

Отчёт по лабораторной работе 8

Архитектура компьютера

Сувд Адисурэн

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация циклов в NASM	6
2.2	Самостоятельное задание	16
3	Выводы	19

Список иллюстраций

2.1	Создание каталога	6
2.2	Программа lab8-1.asm	7
2.3	Запуск программы lab8-1.asm	8
2.4	Измененная программа lab8-1.asm	9
2.5	Запуск измененной программы lab8-1.asm	10
2.6	Исправленная программа lab8-1.asm	11
2.7	Запуск исправленной программы lab8-1.asm	12
2.8	Программа lab8-2.asm	13
2.9	Запуск программы lab8-2.asm	13
2.10	Программа lab8-3.asm	14
2.11	Запуск программы lab8-3.asm	14
2.12	Программа lab8-3.asm	15
2.13	Запуск программы lab8-3.asm	15
2.14	Программа task.asm	17
2.15	Запуск программы task.asm	18

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация циклов в NASM

Создала каталог для программ лабораторной работы № 8 и файл lab8-1.asm (рис. 2.1).

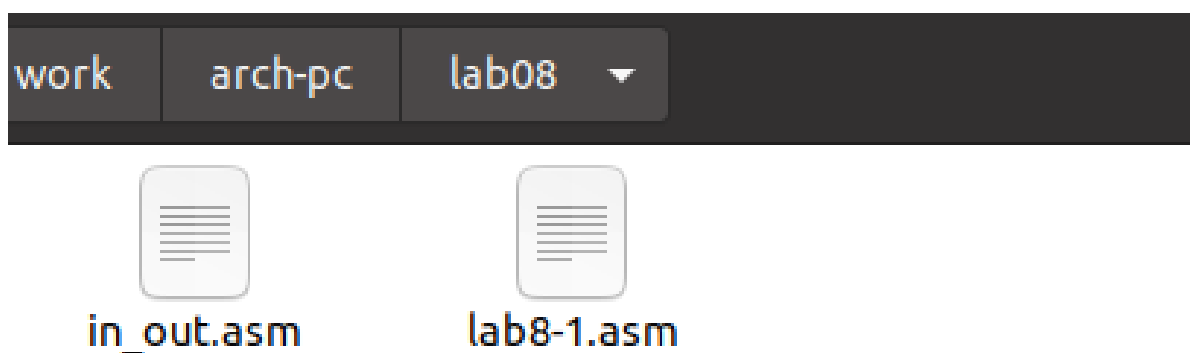


Рис. 2.1: Создание каталога

При реализации циклов в NASM с использованием инструкции `loop` важно помнить, что эта инструкция использует регистр `ecx` в качестве счетчика и на каждом шаге уменьшает его значение на единицу. В качестве примера рассмотрим программу, которая выводит значение регистра `ecx`.

Написала в файл lab8-1.asm текст программы из листинга 8.1. (рис. 2.2).
Создала исполняемый файл и проверила его работу. (рис. 2.3).

```
lab8-1.asm
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. 2.2: Программа lab8-1.asm

```
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 3
3
2
1
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-1 4
Введите N: 4
4
3
2
1
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.3: Запуск программы lab8-1.asm

Этот пример демонстрирует, что использование регистра `ecx` в теле цикла `loop` может привести к некорректной работе программы. Изменила текст программы, добавив изменение значения регистра `ecx` в цикле. (рис. 2.4). Программа теперь запускает бесконечный цикл при нечетном `N` и выводит только нечетные числа при четном `N`. (рис. 2.5).


```

lab8-1.asm
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. 2.4: Измененная программа lab8-1.asm

```
4294950956
4294950954
4294950952
4294950950
4294950948
4294950946
4294^C
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 4
3
1
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.5: Запуск измененной программы lab8-1.asm

Для корректной работы программы с регистром `ecx` в цикле можно использовать стек. Внесла изменения в текст программы, добавив команды `push` и `pop` (для добавления в стек и извлечения из него значений), чтобы сохранить значение счетчика цикла `loop`. (рис. 2.6). Создала исполняемый файл и проверила его работу. (рис. 2.7). Программа теперь выводит числа от $N-1$ до 0, при этом число проходов цикла соответствует значению N .

```

lab8-1.asm
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. 2.6: Исправленная программа lab8-1.asm

```
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$  
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm  
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-1.o -o lab8-1  
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 4  
3  
2  
1  
0  
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-1  
Введите N: 3  
2  
1  
0  
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.7: Запуск исправленной программы lab8-1.asm

Создала файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и написала в него текст программы из листинга 8.2. (рис. 2.8). Компилирую исполняемый файл и запускаю его, указав аргументы. Программа обработала 4 аргумента. Аргументами считаются слова/числа, разделенные пробелом. (рис. 2.9).

```

lab8-2.asm
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 sprintf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку `next`)
19 _end:
20 call quit

```

Рис. 2.8: Программа lab8-2.asm

```

suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-2.o -o lab8-2
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-2
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-2 1 2 3
1
2
3
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$

```

Рис. 2.9: Запуск программы lab8-2.asm

Рассмотрим еще один пример программы, которая выводит сумму чисел, передаваемых в программу как аргументы. (рис. 2.10) (рис. 2.11).

```

lab8-3.asm
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. 2.10: Программа lab8-3.asm

```

suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 3 4 5 6
Результат: 18
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 12 41 37
Результат: 90
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$

```

Рис. 2.11: Запуск программы lab8-3.asm

Изменила текст программы из листинга 8.3 для вычисления произведения

аргументов командной строки. (рис. 2.12) (рис. 2.13).

```
lab8-3.asm
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, 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. 2.12: Программа lab8-3.asm

```
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 lab8-3.o -o lab8-3
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 3 4 5 6
Результат: 360
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./lab8-3 12 41 37
Результат: 18204
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
```

Рис. 2.13: Запуск программы lab8-3.asm

2.2 Самостоятельное задание

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 $f(x)$ следует выбрать согласно таблице 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создала исполняемый файл и проверила его работу на нескольких наборах x . (рис. 2.14) (рис. 2.15).

Для варианта 4

$$f(x) = 2(x - 1)$$


```

task.asm
1  %include 'in_out.asm'
2  SECTION .data
3  msg db "Результат: ",0
4  fx: db 'f(x)= 2(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,2
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. 2.14: Программа task.asm

```

suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ nasm -f elf task.asm
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ld -m elf_i386 task.o -o task
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./task 2
f(x)= 2(x - 1)
Результат: 2
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./task 1
f(x)= 2(x - 1)
Результат: 0
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./task 2
f(x)= 2(x - 1)
Результат: 2
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./task 5
f(x)= 2(x - 1)
Результат: 8
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$ ./task 3 1 3 6 4 1
f(x)= 2(x - 1)
Результат: 24
suvdadiasuren@Ubuntu:~/work/arch-pc/lab08$

```

Рис. 2.15: Запуск программы task.asm

Убедилась, что программа правильно вычисляет $f(1) = 0$, $f(5) = 8$.

3 Выводы

Освоила работу со стеком, циклами и аргументами на ассемблере NASM.