

Лабораторная работа №8

Дисциплина: Архитектура компьютера

Серебрякова Дарья Ильинична

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Реализация циклов в NASM	9
4.2	Обработка аргументов командной строки	12
5	Выполнение заданий для самостоятельной работы	15
6	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Создание каталога для работы	9
4.2	Программа из листинга	10
4.3	Запуск программы	10
4.4	Запуск измененной программы	11
4.5	Измененный текст программы	11
4.6	Проверка работы программы	12
4.7	Написание программы в новом файле	12
4.8	Обработка заданных аргументов	13
4.9	Программа по предложенному листингу	13
4.10	Подсчет суммы заданных аргументов	14
4.11	Измененный текст программы	14
4.12	Подсчет произведения заданных аргументов	14
5.1	Текст программы по заданию для самостоятельной работы	15
5.2	Проверка работы написанной программы	16

Список таблиц

1 Цель работы

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

2 Задание

1. Реализовать циклы в `asm`
2. Познакомиться с обработкой аргументов командной строки

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. На рис. 8.1 показана схема организации стека в процессоре. Стек имеет вершину, адрес последнего добавленного элемента, который хранится в регистре esp (указатель стека). Противоположный конец стека называется дном. Значение, помещённое в стек последним, извлекается первым. При помещении значения в стек указатель стека уменьшается, а при извлечении — увеличивается. Для стека существует две основные операции: • добавление элемента в вершину стека (push); • извлечение элемента из вершины стека (pop).

Команда push размещает значение в стеке, т.е. помещает значение в ячейку памяти, на которую указывает регистр esp, после этого значение регистра esp увеличивается на 4. Данная команда имеет один операнд — значение, которое необходимо поместить в стек.

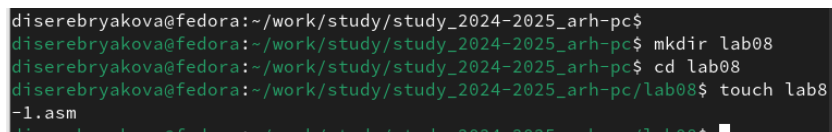
Команда pop извлекает значение из стека, т.е. извлекает значение из ячейки памяти, на которую указывает регистр esp, после этого уменьшает значение регистра esp на 4. У этой команды также один операнд, который может быть регистром или переменной в памяти. Нужно помнить, что извлечённый из стека

элемент не стирается из памяти и остаётся как “мусор”, который будет перезаписан при записи нового значения в стек.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

Создала каталог для программ лабораторной работы 8, перешла в него и создала файл lab8-1.asm (рис. 4.1).



```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$  
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ mkdir lab08  
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc$ cd lab08  
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ touch lab8-1.asm  
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 4.1: Создание каталога для работы

В только что созданный файл вписываю программу из предложенного листинга (рис. 4.2).

```

/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab08/lab8-1.asm
%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. 4.2: Программа из листинга

Создаю исполняемый файл и запускаю его (рис. 4.3).

```

diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-1 la
b8-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-1
bash: ./lab8-1: Нет такого файла или каталога
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-1
Введите N: 8
8
7
6
5
4
3
2
1
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$

```

Рис. 4.3: Запуск программы

Поменяла программу, добавив изменение значение регистра ecx в цикле. Создала исполняемый файл и запустила его (рис. 4.4).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-1 la
b8-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-1
Введите N: 6
5
3
1
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 4.4: Запуск измененной программы

Регистр `ecx` в цикле принимает значения «через одно», то есть каждое следующее значение меньше предыдущего на 2. За счет этого число проходов цикла в два раза меньше введенного с клавиатуры числа `N`

Для использования регистра `ecx` в цикле и сохранения корректности работы программы можно использовать стек. Вношу изменения в текст программы добавив команды `push` и `pop` (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла `loop` (рис. 4.5).

```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab08/lab8-1.asm
%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
sub ecx,1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения `N`
pop ecx
loop label ; `ecx=ecx-1` и если `ecx` не '0'
call quit
```

