

Отчёт по лабораторной работе №7

Дисциплина: Архитектура компьютера

Абрамова Ульяна Михайловна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Команды безусловного перехода	7
3.2	Команды условного перехода	7
3.3	Файл листинга и его структура	8
4	Выполнение лабораторной работы	9
4.1	Реализация переходов в NASM	9
4.2	Изучение структуры файла листинга	13
4.2.1	Объяснение содержимого некоторых строк	13
4.3	Выполнение заданий для самостоятельной работы	15
4.3.1	Написание программы нахождения наименьшей из 3 целочисленных переменных a,b и c	15
4.3.2	Написание программы вычисления выражения при введенных x и a	16
5	Выводы	18
6	Список литературы	19

Список иллюстраций

4.1	Написание программы	9
4.2	Запуск программы	10
4.3	Изменение текста программы	10
4.4	Запуск программы	11
4.5	Изменение программы	11
4.6	Запуск программы	11
4.7	Написание программы	12
4.8	Запуск программы	12
4.9	Дополнительная проверка	12
4.10	Создание и открытие файла листинга	13
4.11	mov eax,B	13
4.12	call atoi	13
4.13	mov eax,[max]	14
4.14	Удаление одного из операндов в инструкции	14
4.15	Попытка создания файла	14
4.16	Отображение ошибки	14
4.17	Написание программы	15
4.18	Запуск программы	16
4.19	Написание программы	16
4.20	Запуск программы	17

List of Tables

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 файла листинга
3. Выполнение заданий 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

3.1 Команды безусловного перехода

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

3.2 Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

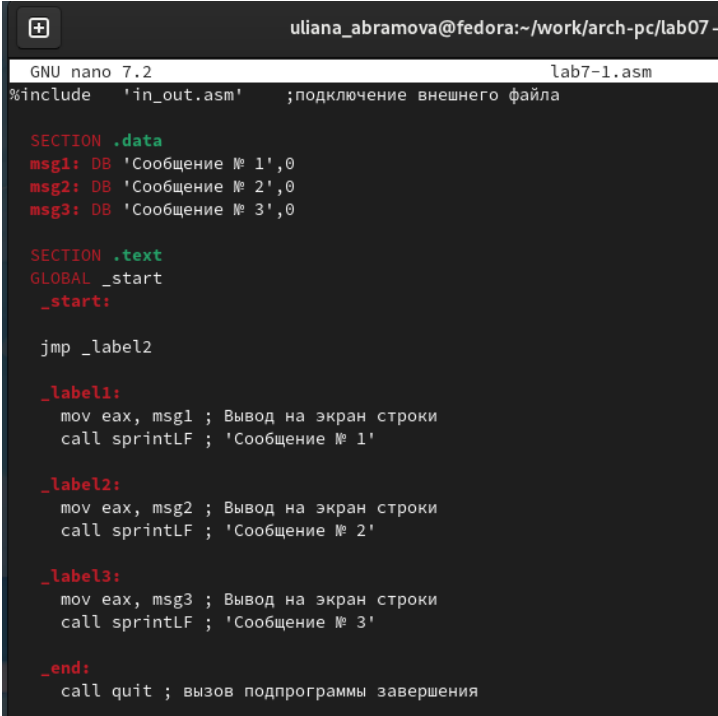
3.3 Файл листинга и его структура

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 и, перейдя в него, создаю файл lab7-1.asm, в который ввожу программу с использованием инструкции jmp (рис. 4.1).



```
uliana_abramova@fedora:~/work/arch-pc/lab07-  
GNU nano 7.2 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 sprintLF ; 'Сообщение № 1'  
  
_label2:  
    mov eax, msg2 ; Вывод на экран строки  
    call sprintLF ; 'Сообщение № 2'  
  
_label3:  
    mov eax, msg3 ; Вывод на экран строки  
    call sprintLF ; 'Сообщение № 3'  
  
_end:  
    call quit ; вызов подпрограммы завершения
```

Рис. 4.1: Написание программы

Создаю исполняемый файл и запускаю его. Программа отработана верно (рис. 4.2).

```

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

```

Рис. 4.2: Запуск программы

Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Изменяю 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. 4.3)

```

GNU nano 7.2 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 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'

_end:
call quit ; вызов подпрограммы завершения

```

Рис. 4.3: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.4)

```

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

```

Рис. 4.4: Запуск программы

Изменяю текст программы, чтобы в выводе было сначала ‘Сообщение №3’, потом ‘Сообщение №2’ и последним ‘Сообщение №1’ (рис. 4.5).

```

GNU nano 7.2 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: Изменение программы

Создаю исполняемый файл и проверяю корректность написания программы (рис. 4.6).

