Лабораторная работа №7

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Софич Андрей Геннадьевич

Содержание

6	Список литературы	27
5	Выводы	26
	4.1 Реализация переходов в NASM	8 15 18
4		8
3	Теоретическое введение	7
2	Задание	6
1	Цель работы	5

Список иллюстраций

4.1	Создание каталога и фаила	8
4.2	Копирование файла	9
4.3	Редактирование программы	9
4.4	Запуск кода	10
4.5	Редактирование программы	10
4.6	Запуск программы	11
4.7		11
4.8	Запуск программы	12
4.9		13
	The Francisco Francisco Control Contro	14
4.11	J- F- F	15
4.12	Создание файла листинга	15
4.13	Файл листинга	16
4.14	Первая строка	16
4.15	Вторая строка	17
4.16	Третья строка	17
4.17	· · · · · · · · · · · · · · · · · · ·	17
	T	17
4.19		18
	To the Property of the Propert	19
	J- F- F	21
		21
4.23	r r r	22
4.24	Создание объектного файла	24
4.25	Запуск и проверка программы	24
4.26	Повторная проверка программы	25

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 Задание

1. Реализация переходов в NASM 2. Изучение структуры файлов листинга 3.3адания 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. Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предва- рительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре. 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, отра- жающий текущее состояние процессора. Инструкция стр является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция стр является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Для начала работы создаю директорию lab07 в каталоге, в котором работал на протяжении предыдущих работ. После чего создаю новый файл lab7-1.asm, чтобы далее записать в него код программы (рис. 4.1).

```
[andreysofich@fedora ~]$ mkdir ~/work/arch-pc/lab07
[andreysofich@fedora ~]$ cd ~/work/arch-pc/lab07
[andreysofich@fedora lab07]$ touch lab7-1.asm
```

Рис. 4.1: Создание каталога и файла

Не забываю также скопировать в созданный каталог файл in_out.asm, так как далее он будет использоваться во всех программах (рис. 4.2).

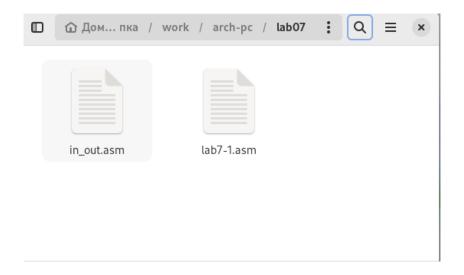


Рис. 4.2: Копирование файла

Открываю файл lab7-1.asm в GNU nano и вставляю теккст программы из листинга 1 (рис. 4.3).

```
GNU nano 7.2 /home/andreysofich/work/arch-pc/lab07/lab7-1.asm %include 'in_out.asm' ; подключение внешнего файла SECTION .data msg1: DB 'Cooбщение No 1',0 msg2: DB 'Cooбщение No 2',0 msg3: DB 'Cooбщение No 3',0 SECTION .text GLOBAL _start _start: jmp _label2 _label1: mov eax, msg1 ; Вывод на экран строки call sprintLF ; 'Cooбщение No 1' _label2: mov eax, msg2 ; Вывод на экран строки call sprintLF ; 'Cooбщение No 2' _label3: mov eax, msg3 ; Вывод на экран строки call sprintLF ; 'Cooбщение No 2' _label3: mov eax, msg3 ; Вывод на экран строки call sprintLF ; 'Cooбщение No 3' _end: call quit ; вызов подпрограммы завершения
```

Рис. 4.3: Редактирование программы

После чего я создаю объектный файл программы, кампаную его и запускаю код (рис. 4.4). Благодаря команде jmp программа сразу перепрыгивает ко второму действию, игнорируя первый этап кода.

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-1.asm
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[andreysofich@fedora lab07]$ ./lab7-1
Сообщение No 2
Сообщение No 3
```

Рис. 4.4: Запуск кода

Далее я изменяю текст программы в соответствии с листингом 2 (рис. 4.5).

