Лабораторная работа №7

Дисциплина: Архитектура компьютера

Мутаев Муртазаали Магомедович

Содержание

Список иллюстраций

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 Задание

1. Знакомство с неусловным переходом
2. Знакомство с условным переходом
3. Файлы листинга
4. Задания для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Знакомство с неусловным переходом

Первым делом я создал папку файл lab7-1.asm в каталоге work/arch-pc/lab07. В этот файл я вставил код из методички. (рис. 3.1).

```
%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 _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. 3.1: Программа с использованием инструкции jmp

Далее я создал исполняемый файл и запустил программу. Вот результат, который она мне выдала (рис. 3.2).

```
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение № 2
Сообщение № 3
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $
```

Рис. 3.2: Результат программы с ипользованием jmp

После, чуть отредактировав код, чтобы “прыгнуть” назад по программе, я получил следующий результат (рис. 3.4).

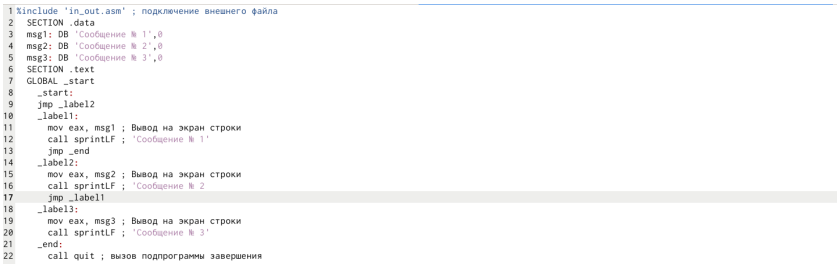


Рис. 3.3: jmp для “прыжка” назад

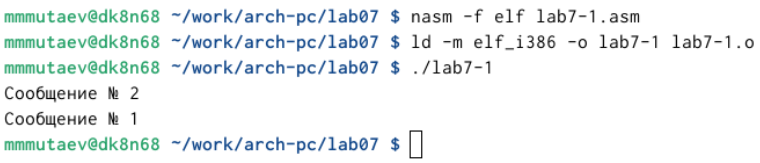


Рис. 3.4: Результат jmp для “прыжка” назад

Попробуем теперь самостоятельно отредактировать код так, чтобы программа вывела сообщения в обратном порядке. Для этого я написал следующий код (рис. 3.5).

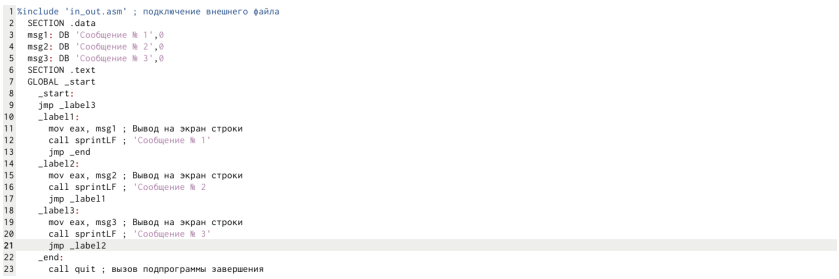


Рис. 3.5: Программа для вывода сообщений в обратном порядке

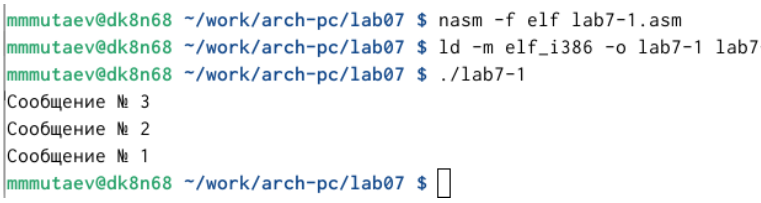


Рис. 3.6: Результат программы для вывода сообщений в обратном порядке

3.2 Знакомство с условным переходом

Я создал новый файл lab7-2.asm и закинул туда код из листинга 7.3

```
%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)
  cmp ecx,[C] ; Сравниваем 'A' и 'C'
  jg check_B ; если 'A>C', то переход на метку 'check_B',
  mov ecx,[C] ; иначе 'ecx = C'
  mov [max],ecx ; 'max = C'
; ---------- Преобразование 'max(A,C)' из символа в число
```

Рис. 3.7: Программа для условного перехода 1

