

Лабораторная работа №8

Дисциплина: Архитектура Компьютера

Первий Анастасич Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	14
4.3	Выполнение заданий для самостоятельной работы	16
5	Выводы	18
6	Листинги	19
6.1	Листинг 8.1. Программа вывода значений регистра еsx	19
6.2	Листинг 8.3. Программа вычисления суммы аргументов командной строки	20
6.3	Листинг 1	20

Список иллюстраций

4.1	Базовые команды	9
4.2	Листинг 8.1	10
4.3	Копирование. Компиляция. Компоновка. Запуск программы. . . .	10
4.4	Листинг 8.1 Измененный	11
4.5	Компиляция. Компоновка. Запуск программы.	12
4.6	Листинг 8.1 Измененный(I)	13
4.7	Компиляция. Компоновка. Запуск программы.	13
4.8	Листинг 8.2	14
4.9	Компиляция. Компоновка. Запуск программы.	14
4.10	Листинг 8.3	15
4.11	Компиляция. Компоновка. Запуск программы.	15
4.12	Листинг 8.3 Измененный	16
4.13	Компиляция. Компоновка. Запуск программы.	16
4.14	Листинг 1	17
4.15	Компиляция. Компоновка. Запуск программы.	17

Список таблиц

1 Цель работы

Целью данной лабораторной работы является приобретение практического опыта в написании программ с использованием циклов и обработкой аргументов командной строки.

2 Задание

0. Общее ознакомление с циклами и обработкой аргументов командной строки
1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

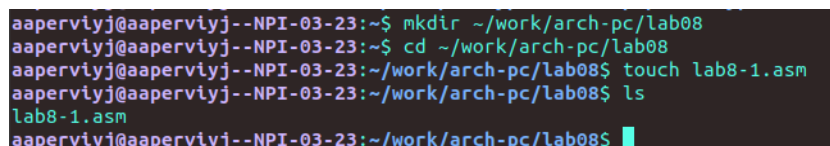
Стек — это структура данных, организованная по принципу LIFO («Last In — First Out» или «последним пришёл — первым ушёл»). Стек является частью архитектуры процессора и реализован на аппаратном уровне. Для работы со стеком в процессоре есть специальные регистры (ss, bp, sp) и команды. Основной функцией 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. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop). Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек. Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после

этого уменьшает значение регистра `esp` на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек. Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop`. Она позволяет организовать безусловный цикл. Инструкция `loop` 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. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды `loop`.

4 Выполнение лабораторной работы

Перед тем как начать выполнять лабораторную работу, необходимо создать директорию и файл, в которых я и буду работать, для этого прописываю следующие команды (Рис.1):

```
mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.asm
```

A screenshot of a terminal window showing a series of commands being executed. The prompt is 'aaperviyj@aaperviyj--NPI-03-23:~\$'. The commands are: 'mkdir ~/work/arch-pc/lab08', 'cd ~/work/arch-pc/lab08', 'touch lab8-1.asm', and 'ls'. The output of 'ls' is 'lab8-1.asm'. The prompt then changes to 'aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08\$' with a green cursor.

```
aaperviyj@aaperviyj--NPI-03-23:~$ mkdir ~/work/arch-pc/lab08
aaperviyj@aaperviyj--NPI-03-23:~$ cd ~/work/arch-pc/lab08
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ touch lab8-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ls
lab8-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$
```

Рис. 4.1: Базовые команды

Теперь, после того, как я создала пространство для дальнейшей работы, я могу переходить к следующему пункту

4.1 Реализация циклов в NASM

Открываю ранее созданный файл и вставляю туда программу из листинга 8.1 (Рис.2)

```

;-----
; 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: Листинг 8.1

Копирую в текущий каталог файл in_out.asm с помощью утилиты cp, так как он будет использоваться в дальнейшем. Создаю исполняемый файл и запускаю его. Замечу, что использование инструкции loop позволяет выводить значения регистра ecx циклично (Рис.3)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ cp ~/in_out.asm in_out.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 9
9
8
7
6
5
4
3
2
1
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$

```

Рис. 4.3: Копирование. Компиляция. Компоновка. Запуск программы.

Изменяю значение ecx в цикле (Рис.4)

'''nasm

(text)

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

'''

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

Рис. 4.4: Листинг 8.1 Измененный

Создаю новый исполняемый файл программы и запускаю его. Замечу, что регистр ecx в цикле принимает совершенно разные значения. И число проходов цикла далеко не соответствует ли значению ☒, введенному с клавиатуры (Рис.5)

```

4292989862
4292989860
4292989858
4292989856
4292989854
4292989852
4292989850
4292989848
4292989846
4292989844
4292989^C
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$

```

Рис. 4.5: Компиляция. Компоновка. Запуск программы.

