ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №8

дисциплина: Архитектура компьютера

Лихтенштейн Алина Алексеевна

Содержание

Список иллюстраций

1 Цель работы

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

2 Выполнение лабораторной работы

Создадим каталог для программ лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm. (рис. 2.1)

```
aaliechtenstein@rudn:~$ mkdir ~/work/arch-pc/lab08
aaliechtenstein@rudn:~$ cd ~/work/arch-pc/lab08
aaliechtenstein@rudn:~/work/arch-pc/lab08$ touch lab8-1.asm
```

Рис. 2.1: Создание каталога лабораторной программы и первого текстового файла языка ассемблера

Рассмотрим пример программы с использованием инструкции jmp. Введем в файл lab8-1.asm текст программы из листинга 8.1. (рис. 2.2)

```
/home/aaliechtenstein/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
        mov eax, msg1 ; Вывод на экран строки
        call sprintLF ; 'Сообщение No 1'

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

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

_end:
        call quit ; вызов подпрограммы завершения
```

Рис. 2.2: Текст программы из листинга 8.1.

Создадим исполняемый файл и запустим его. Результат работы данной программы будет следующим: (рис. 2.3)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-1
Сообщение No 2
Сообщение No 3
aaliechtenstein@rudn:~/work/arch-pc/lab08$
```

Рис. 2.3: Создание исполняемого файла. Запуск исполняемеого файла. Вывод программы

Изменим текст программы в соответствии с листингом 8.2. (рис. 2.4)

```
/home/aaliechtenstein/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end

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

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

_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.4: Текст программы из листинга 8.2.

Создадим исполняемый файл и проверим его работу. (рис. 2.5)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-1
Сообщение No 2
Сообщение No 1
aaliechtenstein@rudn:~/work/arch-pc/lab08$
```

Рис. 2.5: Создание исполняемого файла. Запуск исполняемеого файла. Вывод программы

Изменим текст программы, добавив инструкции jmp, чтобы вывод программы был следующим: (рис. 2.6, 2.7)

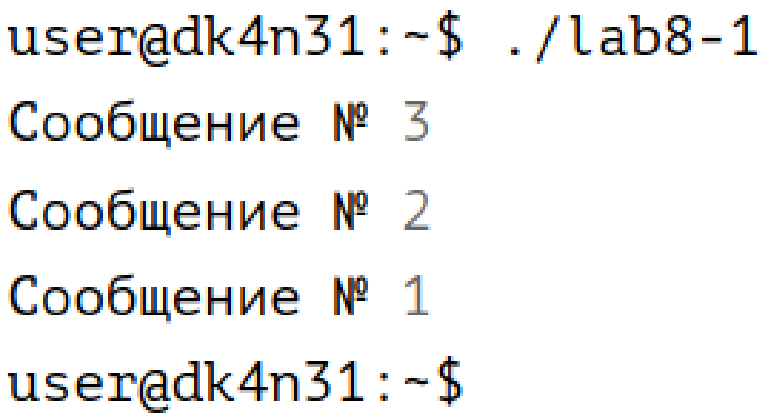

Рис. 2.6: Необходимый результат исполнения программы

```
/home/aaliechtenstein/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm' ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label3

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintLF ; 'Сообщение No 1'
jmp _end

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

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

_end:
call quit ; вызов подпрограммы завершения
```

Рис. 2.7: Отредактированный текст программы

Создадим исполняемый файл и проверим его работу. (рис. 2.8)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
aaliechtenstein@rudn:~/work/arch-pc/lab08$
```

Рис. 2.8: Создание исполняемого файла. Запуск исполняемеого файла. Вывод программы

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08. (рис. 2.9)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 2.9: Создание нового текстового файла .asm

Внимательно изучив текст программы из листинга 8.3, введем его в lab8-2.asm. (рис. 2.10)

```
/home/aaliechtenstein/work/arch-pc/lab08/lab8-2.asm
%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
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. 2.10: Текст программы из листинга 8.3.

Создадим исполняемый файл и проверим его работу для разных значений B. (рис. 2.11)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-2
Введите B: 10
Наибольшее число: 50
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-2
Введите B: 35
Наибольшее число: 50
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./lab8-2
Введите B: 60
Наибольшее число: 60
aaliechtenstein@rudn:~/work/arch-pc/lab08$
```

