

Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Киселева Елизавета Александровна

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
4.3	Задание для самостоятельной работы	14
5	Выводы	18
6	Список литературы	19

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Копирование программы из листинга	9
4.3	Запуск программы	9
4.4	Изменение программы	10
4.5	Запуск измененной программы	10
4.6	Добавление push и pop в цикл программы	11
4.7	Запуск измененной программы	11
4.8	Создание файла	12
4.9	Запуск второй программы	12
4.10	Копирование программы из третьего листинга	13
4.11	Запуск третьей программы	13
4.12	Изменение третьей программы	14
4.13	Запуск измененной третьей программы	14
4.14	Написание программы для самостоятельной работы	15
4.15	Запуск программы для самостоятельной работы	17

Список таблиц

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, и перехожу в него и создаю файл lab8-1.asm (рис. 4.1).

```
eakiseleva1@dk3n55 ~ $ mkdir ~/work/arch-pc/lab08
eakiseleva1@dk3n55 ~ $ cd ~/work/arch-pc/lab08
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
```

Рис. 4.1: Создание каталога и файла

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).

```

eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 4
4
3
2
1
eakiseleva1@dk3n55 ~/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).

```

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 ; 'ecx=ecx-1' и если 'ecx' не '0'
28 ; переход на 'label'
29 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).

```

eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 4
3
1

```

Рис. 4.5: Запуск измененной программы

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

```

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. 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).

```

eakiseval@dk3n55 ~/work/arch-pc/lab08 $ gedit lab8-1.asm
eakiseval@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakiseval@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakiseval@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 4
3
2
1
0

```

Рис. 4.7: Запуск измененной программы

4.2 Обработка аргументов командной строки

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

```
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 printf ; вызываем функцию печати
17 loop next ; переход к обработке следующего
18 ; аргумента (переход на метку 'next')
19 _end:
20 call quit
```

Рис. 4.8: Создание файла

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

```
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ touch lab8-2.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ gedit lab8-2.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
eakiseleva1@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
```

Рис. 4.9: Запуск второй программы

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

```

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. 4.10: Копирование программы из третьего листинга

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

```

eakiseval@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
eakiseval@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
eakiseval@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
eakiseval@dk3n55 ~/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).

```

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 ; Инициализируем esi значением 1 для произведения
14 next:
15 cmp ecx,0h ; проверяем, есть ли еще аргументы
16 jz _end ; если аргументов нет выходим из цикла
17 ; (переход на метку '_end')
18 pop eax ; иначе извлекаем следующий аргумент из стека
19 call atoi ; преобразуем символ в число
20 imul esi,eax ; умножаем промежуточное произведение
21 ; на текущий аргумент esi=esi*eax
22 loop next ; переход к обработке следующего аргумента
23 _end:
24 mov eax, msg ; вывод сообщения "Результат: "
25 call sprint
26 mov eax, esi ; записываем произведение в регистр eax
27 call iprintLF ; печать результата
28 call quit ; завершение программы

```

Рис. 4.12: Изменение третьей программы

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

```

eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ gedit lab8-3.asm
eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 54600

```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программу, которая будет находить сумма значений для функции $f(x) = 30x - 11$, которая соответствует с моему 16 варианту (рис. 4.14).

```

1 %include 'in_out.asm'
2
3 SECTION .data
4 msg_func db "Функция: f(x) = 30*x - 11", 0
5 msg_result db "Результат: ", 0 ; Сообщение для вывода результата
6
7 SECTION .text
8 global _start
9 _start:
10 ; Выводим сообщение о функции
11 mov eax, msg_func
12 call sprintf
13
14 ; Инициализируем стек аргументов
15 pop ecx ; Извлекаем количество аргументов в 'ecx'
16 pop edx ; Извлекаем имя программы в 'edx'
17 sub ecx, 1 ; Уменьшаем количество аргументов (без названия программы)
18 mov esi, 0 ; Инициализируем сумму значений функции в 'esi'
19
20 next:
21 cmp ecx, 0 ; Проверяем, есть ли еще аргументы
22 jz _end ; Если аргументов нет, завершаем цикл
23
24 pop eax ; Извлекаем следующий аргумент из стека (как строку)
25 call atoi ; Преобразуем аргумент из строки в число
26
27 ; Вычисляем f(x) = 30 * eax - 11
28 mov ebx, 30 ; EBX = 30
29 imul eax, ebx ; EAX = 30 * x
30 sub eax, 11 ; EAX = 30 * x - 11
31
32 ; Добавляем результат функции к сумме
33 add esi, eax ; ESI = ESI + f(x)
34
35 dec ecx ; Уменьшаем 'ecx' (явно)
36 jmp next ; Переход к следующему аргументу
37
38 _end:
39 ; Выводим результат
40 mov eax, msg_result ; Загрузка сообщения "Результат: "
41 call sprintf ; Печать сообщения
42 mov eax, esi ; Загружаем сумму в EAX для вывода
43 call sprintf ; Печать суммы
44
45 ; Завершаем программу
46 call quit

```

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

```
%include 'in_out.asm'
```

```
SECTION .data
```

```
msg_func db "Функция: f(x) = 30*x - 11", 0
```

```
msg_result db "Результат: ", 0 ; Сообщение для вывода результата
```

```
SECTION .text
```

```
global _start
```

```
_start:
```

```
    ; Выводим сообщение о функции
```

```

mov eax, msg_func
call sprintf

; Инициализируем стек аргументов
pop ecx          ; Извлекаем количество аргументов в `ecx`
pop edx          ; Извлекаем имя программы в `edx`
sub ecx, 1        ; Уменьшаем количество аргументов (без названия программы)
mov esi, 0        ; Инициализируем сумму значений функции в `esi`

next:
cmp ecx, 0        ; Проверяем, есть ли еще аргументы
jz _end           ; Если аргументов нет, завершаем цикл

pop eax          ; Извлекаем следующий аргумент из стека (как строку)
call atoi        ; Преобразуем аргумент из строки в число

; Вычисляем  $f(x) = 30 * eax - 11$ 
mov ebx, 30      ; EBX = 30
imul eax, ebx    ; EAX = 30 * x
sub eax, 11      ; EAX = 30 * x - 11

; Добавляем результат функции к сумме
add esi, eax     ; ESI = ESI + f(x)

dec ecx          ; Уменьшаем `ecx` (явно)
jmp next         ; Переход к следующему аргументу

_end:
; Выводим результат

```



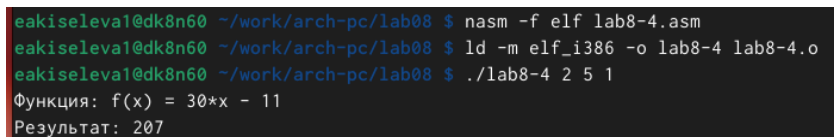
```

mov eax, msg_result    ; Загрузка сообщения "Результат: "
call sprint             ; Печать сообщения
mov eax, esi            ; Загружаем сумму в EAX для вывода
call iprintfLF          ; Печать суммы

; Завершаем программу
call quit

```

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



```

eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
eakiseleva1@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-4 2 5 1
Функция: f(x) = 30*x - 11
Результат: 207

```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

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

6 Список литературы

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