

РУДН. Архитектура компьютеров

Отчёт по лабораторной работе №7

Косинов Никита Андреевич, НПИМбв-02-20

Содержание

1	Цель работы	5
2	Ход работы	6
3	Безусловный переход	7
4	Условный переход	14
5	Файл листинга	18
6	Самостоятельная работа	21
7	Выводы	28

Список иллюстраций

3.1	Создание файла	7
3.2	Код программы 7-1	7
3.3	Исполняемый файл 7-1	9
3.4	Код программы 7-1-1	9
3.5	Исполняемый файл 7-1-1	11
3.6	Код программы 7-1-2	11
3.7	Исполняемый файл 7-1-2	13
4.1	Часть кода программы 7-2	14
4.2	Исполняемый файл 7-2	17
5.1	Создание файла листинга	18
5.2	Фрагмент файла листинга	18
5.3	Фрагмент файла листинга	19
5.4	Изменения кода 7-2	19
5.5	Ошибка!	20
6.1	Исполняемый файл sr-1	23
6.2	Код программы sr-2	24
6.3	Код программы sr-2	24
6.4	Исполняемый файл sr-2	27

Список таблиц

1 Цель работы

Кроме арифметических операций, нам необходимо выполнять логические. То есть, необходимо уметь сравнивать объекты: строки и числа, и, в зависимости от результата, выполнять то или иное действие. В языке Ассемблер предусмотрены такие переходы: без условия и с условием. Оба перехода происходят, например, с помощью меток, отличие в том, что переход в случае наличия условия происходит только при выполнении этого условия.

Цель данной работы - познакомиться с механикой условных и безусловных переходов языка Ассемблер.

2 Ход работы

Лабораторная работа выполнена с использованием консоли **ОС Linux** и языка программирования ассемблера **NASM**.

1. Безусловный переход;
2. Условный переход;
3. Файл листинга.

В конце выполнена самостоятельная работа.

3 Безусловный переход

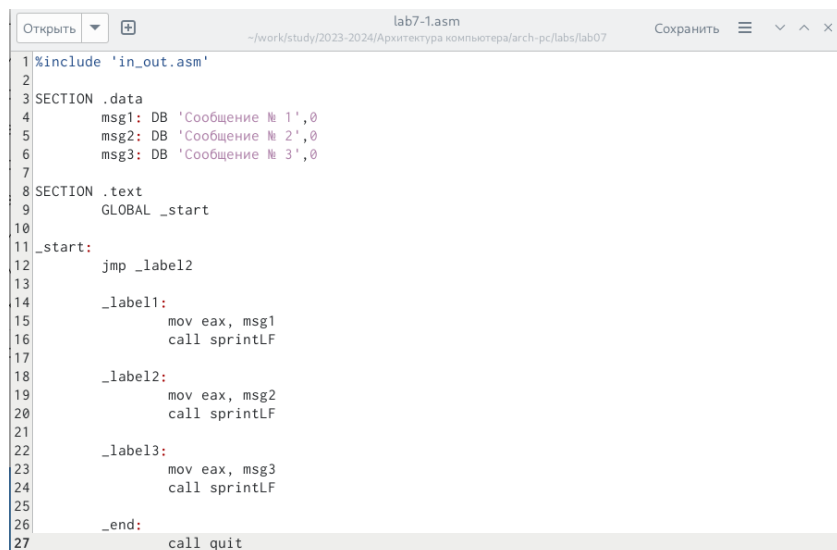
В начале работы познакомимся с тем, как в языке Ассемблер работает система меток.

1. Переходим в рабочий каталог и создадим файл формата *.asm.

```
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ touch lab7-1.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $
```

Рис. 3.1: Создание файла

2. Напишем программу, использующую метки. Переход в участок программы происходит посредством команды **jmp**, далее код продолжает работать сверху вниз.