```
  check_B:
  mov eax,max
  call atoi ; Вызов подпрограммы перевода символа в число
  mov [max],eax ; запись преобразованного числа в `max`
; ---------- Сравниваем 'max(A,C)' и 'B' (как числа)
  mov ecx,[max]
  cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
  jg fin ; если 'max(A,C)>B', то переход на 'fin',
  mov ecx,[B] ; иначе 'ecx = B'
  mov [max],ecx
; ---------- Вывод результата
fin:
  mov eax, msg2
  call sprint ; Вывод сообщения 'Наибольшее число: '
  mov eax,[max]
  call iprintLF ; Вывод 'max(A,B,C)'
  call quit ; Выход
```

Рис. 3.8: Программа для условного перехода 2

Программа выдавала такой результат (рис. 3.9).

```
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ touch lab7-2.asm
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ gedit lab7-2.asm
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 1
Наибольшее число: 50
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 52
Наибольшее число: 52
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 21
Наибольшее число: 50
mmmutaev@dk8n68 ~/work/arch-pc/lab07 $
```

Рис. 3.9: Результат программы с условным переходом

Программа выявляла максимальное число из A, B и C, где B вводится с клавиатуры

3.3 Файлы листинга

Я создал листинг асм файла с помощью команды nasm -f elf -l lab7-2.lst lab7-2.asm и зашел туда (рис. 3.10).

```
1     1                                 %include 'in_out.asm'
2     1                               <1> ;---------------   slen  -------------------
3     2                               <1> ; Функция вычисления длины сообщения
4     3                               <1> slen:
5     4 00000000 53                   <1>     push    ebx
6     5 00000001 89C3                 <1>     mov     ebx, eax
7     6                               <1>
8     7                               <1> nextchar:
9     8 00000003 803800               <1>     cmp     byte [eax], 0
10    9 00000006 7403                 <1>     jz      finished
11   10 00000008 40                   <1>     inc     eax
12   11 00000009 EBF8                 <1>     jmp     nextchar
13   12                               <1>
14   13                               <1> finished:
15   14 0000000B 29D8                 <1>     sub     eax, ebx
16   15 0000000D 5B                   <1>     pop     ebx
17   16 0000000E C3                   <1>     ret
18   17                               <1>
19   18                               <1>
20   19                               <1> ;---------------  sprint  -------------------
21   20                               <1> ; Функция печати сообщения
22   21                               <1> ; входные данные: mov eax,<message>
23   22                               <1> sprint:
24   23 0000000F 52                   <1>     push    edx
25   24 00000010 51                   <1>     push    ecx
26   25 00000011 53                   <1>     push    ebx
27   26 00000012 50                   <1>     push    eax
28   27 00000013 E8E8FFFFFF           <1>     call    slen
29   28                               <1>
30   29 00000018 89C2                 <1>     mov     edx, eax
31   30 0000001A 58                   <1>     pop     eax
32   31                               <1>
```

Рис. 3.10: Изучение структуры листинга

Мне нужно объяснить содержимое 3 любых строк в файле. Воспользуемся рандомайзером (рис. 3.11)

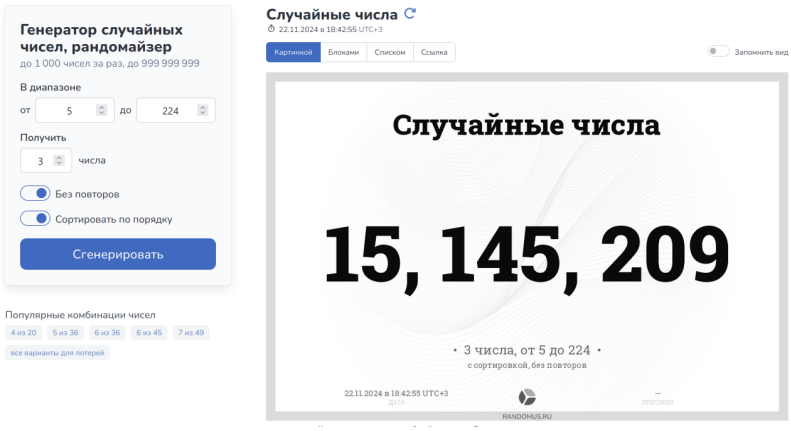


Рис. 3.11: Рандомайзер

Приступим к анализу: во всех строках понятно, что первое число - это номер строки листинга, второе - смещение машинного кода от начала текущего сегмента, 3 число - машинный код, представляющий собой ассемблированную исходную строку в виде шестнадцатеричной последовательности, а 4 блок - текст кода. Рассмотрим именно его

Рис. 3.12: Анализ строки 1

sub eax, ebx - это команда для операции eax - ebx, результат которой запишется в eax

Рис. 3.13: Анализ строки 2

add eax, ebx - это команда для операции eax + ebx, результат которой запишется в eax

Рис. 3.14: Анализ строки 3

mov eax, max - это команда для присвоения eax значения max

Теперь попробуем из любой инструкции с 2 операндами удалить 1 из них. Так я удалил операнд B из строки 21 (рис. 3.15):

Рис. 3.15: Удаление операнда из инструкции

Как и ожидалось, у меня просто не получилось создать файл листинга (рис. 3.16):

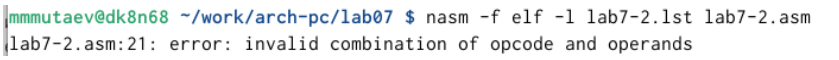


Рис. 3.16: Результат удаления операнда из инструкции

3.4 Задания для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных *a, b* и *c*. Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу

Я написал вот такой код (рис. 3.17):

