

ОТЧЕТ по лабораторной работе №8

дисциплина: Архитектура компьютера

Студент: Идрисов Д.А.

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	15
4.3	Задание для самостоятельной работы	20
5	Выводы	23

Список иллюстраций

4.1	Изменение кода	10
4.2	Запуск программы	11
4.3	Изменение кода	12
4.4	Запуск программы	13
4.5	Изменение кода	14
4.6	Запуск программы	15
4.7	Запуск программы	16
4.8	Изменение кода	17
4.9	Запуск программы	18
4.10	Изменение кода	19
4.11	Запуск программы	20
4.12	Изменение кода	21
4.13	Запуск программы	22

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием циклов и обработкой аргументов командной строки..

2 Задание

1. Изучение стека и цикла в ассемблере
2. Изучение примеров программ со стеком
3. Изучение примеров программ с аргументами
4. Выполнение заданий 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.

Для стека существует две основные операции:

- добавление элемента в вершину стека (push);
- извлечение элемента из вершины стека (pop).

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.

Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

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.

Аналогично команде записи в стек существует команда `pop`, которая восстанавливает из стека все регистры общего назначения, и команда `popf` для перемещения значений из вершины стека в регистр флагов.


4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создал каталог для программ лабораторной работы № 8 и файл lab8-1.asm

При использовании инструкции `loop` в NASM для реализации циклов, необходимо учитывать, что она использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. Для наглядности рассмотрим программу, которая выводит значение регистра `ecx`.

Написал в файл lab8-1.asm текст программы из листинга 8.1. Создал исполняемый файл и проверил его работу.

Открыть  lab8-1.asm
~/work/arch-pc/lab08 Стр. 28, 1

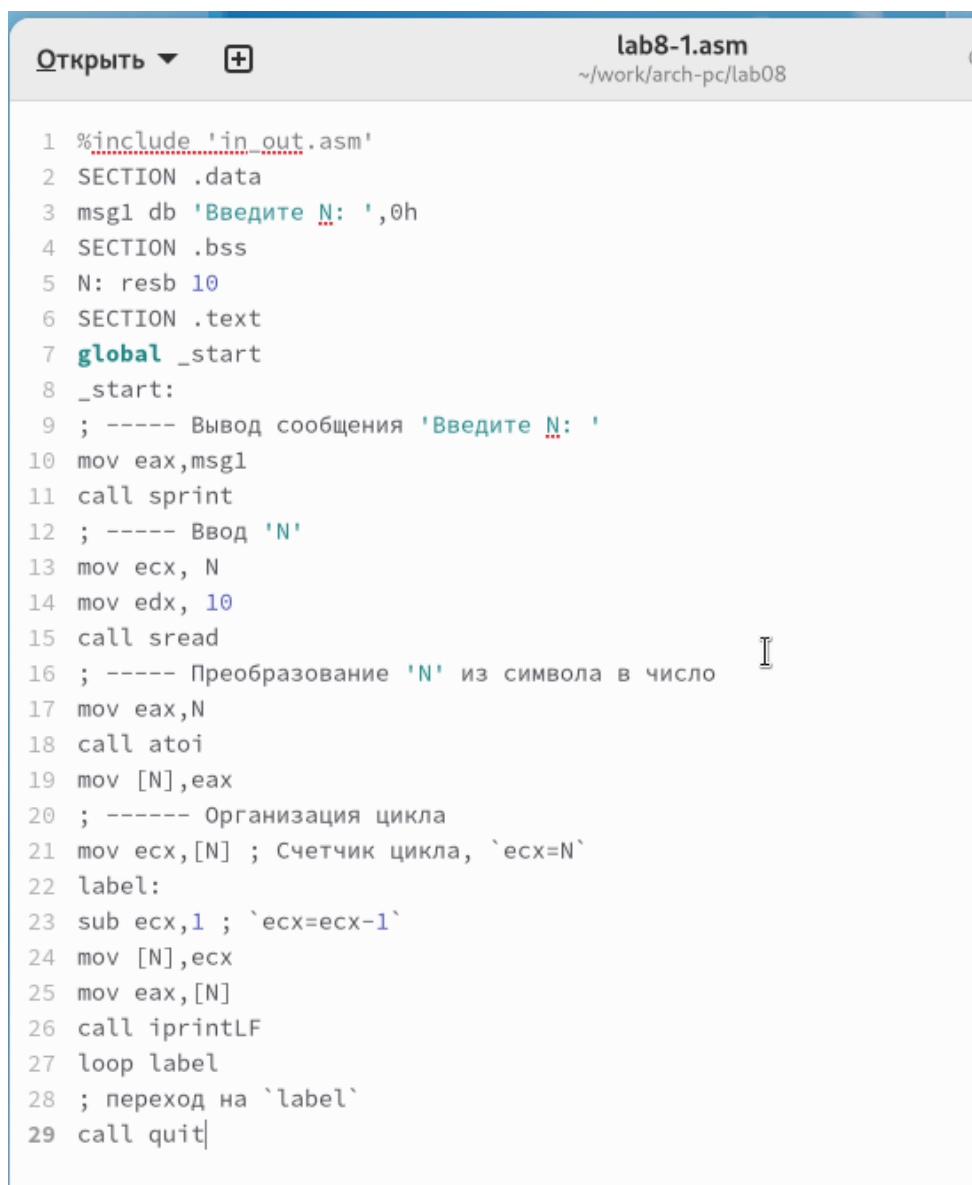
```
1 %include 'in_out.asm'
2 SECTION .data
3 msg1 db 'Введите N: ',0h
4 SECTION .bss
5 N: resb 10
6 SECTION .text
7 global _start
8 _start:
9 ; ----- Вывод сообщения 'Введите N: '
10 mov eax,msg1
11 call sprint
12 ; ----- Ввод 'N'
13 mov ecx, N
14 mov edx, 10
15 call sread
16 ; ----- Преобразование '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, `ecx=N`
22 label:
23 mov [N],ecx
24 mov eax,[N]
25 call iprintLF ; Вывод значения `N`
26 loop label ; `ecx=ecx-1` и если `ecx` не `0`
27 ; переход на `label`
28 call quit
```