```
GNU nano 7.2
%include 'in_out.asm' ; подключение внешнего файла
    : DB 'Сообщение No 1',0
        'Сообщение No 2',0
    : DB 'Сообщение No 3',0
  OBAL _start
jmp _label2
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 2'
jmp _label1
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 3'
call quit ; вызов подпрограммы завершения
```

Рис. 4.5: Редактирование программы

Кампаную созданный объектный файл и запускаю программу (рис. 4.6). Теперь программа с самого начала перепрыгивает ко второму участку кода,после чего (из-за той же команды jmp) переходит к первому участку,где и заканчивает свою работу.

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-1.asm
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[andreysofich@fedora lab07]$ ./lab7-1
Сообщение No 2
Сообщение No 1
```

Рис. 4.6: Запуск программы

Редактирую код программы в соответствии с заданием: сначала выводится "Сообщение №3", затем "Сообщение №2", затем "Сообщение №1", для этого я использую всё ту же команду jmp (рис. 4.7).

```
DB 'Сообщение No 1',0
     DB 'Сообщение No 2',0
       3 'Сообщение No 3',0
   ION .text
    AL _start
jmp _label3
mov eax, msgl ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 2'
jmp _label1
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 3'
jmp _label2
call quit ; вызов подпрограммы завершения
```

Рис. 4.7: Создание новой программы

```
%include 'in_out.asm' ; подключение внешнего файла SECTION .data
```

```
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end
_label2:
mov eax, msg2; Вывод на экран строки
call sprintLF ; 'Сообщение No 2'
jmp _label1
_label3:
mov eax, msg3; Вывод на экран строки
call sprintLF ; 'Сообщение No 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения
```

Запускаю программу и проверяю правильность написания программы (рис.

4.8). После запуска программа выёт результат в правильном порядке.

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-1.asm
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[andreysofich@fedora lab07]$ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
```

Рис. 4.8: Запуск программы

Чтобы писать следующий код, создаю новый файл lab7-2.asm,используя утилиту touch (рис. 4.9).

[andreysofich@fedora lab07]\$ touch lab7-2.asm

Рис. 4.9: Создание файла

Вставляю в созданный файл текст кода, скопировав листинг 7.3 (рис. 4.10). Данная программа позволяет находить наибольшее число из двух констант и одного произвольного числа.

```
GNU nano 7.2
%include 'in_out.asm'
section .data
msgl db 'Введите В: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
; ----- Вывод сообщения 'Введите В: '
mov eax,msgl
call sprint
; ----- Ввод 'В'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'В' 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 'А' и 'С' (как символы)ы
стр есх,[C] ; Сравниваем 'А' иы 'С'
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],еах ; запись преобразованного числа в `max`
; ----- Сравниваем 'max(A,C)' и 'В' (как числа)
mov ecx,[max]
стр есх,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>В', то переход на 'fin',
mov есх,[В] ; иначе 'есх = В'
mov [max],ecx
; ----- Вывод результата
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход
```

Рис. 4.10: Редактирование программы

Запускаю код дважды, в первый раз я ввожу число,которое меньше одной из констант, а во второй-больше (рис. 4.11).В обоих случаях программа выдает наибольшее из трёх чисел.

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-2.asm
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[andreysofich@fedora lab07]$ ./lab7-2
Введите В: 25
Наибольшее число: 50
[andreysofich@fedora lab07]$ ./lab7-2
Введите В: 70
Наибольшее число: 70
```

Рис. 4.11: Запуск программы

4.2 Изучение структуры файлов листинга

Создаю файл листинга для lab7-2.asm (рис. 4.12).

```
[andreysofich@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
```

Рис. 4.12: Создание файла листинга

Открываю созданный файл с помощью mcedit,используя команду "mcedit lab7-2.lst" (рис. 4.13).

Рис. 4.13: Файл листинга

Первая строка,которую я хочу подробно описать,является строка №5 (рис. 4.14). Первое число в строке(самое левое) как раз и указывает на номер строки, после чего мы можем увидеть адрес данной строки (00000001), далее у нас идет машинный код,в который ассемблируется инструкция. То есть инструкция "mov ebx,eax" ассемблируется в 89С3. В данном случае, 89С3-это инструкция на машинном языке по вызову и присваиванию регистра. После этого(в правой части) мы можем увидеть исходный текст программы.



