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: Ниемек Яи Жак

Группа: НММБд-04-24

МОСКВА

2025_ г.

Цель работы

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

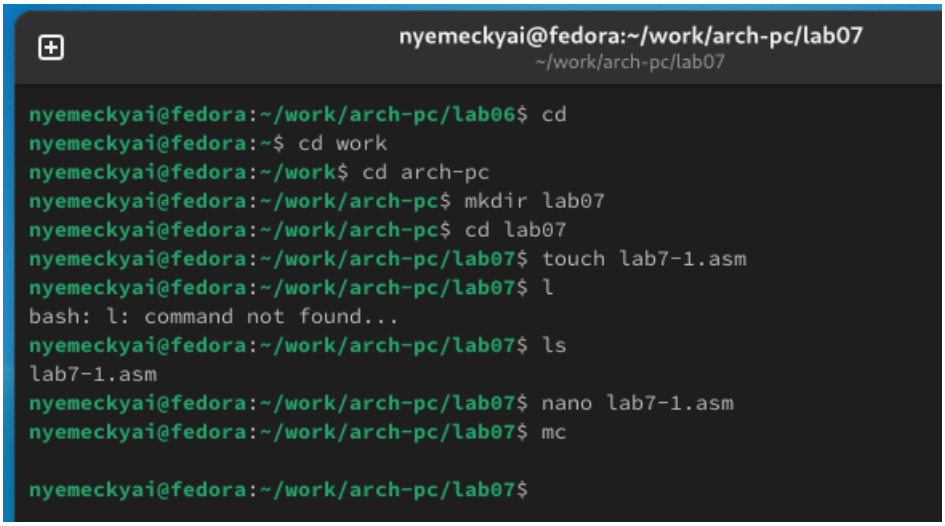
Задание

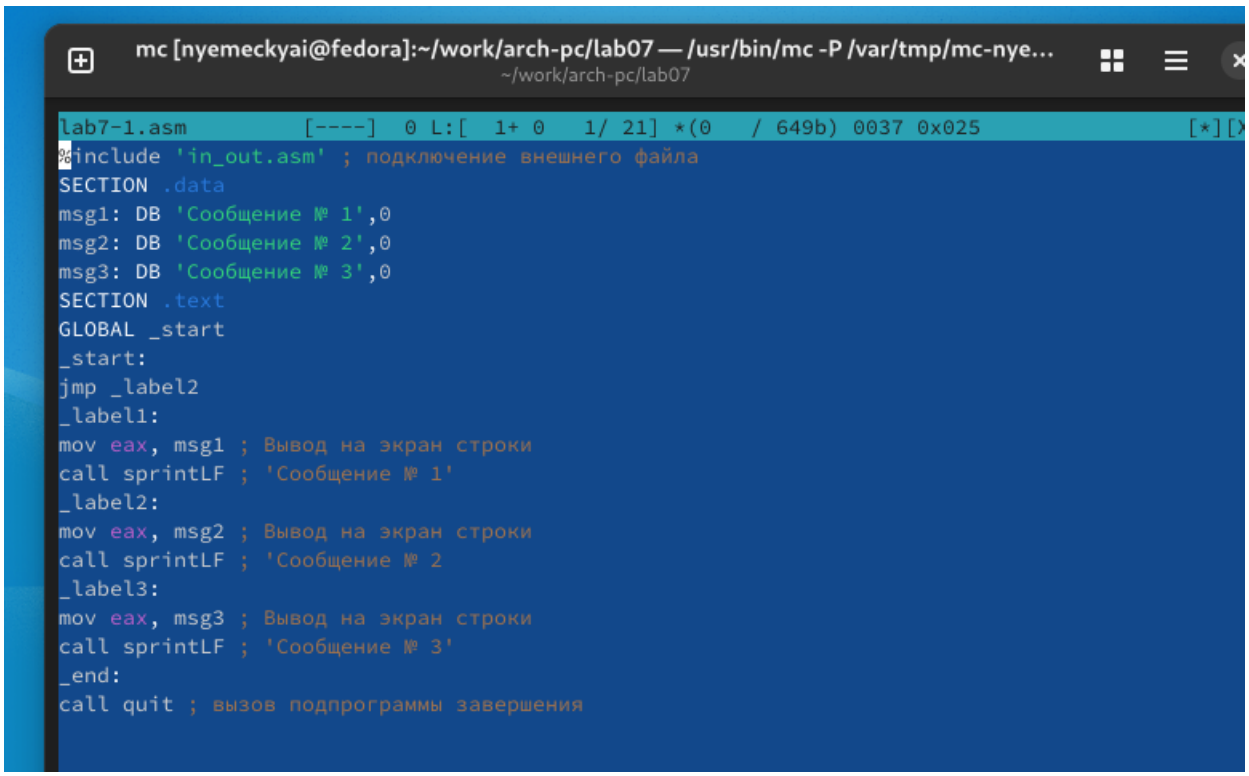
1. Создайте каталог для программам лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm
2. Инструкция jmp в NASM используется для реализации безусловных пере□ходов. Рассмотрим пример программы с использованием инструкции jmp.
Введите в файл lab7-1.asm текст программы из листинга 7.1.
3. Использование инструкции jmp приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: A,B и C. Значения для A и C задаются в программе, значение B вводиться с клавиатуры.
4. Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ -l и задав имя файла листинга в командной строке. Создайте файл листинга для программы из файла lab7-2.asm

Выполнение лабораторной работы

Выполнение лабораторной работы
1) Создаю каталог для программам лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm
Рис. 4.1: Создание каталога и файла
2) Ввожу в файл lab7-1.asm текст программы из листинга 7.1.
Рис. 4.2: Содержимое файла
8
3) Создаю исполняемый файл и запускаю его
Рис. 4.3: Работа файла
4) Изменяю текст программы в соответствии с листингом 7.2
Рис. 4.4: текст программы
5) Создаю исполняемый файл и запускаю его
Рис. 4.5: Работа файла
9
6) Изменяю текст программы изменив инструкции jmp, чтобы вывод про□граммы был следующим: Сообщение № 3 Сообщение № 2 Сообщение № 1
%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 sprintLF ; ‘Сообщение № 1’ jmp _end _label2: mov eax, msg2 ; Вывод на экран строки call sprintLF ; ‘Сообщение № 2’ jmp _label1 _label3: mov eax, msg3 ; Вывод на экран строки call sprintLF ; ‘Сообщение № 3’ jmp _label2 _end: call quit ; вызов подпрограммы завершения
Рис. 4.6: Текст программы
7) Создаю исполняемый файл и запускаю его
10
Рис. 4.7: Работа файла
8) Создаю файл lab7-2.asm и проверяю его создание
Рис. 4.8: Создание файла
9) Ввожу в файл текст листинга 7.3, создаю файл и запускаю его
Рис. 4.9: Работа файла
10) Создаю файл листинга для программы из файла lab7-2.asm и открываю его в текстовом редакторе
Рис. 4.10: Создание файла листинга
11) Открытый файл листинга
11
Рис. 4.11: Открытый файл листинга
17 000000F2 B9[0A000000] mov ecx,B (17 - номер строки, 000000F2 - адрес, B9 - машинный код, [0A000000] - исходный текст программы) 18 000000F7 BA0A000000 mov edx,10 (18 - номер строки, 000000F7 - адрес, BA - машинный код, 0A000000 - исходный текст программы) 19 000000FC E842FFFFFF call sread (19 - номер строки, 000000FC - адрес, E8 - машинный код, 42FFFFFF - исходный текст программы)
12) Копирую файл lab7-2.asm как lab7-2-2.asm и открываю его
Рис. 4.12: Копирование файла
13) Удаляю один из операндов
Рис. 4.13: Измененный текст программы
14) Создаю файл листинга





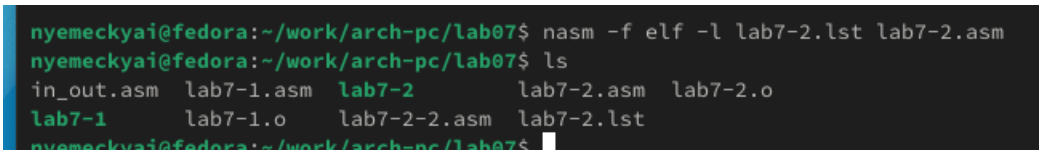

```
nyemeckyai@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
nyemeckyai@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
nyemeckyai@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
nyemeckyai@fedora:~/work/arch-pc/lab07$
```

```
nyemeckyai@fedora:~/work/arch-pc/lab07$ touch lab7-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab07$ nano lab7-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
nyemeckyai@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
nyemeckyai@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 35
Наибольшее число: 50
nyemeckyai@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 70
Наибольшее число: 70
nyemeckyai@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 30
Наибольшее число: 50
nyemeckyai@fedora:~/work/arch-pc/lab07$
```

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

```
lab7-2.lst          [----]  0 L:[  1+ 0   1/225] *(0   /14458b) 0032 0x020          [*][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
   26 00000012 50                   <1>     push    eax
   27 00000013 E8E8FFFFFF           <1>     call    slen
   28                               <1>.....
   29 00000018 89C2                 <1>     mov     edx, eax
1Help  2Save  3Mark  4Replac  5Copy  6Move  7Search  8Delete  9PullDn  10Quit
```

```
lab7-2-2.asm        [----]  0 L:[  1+ 0   1/ 50] *(0   /1744b) 0037 0
%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
; ---------- Преобразование 'B' 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 'A' и 'C' (как символы)
```



Ответы на вопросы для самопроверки

1. Для чего нужен файл листинга NASM? В чём его отличие от текста программы?

