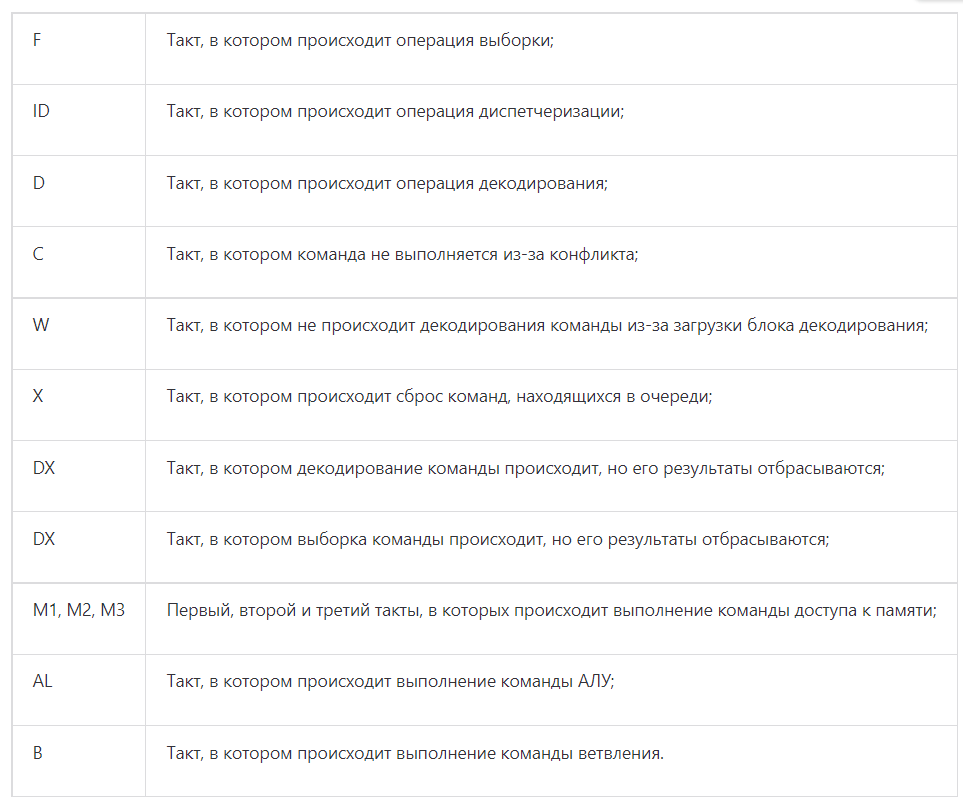
**Лабораторная 1**

****

**Лабораторная 2**

**Задание 1**

Суть: обращение к динамической памяти по последовательным адресам с изменением шага чтения в банке. Так как оперативная память неоднородна, то время доступа отличается.

По у- время доступа к памяти в банке, по х – расстояние между ячейками

Т1 (первый экстремум) минимальный шаг чтения (наилучшее время, т.е. ячейки близко расположены)

Т2 (первый наибольший экстремум) расстоянию (в байтах) между началом 9 двух последовательных страниц одного банка (наихудшее время, т.е. ячейки далеко друг от друга)

Вывод: Переход к близким данным переходится быстрее. Данные лучше сохранить в один пакет. Надо чтобы было чередование.

**Задание 2**

Суть: сравнение эффективности структур данных. (время доступа к элементу структур)

По у – время доступа к элементу, по х – фрагментация структуры (т.е. среднее расстояние между элементами)

Красный график – списки с указателями, Зеленый – массив

Вывод: Массив — примитивная векторная структура, но их можно везде применить. Но иногда нужны списки. Надо улучшать ЭВМ, чтобы решить проблему семантического разрыва.

**Задание 3**

Суть: замер времени двух вариантов подпрограмм последовательного чтения страниц оперативной памяти. В первом варианте выполняется последовательное чтение без дополнительной оптимизации, что приводит к дополнительным двойным обращениям(красный график). Во втором варианте перед циклом чтения страниц используется дополнительный цикл предвыборки, обеспечивающий своевременную загрузку информации в TLB данных (Кэширование страницы, зеленый график).

По у – время доступа к элементу, по х – смещение читаемых данных.

Вывод: можно использовать предварительную загрузку в TLB (если не будет прерывания) в какое ограниченное время. Что сделает обращение к памяти в 1,3 раза быстрее.

**Задание 4**

Суть: При обработке информации, находящейся в нескольких страницах и банках оперативной памяти возникают задержки, связанные с необходимостью открытия и закрытия страниц DRAM памяти. Соответстенно для оптимизации работы необходимо исключить подгрузку ненужных данных в кэш. Неоптимизированный вариант структуры данных представляет собой несколько массивов в оперативной памяти (красный), в то время как оптимизированная структура состоит из чередующихся данных каждого массива(зеленый). Желательно чтобы все необходимое для работы с текущей командой находилось в одном пакете.

По у - время доступа к элементу, по х - количество одновременно обрабатываемых массивов.

Вывод: при оптимизации необходимо, чтобы данные были расположены в одном пакете, что даст быстродействие. Если это возможно реализовать в программировании, то это дат ускорение.

**Задание 5**

Суть: Наборно-ассоциативная кэш-память состоит из линеек данных, организованных в несколько независимых банков. Совокупность таких линеек всех банков принято называть набором. Оптимизировать доступ с помощью кэширования набора линеек.

Первая процедура построена таким образом, что чтение данных выполняется с шагом, кратным размеру банка. Это порождает постоянные конфликты в кэш-памяти. (красный) Вторая процедура оптимизирует размещение данных в кэш с помощью задания смещения востребованных данных на некоторый шаг, достаточный для выбора другого набора. Этот шаг соответствует размеру линейки.(зеленый)

По у – время доступа, по х- смещение читаемой ячейки от начала блока данных.

Вывод: нужно “ходить” по памяти, не равный расстоянию конфликта. С кэшем в 6 раз быстрее. Надо уменьшать конфликты в кэш памяти.

**Задание 6**

Фиолетовый график (верхний) показывает время или количество тактов работы алгоритма QuickSort (сортировка диапазонами). Красный график (средний) показывает время или количество тактов работы неоптимизированного алгоритма Radix-Counting (сортировка поразрядная). Зеленый график (нижний) показывает время или количество тактов работы оптимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting. Ось абсцисс отражает количество 64- разрядных элементов сортируемых массивов. Ось ординат отображает время в микросекундах или количество тактов

Вывод: несмотря на формульную быстроту QuickSort – в среденем O(n\*log(n)) и O(n2), у метода Radix-Counting формульная быстрота работы будет О(n/log(n)), что говорит о его быстродействии.