**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**по лабораторной работе № 8**

*дисциплина: Архитектура компьютера*

Студент: Ниемек Яи Жак

Группа: НММБд-04-24

**МОСКВА**

2025 г.

Цель работы

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

Задание

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, …, ��.

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. 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

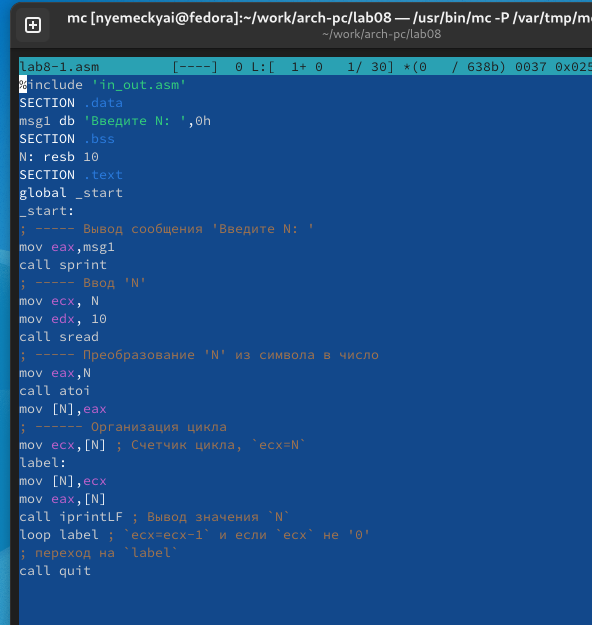
назначения в следующем порядке: ах, сх, dx, bх, sp, bp, si, di. А также команда

pushf, которая служит для перемещения в стек содержимого регистра флагов. Обе эти команды не имеют операндов. 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. 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.

Выполнение лабораторной работы

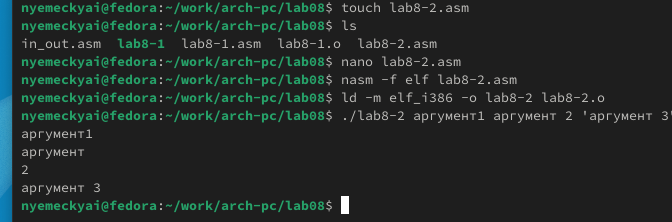
Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



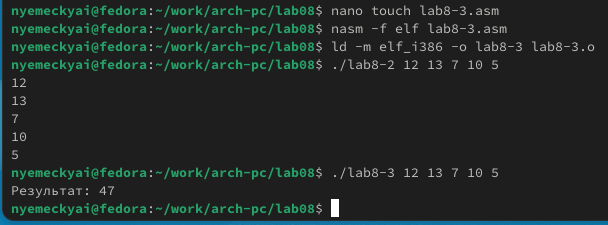
Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.



Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Контент, сгенерированный ИИ, может содержать ошибки.



Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.



Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Выводы

Я приобрел навыки написания программ с использованием циклов и обработкой аргументов командной строк

**1. Опишите работу команды loop.**

Команда loop в ассемблере используется для организации цикла с учётом декремента счётчика. Она выполняет следующее:

* Понижает значение регистра **ECX** (счётчик).
* Если ECX не равен нулю, то выполняется переход на метку, указанную в команде loop. Если значение ECX равно нулю, выполнение программы продолжается с инструкции, следующей за командой loop.

Пример:

mov ecx, 5 ; Инициализируем счётчик цикла (5 итераций)

loop\_start:

; Тело цикла

dec ecx ; Уменьшаем ECX на 1

loop loop\_start ; Если ECX != 0, повторить цикл

В данном примере цикл будет повторяться 5 раз. Команда loop работает, пока значение регистра ECX не станет равным нулю.

**2. Как организовать цикл с помощью команд условных переходов, не прибегая к специальным командам управления циклами?**

Цикл можно организовать, используя команды условных переходов, такие как cmp, jne (не равно), je (равно) и другие, в сочетании с командой для уменьшения счётчика (например, dec или sub). Пример:

mov ecx, 5 ; Инициализация счётчика цикла

loop\_start:

; Тело цикла

dec ecx ; Уменьшаем счётчик

cmp ecx, 0 ; Сравниваем с нулём

jne loop\_start ; Если ECX != 0, повторить цикл

В данном примере мы явно контролируем цикл через условный переход jne, который продолжает цикл, пока значение ECX не станет равным нулю.

**3. Дайте определение понятия «стек».**

Стек — это структура данных, которая работает по принципу **LIFO** (Last In, First Out), то есть последний элемент, который был добавлен в стек, первым извлекается. Стек используется для временного хранения данных, таких как адреса возврата из подпрограмм, локальные переменные и промежуточные значения.

Основные операции стека:

* **Push**: добавление элемента в стек.
* **Pop**: извлечение элемента из стека.
* **Peek**: просмотр верхнего элемента без его извлечения.

Стек используется в большинстве программных систем для управления вызовами функций и рекурсией.

**4. Как осуществляется порядок выборки содержащихся в стеке данных?**

Порядок выборки данных из стека осуществляется по принципу **LIFO** (Last In, First Out):

* **Последний добавленный элемент** извлекается первым.
* Когда команда push добавляет элемент в стек, он помещается на верх стека.
* Когда команда pop извлекает элемент, она удаляет верхний элемент из стека и возвращает его.

Например:

push eax ; Помещает значение регистра EAX в стек

pop ebx ; Извлекает верхний элемент стека в регистр EBX

Таким образом, порядок извлечения данных строго обратен порядку их добавления, что и делает стек полезным для сохранения состояния программ (например, при вызове подпрограмм).