

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 9

дисциплина: Архитектура компьютера

Студент: Волгин А.А.

Группа: НПИбд-01-22

МОСКВА

2022 г.

Цель работы:

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

Порядок выполнения лабораторной работы:

Реализация циклов в NASM.

Создадим каталог для программ лабораторной работы №9, перейдем в него и создадим нужный файл (рис. 1).

```
[aavolgin@fedora ~]$ mkdir ~/work/arch-pc/lab09  
[aavolgin@fedora ~]$ cd ~/work/arch-pc/lab09  
[aavolgin@fedora lab09]$ touch lab9-1.asm  
[aavolgin@fedora lab09]$
```

рис. 1. Создание файла lab9-1.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 (рис. 2).

```

#include 'in_out.asm'

SECTION .data
    msg1 db 'Введите N: ',0h

SECTION .bss
    N:      resb 10

SECTION .text
    global _start
_start:

; ----- Вывод сообщения 'Введите N: '
    mov    eax,msg1
    call    sprint

; ----- Ввод 'N'
    mov    ecx, N
    mov    edx, 10
    call    sread

; ----- Преобразование 'N' из символа в число
    mov    eax,N
    call    atoi
    mov    [N],eax

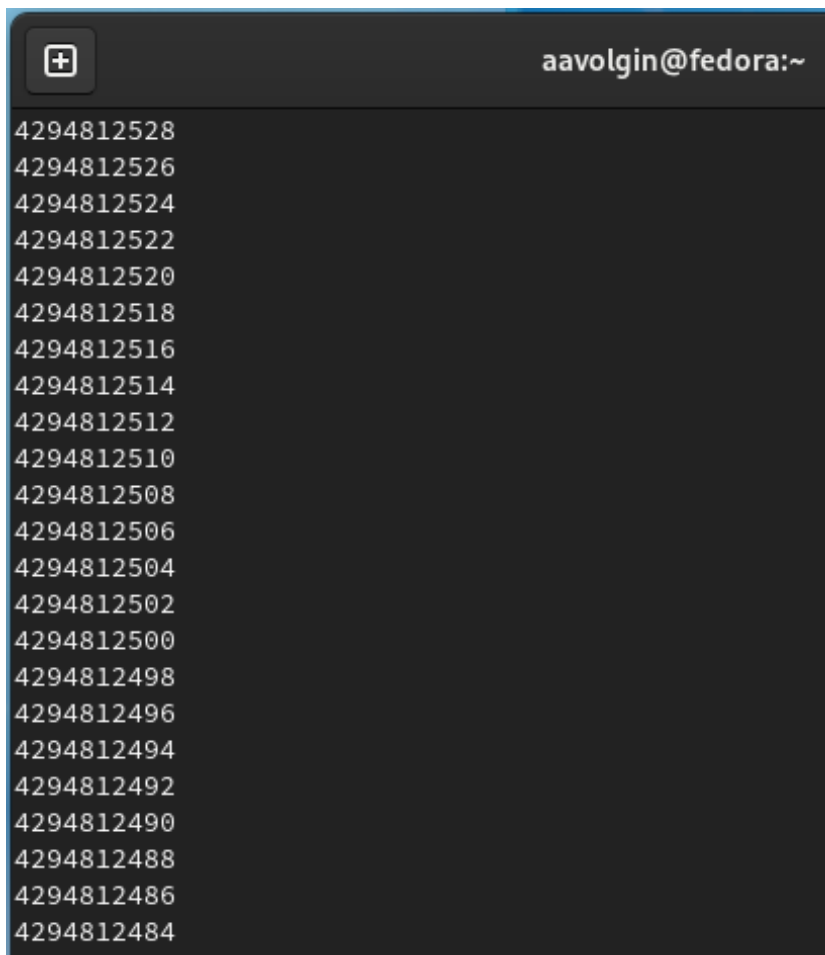
; ----- Организация цикла
    mov    ecx,[N]      ; Счетчик цикла, `ecx=N`
label:
    mov    [N],ecx
    mov    eax,[N]
    call    iprintLF    ; Вывод значения `N`
    loop   label        ; `ecx=ecx-1` и если `ecx` не `0`
                        ; переход на `label`

    call    quit

