Лабораторная работа номер 7.

Команды безусловного и условного переходов в Nasm. Программирование ветвлений.

Сорокин Кирилл

Содержание

Список иллюстраций

1 Цель работы

Научиться писать научиться писать программы на языке ассемблера с использованием условного и безусловного перехода.

2 Задание

Изучить приведённый материал на практике и выполнить самостоятельную работу.

3 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

4 Выполнение лабораторной работы

Создадим необходимые для работы директории и файлы (рис. 4.1).

```
kvsorokin@dk8n59 ~ $ mkdir ~/work/arch-pc/lab07
kvsorokin@dk8n59 ~ $ cd ~/work/arch-pc/lab07
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ touch lab7-1.asm
```

Рис. 4.1: Создание файлов и директорий

Откроем файл lab7-1.asm и введём в него текст программы(рис. 4.2).

```
%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. 4.2: Текст первой программы

После компиляции файлов запустим программу и увидим следующих раезультат(рис. 4.3).

```
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 3
```

Рис. 4.3: Выполнение программы lab7-1

Изменим текст программы так, чтобы она выводила сначала Сообщение 2, потом Сообщение 1(рис. 4.4).

```
%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
```

Рис. 4.4: Видоизменённый текст программы 1

Удостоверимся, в верности выполнения программы(рис. 4.5).

```
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
```

Рис. 4.5: Выполнение изменённой программы 1

Теперь ещё раз изменим программу, чтобы она выводила сообщения в последовательности 3, 2, 1. Для этого в конце каждого блока напишем переход на нужный блок(рис. 4.6).

```
%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
```

Рис. 4.6: Видоизменённый текст программы 2

Удостоверимся, в верности выполнения программы(рис. 4.7).

```
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ gedit lab7-1.asm
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-1 lab7-1.o
kvsorokin@dk8n59 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 3
Сообщение No 2
Сообщение No 1
```

Рис. 4.7: Выполнение изменённой программы 2

Создадим файл lab7-2.asm и впишем в него текст нахождения наибольшего числа из двух данных и одного введённого. (рис. 4.8).

```
%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
```

Рис. 4.8: Текст программы lab7-2

Проверим работу программы для разных значений B.(рис. 4.9).

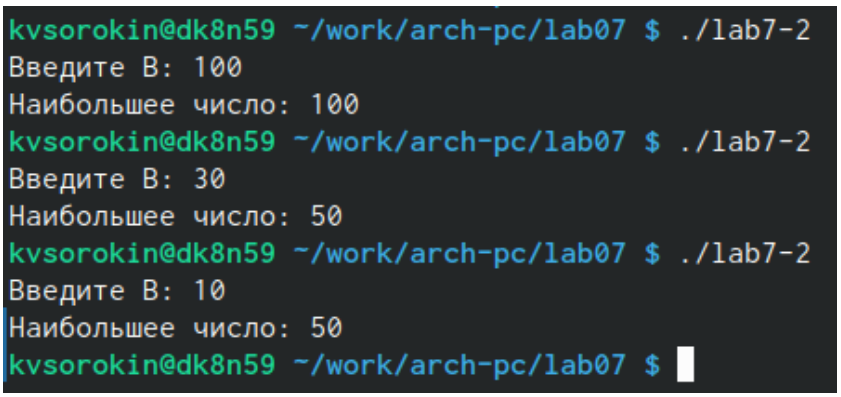

Рис. 4.9: Работа программы lab7-2

Командой ‘nasm -f elf -l lab7-2.lst lab7-2.asm’, создадим файл листинга и изучим его содержимое. Выберем 3 случайные строчки. На них мы осуществляется ввод значения для переменной B. НА строке 17 мы записываем адрес переменной в eax. На строке 18 передаём описатель 10 в edx. А на 19-ой строке мы с использованием полученных данных осуществляем ввод данных в ecx, который в данной ситуации отсылается к B.(рис. 4.10).

Рис. 4.10: Три строчки файла lab7-2.asm

Удалим один операнд у инструкции с двумя операндами. После попытки выполнениея программы увидим ошибку. Заново создав файл листинга на месте удаления операнда увидим ту самую ошибку. (рис. 4.11).

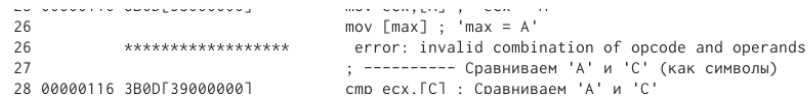

Рис. 4.11: Ошибка в файле листинга

4.1 Самостоятельная работа

Так как в предыдущей самостоятельной работе у нас был вариант 1, то, пользуясь полученными заниями, напишем программу, которая будет сравнивать три числа: 17, 23, 45. (рис. 4.12).

