

Отчёт по лабораторной работе №8

Архитектура компьютеров и операционные системы.

Брыляков Никита Евгеньевич

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	12
4.3	Выполнение заданий для самостоятельной работы	14
5	Вывод	16
6	Список литературы	17

Список иллюстраций

4.1	Создание каталога и файла внутри	7
4.2	Ввод программы	8
4.3	Создание и запуск	8
4.4	Изменение программы	9
4.5	Создание и запуск	10
4.6	Изменение программы	11
4.7	Создание и запуск	11
4.8	Ввод программы	12
4.9	Создание и запуск	12
4.10	Ввод программы	13
4.11	Создание и запуск	13
4.12	Изменение программы	14
4.13	Создание и запуск	14
4.14	Ввод программы	15
4.15	Создание и запуск	15

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. Выполнение заданий для самостоятельной работы

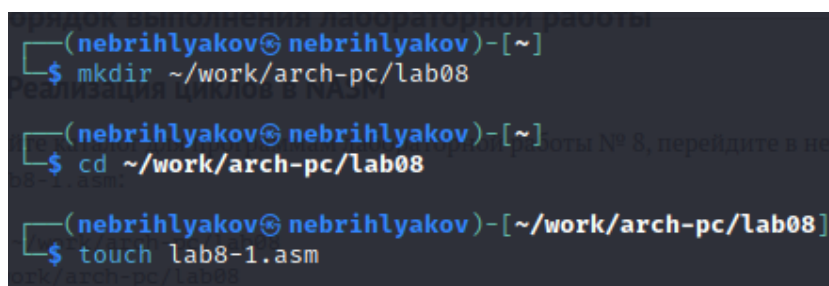
3 Теоретическое введение

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. 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. 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. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm. (рис. [4.1]).



```
(nebrihlyakov@nebrihlyakov)-[~]  
$ mkdir ~/work/arch-pc/lab08  
  
(nebrihlyakov@nebrihlyakov)-[~]  
$ cd ~/work/arch-pc/lab08  
  
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]  
$ touch lab8-1.asm
```

Рис. 4.1: Создание каталога и файла внутри

Ввожу в файл lab8-1.asm текст программы из листинга 8.1. (рис. [4.2]).

```

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
29 |

```

Рис. 4.2: Ввод программы

Создаю исполняемый файл и проверяю его работу. (рис. [4.3]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ nasm -f elf lab8-1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-1 lab8-1.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ./lab8-1
Введите N: 7
7
6
5
4
3
2
1

```

Рис. 4.3: Создание и запуск

Изменяю текст программы, добавив изменение значения регистра ecx в цикле. (рис. [4.4]).


```

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
24 mov [N],ecx
25 mov eax,[N]
26 call iprintLF ; Вывод значения `N`
27 loop label ; `ecx=ecx-1` и если `ecx` не '0'
28 ; переход на `label`
29 call quit
30

```

Рис. 4.4: Изменение программы

Создаю исполняемый файл и запускаю его. (рис. [4.5]).

```
4288286608
4288286606
4288286604
4288286602
4288286600
4288286598
4288286596
4288286594
4288286592
4288286590
4288286588
4288286586
4288286584
4288286582
4288286580
4288286578
4288286576
4288286574
4288286572
4288286570
4288286568
4288286566
4288286564
4288286562
4288286560
4288286558
4288286556
4288286554
```

Рис. 4.5: Создание и запуск

Число проходов цикла не соответствует введённому с клавиатуры значению.

Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчика цикла `loop`. (рис. [4.6]).

```

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
31

```

Рис. 4.6: Изменение программы

Создаю исполняемый файл и запускаю его. (рис. [4.7]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ nasm -f elf lab8-1.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-1 lab8-1.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0

```

Рис. 4.7: Создание и запуск

Число проходов цикла соответствует введённому с клавиатуры значению. Вы-

водит числа от N-1 до 0 включительно.

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.2. (рис. [4.8]).

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

Рис. 4.8: Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. [4.9]).

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

Рис. 4.9: Создание и запуск

Программой было выведено 4 аргумента.

Создаю файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и ввожу в него текст программы из листинга 8.3. (рис. [4.10]).

```
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
30
```

Рис. 4.10: Ввод программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. [4.11]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ nasm -f elf lab8-3.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ./lab8-3 12 23 34
Результат: 69
```

Рис. 4.11: Создание и запуск

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. (рис. [4.12]).

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 SECTION .text
5 global _start
6 _start:
7 pop ecx
8 pop edx ; Извлекаем из стека в `edx` имя программы (второе значение в стеке
9 sub ecx,1 ; Уменьшаем `ecx` на 1 (количество аргументов без названия программ
10 mov esi, 1 ; Используем `esi` для хранения промежуточного результата
11 next:
12 cmp ecx,0h ; проверяем, есть ли еще аргументы
13 jz _end ; если аргументов нет выходим из цикла (переход на метку `_end`)
14 pop eax ; иначе извлекаем следующий аргумент из стека
15 call atoi ; преобразуем символ в число
16 imul esi, eax ; умножаем промежуточный результат на след. аргумент `esi=esi*ea
17 loop next ; переход к обработке следующего аргумента
18 _end:
19 mov eax, msg ; вывод сообщения "Результат: "
20 call sprint
21 mov eax, esi ; записываем сумму в регистр `eax`
22 call iprintf ; печать результата
23 call quit
24
```

Рис. 4.12: Изменение программы

Создаю исполняемый файл и запускаю его, указав аргументы. (рис. [4.13]).

```
(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab08]
$ nasm -f elf lab8-3.asm

(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o lab8-3 lab8-3.o

(nebrihlyakov@nebrihlyakov)~[~/work/arch-pc/lab08]
$ ./lab8-3 2 3 4
Результат: 24
```

Рис. 4.13: Создание и запуск

4.3 Выполнение заданий для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x) = 4x+3$ в соответствии с моим 5 вариантом для $x = x_1, x_2, \dots, x_n$. Значения x_i передаются как аргументы. (рис. [4.14]).

```

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

```

Рис. 4.14: Ввод программы

Создаю исполняемый файл и проверяю его работу на нескольких наборах х (рис. [4.15]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ nasm -f elf zadanie.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ld -m elf_i386 -o zadanie zadanie.o

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ./zadanie 4 6 8
Результат: 81

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab08]
$ ./zadanie 1 7 12 3
Результат: 104

```

Рис. 4.15: Создание и запуск

Всё работает верно.

5 Вывод

При выполнении данной лабораторной работы я приобрёл навыки написания программ с использованием циклов и обработкой аргументов командной строки.

6 Список литературы

- [illegible]