РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

дисциплина: Архитектура компьютера

Студент: С е м е н о в Б о г д а н

Группа: НКАбд-05-25

МОСКВА
2025 г.

Оглавление

1 Цель работы

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

2 Задания

1. 8.3.1 Реализация циклов в NASM
2. 8.3.2 Обработка аргументов командной строки
3. 8.4 Самостоятельная реализация программного кода в соответствии с заданиями лабораторной 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.

На рис. 8.1 показана схема организации стека в процессоре.

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.

Для стека существует две основные операции:

• добавление элемента в вершину стека (push);

• извлечение элемента из вершины стека (pop).

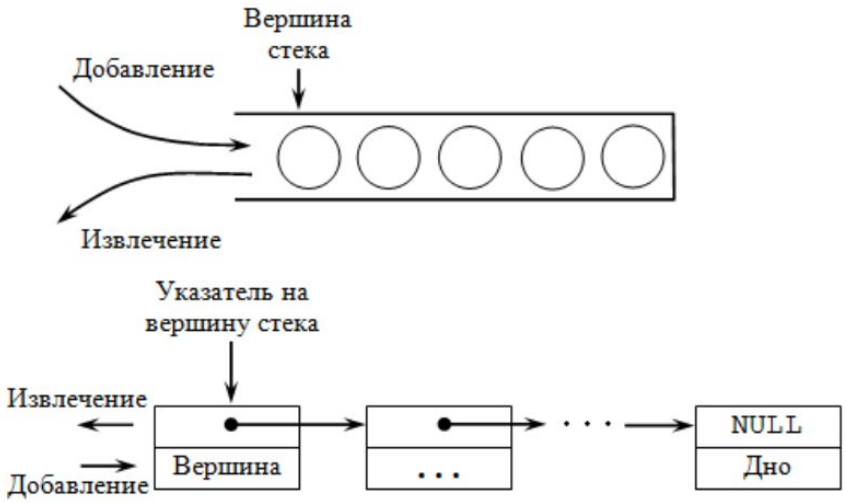

Рис. 8.1. Организация стека в процессоре

4 Выполнение работы

Реализация циклов в NASM

Создадим каталог для программам лабораторной работы №8, перейдем в него и создадим файл lab8-1.asm (рис. 1-2)

Рис. 1 создание каталога и переход в каталог

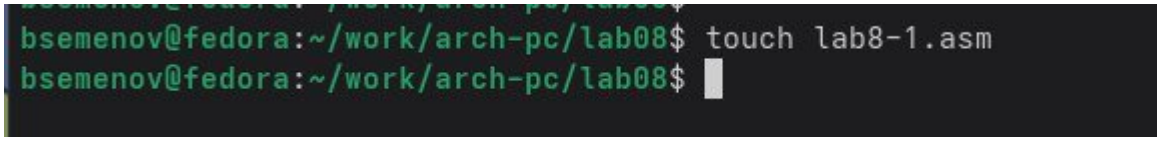

Рис. 2 создание файла lab8-1.asm

Введем в файл lab8-1.asm текст программы из листинга (рис. 3)

```
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

call quit
```

Рис. 3 запись программы

Создадим исполняемый файл и проверим его работу (рисунок 4-5)

```
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
bsemenov@fedora:~/work/arch-pc/lab08$
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 4 создание исполняемого файла

Рис. 5 запускаем и проверяем исполняемый файл

4. Внесём правки в код так, чтобы значение регистра ECX менялось во время выполнения цикла. (рис. 6 - 8)

```
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
sub ecx,1 ; `ecx=ecx-1`
mov [N],ecx
mov eax,[N]
call iprintLF
loop label

call quit
```

Рис. 6 изменение программы

Создадим исполняемый файл и проверьте его работу

```
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
```

Рис. 7 создание и запуск исполняемого файла

```
bsemenov@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 8
7
5
3
1
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 8 проверка работы программы

1. Число итераций сокращается в два раза, поскольку на каждом шаге цикла значение регистра ECX уменьшается на 2.
   Внесём корректировки в код, добавив инструкции PUSH и POP для сохранения состояния счётчика цикла LOOP. (рис. 9)

```
%include 'in_out.asm'

SECTION .data
msg1 db 'Введите N: ',0h

SECTION .bss
N: resb 10

SECTION .text
global _start
_start:

mov eax,msg1
call sprint

mov ecx, N
mov edx, 10
call sread

mov eax,N
call atoi
mov [N],eax

mov ecx,[N]
label:
push ecx
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx
loop label

call quit
```

Рис. 9 изменение программы

Создадим исполняемый файл и проверим его работу (рис. 10 - 1)

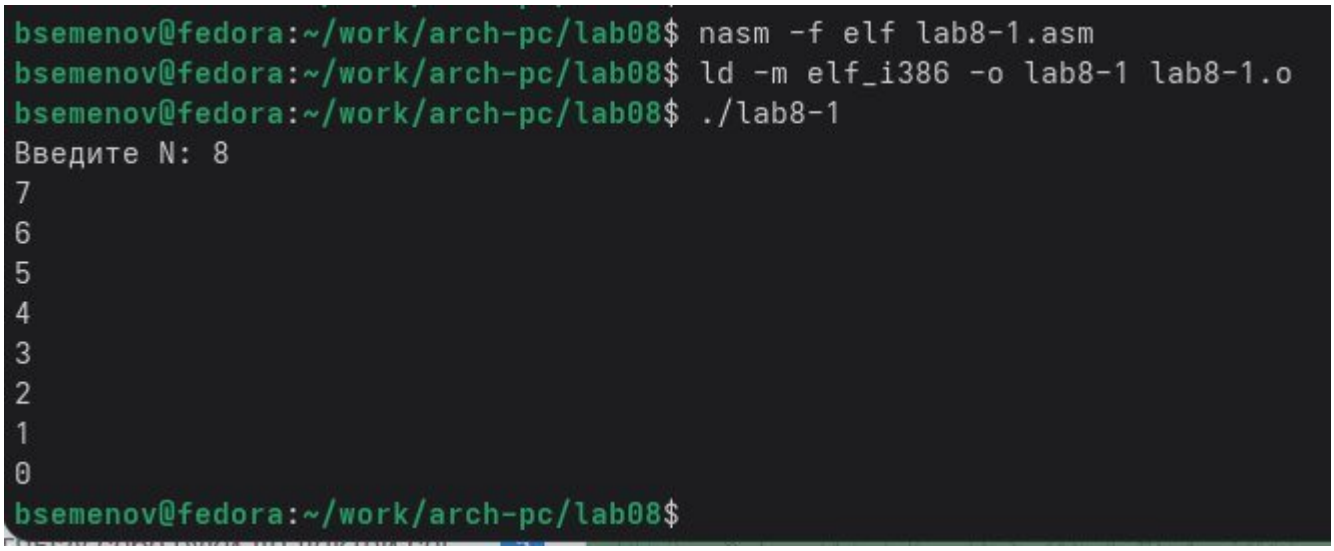

Рис. 10 создание и проверка исполняемого файла

1. В результате, выводимые числа сместились на -1, однако теперь количество итераций цикла соответствует введённому значению N.

Обработка аргументов командной строки

Создадим файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга (рис. 11)

Открыть

lab8-2.asm
~/work/arch-pc/lab08

```
%include "in_out.asm"

SECTION .text
global _start

_start:
pop ecx
pop edx
sub ecx, 1

next:
cmp ecx, 0
jz _end
pop eax
call sprintLF
loop next

_end:
call quit
```

Рис. 11 запись программы

3. Создадим исполняемый файл и запустим его, передав аргументы через командную строку. (Рис. 12 - 13)

```
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 12 создание исполняемого файла

```
bsemenov@fedora:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 13 проверка работы программы

1. После успешной обработки программой всех переданных аргументов создадим в каталоге ~/work/arch pc/lab08 файл lab8-3.asm и запишем в него текст программы согласно листингу.(рис. 14 - 1)

```
bsemenov@fedora:~/work/arch-pc/lab08$ touch lab8-3.asm
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 14 создание файла

```
%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

next:
cmp ecx,0h
jz _end
pop eax
call atoi
add esi,eax

loop next
_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 15 запись программы

Создадим исполняемый файл и запустим его с указанием аргументов командной строки. (рис. 16)

```
bsemenov@fedora:~/work/arch-pc/lab08$
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
bsemenov@fedora:~/work/arch-pc/lab08$ ./lab8-3 8 17 7 12
Результат: 44
```

Рис. 16 создание, запуск и проверка исполняемого файла

Изменим текст программы из листинга для вычисления произведения аргументов командной строки (рис. 17 - 18)

```
%include 'in_out.asm'

SECTION .data
msg db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
pop ecx
pop edx
sub ecx, 1
mov esi, 1

next:
cmp ecx, 0h
jz _end
pop eax
call atoi
mul esi
mov esi, eax

loop next

_end:
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
```

Рис. 17 изменение программы

```
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
bsemenov@fedora:~/work/arch-pc/lab08$ ./lab8-3 8 17 7 12
Результат: 11424
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 18 создание и проверка исполняемого файла

Задание для самостоятельной работы

Напишем программу, которая находит сумму значений функции $f(x)$ для x $=x1,x2,...,xn$, т.е. программа должна выводить значение $f(x1)+f(x2)+...+f(xn)$.

Выражение для f(x)=17+5x. (рис. 19)

Открыть

lab8-4.asm

```
%include "in_out.asm"

SECTION .data
msg_func db "Функция: f(x)=17+5x",0
msg_result db "Результат: ",0

SECTION .text
global _start

_start:
mov eax, msg_func
call sprintLF

pop ecx
pop edx
sub ecx,1
mov esi,0

next:
cmp ecx,0h
jz _end
pop eax
call atoi

mov ebx,5
mul ebx
add eax,17

add esi,eax
```

Рис. 19 запись программы

Создадим исполняемый файл и проверьте его работу на нескольких наборах $x = x1,x2,...,xn$ (рис. 20 - 21)

```
bsemenov@fedora:~/work/arch-pc/lab08$ touch lab8-4.asm
bsemenov@fedora:~/work/arch-pc/lab08$
bsemenov@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-4.asm
bsemenov@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
```

Рис. 20 создание исполняемого файла

```
bsemenov@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3 4
Функция: f(x)=17+5x
Результат: 118
bsemenov@fedora:~/work/arch-pc/lab08$
```

Рис. 21 запуск и проверка работы программы

5 Выводы

В ходе выполнения этой работы я приобрел практические навыки программирования на ассемблере. Научился работать с циклами и обрабатывать аргументы командной строки, что является важной основой для понимания низкоуровневого программирования.

Список литературы

1. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М.: Форум, 2018.

2. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М.: Солон-Пресс, 2017.

3. Новожилов О. П. Архитектура ЭВМ и систем. — М.: Юрайт, 2016.

4. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

5. 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.

6. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб.: Питер, 2013. — 874 с. — (Классика Computer Science).

7. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2015. — 1120 с. — (Классика Computer Science).

8. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658.

9. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

10. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

Онлайн-ресурсы и документация:

11. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

12. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

13. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander.org/.

14. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

15. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

16. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.