```
%include 'in_out.asm'

section .data
    msg1 db "Наименьшее число:",0h
    a dd 12
    b dd 23
    c dd 45

section .bss
    min resb 10

section .text
global _start

_start:
    mov eax, msg1
    call sprint

    mov ecx, [a]
    mov [min], ecx
    cmp ecx, [c]
    jl check_B
    mov ecx, [c]
    mov [min], ecx

check_B:

    mov ecx, [min]
    cmp ecx, [b]
    jl fin
    mov ecx, [b]
    mov [min], ecx

fin:
    mov eax, [min]
    call iprintLF ; Вывод 'min(A,B,C)'
    call quit ; Выход
```

Рис. 4.12: Текст программы lab7s-1

Скомпилируем программу и после запуска убедимся, что выводится именно наименьшее число - 17. (рис. 4.13).

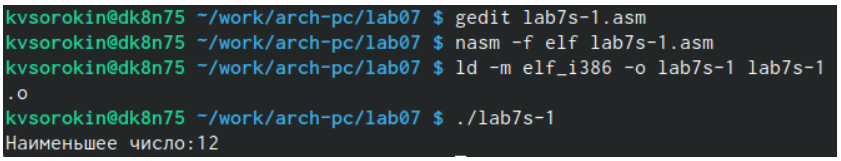

Рис. 4.13: Выполнение программы lab7s-1

Напишем вторую программу, которая принимает два значения, и в случае если значения числа A больше числа X, то выводит результат такого выражения: 2a-x,иначе выводит число 8. (к сожалению, из-за размера программы она полностью на поместилась на скиншот) (рис. **??**).

```
%include 'in_out.asm'

SECTION .data
msg1 db "Введите X:",0h
msg2 db "Ввведите A:",0h

SECTION .bss
max resb 10
x resb 10
a resb 10

SECTION .text
GLOBAL _start

_start:
        mov eax,msg1
        call sprint

        mov ecx,x
        mov edx,10
        call sread

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

        mov eax,msg2
        call sprint

        mov ecx,a
        mov edx,10
        call sread

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

        mov ebx, [x]
        cmp [a], ebx
        jl check

        mov eax, [a]
        mov edx, 2
        mul edx
```

Выполним программу с использованием данных в варинте значений ((1,2) и (2,1)), и будимся, что они верное удовлетворяют условиям выражения. (рис. 4.14).

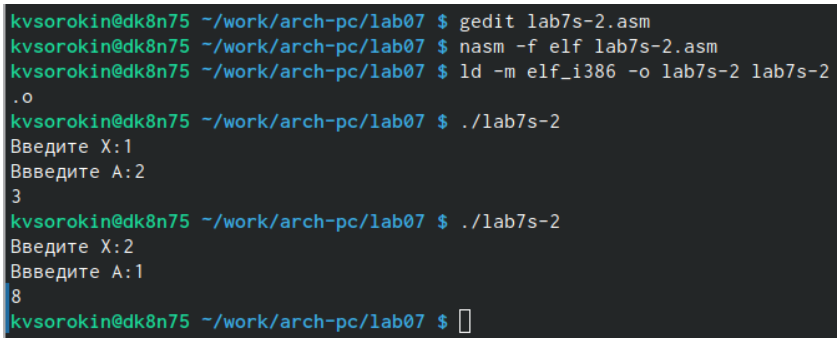

Рис. 4.14: Выполнение программы lab7s-2

5 Выводы

Мы научились писать программы на языке ассемблера с использованием различных переходов.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.
2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.
3. Midnight Commander Development Center. — 2021. — URL: https://midnightcommander.org/.
4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com Learning-bash-Shell-Programming-Nutshell/dp/0596009658.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. :Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. -

874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. -СПб. : Питер,
17. — 1120 с. — (Классика Computer Science)