

Лабораторная работа №9

Архитектура компьютера

Голованова Мария Константиновна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация циклов в NASM	8
4.2	Обработка аргументов командной строки	12
5	Задание для самостоятельной работы	18
6	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Создание каталога для программ лабораторной работы №9 и файла lab9-1.asm	8
4.2	Введение текста программы из листинга 9.1	9
4.3	Создание и запуск исполняемого файла lab9-1	10
4.4	Изменение текста файла lab9-1.asm	10
4.5	Результат работы изменённого файла lab9-1.asm	11
4.6	Изменение текста файла lab9-1.asm	12
4.7	Создание и запуск изменённого исполняемого файла lab9-1	12
4.8	Создание файла lab9-2.asm в каталоге ~/work/arch-pc/lab09	13
4.9	Введение текста программы из листинга 9.2	14
4.10	Создание и запуск исполняемого файла lab9-2	14
4.11	Создание файла lab9-3.asm в каталоге ~/work/arch-pc/lab09	15
4.12	Введение текста программы из листинга 9.3	15
4.13	Создание исполняемого файла lab9-3 и его запуск с указанием аргументов	16
4.14	Изменение текста файла lab9-3.asm для вычисления произведения аргументов командной строки	16
4.15	Создание нового исполняемого файла lab9-3 и его запуск с указанием аргументов	17
5.1	Создание файла lab9-4-var19.asm в каталоге ~/work/arch-pc/lab09	18
5.2	Текст составленной программы в файле lab9-4-var19.asm	18
5.3	Создание исполняемого файла lab9-3 и его проверка на нескольких наборах $x = x_1, x_2, \dots, x_n$	19

Список таблиц

1 Цель работы

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

2 Задание

Написать программы с использованием циклов и обработкой аргументов командной строки. Написать программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$ (т. е. находит $f(x_1) + f(x_2) + \dots + f(x_n)$), где значения x_i передаются как аргументы.

3 Теоретическое введение

Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Основной функцией стека является функция сохранения адресов возврата и передачи аргументов при вызове процедур. Также 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`. Противоположный конец стека называется дном. Последнее добавленное в стек значение извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для работы со стеком в процессоре есть специальные регистры (`ss`, `bp`, `sp`) и команды. Также для стека существует две основные операции: добавление элемента в вершину стека (`push`) и извлечение элемента из вершины стека (`pop`). Для организации циклов существуют специальные инструкции, для каждой максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`, которая позволяет организовать безусловный цикл. Она выполняется в два этапа. Сначала из регистра `ecx` вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке, если равен, то переход не выполняется и управление передаётся команде, следующей сразу после команды `loop`.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Я создала каталог для программ лабораторной работы №9, перешла в него и создала файл lab9-1.asm (рис. 4.1).

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

Рис. 4.1: Создание каталога для программ лабораторной работы №9 и 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 я рассмотрела программу, выводящую значение регистра ecx. Я внимательно изучила текст программы листинга 9.1 и ввела его в файл lab9-1.asm (рис. 4.2).


```

/home/mkgolovanova/work/arch-pc/lab09/lab9-1.asm
;-----
; Программа вывода значений регистра 'ecx'
;-----
#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. 4.2: Введение текста программы из листинга 9.1

Я создала исполняемый файл и проверила его работу (рис. 4.3).

```

[mkgolovanova@fedora lab09]$ nasm -f elf lab9-1.asm
[mkgolovanova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[mkgolovanova@fedora lab09]$ ./lab9-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
[mkgolovanova@fedora lab09]$ █

```

Рис. 4.3: Создание и запуск исполняемого файла lab9-1

Данный пример показывает, что использование регистра есх в теле цикла loop может привести к некорректной работе программы. Я изменила текст программы, добавив изменение значение регистра есх в цикле (рис. 4.4).

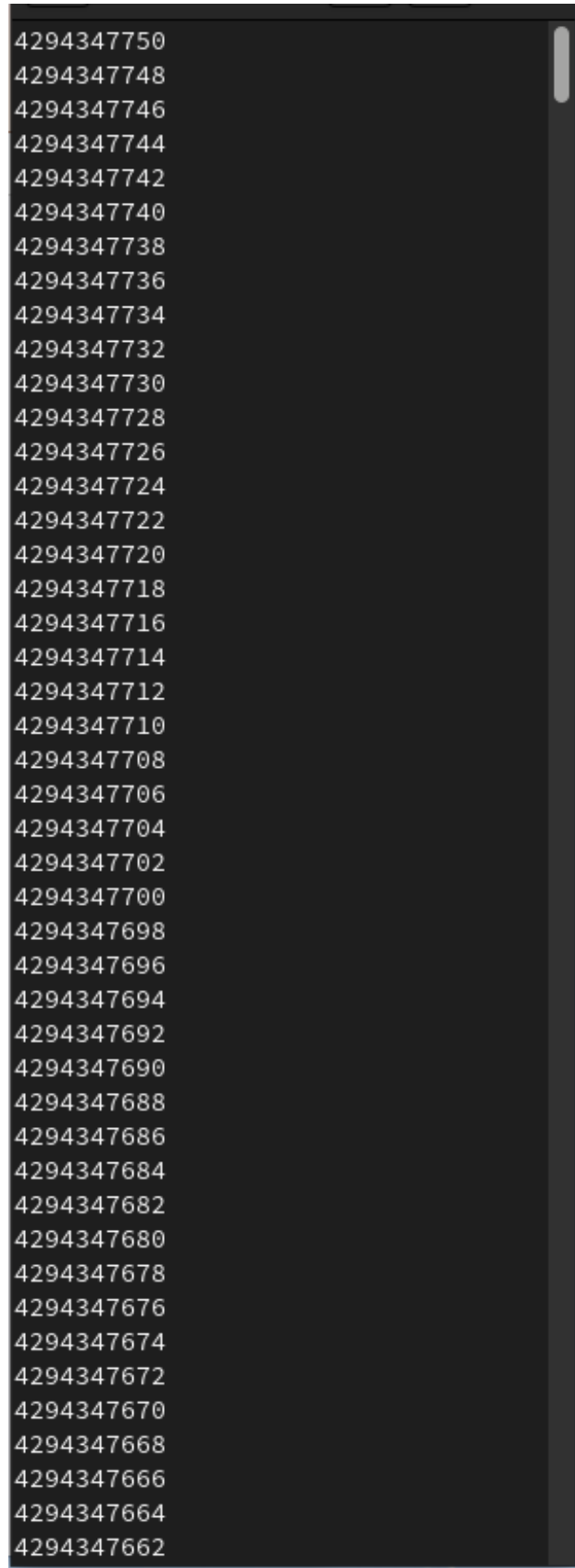
```

mov есх,[N] ; Счетчик цикла, `есх=N`
label:
sub есх,1 ; `есх=есх-1`
mov [N],есх
mov еах,[N]
call iprintLF
loop label█
call quit

```

Рис. 4.4: Изменение текста файла lab9-1.asm

Я создала исполняемый файл и проверила его работу (рис. 4.5). Регистр есх принимает в цикле значения, которые не могут входить в N (слишком большие для введенного N). Число проходов цикла не соответствует значению N, введенному с клавиатуры.



```
4294347750
4294347748
4294347746
4294347744
4294347742
4294347740
4294347738
4294347736
4294347734
4294347732
4294347730
4294347728
4294347726
4294347724
4294347722
4294347720
4294347718
4294347716
4294347714
4294347712
4294347710
4294347708
4294347706
4294347704
4294347702
4294347700
4294347698
4294347696
4294347694
4294347692
4294347690
4294347688
4294347686
4294347684
4294347682
4294347680
4294347678
4294347676
4294347674
4294347672
4294347670
4294347668
4294347666
4294347664
4294347662
```

Рис. 4.5: Результат работы изменённого файла lab9-1.asm

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Я внесла изменения в текст программы, добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 4.6).

```
, ..... Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
push ecx ; добавление значения ecx в стек
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF
pop ecx ; извлечение значения ecx из стека
loop label
call quit
```

Рис. 4.6: Изменение текста файла `lab9-1.asm`

Я создала исполняемый файл и проверила его работу (рис. 4.7). В данном случае число проходов цикла соответствует значению `N`, введенному с клавиатуры.

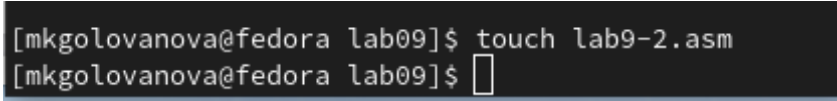
```
[mkgolovanova@fedora lab09]$ nasm -f elf lab9-1.asm
[mkgolovanova@fedora lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[mkgolovanova@fedora lab09]$ ./lab9-1
Введите N: 7
6
5
4
3
2
1
0
[mkgolovanova@fedora lab09]$
```

Рис. 4.7: Создание и запуск изменённого исполняемого файла `lab9-1`

4.2 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 я рассмотрела программу, которая выводит на экран аргументы командной строки. Я создала файл lab9-2.asm в каталоге ~/work/arch-pc/lab09 (рис. 4.8).



```
[mkgolovanova@fedora lab09]$ touch lab9-2.asm  
[mkgolovanova@fedora lab09]$
```

Рис. 4.8: Создание файла lab9-2.asm в каталоге ~/work/arch-pc/lab09

Я внимательно изучила текст программы листинга 9.2 и ввела его в файл lab9-2.asm (рис. 4.9)

```

/home/mkgolovanova/work/arch-pc/lab09/lab9-2.asm
;-----
; Обработка аргументов командной строки
;-----
%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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit

```

Рис. 4.9: Введение текста программы из листинга 9.2

Я создала исполняемый файл и проверила его работу (рис. 4.10). Программа обработала 5 аргументов.

```

[mkgolovanova@fedora lab09]$ ./lab9-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[mkgolovanova@fedora lab09]$

```

Рис. 4.10: Создание и запуск исполняемого файла lab9-2

Я рассмотрела еще один пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Я создала файл lab9-3.asm в каталоге ~/work/arch-pc/lab09 (рис. 4.11) и ввела в него текст программы из листинга 9.3(рис. 4.12)

```
[mkgolovanova@fedora lab09]$ touch lab9-3.asm
[mkgolovanova@fedora lab09]$
```

Рис. 4.11: Создание файла lab9-3.asm в каталоге ~/work/arch-pc/lab09

```
/home/mkgolovanova/work/arch-pc/lab09/lab9-3.asm Из
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
por 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`)
por 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. 4.12: Введение текста программы из листинга 9.3

Я создала исполняемый файл и запустила его, указав аргументы (рис. 4.13)

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

Рис. 4.13: Создание исполняемого файла lab9-3 и его запуск с указанием аргументов

Я изменила текст программы из листинга 9.3 для вычисления произведения аргументов командной строки (рис. 4.14).

```
GNU nano 6.0 /home/mkgolovanova/work/arch-pc/lab09/lab9-3.asm Изменён
#include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
SECTION .text
global _start
_start:
por ecx ; Извлекаем из стека в `ecx` количество аргументов (первое значение в стеке)
por 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, 1 ; Используем `esi` для хранения промежуточных произведений
next:
cmp ecx,0h ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла (переход на метку `_end`)
por eax ; иначе извлекаем следующий аргумент из стека
call atoi ; преобразуем символ в число

mul esi ; добавляем к промежуточному произведению след. аргумент `eax=esi*eax`
mov esi,eax ; записываем промежут. произведение в esi
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprint
mov eax, esi ; записываем произведение в регистр `eax`
call iprintf ; печать результата
call quit ; завершение программы
```

Рис. 4.14: Изменение текста файла lab9-3.asm для вычисления произведения аргументов командной строки

Я создала новый исполняемый файл и запустила его, указав аргументы (рис. 4.15)


```
[mkgolovanova@fedora lab09]$ nasm -f elf lab9-3.asm
[mkgolovanova@fedora lab09]$ ld -m elf_i386 -o lab9-3 lab9-3.o
[mkgolovanova@fedora lab09]$ ./lab9-3 4 3 3 4
Результат: 144
[mkgolovanova@fedora lab09]$ ./lab9-3 7 5
Результат: 35
[mkgolovanova@fedora lab09]$ ./lab9-3 28 2 2
Результат: 112
[mkgolovanova@fedora lab09]$
```

Рис. 4.15: Создание нового исполняемого файла lab9-3 и его запуск с указанием аргументов

5 Задание для самостоятельной работы

Я написала программу, которая находит сумму значений функции $f(x)$ для $x = x_1, x_2, \dots, x_n$, т.е. программа выводит значение $f(x_1) + f(x_2) + \dots + f(x_n)$. Значения x_i передаются как аргументы. Вид функции $f(x)$ я выбрала из таблицы 9.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы №7 (вариант 19). Я создала файл `lab9-4-var19.asm` в каталоге `~/work/arch-pc/lab09` (рис. 5.1) и ввела в него текст составленной программы (рис. 5.2).

```
[mkgolovanova@fedora lab09]$ touch lab9-4-var19.asm
[mkgolovanova@fedora lab09]$
```

Рис. 5.1: Создание файла `lab9-4-var19.asm` в каталоге `~/work/arch-pc/lab09`

```
GNU nano 6.0 /home/mkgolovanova/work/arch-pc/lab09/lab9-4-var19.asm
;var 19: f(x)= 8x - 3
#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 функций от различ. x
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

mov ebx,8 ; ebx=8
mul ebx ; eax=ebx*eax
sub eax, 3 ; eax=eax-3
add esi,eax ; добавляем к промежуточной сумме функций от различ. x значение 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 iprintf ; печать результата
call quit ; завершение программы
```

Рис. 5.2: Текст составленной программы в файле `lab9-4-var19.asm`

Я создала исполняемый файл и проверила его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$ (рис. 5.3).

```
[mkgolovanova@fedora lab09]$ nasm -f elf lab9-4-var19.asm
[mkgolovanova@fedora lab09]$ ld -m elf_i386 -o lab9-4-var19 lab9-4-var19.o
[mkgolovanova@fedora lab09]$ ./lab9-4-var19 1 2 3
Результат: 39
[mkgolovanova@fedora lab09]$ ./lab9-4-var19 1 3 5
Результат: 63
[mkgolovanova@fedora lab09]$ ./lab9-4-var19 2 7 4
Результат: 95
[mkgolovanova@fedora lab09]$
```

Рис. 5.3: Создание исполняемого файла lab9-3 и его проверка на нескольких наборах $x = x_1, x_2, \dots, x_n$

6 Выводы

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

Список литературы