

Отчёт по лабораторной работе №8

Дисциплина: архитектура компьютеров и операционные системы

Светцова Анна Дмитриевна

Содержание

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.

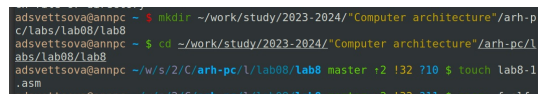
Команда `pop` извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр `esp`, после этого уменьшает значение регистра `esp` на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека 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

4.1 Реализация циклов в NASM

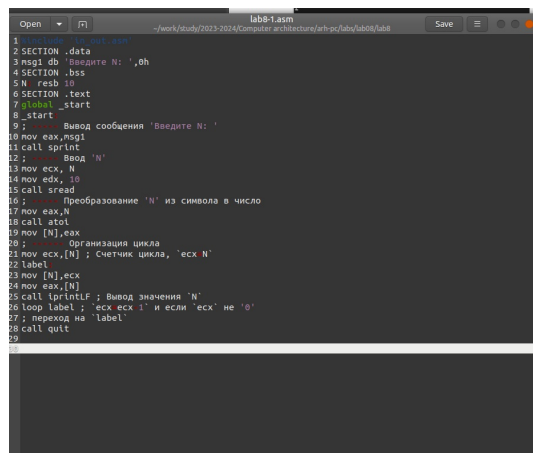
Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл `lab8-1.asm`. (рис. 1).



```
adsveitsova@annpc ~$ mkdir -p /work/study/2023-2024/Computer architecture/arh-pc/labs/lab08/lab8
adsveitsova@annpc ~$ cd /work/study/2023-2024/Computer architecture/arh-pc/labs/lab08/lab8
adsveitsova@annpc ~$ touch lab8-1.asm
adsveitsova@annpc ~$ ls -l /work/study/2023-2024/Computer architecture/arh-pc/labs/lab08/lab8
-rw-rw-r-- 1 adsveitsova annpc 0 2023-12-11 14:00 lab8-1.asm
```

рис.1 Создание файлов для лабораторной работы

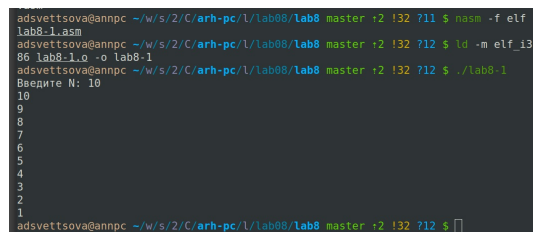
Ввожу в файл `lab8-1.asm` текст программы из листинга 8.1. (рис. 2).



```
1 ; lab8-1.asm
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N resb 10
6 SECTION .text
7 ;label_start
8 _start
9 ; Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprintf
12 ; Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'еск N'
22 label
23 mov [N],ecx
24 mov eax,[N]
25 call printf ; Вывод значения 'N'
26 loop label ; 'еск еск 1' и если 'еск' не '0'
27 ; переход на 'label'
28 call quit
29
```

рис.2 Ввод текста из листинга 8.1

Создаю исполняемый файл и проверяю его работу. (рис. 3).

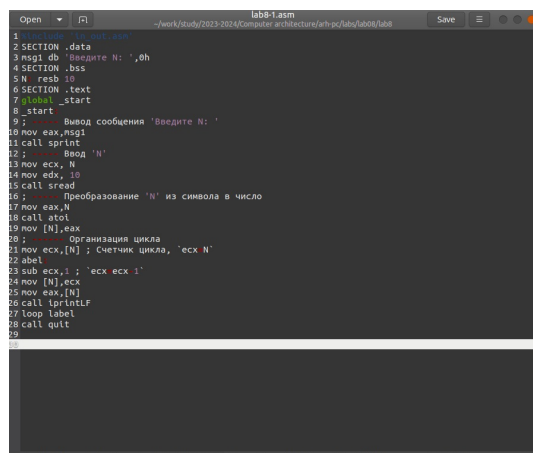


```
adsvettsova@annpc ~/w/s/2/C/ark-pc/1/lab08/lab8 master ✱ 132 711 $ nasm -f elf
lab8-1.asm
adsvettsova@annpc ~/w/s/2/C/ark-pc/1/lab08/lab8 master ✱ 132 712 $ ld -m elf_i386
lab8-1.o -o lab8-1
adsvettsova@annpc ~/w/s/2/C/ark-pc/1/lab08/lab8 master ✱ 132 712 $ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
adsvettsova@annpc ~/w/s/2/C/ark-pc/1/lab08/lab8 master ✱ 132 712 $
```

рис.3 Запуск исполняемого файла

Данная программа выводит числа от N до 1 включительно.