Рис. 4.14: Первая строка

Вторая строка, которую я хочу подробно описать, является строка №11 (рис. 4.16). Первое число в строке(самое левое) как раз и указывает на номер строки, после чего мы можем увидеть адрес данной строки (00000009), далее у нас идет машинный код, в который ассемблируется инструкция. То есть инструкция "jmp nextchar" ассемблируется в ЕВF8. В данном случае, ЕВF8-это инструкция на машинном языке по переходу на другую строку. После этого(в правой части) мы можем увидеть исходный текст программы.

11 00000009 EBF8 <1> jmp nextchar.....

Рис. 4.15: Вторая строка

Третья строка, которую я хочу подробно описать, является строка №39 (рис. 4.17). Первое число в строке(самое левое) как раз и указывает на номер строки, после чего мы можем увидеть адрес данной строки (00000145), далее у нас идет машинный код, в который ассемблируется инструкция. То есть инструкция "стр есх" ассемблируется в 380D. В данном случае, 380D-это инструкция на машинном языке по сравнению чисел. В квадратных скобках рядом можно увидеть адрес, который указывает на значение числа В. После этого(в правой части) мы можем увидеть исходный текст программы.



Рис. 4.16: Третья строка

Далее я открыл файл с программой lab7-2.asm и в инструкции с двумя операндами удалил один из них (рис. 4.18).



Рис. 4.17: Удаление операнда

Выполняю трансляцию с получением файла листинга (рис. ??). Транслятор выводит ошибку при ассемблировании, даже указывая на номер строки, и файл листинга не создается.

```
jg ; если '<u>max(A,C</u>)><u>B</u>', то переход на '<u>fin</u>',
```

Рис. 4.18: Попытка создать файл листинга

4.3 Задания для самостоятельной работы

Создаю файл,в котором буду делать первое самостоятельное задание (рис. 4.19).

[andreysofich@fedora lab07]\$ touch lab7-3.asm

Рис. 4.19: Создание файла

В лабораторной работе №6 мне выпал 12 вариант,именно его я и буду выполнять, мне выпали числа 99,26,29. Пишу программу,которая выбирает наименьшее число из трех заданных чисел. С начала выбираю меньшее из А и В, а после сравниваю его с С (рис. 4.20).

```
GNU nano 7.2
%include 'in_out.asm'
section .data
msg2 db "Наименьшее число:",0h
A dd '99'
B dd '29'
C dd '26'
min resb 10
global _start
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ----- Сравниваем 'А' и 'В' (как символы)
стр есх,[В] ; Сравниваем 'А' и 'В'
jl check_C ; если 'A<B', то переход на метку 'check_C'
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Преобразование 'max(A,C)' из символа в чис
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в `min`
; ----- Сравниваем 'min(A,B)' и 'С' (как числа)
mov ecx,[min]
cmp ecx,[C] ; Сравниваем 'min(A,B)' и 'C'
jl fin ; если 'min(A,B)<C', то переход на 'fin',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx
; ----- Вывод результата
mov eax, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход
```

Рис. 4.20: Редактирование программы

```
%include 'in_out.asm'
section .data
```

```
msg2 db "Наименьшее число:",0h
A dd '99'
B dd '29'
C dd '26'
min resb 10
global _start
_start:
mov eax, C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C], eax ; запись преобразованного числа в 'C'
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'max = A'
; ----- Сравниваем 'А' и 'В' (как символы)
стр есх, [В] ; Сравниваем 'А' и 'В'
jl check_C ; если 'A<B', то переход на метку 'check_C',
mov ecx, \lceil B \rceil; uhave 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Преобразование 'max(A,C)' из символа в число
check C:
mov eax, min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min], eax ; запись преобразованного числа в `min`
; ----- Сравниваем 'min(A,B)' и 'С' (как числа)
mov ecx, [min]
cmp ecx, [С] ; Сравниваем 'min(A,B)' и 'С'
jl fin ; если 'min(A,B)<C', то переход на 'fin',
mov ecx,[C] ; иначе 'ecx = C'
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
```

Запускаю исполняемый, проверяю ответ и убеждаюсь, что программа выдает наименьшее из трёх чисел (рис. 4.21).

