Отчёт по лабораторной работе №8

Программирование цикла. Обработка аргументов командной строки.

Малкина Дарья Александровна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.1 Реализация циклов в NASM

1. Создаём файл lab8-1.asm и вводим в него текст программы из листинга 8.1, после создаём исполнительный файл и запускаем его:

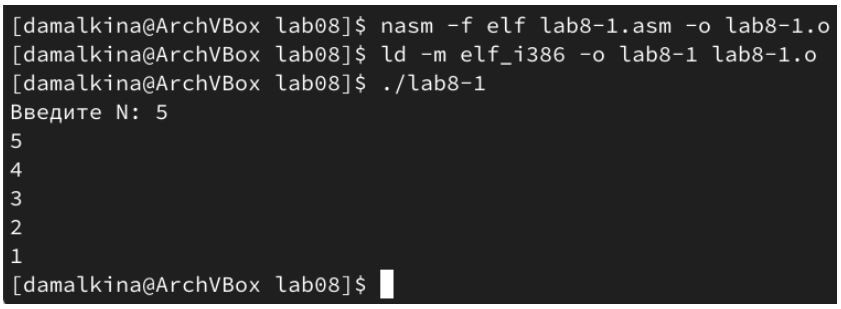

Рис. 2.1: Программа lab8-1

Замечаем, что программа работает некорректно, вносим изменения:

```
; ------ Организация цикла
mov ecx,[N] ; Счетчик цикла, `ecx=N`
label:
sub ecx,1       ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
loop label ; `ecx=ecx-1` и если `ecx` не '0'переход
call quit
```

Рис. 2.2: Изменение программы lab8-1

Создаём исполнительный файли запускаем изменённую программу, в результате получаем бесконечный цикл, который возникает из-за отсутсвия

условия выхода из цикла loop, так как мы переписываем значение N новым значением ecx, то ecx никогда не достигает 0:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-1.asm -o lab8-1.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-1 lab8-1.o
[damalkina@ArchVBox lab08]$ ./lab8-1
Введите N: 5
4
2
0
4294967294
4294967292
4294967290
```

Рис. 2.3: Программа lab8-1 бесконечный цикл

Снова вносим изменения в текст программы, добавляем стек:

```
label:
push ecx         ; добавление значения ecx в стек
sub ecx,1        ; 'ecx=ecx-1'
mov [N],ecx
mov eax,[N]
call iprintLF    ; Вывод значения `N`
pop ecx          ; извлечение значения ecx из стека
loop label       ; `ecx=ecx-1` и если `ecx` не '0'пер
```

Рис. 2.4: Добавление стека в программу lab8-1

Запускаем исправленную программу, теперь вывод верный, программа работает корректно, число проходов цикла соответствует значению N, введенному с клавиатуры:

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

Рис. 2.5: Исправленная программа lab8-1

2.2 Обработка аргументов командной строки

2. Создаём файл lab8-2.asm и вводим в него текст программы из листинга 8.2 после создаём исполняемый файл и запускаем его, указав аргументы:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-2.asm -o lab8-2.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-2 lab8-2.o
[damalkina@ArchVBox lab08]$ ./lab8-2
[damalkina@ArchVBox lab08]$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
[damalkina@ArchVBox lab08]$
```

Рис. 2.6: Программа lab8-2

В итоге программой было обработано четыре аргумента.

3. Создаём ещё файл lab8-3.asm и вводим в него текст программы, которая выводит сумму введённых аргументов, после создаём исполняемый файл и запускаем его, указав аргументы:

```
[damalkina@ArchVBox lab08]$ touch lab8-3.asm
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-3.asm -o lab8-3.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[damalkina@ArchVBox lab08]$ ./lab8-3 12 13 7 10 5
Результат: 47
[damalkina@ArchVBox lab08]$
```

