РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: Архитектура компьютера

Команды безусловного и условного переходов в NASM. Программирование ветвлений

Студент: ТУЙИШИМЕ Тьерри

Группа: НКАбд-05-25

Оглавление

1. Цель работы

Изучение команд условного и безусловного переходов в ассемблере NASM. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

2. Теоретическая часть

2.1. Типы переходов

Безусловный переход: передача управления в определенную точку программы без проверки условий (инструкция `jmp`)

Условный переход: передача управления при выполнении определенного условия (инструкции `je`, `jne`, `jg`, `jl` и др.).

2.2. Регистр флагов

Флаги отражают состояние процессора после выполнения операций:

CF (Carry Flag) — флаг переноса

ZF (Zero Flag) — флаг нуля

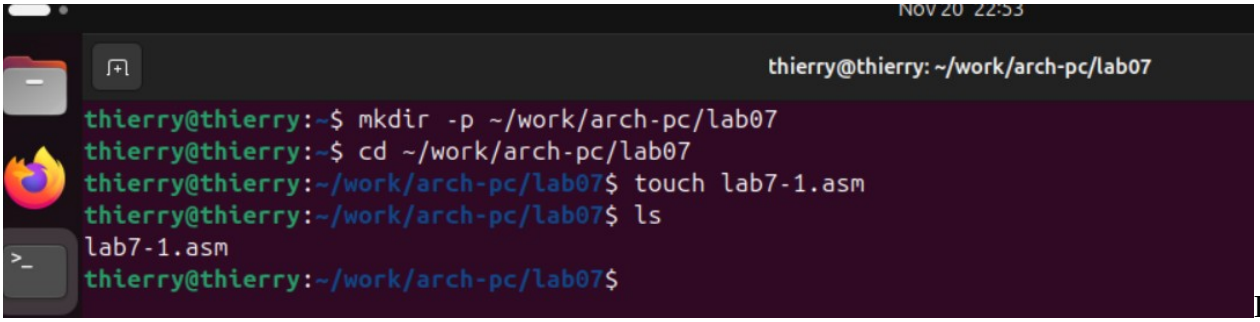
SF (Sign Flag) — флаг знака

OF (Overflow Flag) — флаг переполнения

3. Практическая часть

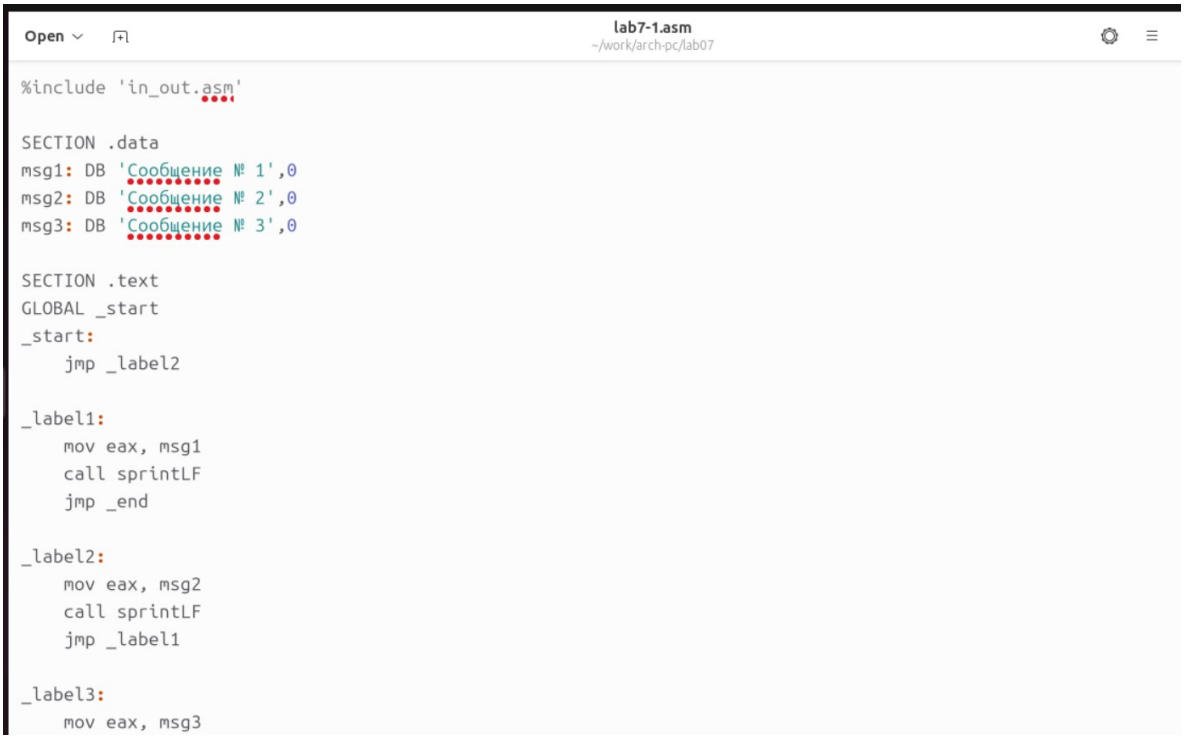
3.1. Задание 1: Работа с безусловными переходами