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

Рис. 3.2: Код программы 7-1

```
%include 'in_out.asm'
```

```
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
```

```
call sprintfLF
```

```
_label2:
```

```
mov eax, msg2
```

```
call sprintfLF
```

```
_label3:
```

```
mov eax, msg3
```

```
call sprintfLF
```

```
_end:
```

```
call quit
```

3. Скомпилируем и запустим.

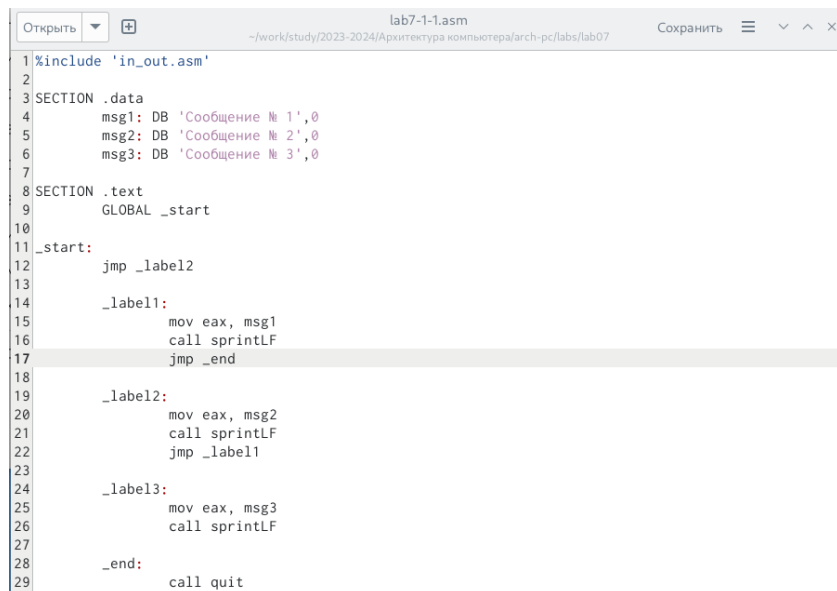

```

nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ cp ../lab06/in_out.asm
.
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf lab7-1.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o lab7
-lab7-1.o
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $

```

Рис. 3.3: Исполняемый файл 7-1

4. Мы видим, что из-за перехода на метку `**_label2**` в самом начале кода программа пропустила всё, написанное до этой метки, а именно вывод “Сообщение №1”.
5. Изменим код программы, добавив переход из в конце второго блока в первый, а также переход с конца первого блока в финальный.



```

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

```

Рис. 3.4: Код программы 7-1-1

```
%include 'in_out.asm'
```

```
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
```

```
    call sprintfLF
```

```
    jmp _end
```

```
_label2:
```

```
    mov eax, msg2
```

```
    call sprintfLF
```

```
    jmp _label1
```

```
_label3:
```

```
    mov eax, msg3
```

```
    call sprintfLF
```

```
_end:
```

```
    call quit
```

6. Скомпилируем и запустим.

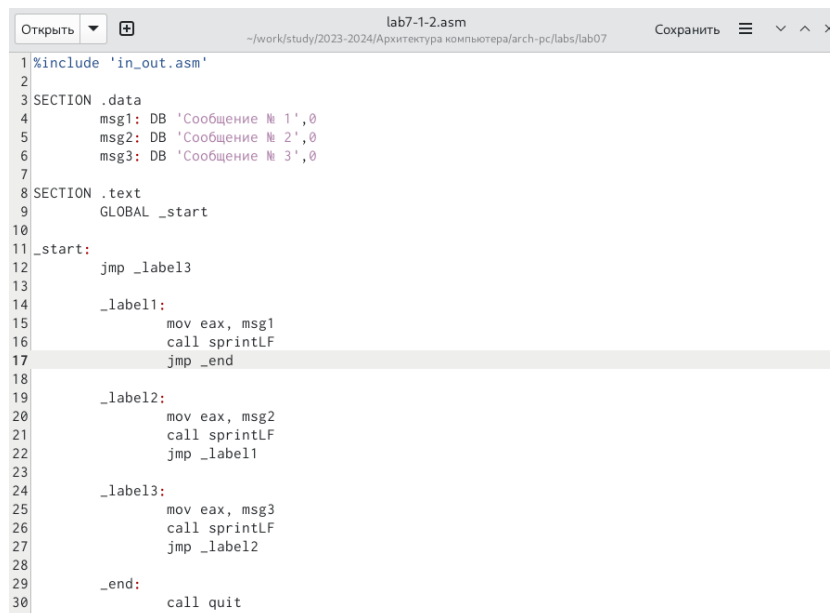
```

nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf lab7-1-1.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o lab7-1-1 lab7-1-1.o
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-1-1
Сообщение № 2
Сообщение № 1
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $

```

Рис. 3.5: Исполняемый файл 7-1-1

7. Как и ожидалось, сначала произошёл вывод 2-го сообщения, затем 1-го и выход из программы.
8. Изменим первый переход, поменяв 2-ю на 3-ю метку, а в третий блок программы добавим переход ко второй метке так, чтобы сообщения выводились в обратном порядке.



```

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

```

Рис. 3.6: Код программы 7-1-2

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg1: DB 'Сообщение № 1',0
```

```
msg2: DB 'Сообщение № 2',0
```

```
msg3: DB 'Сообщение № 3',0
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    jmp _label3
```

```
_label1:
```

```
    mov eax, msg1
```

```
    call sprintfLF
```

```
    jmp _end
```

```
_label2:
```

```
    mov eax, msg2
```

```
    call sprintfLF
```

```
    jmp _label1
```

```
_label3:
```

```
    mov eax, msg3
```

```
    call sprintfLF
```

```
    jmp _label2
```

```
_end:
```

```
    call quit
```

9. Проверим результат

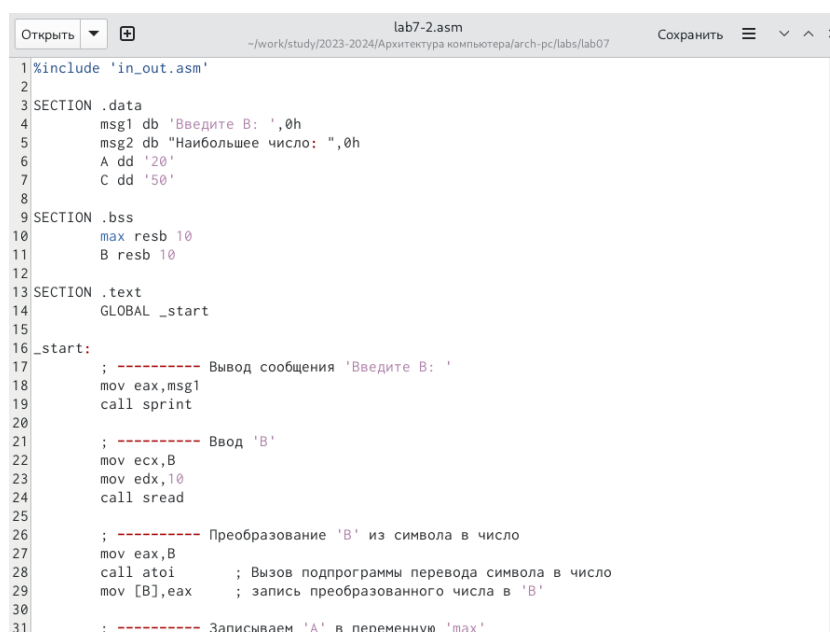
```
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf lab7-1-2.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o lab7-1-2 lab7-1-2.o
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-1-2
Сообщение № 3
Сообщение № 2
Сообщение № 1
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $
```

Рис. 3.7: Исполняемый файл 7-1-2

4 Условный переход

Условный переход - это переход между частями кода, если выполнено некоторое условие. Инструкций по передаче управления по метке много, но все начинаются с буквы **j**, так как работают по принципу **jmp**, только с условием.

1. Напишем программу, определяющую максимум из трёх элементов. Две переменные зададим в самом коде, а третью попросим ввести с клавиатуры пользователя.