Вношу изменения в текст программы, добавив команды `push`, `pop` для сохранения значения счётчика цикла `loop` (Рис. 6)

```

'''nasm
(text)
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
'''

```

```

;-----
; 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`
loop_start:
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: Листинг 8.1 Измененный(I)

Выполняю компиляцию и компоновку, и запускаю исполняемый файл. В данном случае число проходов цикла соответствует значению ☒ введенному с клавиатуры. Счёт идёт, не от 9-ми, а от 8-ми, но включается 0 (Рис. 7)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 9
8
7
6
5
4
3
2
1
0
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$

```

Рис. 4.7: Компиляция. Компоновка. Запуск программы.

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm. Редактирую его, вводя предлагаемую программу из листинга 8.2 (Рис.8)

```
;-----  
; 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. 4.8: Листинг 8.2

Создаю исполняемый файл после редактирования. Запускаю исполняемый файл. Программой было обработано 3 аргумента - ровно те, которые я указала при запуске (Рис.9)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm  
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o  
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./lab8-2 1 3 '5'  
1  
3  
5  
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$
```

Рис. 4.9: Компиляция. Компоновка. Запуск программы.

Создаю файл lab8-3.asm. Ввожу в него программу из листинга 8.3 (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. 4.10: Листинг 8.3

Создаю исполняемый файл. Указываю нужные аргументы. Выполняя устную проверку($1+2+3+4+5=15$), убеждаюсь в правильности работы программы (Рис.11)

```

5
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./lab8-3 1 2 3 4 5
Результат: 15
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$

```

Рис. 4.11: Компиляция. Компоновка. Запуск программы.

Изменяю текст программы для вычисления произведения аргументов командной строки (Рис.12)

```

GNU nano 6.2
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, 1 ; 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
imul esi,eax ; умножаем на промежуточное произведение
; след. аргумент `esi=esi+eax`
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр `eax`
call iprintf ; печать результата
call quit ; завершение программы

```

Рис. 4.12: Листинг 8.3 Измененный

Создаю исполняемый файл. Указываю нужные аргументы. Выполняя устную проверку($234=24$), убеждаюсь в правильности работы программы (Рис.13)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./lab8-3 2 3 4
Результат: 24
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$

```

Рис. 4.13: Компиляция. Компоновка. Запуск программы.

4.3 Выполнение заданий для самостоятельной работы

Создаю файл sr.asm с помощью утилиты touch. Открываю созданный файл для редактирования, ввожу в него текст программы (Листинг 1) для суммирования значений функции, предложенной в варианте 16, полученным мною при

выполнении прошлой лабораторной работы (Рис.14)

```
%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)

    ; Иначе извлекаем следующий аргумент из стека
    pop eax
    call atoi ; Преобразуем символ в число

    ; Вычисляем f(x) = 30x - 11
    imul eax, 30 ; Умножаем на 30
    sub eax, 11 ; Вычитаем 11

    ; Добавляем к промежуточной сумме
    add esi, eax

    loop next ; Переход к обработке следующего аргумента

_end:
    ; Выводим сообщение "Результат: "
    mov eax, msg
    call sprint

    ; Печать результата
    mov eax, esi ; Записываем сумму в регистр eax
    call iprintLF

    ; Завершение программы
    call quit
```

Рис. 4.14: Листинг 1

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку $((301-11)3=57)$ и убеждаемся в правильности работы программы (Рис.15)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ nasm -f elf sr.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ld -m elf_i386 -o sr sr.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$ ./sr 1 1 1
Результат: 57
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab08$
```

Рис. 4.15: Компиляция. Компоновка. Запуск программы.

5 Выводы

По ходу выполнения лабораторной работы я приобрела практический опыт в написании программ с использованием циклов и обработкой аргументов командной строки.

6 Листинги

6.1 Листинг 8.1. Программа вывода значений регистра ecx

```
'''nasm
(text)
;----- ; 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
```

''' ## Листинг 8.2. Программа выводящая на экран аргументы командной строки

```
'''nasm
(text)
;----- ; 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 sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего ; аргумента (переход на метку next)
_end: call quit
'''

```

6.2 Листинг 8.3. Программа вычисления суммы аргументов командной строки

```

'''nasm
(text)
%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 аргумен-
тов 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 ; пе-
реход 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
'''

```

6.3 Листинг 1

```

'''nasm
(text)
%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)

; Иначе извлекаем следующий аргумент из стека
        pop eax
        call atoi ; Преобразуем символ в число

; Вычисляем  $f(x) = 30x - 11$ 
        imul eax, 30 ; Умножаем на 30
        sub eax, 11 ; Вычитаем 11

; Добавляем к промежуточной сумме
        add esi, eax

loop next ; Переход к обработке следующего аргумента

_end: ; Выводим сообщение "Результат:"
      mov eax, msg
      call sprint

; Печать результата
      mov eax, esi ; Записываем сумму в регистр eax
      call iprintLF

; Завершение программы
      call quit

;”

```