Шаг 1: Создание каталога и файла программы



Шаг 2: Написание программы с использованием `jmp`

Создайте файл `lab7-1.asm` со следующим содержимым:



Шаг 3: Трансляция и запуск программы

Результат выполнения:

- Сообщение № 2
- Сообщение № 1

Шаг 4: Изменение программы для другого порядка вывода

Измените программу так, чтобы вывод был:

- Сообщение № 3
- Сообщение № 2
- Сообщение № 1

Модифицированный код:

Open

lab7-1.asm
~/work/arch-pc/lab07

```
%include 'in_out.asm'

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
    call sprintLF
    jmp _end

_label2:
    mov eax, msg2
    call sprintLF
    jmp _label1

_label3:
    mov eax, msg3
```

3.2. Задание 2: Работа с условными переходами

Шаг 1: Создание файла программы

```
thierry@thierry:~/work/arch-pc/lab07$ touch lab7-2.asm
thierry@thierry:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
thierry@thierry:~/work/arch-pc/lab07$
```

Шаг 2: Написание программы для нахождения наибольшего числа

Создайте файл `lab7-2.asm`:

Open — lab7-2.asm — ~/work/arch-pc/lab07

```
%include 'in_out.asm'

section .data
msg1 db 'Введите B: ',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 B: '
    mov eax, msg1
    call sprint

    ; Ввод 'B'
    mov ecx, B
    mov edx, 10
    call sread
```

Шаг 3: Трансляция и тестирование программы

```
thierry@thierry:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
thierry@thierry:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
thierry@thierry:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 30
Наибольшее число: 50
thierry@thierry:~/work/arch-pc/lab07$
```

Тестирование для разных значений B:

- Тест 1: B = 30
- Введите B: 30

Наибольшее число: 50

```
thierry@thierry:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
thierry@thierry:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
thierry@thierry:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 30
Наибольшее число: 50
thierry@thierry:~/work/arch-pc/lab07$
```

- Тест 2: B = 60
- Введите B: 60

Наибольшее число: 60

```
thierry@thierry:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
thierry@thierry:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-2.o -o lab7-2
thierry@thierry:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 60
Наибольшее число: 60
thierry@thierry:~/work/arch-pc/lab07$
```

