Лабораторная работа №8

Дисциплина: Архитектура Компьютера

Первий Анастасич Андреевна

Содержание

# 1 Цель работы

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

# 2 Задание

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

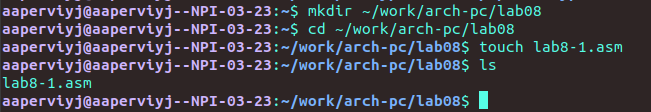
# 3 Теоретическое введение

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: ах, сх, dx, bх, 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. Иструкция 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

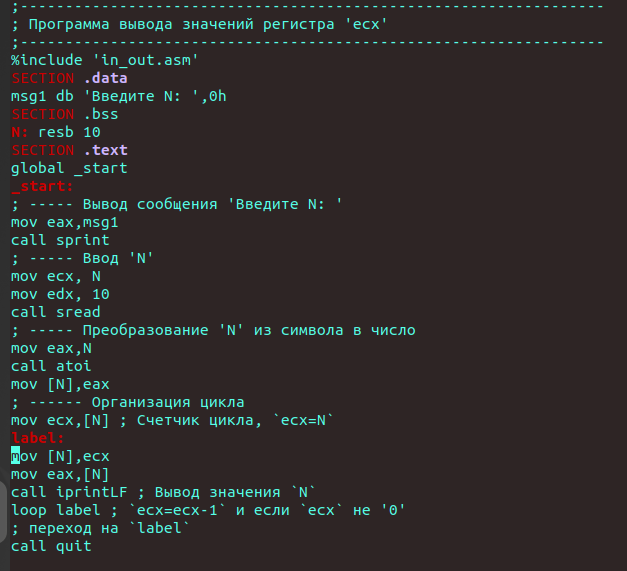


Базовые команды

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

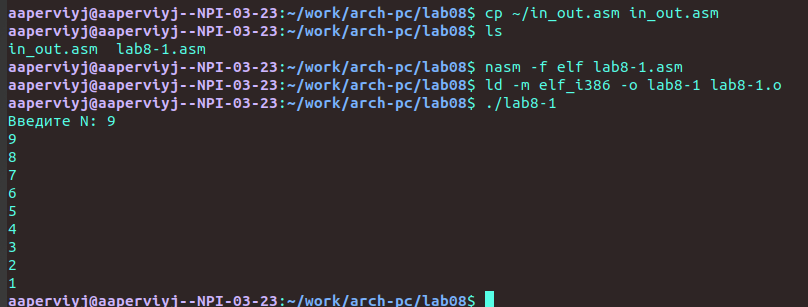
## 4.1 Реализация циклов в NASM

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



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)



Копирование. Компиляция. Компоновка. Запуск программы.

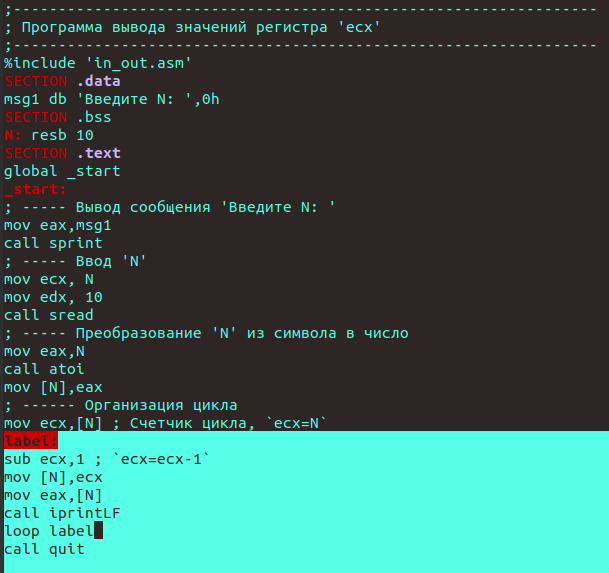
Изменяю значение ecx в цикле (Рис.4)

’’’nasm

(text)

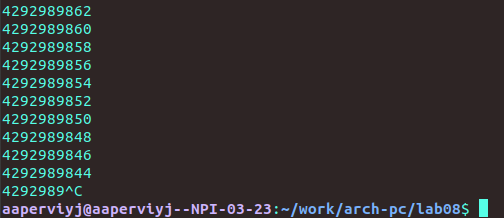
label: sub ecx,1 ; ecx=ecx-1 mov [N],ecx mov eax,[N] call iprintLF loop label

’’’



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 𝑁, введенному с клавиатуры (Рис.5)



Компиляция. Компоновка. Запуск программы.

