Отчет по лабораторной работе №8

Дисциплина: Архитектура компьютера

Мутаев Муртазаали Магомедович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

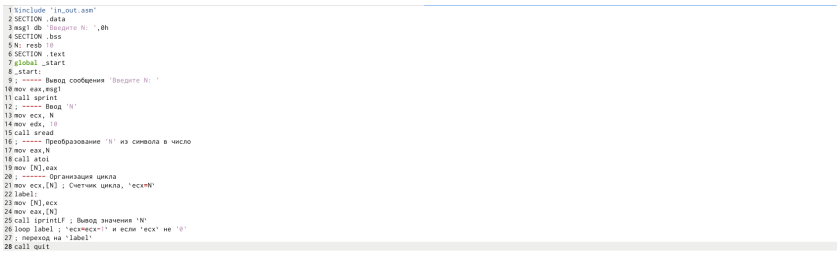
2 Задание

1. Реализация циклов NASM
2. Обработка аргументов командной строки
3. Задание для самостоятельной работы

3 Выполнение лабораторной работы

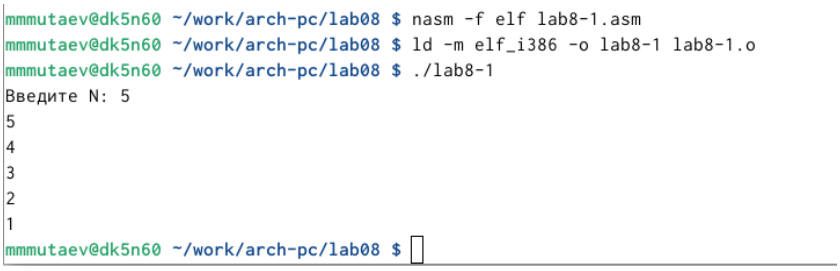
3.1 Реализация циклов NASM

Я создал каталог для программам лабораторной работы № 8, перешел в него и создал файл lab8-1.asm. Далее ввел текст программы из Листинга 8.1, создал исполняемый файл и запустил его:



```
%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. 3.1: Листинг 8.1



```
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
5
4
3
2
1
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $
```

Рис. 3.2: Результат листинга 8.1

Программа выводит значения регистра ecx. Его значение уменьшается на 1 с каждым проходом цикла.

Добавим в тело цикла уменьшение ecx и проверим работу программы:

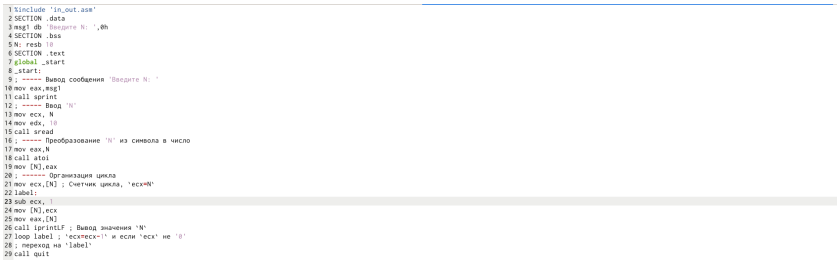


Рис. 3.3: Попытка обмануть Листинг 8.1

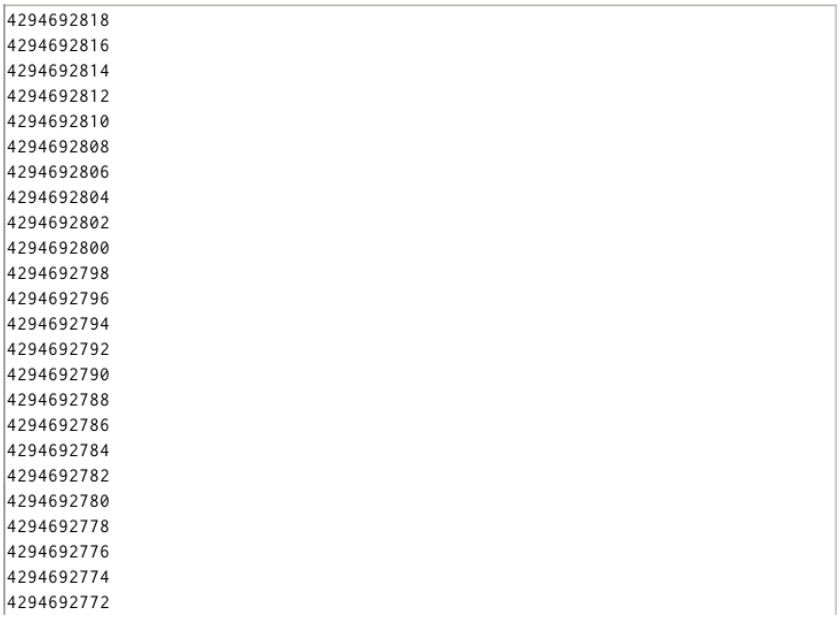


Рис. 3.4: Результат попытки обмануть Листинг 8.1

Рис. 3.5: Удачная попытка обмануть Листинг 8.1

Для использования регистра ecx в цикле и сохранения корректности работы программы можно использовать стек. Внесем изменения в текст программы, до-

бавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop:

```
22 label:
23 push ecx
24 sub ecx,1
25 mov [N],ecx
26 mov eax,[N]
27 call iprintLF
28 pop ecx
29 loop label
```

Рис. 3.6: Использование push и pop

В результате программа выдала мне значение (ecx-1), но в данном случае количество проходов цикла соответствует введенному числу N.