3.3. Задание 3: Изучение файла листинга

Шаг 1: Создание файла листинга

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

Шаг 2: Просмотр файла листинга

- cat lab7-2.lst

Пример содержимого файла листинга:

```
thierry@thierry:~/work/arch-pc/lab07$ cat lab7-2.lst
     1                                  %include 'in_out.asm'
     1                              <1> ; in_out.asm
     2                              <1> section .text
     3                              <1>
     4                              <1> ;------------------------------------
     5                              <1> ; _sread - чтение строки из stdin
     6                              <1> ; ARGS: ECX = buffer, EDX = buffer size
     7                              <1> ;------------------------------------
     8                              <1> _sread:
     9 00000000 50                  <1>     push    eax
    10 00000001 53                  <1>     push    ebx
    11 00000002 B803000000          <1>     mov     eax, 3      ; sys_read
    12 00000007 BB00000000          <1>     mov     ebx, 0      ; stdin
    13 0000000C CD80                <1>     int     0x80
    14 0000000E 5B                  <1>     pop     ebx
    15 0000000F 58                  <1>     pop     eax
    16 00000010 C3                  <1>     ret
    17                              <1>
    18                              <1> ;------------------------------------
    19                              <1> ; _swrite - запись строки в stdout
    20                              <1> ; ARGS: EAX = string pointer, EBX = string length
    21                              <1> ;------------------------------------
    22                              <1> _swrite:
    23 00000011 50                  <1>     push    eax
```

Шаг 3: Анализ структуры листинга

- **Номер строки:** порядковый номер строки в листинге
- **Адрес:** смещение машинного кода от начала сегмента
- **Машинный код:** шестнадцатеричное представление инструкции
- **Исходный текст:** текст программы на ассемблере

Шаг 4: Создание ошибки и анализ

Удалите один операнд в инструкции и выполните трансляцию:

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

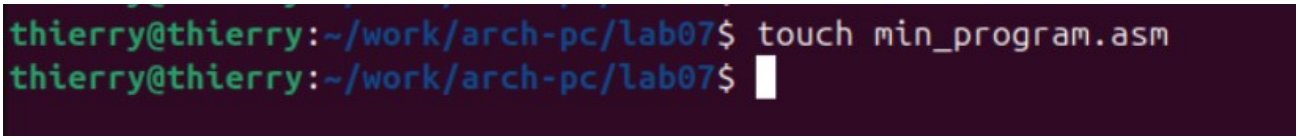
Результат: Транслятор выведет сообщение об ошибке, файл листинга не будет создан.

4. Задание для самостоятельной работы

4.1. Задание 1: Нахождение наименьшего из трех чисел

Вариант 1: a=17, b=23, c=45

Шаг 1: Создание файла программы



Шаг 2: Написание кода программы

Open

min_program.asm
~/work/arch-pc/lab07

```
%include 'in_out.asm'

section .data
msg db "Наименьшее число: ",0
a dd 17
b dd 23
c dd 45

section .bss
min resd 1

section .text
global _start
_start:
    ; Записываем a в min
    mov eax, [a]
    mov [min], eax

    ; Сравниваем min и b
    cmp eax, [b]
    jl check_c
    mov eax, [b]
    mov [min], eax
```

Шаг 3: Компиляция и выполнение

```
thierry@thierry:~/work/arch-pc/lab07$
thierry@thierry:~/work/arch-pc/lab07$ nasm -f elf min_program.asm
thierry@thierry:~/work/arch-pc/lab07$ ld -m elf_i386 min_program.o -o min_program
thierry@thierry:~/work/arch-pc/lab07$ ./min_program
Наименьшее число: 17
thierry@thierry:~/work/arch-pc/lab07$
```

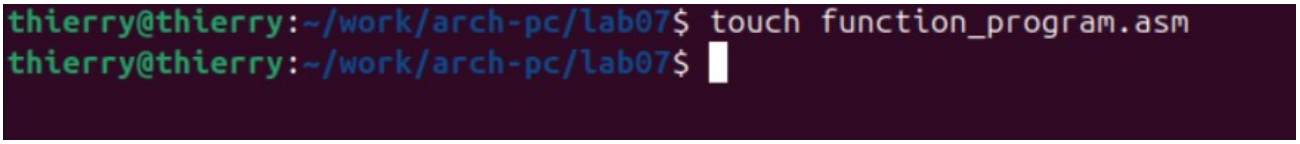
Результат выполнения:

Наименьшее число: 17

4.2. Задание 2: Вычисление функции f(x)

Тестовые значения: (x_1=1, a_1=2), (x_2=2, a_2=1)