```

рис. 2. Текст программы lab9-1

Создадим исполняемый файл и проверим его работу (рис. 3).



```
aavolgin@fedora:~  
4294812528  
4294812526  
4294812524  
4294812522  
4294812520  
4294812518  
4294812516  
4294812514  
4294812512  
4294812510  
4294812508  
4294812506  
4294812504  
4294812502  
4294812500  
4294812498  
4294812496  
4294812494  
4294812492  
4294812490  
4294812488  
4294812486  
4294812484
```

рис. 3. Результат работы программы lab9-1

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 по следующему примеру (рис. 4).

```
label:  
    sub ecx,1      ; `ecx=ecx-1`  
    mov [N],ecx  
    mov eax,[N]  
    call iprintLF  
  
    loop label
```

рис. 4. Пример изменения части программы lab9-1

Создадим исполняемый файл и проверим его работу (рис. 5).

```
[aavolgin@fedora lab09]$ nasm -f elf lab9-1.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-1.o -o lab9-1
[aavolgin@fedora lab09]$ ./lab9-1
Введите N: 3
3
2
1
[aavolgin@fedora lab09]$
```

рис. 5. Результат работы измененной программы lab9-1

Как видим, все работает. Регистр `ecx` принимает все значения от `N` до 1 включительно, что соответствует числу проходов цикла, введенному с клавиатуры.

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Внесем изменения в текст программы по примеру, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 6).

```
label:
push ecx
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx

loop label
```

рис. 6. Внесение команд `push` и `pop` в текст программы lab9-1

Создадим исполняемый файл и проверим его работу (рис. 7).

```
[aavolgin@fedora lab09]$ nasm -f elf lab9-1.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-1.o -o lab9-1
[aavolgin@fedora lab09]$ ./lab9-1
Введите N: 4
4
3
2
1
[aavolgin@fedora lab09]$
```

рис. 7. Результат работы измененной программы lab9-1

Как видим, программа работает корректно, число проходов цикла соответствует значению N, введенному с клавиатуры.

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 (рис. 8). Создадим в каталоге лабораторной работы №9 файл lab9-2 и введем текст из рис. 8.

```

#include    'in_out.asm'

SECTION .text
global _start

_start:
    pop     ecx        ; Извлекаем из стека в `ecx` количество
                        ; аргументов (первое значение в стеке)
    pop     edx        ; Извлекаем из стека в `edx` имя программы
                        ; (второе значение в стеке)
    sub     ecx, 1      ; Уменьшаем `ecx` на 1 (количество
                        ; аргументов без названия программы)

next:
    cmp     ecx, 0      ; проверяем, есть ли еще аргументы
    jz      _end        ; если аргументов нет выходим из цикла
                        ; (переход на метку `_end`)
    pop     eax        ; иначе извлекаем аргумент из стека
    call    sprintfLF   ; вызываем функцию печати
    loop    next        ; переход к обработке следующего
                        ; аргумента (переход на метку `next`)

_end:
    call    quit

```

рис. 8. Текст программы lab9-2

Затем создадим исполняемый файл и запустим программу, указав следующие аргументы (рис. 9).

```

[aavolgin@fedora lab09]$ nasm -f elf lab9-2.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-2.o -o lab9-2
[aavolgin@fedora lab09]$ ./lab9-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[aavolgin@fedora lab09]$

```

рис. 9. Результат работы программы lab9-2

Как видим, программа восприняла “аргумент” и “2” как отдельные аргументы, в то время как ‘аргумент 3’ как один. Соответственно программой было обработано 4 аргумента.

