$
```

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.17)



```
emkurilkoryumin@fedora:~/work/study/2023-2024/Архитекту...
[emkurilkoryumin@fedora lab7]$ nasm -f elf sr-2.asm
[emkurilkoryumin@fedora lab7]$ ld -m elf_i386 -o sr-2 sr-2.o
[emkurilkoryumin@fedora lab7]$ ./sr-2
Введите значение переменной x: 3
Введите значение переменной a: 0
Результат: 7
[emkurilkoryumin@fedora lab7]$ ./sr-2
Введите значение переменной x: 3
Введите значение переменной a: 2
Результат: 8
[emkurilkoryumin@fedora lab7]$
```

Рис. 4.17: 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 '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 4.

%include 'in_out.asm' ; подключение внешнего файла

SECTION .data ; секция инициированных данных msg1: DB 'Введите значение
переменной x:',0 msg2: DB 'Введите значение переменной a:',0 rem: DB 'Результат:',0

SECTION .bss ; секция не инициированных данных

x: RESB 80 ; Переменная, чье значение будем вводить с клавиатуры, выделен-
ный размер - 80 байт a: RESB 80 ; Переменная, чье 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 подпро-
 граммы ввода сообщения mov eax,x; вызов подпрограммы преобразования call
 atoi ; ASCII кода в число, eax=x mov [x],eax mov eax, msg2 ; запись адреса выводи-
 мого 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 A != 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 резуль-
 тата 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