

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров

Гафоров Нурмухаммад Вомикович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание файлов для лабораторной работы	9
4.2	Ввод текста из листинга 8.1	9
4.3	Запуск исполняемого файла	10
4.4	Изменение текста программы	10
4.5	Запуск обновленной программы	11
4.6	Изменение текста программы	11
4.7	Запуск исполняемого файла	12
4.8	Ввод текста программы из листинга 8.2	12
4.9	Запуск исполняемого файла	12
4.10	Ввод текста программы из листинга 8.3	13
4.11	Запуск исполняемого файла	13
4.12	Изменение текста прорааммы	14
4.13	Запуск исполняемого файла	14
4.14	Текст Программы	15
4.15	Запуск исполняемого файла и проверка его работу	15

Список таблиц

1 Цель работы

Приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

1. Реализация циклов в NASM.
2. Обработка аргументов командной строки.
3. Задание для самостоятельной работы.

3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Кроме того, в нём выделяется память для локальных переменных и могут временно храниться значения регистров. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является

инструкция loop. Она позволяет организовать безусловный цикл.

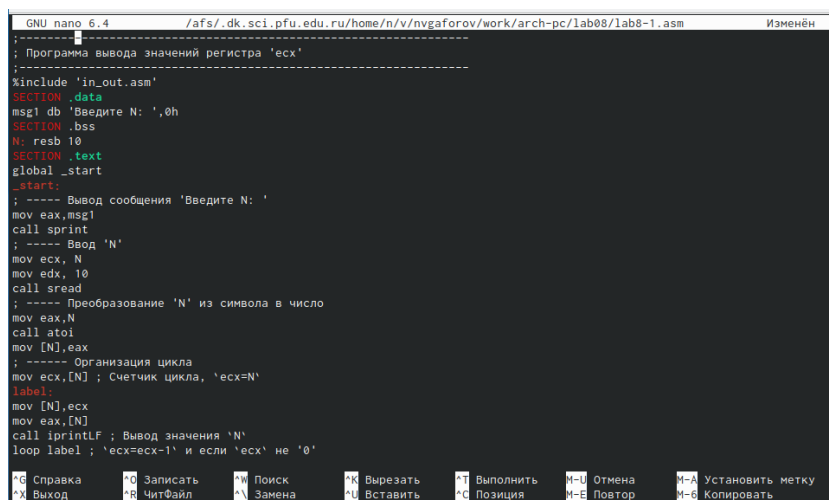
4 Выполнение лабораторной работы

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm.(рис. [4.1])

```
nvgaforov@dk8n80 ~ $ mkdir ~/work/arch-pc/lab08
nvgaforov@dk8n80 ~ $ cd ~/work/arch-pc/lab08
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ touch lab8-1.asm
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab8-1.asm текст программы из листинга 8.1.(рис. [4.2])



```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab08/lab8-1.asm
;-----
; Программа вывода значений регистра 'ecx'
;-----
#include "in_out.asm"
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ---- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ---- Ввод 'N'
mov ecx, N
mov edx, 10
call read
; ---- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, 'ecx'=N
label:
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
```

Рис. 4.2: Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу (рис. [4.3])

```

nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 50
50
49
48
47
46
45
44
43
42
41
40
39
38
37
36
35
34
33

```

Рис. 4.3: Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно. Изменяю текст программы, добавив изменение значения регистра ecx в цикле (рис. [4.4])

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab08/lab8-1.asm
; Программа вывода значений регистра 'ecx'
;-----
%include 'in_out.asm'
SECTION .data
msg1 db 'Введите N: ',0h
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ---- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ---- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ---- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, 'ecx=N'
label:
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintf ; Вывод значения 'N'

```

Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.5])

```
4279727288
4279727286
4279727284
4279727282
4279727280
4279727278
4279727276
4279727274
4279727272
4279727270
4279727268
4279727266
4279727264
4279727262
4279727260
4279727258
4279727256
4279727254
4279727252
4279727250
4279727248
4279727246
4279727244
4279727242
4279727240
4279727238
4279727236
4279727234
4279727232
4279727230
4279727228
4279727226
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $
```

Рис. 4.5: Запуск обновленной программы

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению. Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop`. (рис. [4.6])

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/vnvgaforov/work/arch-pc/lab08/lab8-1.asm
SECTION .bss
N: resb 10
SECTION .text
global _start
_start:
; ----- Вывод сообщения 'Введите N: '
mov eax,msg1
call sprint
; ----- Ввод 'N'
mov ecx, N
mov edx, 10
call sread
; ----- Преобразование 'N' из символа в число
mov eax,N
call atoi
mov [N],eax
; ----- Организация цикла
mov ecx,[N] ; Счетчик цикла, 'ecx=N'
label:
push ecx ; добавление значения ecx в стек
sub ecx,1 ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx ; извлечение значения ecx из стека
loop label ; 'ecx=ecx-1' и если 'ecx' не '0'
; переход на 'label'
call quit
```

Рис. 4.6: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. [4.7])

```

nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
nvgaforov@dk8n80 ~/work/arch-pc/lab08 $

```

Рис. 4.7: Запуск исполняемого файла

##Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.2 (рис. [4.8])

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab08/lab8-2.asm
;-----
; Обработка аргументов командной строки
;-----
%include 'in_out.asm'
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в 'ecx' количество
    ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в 'edx' имя программы
    ; (второе значение в стеке)
    sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество
    ; аргументов без названия программы)
next:
    cmp ecx, 0 ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
    ; (переход на метку '_end')
    pop eax ; иначе извлекаем аргумент из стека
    call sprintf ; вызываем функцию печати
    loop next ; переход к обработке следующего
    ; аргумента (переход на метку 'next')
_end:
    call quit

```

Рис. 4.8: Ввод текста программы из листинга 8.2

Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. [4.9])

```

nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $

```

Рис. 4.9: Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличие от аргумента 3, поэтому из-за пробела программа считает “2” как отдельный аргумент.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm в каталоге ~/work/archpc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. [4.10])

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab08/lab8-3.asm
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
    pop ecx ; Извлекаем из стека в 'ecx' количество
    ; аргументов (первое значение в стеке)
    pop edx ; Извлекаем из стека в 'edx' имя программы
    ; (второе значение в стеке)
    sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
    ; аргументов без названия программы)
    mov esi, 0 ; Используем 'esi' для хранения
    ; промежуточных сумм
next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end ; если аргументов нет выходим из цикла
    ; (переход на метку '_end')
    pop eax ; иначе извлекаем следующий аргумент из стека
    call atoi ; преобразуем символ в число
    add esi,eax ; добавляем к промежуточной сумме
    ; след. аргумент 'esi=esi+eax'
    loop next ; переход к обработке следующего аргумента
_end:
    mov eax, msg ; вывод сообщения "Результат: "
    call sprint
    mov eax, esi ; записываем сумму в регистр 'eax'
    call iprintf ; печать результата
    call quit ; завершение программы

```

Рис. 4.10: Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. [4.11])

```

nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ touch lab8-3.asm
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ./lab8-3 7 12 15 5
Результат: 39
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $

```

Рис. 4.11: Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. [4.12])

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/n/v/nvgaforov/work/arch-pc/lab08/lab8-3.asm
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в 'ecx' количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в 'edx' имя программы
; (второе значение в стеке)
sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
; аргументов без названия программы)
mov esi, 1 ; Используем 'esi' для хранения
; промежуточных сумм
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку '_end')
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число
imul esi,eax ; добавляем к промежуточной сумме
; след. аргумент 'esi=esi+eax'
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем сумму в регистр 'eax'
call iprintf ; печать результата
call quit ; завершение программы

```

Рис. 4.12: Изменение текста прораммы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. [4.13])

```

nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $ ./lab8-3 7 12 15 5
Результат: 6300
nvgaforov@dk5n59 ~/work/arch-pc/lab08 $

```

Рис. 4.13: Запуск исполняемого файла

##Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x) = 5 \cdot (2 + x)$ в соответствии для $x = x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы. (рис. [4.14])

```
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi, 0
mov edi,3
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,10
mul edi
add esi,eax
loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Прочитано 26 строк

Справка Записать Поиск Вырезать Выполнить Отмена Установить ме
Выход ЧитФайл Замена Вставить Позиция Повтор Копировать

Рис. 4.14: Текст Программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. (рис. [4.15])

```
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ touch lab8-4.asm
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Результат: 114
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ ./lab8-4 5 7 9 11
Результат: 216
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $ ./lab8-4 12 18 23 25
Результат: 354
nvgaforov@dk3n65 ~/work/arch-pc/lab08 $
```

Рис. 4.15: Запуск исполняемого файла и проверка его работы

Текст программы:

```
%include 'in_out.asm' SECTION .data msg db "Результат:",0 SECTION .text global
_start _start: pop ecx pop edx sub ecx,1 mov esi, 0 mov edi,3 next: cmp ecx,0h jz _end
pop eax call atoi add eax,10 mul edi add esi,eax loop next _end: mov eax, msg call
sprint mov eax, esi call iprintLF call quit
```

5 Выводы

Благодаря данной лабораторной работе я приобрела навыки написания программ использованием циклов и обработкой аргументов командной строки, что поможет мне при выполнении последующих лабораторных работ.

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://midnight-commander.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,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. 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.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
 17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
 18. — 1120 с. — (Классика Computer Science).