Рис. 4.1: Изменение кода

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

Рис. 4.2: Запуск программы

Однако, в данном примере становится очевидно, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Чтобы исправить это, мы можем использовать стек для сохранения значения счетчика цикла `loop`. Программа запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N`.



```
1 %include 'in_out.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 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, `ecx=N`
22 label:
23 sub ecx,1 ; `ecx=ecx-1`
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF
27 loop label
28 ; переход на `label`
29 call quit
```


Рис. 4.3: Изменение кода

```
4294907741
4294907739
4294907737
4294907735
4294907733
4294907731
4294907729
4294907727
4294907725
4294^C
[daidrisov@fedora lab08]$ ./lab8-1
Введите N: 4
3
1
[daidrisov@fedora lab08]$
```

Рис. 4.4: Запуск программы

Внесем изменения в текст программы, добавив команды `push` и `pop`, чтобы сохранить и извлечь значение счетчика из стека соответственно. После этого создадим исполняемый файл и проверим его работу. Таким образом, мы обеспечим корректность работы программы.

Создал исполняемый файл и проверьте его работу. Программа выводит числа от $N-1$ до 0, число проходов цикла соответствует N .

Открыть ▾ 

lab8-1.asm
~/work/arch-pc/lab08

Стр. 30, Поз. 10

```
1 %include 'in_out.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 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, `ecx=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 iprintLF
28 pop ecx ; извлечение значения ecx из стека
29 loop label
30 call quit
```

Рис. 4.5: Изменение кода

```
[daidrisov@fedora lab08]$  
[daidrisov@fedora lab08]$ nasm -f elf lab8-1.asm  
[daidrisov@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1  
[daidrisov@fedora lab08]$ ./lab8-1  
Введите N: 4  
3  
2  
1  
0  
[daidrisov@fedora lab08]$ ./lab8-1  
Введите N: 7  
6  
5  
4  
3  
2  
1  
0  
[daidrisov@fedora lab08]$
```

Рис. 4.6: Запуск программы

4.2 Обработка аргументов командной строки

Создал файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввел в него текст программы из листинга 8.2.


Создал исполняемый файл и запустил его, указав аргументы. Программа обработала 5 аргументов. Аргументами считаются слова/числа, разделенные пробелом.

![Изменение кода(image/07.png){ #fig:007 width=70%, height=70% }]

```
[daidrisov@fedora lab08]$  
[daidrisov@fedora lab08]$ nasm -f elf lab8-2.asm  
[daidrisov@fedora lab08]$ ld -m elf_i386 lab8-2.o -o lab8-2  
[daidrisov@fedora lab08]$ ./lab8-2  
[daidrisov@fedora lab08]$  
[daidrisov@fedora lab08]$ ./lab8-2 argument 1 argument 2 'argument3'  
argument  
1  
argument  
2  
argument3  
[daidrisov@fedora lab08]$
```

Рис. 4.7: Запуск программы

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы.

Открыть ▾ 

lab8-3.asm
~/work/arch-pc/lab08

Стр. 29, Поз. 33


```
1 %include 'in_out.asm'
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 sprint
27 mov eax, esi ; записываем сумму в регистр `eax`
28 call iprintLF ; печать результата
29 call quit ; завершение программы
```

Рис. 4.8: Изменение кода

```
[daidrisov@fedora lab08]$  
[daidrisov@fedora lab08]$ nasm -f elf lab8-3.asm  
[daidrisov@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3  
[daidrisov@fedora lab08]$ ./lab8-3  
Результат: 0  
[daidrisov@fedora lab08]$ ./lab8-3 3 7 9  
Результат: 19  
[daidrisov@fedora lab08]$
```

Рис. 4.9: Запуск программы

Изменил текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

Открыть ▾ 

lab8-3.asm
~/work/arch-pc/lab08

Стр. 32, Поз. 33

```
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 mov ebx,eax
22 mov eax,esi
23 mul ebx
24 mov 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 eax,esi ; записываем сумму в регистр `eax`
31 call iprintLF ; печать результата
32 call quit ; завершение программы
```

Рис. 4.10: Изменение кода

```
[daidrisov@fedora lab08]$ nasm -f elf lab8-3.asm
[daidrisov@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[daidrisov@fedora lab08]$ ./lab8-3
Результат: 1
[daidrisov@fedora lab08]$ ./lab8-3 3 7 9
Результат: 189
[daidrisov@fedora lab08]$
```

Рис. 4.11: Запуск программы

4.3 Задание для самостоятельной работы

Напишите программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа должна выводить значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x передаются как аргументы. Вид функции $f(x)$ выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах x .

для варианта 17 $f(x) = 10(x - 1)$

```
3 msg db "Результат: ",0
4 fx: db 'f(x)= 10(x - 1)',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 sub| eax,1
22 mov ebx,10
23 mul ebx
24 add esi,eax
25
26 loop next
27
28 _end:
29 mov eax, msg
30 call sprintf
31 mov eax, esi
32 call iprintLF
33 call quit
```

Рис. 4.12: Изменение кода

Для проверки я запустил сначала с одним аргументом. Так, при подстановке $f(0) = 7 * 1 = 7$, $f(1) = 7 * 2 = 14$ Затем подал несколько аргументов и получил сумму значений функции.

```
[daidrisov@fedora lab08]$  
[daidrisov@fedora lab08]$ nasm -f elf task.asm  
[daidrisov@fedora lab08]$ ld -m elf_i386 task.o -o task  
[daidrisov@fedora lab08]$ ./task 1  
f(x)= 10(x - 1)  
Результат: 0  
[daidrisov@fedora lab08]$ ./task 2  
f(x)= 10(x - 1)  
Результат: 10  
[daidrisov@fedora lab08]$ ./task 7 6 9  
f(x)= 10(x - 1)  
Результат: 190  
[daidrisov@fedora lab08]$
```

Рис. 4.13: Запуск программы

5 Выводы

Освоили работы со стеком, циклом и аргументами на ассемблере `naasm`.