Рис. 2.7: Программа lab8-3 сумма аргументов

Изменим текст программы так, что бы выводом было произведение введённых аргументов:

```
sub ecx,1
mov esi, 1      ; Используем `esi` для хранения промежуточных результатов

next:
cmp ecx,0h
jz _end

pop eax
call atoi
mul esi         ; домножаем промежуточный результат на след. аргумент `esi=esi*eax`
mov esi, eax    ; сохраняем результат в esi
```

Рис. 2.8: Изменение программы lab8-3

Создаём исполнительный файл и запускаем его, указав аргументы:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-3.asm -o lab8-3.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-3 lab8-3.o
[damalkina@ArchVBox lab08]$ ./lab8-3 3 4 2
Результат: 24
[damalkina@ArchVBox lab08]$ ./lab8-3 5 2 3 5
Результат: 150
[damalkina@ArchVBox lab08]$ ./lab8-3 6 4 2 8 10
Результат: 3840
[damalkina@ArchVBox lab08]$
```

Рис. 2.9: Программа lab8-3 произведение аргументов

Проверяем, проведя расчёты вручную, и убеждаемся, что программа работает корректно.

3 Задание для самостоятельной работы

1. Напишем программу, которая будет находить сумму значений f(x)=4*x+3. Зададим сообщения, которые будут выводся по окончании работы программы:

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg1 db "Функция: 4x+3",0
5 msg2 db "Результат: ",0
6 SECTION .text
7 global _start
8
```

Рис. 3.1: msg1, msg2

Пропишем извлечение из стека в ecx количество аргументов и извлечение из стека в edx имя программы, а также переход к следующим аргументам и используем esi для хранения промежуточных результатов:

```
 9 _start:
10
11 pop ecx          ; Извлекаем из стека в `ecx` количество аргументов
12 pop edx          ; Извлекаем из стека в `edx` имя программы
13 sub ecx,1        ; Уменьшаем `ecx` на 1
14 mov esi, 0       ; Используем `esi` для хранения промежуточных результатов
15
```

Рис. 3.2: Извлечение аргументов из стека и переход к след. аргументам

Затем пропишем вычисление, сохраняя результат в esi, и переход к обработке следующего аргумента, если есть ещё аргументы:

```
16 next:
17 cmp ecx,0h      ; проверяем, есть ли еще аргументы
18 jz _end         ; если аргументов нет выходим из цикла
19
20 pop eax         ; иначе извлекаем следующий аргумент из стека
21 call atoi       ; преобразуем символ в число
22 mov ebx, 4      ; 'ebx = 4'
23 mul ebx         ; домножаем '4' на аргумент 'eax=4*eax'
24 add eax, 3      ; прибавляем '3+4*eax'
25 add esi, eax    ; складываем результат с предыдущими результатами
26
27 loop next       ; переход к обработке следующего аргумента
```

Рис. 3.3: Вычисление

Наконец напишем вывод сообщений msg1 и msg2, вывод результата и завершение программы:

```
29 _end:
30 mov eax, msg1    ; вывод сообщения "Функция: 4x+3"
31 call sprintLF
32 mov eax, msg2    ; вывод сообщения "Результат: "
33 call sprint
34 mov eax, esi     ; записываем произведение в регистр `eax
35 call iprintLF    ; печать результата
36 call quit        ; завершение программы
```

Рис. 3.4: Вывод и выход из программы

Создаём исполнительный файл и запускаем его, проверим работу программы с разными аргументами:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-var5.asm -o lab8-var5.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-var5 lab8-var5.o
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2
Функция: 4x+3
Результат: 18
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3 4
Функция: 4x+3
Результат: 52
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3
Функция: 4x+3
Результат: 33
[damalkina@ArchVBox lab08]$ ./lab8-var5 2 5 0
Функция: 4x+3
Результат: 37
[damalkina@ArchVBox lab08]$
```

Рис. 3.5: Программа lab8-var5

Проверяем, проведя расчёты вручную, и убеждаемся, что программа работает корректно.

4 Выводы

В ходе лабораторной работы мы изучили программирование циклов и обработку аргументов командной строки, также попрактиковались работать с циклами, аргументами и ошибками, которые возникают при неверном управлении циклами и стеком.