

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Самарханова Полина Тимуровна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
4.1	Реализация циклов в NASM	7
4.2	Обработка аргументов командной строки	14
4.3	Задание для самостоятельной работы	18
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога	7
4.2	Создание файла	7
4.3	Копирование файла in_out.asm	7
4.4	Редактирование программы	8
4.5	Запуск программы	9
4.6	Редактирование программы	10
4.7	Запуск программы	11
4.8	Запуск программы	12
4.9	Редактирование программы	13
4.10	Запуск программы	14
4.11	Создание файла	14
4.12	Редактирование программы	15
4.13	Запуск программы	15
4.14	Создание файла	15
4.15	Редактирование программы	16
4.16	Запуск программы	16
4.17	Редактирование программы	17
4.18	Запуск программы	17
4.19	Создание файла	18
4.20	Редактирование программы	19
4.21	Запуск программы	19

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. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Я создала директорию, в которой далее выполняла лабораторную работу (рис. 4.1).

```
[spolina@fedora ~]$ mkdir ~/work/arch-pc/lab08
```

Рис. 4.1: Создание каталога

Перехожу в созданный каталог и создаю файл lab8.asm (рис. 4.2).

```
[spolina@fedora ~]$ cd ~/work/arch-pc/lab08  
[spolina@fedora lab08]$ touch lab8.asm
```

Рис. 4.2: Создание файла

Далее я скопировала файл in_out.asm (рис. 4.3).

```
[spolina@fedora lab08]$ cp ~/Загрузки/in_out.asm in_out.asm  
[spolina@fedora lab08]$ ls  
in_out.asm lab8.asm
```

Рис. 4.3: Копирование файла in_out.asm

После чего я записала текст кода из листинга 8.1 (рис. 4.4). Эта программа запрашивает число N, и выдает все числа перед N вместе с ним до 0.

```

GNU nano 7.2
#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:
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'
; переход на `label`
call quit

```

Рис. 4.4: Редактирование программы

Далее я создала исполняемый код и проверила, что он работает(рис. 4.5).


```
[spolina@fedora lab08]$ nasm -f elf lab8.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[spolina@fedora lab08]$ ./lab8
Введите N: 10
10
9
8
7
6
5
4
3
2
1
```

Рис. 4.5: Запуск программы

После я отредактировала код, добавив изменение значения регистра `ecx` в цикле (рис. 4.6).

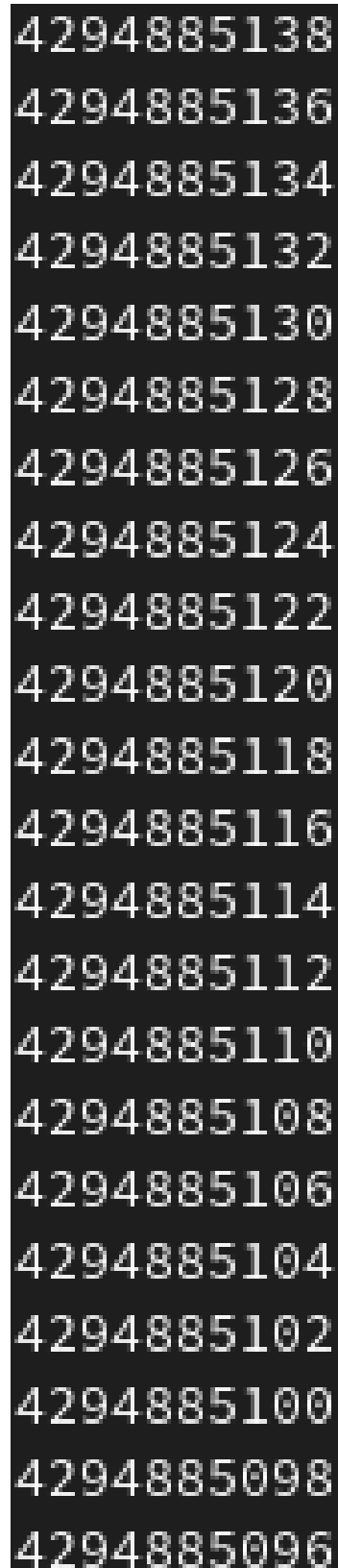
```

GNU nano 7.2
#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 iprintLF
loop label
call quit

```

Рис. 4.6: Редактирование программы

Я запустила программу. Код стал заикливаться и бесконечно передавать последовательные значения (рис. 4.7).



4294885138
4294885136
4294885134
4294885132
4294885130
4294885128
4294885126
4294885124
4294885122
4294885120
4294885118
4294885116
4294885114
4294885112
4294885110
4294885108
4294885106
4294885104
4294885102
4294885100
4294885098
4294885096

Рис. 4.7: Запуск программы

Пробую еще раз запустить программу,но с другим числом, теперь она выдаёт предыдущие числа,но перескакивает через 1 (рис. 4.8).

```
[spolina@fedora lab08]$ ./lab8
Введите N: 26
25
23
21
19
17
15
13
11
9
7
5
3
1
```

Рис. 4.8: Запуск программы

Еще раз отредактировала код программы,добавив команды push и pop для сохранения значения счетчика цикла loop (рис. 4.9).

```

GNU nano 7.2
#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:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit

```

Рис. 4.9: Редактирование программы

Создаю и запускаю исполняемый файл (рис. 4.10). Теперь программа показывает все предыдущие числа до 0, не включая заданное N

```
[spolina@fedora lab08]$ nasm -f elf lab8.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8 lab8.o
[spolina@fedora lab08]$ ./lab8
Введите N: 17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
```

Рис. 4.10: Запуск программы

4.2 Обработка аргументов командной строки

Далее я создала новый файл lab8-2.asm, используя команду touch (рис. 4.11).

```
[spolina@fedora lab08]$ touch lab8-2.asm
```

Рис. 4.11: Создание файла

После я открыла файл в GNU nano и записала код из листинга 8.2 (рис. 4.12). Данная программа позволяет выводить на экран аргументы командной строки.

```

GNU nano 7.2
#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. 4.12: Редактирование программы

Запускаю исполняемый файл вместе с аргументами (аргумент1, аргумент2, 'аргумент3') (рис. 4.13).

```

[spolina@fedora lab08]$ nasm -f elf lab8-2.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[spolina@fedora lab08]$ ./lab8-2 аргумент1 аргумент2 "аргумент3"
аргумент1
аргумент2
аргумент3

```

Рис. 4.13: Запуск программы

Далее я создала новый файл lab8-3.asm, используя команду touch (рис. 4.14).

```

[spolina@fedora lab08]$ touch lab8-3.asm

```

Рис. 4.14: Создание файла

После я открыла файл в GNU nano и записала код из листинга 8.3 (рис. 4.15). Данная программа позволяет выводить на экран сумму аргументов командной строки.

```
GNU nano 7.2
#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 iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.15: Редактирование программы

Далее я запустила исполняемый файл вместе с аргументами (рис. 4.16).

```
[spolina@fedora lab08]$ nasm -f elf lab8-3.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[spolina@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.16: Запуск программы

Отредактировала код программы так, чтобы она выводила произведение всех аргументов (рис. 4.17).

```
GNU nano 7.2
#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
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
pop eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в числ
mul esi
mov 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 iprintLF ; печать результата
call quit ; завершение программы
```

Рис. 4.17: Редактирование программы

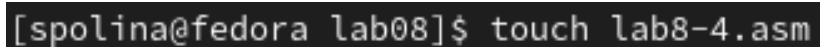
После я запустила исполняемый файл вместе с аргументами (рис. 4.18). Программа выдаёт произведение всех аргументов.

```
[spolina@fedora lab08]$ nasm -f elf lab8-3.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[spolina@fedora lab08]$ ./lab8-3 5 9 2
Результат: 90
```

Рис. 4.18: Запуск программы

4.3 Задание для самостоятельной работы

Я создала файл lab8-4.asm в котором написала код для последней задачи (рис. 4.19).

A terminal window with a dark background. The prompt is [spolina@fedora lab08]\$ and the command touch lab8-4.asm has been entered.

```
[spolina@fedora lab08]$ touch lab8-4.asm
```

Рис. 4.19: Создание файла

Далее написала код программы, который позволяет вывести сумму всех преобразованных аргументов из варианта №9 (рис. 4.20).

```

GNU nano 7.2
#include 'in_out.asm'
SECTION .data
msg db "Ответ: ",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,10
mul ebx
sub eax,4
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 iprintLF ; печать результата
call quit ; завершение программы

```

Рис. 4.20: Редактирование программы

Запустила исполняемый файл (рис. 4.21). Программа выдала верную сумму всех преобразованных аргументов.

```

[spolina@fedora lab08]$ nasm -f elf lab8-4.asm
[spolina@fedora lab08]$ ld -m elf_i386 -o lab8-4 lab8-4.o
[spolina@fedora lab08]$ ./lab8-4 3 4 6 7
Ответ: 184

```

Рис. 4.21: Запуск программы

5 Выводы

В данной лабораторной работе я научилась работать с циклами, выводом аргументов и функций.

Список литературы

1. Лабораторная работа №8