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

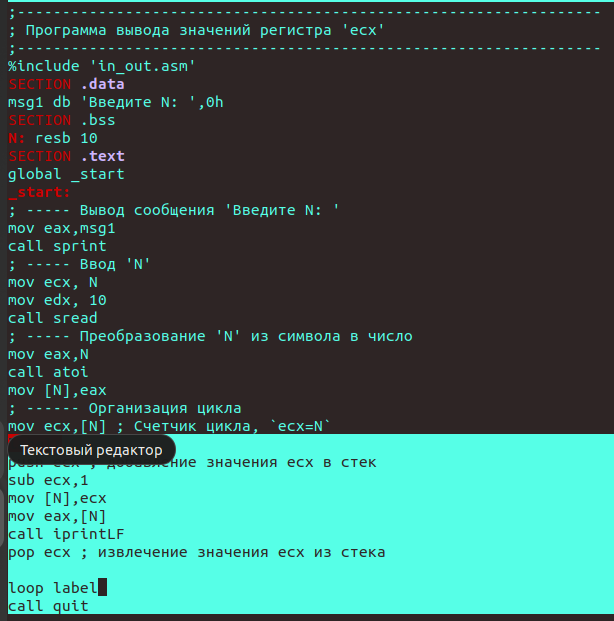
’’’nasm

(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

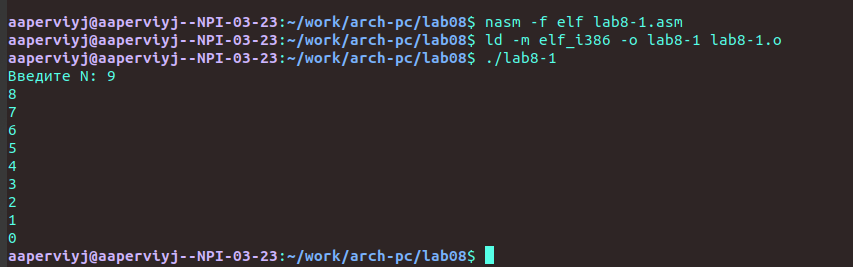
loop label

’’’



Листинг 8.1 Измененный(I)

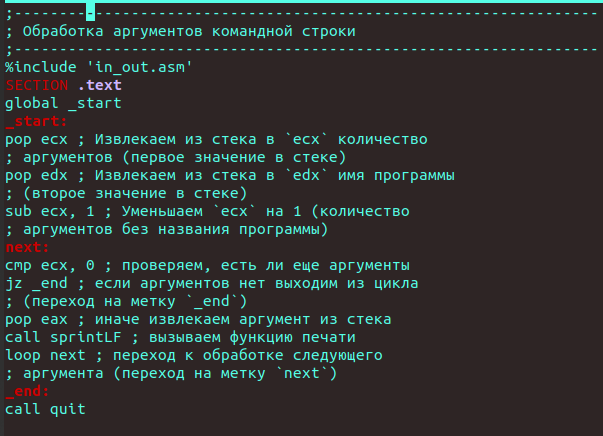
Выполняю компиляцию и компоновку, и запускаю исполняемый файл. В данном случае число проходов цикла соответствует значению 𝑁 введенному с клавиатуры. Счёт идёт, не от 9-ми, а от 8-ми, но включается 0 (Рис. 7)



Компиляция. Компоновка. Запуск программы.

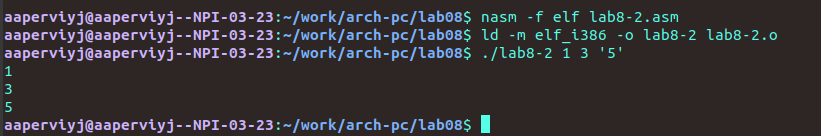
## 4.2 Обработка аргументов командной строки

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



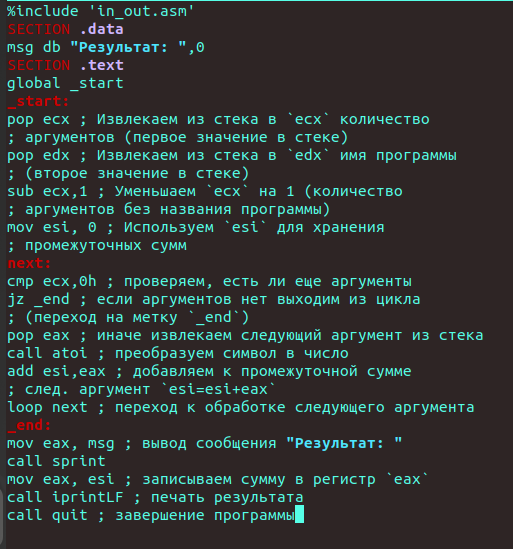
Листинг 8.2

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



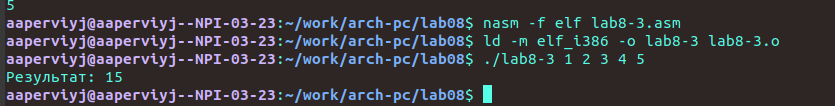
Компиляция. Компоновка. Запуск программы.

Создаю файл lab8-3.asm. Ввожу в него программу из листинга 8.3 (Рис.10)



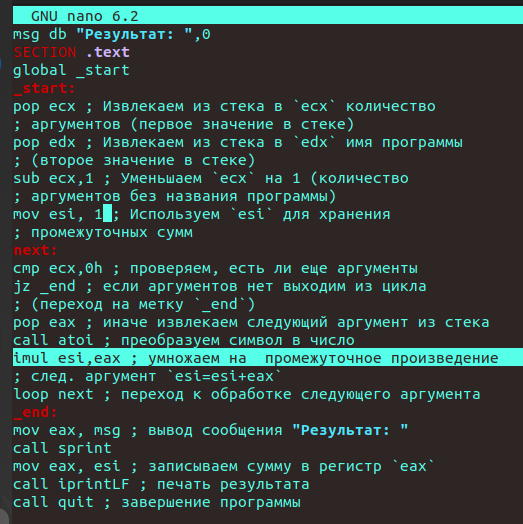
Листинг 8.3

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



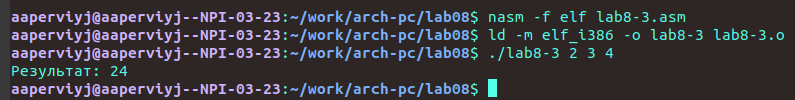
Компиляция. Компоновка. Запуск программы.

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



Листинг 8.3 Измененный

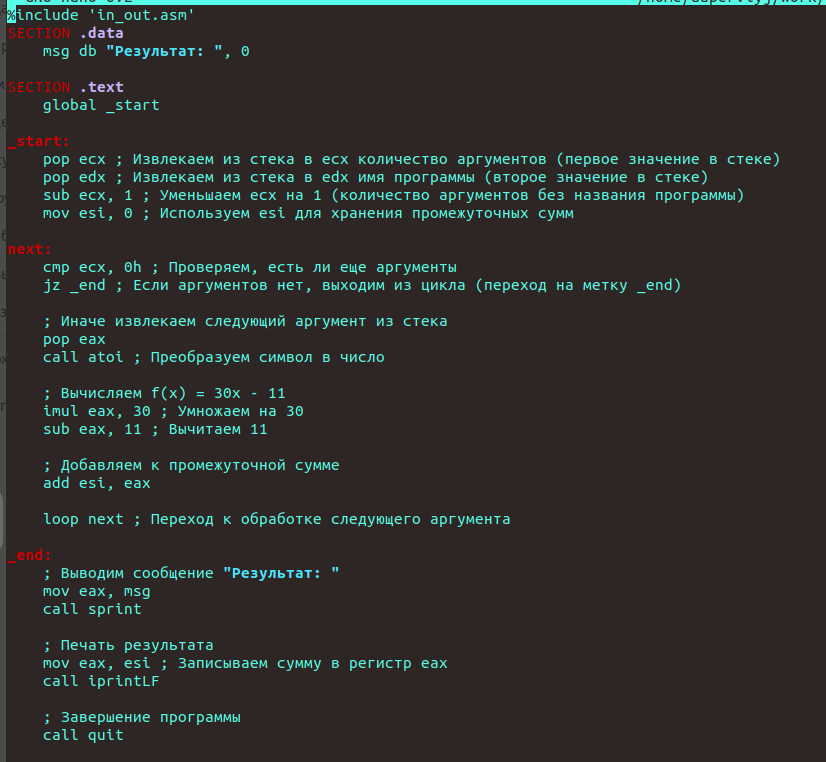
Создаю исполняемый файл. Указываю нужные аргументы. Выполняя устную проверку(2*3*4=24), убеждаюсь в правильности работы программы (Рис.13)



Компиляция. Компоновка. Запуск программы.

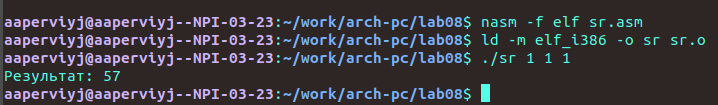
## 4.3 Выполнение заданий для самостоятельной работы

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)



Листинг 1

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку ((30*1-11)*3=57) и убеждаемся в правильности работы программы (Рис.15)



Компиляция. Компоновка. Запуск программы.

# 5 Выводы

По ходу выполнения лебораторной работы я приобрела практический опыт в написании программ с использованием циклов и обраткой 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 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

’’’

## 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

’’’