```

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

```

Рис. 4.6: Запуск программы

Создаю в том же каталоге файл lab7-2.asm и ввожу в нем программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С (рис. 4.7)

```
uliana_abramova@fedora:~/work/arch-pc/lab07 — nano lab7
GNU nano 7.2 lab7-2.asm
#include 'in_out.asm'
section .data
    msg1 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
_start:
; ----- Вывод сообщения 'Введите В: '
    mov eax,msg1
    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 'В'
; ----- Записываем 'А' в переменную 'max'
    mov ecx,[A] ; 'ecx = A'
    mov [max],ecx ; 'max = A'
; ----- Сравниваем 'А' и 'С' (как символы)
    cmp ecx,[C] ; Сравниваем 'А' и 'С'
    jg check_B ; если 'А>С', то переход на метку 'check_B',
    mov ecx,[C] ; иначе 'ecx = C'
    mov [max],ecx ; 'max = C'
```

Рис. 4.7: Написание программы

Создаю исполняемый файл и проверяю его работу для различных значений В (рис. 4.8,4.9)

```
uliana_abramova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
uliana_abramova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
uliana_abramova@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 5
Наибольшее число: 50
```

Рис. 4.8: Запуск программы

```
uliana_abramova@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 100
Наибольшее число: 100
```

Рис. 4.9: Дополнительная проверка

4.2 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm и открываю с помощью текстового редактора mcedit для ознакомления с его форматом и содержанием (рис. 4.10)

```
uliana_abramova@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
uliana_abramova@fedora:~/work/arch-pc/lab07$ mcedit lab7-2.lst
```

Рис. 4.10: Создание и открытие файла листинга

4.2.1 Объяснение содержимого некоторых строк

1. Эта строка находится на 21 месте, ее адрес “00000101”, Машинный код - B8 [0A000000],

mov eax,B – исходный текст программы, означающий что в регистр eax мы вносим значения переменной B. (рис. 4.11)

```
21 00000101 B8[0A000000] mov eax,B
```

Рис. 4.11: mov eax,B

2. Эта строка находится на 35 месте, ее адрес “00000134”, Машинный код - E863FFFFFF, call atoi – исходный текст программы, означающий что символ лежащий в строке выше переводится в число. (рис. 4.12)

```
35 00000135 E862FFFFFF call atoi
```

Рис. 4.12: call atoi

3. Эта строка находится на 47 месте, ее адрес “00000162”, Машинный код - A1[00000000], mov eax,[max] – исходный текст программы, означающий что число хранившееся в переменной max записывается в регистр eax. (рис. 4.13)

```
47 00000163 A1[00000000]          mov eax,[max]
```

Рис. 4.13: mov eax,[max]

В инструкции 'mov eax,max' удаляю max и пытаюсь создать исполняемый файл, однако возникает ошибка, так как для программы требуется два операнда (рис. 4.14,4.15).

```
check_B:
    mov    eax
```

Рис. 4.14: Удаление одного из операндов в инструкции

```
uliana_abramova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
lab7-2.asm:34: error: invalid combination of opcode and operands
```

Рис. 4.15: Попытка создания файла

В файле листинга показывается где именно ошибка и с чем она связана (рис. 4.16)

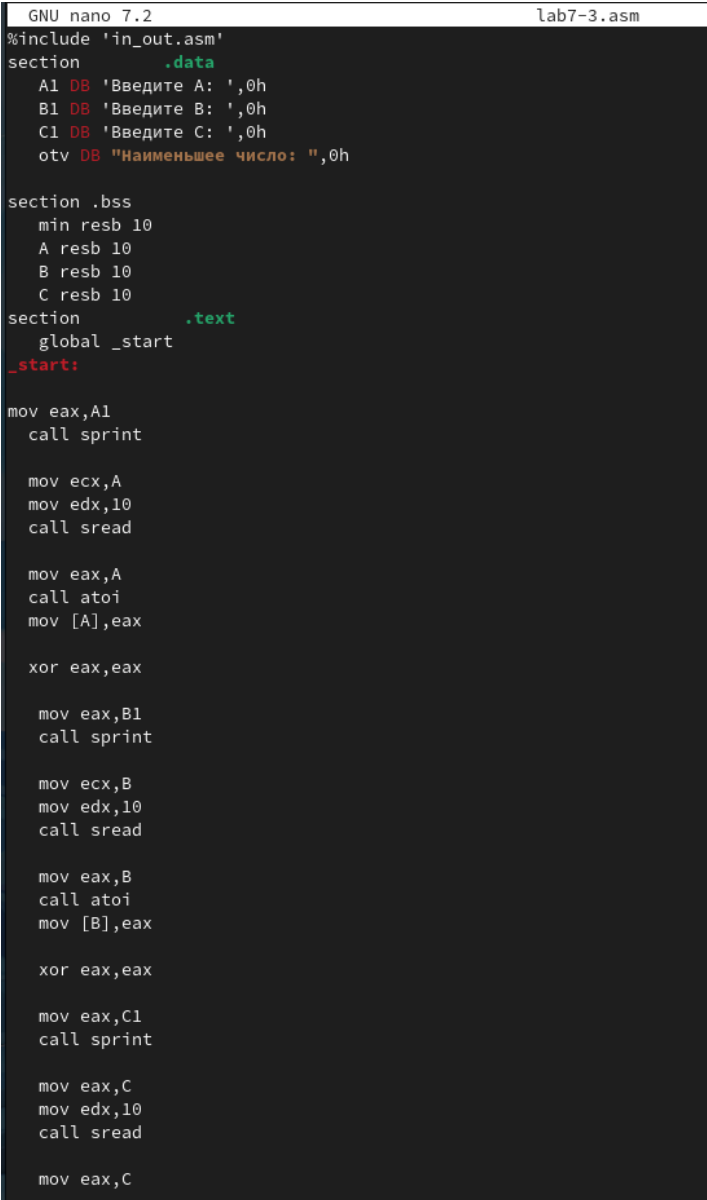
```
34                                     mov eax
34 ***** error: invalid combination of opcode and operands
```

Рис. 4.16: Отображение ошибки

4.3 Выполнение заданий для самостоятельной работы

4.3.1 Написание программы нахождения наименьшей из 3 целочисленных переменных a,b и c

Так как числа были изначально указаны в условии, я сделала возможным их ввести с клавиатуры. Числа: a=94; b=5; c=58 (рис. 4.17)



