Отчёт по лабораторной работе №8

Петлин Артём Дмитриевич

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Задание

1. Напишите программу, которая находит сумму значений функции □(□) для □ = □1, □2, ..., □□, т.е. программа должна выводить значение □(□1) + □(□2) + ... + □(□□). Значения □□ передаются как аргументы. Вид функции □(□) 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 □ = □1, □2, ..., □□.

   Пример работы программы для функции □(□) = □ + 2 и набора □1 = 1, □2 = 2, □3 = 3,

   ```
   user@dk4n31:~$ ./main 1 2 3 4
   Функция: f(x)=x+2
   Результат: 18
   user@dk4n31:~$
   ```

3 Теоретическое введение

3.1 Организация стека

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.
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).

3.1.1 Добавление элемента в стек.

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.

Примеры:

```
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
```

Существует ещё две команды для добавления значений в стек. Это команда pusha, которая помещает в стек содержимое всех регистров общего назначения в следующем порядке: ax, cx, dx, bx, sp, bp, si, di. А также команда pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов.

3.1.2 Извлечение элемента из стека

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.

Примеры:

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

Аналогично команде записи в стек существует команда popa, которая восстанавливает из стека все регистры общего назначения, и команда popf для перемещения значений из вершины стека в регистр флагов.

3.2 Инструкции организации циклов

Для организации циклов существуют специальные инструкции. Для всех инструкций максимальное количество проходов задаётся в регистре ecx. Наиболее простой является инструкция loop. Она позволяет организовать безусловный цикл, типичная структура которого имеет следующий вид:

```
mov ecx, 100 ; Количество проходов
NextStep:
...
... ; тело цикла
...
loop NextStep ; Повторить `ecx` раз от метки NextStep
```

Иструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

4 Выполнение лабораторной работы

4.1 Реализация циклов в NASM

```
petlin@fedora:~$ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab08
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ls
lab8-1.asm  presentation  report
```

```
lab8-1.asm          [----]  0 L:[  1+28  29/ 29] *(663 / 663b) <EOF>
%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
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-1
Введите N: 10
10
9
8
7
6
5
4
3
2
1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Переходим в каталог ЛБ8 и создаём файл lab8-1.asm, в который вводим текст программы из листинга 8.1. Создаём исполняем файл и проверяем его работу.

```
lab8-1.asm          [B---]  0 L:[  1+22  23/ 30] *(489 / 688b) 0032 0x020
%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'
<------>    ; переход на `label`
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-1
Введите N: 10
9
7
5
3
1
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Изменяем текст программы добавив изменение значение регистра ecx в цикле. Создаём исполняем файл и проверяем его работу. На вход подается число 10, в цикле label регистр ecx уменьшается на 2. Число проходов цикла не соответсвует числу N, так как уменьшается на 2.

```
lab8-1.asm          [----]  0 L:[  1+31  32/ 32] *(749 / 749b) <EOF>
%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 ; Вывод значения `N`
 pop ecx
 loop label ; `ecx=ecx-1` и если `ecx` не '0'
<------>    ; переход на `label`
 call quit
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-1.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-1 lab8-1.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-1
Введите N: 10
9
8
7
6
5
4
3
2
1
0
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Вносим изменения в текст программы добавив команды push и pop (добавления в стек и извлечения из стека) для сохранения значения счетчика цикла loop. Создаём исполняем файл и проверяем его работу. В данном случае число проходов цикла соответсвует числу N.

4.2 Обработка аргументов командной строки

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2.asm  presentation  report
```

```
lab8-2.asm          [----]  0 L:[  1+20  21/ 21] *(959 / 959b) <EOF>
%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
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-2.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-2 lab8-2.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-2 1 5 '7'
1
5
7
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Создаём файл lab8-2.asm, в который вводим текст программы из листинга 8.2. Создаём исполняем файл и проверяем его работу. Программа обработала 3 аргумента.

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ touch lab8-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ls
in_out.asm  lab8-1  lab8-1.asm  lab8-1.o  lab8-2  lab8-2.asm  lab8-2.o  lab8-3.asm  presentation  report
```

```
lab8-3.asm          [-M--]  0 L:[  1+29  30/ 30] *(1452/1452b) <EOF>
%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
```

```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Создаём файл lab8-3.asm, в который вводим текст программы из листинга 8.3. Создаём исполняем файл и проверяем его работу. Программа работает корректно.

