РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

дисциплина: Архитектура компьютера

Студент: Терентьев Максим Сергеевич

Группа: НКАбд-05-25

МОСКВА

2025 г.

Содержание

1. Цель работы

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

2. Порядок выполнения лабораторной работы

2.1. Реализация циклов в NASM

Создадим каталог для выполнения лабораторной работы, перйдем в него и сделаем файл lab8-1.asm, а также скопируем в него файл in_out.asm из прошлой лабораторной работы:

```
msterentjev@rudn:~$ mkdir ~/work/arch-pc/lab08
msterentjev@rudn:~$ cp ~/work/arch-pc/lab07/in_out.asm ~/work/arch-pc/lab08/in_out.asm
msterentjev@rudn:~$ cd ~/work/arch-pc/lab08/
msterentjev@rudn:~/work/arch-pc/lab08$ touch lab8-1.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ls
in_out.asm  lab8-1.asm
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 1. Создание lab8-1.asm и каталога lab08

Введём в созданный файл код из листинша 8.1:

Открыть | *lab8-1.asm ~/work/arch-pc/lab08 | Сохранить

```
;---
; 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'

    call quit
```

Рис. 2. Текст программы lab8-1.asm

Теперь создадим исполняемый файл и проверим его работу:

```
msterentjev@rudn:~/work/arch-pc/lab08$ gedit lab8-1.asm
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f -elf lab8-1.asm
nasm: fatal: unrecognised output format `-elf' - use -hf for a list
Type nasm -h for help.
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 12
12
11
10
9
8
7
6
5
4
3
2
1
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 3. Создание исполн. файла и работа программы lab8-1.asm

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:

Открыть | *lab8-1.asm ~/work/arch-pc/lab08 | Сохранить

```
;---
; 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  ; Вывод значения `N`
    loop label     ; `ecx=ecx-1` и если `ecx` не '0'

    call quit
```

Рис. 4. Измененный текст программы lab8-1.asm

Создадим исполняемый файл и проверим его работу:

```
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-1.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-1
Введите N: 5
4
2
0
4294967294
4294967292
4294967290
4294967288
4294967286
4294967284
4294967282
4294967280
4294967278
4294967276
4294967274
4294967272
4294967270
```

Рис. 5. Работа lab8-1.asm с изменением значение регистра ecx в цикле

Регистр ecx принимает следующие значения (в десятичном представлении):

- Начальное значение: 5
- После первой итерации: 3 (5 - 1 - 1)
- После второй: 1 (3 - 1 - 1)
- После третьей: 4294967295 (1 - 1 - 1, происходит переполнение)
- Далее: 4294967293, 4294967291, ...

Число проходов цикла не соответствует N:

При вводе N = 5 ожидалось 5 проходов цикла, но:

- Фактически цикл становится БЕСКОНЕЧНЫМ
- После трех "корректных" итераций происходит переполнение
- Программа продолжает работать бесконечно с очень большими числами

Изменим программу еще раз, добавив push и pop:

Открыть *lab8-1.asm ~/work/arch-pc/lab08 Сохра

```
;---
; 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:
    push ecx       ; Dobavlenie ecx v stek
    sub ecx,1      ; 'ecx=ecx-1'
    mov [N],ecx
    mov eax,[N]
    call iprintLF  ; Вывод значения `N`
    pop ecx        ; Izvlechenye ecx iz steka
    loop label     ; `ecx=ecx-1` и если `ecx` не '0'

    call quit
```

Рис. 6. Текст программы lab8-1.asm с использованием стека

Еще раз сделаем исполняемый файл и проверим работу программы:

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

Рис. 7. Работа lab8-1.asm с использованием стека

Как видим, программа снова работает корректно. Значение N введенное с клавиатуры совпадает с количеством проходов цикла.

2.2. Обработка аргументов командной строки

Создадим файл lab8-2.asm и вставим в него текст из листинга 8.2:

Открыть *lab8-2.asm ~/work/arch-pc/lab08 Сохранить

```
; ---
; Обработка аргументов командной строки
; ---

%include 'in_out.asm'

SECTION .text
global _start

_start:
    pop ecx     ; Извлекаем из стека в `ecx` количество аргументов
    pop edx     ; Извлекаем из стека в `edx` имя программы
    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 - выход из цикла
    pop eax     ; Иначе извлекаем аргумент из стека
    call sprintLF ; Вызываем функцию печати
    loop next   ; Переход к обработке следующего аргумента

_end:
    call quit
```

Рис. 8. Текст программы lab8-2.asm из листинга 8.2

Создадим исполняемый файл и запустим его, указав аргументы:

```
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-2.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-2
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-2 аргумент1 аргумент 2 'аргумент 3'
аргумент1
аргумент
2
аргумент 3
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 9. Работа lab8-2.asm с аргументами

Как видим, программа обработала 4 аргумента.

Рассмотрим еще один пример программы которая выводит сумму чисел, которые передаются в программу как аргументы. Создадим файл lab8-3.asm и введём в него текст программы из листинга 8.3:

Открыть | *lab8-3.asm ~/work/arch-pc/lab08 | Сохранить

```
%include 'in_out.asm'

SECTION .data
    msg db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx         ; Извлекаем из стека в `ecx` количество аргументов
    pop edx         ; Извлекаем из стека в `edx` имя программы
    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
    pop eax         ; Иначе извлекаем следующий аргумент из стека
    call atoi       ; Преобразуем символ в число
    add esi,eax     ; Добавляем к промежуточной сумме следующий 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. 10. Текст программы lab8-3.asm

Создадим исполняемый файл и проверим его работу:

```
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-3 12 8 10 5
Результат: 35
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 11. Работа программы lab8-3.asm

Как видим, программа работает корректно: 12+8+10+5 действительно равно 35.

Изменим программу для работы с умножением:

Открыть | lab8-3.asm ~/work/arch-pc/lab08 | Сохранить

```
%include 'in_out.asm'

SECTION .data
    msg db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx         ; Извлекаем из стека в `ecx` количество аргументов
    pop edx         ; Извлекаем из стека в `edx` имя программы
    sub ecx,1       ; Уменьшаем `ecx` на 1 (количество аргументов без названия программы)
    mov esi, 1      ; Используем `esi` для хранения произведения (начальное значение 1)

next:
    cmp ecx,0h      ; Проверяем, есть ли еще аргументы
    jz _end         ; Если аргументов нет - выходим из цикла
    pop eax         ; Иначе извлекаем следующий аргумент из стека
    call atoi       ; Преобразуем символ в число
    imul esi, eax   ; Умножаем промежуточное произведение на следующий аргумент
    loop next       ; Переход к обработке следующего аргумента

_end:
    mov eax, msg    ; Вывод сообщения "Результат: "
    call sprint
    mov eax, esi    ; Записываем произведение в регистр `eax`
    call iprintLF   ; Печать результата
    call quit       ; Завершение программы
```

Рис. 12. Измененный код lab8-3.asm под умножение

Создадим исполняемый файл и проверим его работу:

```
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./lab8-3 8 5 10
Результат: 400
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 13. Работа lab8-3.asm с умножением

Как видим, все работает корректно.

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 № 6. Создайте исполняемый файл и проверьте его работу на нескольких наборах x = x1,x2,...,xn.

Мой вариант – 11, значит функция - 15x +2.

Создадим файл labobus8.asm и вставим в него код получившейся программы:

Открыть | *labobus8.asm ~/work/arch-pc/lab08 | Сохранить

```
%include 'in_out.asm'

SECTION .data
    func_msg db "Функция: f(x)=15x+2",0
    result_msg db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx             ; Извлекаем количество аргументов
    pop edx             ; Извлекаем имя программы
    sub ecx, 1          ; Уменьшаем количество аргументов (без имени программы)
    mov esi, 0          ; Инициализируем сумму результатов функции

    ; Выводим информацию о функции
    mov eax, func_msg
    call sprintLF

    cmp ecx, 0          ; Проверяем, есть ли аргументы
    jz _end             ; Если нет аргументов - выходим

next:
    pop eax             ; Извлекаем аргумент из стека
    call atoi           ; Преобразуем в число

    ; Вычисляем f(x) = 15x + 2
    mov ebx, eax        ; Сохраняем x в ebx
    imul eax, 15        ; Умножаем на 15: eax = 15x
    add eax, 2          ; Прибавляем 2: eax = 15x + 2

    add esi, eax        ; Добавляем результат к общей сумме

    loop next           ; Переход к следующему аргументу

_end:
    mov eax, result_msg ; Вывод сообщения "Результат: "
    call sprint
    mov eax, esi        ; Записываем сумму в eax
    call iprintLF       ; Выводим результат
    call quit           ; Завершаем программу
```

Рис. 14. Код программы labobus8.asm

Создадим исполняемый файл и проверим работу программы:

```
msterentjev@rudn:~/work/arch-pc/lab08$ nasm -f elf labobus8.asm
msterentjev@rudn:~/work/arch-pc/lab08$ ld -m elf_i386 -o labobus8 labobus8.o
msterentjev@rudn:~/work/arch-pc/lab08$ ./labobus8 1 2 3 4
Функция: f(x)=15x+2
Результат: 158
msterentjev@rudn:~/work/arch-pc/lab08$ ./labobus8 0 4 1 5
Функция: f(x)=15x+2
Результат: 158
msterentjev@rudn:~/work/arch-pc/lab08$ ./labobus8 1 1 1 1
Функция: f(x)=15x+2
Результат: 68
msterentjev@rudn:~/work/arch-pc/lab08$
```

Рис. 15. Работа программы labobus8.asm

Первый случай: 15*1+2 + 15*2+2 + 15*3+2 + 15*4+2 = 17 + 32 + 47 + 62 = 158 =>верно

Второй случай: 15*0+2 + 15*4+2 + 15*1+2 + 15*5+2 = 2 + 62 + 17 + 77 = 158 => верно
Третий случай: 15*1+2 + 15*1+2 + 15*1+2 + 15*1+2 = 17 + 17 + 17 + 17 = 68 => верно
Как мы видим, программа работает корректно.

Выводы

Приобретены навыки работы с циклами и аргументами командной строки в ассемблере. Реализована программа вычисления суммы значений функции для заданных аргументов.