

Лабораторная работа №8

Программирование цикла. Обработка аргументов командной строки

Карпова Есения Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	19
	Список литературы	20

Список иллюстраций

4.1	Создание каталога и файла	9
4.2	Запуск исполняемого файла	9
4.3	Запуск исполняемого файла	10
4.4	Измененный текст программы	11
4.5	Запуск исполняемого файла	12
4.6	Текст программы файла lab8-2.asm	13
4.7	Запуск исполняемого файла	13
4.8	Запуск исполняемого файла	14
4.9	Измененный текст программы	15
4.10	Запуск исполняемого файла	16
4.11	Программа, находящая сумму значений функции	17
4.12	Запуск исполняемого файла	18

Список таблиц

1 Цель работы

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

2 Задание

1. Реализация циклов в NASM
2. Обработка аргументов командной строки
3. Задание для самостоятельной 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. 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. push -10; Поместить -10 в стек push ebx; Поместить значение регистра ebx в стек push [buf]; Поместить значение переменной buf в стек push word [ax] ; Поместить в стек слово по адресу в ax

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 pop eax; Поместить значение из стека в

регистр `eax` `pop [buf]`; Поместить значение из стека в `pop word[si]`; Поместить значение из стека в слово по адресу в `si`

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре `ecx`. Наиболее простой является инструкция `loop mov ecx, 100`; Количество проходов `NextStep: ...`; тело цикла `... loop NextStep`; Повторить `ecx` раз от метки `NextStep`

4 Выполнение лабораторной работы

Создаю каталог для программ лабораторной работы № 8, перехожу в него и создаю файл lab8-1.asm (рис. 4.1).

```
eakarpova@dk6n45 ~ $ mkdir ~/work/arch-pc/lab08
eakarpova@dk6n45 ~ $ cd ~/work/arch-pc/lab08
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ touch lab8-1.asm
```

Рис. 4.1: Создание каталога и файла

После того, как я ввела в файл текст программы из листинга, создаю исполняемый файл, чтобы проверить его работу(рис. 4.2).

```
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
```

Рис. 4.2: Запуск исполняемого файла

Программа вводит числа от N до 1 включительно Данный пример показывает, что использование регистра esx в теле цикла loop может привести к некорректной

работе программы. Изменяю текст программы добавив значение регистра `ecx` в цикле и запускаю исполняемый файл, чтобы проверить его работу (рис. 4.3).

```
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
7
5
3
1
```

Рис. 4.3: Запуск исполняемого файла

В данном случае число проходов цикла не соответствует введенному с клавиатуры значению. Вношу изменения в текст программы, добавив команды `push` и `pop` для сохранения значения счетчиков цикла `loop` (рис. 4.4).

```

;-----
; 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      ; добавление значения ecx в стек
sub ecx, 1    ; ecx=ecx-1
mov [N],ecx
mov eax,[N]
call iprintLF ; Вывод значения 'N'
pop ecx       ; извлечение значения ecx из стека
loop label    ; 'ecx=ecx-1' и если 'ecx' не '0'
              ; переход на 'label'
call quit

```

Рис. 4.4: Измененный текст программы

Создаю исполняемый файл и проверяю его работу (рис. 4.5).

```
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ nasm -f elf lab8-1.asm
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-1 lab8-1.o
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
```

Рис. 4.5: Запуск исполняемого файла

В данном случае число проходов цикла соответствует введенному с клавиатуры значению. На выходе пользователь получает числа от N-1 до 0 включительно

2. Обработка аргументов командной строки

Создаю файл lab8-2.asm и ввожу в него текст программы из листинга 8.2.(рис. 4.6).

```

;-----
; 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 printf; вызываем функцию печати
loop next; переход к обработке следующего
        ; аргумента (переход на метку 'next')
_end:
call quit

```

Рис. 4.6: Текст программы файла lab8-2.asm

Создаю исполняемый файл и запускаю его, указав случайные аргументы (рис. 4.7).

```

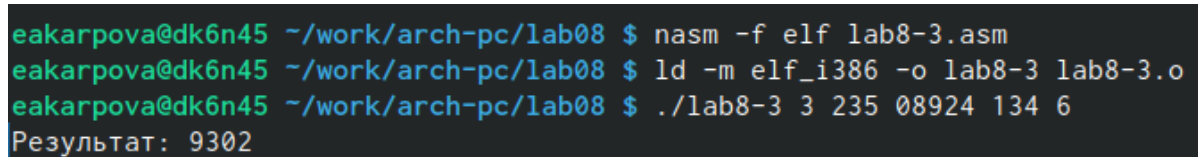
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ nasm -f elf lab8-2.asm
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-2 lab8-2.o
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ./lab8-2 5 6 'что'
5
6
что

```

Рис. 4.7: Запуск исполняемого файла

Программа вывела три введенных ей аргумента.

Рассмотрим пример программы, которая выводит сумму чисел, которые передаются в программу как аргументы. Создаю файл lab8-3.asm и ввожу в него текст программы из листинга 8.3. Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.8).



```
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
eakarpova@dk6n45 ~/work/arch-pc/lab08 $ ./lab8-3 3 235 08924 134 6
Результат: 9302
```

Рис. 4.8: Запуск исполняемого файла

Изменяю текст программы из листинга 8.3 для вычисления произведения аргументов командной строки(рис. 4.8).

```

%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 произведений
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      ; добавляем к промежуточному произведению
              ; след. аргумент 'esi=esi*eax'
mov 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. 4.9: Измененный текст программы

Создаю исполняемый файл и запускаю его, указав аргументы (рис. 4.10).

```
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-3.asm
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-3 lab8-3.o
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ ./lab8-3 1 2 3 4 5
Результат: 120
```

Рис. 4.10: Запуск исполняемого файла

3. Задание для самостоятельной работы

Пишу текст программы, которая находит сумму значений функции $f(x) = 10x - 4$ в соответствии с моим девятым вариантом(рис. 4.11).


```

%include          'in_out.asm'

SECTION .data
msg1: DB 'Функция: f(x) = 10x - 4 ', 0
msg2: DB 'Результат: ', 0

SECTION .text
global _start

_start:
pop ecx          ; извлекаем из стека в 'ecx'
pop edx          ; извлекаем из стека в 'edx'
sub ecx, 1       ; уменьшение 'ecx' на 1
mov esi, 0       ; использование esi для хранения промежуточных сумм
mov edi, 10      ; хранение 10 в edi

next:
cmp ecx, 0h      ; проверяем, есть ли еще аргументы
jz _end          ; если аргументов нет - выходим из цикла
pop eax          ; иначе извлекаем следующий аргумент
call atoi
mul edi          ; eax = eax * edi
add eax, -4      ; eax = eax - 4
add esi, eax     ; добавление к промежуточной сумме
loop next

_end:
mov eax, msg1
call sprintf
mov eax, msg2
call sprintf
mov eax, esi
call iprintLF
call quit

```

Рис. 4.11: Программа, находящая сумму значений функции

Запускаю исполняемый файл, чтобы проверить работу программы (рис. 4.12).

```
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ nasm -f elf lab8-4.asm
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ ld -m elf_i386 -o lab8-4 lab8-4.o
eakarpova@dk8n62 ~/work/arch-pc/lab08 $ ./lab8-4 1 2 3
Функция:  $f(x) = 10x - 4$ 
Результат: 48
```

Рис. 4.12: Запуск исполняемого файла

5 Выводы

В ходе выполнения лабораторной работы я приобрела навыки написания программ с использованием циклов и обработкой аргументов командной строки.

Список литературы

Лабораторная работа №8 - Демидова А.В.