```
GNU nano 7.2 lab7-3.asm
#include 'in_out.asm'
section .data
    A1 DB 'Введите A: ',0h
    B1 DB 'Введите B: ',0h
    C1 DB 'Введите C: ',0h
    otv DB "Наименьшее число: ",0h

section .bss
    min resb 10
    A resb 10
    B resb 10
    C resb 10

section .text
    global _start
_start:

    mov eax,A1
    call sprint

    mov ecx,A
    mov edx,10
    call sread

    mov eax,A
    call atoi
    mov [A],eax

    xor eax,eax

    mov eax,B1
    call sprint

    mov ecx,B
    mov edx,10
    call sread

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

    xor eax,eax

    mov eax,C1
    call sprint

    mov eax,C
    mov edx,10
    call sread

    mov eax,C
```

Рис. 4.17: Написание программы

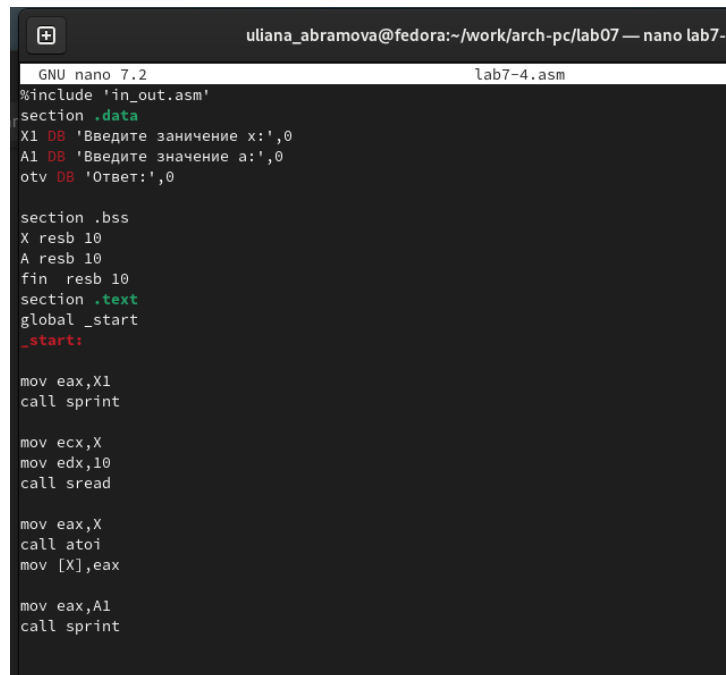
Создаю исполняемый файл и проверяю верную отработку программы (рис. 4.18).

```
uliana_abramova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
uliana_abramova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
uliana_abramova@fedora:~/work/arch-pc/lab07$ ./lab7-3
Введите A: 94
Введите B: 5
Введите C: 58
Наименьшее число: 5
```

Рис. 4.18: Запуск программы

4.3.2 Написание программы вычисления выражения при введенных x и a

Программа написана для вида функции 3 варианта (рис. 4.19).



```
uliana_abramova@fedora:~/work/arch-pc/lab07 — nano lab7-
GNU nano 7.2 lab7-4.asm
#include 'in_out.asm'
section .data
X1 DB 'Введите значение x:',0
A1 DB 'Введите значение a:',0
otv DB 'Ответ:',0

section .bss
X resb 10
A resb 10
fin resb 10
section .text
global _start
_start:

mov eax,X1
call sprint

mov ecx,X
mov edx,10
call sread

mov eax,X
call atoi
mov [X],eax

mov eax,A1
call sprint
```

Рис. 4.19: Написание программы

Создаю исполняемый файл и проверяю работу программы (рис. 4.20).


```
uliana_abramova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
uliana_abramova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
uliana_abramova@fedora:~/work/arch-pc/lab07$ ./lab7-4
Введите значение x:3
Введите значение a:4
Ответ:9
uliana_abramova@fedora:~/work/arch-pc/lab07$ ./lab7-4
Введите значение x:1
Введите значение a:4
Ответ:5
```

Рис. 4.20: Запуск программы

5 Выводы

Я изучила команды условного и безусловного перехода. Приобрела навыки написания программ с переходами.

6 Список литературы

1. Архитектура ЭВМ