Рис. 4.5: Измененный текст программы

Создаю исполняемый файл и запускаю его. Теперь число проходов цикла соответствует введенному с клавиатуры значению `N`, но выводимые числа теперь

начинаются от N-1 до 0 (рис. 4.6).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-1.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-1 la
b8-1.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-1
Введите N: 6
5
4
3
2
1
0
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-1
Введите N: 7
6
5
4
3
2
1
0
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 4.6: Проверка работы программы

4.2 Обработка аргументов командной строки

Создаю файл lab8-2.asm и ввожу в него текст программы из предложенного листинга (рис. 4.7).

```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab08
%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 sprintf ; вызываем функцию печати
loop next ; переход к обработке следующего
           ; аргумента (переход на метку `next`)
_end:
    call quit
```

Рис. 4.7: Написание программы в новом файле

Создаю исполняемый файл и запускаю его, указав предложенные аргументы.

Командой было обработано 4 аргумента – столько же, сколько и введено (рис. 4.8).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-2.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-2
arg1
arg
2
arg3
```

Рис. 4.8: Обработка заданных аргументов

Командой touch создаю файл lab8-3.asm и ввожу в него текст программы из предложенного листинга (рис. 4.9).

```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/
%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. 4.9: Программа по предложенному листингу

Создаю исполняемый файл и запускаю его, указав несколько чисел в качестве аргументов. Программа считает их сумму и выводит результат на экран (рис.

4.10).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-3.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-3 la
b8-3.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-3 6 7 2
Результат: 15
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 4.10: Подсчет суммы заданных аргументов

Изменяю текст программы так, чтобы она считала произведение введенных аргументов и выводила на экран (рис. 4.11).

```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab08/lab8-3.asm  Измен
%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, eax
mov esi,eax ; умножаем промежуточную сумму на значение аргумента
; 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. 4.11: Измененный текст программы

Создаю исполняемый файл и запускаю его. Теперь программа перемножает заданные аргументы (рис. 4.12).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-3.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-3 la
b8-3.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-3 3 4 2
Результат: 24
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 4.12: Подсчет произведения заданных аргументов

5 Выполнение заданий для самостоятельной работы

Командой touch создаю файл lab8-4.asm и прописываю в нем текст программы, которая будет подсчитывать сумму значений функции при заданных аргументах (рис. 5.1).

```
/home/diserebryakova/work/study/study_2024-2025_arh-pc/lab08/lab8-4.asm
%include 'in_out.asm'
SECTION .data
msg1 db "Функция f(x)=7*(x+1) "
msg2 db "Результат: ",0
SECTION .text
global _start
_start:
mov eax,msg1
call sprintf
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 eax,1 ; eax = x+1
mov ebx,7
mul ebx ; eax = eax*ebx = (x+1)*7
add esi,eax
loop next ; переход к обработке следующего аргумента
_end:
mov eax, msg2 ; вывод сообщения "Результат: "
call sprintf
mov eax, esi ; записываем сумму в регистр 'eax'
call iprintfLF ; печать результата
call quit ; завершение программы
```

Рис. 5.1: Текст программы по заданию для самостоятельной работы

Создаю исполняемый файл и запускаю его несколько раз на разных значениях аргументов. Результат получается верный, значит программа написана корректно (рис. 5.2).

```
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ nasm -f elf lab8-4.asm
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ld -m elf_i386 -o lab8-4 lab8-4.o
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-4 1 2
Функция  $f(x)=7*(x+1)$  Результат:
Результат: 35
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$ ./lab8-4 1 2 1
Функция  $f(x)=7*(x+1)$  Результат:
Результат: 49
diserebryakova@fedora:~/work/study/study_2024-2025_arh-pc/lab08$
```

Рис. 5.2: Проверка работы написанной программы

6 Выводы

Приобретены навыки написания программ с использованием циклов и обработкой аргументов командной строки

Список литературы

1. Лабораторная работа 8