Отчет по лабораторной работе №8

Дисциплина: архитектура компьютера

Горбунова Яна Сергеевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация циклом в NASM
2. Обработка аргументов командной строки
3. Самостоятельное написание программы по материалам лабораторной работы

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.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создаю каталог для программ лабораторной работы №8 (рис. 4.1).

```
ysgorbunova@dk8n60 ~ $ mkdir ~/work/arch-pc/lab08
ysgorbunova@dk8n60 ~ $ cd ~/work/arch-pc/lab08
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 4.1: Создание каталога

Копирую в созданный файл программу из листинга. (рис. 4.2).

```
%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
```

Рис. 4.2: Копирование программы из листинга

Запускаю программу, она показывает работу циклов в NASM (рис. 4.3).

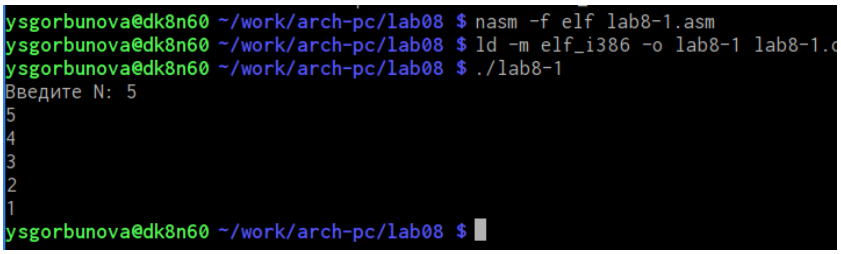

Рис. 4.3: Запуск программы

Заменяю программу изначальную так, что в теле цикла я изменяю значение регистра ecx (рис. 4.4).

```
%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
```

Рис. 4.4: Изменение программы

Из-за того, что теперь регистр ecx на каждой итерации уменьшается на 2 значения, количество итераций уменьшается вдвое (рис. 4.5).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 6
5
3
1
```

Рис. 4.5: Запуск измененной программы

Добавляю команды push и pop в программу (рис. 4.6).

```
%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 ; Инициализируем esi значением 1 для произведения
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
imul esi,eax ; умножаем промежуточное произведение
; на текущий аргумент esi=esi*eax
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. 4.6: Добавление push и pop в цикл программы

Теперь количество итераций совпадает введенному N, но произошло смещение выводимых чисел на -1 (рис. 4.7).

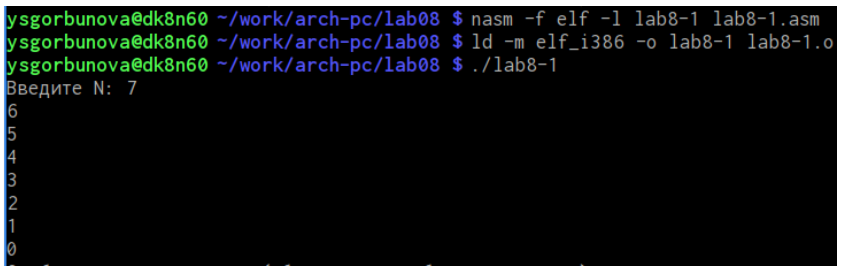

Рис. 4.7: Запуск измененной программы

4.2 Обработка аргументов командной строки

Создаю новый файл для программы и копирую в него код из следующего листинга (рис. 4.8).

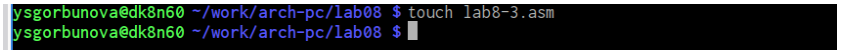

Рис. 4.8: Копирование программы из листинга

Компилирую программу и запускаю, указав аргументы. Программой было обратоно то же количество аргументов, что и было введено (рис. 4.9).

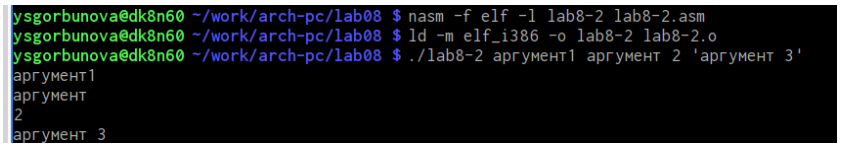

Рис. 4.9: Запуск второй программы

Создаю новый файл для программы и копирую в него код из третьего листинга (рис. 4.10).

