

Лабораторная работа №7

Дисциплина: Архитектура компьютера

Серебрякова Дарья Ильинична

Содержание

1	Цель работы	5
2	Задания	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация переходов в NASM	9
4.2	Изучение структуры файлов листинга	12
4.3	Выполнение заданий для самостоятельной работы	13
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Создание каталога для файлов	9
4.2	Текст программы	9
4.3	Запуск программы	10
4.4	Измененная программа 1	10
4.5	Измененная программа 2	11
4.6	Запуск исполняемого файла	11
4.7	Программа с предложенного листинга	12
4.8	Создание файла листинга	12
4.9	Файл листинга	13
4.10	Написание программы в новом файле	14
4.11	Проверка написанной программы	14
4.12	Первая часть программы	15
4.13	Вторая часть программы	16
4.14	Проверка написанной программы	16

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 Задания

1. Реализация переходов в NASM
2. Изучение структуры файла листинга

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.

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление: `jmp` Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре

Как отмечалось 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.

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную

информацию.

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся. Итак, структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

4 Выполнение лабораторной работы

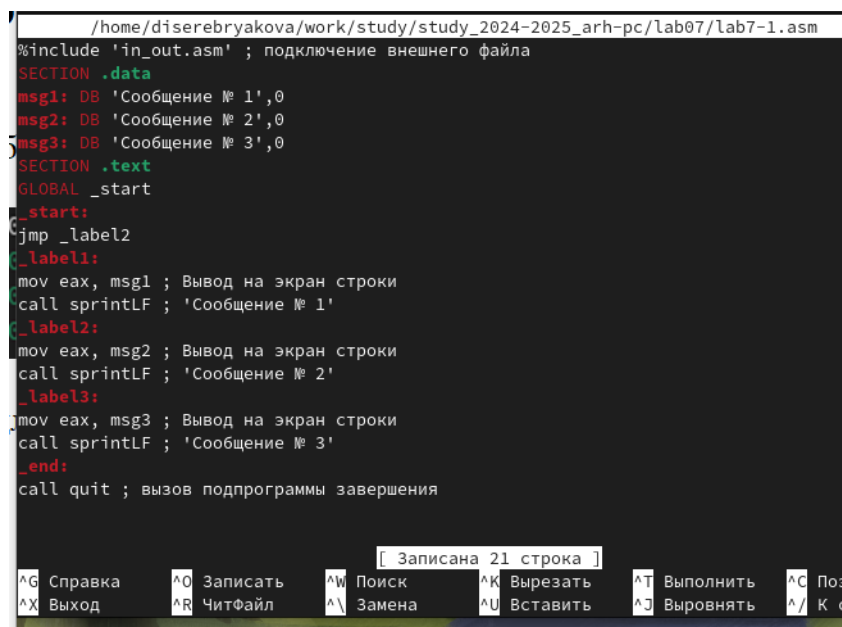
4.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm (рис. 4.1).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ mkdir lab07
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ cd lab07
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ touch lab7-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$
```

Рис. 4.1: Создание каталога для файлов

Ввожу в файл lab7-1.asm текст программы из предложенного листинга (рис. 4.2).



```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab07/lab7-1.asm
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call printf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Рис. 4.2: Текст программы

Копирую файл in_out.asm в каталог lab07. Создаю исполняемый файл и запускаю его (рис. 4.3).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$
```

Рис. 4.3: Запуск программы

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

Далее изменяю текст программы в соответствии с предложенным листингом. Создаю исполняемый файл и запускаю его (рис. 4.4).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$
```

Рис. 4.4: Измененная программа 1

В результате изменения текста программы поменялся порядок выполнения инструкций. Выполнение инструкций началось с метки `_label2`, она направила на ветку `_label1`, которая отправляет в конец программы.

Изменяю текст программы, чтобы программа выводила сообщения 3, 2, 1 (рис. 4.5).

```

/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab07/lab7-1.asm
%include 'in_out.asm' ; подключение внешнего файла
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 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 4.5: Измененная программа 2

Создаю исполняемый файл и запускаю его (рис. 4.6).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$

```

Рис. 4.6: Запуск исполняемого файла

Вывод моей программы совпадает с тем, что должно быть, значит текст программы изменен верно.

Создаю файл lab7-2.asm в каталоге lab07. Внимательно изучаю текст программы из предложенного листинга и ввожу его в lab7-2.asm. Создаю исполняемый файл и запускаю его (рис. 4.7).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-2.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-2
Введите В: 34
Наибольшее число: 50
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-2
Введите В: 54
Наибольшее число: 54
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-2
Введите В: 26
Наибольшее число: 50
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$
```

Рис. 4.7: Программа с предложенного листинга

Проверяю работу программы для различных значений В, все выполняется верно

4.2 Изучение структуры файлов листинга

Создаю файл листинга для программы из файла lab7-2.asm и открываю его с помощью текстового редактора mcedit (рис. 4.8).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ mcedit lab7-2.lst
```

Рис. 4.8: Создание файла листинга

Листинг программы имеет три столбца, в которых отображаются команды исходной ассемблерной программы и соответствующие им объектные коды:

1. Левый столбец содержит шестнадцатеричное значение смещения адреса команды (счетчик команд — IP) от начала сегмента;
2. правый столбец содержит операторы и псевдооператоры ассемблера (команды и директивы программы);
3. в средней части размещены коды: для сегмента стека и сегмента данных — числа, запоминаемые в соответствующих ячейках памяти; для сегмента команд это коды машинных команд МП, соответствующих операторам ассемблера (рис. 4.9).

```

lab7-2.lst      [-----] 53 L: [ 1+ 0 1/225] *(53 /14458b) 0111 0x06F  [*][X]
1               %include 'in_out.asm'
1               <1> ;----- slen -----
2               <1> ; Функция вычисления длины сообщения
3               <1> slen:.....
4 00000000 53    <1> push  ebx.....
5 00000001 89C3  <1> mov   ebx, eax.....
6               <1>.....
7               <1> nextchar:.....
8 00000003 803800 <1> cmp   byte [eax], 0...
9 00000006 7403   <1> jz    finished.....
10 00000008 40    <1> inc   eax.....
11 00000009 EBF8  <1> jmp   nextchar.....
12               <1>.....
13               <1> finished:
14 0000000B 29D8  <1> sub   eax, ebx
15 0000000D 5B    <1> pop   ebx.....
16 0000000E C3    <1> ret.....
17               <1>.....
18               <1>.....
19               <1> ;----- sprint -----
20               <1> ; Функция печати сообщения
21               <1> ; входные данные: mov eax,<message>
22               <1> sprint:
23 0000000F 52    <1> push  edx
24 00000010 51    <1> push  ecx
25 00000011 53    <1> push  ebx

```

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ить 7Поиск 8Удалить 9МенюМС 10Выход

Рис. 4.9: Файл листинга

4.3 Выполнение заданий для самостоятельной работы

Для выполнения первого задания командой touch создаю файл lab7-3сору.asm, открываю его и пишу текст необходимой программы (рис. 4.10).

```

GNU nano 7.2 /home/diserebryakova/work/study/study_2024-2025_arh-pc/lab07/lab7-3copy.asm
#include 'in_out.asm'
section .data
msg1 db 'Наименьшая переменная: ',0h
A dd '81'
B dd '22'
C dd '72'
section .bss
min resb 10
section .text
global _start
_start:
; ----- Преобразуем переменные из символов в числа
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
mov eax,C
call atoi ; Вызов подпрограммы перевода символа в число
mov [C],eax ; запись преобразованного числа в 'C'
mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'
; ----- Записываем C в переменную min
mov ecx,[C]
mov [min],ecx
; ----- Сравниваем 'C' и 'A'
cmp ecx,[A] ; Сравниваем 'C' и 'A'
jle check_B; если 'C<=A', то переход на метку 'check_B',
mov ecx,[A]
mov [min],ecx ; иначе 'min = A' То есть 'A' меньше
check_B:
; ----- Сравниваем 'min(A,C)' и 'B'
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jle fin ; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[B]
mov [min],ecx ; иначе 'min = B'
; ----- Вывод результата
fin:
mov eax,msg1
call sprint ; Вывод сообщения 'Наименьшая переменная: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

Рис. 4.10: Написание программы в новом файле

Создаю исполняемый файл и проверяю правильность написанной программы (рис. 4.11).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/labs/lab07/report$
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-3copy.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-3copy
lab7-3copy.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-3copy
Наименьшая переменная: 22

```

Рис. 4.11: Проверка написанной программы

Наименьшая переменная найдена верно, значит программа написана правильно. Приступаю к выполнению второго задания. Создаю файл lab7-4.asm и пишу в нем текст программы. Начало (рис. 4.12).

```

/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab07/lab7-4.
%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
result resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите x: '
mov eax,msg1
call sprint
; ----- Ввод 'X'
mov ecx,X
mov edx,10
call sread
; ----- Преобразование 'x' из символа в число
mov eax,X
call atoi ; Вызов подпрограммы перевода символа в число
mov [X],eax ; запись преобразованного числа в 'X'
; ----- Вывод сообщения 'Введите A: '
mov eax,msg2
call sprint
; ----- Ввод 'A'
mov ecx,A
mov edx,10
call sread
; ----- Преобразование 'A' из символа в число
mov eax,A
call atoi ; Вызов подпрограммы перевода символа в число
mov [A],eax ; запись преобразованного числа в 'A'
; ----- Сравниваем x и a для получения формулы для вычисления

```

Рис. 4.12: Первая часть программы

Конец (рис. 4.13).

```

; ----- Сравниваем x и a для получения формулы для вычисления
mov eax,[X]
mov ebx,[A]
cmp eax,ebx
jl sekond_var ; Если x < a, то перейти к 3*a+1
; ----- Вычисление 3*x+1
imul eax,eax,3 ; eax = 3*x
add eax,1 ; eax = 3*x+1
jmp result_f
sekond_var:
; ----- Вычисление 3*a+1
mov eax,[A] ; eax = a
imul eax,eax,3 ; eax = 3*a+1
add eax,1
result_f:
mov [result],eax
; ----- Вывод результата
fin:
mov eax,msg3
call sprint ; вывод 'f(x) = '
mov eax,[result]
call iprintLF ; Вывод [result]
call quit

```

Рис. 4.13: Вторая часть программы

Создаю исполняемый файл, запускаю его и проверяю работу программы на предложенных для моего варианта значениях (рис. 4.14).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ nasm -f elf lab7-4.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ld -m elf_i386 -o lab7-4 lab
7-4.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-4
Введите x: 2
Введите a: 3
f(x) = 10
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab07$ ./lab7-4
Введите x: 4
Введите a: 2
f(x) = 13

```

Рис. 4.14: Проверка написанной программы

Программа выводит верный результат, а это значит, что она написана верно.

5 Выводы

В ходе выполнения лабораторной работы были изучены команды условного и безусловного переходов, приобретены навыки написания программ с использованием переходов и ознакомление с назначением и структурой файла листинга

Список литературы

1. Лабораторная работа №7