

Отчёт по лабораторной работе № 8

Простейший вариант

Сахно Алёна Юрьевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическая часть	7
4	Задание для самостоятельной работы	16
5	Выводы	19
	Список литературы	20

Список иллюстраций

3.1	Листинг 8.1. Программа вывода значений регистра еsx	9
3.2	Измените текст программы	11
3.3	Листинг 8.2. Программа выводящая на экран аргументы командной строки	12
3.4	Результат	13
3.5	Листинг 8.3. Программа вычисления суммы аргументов командной строки	14
3.6	Результат работы	15
3.7	Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.	15
4.1	Написание программы	17
4.2	Результат самостоятельная работа 14	18

Список таблиц

1 Цель работы

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

2 Задание

1. Теоретическая часть
2. Задачи для самостоятельной работы
3. Варианты заданий

3 Теоретическая часть

8.2.1. Организация стека

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. На рис. 8.1 показана схема организации стека в процессоре. 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).

8.1. Добавление элемента в стек.

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. Примеры:

push -10 ; Поместить -10 в стек push ebx ; Поместить значение регистра ebx

в стек `push [buf]` ; Поместить значение переменной `buf` в стек `push word [ax]` ; Поместить в стек слово по адресу в `ax`

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.

#Выполнение самостоятельной работы

8.3. Реализация циклов в NASM

Создайте каталог для программ лабораторной работы № 8, перейдите в него и создайте файл `lab8-1.asm`:

```
mkdir ~/work/arch-pc/lab08 cd ~/work/arch-pc/lab08 touch lab8-1.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`. Внимательно изучите текст программы (Листинг 8.1). Листинг 8.1. Программа вывода значений регистра `ecx` (рис. 3.1).


```

GNU nano 7.2 /home/aysakhno/work/arch-pc/lab08/lab8-1.asm
%include 'in_out.asm'

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

SECTION .bss
    N:    resb 1
    0

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

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/_ К строке

```

Рис. 3.1: Листинг 8.1. Программа вывода значений регистра ecx

Введите в файл lab8-1.asm текст программы из листинга 8.1. Создайте исполняемый файл и проверьте его работу. Данный пример показывает, что использование регистра ecx в теле цикла loop может привести к некорректной работе программы. Измените текст программы добавив изменение значение регистра ecx в цикле:

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

(рис. ??).

```

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

```

Архитектура ЭВМ

loop label

Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр `ecx` в цикле? Соответствует ли число проходов цикла значению \square введенному с клавиатуры?

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

```

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

```

Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению \square введенному с клавиатуры?

(рис. 3.2).

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

Рис. 3.2: Измените текст программы

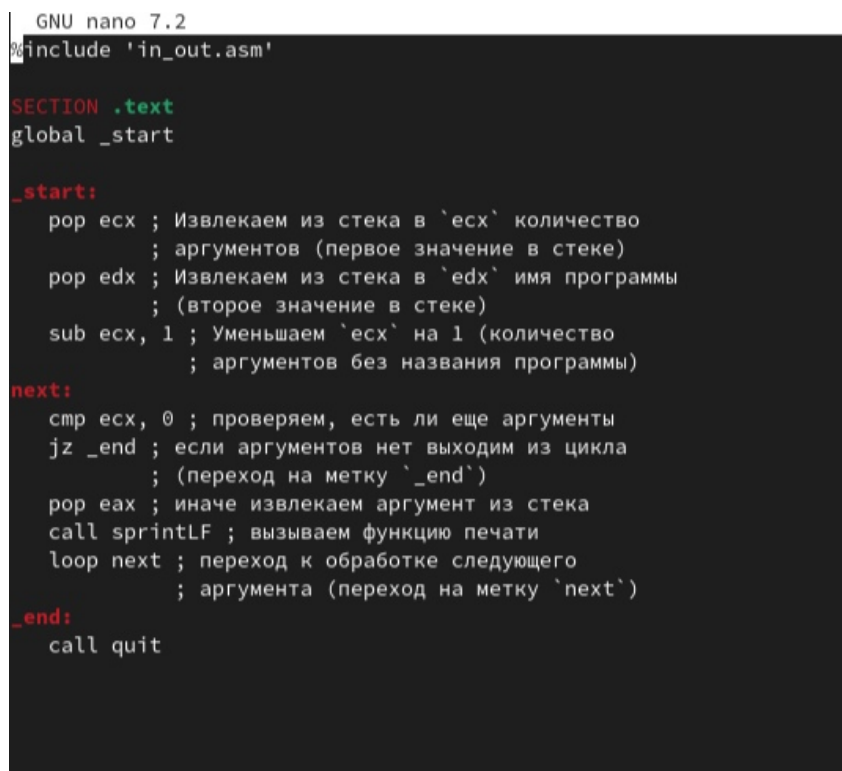
Архитектура ЭВМ loop label Создайте исполняемый файл и проверьте его работу. Какие значения принимает регистр есх в цикле? Соответствует ли число проходов цикла значению введенному с клавиатуры? Для использования регистра есх в цикле и сохранения корректности работы программы можно использовать стек. Внесите изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop: label: push есх ; добавление значения есх в стек sub есх,1 mov [N],есх mov еах,[N] call iprintLF pop есх ; извлечение значения есх из стека loop label Создайте исполняемый файл и проверьте его работу. Соответствует ли в данном случае число проходов цикла значению введенному с клавиатуры?

8.3.2. Обработка аргументов командной строки

При разработке программ иногда возникает 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. Внимательно изучите текст программы (Листинг 8.2).

(рис. 3.3).



```
GNU nano 7.2
%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. 3.3: Листинг 8.2. Программа выводящая на экран аргументы командной строки

Создайте файл lab8-2.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.2. Создайте исполняемый файл и запустите его, указав аргументы: user@dk4n31:~\$./lab8-2 аргумент1 аргумент 2 'аргумент 3'

Сколько аргументов было обработано программой?