Файл листинга NASM используется для анализа скомпилированного кода и отладки программы. Он содержит как исходный код программы, так и соответствующие ему машинные инструкции и адреса.

Отличия от текста программы:

- Текст программы содержит только исходный код, написанный на ассемблере.
- Файл листинга включает скомпилированные инструкции в шестнадцатеричном представлении, что помогает отладке и анализу кода.

Создать листинг можно с помощью команды:

```
nasm -l program.lst program.asm
```

---

2. Каков формат файла листинга NASM? Из каких частей он состоит?

Файл листинга NASM состоит из следующих частей:

- **Адреса инструкций** – адреса в памяти, где будут находиться инструкции.
- **Машинный код** – шестнадцатеричное представление команд, которые процессор будет исполнять.
- **Исходный код** – команды, написанные программистом на ассемблере.

Пример строки из файла листинга:

```
00000000  B8 04 00 00 00  mov eax, 4
```

Здесь:

- `00000000` – адрес инструкции,
- `B8 04 00 00 00` – машинный код,
- `mov eax, 4` – исходный код.

---

3. Как в программах на ассемблере можно выполнить ветвление?

Ветвление выполняется с помощью команд условных (`JMP`, `JE`, `JNE`, `JG`, `JL`, и др.) и безусловных (`JMP`) переходов.
Пример ветвления с использованием `CMP` и `JNE`:

```
mov eax, 5
cmp eax, 3   ; Сравниваем EAX с 3
jne not_equal ; Переход, если не равно
mov ebx, 1   ; Если равно, выполнить эту строку
not_equal:
mov ebx, 0   ; Если не равно, выполнить эту строку
```

---

4. Какие существуют команды безусловного и условных переходов в языке ассемблера?

- **Безусловный переход:**
  - `JMP` – безусловный переход на указанную метку.
- **Условные переходы:**
  - `JE` (Jump if Equal) – переход, если равно.
  - `JNE` (Jump if Not Equal) – переход, если не равно.
  - `JG` (Jump if Greater) – переход, если больше.
  - `JL` (Jump if Less) – переход, если меньше.
  - `JGE` (Jump if Greater or Equal) – переход, если больше или равно.
  - `JLE` (Jump if Less or Equal) – переход, если меньше или равно.

---

5. Опишите работу команды сравнения `CMP`.

Команда `CMP` сравнивает два значения, вычитая одно из другого, но **не изменяя их**. Она только устанавливает флаги процессора, которые затем используются командами условных переходов.

Пример:

```
cmp eax, ebx  ; Сравнивает EAX и EBX
je equal      ; Переход к метке equal, если EAX == EBX
jne not_equal  ; Переход к метке not_equal, если EAX != EBX
```

Если `EAX == EBX`, устанавливается флаг ZF (Zero Flag), и выполняется `JE equal`.

---

6. Каков синтаксис команд условного перехода?

Общий синтаксис:

```
CMP <операнд1>, <операнд2>
J<условие> <метка>
```

Пример:

```
cmp eax, 5  ; Сравниваем EAX с 5
jl less     ; Переход к less, если EAX < 5
```

```
jg greater  ; Переход к greater, если EAX > 5
```

---

7. Приведите пример использования команды сравнения и команд условного перехода.

Пример программы, которая проверяет, является ли число положительным, отрицательным или нулём:

```
section .text
global _start

_start:
    mov eax, -5  ; Устанавливаем число для проверки

    cmp eax, 0
    je is_zero   ; Если равно 0, переход к метке is_zero
    jl is_negative  ; Если меньше 0, переход к is_negative
    jg is_positive  ; Если больше 0, переход к is_positive

is_zero:
    ; Действия, если число равно нулю
    jmp end

is_negative:
    ; Действия, если число отрицательное
    jmp end

is_positive:
    ; Действия, если число положительное

end:
    ; Завершение программы
```

---

8. Какие флаги анализируют команды безусловного перехода?

Безусловный переход (`JMP`) не анализирует флаги, но условные переходы анализируют **флаги процессора**:

- **ZF (Zero Flag)** – устанавливается, если результат сравнения равен 0.
- **SF (Sign Flag)** – устанавливается, если результат отрицательный.
- **CF (Carry Flag)** – указывает на заимствование при вычитании.
- **OF (Overflow Flag)** – устанавливается при арифметическом переполнении.

Пример анализа флагов:

```
cmp eax, ebx
je equal  ; Переход, если ZF = 1 (EAX == EBX)
jg greater  ; Переход, если SF = OF (EAX > EBX)
```

Флаги помогают процессору определять условия для выполнения различных команд перехода.

Выводы

Мною изумлены команды условного и безусловного переходов, приобретены навыки написания программ с использованием переходов, я ознакомилась с назначением и структурой файла листинга.