

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Чекмарев Александр Дмитриевич | группа: НПИбд 02-23

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация циклов в NASM	6
2.2	Обработка аргументов командной строки	14
3	Самостоятельная работа	19
4	Выводы	22

Список иллюстраций

2.1	Рис 2.1.1: Создание каталога и файла .asm	6
2.2	Рис 2.1.2: Демонстрация текста программы в файле	7
2.3	Рис 2.1.3: Создание файла и его проверка	8
2.4	Рис 2.1.4: Демонстрация измененного текста программы	9
2.5	Рис 2.1.5: Создание файла и его проверка (1)	10
2.6	Рис 2.1.6: Проверка с другим значением N=5 (2)	11
2.7	Рис 2.1.7: Демонстрация измененного текста программы	13
2.8	Рис 2.1.8: Создание файла и его проверка	14
2.9	Рис 2.2.1: Создание файла .asm	15
2.10	Рис 2.2.2: Демонстрация текста программы	15
2.11	Рис 2.2.3: Создание файла и его проверка	16
2.12	Рис 2.2.4: Создание файла .asm	16
2.13	Рис 2.2.5: Демонстрация текста программы	17
2.14	Рис 2.2.6: Создание файла и его проверка с указанием аргументов	17
2.15	Рис 2.2.7: Демонстрация измененного текста программы	18
2.16	Рис 2.2.8: Создание файла и его проверка	18
3.1	Рис 3.1.1: Создание файла	19
3.2	Рис 3.1.2: Демонстрация программы для задания	20
3.3	Рис 3.1.3: Проверка программы	21

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация циклов в NASM

Создадим каталог для программ лабораторной работы № 8, перейдем в него и создадим файл *lab8-1.asm*:

```
adchekmarev@alexanderchekmarev:~$ mkdir ~/work/arch-pc/lab08
adchekmarev@alexanderchekmarev:~$ cd ~/work/arch-pc/lab08
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch lab8-1.asm
```

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .asm

При реализации циклов в NASM с использованием инструкции `loop` необходимо помнить о том, 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`. Внимательно изучим текст программы (Листинг 8.1).

Введем в файл *lab8-1.asm* текст программы из листинга 8.1.

report.md	×	lab8-1.asm
<pre>1 ;----- 2 ; Программа вывода значений регистра 'ecx' 3 ;----- 4 5 %include 'in_out.asm' 6 7 SECTION .data 8 msg1 db 'Введите N: ',0h 9 10 SECTION .bss 11 N: resb 10 12 13 SECTION .text 14 global _start 15 _start: 16 17 ; ----- Вывод сообщения 'Введите N: ' 18 mov eax,msg1 19 call sprint 20 21 ; ----- Ввод 'N' 22 mov ecx, N 23 mov edx, 10 24 call sread 25 26 ; ----- Преобразование 'N' из символа в число 27 mov eax,N 28 call atoi 29 mov [N],eax 30 31 ; ----- Организация цикла 32 mov ecx,[N] ; Счетчик цикла, `ecx=N` 33 label: 34 mov [N],ecx 35 mov eax,[N] 36 call iprintLF ; Вывод значения `N` 37 loop label ; `ecx=ecx-1` и если `ecx` не `0` 38 ; переход на `label` 39 call quit 40</pre>		

Рис. 2.2: Рис 2.1.2: Демонстрация текста программы в файле

Создадим исполняемый файл и запустим его. Результат работы данной программы будет следующим:

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

Рис. 2.3: Рис 2.1.3: Создание файла и его проверка

Данный пример показывает, что использование регистра `ecx` в теле цикла *loop* может привести к некорректной работе программы.

Изменим текст программы добавив изменение значение регистра `ecx` в цикле:


```

2 ; Программа вывода значений регистра 'ecx'
3 ;-----
4
5 %include 'in_out.asm'
6
7 SECTION .data
8     msg1 db 'Введите N: ',0h
9
10 SECTION .bss
11     N: resb 10
12
13 SECTION .text
14     global _start
15 _start:
16
17 ; ----- Вывод сообщения 'Введите N: '
18     mov eax,msg1
19     call sprint
20
21 ; ----- Ввод 'N'
22     mov ecx, N
23     mov edx, 10
24     call sread
25
26 ; ----- Преобразование 'N' из символа в число
27     mov eax,N
28     call atoi
29     mov [N],eax
30
31 ; ----- Организация цикла
32     mov ecx,[N] ; Счетчик цикла, `ecx=N`
33 label:
34     sub ecx,1 ; `ecx=ecx-1`
35     mov [N],ecx
36     mov eax,[N]
37     call iprintLF
38
39     loop label
40
41     call quit
42

```

Рис. 2.4: Рис 2.1.4: Демонстрация измененного текста программы

Создадим исполняемый файл и проверим его работу.

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

Рис. 2.5: Рис 2.1.5: Создание файла и его проверка (1)

Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению `N` введенному с клавиатуры?

Если ввести 2, то подсчет будет до 1, что не соответствует. Если к примеру ввести 5, то это явно число проходов цикла НЕ будет соответствовать введенному с клавиатуры значению

4289298308
4289298306
4289298304
4289298302
4289298300
4289298298
4289298296
4289298294
4289298292
4289298290
4289298288
4289298286
4289298284
4289298282
4289298280
4289298278
4289298276
4289298274
4289298272
4289298270
4289298268
4289298266
4289298264
4289298262
4289298260
4289298258
4289298256
4289298254
4289298252
4289298250
4289298248
4289298246
4289298244
4289298242
4289298240
4289298238
4289298236
4289298234
4289298232
4289298230
428929822

Рис. 2.6: Рис 2.1.6: Проверка с другим значением N=5 (2)

Для использования регистра *ecx* в цикле и сохранения корректности работы программы можно использовать стек. Внесем изменения в текст программы добавив команды *push* и *pop* (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла *loop*:

```

5 %include 'in_out.asm'
6
7 SECTION .data
8     msg1 db 'Введите N: ',0h
9
10 SECTION .bss
11     N: resb 10
12
13 SECTION .text
14     global _start
15 _start:
16
17 ; ----- Вывод сообщения 'Введите N: '
18     mov eax,msg1
19     call sprint
20
21 ; ----- Ввод 'N'
22     mov ecx, N
23     mov edx, 10
24     call sread
25
26 ; ----- Преобразование 'N' из символа в число
27     mov eax,N
28     call atoi
29     mov [N],eax
30
31 ; ----- Организация цикла
32     mov ecx,[N]    ; Счетчик цикла, `ecx=N`
33 label:
34     push ecx        ; добавление значения ecx в стек
35     sub ecx,1
36     mov [N],ecx
37     mov eax,[N]
38     call iprintLF
39     pop ecx         ; извлечение значения ecx из стека
40
41     loop label
42
43     call quit
44

```

Рис. 2.7: Рис 2.1.7: Демонстрация измененного текста программы

Создадим исполняемый файл и проверим его работу.

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

Рис. 2.8: Рис 2.1.8: Создание файла и его проверка

Соответствует ли в данном случае число проходов цикла значению N введенному с клавиатуры?

Число проходов цикла будет соответствовать введенному значению с клавиатуры (1-(2), 0-(1))

2.2 Обработка аргументов командной строки

При разработке программ иногда встает необходимость указывать аргументы, которые будут использоваться в программе, непосредственно из командной строки при запуске программы.

При запуске программы в NASM аргументы командной строки загружаются в стек в обратном порядке, кроме того в стек записывается имя программы и общее количество аргументов. Последние два элемента стека для программы, скомпилированной NASM, – это всегда имя программы и количество переданных аргументов.

Таким образом, для того чтобы использовать аргументы в программе, их просто нужно извлечь из стека. Обработку аргументов нужно проводить в цикле. Т.е. сначала нужно извлечь из стека количество аргументов, а затем циклично для каждого аргумента выполнить логику программы. В качестве примера рассмотрим программу, которая выводит на экран аргументы командной строки. Внимательно изучим текст программы (Листинг 8.2)

Создадим файл *lab8-2.asm* в каталоге *~/work/arch-pc/lab08* и введем в него текст

программы из листинга 8.2.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 2.9: Рис 2.2.1: Создание файла .asm

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

Рис. 2.10: Рис 2.2.2: Демонстрация текста программы

Создадим исполняемый файл и запустим его, указав аргументы:

```

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

```

Рис. 2.11: Рис 2.2.3: Создание файла и его проверка

Сколько аргументов было обработано программой?