```
lab8-3.asm          [-M--]  0 L:[  1+27  28/ 28] *(873 / 873b) <EOF>
%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
next:
 cmp ecx,0h
 jz _end
 pop eax
 call atoi
 mul esi
 mov esi,eax
 loop next
_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
```

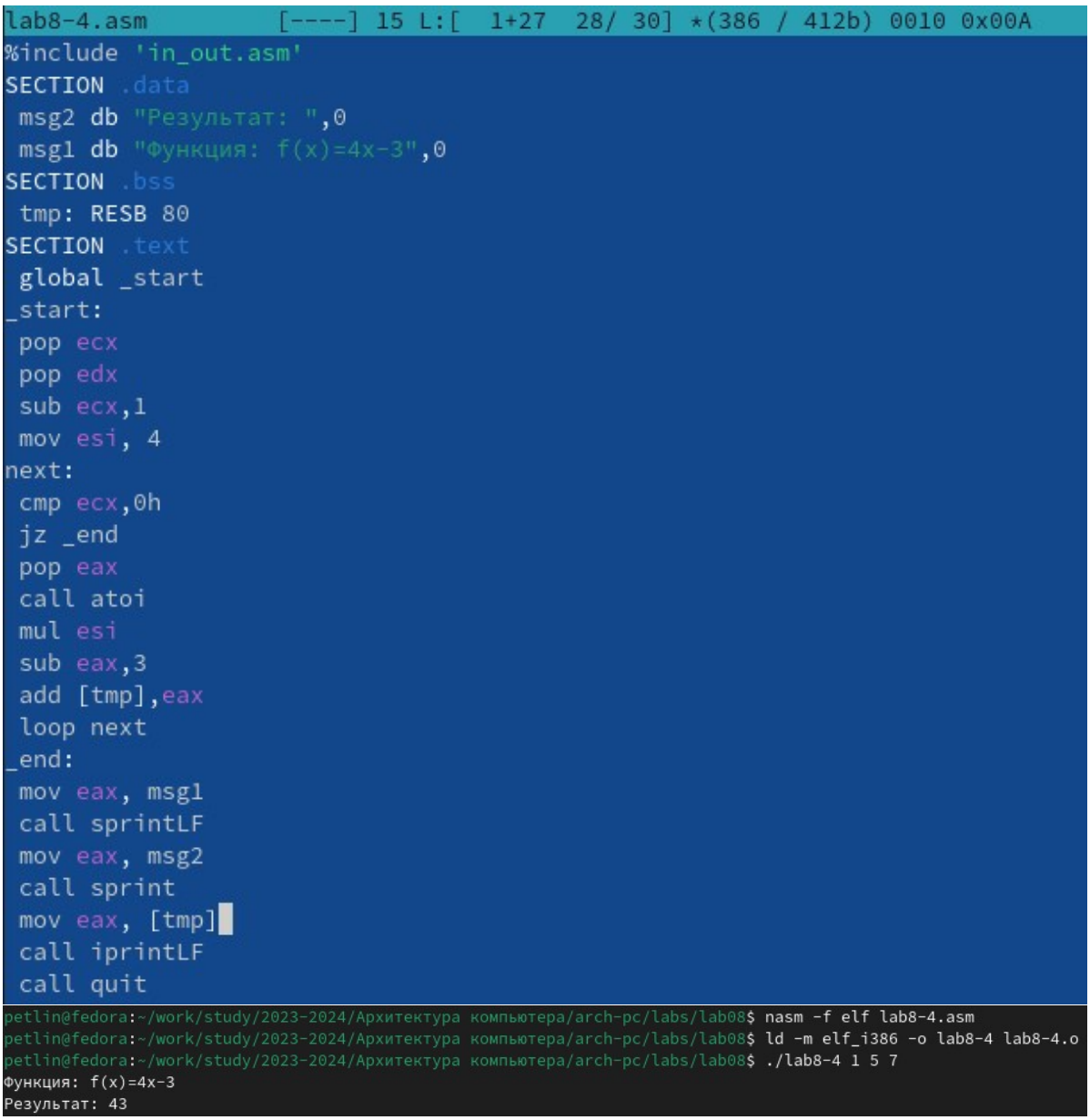
```
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ nasm -f elf lab8-3.asm
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$ ./lab8-3 2 5 7
Результат: 70
petlin@fedora:~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab08$
```

Изменяем текст программы из листинга 8.3 для вычисления произведения аргументов командной строки. Создаём исполняем файл и проверяем его работу. Программа работает корректно.

5 Задания для самостоятельной работы

Вариант №6

Функция: f(x)=4x-3:



Создаём файл lab8-4.asm, в который вводим текст программы, вычисляющей сумму значений функции для аргументов. Создаём исполняем файл и проверяем его работу. Программа работает корректно.

6 Выводы

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

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 c. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 c. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).