

Лабораторная работа №7

**Команды безусловного и условного переходов в Nasm.
Программирование ветвлений.**

Перфилов Александр Константинович | группа: НПИбд 02-23

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Самостоятельная работа	15
4	Выводы	18

Список иллюстраций

2.1	Рис 2.1.1: Создание каталога и файла .asm	6
2.2	Рис 2.1.2: Программа	7
2.3	Рис 2.1.3: Создание исполняемого файла и проверка работы . . .	7
2.4	Рис 2.1.4: Измененная программа	8
2.5	Рис 2.1.5: Создание исполняемого файла и проверка работы . . .	8
2.6	Рис 2.1.6: Демонстрация измененного текста программы	9
2.7	Рис 2.1.7: Проверка работы программы	9
2.8	Рис 2.1.8: Создание файла .asm	10
2.9	Рис 2.1.9: Программа	11
2.10	Рис 2.1.10: Создание исполняемого файла и проверка работы с разными значениями В	12
2.11	Рис 2.2.1: Создание файла	12
2.12	Рис 2.2.2: Вид файла .lst	13
2.13	Рис 2.2.3: Взятые строки	13
2.14	Рис 2.2.4: Удаление операнды	14
2.15	Рис 2.2.5: Трансляция файла	14
2.16	Рис 2.2.6: Ошибка в файле .lst	14
3.1	Рис 3.1.1: Создание файла	15
3.2	Рис 3.1.2: Программа	16
3.3	Рис 3.1.3: Проверка программы	17
3.4	Рис 3.1.4: Загрузка файлов на github	17

Список таблиц

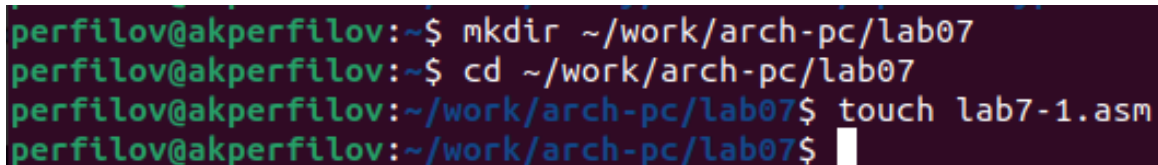
1 Цель работы

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

2 Выполнение лабораторной работы

Реализация переходов в NASM

Создадим каталог для программ лабораторной работы № 7, перейдите в него и создадим файл lab7-1.asm



```
perfilov@akperfilov:~$ mkdir ~/work/arch-pc/lab07
perfilov@akperfilov:~$ cd ~/work/arch-pc/lab07
perfilov@akperfilov:~/work/arch-pc/lab07$ touch lab7-1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .asm

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введем в файл lab7-1.asm текст программы из листинга 7.1

```

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 sprintfLF ; 'Сообщение № 1'
13 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintfLF ; 'Сообщение № 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintfLF ; 'Сообщение № 3'
19 _end:
20 call quit ; вызов подпрограммы завершения

```

Рис. 2.2: Рис 2.1.2: Программа

Создадим исполняемый файл и запустим его. Результат работы данной программы будет следующим:

```

perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
perfilov@akperfilov:~/work/arch-pc/lab07$

```

Рис. 2.3: Рис 2.1.3: Создание исполняемого файла и проверка работы

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`). Измените текст программы в соответствии с листингом 7.2

```
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 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. 2.4: Рис 2.1.4: Измененная программа

Создадим исполняемый файл и проверим его работу.

```
perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 2.5: Рис 2.1.5: Создание исполняемого файла и проверка работы

Изменим текст программы добавив или изменив инструкции `jmp`.

```
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. 2.6: Рис 2.1.6: Демонстрация измененного текста программы

Проверим работу программы

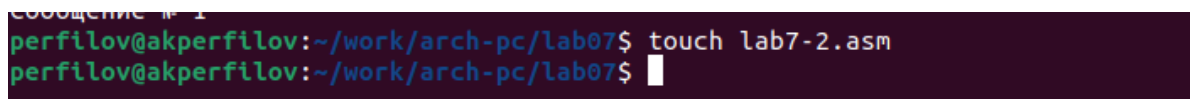
```
perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 2.7: Рис 2.1.7: Проверка работы программы

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: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры

Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07.



```
сообщение № 1
perfilov@akperfilov:~/work/arch-pc/lab07$ touch lab7-2.asm
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 2.8: Рис 2.1.8: Создание файла .asm

Внимательно изучим текст программы из листинга 7.3 и введем в lab7-2.asm.

```

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
46 call sprint ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call incintf ; Вывод 'max(A B C)'

```

Рис. 2.9: Рис 2.1.9: Программа

Создадим исполняемый файл и проверим его работу для разных значений В.

```
perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
perfilov@akperfilov:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 9
Наибольшее число: 50
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 2
Наибольшее число: 50
perfilov@akperfilov:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 51
Наибольшее число: 51
perfilov@akperfilov:~/work/arch-pc/lab07$ █
```

Рис. 2.10: Рис 2.1.10: Создание исполняемого файла и проверка работы с разными значениями В

Обратим внимание, 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 сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов 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 `lab7-2.asm`

```
perfilov@akperfilov:~/work/arch-pc/lab07$ touch lab7-2.asm
```

Рис. 2.11: Рис 2.2.1: Создание файла

Откроем файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`

```

/home/perfilov/work/arch-pc/lab07/lab7-2.lst [----] 0 L: [ 1+ 0 1/225] *(0 /14458b) 0032 0x020
1      %include 'in_out.asm'
2      ;----- 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 E8E8FFFF <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> .....

```

Рис. 2.12: Рис 2.2.2: Вид файла .lst

Внимательно ознакомимся с его форматом и содержанием.

Возьмем 3, 4 и 5 строки кода

```

2      ;----- slen -----
3      <1> ; Функция вычисления длины сообщения
4      <1> slen:.....
5      00000000 53      <1> push     ebx.....
6      00000001 89C3    <1> mov      ebx, eax.....

```

Рис. 2.13: Рис 2.2.3: Взятые строки

3 - номер строки кода, “; Функция вычисления длины сообщения” - оно не имеет отношения к работе кода. 4 - номер строки кода, “slen:.....” - название функции. 5 - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Откроем файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга:

```
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:
..
```

Рис. 2.14: Рис 2.2.4: Удаление операнды

Выполним трансляцию с измененной программой

```
perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 2.15: Рис 2.2.5: Трансляция файла

```
28      cmp ecx, ; Сравниваем 'A' и 'C'
28      *****
29 0000011C 7F0C      jg check_B ; если 'A>C', то переход на метку 'check_B',
```

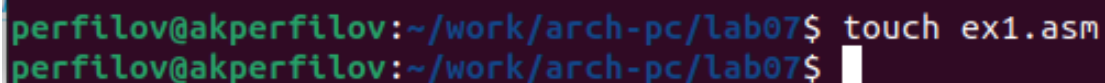
Рис. 2.16: Рис 2.2.6: Ошибка в файле .lst

На выходе мы не получаем файла из-за ошибки.

3 Самостоятельная работа

Задание№1 Напишите программу нахождения наименьшей из 3 целочисленных переменных a,b и c. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.

Создадим новый файл ex1.asm и напишем программу нахождения наименьшей из 3 целочисленных переменных a,b и c для варианта 19 (46, 32, 74).



```
perfilov@akperfilov:~/work/arch-pc/lab07$ touch ex1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 3.1: Рис 3.1.1: Создание файла

Возьмем за основу код lab7-2.asm и переделаем его под 1-ое задание

```

1 %include 'in_out.asm'
2 section .data
3 msg db "Наименьшее число: ",0h
4 A dd '46'
5 B dd '32'
6 C dd '74'
7 section .bss
8 min resb 10
9
10 section .text
11 global _start
12 _start:
13
14 mov eax,B
15 call atoi
16 mov [B],eax
17
18 mov ecx,[A]
19 mov [min],ecx
20
21 cmp ecx,[C]
22 jl check_B
23 mov ecx,[C]
24 mov [min],ecx
25
26 check_B:
27 mov eax,min
28 call atoi
29 mov [min],eax
30
31 mov ecx,[min]
32 cmp ecx,[B]
33 jl fin
34 mov ecx,[B]
35 mov [min],ecx
36
37 fin:
38 mov eax, msg
39 call sprint
40 mov eax,[min]
41 call iprintLF
42 call quit

```

Рис. 3.2: Рис 3.1.2: Программа

Проверим программу

```
perfilov@akperfilov:~/work/arch-pc/lab07$ nasm -f elf ex1.asm
perfilov@akperfilov:~/work/arch-pc/lab07$ ld -m elf_i386 -o ex1 ex1.o
perfilov@akperfilov:~/work/arch-pc/lab07$ ./ex1
Наименьшее число: 32
perfilov@akperfilov:~/work/arch-pc/lab07$
```

Рис. 3.3: Рис 3.1.3: Проверка программы

Загрузим все файлы на github

```
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git add .
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git commit -am 'feat(main): add files lab-7'
[master c707075] feat(main): add files lab-7
24 files changed, 180 insertions(+), 119 deletions(-)
create mode 100644 labs/lab06/report/image.zip
delete mode 100644 labs/lab07/report/image/placeimg_800_600_tech.jpg
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.10.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.4.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.5.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.6.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.7.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.8.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.1.9.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.3.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.4.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.5.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 2.2.6.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.1.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.2.png"
create mode 100644 "labs/lab07/report/image/\320\240\320\270\321\201 3.1.3.png"
create mode 100644 labs/lab07/report/report.docx
rewrite labs/lab07/report/report.md (71%)
create mode 100644 labs/lab07/report/report.pdf
perfilov@akperfilov:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git push
Перечисление объектов: 39, готово.
Подсчет объектов: 100% (39/39), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (31/31), готово.
Запись объектов: 100% (31/31), 2.75 МиБ | 2.25 МиБ/с, готово.
Всего 31 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:AlexanderPERfilovKonstantinivich/study_2023-2024_arh-pc.git
 0d1322c..c707075 master -> master
```

Рис. 3.4: Рис 3.1.4: Загрузка файлов на github

4 Выводы

Я изучил команды условного и безусловного переходов. Приобрел навыки написания программ с использованием переходов. Познакомился с назначением и структурой файла листинга.