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-3.asm
lab7-3.asm:7: warning: uninitialized space declared in non-BSS section `.data': zeroi
zeroing]
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[andreysofich@fedora lab07]$ ./lab7-3
Наименьшее число:26
```

Рис. 4.21: Запуск программы

Для следующего задания создаю ещё один файл lab7-4.asm (рис. 4.22).

[andreysofich@fedora lab07]\$ touch lab7-4.asm

Рис. 4.22: Создание файла

Пишу программу по заданию 12 варианта (рис. 4.23). Прошу на вводу x и а, после чего преобразую их из символов в числа, сравниваю x с 5 и в зависимости от значения x выполняю действия (либо x*а либо x-5)

```
GNU nano 7.2
%include 'in_out.asm'
section .data
msg1 db "Введите х:",0h
msg2 db "Введите а:",0h
msg3 db "Ответ:",0h
x resb 10
a resb 10
global _start
mov eax,msgl
call sprint
mov ecx,x
mov edx,10
call sread
mov eax,x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],еах ; запись преобразованного числа в 'x'
mov eax,msg2
call sprint
mov ecx,a
mov edx,10
call sread
; ----- Преобразование 'а' из символа в число
mov eax,a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'
mov eax,[x]
cmp eax,5
jl fin
jmp finl
mov eax, msg3
call sprint
mov eax,[x]
mov ebx,[a]
mul ebx
call iprintLF
call quit ; Выход
mov eax,msg3
call sprint
mov eax,[x]
add eax,-5
call iprintLF
call quit ; Выход
```

Рис. 4.23: Редактирование программы

```
%include 'in_out.asm'
section .data
```

```
msg1 db "Введите х:",0h
msg2 db "Введите a:",0h
msg3 db "OTBET:",0h
x resb 10
a resb 10
global _start
_start:
mov eax,msg1
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x],eax ; запись преобразованного числа в 'x'
mov eax,msg2
call sprint
mov ecx, a
mov edx, 10
call sread
; ----- Преобразование 'а' из символа в число
mov eax, a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a],eax ; запись преобразованного числа в 'a'
mov eax,[x]
cmp eax, 5
jl fin
```

```
jmp fin1
fin:
mov eax, msg3
call sprint
mov eax,[x]
mov ebx, [a]
mul ebx
call iprintLF
call quit ; Выход
fin1:
mov eax,msg3
call sprint
mov eax,[x]
add eax, -5
call iprintLF
call quit ; Выход
```

Создаю объектный файл (рис. 4.24).

```
[andreysofich@fedora lab07]$ nasm -f elf lab7-4.asm
```

Рис. 4.24: Создание объектного файла

Компаную и запускаю исполняемый файл,вводя числа для первой проверки: x=3,a=7 (рис. 4.25). Программа выдает нужный ответ,умножая 3*7.

```
[andreysofich@fedora lab07]$ ld -m elf_i386 -o lab7-4 lab7-4.o
[andreysofich@fedora lab07]$ ./lab7-4
Введите х:6
Введите a:4
Ответ:1
```

Рис. 4.25: Запуск и проверка программы

Выполняю повторную проверку программы, вводя пару других чисел:x=6,a=6 (рис. 4.26). Теперь программа видит,что x>5и вычитает из него 5, программа работает верно.

```
[andreysofich@fedora lab07]$ ./lab7-4
Введите х:3
Введите a:7
Ответ:21
```

Рис. 4.26: Повторная проверка программы

5 Выводы

В этой работе я научился работать с переходами в NASM, разобрался со структурой листинговых файлов и научился применять эти знания для написания программ.

6 Список литературы

Лабораторная работа №7.