Лабораторная работа №8.

Программирование цикла. Обработка аргументов командной строки

Хорошева Алёна Евгеньевна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоить навыки написания программ, используя циклы и обработку аргументов командной строки.

2 Задание

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

3 Теоретическое введение

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).

2. **Добавление элементы 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. **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.

4. **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

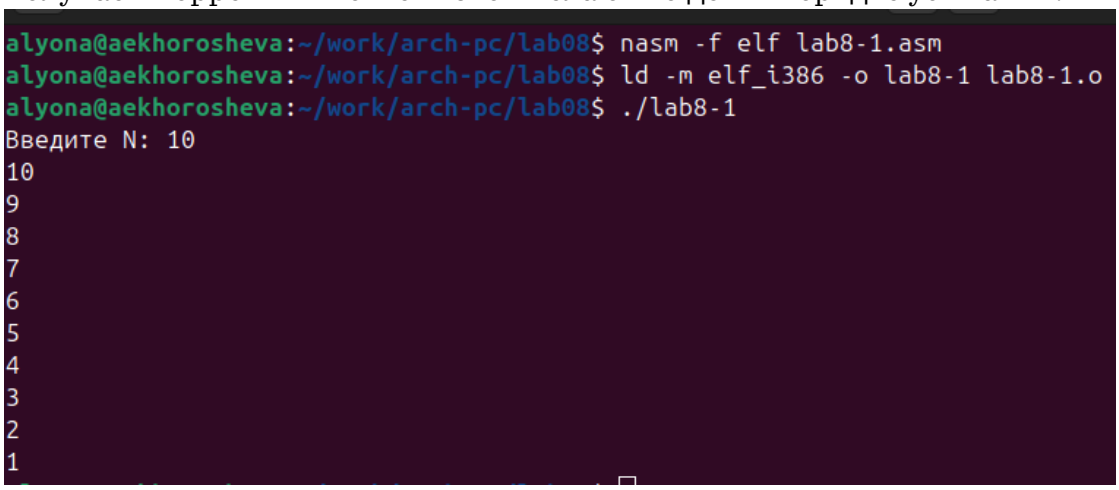
   ...

   loop NextStep ; Повторить ecx раз от метки NextStep

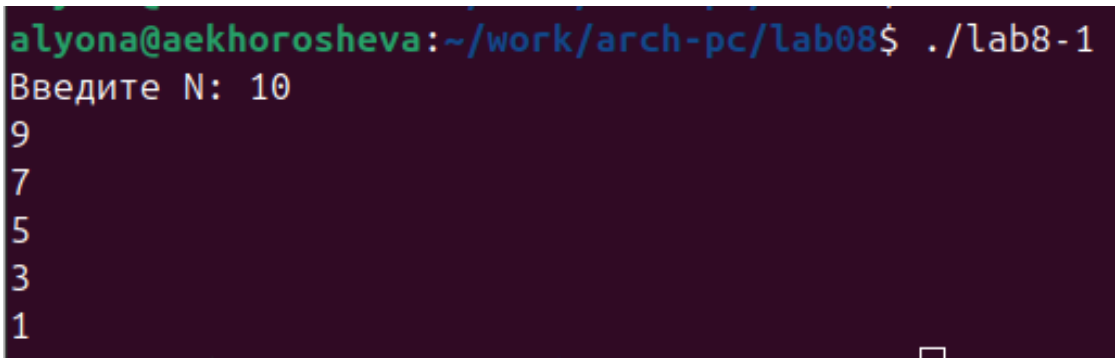
   Иструкция loop выполняется в два этапа. Сначала из регистра ecx вычитается единица и его значение сравнивается с нулём. Если регистр не равен нулю, то выполняется переход к указанной метке. Иначе переход не выполняется и управление передаётся команде, которая следует сразу после команды loop.

4 Выполнение лабораторной работы

1. В каталоге lab08 создаём файл lab8-1.asm и вводим туда текст программы из листинга 8.1. Программа должна выводить значения регистра ecx, который меняется в течение цикла. Цикл повторяется n раз, при этом n - вводится с клавиатуры. Запускаем исполняемый файл, вводим n = 10 и получаем корректный ответ - это числа от 10 до 1 в порядке убывания.



2. Теперь внесем изменения изменения регистра ecx внутри цикла. Запускаем исполняемый файл и видим, что программа работает некорректно. Число проходов теперь не соответствует значению N. Это произошло, потому что вычитание единицы в регистре ecx происходит перед тем, как значение данного регистра передается в eax для вывода на экран. Поэтому результат выводится с 9 до 1, при этом уменьшаясь сразу на 2.



