

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 8

дисциплина: Архитектура компьютера

Студент: Волгин А.А.

Группа: НПИбд-01-22

МОСКВА

2022 г.

Цель работы:

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

Порядок выполнения лабораторной работы:

Реализация переходов в NASM.

Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рис. 1).

```
[aavolgin@fedora ~]$ mkdir ~/work/arch-pc/lab08
[aavolgin@fedora ~]$ cd ~/work/arch-pc/lab08
[aavolgin@fedora lab08]$ touch lab8-1.asm
[aavolgin@fedora lab08]$
```

рис. 1. Создание каталога и файла lab8-1.asm

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введем в файл lab8-1.asm следующий текст программы (рис. 2).

```
lab8-1.asm [-M--] 9
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 sprintf

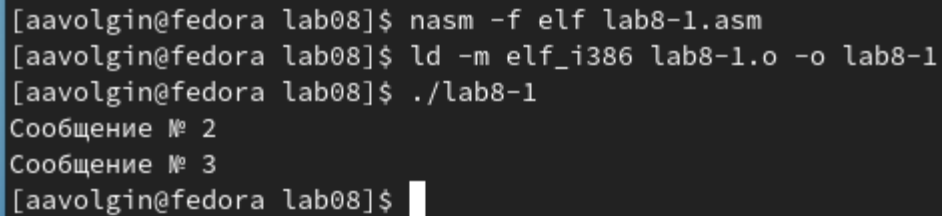
_label2:
mov eax, msg2
call sprintf

_label3:
mov eax, msg3
call sprintf

_end
call quit
```

рис. 2. Текст программы lab8-1

Создадим исполняемый файл и запустим его (рис. 3).



```
[aavolgin@fedora lab08]$ nasm -f elf lab8-1.asm
[aavolgin@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[aavolgin@fedora lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 3
[aavolgin@fedora lab08]$
```

рис. 3. Результат работы программы lab8-1

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`) (рис. 4).

```

lab8-1.asm [----] 11
#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 sprintLF
jmp _end

_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit

```

рис. 4. Измененный текст программы lab8-1

Создадим исполняемый файл и запустим его (рис. 5).

```

[aavolgin@fedora lab08]$ nasm -f elf lab8-1.asm
[aavolgin@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[aavolgin@fedora lab08]$ ./lab8-1
Сообщение № 2
Сообщение № 1
[aavolgin@fedora lab08]$

```

рис. 5. Результат работы измененной программы lab8-1

Далее изменим текст программы lab8-1 так, чтобы сообщения выводились в обратном порядке, затем запустим программу (рис. 6-7).

```

lab8-1.asm [----] 11 L
#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 printf
jmp _end

_label2:
mov eax, msg2
call printf
jmp _label1

_label3:
mov eax, msg3
call printf
jmp _label2

_end:
call quit

```

рис. 6. Измененный текст программы lab8-1

```

[aavolgin@fedora lab08]$ nasm -f elf lab8-1.asm
[aavolgin@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[aavolgin@fedora lab08]$ ./lab8-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
[aavolgin@fedora lab08]$

```

рис. 7. Результат работы измененной программы lab8-1

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: A, B и C. Значения для A и C задаются в программе, значение B вводится с клавиатуры.

Создадим файл `lab8-2.asm` в каталоге `~/work/arch-pc/lab08` и введем в него следующий текст программы (рис. 8-9).

```
lab8-2.asm [----] 0 L:[ 1+ ]
%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:

mov eax, msg1
call sprint

mov ecx, B
mov edx, 10
call sread

mov eax, B
call atoi
mov [B],eax
```

рис. 8. Текст программы lab8-2 (1)

```
mov ecx,[A]
mov [max],ecx

cmp ecx,[C]
jg check_B
mov ecx,[C]
mov [max],ecx

check_B:
mov eax,max
call atoi
mov [max],eax

mov ecx,[max]
cmp ecx,[B]
jg fin
mov ecx,[B]
mov [max],ecx

fin:
mov eax, msg2
call sprint
mov eax,[max]
call iprintLF
call quit
```

рис. 9. Текст программы lab8-2 (2)

Создадим файл и проверим его работу для разных значений В (рис. 10).

```
[aavolgin@fedora lab08]$ nasm -f elf lab8-2.asm
[aavolgin@fedora lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2
[aavolgin@fedora lab08]$ ./lab8-2
Введите В: 10
Наибольшее число: 50
[aavolgin@fedora lab08]$ ./lab8-2
Введите В: 30
Наибольшее число: 50
[aavolgin@fedora lab08]$ ./lab8-2
Введите В: 51
Наибольшее число: 51
[aavolgin@fedora lab08]$
```

рис. 10. Работа программы lab8-2

Как видим, все работает корректно.

Обратим внимание, что 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 сравнить все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов в числа, над ними можно корректно проводить арифметические операции.

Изучение структуры файлы листинга.

Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла `lab8-2.asm` (рис. 11).

```
[aavolgin@fedora lab08]$ nasm -f elf -l lab8-2.lst lab8-2.asm
[aavolgin@fedora lab08]$
```

рис. 11. Создание файла листинга для программы lab8-2

Затем откроем этот файл (рис. 12-13).


```

/home/aavolgin/work/arch-pc/lab08/lab8-2.lst      2600/13433      19%
1          %include 'in_out.asm'
2          <1> ;----- slen -----
-----
3          <1> ; Функция вычисления длины сообщения
4          <1> slen:
5 00000000 53          <1>     push     ebx
6 00000001 89C3        <1>     mov      ebx, eax
7          <1>
8          <1> nextchar:
9 00000003 803800      <1>     cmp      byte [eax], 0
10 00000006 7403       <1>     jz       finished
11 00000008 40         <1>     inc      eax
12 00000009 EBF8       <1>     jmp      nextchar
13          <1>
14          <1> finished:
15 0000000B 29D8       <1>     sub      eax, ebx
16 0000000D 5B         <1>     pop      ebx
17 0000000E C3         <1>     ret
18          <1>
19          <1>
20          <1> ;----- sprint -----
-----
21          <1> ; Функция печати сообщения
22          <1> ; входные данные: mov eax,<message>
23          <1> sprint:
24 0000000F 52         <1>     push     edx
25 00000010 51         <1>     push     ecx
26 00000011 53         <1>     push     ebx
27 00000012 50         <1>     push     eax
28 00000013 E8E8FFFFFF <1>     call     slen
29          <1>
30 00000018 89C2       <1>     mov      edx, eax
31 0000001A 58         <1>     pop      eax
32          <1>
33 0000001B 89C1       <1>     mov      ecx, eax
34 0000001D BB01000000  <1>     mov      ebx, 1
35 00000022 B804000000  <1>     mov      eax, 4
36 00000027 CD80       <1>     int      80h
37          <1>
38 00000029 5B         <1>     pop      ebx
39 0000002A 59         <1>     pop      ecx
40 0000002B 5A         <1>     pop      edx
41 0000002C C3         <1>     ret
42          <1>
43          <1>

```

рис. 12. Файл листинга программы lab8-2 (1)

```

2
3                                SECTION .data
4 00000000 D092D0B2D0B5D0B4D0-   msg1 db 'Введите B: ',0h
4 00000009 B8D182D0B520423A20-
4 00000012 00
5 00000013 D09DD0B0D0B8D0B1D0-   msg2 db "Наибольшее число: ",0h
5 0000001C BED0BBBD18CD188D0B5-
5 00000025 D0B520D187D0B8D181-
5 0000002E D0BBBD0BE3A2000
6 00000035 32300000               A dd '20'
7 00000039 35300000               C dd '50'
8
9                                SECTION .bss
10 00000000 <res Ah>               max resb 10
11 0000000A <res Ah>               B resb 10
12
13                                SECTION .text
14
15                                GLOBAL _start
16                                _start:
17
18 000000E8 B8[00000000]           mov eax, msg1
19 000000ED E81DFFFFFF             call sprint
20
21 000000F2 B9[0A000000]           mov ecx, B
22 000000F7 BA0A000000             mov edx, 10
23 000000FC E842FFFFFF             call sread
24
25 00000101 B8[0A000000]           mov eax, B
26 00000106 E891FFFFFF             call atoi
27 0000010B A3[0A000000]           mov [B],eax
28
29 00000110 8B0D[35000000]         mov ecx,[A]
30 00000116 890D[00000000]         mov [max],ecx
31
32 0000011C 3B0D[39000000]         cmp ecx,[C]
33 00000122 7F0C                  jg check_B
34 00000124 8B0D[39000000]         mov ecx,[C]
35 0000012A 890D[00000000]         mov [max],ecx
36
37                                check_B:
38 00000130 B8[00000000]           mov eax,max
39 00000135 E862FFFFFF             call atoi

```

1Помощь 2Разобрать 3Выход 4Нех 5Перейти 6 7Поиск 8Исходный 9Формат 10Выход

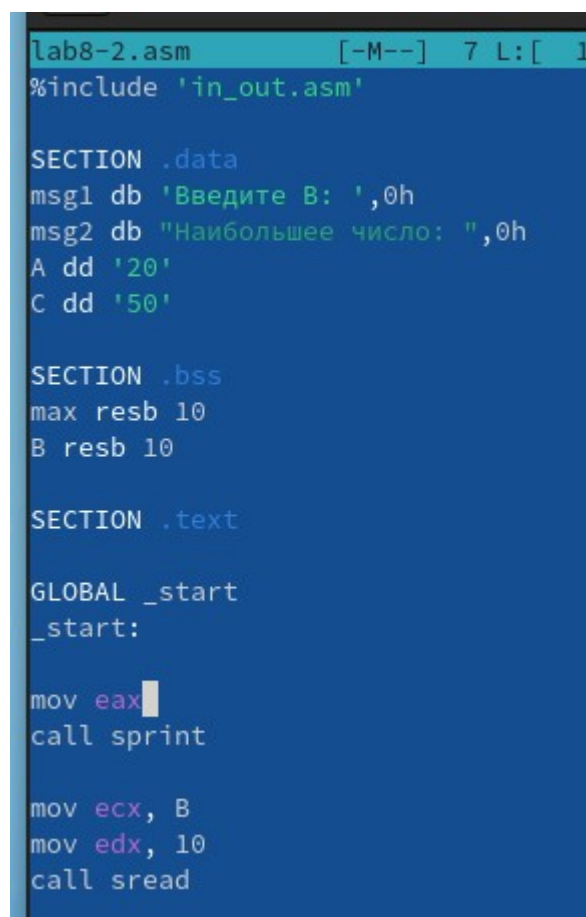
рис. 13. Файл листинга программы lab8-2 (2)

Как видим на рис. 12 показаны некоторые функции, прописанные в файле in_out.asm, который мы подключаем, на рис. 13 отображена непосредственно часть текста программы lab8-2, разберем несколько строк из этого текста:

Строка 10: после обозначения строки видим 00000000 это адрес, т.е. смещение машинного кода от начала текущего сегмента, поскольку строка 10 является самым начало сегмента SECTION .bss, ее адрес будет 00000000, затем идет машинный код: <res Ah> показывает, что было зарезервировано A байт (то есть 10 байт) памяти для переменной max, которая уже отображена в самой правой строке: max resb 10 – это код программы, здесь мы выделяем память из 10 однобайтовых ячеек по адресу с меткой max.

Строка 33: ее адрес уже равняется 00000122, 7FOC – ассемблированная инструкция jg, которая используется в этой строке для условной передачи управления по результатам арифметического сравнения в 32 строке ecx и [C].

Откроем файл с программой lab8-2.asm и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга (14-15).



```
lab8-2.asm      [-M--]  7 L:[ 1
%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:

mov eax, msg1
call sprint

mov ecx, B
mov edx, 10
call sread
```

рис. 14. Удаление операнда msg1 в строке move ax, msg1

```
/home/aavolgin/work/arch-pc/lab08/lab8-2.lst 13518/13518 100%
11 0000000A <res Ah> B resb 10
12
13 SECTION .text
14
15 GLOBAL _start
16 _start:
17
18 mov eax
18 ***** error: invalid combination of opcode a
nd operands
19 000000E8 E822FFFFFF call sprint
20
21 000000ED B9[0A000000] mov ecx, B
22 000000F2 BA0A000000 mov edx, 10
23 000000F7 E847FFFFFF call sread
24
25 000000FC B8[0A000000] mov eax, B
26 00000101 E896FFFFFF call atoi
27 00000106 A3[0A000000] mov [B],eax
28
29 0000010B 8B0D[35000000] mov ecx,[A]
30 00000111 890D[00000000] mov [max],ecx
31
32 00000117 3B0D[39000000] cmp ecx,[C]
33 0000011D 7F0C jg check_B
34 0000011F 8B0D[39000000] mov ecx,[C]
35 00000125 890D[00000000] mov [max],ecx
36
37 check_B:
38 0000012B B8[00000000] mov eax,max
39 00000130 E867FFFFFF call atoi
40 00000135 A3[00000000] mov [max],eax
41
42 0000013A 8B0D[00000000] mov ecx,[max]
43 00000140 3B0D[0A000000] cmp ecx,[B]
44 00000146 7F0C jg fin
45 00000148 8B0D[0A000000] mov ecx,[B]
46 0000014E 890D[00000000] mov [max],ecx
47
48 fin:
49 00000154 B8[13000000] mov eax, msg2
50 00000159 E8B1FEFFFF call sprint
51 0000015E A1[00000000] mov eax,[max]
52 00000163 E81EFFFFFF call iprintLF
53 00000168 E86EFFFFFF call quit
1По~щъ 2Раз~рн 3Выход 4Нех 5Пер~ти 6 7Поиск 8Исх~ый 9Формат10Выход
```

рис. 15. Листинг программы с удаленным операндом

Как видим, ассемблер немного ругается: в листинге отображается, что указана неверная комбинация операндов как раз в той строке, в которой мы убрали один операнд.

Порядок выполнения самостоятельной работы:

Напишем программу (lab8-3) нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения для моего варианта (15) будут следующими: a = 32, b = 6, c = 54. Создадим исполняемый файл и проверим его работу (рис. 16-17).

```

lab8-3.asm      [----]  7  L: [
#include 'in_out.asm'

SECTION .data
msg db "Наименьшее число: ",0h
A dd '32'
B dd '6'
C dd '54'

SECTION .bss
min resb 10

SECTION .text

GLOBAL _start
_start:

mov ecx,[A]
mov [min],ecx

cmp ecx,[B]
jl check_C
mov ecx,[B]
mov [min],ecx

check_C:
mov eax,min
call atoi
mov [min],eax

mov ecx,[min]
cmp ecx,[C]
jl fin
mov ecx,[C]
mov [min],ecx

fin:
mov eax,msg
call sprint
mov eax,[min]
call iprintLF
call quit

```

рис. 16. Текст программы lab8-3

В данном случае я сначала сравниваю A и B, если $A < B$, идем сразу на метку check_C, если нет, то присваиваем регистру ecx значение B, тот же процесс происходит, когда сравниваем ecx и C, только теперь программа переходит на метку fin.

```
[aavolgin@fedora lab08]$ nasm -f elf lab8-3.asm
[aavolgin@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[aavolgin@fedora lab08]$ ./lab8-3
Наименьшее число: 6
[aavolgin@fedora lab08]$
```

рис. 17. Результат работы программы

Как видим, все работает корректно.

Вывод:

Во время выполнения лабораторной работы были изучены команды условного и безусловного переходов, приобретены навыки написания программ с использованием переходов, изучено назначение и структура файла листинга.