

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Пихтовникова Алёна Владимировна

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	10
4.3	Задание для самостоятельной работы	13
5	Выводы	16
6	Список литературы	17

Список иллюстраций

4.1	Создание каталога	7
4.2	Копирование программы из листинга	8
4.3	Запуск программы	8
4.4	Изменение программы	9
4.5	Запуск измененной программы	9
4.6	Добавление push и pop в цикл программы	10
4.7	Запуск измененной программы	10
4.8	Копирование программы из листинга	11
4.9	Запуск второй программы	11
4.10	Копирование программы из третьего листинга	12
4.11	Запуск третьей программы	12
4.12	Изменение третьей программы	12
4.13	Запуск измененной третьей программы	13
4.14	Написание программы для самостоятельной работы	13
4.15	Запуск программы для самостоятельной работы	15

1 Цель работы

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

2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Самостоятельное написание программы по материалам лабораторной работы

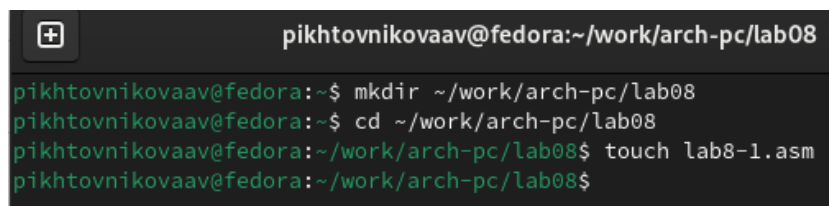
3 Теоретическое введение

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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

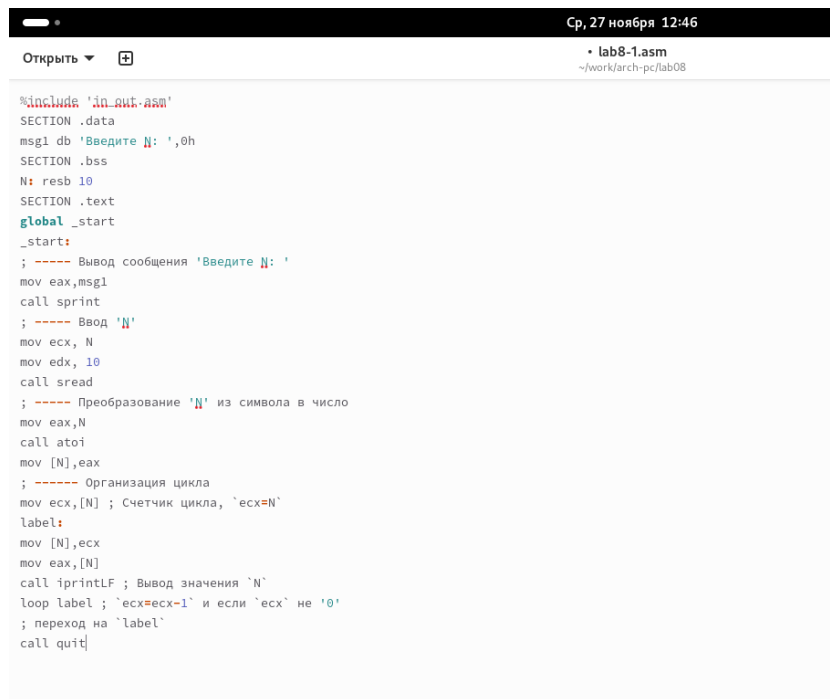
Создаю каталог для программ лабораторной работы №8 (рис. 4.1).

A terminal window with a dark background. The title bar shows a plus icon and the text 'pikhtovnikovaav@fedora:~/work/arch-pc/lab08'. The terminal contains four lines of text: 'pikhtovnikovaav@fedora:~\$ mkdir ~/work/arch-pc/lab08', 'pikhtovnikovaav@fedora:~\$ cd ~/work/arch-pc/lab08', 'pikhtovnikovaav@fedora:~/work/arch-pc/lab08\$ touch lab8-1.asm', and 'pikhtovnikovaav@fedora:~/work/arch-pc/lab08\$'.

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

Рис. 4.1: Создание каталога

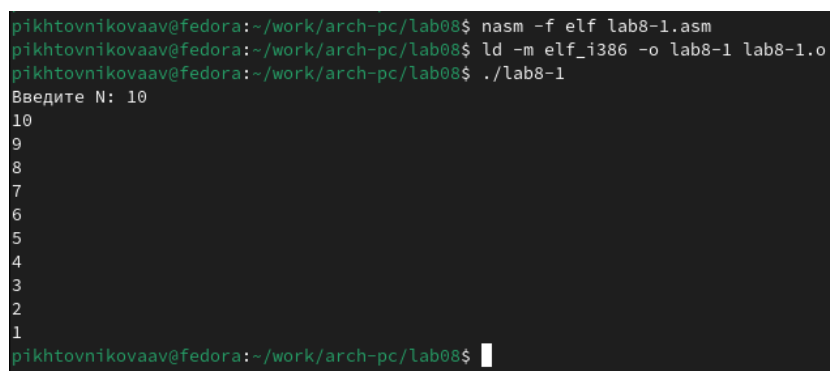
Копирую в созданный файл программу из листинга. (рис. 4.2).



```
%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. 4.2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 4.3).



```
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$
```

Рис. 4.3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра ecx (рис. 4.4).


```

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

```

Рис. 4.4: Изменение программы

Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 4.5).

```

pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$

```

Рис. 4.5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 4.6).

```

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

```

Рис. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 4.7).

```

pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$

```

Рис. 4.7: Запуск измененной программы

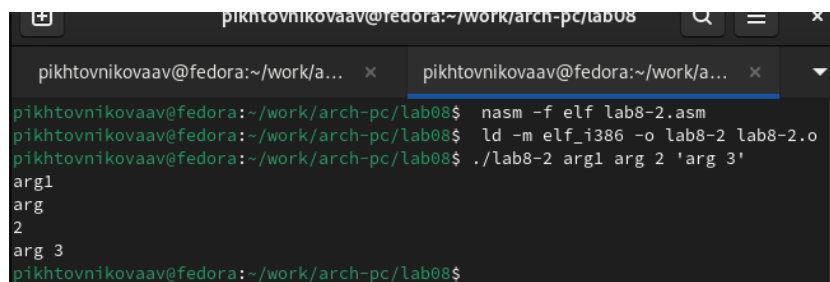
4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 4.8).

```
Файл  Правка  Поиск  Просмотр  Документ  Помощь
%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. 4.8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обработано то же количество аргументов, что и было введено (рис. 4.9).



```
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg2 'arg 3'
arg1
arg2
2
arg 3
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$
```

Рис. 4.9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 4.10).

lab8-2.asm	lab8-3.asm
<pre>%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 ; проверим, 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 iprintf ; печать результата call quit ; завершение программы</pre>	

Рис. 4.10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 4.11).

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

Рис. 4.11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 4.12).

lab8-2.asm	lab8-3.asm
<pre>%include 'in_out.asm' SECTION .data msg db "Результат: ", 0 SECTION .text GLOBAL _start _start: pop ecx pop edx sub ecx, 1 mov esi, 1 next: cmp ecx, 0h jz _end pop eax call atoi mul esi, eax mov esi, eax loop next _end: mov eax, msg call sprint mov eax, esi call iprintfLF --</pre>	

Рис. 4.12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 4.13).

```

Результат: 47
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-3 111 1 6
Результат: 666

```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумму значений для функции $f(x) = 3x-1$, которая совпадает с моим вторым вариантом (рис. 4.14).

```

%include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 3x-1", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start

_start:
; ----- Вывод сообщения о функции
mov eax, msg_func
call printf

; ----- Инициализация
pop ecx ; Считываем количество входных данных (N)
sub ecx, 1 ; Уменьшаем на 1, чтобы учесть начальное значение
mov esi, 0 ; Инициализация суммы (результата)

next:
cmp ecx, 0h ; Проверяем, не достигли ли нуля
jz _end ; Если да, переходим к окончанию

pop eax ; Считываем следующее значение (x)
call atoi ; Преобразуем строку в число

; ----- Вычисление f(x) = 3x - 1
mov ebx, 3 ; Устанавливаем множитель 3
mul ebx ; Умножаем eax на 3 (eax = 3 * x)
sub eax, 1 ; Вычитаем 1 (eax = 3 * x - 1)

```

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

```

#include 'in_out.asm'
SECTION .data
msg_func db "Функция: f(x) = 3x-1", 0
msg_result db "Результат: ", 0
SECTION .text
GLOBAL _start

```

```

_start:
    ; ----- Вывод сообщения о функции
    mov eax, msg_func
    call sprintLF

    ; ----- Инициализация
    pop ecx          ; Считываем количество входных данных (N)
    sub ecx, 1       ; Уменьшаем на 1, чтобы учесть начальное значение
    mov esi, 0       ; Инициализация суммы (результата)

next:
    cmp ecx, 0h      ; Проверяем, не достигли ли нуля
    jz _end          ; Если да, переходим к окончанию

    pop eax          ; Считываем следующее значение (x)
    call atoi        ; Преобразуем строку в число

    ; ----- Вычисление  $f(x) = 3x - 1$ 
    mov ebx, 3       ; Устанавливаем множитель 3
    mul ebx          ; Умножаем eax на 3 ( $eax = 3 * x$ )
    sub eax, 1       ; Вычитаем 1 ( $eax = 3 * x - 1$ )

    add esi, eax     ; Добавляем результат к сумме

    loop next        ; Переход к следующему значению

_end:
    ; ----- Вывод результата
    mov eax, msg_result

```

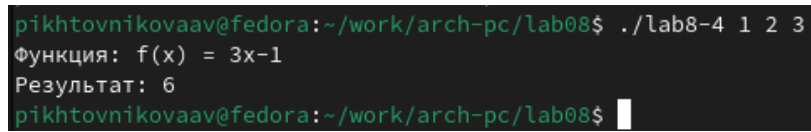
```
call sprint
```

```
mov eax, esi ; Загружаем итоговое значение в eax
```

```
call iprintLF ; Печатаем результат
```

```
call quit ; Завершение программы
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 4.15).



```
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$ ./lab8-4 1 2 3
Функция: f(x) = 3x-1
Результат: 6
pikhtovnikovaav@fedora:~/work/arch-pc/lab08$
```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов а также научилась обрабатывать аргументы командной строки.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.