Всего было обработано 4 аргумента, так как “аргумент” и “2” не взяты в одинарные кавычки, в отличие от 4-го аргумента

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл *lab8-3.asm* в каталоге *~/work/arch-pc/lab08* и введем в него текст программы из листинга 8.3.

```

adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch lab8-3.asm

```

Рис. 2.12: Рис 2.2.4: Создание файла .asm


```

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  pop ecx      ; Извлекаем из стека в `ecx` количество
11               ; аргументов (первое значение в стеке)
12  pop edx      ; Извлекаем из стека в `edx` имя программы
13               ; (второе значение в стеке)
14  sub ecx,1    ; Уменьшаем `ecx` на 1 (количество
15               ; аргументов без названия программы)
16  mov esi, 0   ; Используем `esi` для хранения
17               ; промежуточных сумм
18 next:
19  cmp ecx,0h   ; проверяем, есть ли еще аргументы
20  jz _end      ; если аргументов нет выходим из цикла
21               ; (переход на метку `_end`)
22  pop eax      ; иначе извлекаем следующий аргумент из стека
23  call atoi    ; преобразуем символ в число
24  add 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. 2.13: Рис 2.2.5: Демонстрация текста программы

Создадим исполняемый файл и запустим его, указав аргументы.

```

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

```

Рис. 2.14: Рис 2.2.6: Создание файла и его проверка с указанием аргументов

Изменим текст программы из листинга 8.3 для вычисления произведения

аргументов командной строки.

```
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     pop ecx      ; Извлекаем из стека в `ecx` количество
11                  ; аргументов (первое значение в стеке)
12     pop edx      ; Извлекаем из стека в `edx` имя программы
13                  ; (второе значение в стеке)
14     sub ecx,1     ; Уменьшаем `ecx` на 1 (количество
15                  ; аргументов без названия программы)
16     mov esi, 1    ; Используем `esi` для хранения
17                  ; промежуточных сумм
18 next:
19     cmp ecx,0h    ; проверяем, есть ли еще аргументы
20     jz _end       ; если аргументов нет выходим из цикла
21                  ; (переход на метку `_end`)
22     pop eax       ; иначе извлекаем следующий аргумент из стека
23     call atoi     ; преобразуем символ в число
24     mul esi
25     mov esi,eax   ; добавляем к промежуточной сумме
26                  ; след. аргумент `esi=esi+eax`
27     loop next     ; переход к обработке следующего аргумента
28 _end:
29     mov eax, msg   ; вывод сообщения "Результат: "
30     call sprint
31     mov eax, esi   ; записываем сумму в регистр `eax`
32     call iprintLF  ; печать результата
33     call quit      ; завершение программы
```

Рис. 2.15: Рис 2.2.7: Демонстрация измененного текста программы

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 54600
```

Рис. 2.16: Рис 2.2.8: Создание файла и его проверка

3 Самостоятельная работа

Задание №1 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_i 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 $x = x_1, x_2, \dots, x_n$.

Создадим новый файл `task1.asm` и напомним программу нахождения суммы значения функции $f(x)$ для варианта 4 ($2(x-1)$).



```
alexanderchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch task1.asm
```

Рис. 3.1: Рис 3.1.1: Создание файла

Возьмем за основу код из `lab8-3.asm` и переделаем его

```

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     dec eax
26     mov ebx,2
27     mul ebx
28     add esi,eax
29
30     loop next
31 _end:
32     mov eax, msg
33     call sprintf
34     mov eax, esi
35     call iprintLF
36     call quit

```

Рис. 3.2: Рис 3.1.2: Демонстрация программы для задания

Проверим программу с несколькими значениями x

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ nasm -f elf task1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ld -m elf_i386 -o task1 task1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 4 2 3 14
Результат: 38
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 1 1 1 0
Результат: 4294967294
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 12 23 12
Результат: 88
```

Рис. 3.3: Рис 3.1.3: Проверка программы

Программа работает корректно

Загрузим все файлы на github по окончании лаб. работы

4 Выводы

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