```
1 %include 'in_out.asm'
2
3 SECTION .data
4     msg1 db 'Введите B: ',0h
5     msg2 db "Наибольшее число: ",0h
6     A dd '20'
7     C dd '50'
8
9 SECTION .bss
10    max resb 10
11    B resb 10
12
13 SECTION .text
14    GLOBAL _start
15
16 _start:
17    ; ----- Вывод сообщения 'Введите B: '
18    mov eax,msg1
19    call sprint
20
21    ; ----- Ввод 'B'
22    mov ecx,B
23    mov edx,10
24    call sread
25
26    ; ----- Преобразование 'B' из символа в число
27    mov eax,B
28    call atoi      ; Вызов подпрограммы перевода символа в число
29    mov [B],eax    ; запись преобразованного числа в 'B'
30
31    : ----- Записываем 'A' в переменную 'max'
```

Рис. 4.1: Часть кода программы 7-2

```
%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

```


2. Разберём код:

3.1. Первыми двумя блоками выведем приглашение ввести третье число и считаем его с клавиатуры.

3.2. Преобразуем его в число функцией **atoi** подключенного файла.

3.3. Запишем значение первой переменной в созданный буфер *max*.

3.4. Сравним переменные *A* и *C* инструкцией **cmp**, но как символы. Так можно сделать, т.к. числа в десятичной записи сравниваются в лексикографическом порядке, и коды цифры в кодировке **ASCII** идут по возрастанию самих цифр.

3.5. Результат сравнения переменных запишется в регистр флагов. Инструкция **jb** сработает в случае, если $A > C$ и позволяет сразу перейти к сравнению $max=A$ и *B*. В противном случае, поменяем значение *max* на *C*, и уже после перейдём к сравнению $max=C$ и *B*.

3.6. Находим максимум окончательно и выводим результат на экран.

3. Скомпилируем и запустим программу. Попробуем её в работе для разных значений третьей переменной. Убеждаемся, что программа работает корректно!

```
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf lab7-2.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-2
Введите B: 10
Наибольшее число: 50
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-2
Введите B: 30
Наибольшее число: 50
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-2
Введите B: 100
Наибольшее число: 100
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./lab7-2
Введите B: qwerty
Наибольшее число: 50
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $
```

Рис. 4.2: Исполняемый файл 7-2

5 Файл листинга

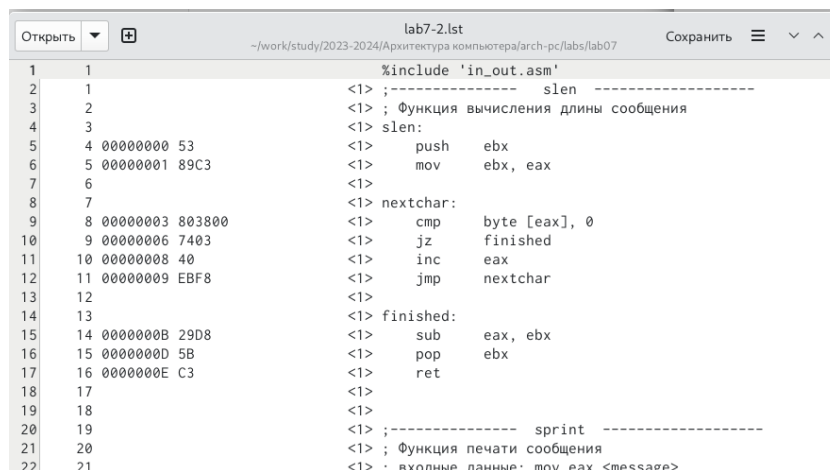
Файл листинга - документ, использующийся при отладке кода программы. Он содержит код на написанном языке, а также машинный код.

1. Создадим файл листинга предыдущей программы, указав ключ **-l** при создании объектного файла.

