Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки.

Чекмарев Александр Дмитриевич | группа: НПИбд 02-23

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

Создадим каталог для программам лабораторной работы № 8, перейдем в него и создадим файл *lab8-1.asm*:

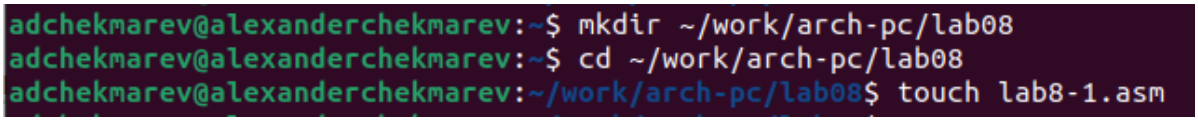

Рис. 2.1: Рис 2.1.1: Создание каталога и файла .asm

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 изучим текст программы (Листинг 8.1).

Введем в файл *lab8-1.asm* текст программы из листинга 8.1.

report.md × lab8-1.asm

```
;-------------------------------------------------------------
; 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

```

Рис. 2.2: Рис 2.1.2: Демонстрация текста программы в файле

Создадим исполняемый файл и запустим его. Результат работы данной программы будет следующим:

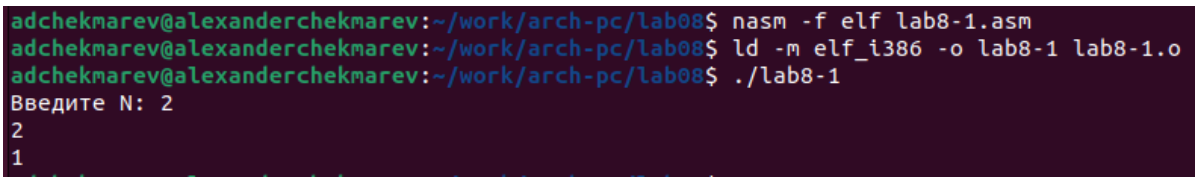

Рис. 2.3: Рис 2.1.3: Создание файла и его проверка

Данный пример показывает, что использование регистра ecx в теле цилка *loop* может привести к некорректной работе программы.

Изменим текст программы добавив изменение значение регистра ecx в цикле:

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

   loop label

   call quit
```

Рис. 2.4: Рис 2.1.4: Демонстрация измененого текста программы

Создадим исполняемый файл и проверим его работу.

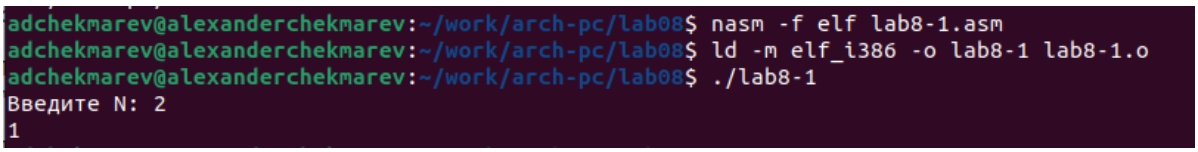

Рис. 2.5: Рис 2.1.5: Создание файла и его проверка (1)

Какие значения принимает регистр ecx в цикле? Соответствует ли число проходов цикла значению N введенному с клавиатуры?

Если ввести 2, то подсчет будет до 1, что не соответсвует. Если к примеру ввести 5, то это явно число проходов цикла НЕ будет соответствовать введенному с клавиатуры значению

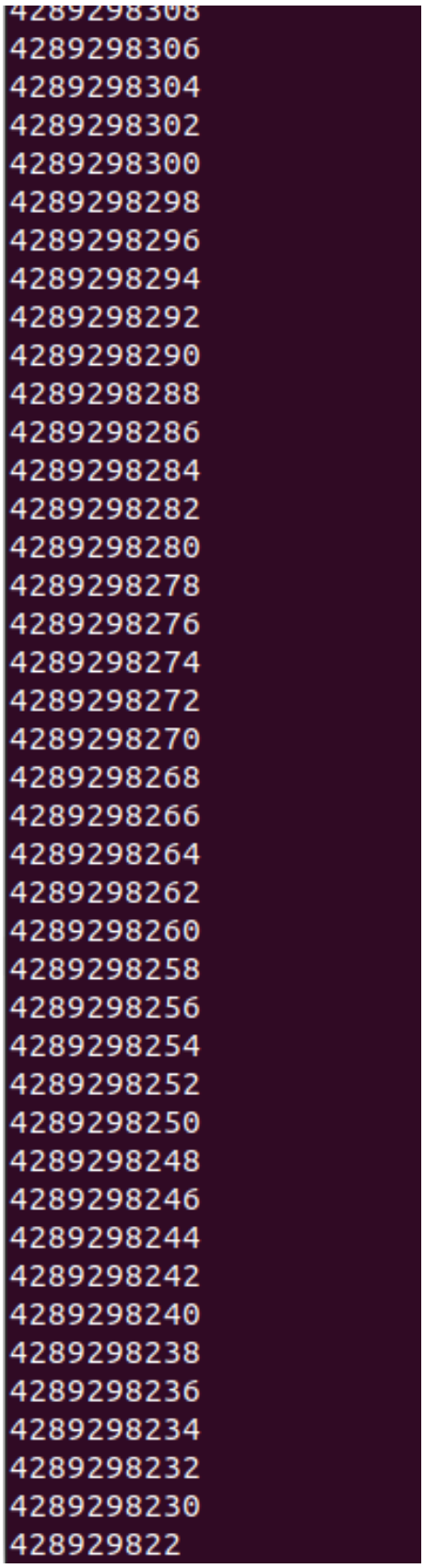

Рис. 2.6: Рис 2.1.6: Проверка с другим значением N=5 (2)

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

```
%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:
   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

   call quit
```

Рис. 2.7: Рис 2.1.7: Демонстрация измененого текста программы

Создадим исполняемый файл и проверим его работу.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 2
1
0
```

Рис. 2.8: Рис 2.1.8: Создание файла и его проверка

Соответствует ли в данном случае число проходов цикла значению N введенному с клавиатуры?

