

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: Архитектура компьютера

Студент: Армихос Гонзалез Карла

Группа: НКАбд-02-24

МОСКВА

2024 г.

Содержание

1. Цель работы.....	4
2. Задание.....	5
3. Технические введение.....	6
4. Порядок выполнения лабораторной работы.....	9
5. Задание для самостоятельной работы.....	14
6. Выводы.....	17
7. Список используемой литературы.....	18

Список иллюстраций

пример использования команды сравнения и команд условного перехода.....	7
рис. 4.1.1 Создание директории.....	9
рис. 4.2 Программа jmp.....	9
рис. 4.3 Запустить программу (2-3).....	10
рис. 4.4 Добавление новых инструкций в программу.....	11
рис. 4.5 Запустить программу(2,1).....	11
рис. 4.1.7 Изменяем программу.....	11
рис 4.7 Запустить программу.....	12
рис 4.8 Создайте файл lab7-2.asm.....	12
рис 4.9.....	12
рис 4.10 программа.....	13
рис 5.2 Программа.....	16

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. Работа с числовыми символами.
2. Запускать разные программы
3. Выполнение арифметических задач.
4. Выполнять работу самостоятельно

3. Технические введение

Файл листинга в NASM используется для анализа и отладки программы. Он представляет собой текстовый файл, в котором содержатся адреса инструкций, машинный код, исходный текст программы и дополнительные комментарии. Основное отличие от текста программы заключается в том, что текст программы — это чисто исходный код на языке ассемблера, а листинг дополнительно содержит машинные коды и помогает понять, как исходный код транслируется в команды для процессора.

Файл листинга NASM состоит из следующих частей:

- **Адресация:** Показаны адреса каждой команды в памяти.
- **Машинный код:** Байт-код инструкций, который генерируется компилятором.
- **Исходный текст:** Код на языке ассемблера.
- **Комментарии:** Добавлены для наглядности, чтобы облегчить понимание.

Пример структуры:

```
00000000 B8 01 00 00 00  mov eax, 1
```

Ветвление в ассемблере выполняется с помощью переходов. Оно может быть:

- **Безусловным** (выполняется всегда).
- **Условным** (зависит от результата предыдущих операций, таких как сравнение или арифметика).
- **Безусловные переходы:**
jmp — переход на указанную метку или адрес.
- **Условные переходы:**
je / jz — переход при равенстве / нулевом флаге.
- *jne / jnz* — переход при неравенстве / отсутствии нулевого флага.
- *jg / jnle* — переход, если больше (без знака).
- *jl / jnge* — переход, если меньше (без знака).

- `jge / jnl` — переход, если больше или равно.
- `jle / jng` — переход, если меньше или равно.

Команда **cmp** используется для сравнения двух операндов. Она выполняет вычитание второго операнда из первого, не сохраняя результат, а только устанавливая флаги процессора:

- **ZF (Zero Flag)**: Устанавливается, если операнды равны.
- **CF (Carry Flag)**: Устанавливается, если произошло заимствование (в случае беззнаковых чисел).
- **SF (Sign Flag)**: Указывает знак результата.
- **OF (Overflow Flag)**: Устанавливается при переполнении.

Команд условного перехода

Синтаксис условного перехода в NASM:

<команда перехода> <метка>

Пример:

```
je equal_label
```

пример использования команды сравнения и команд условного перехода.

Пример:

```
mov eax, 5
```

```
cmp eax, 10 ; Сравнение 5 и 10
```

```
jl less_label ; Переход, если eax < 10
```

```
; Продолжение программы
```

```
less_label:
```

```
    ; Код выполняется только если eax < 10
```

Безусловные переходы (`jmp`) не анализируют флаги. Они всегда выполняются, вне зависимости от состояния процессора.

4. Порядок выполнения лабораторной работы

Создайте каталог для программ лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm: (рис. 4.1)

```
gkarmikhos@fedora:~$ mkdir ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab07
gkarmikhos@fedora:~$ cd ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab07
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

рис. 4.1.1 Создание директории

В файле 7-1. asm напишите программу вывода значения регистра eax (рис. 4.2)

```
%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. 4.2 Программа jmp

Создайте исполняемый файл и запустите его. В этом случае, когда отображается значение регистра `jmp`, результатом будет: сообщение 2, сообщение 3 (рис.4.3)

```
gkarmikhos@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
gkarmikhos@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
gkarmikhos@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1.asm
bash: ./lab7-1.asm: Permission denied
gkarmikhos@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
gkarmikhos@fedora: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

рис. 4.3 Запустить программу (2-3)

Изменить текст программы: добавляем `jmp _end` в `_label1`: и `jmp _label1` в `_label2` (рис. 4.4).

```
%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

^G Help      ^O Write Out  ^W Where Is   ^K C
^X Exit      ^R Read File  ^\ Replace    ^U P
```

рис. 4.4 Добавление новых инструкций в программу

При сохранении и запуске program мы получаем в результате сообщение 2 и сообщение3(рис.4.5)

```
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

рис. 4.5 Запустить программу(2,1)

Мы модифицируем программу так, чтобы при отображении она отображалась в порядке 3,2,1 (рис. 4.6)

```
/home/gkarmikhos/work/study/2024-2025/Архитектура комп
%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 sprintfLF ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintfLF ; 'Сообщение № 3'
jmp _label2

_end:
call quit ; вызов подпрограммы завершения
```

рис. 4.1.7 Изменяем программу

Запускаем программу, чтобы увидеть ее результат (рис 4.7)

```
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1  
Сообщение № 3  
Сообщение № 2  
Сообщение № 1  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

рис 4.7 Запустить программу

Создайте файл lab7-2.asm рис 4.8

```
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$  
touch lab7-2.asm  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

рис 4.8 Создайте файл lab7-2.asm

рис 4.9

```
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-2  
.asm  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o  
lab7-2 lab7-2.o  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2  
Введите В: 6  
Наибольшее число: 50  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2  
Введите В: 55  
Наибольшее число: 55  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2  
Введите В: 9  
Наибольшее число: 50  
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ █
```

рис 4.9

Копируем программу (рис 4.10)

```
gkarmikhos@fedora:~/work/study/2024-2025/Архитектура компьютера/ar...
lab7-2.lst [----] 22 L: [ 1+ 9 10/230] *(665 /14720b) 0032 0x020 [*]
1 %include 'in_out.asm'
2 <1> ;----- slen -----
3 <1> ; Функция вычисления длины сообщения
4 <1> slen:.....
5 <1> push ebx.....
6 <1> mov ebx, eax.....
7 <1>.....
8 <1> nextchar:.....
9 <1> cmp byte [eax], 0...
10 <1> jz finished.....
11 <1> inc eax.....
12 <1> jmp nextchar.....
13 <1>.....
14 <1> finished:
15 <1> sub eax, ebx
16 <1> pop ebx.....
17 <1> ret.....
18 <1>.....
19 <1> ;----- sprint -----
20 <1> ; Функция печати сообщения
21 <1> ; входные данные: mov eax,<message>
22 <1> sprint:
23 <1> push edx
24 <1> push ecx
25 <1> push ebx
26 <1> push eax
27 <1> call slen
28 <1>.....
29 <1> mov edx, eax
30 <1> pop eax
31 <1>.....
32 <1> mov ecx, eax
33 <1> mov ebx, 1
34 <1> mov eax, 4
35 <1> int 80h
36 <1>.....
37 <1> pop ebx
38 <1> pop ecx
39 <1> pop edx
40 <1> ret
41 <1>.....
42 <1>.....
43 <1> ;----- sprintLF -----
44 <1> ; Функция печати сообщения с переводом строки
```

рис 4.10 программа

5. Задание для самостоятельной работы

Написать программу вычисления выражения (рис 5.1) (5.2).

```
SECTION .data
    msg_result db "Resultado: ", 0
    msg_input db "Valor de x: ", 0
    newline db 0xA, 0
    res db 0 ; Espacio para almacenar el resultado

SECTION .bss
    x resb 4 ; Variable para almacenar x

SECTION .text
GLOBAL _start

_start:
    ; Imprimir mensaje para introducir x
    mov eax, msg_input
    call sprintf

    ; Leer valor de x
    call read_int
    mov [x], eax ; Guardar x

    ; Comparar x con 4
    mov eax, [x]
    cmp eax, 4
    jl less_than_4 ; Si x < 4, saltar a less_than_4
    jge greater_or_equal_4 ; Si x >= 4, saltar a greater_or_equal_4

less_than_4:
    ; Calcular x + b (b = 83)
    mov eax, [x]
    add eax, 83
    mov [res], eax ; Guardar el resultado
    jmp print_result ; Saltar para imprimir el resultado

greater_or_equal_4:
    ; Calcular a * x - c (a = 79, c = 41)
```

рис 5.1

SECTION .data

```
msg_result db "Resultado: ", 0
msg_input db "Valor de x: ", 0
res db 0      ; Espacio para almacenar el resultado
newline db 0xA, 0
```

SECTION .bss

```
x resb 4      ; Variable para almacenar x
a resb 4      ; Variable para a (constante calculada)
```

SECTION .text**GLOBAL _start****_start:**

```
; Imprimir mensaje para introducir x
mov eax, msg_input
call sprintf

; Leer valor de x
call read_int
mov [x], eax ; Guardar x

; Comparar x con 4
mov eax, [x]
cmp eax, 4
jl less_than_4 ; Si x < 4, saltar a less_than_4
jge greater_or_equal_4 ; Si x >= 4, saltar a greater_or_equal_4
```

less_than_4:

```
; Calcular x + 4
mov eax, [x]
add eax, 4
mov [res], eax ; Guardar el resultado
jmp print_result ; Saltar para imprimir el resultado
```

greater_or_equal_4:

```
; Calcular a * x (a = 1/7)
; Definimos a como 1/7 (en formato fijo o flotante para simplificar)
mov eax, [x]
mov ebx, 7 ; Divisor (a = 1/7)
div ebx ; eax = x / 7
mov [res], eax ; Guardar el resultado
```

print_result:

```
; Imprimir "Resultado: "
mov eax, msg_result
call sprintf
```

^G Help**^O** Write Out**^W** Where Is**^K** Cut**^T****^X** Exit**^R** Read File**^** Replace**^U** Paste**^J**

рис 5.2 Программа

6. Выводы

Использовали метки (`jmp`, `je`, `jl`, `jge`) для управления потоком программы и реализации условий.

Узнали, как считывать и выводить данные с помощью процедур `sprintf`, `read_int` и `print_int`.

Управляли секциями данных (`.data` и `.bss`) для хранения констант, переменных и промежуточных результатов.

Мы модифицировали программу с метками для изменения порядок вывода заранее определённых сообщений.

7. Список используемой литературы

- 1) Архитектура ЭВМ