Изменяю текст программы, добавив изменение значения регистра еск в цикле. (рис. 4).



```
1 ; lab8-1.asm
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N resb 10
6 SECTION .text
7 ;label_start
8 _start
9 ; Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprintf
12 ; Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; Преобразование 'N' из символа в число
17 mov eax,N
18 call atoi
19 mov [N],eax
20 ; Организация цикла
21 mov ecx,[N] ; Счетчик цикла, 'еск N'
22 label
23 mov [N],ecx
24 mov eax,[N]
25 call printf ; Вывод значения 'N'
26 loop label ; 'еск еск 1' и если 'еск' не '0'
27 ; переход на 'label'
28 call quit
29
```

рис.4 Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 5).

```

adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ nasm -f elf
lab8-1.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ ld -m elf_i386
86 lab8-1.o -o lab8-1
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1

```

рис.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. 6).

```

1 include "io.inc.asm"
2 SECTION .data
3 msg1 db "Введите N: ",0h
4 SECTION .bss
5 N resb 10
6 SECTION .text
7 global _start
8 _start:
9     ; Вывод сообщения "Введите N: "
10    mov eax,msg1
11    call sprint
12    ; Ввод 'N'
13    mov ecx, N
14    mov edx, 10
15    call read
16    ; Преобразование 'N' из символа в число
17    mov eax,N
18    call atoi
19    mov [N],eax
20    ; Организация цикла
21    mov ecx,[N] ; Счетчик цикла, "еск N"
22    label:
23    push ecx ; добавление значения ecx в стек
24    sub ecx,1
25    mov [N],ecx
26    mov eax,[N]
27    call iprintf
28    pop ecx ; извлечение значения ecx из стека
29    loop label
30    call quit
31

```

рис.6 Изменение текста программы

Создаю исполняемый файл и проверяю его работу.(рис. 7).

```

4 adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ nasm -f elf
lab8-1.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ ld -m elf_i386
86 lab8-1.o -o lab8-1
adsvettsova@annpc ~/w/s/2/C/arch-pc/l/lab08/lab8 master +2 132 712 $ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0

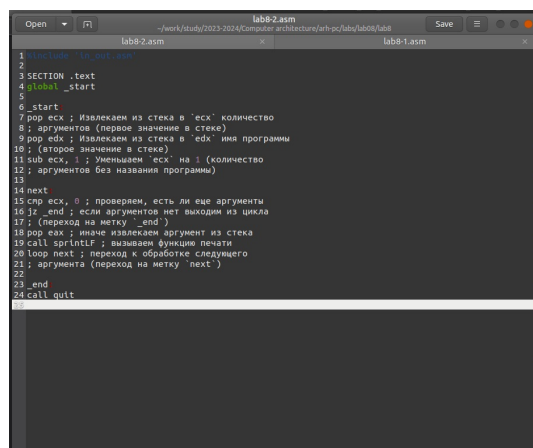
```

рис.7 Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению и выводит числа от N-1 до 0 включительно.

4.2 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. 8).



```
1 inc ecx
2
3 SECTION .text
4 global _start
5
6 _start
7 pop ecx ; Извлекаем из стека в 'ecx' количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в 'edx' имя программы
10 ; (второе значение в стеке)
11 sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество
12 ; аргументов без названия программы)
13
14 next
15 cmp ecx, 0 ; проверяем, есть ли еще аргументы
16 jz _end ; если аргументов нет выходим из цикла
17 ; (переход на метку '_end')
18 pop ecx ; иначе извлекаем аргумент из стека
19 call sprintf ; вызываем функцию печати
20 loop next ; переход к обработке следующего
21 ; аргумента (переход на метку 'next')
22
23 _end
24 call quit
```

рис.8 Ввод текста программы из листинга 8.2

Создаю исполняемый файл и запускаю его, указав нужные аргументы. (рис. 9).

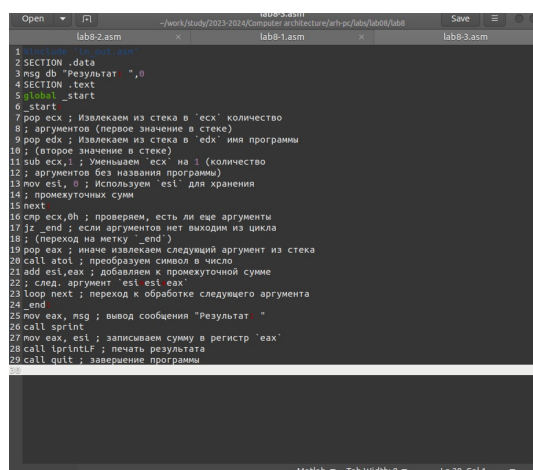


```
adsvettsova@annpc ~/w/s/2/C/arch-pc/1/lab08/lab8 master +2 132 712 $ nasm -f elf
lab8-2.asm
adsvettsova@annpc ~/w/s/2/C/arch-pc/1/lab08/lab8 master +2 132 712 $ ld -m elf_i386
lab8-2.o -o lab8-2
adsvettsova@annpc ~/w/s/2/C/arch-pc/1/lab08/lab8 master +2 132 712 $ ./lab8-2 argument1
argument1
argument
argument
argument 2
adsvettsova@annpc ~/w/s/2/C/arch-pc/1/lab08/lab8 master +2 132 712 $ 12:36:33
```

рис.9 Запуск исполняемого файла

Программа вывела 4 аргумента, так как аргумент 2 не взят в кавычки, в отличии от аргумента 3, поэтому из-за пробела программа считывает “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. 10).



```
1
2 SECTION .data
3 msg db "Результат ",0
4 SECTION .text
5 global _start
6 _start
7 pop ecx ; Извлекаем из стека в 'ecx' количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в 'edx' имя программы
10 ; (второе значение в стеке)
11 sub ecx, 1 ; Уменьшаем 'ecx' на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем 'esi' для хранения
14 ; промежуточных сумм
15 next
16 cmp ecx, 0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку '_end')
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 add esi, eax ; добавляем к промежуточной сумме
22 ; след. аргумент 'esi esi eax'
23 loop next ; переход к обработке следующего аргумента
24 _end
25 mov eax, msg ; вывод сообщения "Результат "
26 call sprintf
27 mov eax, esi ; записываем сумму в регистр 'eax'
28 call fprintf ; печать результата
29 call quit
```

рис.10 Ввод текста программы из листинга 8.3

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 11).

```

adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ touch lab8-3
.asm
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ nasm -f elf
lab8-3.asm
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ ld -m elf_i386
86 lab8-3.o -o lab8-3
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ ./lab8-3
Результат: 0
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ ./lab8-3 10
20 30
Результат: 60

```

рис.11 Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. 12).

```

1  #include <stdio.h>
2  SECTION .data
3  msg db "Результат ",0
4  SECTION .text
5  global _start
6  _start:
7  pop ecx ; Извлекаем из стека в 'ecx' количество
8  ; аргументов (первое значение в стеке)
9  pop edx ; Извлекаем из стека в 'edx' имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
12 ; аргументов без названия программы)
13 mov esi,1 ; Используем 'esi' для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку _end)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mul esi
22 mov esi,eax ; добавляем к промежуточной сумме
23 ; след. аргумент 'esi' esi eax
24 loop next ; переход к обработке следующего аргумента
25 _end:
26 mov eax,msg ; вывод сообщения "Результат "
27 call printf
28 mov eax,esi ; записываем сумму в регистр 'eax'
29 call printf ; печать результата
30 call quit ; завершение программы

```

рис.12 Изменение текста программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. 13).

```

Результат: 0
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ nasm -f elf
lab8-3.asm
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ ld -m elf_i386
86 lab8-3.o -o lab8-3
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $ ./lab8-3 10
20 30
Результат: 6000
adsvettsova@annpc ~/w/s/2/C/arb-pc/1/lab08/lab8 master +2 132 712 $

```

рис.13 Запуск исполняемого файла

4.3 Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x) = 7 + 2 \cdot x$ в соответствии с моим номером варианта (8) для $x = x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы. (рис. 14).

```

1 ; Задача: посчитать сумму аргументов
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx ; Извлекаем из стека в 'ecx' количество
8 ; аргументов (первое значение в стеке)
9 pop edx ; Извлекаем из стека в 'edx' имя программы
10 ; (второе значение в стеке)
11 sub ecx,1 ; Уменьшаем 'ecx' на 1 (количество
12 ; аргументов без названия программы)
13 mov esi, 0 ; Используем 'esi' для хранения
14 ; промежуточных сумм
15 next:
16 cmp ecx,0h ; проверяем, есть ли еще аргументы
17 jz _end ; если аргументов нет выходим из цикла
18 ; (переход на метку _end)
19 pop eax ; иначе извлекаем следующий аргумент из стека
20 call atoi ; преобразуем символ в число
21 mov ebx,2
22 mul ebx
23 add eax,7
24 add esi,eax ; добавляем к промежуточной сумме
25 ; след. аргумент esi=esi+eax
26 loop next ; переход к обработке следующего аргумента
27 _end:
28 mov eax, msg ; вывод сообщения "Результат: "
29 call sprint
30 mov ecx, esi ; записываем сумму в регистр 'ecx'
31 call iprintf ; печать результата

```

рис.14 Текст программы

Создаю исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$. (рис. 15).

```

adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ gcc -f elf
task.asm
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ ld -m elf_13
86 task.o -o task
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ ./task 1 2 3
4
Результат: 48
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ ./task 5 6 7
8
Результат: 80
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ ./task 17 39
6 18
Результат: 188
adsvetsova@annpc ~/w/s/2/C/arrh-pc/l/lab08/lab8 master r2 132 712 $ 12:56:38

```

рис.15 Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Текст программы:

```

#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 mov ebx,2
mul ebx add eax,7 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 ecx, esi ; записываем сумму в регистр
eax call iprintf ; печать результата call quit ; завершение
программы

```

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.

6 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/Learningbash-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. — БХВПетербург, 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, 2015. — 1120 с. — (Классика Computer Science).