Шаг 1: Создание файла программы



Шаг 2: Написание кода программы

Open

function_program.asm
~/work/arch-pc/lab07

```
%include 'in_out.asm'

section .data
msg_x db "Введите x: ",0
msg_a db "Введите a: ",0
msg_res db "Результат: ",0

section .bss
x resd 1
a resd 1

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

Шаг 3: Компиляция и тестирование

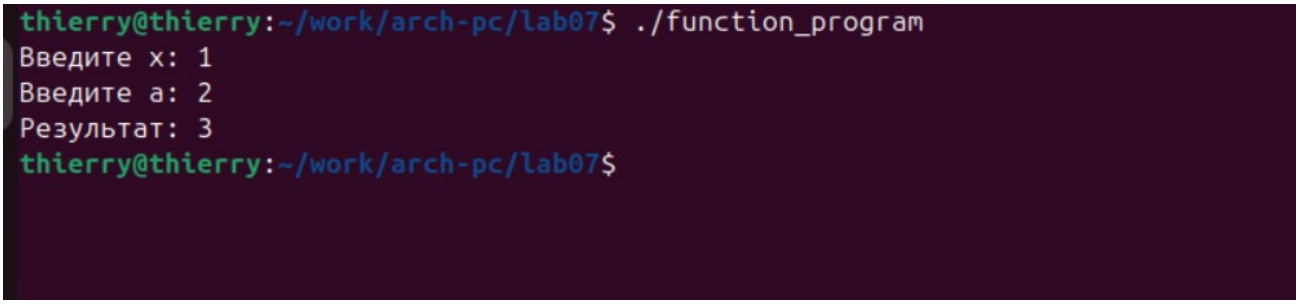
```
thierry@thierry:~/work/arch-pc/lab07$
thierry@thierry:~/work/arch-pc/lab07$ nasm -f elf function_program.asm
thierry@thierry:~/work/arch-pc/lab07$ ld -m elf_i386 function_program.o -o function_program
thierry@thierry:~/work/arch-pc/lab07$ ./function_program
```

Тестирование

Тест 1: x=1, a=2

- Введите x: 1
- Введите a: 2

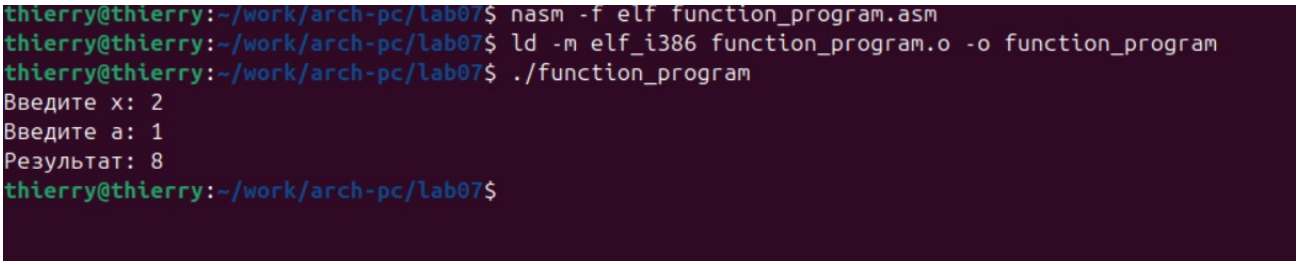
Результат: 3



Тест 2: x=2, a=1

- Введите x: 2
- Введите a: 1

Результат: 8



5. Выводы

1. Освоены команды переходов: изучены безусловные (`jmp`) и условные (`je`, `jne`, `jg`, `jl` и др.) переходы в ассемблере NASM.

2. Изучен регистр флагов: понимание работы флагов CF, ZF, SF, OF необходимо для корректного использования условных переходов.

3. Освоена инструкция `cmp`: команда сравнения является основой для организации условных переходов.

4. Изучен файл листинга: понимание структуры листинга помогает в отладке программ.

5. Приобретены практические навыки: написаны программы с использованием ветвлений для решения практических задач.

6. Освоена работа с функциями: использование внешних файлов (`in_out.asm`) для ввода-вывода данных.

6. Список литературы

1. Лабораторная работа №7 - Методические указания

2. NASM Documentation - Official NASM Manual

3. Assembly Language Programming - Fundamentals and Techniques