

Лабораторная работа №8

Простейший вариант

Сырцева Анастасия Романовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Самостоятельная работа	20
6	Выводы	24

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Копирование внешнего файла	8
4.3	Текст программы вывода значений регистра еsx	9
4.4	Создание исполняемого файла	10
4.5	Результат работы программы	10
4.6	Изменённая часть программы	10
4.7	Результат работы изменённой программы	11
4.8	Изменённая часть текста программы	12
4.9	Создание исполняемого файла	12
4.10	Запуск исполняемого файла изменённой программы	13
4.11	Создание файла	13
4.12	Программа, выводящей на экран аргументы командной строки . .	14
4.13	Результат работы программы	15
4.14	Создание нового файла	15
4.15	Программы для вычисления суммы аргументов командной строки	16
4.16	Результат работы программы. Сумма аргументов командной строки	17
4.17	Программы для вычисления произведения аргументов командной строки	18
4.18	Результат работы программы	19
4.19	Проверка работы программы с другими аргументами	19
5.1	Создание файла lab8-4.asm	20
5.2	Мой вариант самостоятельной работы	20
5.3	Программа вычисления суммы функций	21
5.4	Результат работы программы с несколькими аргументами	23

Список таблиц

1 Цель работы

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

2 Задание

- Изучение программы с использованием циклов
- Знакомство с использованием аргументов командной строки
- Обработка аргументов командной строки

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). 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. Извлечённый из стека элемент не стирается из памяти, а будет перезаписан при записи нового значения в стек.

4 Выполнение лабораторной работы

Создаю рабочую папку для лабораторной работы №8, перехожу в неё и создаю файл lab8-1.asm(рис. 4.1).

```
arsihrceva@dk3n55 ~/work/arch-pc $ mkdir lab08
arsihrceva@dk3n55 ~/work/arch-pc $ cd lab08
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 4.1: Создание каталога и файла

Внимательно изучаю листинг программы вывода значений регистра еsx. Для корректной работы копирую внешний файл in_out.asm в рабочий каталог(рис. 4.2).

```
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ cp ~/work/in_out.asm ~/work/arch-pc/lab08
```

Рис. 4.2: Копирование внешнего файла

Открываю файл и ввожу текст упомянутой программы(рис. 4.3).


```

1 %include 'in_out.asm'
2
3 SECTION .data
4     msg1 db 'Введите N: ',0h
5
6 SECTION .bss
7     N: resb 10
8
9 SECTION .text
10    global _start
11 _start:
12
13    mov eax,msg1
14    call sprint
15
16    mov ecx,N
17    mov edx, 10
18    call sread
19
20    mov eax,N
21    call atoi
22    mov[N],eax
23
24    mov ecx,[N]
25 label:
26    mov [N], ecx
27    mov eax,[N]
28    call iprintLF
29    loop label
30
31    call quit

```

Рис. 4.3: Текст программы вывода значений регистра ecx

Создаю исполняемый файл(рис. 4.4).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
```

Рис. 4.4: Создание исполняемого файла

Проверяю результат работы программы(рис. 4.5).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 7
7
6
5
4
3
2
1
```

Рис. 4.5: Результат работы программы

Изменяю текст программы, добавив изменение значения регистра ecx в цикле(рис. 4.6).

```
25 label:
26         sub ecx, 1
27         mov [N], ecx
28         mov eax, [N]
29         call iprintLF
30         loop label
```

Рис. 4.6: Изменённая часть программы

Создаю и запускаю исполняемый файл. В результате получается бесконечный цикл(рис. 4.7).

4289746058
4289746056
4289746054
4289746052
4289746050
4289746048
4289746046
4289746044
4289746042
4289746040
4289746038
4289746036
4289746034
4289746032
4289746030
4289746028
4289746026
42

Рис. 4.7: Результат работы изменённой программы

Вношу изменения в текст программы, добавив команды push и pop для сохранения значения счетчика цикла loop(рис. 4.8).

```
25 label:
26         push ecx
27         sub ecx, 1
28         mov [N], ecx
29         mov eax, [N]
30         call iprintLF
31         pop ecx
32
33         loop label
```

Рис. 4.8: Изменённая часть текста программы

Создаю исполняемый файл изменённой программы(рис. 4.9).

```
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
```

Рис. 4.9: Создание исполняемого файла

Запускаю его и проверяю работу(рис. 4.10).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0
```

Рис. 4.10: Запуск исполняемого файла изменённой программы

В данном случае число проходов цикла соответствует значению, введённому с клавиатуры. Создаю файл lab8-2.asm(рис. 4.11).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-2.asm
```

Рис. 4.11: Создание файла

Внимательно изучаю текст программы из условия лабораторной работы. Открываю созданный файл для редактирования и ввожу текст программы, выводящей на экран аргументы командной строки(рис. 4.12).

```

1 %include 'in_out.asm'
2
3 SECTION .text
4 global _start
5
6 _start:
7     pop ecx
8
9     pop edx
10
11     sub ecx, 1
12
13 next:
14     cmp ecx, 0
15     jz _end
16
17     pop eax
18     call sprintf
19     loop next
20
21 _end:
22     call quit

```

Рис. 4.12: Программа, выводящей на экран аргументы командной строки

Создаю исполняемый файл. При его запуске указываю аргументы: аргумент1 аргумент 2 'аргумент 3'(рис. 4.13).

```

arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3

```

Рис. 4.13: Результат работы программы

Команда обработала 4 аргумента. “аргумент” и “2” считаются разными аргументами, так как между ними стоит пробел. Создаю файл lab8-3.asm для следующей программы(рис. 4.14).

```

arsihrceva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-3.asm

```

Рис. 4.14: Создание нового файла

В созданный файл ввожу текст программы для вычисления суммы аргументов командной строки(рис. 4.15).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11     pop ecx
12
13     pop edx
14
15     sub ecx,1
16
17     mov esi, 0
18
19 next:
20     cmp ecx,0h
21     jz _end
22
23     pop eax
24     call atoi
25     add esi,eax
26
27     loop next
28
29 _end:
30     mov eax, msg
31     call sprint
32     mov eax, esi
33     call iprintLF
34     call quit

```

Рис. 4.15: Программы для вычисления суммы аргументов командной строки

Создаю и запускаю исполняемый файл. При запуске указываю аргументы: 12, 13, 7, 10, 5(рис. 4.16).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 4.16: Результат работы программы. Сумма аргументов командной строки

$12+13+7+10+5=47$. Это совпадает с результатом программы, следовательно она работает правильно. Изменяю текст программы в файле lab8-3.asm так, чтобы программы вычисляла произведение аргументов командной строки(рис. 4.17).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11     pop ecx
12
13     pop edx
14
15     sub ecx,1
16
17     mov esi,1
18     mov eax,1
19
20 next:
21     cmp ecx,0
22     jz _end
23
24     pop eax
25     call atoi
26     mov ebx,eax
27     mov eax,esi
28     imul ebx
29     mov esi,eax
30
31     loop next
32
33 _end:
34     mov eax, msg
35     call sprint
36     mov eax, esi
37     call iprintLF
38     call quit

```

Рис. 4.17: Программы для вычисления произведения аргументов командной строки

Создаю исполняемый файл. При его запуске указываю некоторые аргументы(рис. 4.18), запускаю ещё раз с другими аргументами для проверки(рис. 4.19).

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 2 4
Результат: 8
```

Рис. 4.18: Результат работы программы

```
arsihrcева@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 3 4 5
Результат: 60
```

Рис. 4.19: Проверка работы программы с другими аргументами

5 Самостоятельная работа

Создаю файл lab8-4.asm для выполнения самостоятельной работы(рис. 5.1).

```
arsihrcuva@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-4.asm
```

Рис. 5.1: Создание файла lab8-4.asm

Нахожу свой вариант из 7 лабораторной работы. Это вариант 1(рис. 5.2).

Номер варианта	$f(x)$
1	$2x + 15$

Рис. 5.2: Мой вариант самостоятельной работы

Открываю файл и ввожу текст программы для вычисления суммы функций вида $f(x)=2x+15$, где x - аргументы командной строки(рис. 5.3).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11     pop ecx
12
13     pop edx
14
15     sub ecx,1
16
17     mov esi,0
18
19 next:
20     cmp ecx,0
21     jz _end
22
23     mov ebx,2
24     pop eax
25     call atoi
26     mul ebx
27     add eax,15
28     add esi,eax
29
30     loop next
31
32 _end:
33     mov eax, msg
34     call sprint
35     mov eax, esi
36     call iprintLF
37     call quit

```

Рис. 5.3: Программа вычисления суммы функций

Создаю и запускаю исполняемый файл. При запуске указываю некоторые аргументы. Для проверки запускаю программу ещё раз и ввожу новые аргументы(рис. 5.4).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg db "Результат: ",0
5
6 SECTION .text
7 global _start
8
9 _start:
10
11     pop ecx
12
13     pop edx
14
15     sub ecx,1
16
17     mov esi,0
18
19 next:
20     cmp ecx,0
21     jz _end
22
23     mov ebx,2
24     pop eax
25     call atoi
26     mul ebx
27     add eax,15
28     add esi,eax
29
30     loop next
31
32 _end:
33     mov eax, msg
34     call sprint
35     mov eax, esi
36     call iprintLF
37     call quit

```

Рис. 5.4: Результат работы программы с несколькими аргументами

6 Выводы

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