

Отчет по лабораторной работе №8

Программирование цикла. Обработка аргументов командной строки.

Ашуров Захид Фамил оглы

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
5	Самостоятельная работы	15
6	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание папки и переход в него	10
4.2	Создание файла lab8-1.asm	10
4.3	Вписывание текста из листинга в lab8-1.asm	10
4.4	Некорректный запуск	11
4.5	Редактирование файла lab8-1.asm	11
4.6	Создание и запускание исполняемого файла	11
4.7	Внесения изменения в lab8-1.asm	12
4.8	Создание и запускание исполняемого файл	12
4.9	Создание файла lab8-2.asm	12
4.10	Вписывание текста из листинга в файл lab8-2.asm	13
4.11	Создание и запускание исполняемого файла	13
4.12	Создание и редактирование файла lab8-3.asm	14
4.13	Создание и запускание исполняемого файла	14
5.1	Создание файла lab8-4.asm	15
5.2	Записывание кода	15
5.3	Создание и запускание исполняемого файла	15

Список таблиц

1 Цель работы

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

2 Задание

Реализация циклов в NASM

Обработка аргументов командной строки

3 Теоретическое введение

- Организация стека
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).
- 8.2.1.1. Добавление элемента в стек.

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.

push -10 ; Поместить -10 в стек
push ebx ; Поместить значение регистра ebx в стек
push [buf] ; Поместить значение переменной buf в стек
push word [ax] ; Поместить в стек слово по адресу в ax

Существует ещё две команды для добавления значений в стек. Это команда `pusha`, кото

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.

8.2.1.2. Извлечение элемента из стека.

Команда `pop` извлекает значение из стека, т.е. извлекает значение из ячейки памяти

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.

Примеры:

`pop eax` ; Поместить значение из стека в регистр `eax` `pop [buf]` ; Поместить значение из стека в `buf` `pop word[si]` ; Поместить значение из стека в слово по адресу в `si`

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

Для организации циклов существуют специальные инструкции. Для всех инструкций

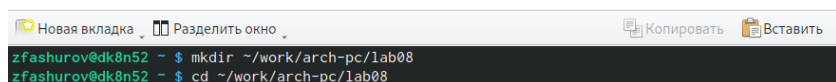
максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`. Она позволяет организовать безусловный цикл, типичная структура которого имеет следующий вид:

`mov ecx, 100` ; Количество проходов `NextStep: ...` ; тело цикла `... loop NextStep` ; Повторить `ecx` раз от метки `NextStep`

Иструкция `loop` выполняется в два этапа. Сначала из регистра `ecx` вычитается единица, его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.

4 Выполнение лабораторной работы

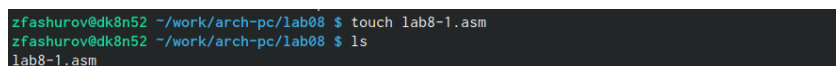
Создаем папку и переходим в него (Рис. 4.1)



```
Новая вкладка  Разделить окно  Копировать  Вставить
zfashurov@dk8n52 ~ $ mkdir ~/work/arch-pc/lab08
zfashurov@dk8n52 ~ $ cd ~/work/arch-pc/lab08
```

Рис. 4.1: Создание папки и переход в него

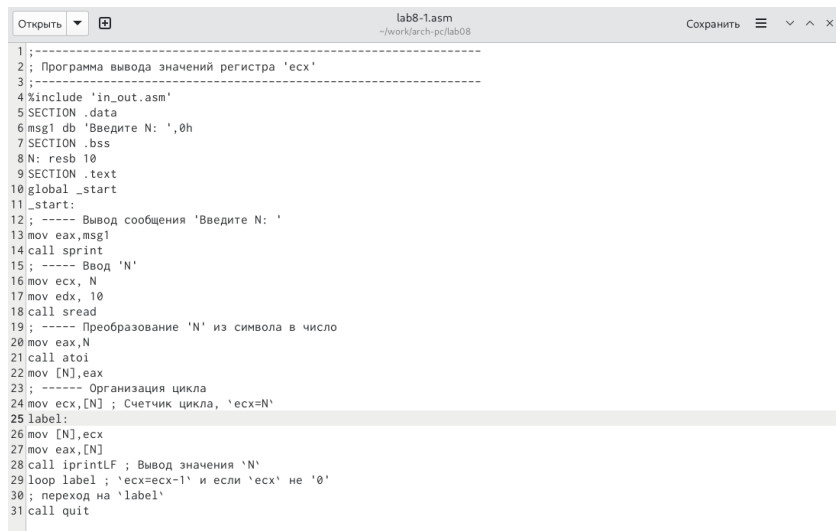
Создаем файл lab8-1.asm (Рис. 4.2).



```
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ touch lab8-1.asm
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ ls
lab8-1.asm
```