```
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ gedit lab7-2.lst
```

Рис. 5.1: Создание файла листинга

2. Рассмотрим структура созданного документа.



```
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>
```

Рис. 5.2: Фрагмент файла листинга

```

173      3
174      4 00000000 D092D0B2D0B5D0B4D0- SECTION .data
175      4 00000009 B8D182D0B520423A20- msg1 db 'Введите B: ',0h
176      4 00000012 00
177      5 00000013 D09DD0B0D0B8D0B1D0- msg2 db "Наибольшее число: ",0h
178      5 0000001C BED0BBD18CD188D0B5-
179      5 00000025 D0B52D0187D0B8D181-
180      5 0000002E D0BBD0BE3A2000
181      6 00000035 32300000 A dd '20'
182      7 00000039 35300000 C dd '50'
183      8
184      9 SECTION .bss
185     10 00000000 <res Ah> max resb 10
186     11 0000000A <res Ah> B resb 10
187     12
188     13 SECTION .text
189     14 GLOBAL _start
190     15
191     16 _start:
192     17 ; ----- Вывод сообщения 'Введите B: '
193     18 000000E8 B8[00000000] mov eax,msg1
194     19 000000ED E81DFFFFFF call sprint
195     20
196     21 ; ----- Ввод 'B'
197     22 000000F2 B9[0A000000] mov ecx,B

```

Рис. 5.3: Фрагмент файла листинга

3. Проследим за структурой листинга:

3.1. Номера строк исходной программы, причём в случае нескольких команд машинного кода по одной строке написанного нами, этот номер несколько раз дублируется.

3.2. Адрес, на сколько мы сместились от начала сегмента.

3.3. Машинный код каждой инструкции в шестнадцатиричной системе счисления.

3.4. Исходный код программы и написанные в нём комментарии.

4. Изменим код, убрав из команды **mov** второй операнд.

```

14 GLOBAL _start
15
16 _start:
17 ; ----- Вывод сообщения 'Введите B: '
18 mov eax
19 call sprint
20
21 ; ----- Ввод 'B'
22 mov ecx,B

```

Рис. 5.4: Изменения кода 7-2

5. Попробуем создать файл листинга. Но у нас это не выйдет, так как в программе допущена ошибка *неправильное использование инструкции и операндов*.

Соответственно, компиляция прерывается, и файл листинга не создаётся. Таким образом, последний нужен именно для отладки работающей программы, но не для поиска допущенных синтаксических ошибок.

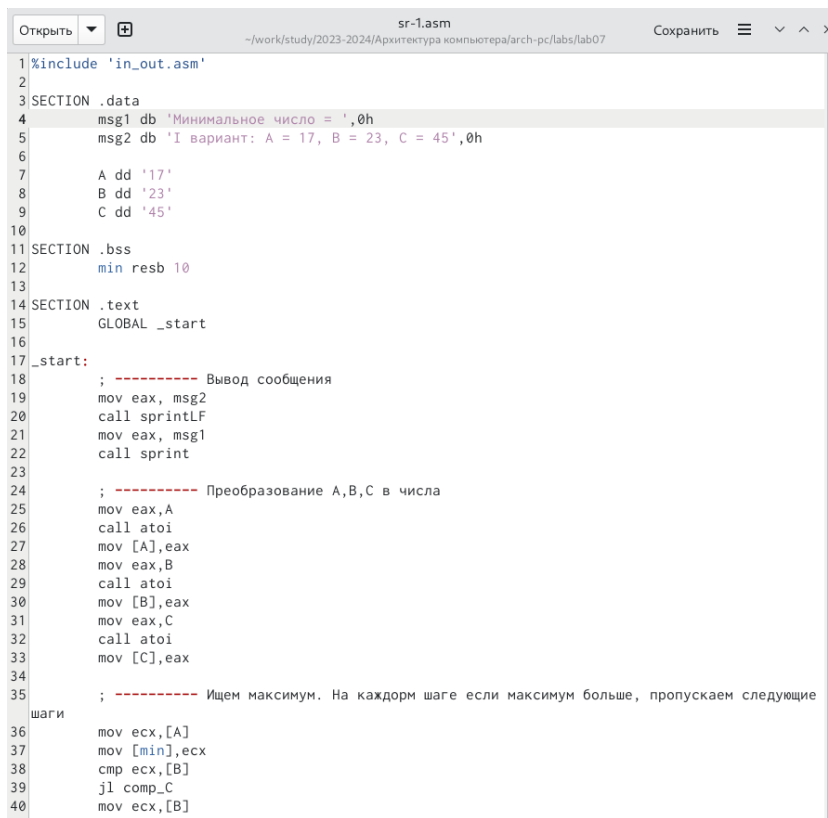
```
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf -l lab7-2.  
lst lab7-2.asm  
lab7-2.asm:18: error: invalid combination of opcode and operands  
nakosinov@dk8n52 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $
```

Рис. 5.5: Ошибка!

6 Самостоятельная работа

Текущая самостоятельная работа состоит из двух задач I варианта.

1. В первой задаче требуется найти минимум значений трёх чисел: $a=17$, $b=23$, $c=45$. Напишем программу, добавив комментарии.