Рис. 2.11: Создание исполняемого файла. Запуск исполняемеого файла. Вывод программы

Обычно nasm создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ -l и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла lab8-2.asm. (рис. 2.12)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf -l lab8-2.lst lab8-2.asm
```

Рис. 2.12: Создание файла листинга программы

Откроем файл листинга lab8-2.lst с помощью текстового редактора mcedit. (рис. 2.13, 2.14)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ mcedit lab8-2.lst
```

Рис. 2.13: Команда для открытия файла в текстовом редакторе mcedit

```
/home/aaliechtenstein/work/arch-pc/lab08/lab8-2.lst   [----] 25 L:[  1+ 0   1/225] *(25  /14495b) 0032 0x020        [*][X]
     1                                  %include 'in_out.asm'
     2                          <1> ;---------------   slen  -------------------
     3                          <1> ; Функция вычисления длины сообщения
     4                          <1> slen:.....................
     5 00000000 53              <1>     push    ebx..........
     6 00000001 89C3            <1>     mov     ebx, eax.....
     7                          <1>.....
     8                          <1> nextchar:.................
     9 00000003 803800          <1>     cmp     byte [eax], 0...
    10 00000006 7403            <1>     jz      finished.....
    11 00000008 40              <1>     inc     eax..........
    12 00000009 EBF8            <1>     jmp     nextchar.....
    13                          <1>.....
    14                          <1> finished:
    15 0000000B 29D8            <1>     sub     eax, ebx
    16 0000000D 5B              <1>     pop     ebx..........
    17 0000000E C3              <1>     ret......................
    18                          <1>.
    19                          <1>.
    20                          <1> ;---------------  sprint  -------------------
    21                          <1> ; Функция печати сообщения
    22                          <1> ; входные данные: mov eax,<message>
    23                          <1> sprint:
    24 0000000F 52              <1>     push    edx
    25 00000010 51              <1>     push    ecx
    26 00000011 53              <1>     push    ebx
    27 00000012 50              <1>     push    eax
    28 00000013 E8E8FFFFFF      <1>     call    slen
    29                          <1>.....
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перем~тить 7Поиск 8Удалить 9МенюMC 10Выход
```

Рис. 2.14: Файл листинга в текстовом редакторе mcedit

Внимательно ознакомимся с его форматом и содержимым. Подробно объясним содержимое трёх строк файла листинга по выбору. (рис. 2.15)

Рис. 2.15: Три строки из файла листинга

Инструкция mov ebx, 0 начинается по смещению 00000045 в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция mov ebx, 0 ассемблируется в BB00000000 (в шестнадцатеричном представлении);

Инструкция mov eax, 3 начинается по смещению 0000004A в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция int 80h ассемблируется в B803000000;

Инструкция int 80h начинается по смещению 0000004F в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция int 80h ассемблируется в CD80; CD80 — это инструкция на машинном языке, вызывающая прерывание ядра);

Откроем файл с программой lab8-2.asm и в любой инструкции с двумя операндами удалим один операнд. (рис. 2.16, 2.17)

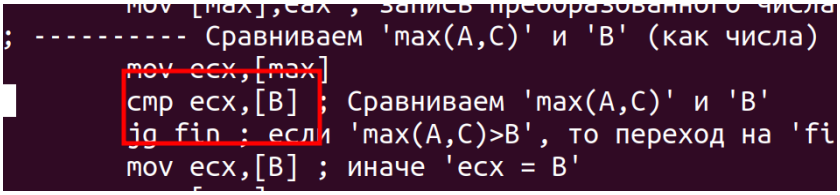

Рис. 2.16: Операнд [B] не удален

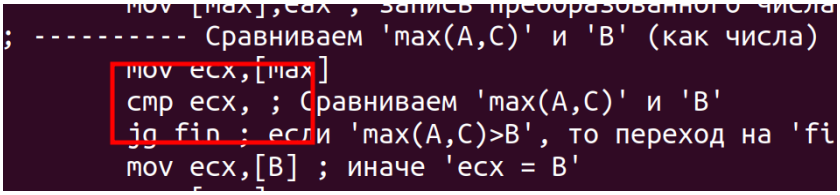

Рис. 2.17: Операнд [B] удален

Выполним трансляцию с получением файла листинга. (рис. 2.18)

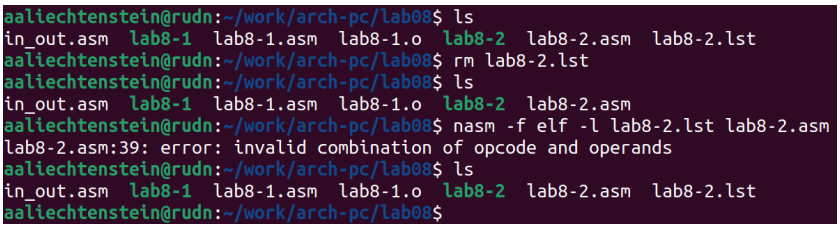

Рис. 2.18: Трансляция файла для получения файла листинга

Компилятор выводит ошибку: error: invalid combination of opcode and operands. Создается такой же файл листинга, но только с удаленным операндом. (рис. 2.19)

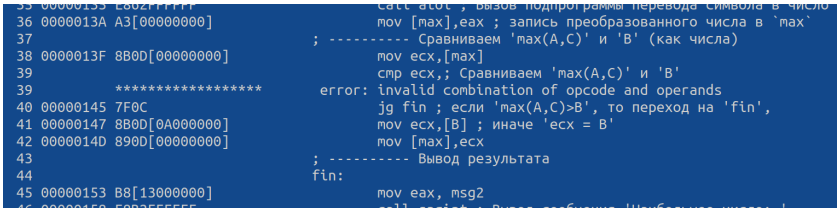

Рис. 2.19: Текст файла листинга с ошибкой (удален операнд)

3 Выполнение заданий для самостоятельной работы

1. Напишем программу нахождения наименьшей из 3 целочисленных переменных a, b и c (Вариант 14: a = 81, b = 22, c = 72). (рис. 3.1, 3.2, 3.3)

Номер варианта	Значения a, b, c
14	81,22,72

Рис. 3.1: Вариант 14

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ touch task8-1.asm
```

Рис. 3.2: Создание текстового файла для выполнения первого задания

```
/home/aaliechtenstein/work/arch-pc/lab08/task8-1.asm
%include 'in_out.asm'
section .data

        msg db "Наименьшее число: ",0h

        A dd 81
        B dd 22
        C dd 72

section .bss
        min resb 10


section .text
        global _start
_start:

; ---------- Записываем 'A' в переменную 'min'
        mov ecx, [A] ; 'ecx = A'
        mov [min], ecx ; 'min = A'
; ---------- Сравниваем 'A' и 'C' (как символы)
        cmp ecx, [C] ; Сравниваем 'A' и 'C'
        jl check_B ; если 'A<C', то переход на метку 'check_B',
        mov ecx,[C] ; иначе 'ecx = C'
        mov [min],ecx ; 'min = C'
; ---------- Преобразование 'min(A,C)' из символа в число
check_B:
        mov eax, [min]
        mov [min],eax ; запись преобразованного числа в `min`
; ---------- Сравниваем 'min(A,C)' и 'B' (как числа)
        mov ecx,[min]
        cmp ecx,[B]; Сравниваем 'min(A,C)' и 'B'
        jl       fin ; если 'min(A,C)<B', то переход на 'fin',
        mov ecx,[B] ; иначе 'ecx = B'
        mov [min],ecx
; ---------- Вывод результата
fin:
        mov eax, msg
        call sprint ; Вывод сообщения 'Наименьшее число: '
        mov eax,[min]
        call iprintLF ; Вывод 'min(A,B,C)'
        call quit ; Выход
```

Рис. 3.3: Текст программы первого задания

Создадим исполняемый файл и проверим его работу. (рис. 3.4)

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ nasm -f elf task8-1.asm
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o task8-1 task8-1.o
aaliechtenstein@rudn:~/work/arch-pc/lab08$ ./task8-1
Наименьшее число: 22
```

Рис. 3.4: Результат выполнения первой программы

Вывод программы правильный.

2. Напишем программу, которая для введенных с клавиатуры значений x и a

вычисляет значение заданной функции f(x) и выводит результат вычислений. (рис. 3.5, 3.6, 3.7)

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
14	$\begin{cases} 3a+1, & x < a \\ 3x+1, & x \geq a \end{cases}$	(2;3)	(4;2)

Рис. 3.5: Вариант 14

```
aaliechtenstein@rudn:~/work/arch-pc/lab08$ touch task-2.asm
```

Рис. 3.6: Создание текстового файла для выполнения второго задания

```
/home/aaliechtenstein/work/arch-pc/lab08/task8-2.asm
%include 'in_out.asm'
section .data

        msg1 db "3*a + 1, x < a", 0h
        msg2 db "3*x + 1, x >= a", 0h
        msg3 db "Введите x: ", 0h
        msg4 db "Введите a: ", 0h
        msg5 db "Результат: ", 0h

section .bss

        min resb 10
        x resb 10
        a resb 10
        res resb 10

section .text
        global _start
_start:

        mov eax, msg1 ; first condition
        call sprintLF

        mov eax, msg2 ; second condition
        call sprintLF

        mov eax, msg3 ; x
        call sprint

        mov ecx, x
        mov edx, 10
        call sread

        mov eax, x
        call atoi
        mov [x], eax

        mov eax, msg4 ; a
        call sprint

        mov ecx, a
        mov edx, 10
        call sread

        mov eax, a
        call atoi
        mov [a], eax

        mov ecx, [x]
        cmp ecx, [a]
        jge second_condition

first_condition:
        mov eax, 3
        mov ebx, [a]
        imul ebx
        add eax, 1
        mov [res], eax
        jmp fin

second_condition:
        mov eax, 3
        mov ebx, [x]
        imul ebx
        add eax, 1
        mov [res], eax
        jmp fin

fin:
        mov eax, msg5
        call sprint
        mov eax, [res]
        call iprintLF
        call quit
```

Рис. 3.7: Текст программы второго задания

Создайте исполняемый файл и проверьте его работу для значений x и a (Вариант 14: {x = 2, a = 3}, {x = 4, a = 2}). (рис. 3.8)

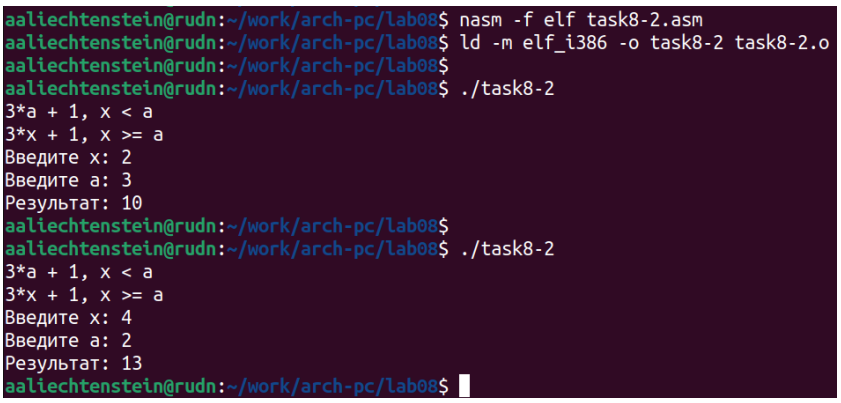

Рис. 3.8: Результат выполнения второй программы

4 Выводы

В процессе выполнения лабораторной работы были изучены команды условного и безусловного переходов, назначение и структура файла листинга. Приобретены навыки написания программ с использованием переходов.