3. Внесем еще одно изменение в текст программы. Теперь используем стек для добавления туда значение регистра ecx и последующего извлечения из стека, начиная с последнего добавленного в него элемента.

   label:

   push ecx ; добавление значения ecx в стек

   sub ecx,1

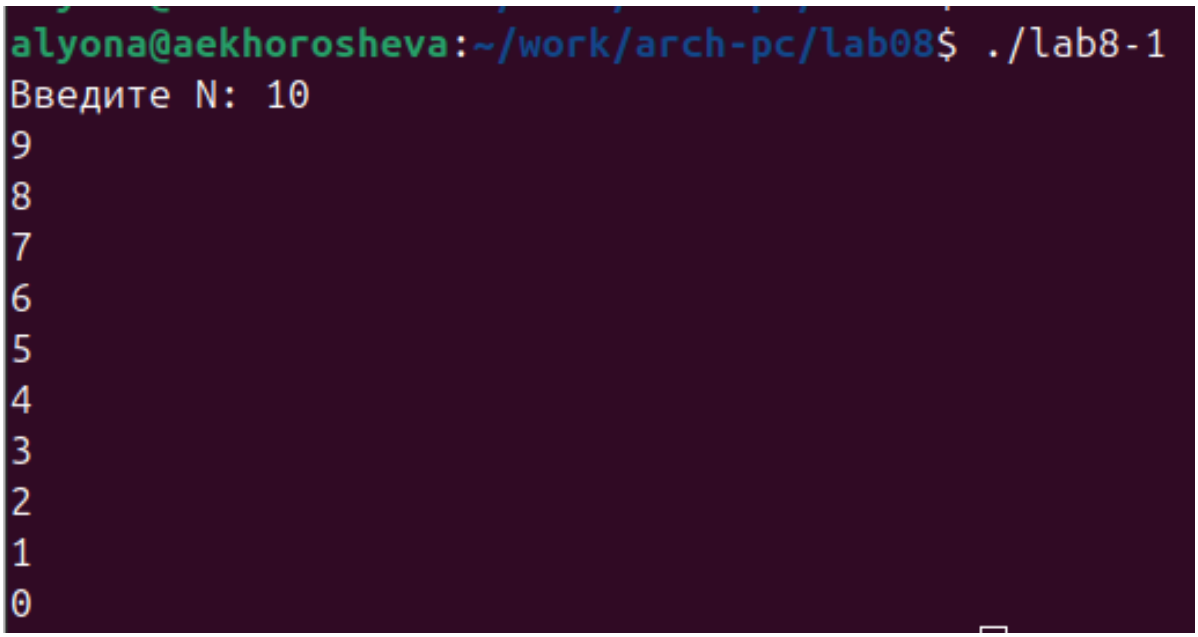
   mov [N],ecx

   mov eax,[N]

   call iprintLF

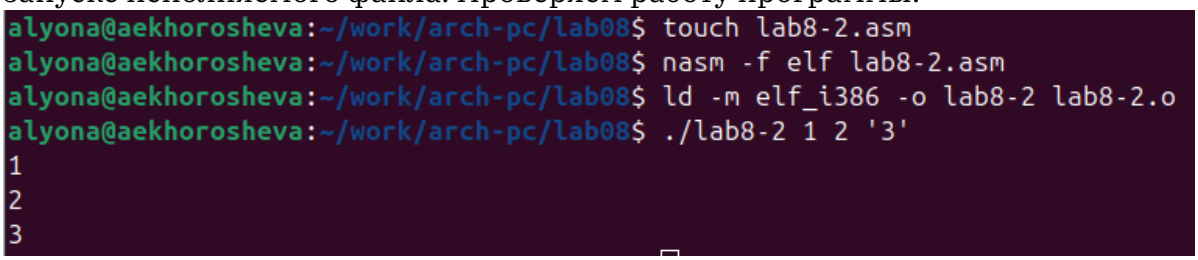
   pop ecx ; извлечение значения ecx из стека

   loop label



В результате имеем, что число проходов соответствует N, но вывод начинается с 9 и заканчивается на 0 - т.е. всего 10 проходов, что соответствует введенному значению N.

4. Создаем файл lab8-2.asm и пишет туда программу из листинга 8.2. Данная программа должна выводить на экран аргументы, которые мы введем при запуске исполняемого файла. Проверяем работу программы:



Было указано три аргумента и все они обработались программой и вывелись на экран.

