|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт Информационных Технологий

Кафедра Вычислительной Техники (ВТ)

**Отчёт**

по дисциплине

«Разработка программно-аппаратного обеспечения информационных и автоматизированных систем»

|  |  |
| --- | --- |
|  | |
| Выполнили студенты группы ИВМО-02-24  Принял преподаватель | Кутепов А.О.  Рьянов А.Е.    Скрябин Ю.М. |
|  |  |

Работа выполнена «\_\_\_»\_\_\_\_\_\_\_\_20\_\_\_г

«Зачтено»

«\_\_\_»\_\_\_\_\_\_\_\_20\_\_\_г

Москва 2024

Оглавление

[Пользовательский интерфейс 3](#_Toc181275909)

[Порядок работы с приложением 5](#_Toc181275910)

[Обработка исключений 6](#_Toc181275911)

[Код ассемблера 7](#_Toc181275912)

[Задание №1. Поиск максимума в массиве (Архитектура Гарвардская, двухадресная, без знака) 7](#_Toc181275913)

[Задание №2. Свертка двух массивов (Архитектура Гарвардская, двухадресная, без знака) 8](#_Toc181275914)

[Используемые команды ассемблера и их машинное представление 10](#_Toc181275915)

# Пользовательский интерфейс

Разработано web приложение с интуитивно-понятным интерфейсом на FastAPI (Python, Jinja2, HTML, CSS, JavaScript) для эмулирования работы программы на ассемблере для:

1. нахождения максимума в массиве
2. нахождение свёртки массива

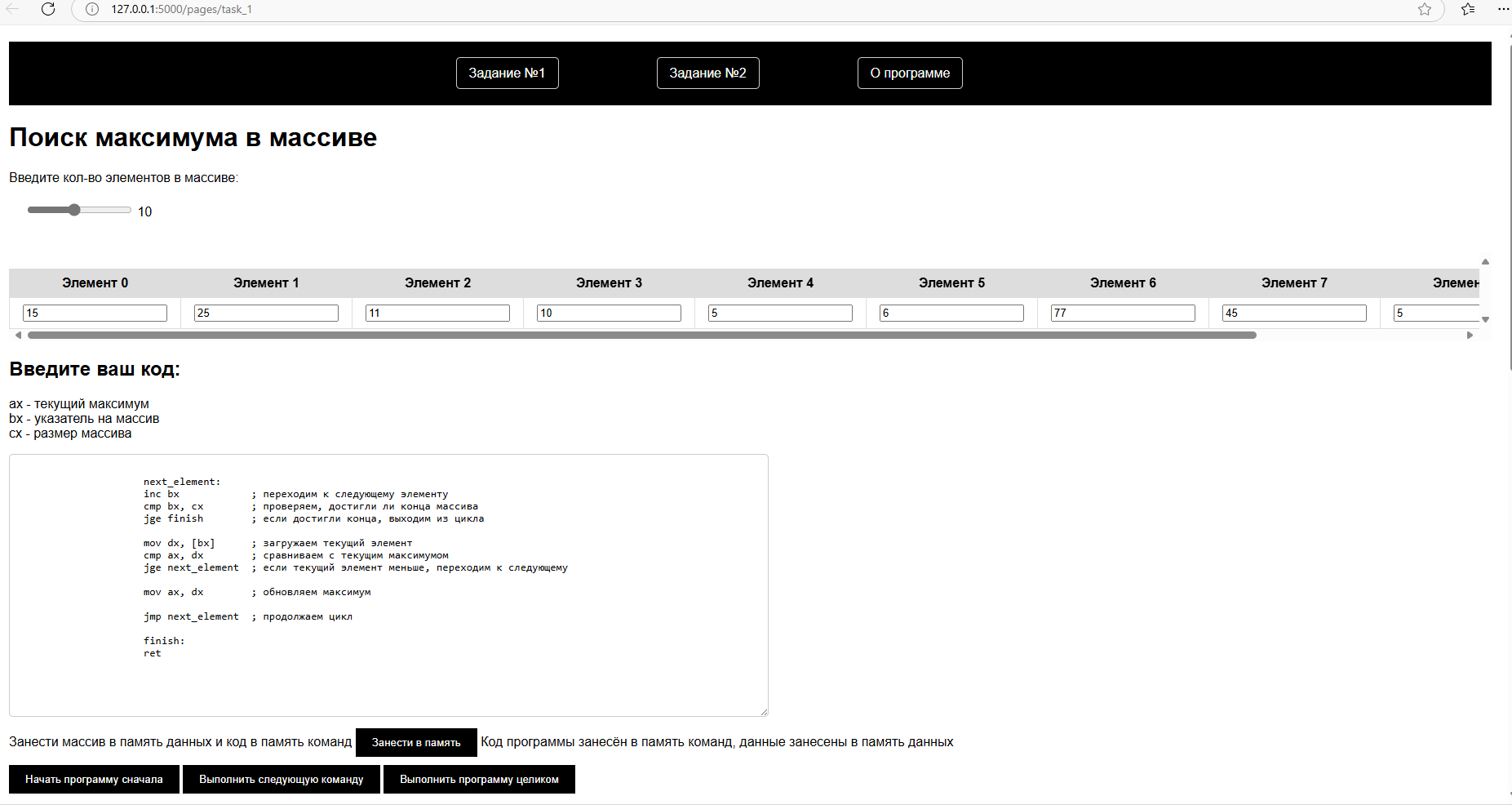


Рис 1.

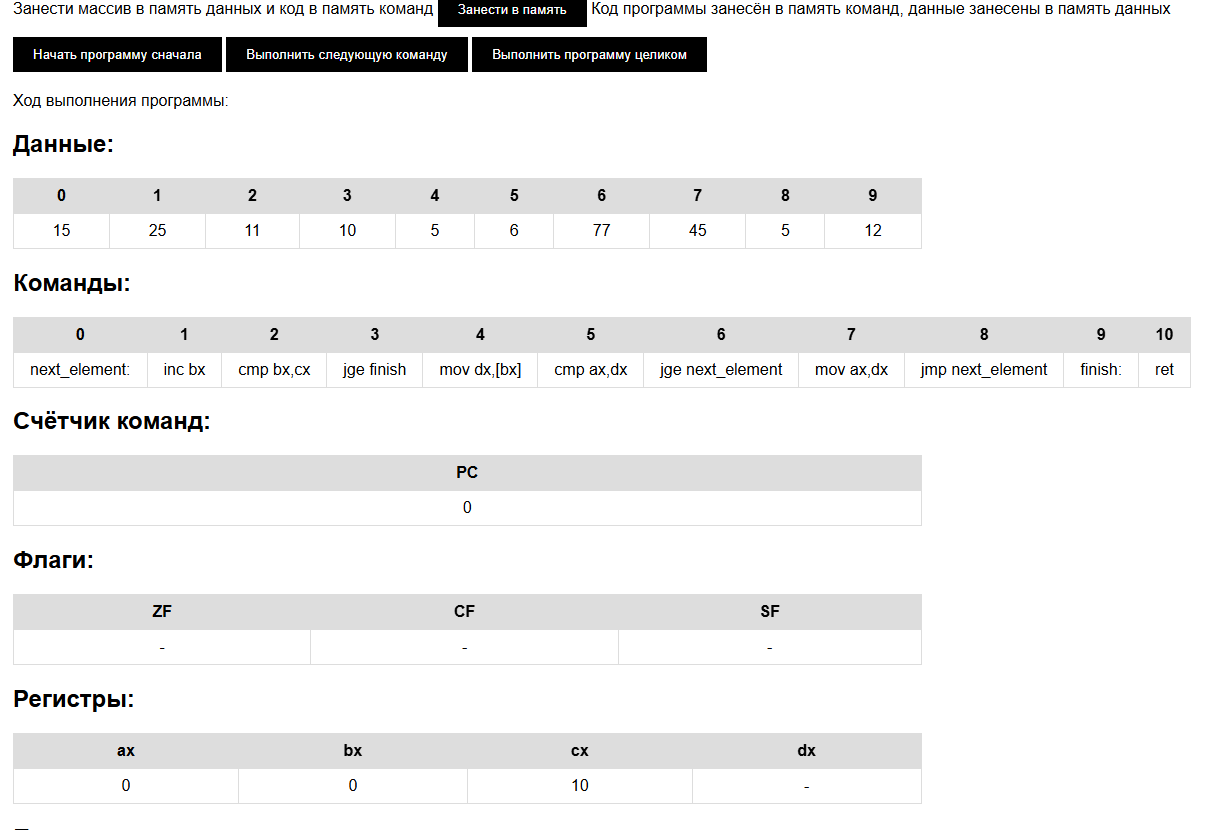


Рис 2.

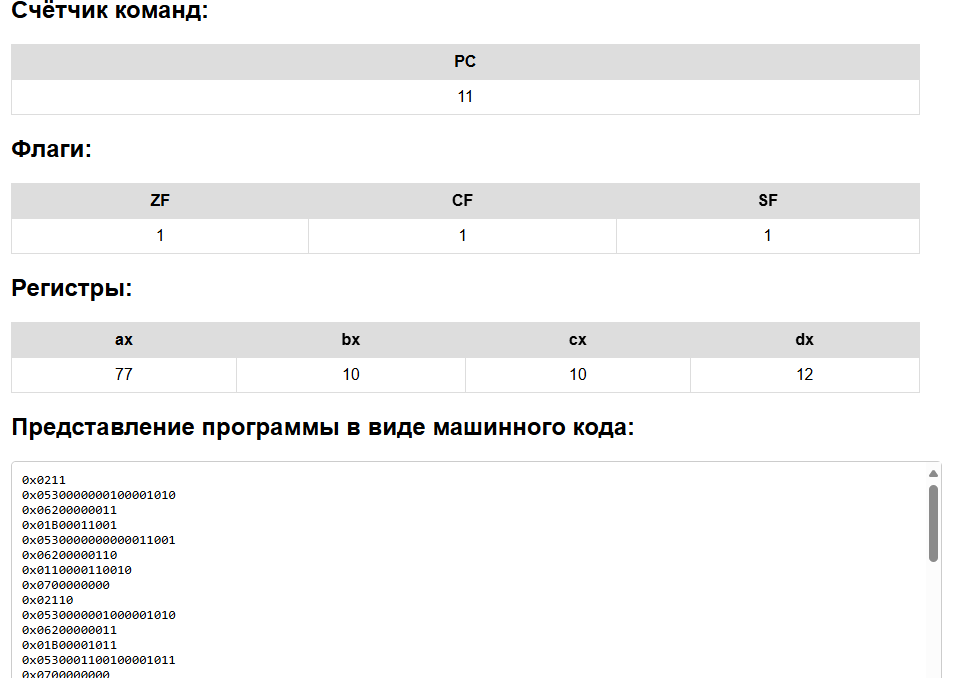


Рис 3.

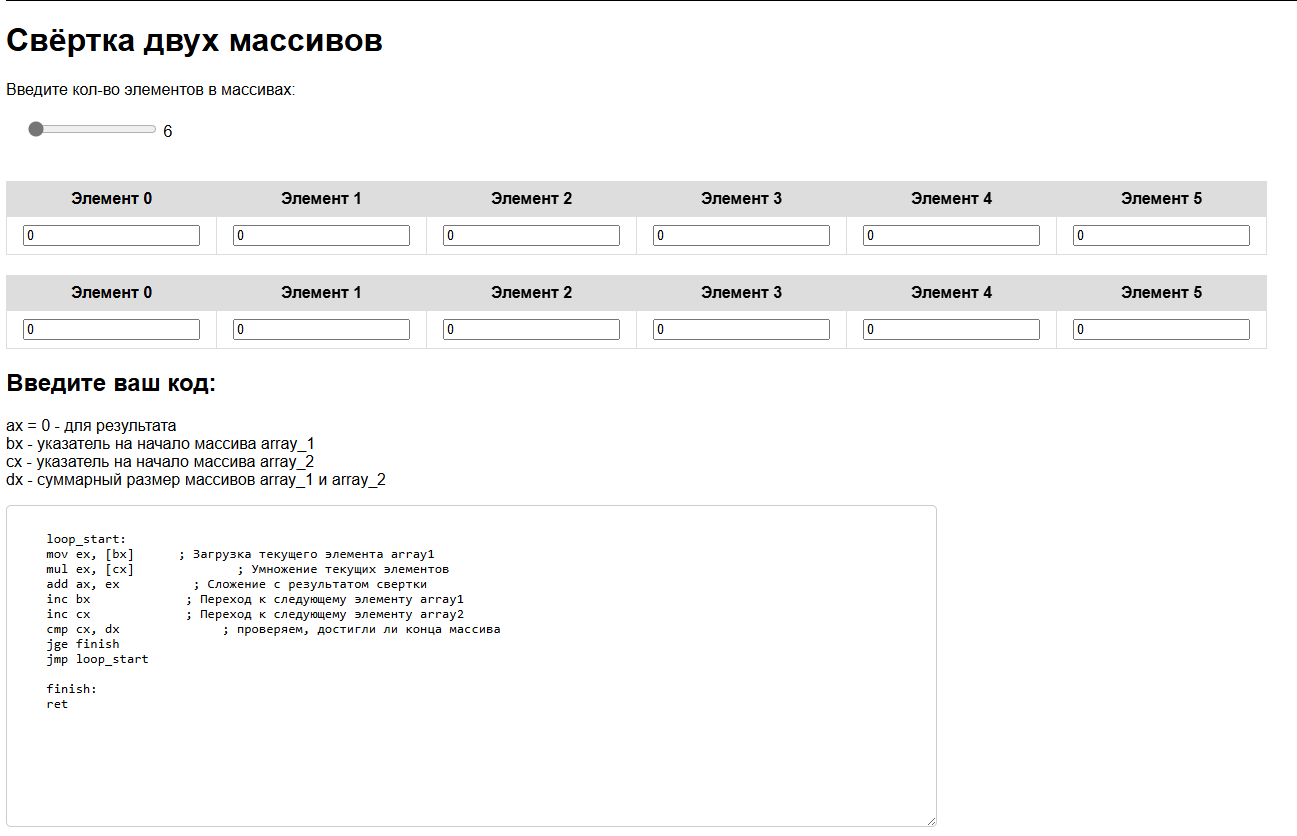


Рис 4.

# Порядок работы с приложением

1. Пользователь задаёт кол-во элементов в массиве
2. Вводит сами эл-ы в массив
3. Вводит/изменяет код (по умолчанию уже введён)
4. Заносит данные и программу в память данных и память нажав на кнопку «Занести в память»
5. Можно выполнить программу сразу – «Выполнить программу целиком», либо пошагово нажимая кнопку – «Выполнить следующую команду»
6. Для начала работы сначала, обнуления регистров, флагов – «Начать программу сначала»

Отображаются введённые данные, команды с операндами, счётчик команд, флаги, регистры.

# Обработка исключений

1. Несуществующая метка

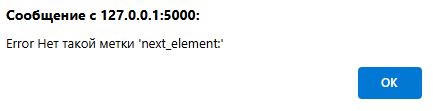
****

Рис 5.

1. Несуществующая команда

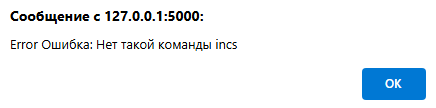


Рис 6.

1. Неверный тип операнда

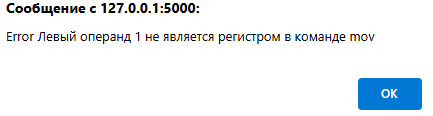


Рис 7.

4)Несоответствие кол-ву операндов

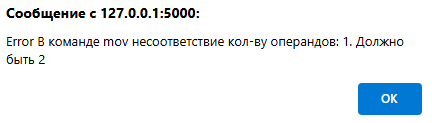


Рис 8.

# Код ассемблера

## Задание №1. Поиск максимума в массиве (Архитектура Гарвардская, двухадресная, без знака)

**Описание задачи:**

Написать программу на ассемблере, которая находит максимальный элемент в массиве целых чисел без знака, используя архитектуру Гарвардская и двухадресные команды.

**Входные данные**

Массив из 6-15 элементов, хранящийся в памяти данных.

**Выходные данные**

Максимальный элемент массива, хранящийся в регистре.

**Регистры**

• AX: Хранит максимальный элемент массива.  
• BX: Указатель на начало массива (хранится в памяти данных).  
• CX: Счетчик элементов массива.  
• DX: Используется для хранения текущего элемента массива

**Код:**

section .data

array db 3, 5, 1, 7, 2, 8, 4, 6, 9, 0 ; массив из 10 элементов

length db 10 ; длина массива

\_start:

; Инициализация указателей и переменных

mov cx, [length] ; длина массива

mov bx, [array] ; указатель массива

mov ax, [bx] ; первый элемент массива как текущий максимум

next\_element:

inc bx ; переходим к следующему элементу

cmp bx, cx ; проверяем, достигли ли конца массива

jge finish ; если достигли конца, выходим из цикла

mov dx, [bx] ; загружаем текущий элемент

cmp ax, dx ; сравниваем с текущим максимумом

jge next\_element ; если текущий элемент меньше, переходим к следующему

mov ax, dx ; обновляем максимум

jmp next\_element ; продолжаем цикл

finish:

ret

**Объяснение**

1. Инициализация: Программа загружает сегмент данных, инициализирует CX с размером массива, BX с адресом начала массива (из array\_ptr) и DX первым элементом массива.
2. Цикл: В цикле loop\_start текущий элемент загружается в AX и сравнивается с DX. Если текущий элемент больше, то DX обновляется.
3. Переход к следующему элементу: BX увеличивается на 2 для перехода к следующему элементу массива (размер слова 2 байта).
4. Счетчик: Счетчик CX уменьшается на 1.
5. Проверка счетчика: Если CX не равен нулю, цикл продолжается.
6. Вывод результата: Максимальный элемент находится в регистре AX. Добавьте код для вывода результата по своему усмотрению.

**Заметки:**

• Гарвардская архитектура: Команды и данные хранятся в разных адресных пространствах. В этом примере, array\_ptr хранит адрес начала массива в памяти данных, доступ к которому осуществляется через ds:bx.  
• Двухадресная команда add: В программе используется add bx, 2 для перехода к следующему элементу массива.  
• Сравнение без знака: Вместо jg используется ja (jump above), чтобы сравнить числа без учета знака.

## Задание №2. Свертка двух массивов (Архитектура Гарвардская, двухадресная, без знака)

**Описание задачи**

Написать программу на ассемблере, которая выполняет свертку двух массивов с 6 элементами, содержащих целые числа без знака, используя архитектуру Гарвардская и двухадресные команды.

**Входные данные**

Два массива (array1 и array2), хранящихся в памяти данных.

**Выходные данные Результат свертки, хранящийся в регистре AX.**

**Регистры**

• AX: Используется для хранения результата свертки.  
• BX: Указатель на первый элемент массива array1 (хранится в памяти данных).  
• CX: Указатель на первый элемент массива array2 (хранится в памяти данных).  
• DX: Счетчик элементов массива.

**Код**

.data

array\_1 dw 5, 2, 7, 1, 4, 3 ; Первый массив (без знака)

array\_2 dw 1, 3, 2, 5, 0, 2 ; Второй массив (без знака)

array\_size dw 12 ; Размер массива

mov bx, [array\_1] ; Загрузка указателя на начало массива array1

mov cx, [array\_2] ; Загрузка указателя на начало массива array2

mov dx, array\_size ; Загрузка размера массива

mov ax, 0 ; Инициализация результата свертки

loop\_start:

mov ex, [bx] ; Загрузка текущего элемента array1

mul ex, [cx] ; Умножение текущих элементов

add ax, ex ; Сложение с результатом свертки

inc bx ; Переход к следующему элементу array1

inc cx ; Переход к следующему элементу array2

cmp cx, dx ; проверяем, достигли ли конца массива

jge finish

jmp loop\_start

finish:

ret

**Объяснение:**

1. Инициализация: Программа загружает сегмент данных, инициализирует BX, CX, DX и AX начальными значениями.
2. Цикл: В цикле loop\_start текущие элементы массивов array1 и array2 умножаются, результат складывается с результатом свертки (AX).
3. Переход к следующему элементу: BX и CX увеличиваются на 2 для перехода к следующему элементу.
4. Счетчик: Счетчик DX уменьшается на 1.
5. Проверка счетчика: Если DX не равен нулю, цикл продолжается.
6. Вывод результата: Результат свертки хранится в регистре AX. Добавьте код для вывода результата по своему усмотрению.

**Заметки:**

• Гарвардская архитектура: Команды и данные хранятся в разных адресных пространствах. В этом примере, array1\_ptr и array2\_ptr хранят адреса начала массивов в памяти данных, доступ к которым осуществляется через ds:bx и ds:cx.  
• Двухадресная команда add: В программе используются add bx, 2 и add cx, 2 для перехода к следующему элементу массива.  
• Числа без знака: Программа использует числа без знака, поэтому нет необходимости в команде mul (умножение без знака).

# Используемые команды ассемблера и их машинное представление

**1.Команда MOV** в ассемблере перемещает значение из источника в приёмник. Она копирует содержимое источника  
 и помещает его в приёмник, не изменяя при этом никакие флаги.  
 Источником и приёмником могут быть регистры общего назначения, сегментные регистры или области памяти. **Код**: 0x01

**2.Инструкция CMP** (от слова compare - сравнить) позволяет сравнить значения и установить флаги.Результат получается после вычитания первого значения из второго: если разность больше нуля —первое больше второго, разность меньше нуля — второе больше первого, нуль — равны. Если при вычитании чисел получился нуль, то CMP записывает 1 во флаг ZF, если нет, то записывает во флаг CF 0 если число положительное и 1 если отрицательное. Флаг знака SF устанавливается, если результат отрицательный. **Код**: 0x02

**3.Команда JA** в ассемблере выполняет условный переход.  
Если флаг переноса (CF) равен нулю и флаг нуля (ZF) равен нулю, то выполняется переход к указанному адресу. Результат операции не сохраняется. **Код**: 0x03

**4.Команда ADD** в ассемблере выполняет сложение двух операндов и сохраняет результат в первом операнде (приёмнике). **Код**: 0x04

**5.Инструкция DEC** в ассемблере уменьшает число на единицу.

**Код**: 0x05

**6.Инструкция INC** в ассемблере увеличивает число на единицу. **Код**: 0x06

**7.Команда MUL** в ассемблере выполняет умножение двух операндов и сохраняет результат в первом операнде (приёмнике). . **Код**: 0x07

**8.Инструкция JUMP** гаходит команду в PROGRAM\_COMMANDS и устанавливает счётчик команд(PC) на неё. **Код**: 0x08

**9.Команда RET** завершает работу всей програмы. **Код**: 0x09

**Машинное представление операнда в зависимости от его типа:**

Прямая адресация – 00b

Непосредственный – 01b

Регистровый – 10b

Относительная адресация – 10b

Косвенно-регистровая адресация– 11b