```
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 5
4
3
2
1
0
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $
```

Рис. 3.7: Результат использования push и pop

3.2 Обработка аргументов командной строки

При разработке программ иногда встает 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. Обра-

ботку 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 из Листинга 8.2, которая выводит на экран аргументы 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
```

Рис. 3.8: Листинг 8.2

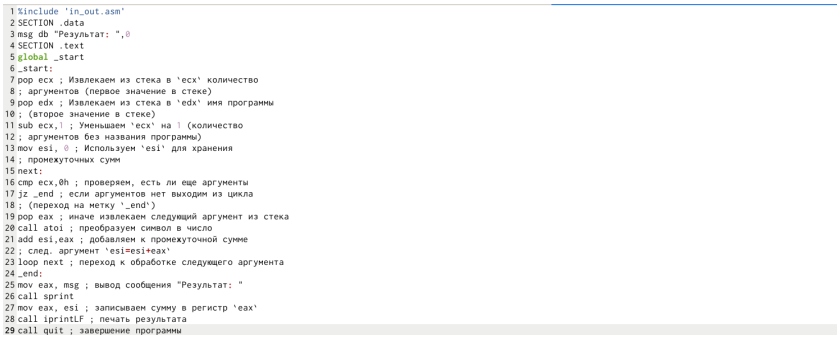
После запустил программу, введя *./lab8-2 аргумент1 аргумент 2 ‘аргумент 3’*. Мне выдало следующий результат:

```
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $
```

Рис. 3.9: Результат листинга 8.2

Таким образом программа обработала 4 аргумента.

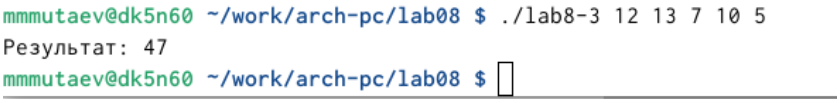
Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Для этого создадим файл lab8-3.asm и введем туда текст из Листинга 8.3:



```
%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. 3.10: Листинг 8.3

В результате программа выводит нам сумму введенных чисел:



```
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $
```

Рис. 3.11: Результат листинга 8.3

Теперь попробуем самостоятельно изменить текст программы так, чтобы в результате у нас выводилось произведение введенных чисел. Для этого я первоначально присвоил esi значение 1, чтобы при умножении числа результат не обнулялся. После этого заменил строку *add esi, eax* на 2 строки:

mul esi

mov esi, eax

Т.е. я сначала умножил eax на esi, а потом присвоил esi значение eax.

```
%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
mul esi
mov esi, eax
; 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. 3.12: Листинг 8.3 (Умножение)

```
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-3 5 6
Результат: 30
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-3 3 4 10
Результат: 120
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-3 10 10 5 1
Результат: 500
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $ ./lab8-3 10 10 5 1 0
Результат: 0
mmmutaev@dk5n60 ~/work/arch-pc/lab08 $
```

Рис. 3.13: Результат Листинга 8.3 (Умножение)

3.3 Задание для самостоятельной работы

Напишите программу, которая находит сумму значений функции *f(x)* для *x = x1, x2, ..., xn*, т.е. программа должна выводить значение *f(x1) + f(x2) + ... + f(xn)*. Значения *xi* 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 = x1, x2, ..., xn*.

Возьмем за основу программу для нахождения суммы аргументов. Вытаскивая каждый аргумент из стека, отредактируем их в соответствии с функцией, т.е. подставим их вместо *x*, и уже измененные значения будем добавлять в переменную *esi*. Вот программа, которая у меня получилась:

```
%include 'in_out.asm'
SECTION .data
msg1 db "Функция f(x) = 3(10+x)", 0
msg2 db "Результат: ", 0
SECTION .text
global _start
_start:

  pop ecx
  pop edx
  sub ecx,1

  mov esi, 0

next:
  cmp ecx,0h
  jz _end
  pop eax
  call atoi
  add eax, 10
  mov ebx, 3
  mul ebx
  add esi, eax
loop next

_end:
  mov eax, msg1
  call sprintLF
  mov eax, msg2
  call sprint
  mov eax, esi
  call iprintLF
  call quit
```

Рис. 3.14: Задание для самостоятельной работы

И вот наш результат:

```
mmmutaev@dk8n60 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
mmmutaev@dk8n60 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
mmmutaev@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3 4
Функция f(x) = 3(10+x)
Результат: 150
mmmutaev@dk8n60 ~/work/arch-pc/lab08 $
```

Рис. 3.15: Задание для самостоятельной работы. Результат

4 Выводы

Я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строки.