5. Создаём файл lab8-3.asm для программы из листинга 8.3. для вычисления суммы аргументов командной строки. Вводим следующие значения аргументов: 12 13 7 10 5 из данного для выполнения лабораторной примера. Запуск исполняемого файла должен вывести результат 47. Проверяем работу программы:

```
alyona@aekhorosheva:~/work/arch-pc/lab08$ touch lab8-3.asm
alyona@aekhorosheva:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
alyona@aekhorosheva:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
alyona@aekhorosheva:~/work/arch-pc/lab08$ ./lab8-3 12 13 7 10 5
Результат: 47
```

6. Самостоятельно изменяем текст программы так, чтобы она вычисляла произведение аргументов вместо их суммы.

Во-первых, меняем значение esi с 0 на 1, иначе произведение будет обнуляться при любых введенных аргументах.

```
mov esi, 1 ;
```

Во-вторых, меняем строки внутри цикла - после преобразования аргумента из символа в число выполняем умножение командой mul и затем помещаем в esi значение eax, куда сохранился результат умножения.

```
pop eax ; иначе извлекаем с
call atoi ; преобразуем сим
mul esi
mov esi,eax
```

Теперь запускаем новую программу со значениями 2 2 2 2 2. Результат должен быть: 32. Проверяем:

```
alyona@aekhorosheva:~/work/arch-pc/lab08$ nasm -f elf lab8-3.asm
alyona@aekhorosheva:~/work/arch-pc/lab08$ ld -m elf_i386 -o lab8-3 lab8-3.o
alyona@aekhorosheva:~/work/arch-pc/lab08$ ./lab8-3 2 2 2 2 2
Результат: 32
```

5 Задание для самостоятельной работы

1. Необходимо написать программу, вычисляющую сумму функций для различных значений x. Мой вариант №1 - это функция 2x +15.
   Начинаем с подключением внешнего файла и выделения места в памяти для строк с выводом результата и заданной функции:
   %include ‘in_out.asm’
   SECTION .data
   msg1 db “Результат:”,0
   msg2 db “Функция: 2x+15”,0

2. Основная часть программы начинается с работы со стеком. Используя инструкцию pop извлекаем значения ecx и edx, которые не участвуют в дальнейшем вычислении. Используем esi для хранения суммы - задаем начальное значение 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
```

3. Переходим к написанию цикла “next:”. Сначала сравниваем с 0 текущее значение ecx(кол-во аргументов). Если ecx = 0, то сразу перейдем к концу программы для вывода результатов. Иначе остаёмся в цикле: извлекаем из стека значение аргумента, преобразуем в число. В регистр ebx запишем число 2 и выполним команду mul ebx - таким образом, имеем значение 2x в регистре eax. Теперь прибавляем это значение к esi, чтобы можно было и дальше вычислять сумму с другими аргументами. Также прибавим 15 и после этого можно перейти к повторению цикла уже со следующим числом стека.

```
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
mov ebx, 2
mul ebx
add esi, eax
add esi, 15 ; добавляем к промежуточной сумме
loop next ; переход к обработке следующего аргумента
```

4. Последняя часть - завершение программы. Выводим заданную функцию на экран: 2x + 15. Для перехода на следующую строку для вывода результата используем функцию sprintLF вместо sprint. В регистр eax записываем

значение итоговой суммы из esi.

_end:

mov eax, msg2 ; вывод сообщения “Функция: 2x+15”

call sprintLF

mov eax, msg1; вывод сообщения “Результат:”

call sprint

mov eax, esi ; записываем сумму в регистр eax

call iprintLF ; печать результата

call quit ; завершение программы

5. Запускаем исполняемый файл. Для значений 1 2 3 должны получить 57. Проверяем:

```
alyona@aekhorosheva:~/work/arch-pc/lab08$ ./lab8-4 1 2 3
Функция: 2x+15
Результат: 57
```

6. Для того, чтобы убедиться в корректности выполнения программы, запускаем её еще 2 раза с различными значениями аргументов и отличных друг от друга по количеству. Получаем верные результаты вычислений:

```
alyona@aekhorosheva:~/work/arch-pc/lab08$ ./lab8-4 2 3 4 5
Функция: 2x+15
Результат: 88
alyona@aekhorosheva:~/work/arch-pc/lab08$ ./lab8-4 4 1 5 2 4
Функция: 2x+15
Результат: 107
```

6 Выводы

В результате выполнения лабораторной работы были получены теоретические знания о такой структуре данных, как *стек*.
Освоены на практике команды для работы со стеком: push и pop - для добавления и извлечения соответственно. Также получены навыки написания программ с циклами и обработкой аргументов, введенных с клавиатуры.
Самостоятельно была написана программа для нахождения значений функции, зависящей от x, с различными аргументами.

Список литературы