Число проходов цикла будет соответствовать введенному значению с клавиатуры (1-(2), 0-(1))

2.2 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. 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 изучим текст программы (Листинг 8.2)

Создадим файл *lab8-2.asm* в каталоге ~/work/arch-pc/lab08 и введем в него текст

программы из листинга 8.2.

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch lab8-2.asm
```

Рис. 2.9: Рис 2.2.1: Создание файла .asm

```
;
; 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
```

Рис. 2.10: Рис 2.2.2: Демонстрация текста программы

Создадим исполняемый файл и запустим его, указав аргументы:

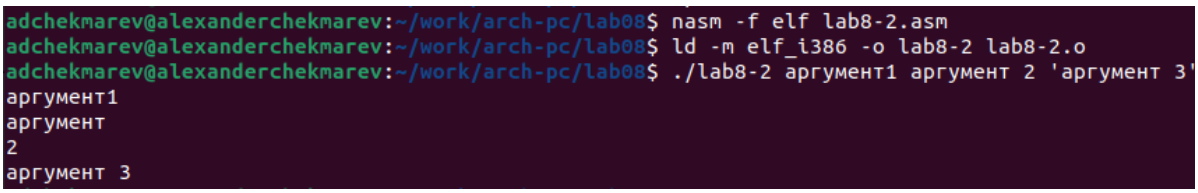

Рис. 2.11: Рис 2.2.3: Создание файла и его проверка

Сколько аргументов было обработано программой?

Всего было обработано 4 аргумента, так как “аргумент” и “2” не взяты в одинарные кавычки, в отличии от 4-го аргумента

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл *lab8-3.asm* в каталоге ~/work/arch-pc/lab08 и введем в него текст программы из листинга 8.3.

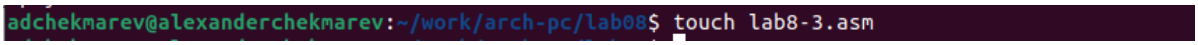

Рис. 2.12: Рис 2.2.4: Создание файла .asm

```
%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. 2.13: Рис 2.2.5: Демонстрация текста программы

Создадим исполняемый файл и запустим его, указав аргументы.

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

Рис. 2.14: Рис 2.2.6: Создание файла и его проверка с указанием аргументов

Изменим текст программы из листинга 8.3 для вычисления произведения

аргументов командной строки.

```
%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. 2.15: Рис 2.2.7: Демонстрация измененого текста программы

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

Рис. 2.16: Рис 2.2.8: Создание файла и его проверка

3 Самостоятельная работа

Задание№1 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.

Создадим новый файл *task1.asm* и напишем программу нахождения суммы значения функции f(x) для варианта 4 (2(x-1)).

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ touch task1.asm
```

Рис. 3.1: Рис 3.1.1: Создание файла

Возьмем за основу код из *lab8-3.asm* и переделаем его

```
%include 'in_out.asm'

SECTION .data
msg 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
   dec eax
   mov ebx,2
   mul ebx
   add esi,eax

   loop next
_end:
   mov eax, msg
   call sprint
   mov eax, esi
   call iprintLF
   call quit
```

Рис. 3.2: Рис 3.1.2: Демонстрация программы для задания

Проверим программу с несколькими значениями x

```
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ nasm -f elf task1.asm
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ld -m elf_i386 -o task1 task1.o
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 4 2 3 14
Результат: 38
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 1 1 1 0
Результат: 4294967294
adchekmarev@alexanderchekmarev:~/work/arch-pc/lab08$ ./task1 12 23 12
Результат: 88
```

Рис. 3.3: Рис 3.1.3: Проверка программы

Программа работает корректно

Загрузим все файлы на github по окончании лаб. работы

```
adchekmarev@alexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git add .
adchekmarev@alexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git commit -am 'fea
t(main): add files lab-8'
[master 75fa307] feat(main): add files lab-8
 24 files changed, 187 insertions(+), 119 deletions(-)
 create mode 100644 labs/lab07/report/image.zip
 delete mode 100644 labs/lab08/report/image/placeimg_800_600_tech.jpg
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.1.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.2.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.3.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.4.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.5.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.6.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.7.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.1.8.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.1.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.2.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.3.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.4.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.5.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.6.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.7.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 2.2.8.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 3.1.1.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 3.1.2.png"
 create mode 100644 "labs/lab08/report/image/\320\240\320\270\321\201 3.1.3.png"
 create mode 100644 labs/lab08/report/report.docx
 rewrite labs/lab08/report/report.md (71%)
 create mode 100644 labs/lab08/report/report.pdf
adchekmarev@alexanderchekmarev:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs$ git push
Перечисление объектов: 39, готово.
Подсчет объектов: 100% (39/39), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (31/31), готово.
Запись объектов: 100% (31/31), 3.25 МиБ | 1.58 МиБ/с, готово.
Всего 31 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:nenokixd/study_2023-2024_arh-pc.git
   185242f..75fa307  master -> master
```

Рис. 3.4: Рис 3.1.4: Загрузка файлов на github

4 Выводы

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