```
%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 ; Инициализируем esi значением 1 для произведения
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
imul esi,eax ; умножаем промежуточное произведение
; на текущий аргумент esi=esi*eax
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. 4.10: Копирование программы из третьего листинга

Компилирую программу и запускаю, указав в качестве аргументов некоторые числа, программа их складывает (рис. 4.11).

```
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 47
ysgorbunova@dk8n60 ~/work/arch-pc/lab08 $
```

Рис. 4.11: Запуск третьей программы

Изменяю поведение программы так, чтобы указанные аргументы она умножала, а не складывала (рис. 4.12).

```
%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`
```

Рис. 4.12: Изменение третьей программы

Программа действительно теперь умножает данные на вход числа (рис. 4.13).

```
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ nasm -f elf -l lab8-3 lab8-3.asm
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 5 7 15
Результат: 525
```

Рис. 4.13: Запуск измененной третьей программы

4.3 Задание для самостоятельной работы

Пишу программму, которая будет находить сумма значений для функции f(x) = 10x-4, которая совпадает с моим девытым варинтом (рис. 4.14).

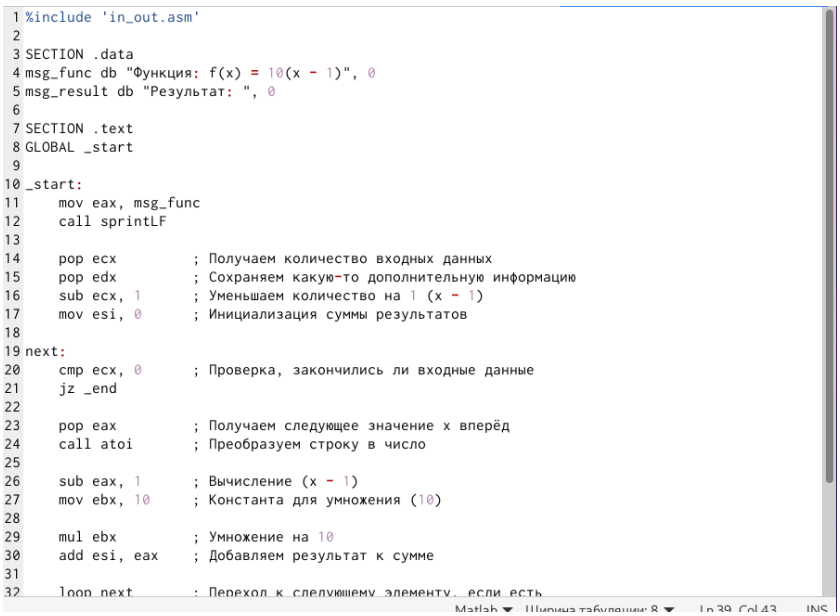

Рис. 4.14: Написание программы для самостоятельной работы

Код программы:

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x) = 10(x - 1)", 0
msg_result db "Результат: ", 0

SECTION .text
GLOBAL _start

_start:
    mov eax, msg_func
    call sprintLF

    pop ecx          ; Получаем количество входных данных
    pop edx          ; Сохраняем какую-то дополнительную информацию
```

```
    sub ecx, 1      ; Уменьшаем количество на 1 (x - 1)
    mov esi, 0      ; Инициализация суммы результатов

next:
    cmp ecx, 0      ; Проверка, закончились ли входные данные
    jz _end

    pop eax         ; Получаем следующее значение x вперёд
    call atoi       ; Преобразуем строку в число

    sub eax, 1      ; Вычисление (x - 1)
    mov ebx, 10     ; Константа для умножения (10)

    mul ebx         ; Умножение на 10
    add esi, eax    ; Добавляем результат к сумме

    loop next       ; Переход к следующему элементу, если есть

_end:
    mov eax, msg_result
    call sprint      ; Печатаем сообщение с результатом
    mov eax, esi     ; Перемещаем общую сумму в eax
    call iprintLF    ; Печатаем результат
    call quit        ; Завершаем программу
```

Проверяю работу программы, указав в качестве аргумента несколько чисел (рис. 4.15).

```
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $ ./lab8-3 12 13 7 10 5
Результат: 54600
ysgorbunova@dk3n55 ~/work/arch-pc/lab08 $
```

Рис. 4.15: Запуск программы для самостоятельной работы

5 Выводы

В результате выполнения данной лабораторной работы я приобрела навыки написания программ с использованием циклов а также научилась обрабатывать аргументы командной строки.

6 Список литературы

1. Курс на ТУИС
2. Лабораторная работа №8
3. Программирование на языке ассемблера NASM Столяров А. В.