```
1 %include 'in_out.asm'
2
3 SECTION .data
4     msg1 db 'Минимальное число = ',0h
5     msg2 db 'I вариант: A = 17, B = 23, C = 45',0h
6
7     A dd '17'
8     B dd '23'
9     C dd '45'
10
11 SECTION .bss
12     min resb 10
13
14 SECTION .text
15     GLOBAL _start
16
17 _start:
18     ; ----- Вывод сообщения
19     mov eax, msg2
20     call sprintf
21     mov eax, msg1
22     call sprintf
23
24     ; ----- Преобразование A,B,C в числа
25     mov eax,A
26     call atoi
27     mov [A],eax
28     mov eax,B
29     call atoi
30     mov [B],eax
31     mov eax,C
32     call atoi
33     mov [C],eax
34
35     ; ----- Ищем максимум. На каждом шаге если максимум больше, пропускаем следующие
36     ; шаги
37     mov ecx,[A]
38     mov [min],ecx
39     cmp ecx,[B]
40     jl comp_C
41     mov ecx,[B]
42     cmp ecx,[C]
43     jl fin
44     mov ecx,[C]
45     mov [min],ecx
46
47     ; ----- Вывод максимума
48     fin:
49     mov eax,[min]
50     call iprintLF
51     call quit
52
```

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg1 db 'Минимальное число = ',0h
```

```
msg2 db 'I вариант: A = 17, B = 23, C = 45',0h
```

```
A dd '17'
```

```
B dd '23'
```

```
C dd '45'
```

```
SECTION .bss
```

```
min resb 10
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
; ----- Вывод сообщения
```

```
mov eax, msg2
```

```
call sprintf
```

```
mov eax, msg1
```

```
call sprintf
```

```
; ----- Преобразование A,B,C в числа
```

```
mov eax,A
```

```
call atoi
```

```
mov [A],eax
```

```
mov eax,B
```

```
call atoi
```

```
mov [B],eax
```

```
mov eax,C
```

```
call atoi
```

```
mov [C],eax
```

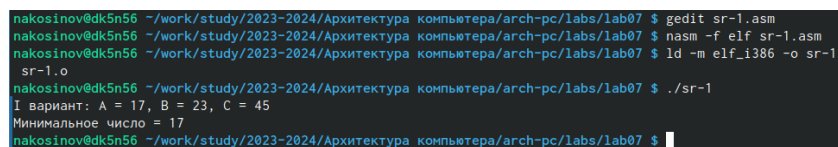
```

; ----- Ищем максимум. На каждом шаге если максимум больше, пропускаем
mov ecx,[A]
mov [min],ecx
cmp ecx,[B]
jl comp_C
mov ecx,[B]
mov [min],ecx
comp_C:
    cmp ecx,[C]
    jl fin
    mov ecx,[C]
    mov [min],ecx

; ----- Вывод максимума
fin:
    mov eax,[min]
    call iprintLF
    call quit

```

2. Скомпилируем и запустим.



```

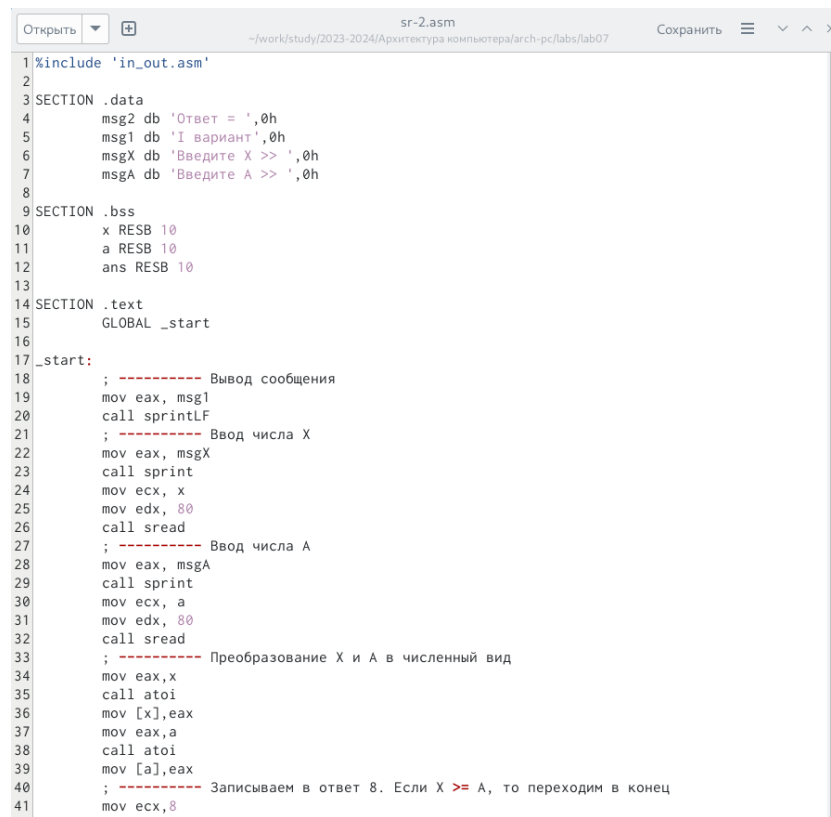
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ gedit sr-1.asm
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf sr-1.asm
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o sr-1 sr-1.o
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./sr-1
I вариант: A = 17, B = 23, C = 45
Минимальное число = 17
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $

```

Рис. 6.1: Исполняемый файл sr-1

3. Во второй задаче требуется вычислить значение функции с условием, в зависимости от введенных с клавиатуры чисел и . Результатом должно

быть число, равное , если и число, если . Напишем код программы, добавив комментарии.

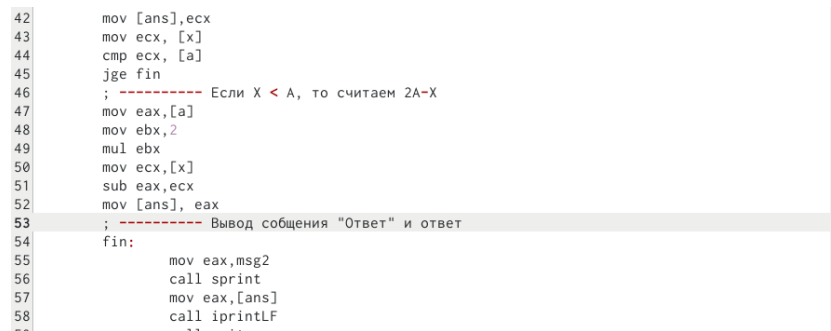


```

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg2 db 'Ответ = ',0h
5     msg1 db 'I вариант',0h
6     msgX db 'Введите X >> ',0h
7     msgA db 'Введите A >> ',0h
8
9 SECTION .bss
10    x RESB 10
11    a RESB 10
12    ans RESB 10
13
14 SECTION .text
15     GLOBAL _start
16
17 _start:
18     ; ----- Вывод сообщения
19     mov eax, msg1
20     call sprintf
21     ; ----- Ввод числа X
22     mov eax, msgX
23     call sprintf
24     mov ecx, x
25     mov edx, 80
26     call sread
27     ; ----- Ввод числа A
28     mov eax, msgA
29     call sprintf
30     mov ecx, a
31     mov edx, 80
32     call sread
33     ; ----- Преобразование X и A в численный вид
34     mov eax, x
35     call atoi
36     mov [x], eax
37     mov eax, a
38     call atoi
39     mov [a], eax
40     ; ----- Записываем в ответ 8. Если X >= A, то переходим в конец
41     mov ecx, 8

```

Рис. 6.2: Код программы sr-2



```

42     mov [ans], ecx
43     mov ecx, [x]
44     cmp ecx, [a]
45     jge fin
46     ; ----- Если X < A, то считаем 2A-X
47     mov eax, [a]
48     mov ebx, 2
49     mul ebx
50     mov ecx, [x]
51     sub eax, ecx
52     mov [ans], eax
53     ; ----- Вывод сообщения "Ответ" и ответ
54     fin:
55         mov eax, msg2
56         call sprintf
57         mov eax, [ans]
58         call iprintLF

```

Рис. 6.3: Код программы sr-2

%include 'in_out.asm'

SECTION .data

```
msg2 db 'Ответ = ',0h
msg1 db 'I вариант',0h
msgX db 'Введите X >> ',0h
msgA db 'Введите A >> ',0h
```

SECTION .bss

```
x RESB 10
a RESB 10
ans RESB 10
```

SECTION .text

GLOBAL _start

_start:

```
; ----- Вывод сообщения
mov eax, msg1
call printf
; ----- Ввод числа X
mov eax, msgX
call printf
mov ecx, x
mov edx, 80
call read
; ----- Ввод числа A
mov eax, msgA
call printf
mov ecx, a
```

```

mov edx, 80
call sread
; ----- Преобразование X и A в численный вид
mov eax,x
call atoi
mov [x],eax
mov eax,a
call atoi
mov [a],eax
; ----- Записываем в ответ 8. Если X >= A, то переходим в конец
mov ecx,8
mov [ans],ecx
mov ecx, [x]
cmp ecx, [a]
jge fin
; ----- Если X < A, то считаем 2A-X
mov eax,[a]
mov ebx,2
mul ebx
mov ecx,[x]
sub eax,ecx
mov [ans], eax
; ----- Вывод сообщения "Ответ" и ответ
fin:
    mov eax,msg2
    call sprint
    mov eax,[ans]
    call iprintLF
    call quit

```

4. Скомпилируем и запустим. Проверим работу программы на значениях из варианта и пары собственных

```
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ gedit sr-2.asm
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ nasm -f elf sr-2.asm
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ld -m elf_i386 -o sr-2
sr-2.o
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./sr-2
I вариант
Введите X >> 1
Введите A >> 2
Ответ = 3
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./sr-2
I вариант
Введите X >> 2
Введите A >> 1
Ответ = 8
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./sr-2
I вариант
Введите X >> 4
Введите A >> 4
Ответ = 8
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $ ./sr-2
I вариант
Введите X >> 3
Введите A >> 4
Ответ = 5
nakosinov@dk5n56 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab07 $
```

Рис. 6.4: Исполняемый файл sr-2

7 Выводы

В ходе данной лабораторной работы мы научились пользоваться переходами между частями кода с условием и без, а также познакомились со структурой отладочного документа - файла листинга.