```
msg1 db "Наименьшее число: ",0h
a dd 95
b dd 2
c dd 61
section .bss
min resb 10
section .text
global _start
_start:

  mov eax, [a]
  mov [min], eax

  mov ebx, [b]
  mov ecx, [c]

  cmp eax, ebx
  jl _A
  mov [min], ebx

_A:
  mov ebx, [min]
  cmp ebx, ecx
  jl fin
  mov [min], ecx

fin:
  mov eax, msg1
  call sprint
  mov eax, [min]
  call iprintLF
  call quit
```

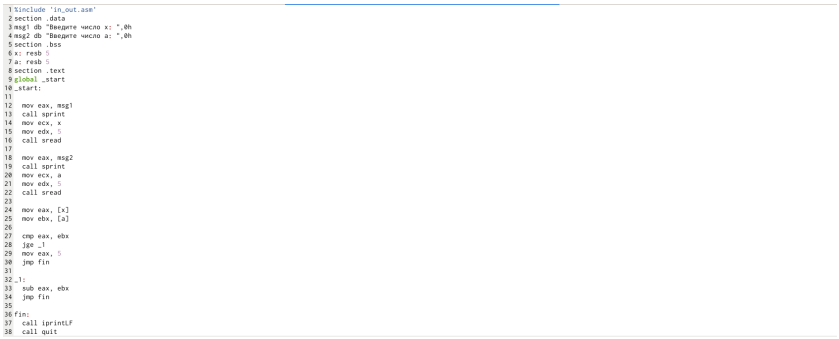
Рис. 3.17: СР 1

```
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ ./lab7-3
Наименьшее число: 2
```

Рис. 3.18: Результат СР 1

1. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции f(x) и выводит результат вычислений. Вид функции f(x) выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6

Я написал следующий код (рис. 3.19):



```
%include 'in_out.asm'
section .data
msg1 db "Введите число x: ",0h
msg2 db "Введите число a: ",0h
section .bss
x: resb 5
a: resb 5
section .text
global _start
_start:

  mov eax, msg1
  call sprint
  mov ecx, x
  mov edx, 5
  call sread

  mov eax, msg2
  call sprint
  mov ecx, a
  mov edx, 5
  call sread

  mov eax, [x]
  mov ebx, [a]

  cmp eax, ebx
  jge _!
  mov eax, 5
  jmp fin

_!:
  sub eax, ebx
  jmp fin

fin:
  call iprintLF
  call quit
```

Рис. 3.19: СР 2

```
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ ./lab7-4
Введите число x: 2
Введите число a: 1
1
mmmutaev@dk2n26 ~/work/arch-pc/lab07 $ ./lab7-4
Введите число x: 1
Введите число a: 2
5
```

Рис. 3.20: Результат СР 2

4 Выводы

Я изучил команды условного и безусловного переходов и приобрел навыки написания программ с использованием переходов, а также познакомился с назначением и структурой файла листинга