Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Бунин Арсений Викторович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Создать файл на языке Ассемблер, выводящий убывающий ряд цифр с использованием цикла
2. Создать файл на языке Ассемблер, выводящий введенные аргументы
3. Создать файл на языке Ассемблер, складывающий и перемножающий введенные числа
4. Выполнить индивидуальное задание по написанию программы на Ассемблере
5. Загрузить файлы на github

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

Для стека существует две основные операции: добавление элемента в вершину стека (push) и извлечение элемента из вершины стека (pop).

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

4 Выполнение лабораторной работы

Создаем исполняемый файл(рис. 4.1) (рис. 4.2)

```
[arsenii@fedora ~]$ mkdir /home/arsenii/work/study/2023-2024/"Архитектура компью
тера"/arch-pc/labs/lab08
[arsenii@fedora ~]$ cd /home/arsenii/work/study/2023-2024/"Архитектура компьютер
а"/arch-pc/labs/lab08
[arsenii@fedora lab08]$ touch lab8-1.asm
[arsenii@fedora lab08]$
```

Рис. 4.1: Создание исполняемого файла

Результат работы программы, выводящей убывающий ряд чисел(рис. 4.2)

```
[arsenii@fedora lab08]$ nasm -f elf lab8-1.asm
[arsenii@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[arsenii@fedora lab08]$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
[arsenii@fedora lab08]$
```

Рис. 4.2: Результат работы программы

При изменении кода программы она выходит в бесконечный цикл

При добавлении в программу записи значений и выгрузку в/из стека работа программы корректна (рис. 4.3).

```
[arsenii@fedora lab08]$ nasm -f elf lab8-1.asm
[arsenii@fedora lab08]$ ld -m elf_i386 lab8-1.o -o lab8-1
[arsenii@fedora lab08]$ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0
[arsenii@fedora lab08]$
```

Рис. 4.3: Результат работы программы

Результат работы программы, принимающей на вход аргументы в виде строки и выводящей их на экран. Аргументы разделяются пробелом или заключаются в кавычки (рис. 4.4).

```
[arsenii@fedora lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[arsenii@fedora lab08]$
```

Рис. 4.4: Результат работы программы

Результат работы программы, складывающей числа(рис. 4.5)

```
[arsenii@fedora lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[arsenii@fedora lab08]$
```

Рис. 4.5: Результат работы программы

Результат работы программы, перемножающей числа(рис. 4.6)

```
[arsenii@fedora lab08]$ nasm -f elf lab8-3.asm
[arsenii@fedora lab08]$ ld -m elf_i386 lab8-3.o -o lab8-3
[arsenii@fedora lab08]$ ./lab8-3 1 2 3 5
Результат: 30
```

Рис. 4.6: Результат работы программы

Текст программы, перемножающей числа (рис. 4.7)

```
%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 ; умножаем на промежуточное число
mov esi,eax; перекладываем в esi результат
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. 4.7: Текст программы

5 Самостоятельная работа

Напишите программу, которая находит сумму значений функции для нескольких введенных аргументов. Функция имеет вид f(x) = 8x-3

Код программы(рис. 5.1)

```
%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  регистр eax
mov ebx,8 ; EBX=8
mul ebx ; EAX=EAX*EBX
sub eax,3 ; EAX=EAX-3
add esi,eax ;
; след. аргумент
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. 5.1: Код программы

Результат работы программы (рис. 5.2)

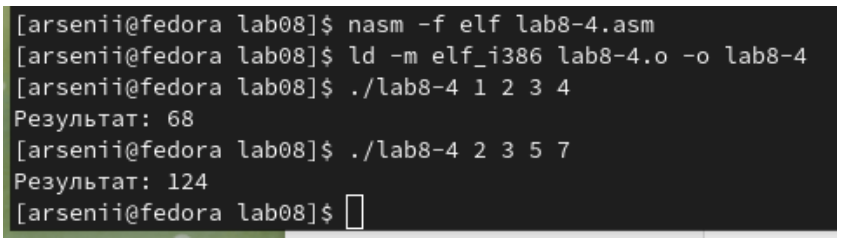

Рис. 5.2: Работа программы

6 Выводы

Освоили написание программ с использованием циклов и обработкой аргументов командной строки

Список литературы