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл lab9-3.asm в том же каталоге и введем в него следующий текст программы (рис. 10).

```
%include      'in_out.asm'

SECTION .data
msg db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx    ; Извлекаем из стека в `ecx` количество
                ; аргументов (первое значение в стеке)
    pop edx    ; Извлекаем из стека в `edx` имя программы
                ; (второе значение в стеке)
    sub ecx,1  ; Уменьшаем `ecx` на 1 (количество
                ; аргументов без названия программы)
    mov esi, 0 ; Используем `esi` для хранения
                ; промежуточных сумм

next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz  _end   ; если аргументов нет выходим из цикла
                ; (переход на метку `_end`)
    pop eax    ; иначе извлекаем следующий аргумент из стека
    call atoi  ; преобразуем символ в число
    add esi,eax ; добавляем к промежуточной сумме
                ; след. аргумент `esi=esi+eax`
    loop next  ; переход к обработке следующего аргумента

_end:
    mov eax,msg ; вывод сообщения "Результат: "
    call sprint
    mov eax,esi ; записываем сумму в регистр `eax`
    call iprintLF ; печать результата
    call quit    ; завершение программы
```

рис. 10. Текст программы lab9-3

Затем создадим исполняемый файл и запустим его, указав аргументы (рис. 11).

```

[aavolgin@fedora lab09]$ nasm -f elf lab9-3.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-3.o -o lab9-3
[aavolgin@fedora lab09]$ ./lab9-3
Результат: 0
[aavolgin@fedora lab09]$ ./lab9-3 12 13 7 10 5
Результат: 47
[aavolgin@fedora lab09]$

```

рис. 11. Результат работы программы lab9-3

Как видим, все работает корректно.

Изменим строку

add esi,ecx

на

move ebx, eax

mov eax, esi

mul ecx

mov esi, eax

а также присвоим esi значение 1, чтобы программа выводила произведение аргументов командной строки и запустим ее (рис. 12).

```

[aavolgin@fedora lab09]$ nasm -f elf lab9-3.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-3.o -o lab9-3
[aavolgin@fedora lab09]$ ./lab9-3 2 5
Результат: 10
[aavolgin@fedora lab09]$

```

рис. 12. Результат работы программы

Порядок выполнения самостоятельной работы:

Напишем 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 $f(x)$ выберем в соответствии с вариантом, полученным при выполнении лабораторной работы № 7 (вариант 15). Создадим исполняемый файл и проверим его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$.

$$f(x) = 6x + 13$$


```

lab9-sm.asm      [-----] 11 L: [
#include 'in_out.asm'

section .data
msg db "Результат: ",0
msgf db "Функция: f(x)=6x+13",0

section .text
global _start

_start:
pop ecx

pop edx

sub ecx,1

mov esi,0

next:
cmp ecx, 0h
jz _end

pop eax
call atoi
mov ebx,6
mul ebx
add eax,13
add esi,eax

loop next

_end:
mov eax, msgf
call sprintfLF
mov eax, msg
call sprintf
mov eax, esi
call iprintLF
call quit

```

рис. 13. Текст программы lab9-sm

```
[aavolgin@fedora lab09]$ nasm -f elf lab9-sm.asm
[aavolgin@fedora lab09]$ ld -m elf_i386 lab9-sm.o -o lab9-sm
[aavolgin@fedora lab09]$ ./lab9-sm
Функция:  $f(x)=6x+13$ 
Результат: 0
[aavolgin@fedora lab09]$ ./lab9-sm 1 2 3
Функция:  $f(x)=6x+13$ 
Результат: 75
[aavolgin@fedora lab09]$ ./lab9-sm 1 4 8 8
Функция:  $f(x)=6x+13$ 
Результат: 178
[aavolgin@fedora lab09]$ ./lab9-sm 9 11
Функция:  $f(x)=6x+13$ 
Результат: 146
[aavolgin@fedora lab09]$ ./lab9-sm 4 20
Функция:  $f(x)=6x+13$ 
Результат: 170
[aavolgin@fedora lab09]$
```

рис. 14. Результат работы программы lab9-sm

Все работает корректно.

Вывод:

Во время выполнения лабораторной работы были приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки.