

Отчет по лабораторной работе №7

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

Корчагин Алексей Павлович

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Задание Самостоятельной работы	16
5	Выводы	21
	Список литературы	22

Список иллюстраций

3.1	Создание каталога и файла	8
3.2	Редактирование файла	8
3.3	Создание и запуск исполняемого файла	9
3.4	Редактирование файла	9
3.5	Создание исполняемого файла	9
3.6	Редактирование файла	10
3.7	Создание исполняемого файла	10
3.8	Создание и редактирование файла	11
3.9	Создание исполняемого файла	12
3.10	Создание файла	12
3.11	Листинг	13
3.12	Редактирование файла	14
3.13	ошибка	14
3.14	ошибка в листинге	15
4.1	Программа поиска минимума	17
4.2	Создание и тест исполняющего файла	18
4.3	Тест программы	20
4.4	Тест программы	20

Список таблиц

1 Цель работы

Изучить команды условных и безусловных переходов. Преобрести навыки написания программ с использованием переходов. Познакомится с назначением и структурой файла

2 Теоретическое введение

В Ассемблере безусловный переход выполняется с помощью инструкции `jmp`, она включает в себя адрес перехода, то-есть куда следует передать управление:

`jmp`

`jmp label` - переход на метку `label` `jmp (label в квадратных скобках)` - переход по адресу в памяти, помеченному меткой `label` `jmp eax` - переход по адресу из регистра `eax`

Как показано 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 (такие как CF PF AF ZF SF)

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: `cmp`, Команда `cmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не созда-

ётся. Структура листинга:

номер строки

адрес — это смещение машинного кода от начала текущего сегмента

машинный код представляет собой ассемблированную исходную строку в виде шестнадцатер

исходный текст программы

3 Выполнение лабораторной работы

Создал каталог для программ необходимых для выполнения лабораторной работы №7, также создал файл lab7-1.asm(рис. 3.1).

```
apkorchagin@dk5n52 ~ $ mkdir ~/work/arch-pc/lab07
apkorchagin@dk5n52 ~ $ cd ~/work/arch-pc/lab07
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ touch lab7-1.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $
```

Рис. 3.1: Создание каталога и файла

Ввёл текст программы в файл lab7-1.asm(рис. 3.2).

```
1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение No 1',0
4 msg2: DB 'Сообщение No 2',0
5 msg3: DB 'Сообщение No 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 No 1'
13 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintfLF ; 'Сообщение No 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintfLF ; 'Сообщение No 3'
19 _end:
20 call quit ; вызов подпрограммы завершения
```

Рис. 3.2: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. 3.3).


```

apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ █

```

Рис. 3.3: Создание и запуск исполняемого файла

Изменяю текст программы, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу(рис. 3.4).

```

3 msg1: DB 'Сообщение No 1',0
4 msg2: DB 'Сообщение No 2',0
5 msg3: DB 'Сообщение No 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 No 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение No 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение No 3'
21 _end:
22 call quit ; вызов подпрограммы завершения

```

Рис. 3.4: Редактирование файла

Создаю исполняемый файл и запускаю его(рис. 3.5).

```

apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1

```

Рис. 3.5: Создание исполняемого файла

Изменяю текст программы, чтобы она выводила сначала ‘Сообщение № 3’,

потом 'Сообщение№ 2', а затем 'Сообщение № 1' (рис. 3.6).

```
4 msg2: DB 'Сообщение No 2',0
5 msg3: DB 'Сообщение No 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 No 1 '
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение No 2 '
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение No 3 '
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
```

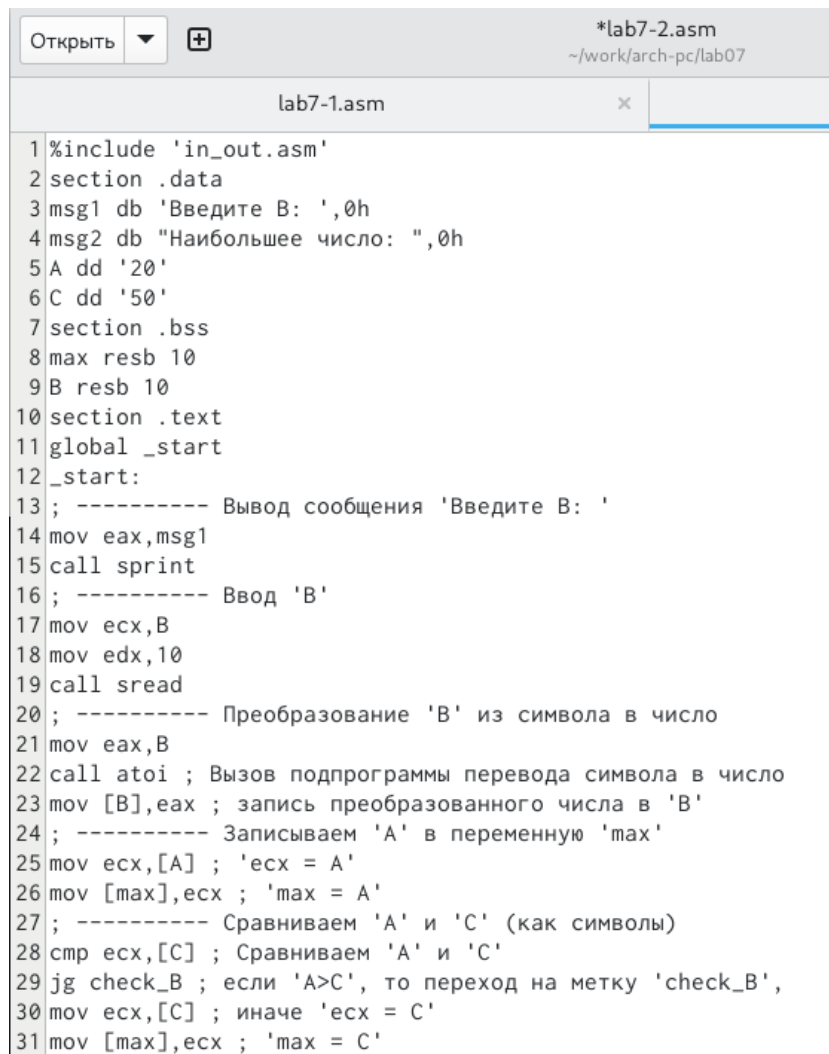
Рис. 3.6: Редактирование файла

Создал и запусти исполняемый файл(рис. 3.7).

```
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $
```

Рис. 3.7: Создание исполняемого файла

Создал файл lab7-2.asm и ввёл в файл lab7-2.asm текст программы, которая определяет и выводит на экран наибольшую из целочисленных переменных: А,В и С(рис. 3.8).



```
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'
```

Рис. 3.8: Создание и редактирование файла

Создаю исполняемый файл и запускаю его несколько раз для разных значений В, чтобы проверить его работу(рис. 3.9).

```

apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ touch lab7-2.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 5
Наибольшее число: 50
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ 2
bash: 2: команда не найдена
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 5
Наибольшее число: 50
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 10
Наибольшее число: 50
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 90
Наибольшее число: 90
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ █

```

Рис. 3.9: Создание исполняемого файла

Создаю файл листинга для программы из файла lab7-2.asm, указав ключ -l(рис. 3.10).

```

apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ █

```

Рис. 3.10: Создание файла

Открываю файл листинга lab7-2.lst (рис. 3.11).

lab7-2.asm		lab7-1.asm	lab7-2.lst
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>	
23	22	<1> sprint:	
24	23 0000000F 52	<1> push edx	
25	24 00000010 51	<1> push ecx	
26	25 00000011 53	<1> push ebx	
27	26 00000012 50	<1> push eax	
28	27 00000013 E8E8FFFFFF	<1> call slen	
29	28	<1>	
30	29 00000018 89C2	<1> mov edx, eax	
31	30 0000001A 58	<1> pop eax	

Рис. 3.11: Листинг

Объяснение содержимого трёх строк файла листинга:

10 строка: Первые цифры [10] - это номер строки файла листинга. Следующие цифры [00000008] адрес — это смещение машинного кода от начала текущего сегмента, состоит из 8 чисел. следующие числа [40] - это машинный код, который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, поэтоу и появляются буквы латынского алфавита. следующее [inc eax] - исходный текст программы, которая просто состоит из строк исходной программы вместе с комментариями

11 строка: Первые цифры [11] - это номер строки файла листинга. Следующие цифры [00000009] адрес — это смещение машинного кода от начала текущего сегмента, состоит из 8 чисел. следующие числа [EBF8] - это машинный код, который представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, поэтоу и появляются буквы латынского алфавита. следующее [jmp nextchar] - исходный текст программы, которая просто состоит из строк исходной программы вместе с комментариями

12 строка: содержит номер строки [11], адресс [00000009], машинный код [EBF8] и содержимое строки кода [jmp nextchar]

Открываю файл с программой lab7-2.asm и в одной из инструкций с двумя операндами удаляю один операнд(рис. 3.12).

```
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
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'
```

Рис. 3.12: Редактирование файла

Пробую выполнить трансляцию с получением файла листинга(рис. 3.13).

```
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:17: error: invalid combination of opcode and operands
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $
```

Рис. 3.13: ошибка

В листинге, как и в коде, появилась ошибка(рис. 3.14).

```
17          *****      error: invalid combination of opcode and operands
18 000000F2 BA0A000000      mov edx,10
19 000000F7 E847FFFFFF      call sread
```

Рис. 3.14: ошибка в листинге

4 Задание Самостоятельной работы

Создал файл lab7-3.asm и написал в нём программу, значения взял из примера моего варианта, то есть из примера второго варианта (рис. 4.1).


```

#include 'in_out.asm'
section .data
msg1 db ' a = ',0h
msg2 db ' b = ',0h
msg3 db ' c = ',0h
msg4 db "Наименьшее число: ",0h
a dd '82'
b dd '59'
c dd '61'

section .bss
max resb 10

section .text
global _start
_start:
; ----- Вывод всех чисел:
mov eax,msg1
call sprint
mov eax,a
call atoi
call iprintLF

mov eax,msg2
call sprint
mov eax,b
call atoi
call iprintLF

mov eax,msg3
call sprint

```

Рис. 4.1: Программа поиска минимума

Создал исполняемый файл и проверил его работу. Все работает корректно (рис.

4.2).

```
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-3
a = 82
b = 59
c = 61
Минимальное число: 59
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ □
```

Рис. 4.2: Создание и тест исполняющего файла

Код программы:

```
%include 'in_out.asm' section .data msg1 db ' a = ',0h msg2 db ' b = ',0h msg3 db ' c
= ',0h msg4 db "Минимальное число:",0h a dd '82' b dd '59' c dd '61'
section .bss max resb 10
section .text global _start _start: ; ---- Вывод чисел: mov eax,msg1 call sprint
mov eax,a call atoi call iprintLF
mov eax,msg2 call sprint mov eax,b call atoi call iprintLF
mov eax,msg3 call sprint mov eax,c call atoi call iprintLF
;-----Сравнение чисел mov eax,b call atoi ;перевод символа в число mov
[b],eax ; запись преобразованного числа в b ;----- запись b в переменную max
mov ecx,[a] ; mov [max],ecx ; ;-----сравнение чисел a с cmp ecx,[c]; if a>c jl
check_b ; то переход на метку mov ecx,[c] ; mov [max],ecx ; ;---метка check_b
check_b: mov eax,max ; call atoi mov [max],eax ; ;----- mov ecx,[max] ; cmp ecx,[b]
; jl check_c ; mov ecx,[b] ; mov [max],ecx ; ;----- check_c: mov eax,msg4 ; call
sprint ; mov eax,[max]; call iprintLF ; call quit
```

2 Задание самостоятельной работы

Создал файл lab7-4.asm и написал в нём программу, которая для введенных с клавиатуры значений x и a вычисляет значение функции f(x) и выводит результат вычислений. (рис. ??).

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите x: ',0h
4 msg2 db 'Введите a: ',0h
5 msg3 db 'f(x) = ',0h
6
7 section .bss
8 x resb 10
9 a resb 10
10
11 section .text
12 global _start
13 _start:
14 mov eax,msg1
15 call sprint
16 mov ecx,x
17 mov edx,10
18 call sread
19 mov eax,x
20 ;-----
21 call atoi
22 mov [x],eax
23 ;-----
24
25 mov eax,msg2
26 call sprint
27 mov ecx,a
28 mov edx,10
29 call sread
30 mov eax,a ;
31 call atoi
32 mov [a],eax ;
33 ;-----
34 mov ecx,[a]
35 cmp ecx,[x] ;x<a
36 jg check_a ;
37 mov ecx,[x]
38 check_a:
39 add ecx,-1;
40 mov eax,msg3 ;
41 call sprint ;
42 mov eax,ecx ;
43 call iprintLF;
44 call quit ;
45

```

Создал исполняемый файл и провёл тест с первой парой значений для варианта 2(рис. 4.3).

```
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ touch lab7-4.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 5
Введите a: 7
f(x) = 6
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ █
```

Рис. 4.3: Тест программы

Тестирую со второй парой значений(рис. 4.4).

```
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x: 6
Введите a: 4
f(x) = 5
apkorchagin@dk5n52 ~/work/arch-pc/lab07 $ █
```

Рис. 4.4: Тест программы

Код программы %include 'in_out.asm' section .data msg1 db 'Введите x:',0h msg2 db 'Введите a:',0h msg3 db 'f(x) =',0h

section .bss x resb 10 a resb 10

section .text global _start _start: mov eax,msg1 call sprint mov ecx,x mov edx,10 call sread mov eax,x ;---- call atoi mov [x],eax ;----

mov eax,msg2 call sprint mov ecx,a mov edx,10 call sread mov eax,a ; call atoi mov [a],eax ; ;---- mov ecx,[a] cmp ecx,[x] ;x<a jg check_a ; mov ecx,[x] check_a: add ecx,-1; mov eax,msg3 ; call sprint ; mov eax,ecx ; call iprintLF; call quit ;

5 Выводы

По ходу выполнения лабораторной работы я изучил команды условных и безусловных переходов в Ассемблере.

Список литературы