(рис. 3.4).

```

aysakhno@fedora:~/work/arch-pc/lab08$ touch lab8-2.asm
aysakhno@fedora:~/work/arch-pc/lab08$ mc

aysakhno@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
aysakhno@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
aysakhno@fedora:~/work/arch-pc/lab08$ ./lab8-2
aysakhno@fedora:~/work/arch-pc/lab08$ ./lab8-2 arg1 arg2 'arg3'
arg1
arg2
arg3
aysakhno@fedora:~/work/arch-pc/lab08$

```

Рис. 3.4: Результат

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создайте файл lab8-3.asm в каталоге ~/work/arch-pc/lab08 и введите в него текст программы из листинга 8.3. (рис. 3.5).

```

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,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
    mul esi
    mov 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

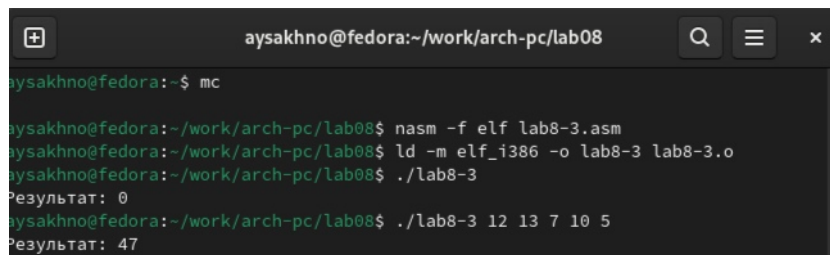
```

Рис. 3.5: Листинг 8.3. Программа вычисления суммы аргументов командной строки

Создайте исполняемый файл и запустите его, указав аргументы. Пример результата работы программы:

```
user@dk4n31:~$ ./main 12 13 7 10 5 Результат: 47 user@dk4n31:~$
```

(рис. 3.6).

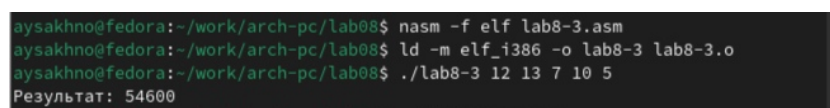


```
aysakhno@fedora:~/work/arch-pc/lab08$ mc
aysakhno@fedora:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
aysakhno@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
aysakhno@fedora:~/work/arch-pc/lab08$ ./lab8-3
Результат: 0
aysakhno@fedora:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
```

Рис. 3.6: Результат работы

Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

(рис. 3.7).



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

Рис. 3.7: Измените текст программы из листинга 8.3 для вычисления произведения аргументов командной строки.

4 Задание для самостоятельной работы

1. Напишите 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 $f(x)$ выбрать из таблицы 8.1 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу на нескольких наборах $x = x_1, x_2, \dots, x_n$

Пример работы программы для функции $f(x) = x + 2$ и набора $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4$:

```
user@dk4n31:~$ ./main 1 2 3 4
```

Функция: $f(x)=x+2$

Результат: 18

```
user@dk4n31:~$
```

(рис. 4.1).



```
GNU nano 7.2 /home/aysakhno/work/arch-pc/lab08/variant-14.asm
#include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x)=7(x+1)", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_func
    call sprint

    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi

    mov ebx, eax
    add ebx, 1
    mov eax, 7
    imul eax, ebx

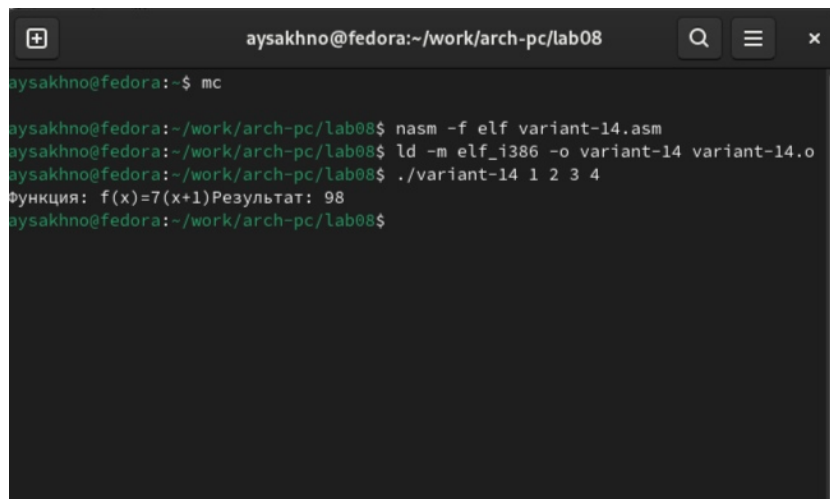
    add esi, eax

    loop next

_end:
    mov eax, msg_result
    call sprint
    mov eax, esi
    call iprintLF
    call quit
```

Рис. 4.1: Написание программы

(рис. 4.2).



```
aysakhno@fedora:~/work/arch-pc/lab08$ mc
aysakhno@fedora:~/work/arch-pc/lab08$ nasm -f elf variant-14.asm
aysakhno@fedora:~/work/arch-pc/lab08$ ld -m elf_i386 -o variant-14 variant-14.o
aysakhno@fedora:~/work/arch-pc/lab08$ ./variant-14 1 2 3 4
Функция:  $f(x)=7(x+1)$ Результат: 98
aysakhno@fedora:~/work/arch-pc/lab08$
```

Рис. 4.2: Результат самостоятельная работа 14

5 Выводы

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

Список литературы

::: {#refs} :::https://esystem.rudn.ru/pluginfile.php/2089095/mod_resource/content