Рис. 4.2: Создание файла lab8-1.asm

Вписываем текст из листинга в lab8-1.asm (Рис. 4.3).



```
Открыть  lab8-1.asm  Сохранить
~/work/arch-pc/lab08

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

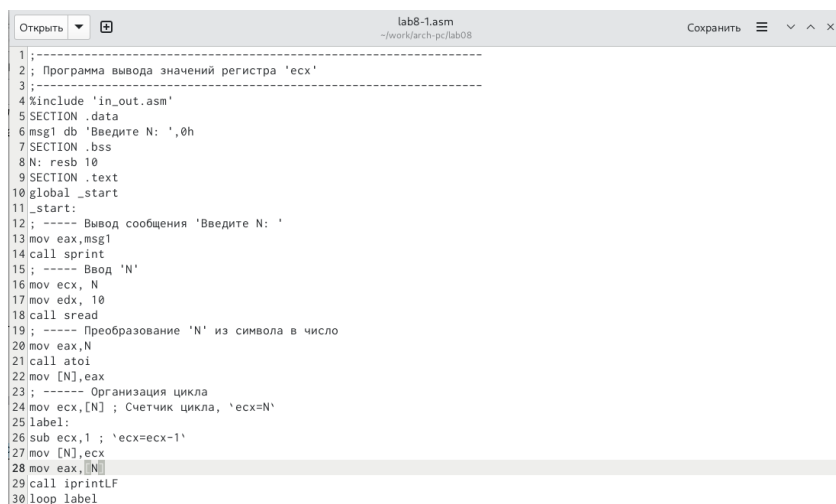
Рис. 4.3: Вписывание текста из листинга в lab8-1.asm

Некорректный запуск (Рис. 4.4).

```
zfashurov@dk8n75 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08/report $ nasm -f elf lab8-1.asm
nasm: fatal: unable to open input file 'lab8-1.asm' No such file or directory
```

Рис. 4.4: Некорректный запуск

Редактируем файл lab8-1.asm (Рис. 4.5).



```
1;-----
2; Программа вывода значений регистра 'ecx'
3;-----
4%include 'in_out.asm'
5SECTION .data
6msg1 db 'Введите N: ',0h
7SECTION .bss
8N: resb 10
9SECTION .text
10global _start
11_start:
12; ---- Вывод сообщения 'Введите N: '
13mov eax,msg1
14call sprint
15; ---- Ввод 'N'
16mov ecx, N
17mov edx, 10
18call sread
19; ---- Преобразование 'N' из символа в число
20mov eax,N
21call atoi
22mov [N],eax
23; ---- Организация цикла
24mov ecx,[N] ; Счетчик цикла, 'ecx=N'
25label:
26sub ecx,1 ; 'ecx=ecx-1'
27mov [N],ecx
28mov eax,[N]
29call iprintf
30loop label
```

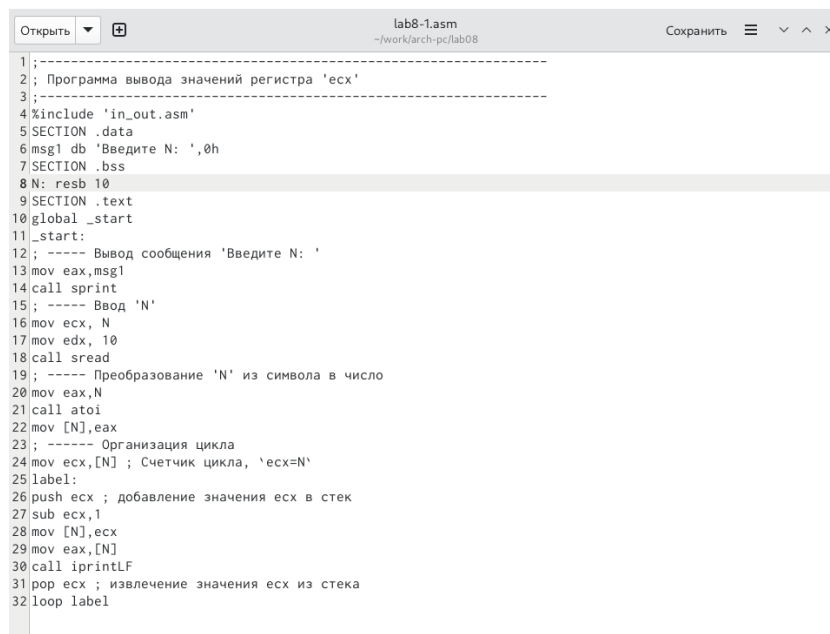
Рис. 4.5: Редактирование файла lab8-1.asm

Создаем и запускаем исполняемый файл (Рис. 4.6).

```
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 4.6: Создание и запускание исполняемого файла

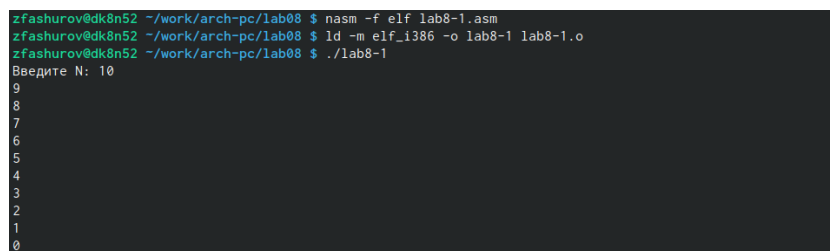
Вносим изменения в файл lab8-1.asm(Рис. 4.7).



```
1;-----  
2; Программа вывода значений регистра 'ecx'  
3;-----  
4#include 'in_out.asm'  
5SECTION .data  
6msg1 db 'Введите N: ',0h  
7SECTION .bss  
8N: resb 10  
9SECTION .text  
10global _start  
11_start:  
12; ----- Вывод сообщения 'Введите N: '  
13mov eax,msg1  
14call sprint  
15; ----- Ввод 'N'  
16mov ecx, N  
17mov edx, 10  
18call sread  
19; ----- Преобразование 'N' из символа в число  
20mov eax,N  
21call atoi  
22mov [N],eax  
23; ----- Организация цикла  
24mov ecx,[N] ; Счетчик цикла, 'ecx=N'  
25label:  
26push ecx ; добавление значения ecx в стек  
27sub ecx,1  
28mov [N],ecx  
29mov eax,[N]  
30call iprintLF  
31pop ecx ; извлечение значения ecx из стека  
32loop label
```

Рис. 4.7: Внесения изменения в lab8-1.asm

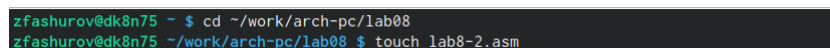
Создаем и запускаем исполняемый файл (Рис. 4.8).



```
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm  
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o  
zfashurov@dk8n52 ~/work/arch-pc/lab08 $ ./lab8-1  
Введите N: 10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
0
```

Рис. 4.8: Создание и запускание исполняемого файл

Создаем файл lab8-2.asm (Рис. 4.9).



```
zfashurov@dk8n75 ~ $ cd ~/work/arch-pc/lab08  
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ touch lab8-2.asm
```

Рис. 4.9: Создание файла lab8-2.asm

Вписываем текст из листинга в файл lab8-2.asm(Рис. 4.10).

```
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ gedit lab8-2.asm
lab8-2.asm
~/work/arch-pc/lab08
Сохранить

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

Рис. 4.10: Вписывание текста из листинга в файл lab8-2.asm

Создаем и запускаем исполняемый файл (Рис. 4.11).

```
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ./lab8-2
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.11: Создание и запускание исполняемого файла

Создаем и редактируем файл lab8-3.asm (Рис. 4.12).

```
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ touch lab8-3.asm
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ gedit lab8-3.asm

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

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. 4.12: Создание и редактирование файла lab8-3.asm

Создаем и запускаем исполняемый файл (Рис. 4.13).

```
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.13: Создание и запускание исполняемого файла

5 Самостоятельная работы

Создаем файл lab8-4.asm (Рис. 5.1).

```
zfashurov@dk8n75 ~ $ cd ~/work/arch-pc/lab08
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ touch lab8-4.asm
```

Рис. 5.1: Создание файла lab8-4.asm

Записываем код (Рис. 5.2).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/z/f/zfashurov/work/arch-pc/lab08/lab8-4.asm
#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,5
next:
cmp ecx,0h
jz _end
pop eax
call atoi
add eax,2
mul edi
add esi,eax
loop next
_end:
mov eax,msg
call sprint
mov eax,esi
call iprintfLF
call quit
```

Рис. 5.2: Записывание кода

Создаем и запускаем исполняемый файл (Рис. 5.3).

```
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
zfashurov@dk8n75 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Результат: 90
```

Рис. 5.3: Создание и запускание исполняемого файла

6 Выводы

Я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы