Лабораторная работа № 2

ИСПОЛЬЗОВАНИЕ ЦИКЛОВ

1. **ЦЕЛЬ РАБОТЫ:** освоение принципов построения приложений на языке ассемблера для системы Texas Instruments, ознакомление с командами и правилами построения программ в соответствии с особенностями организации циклов.

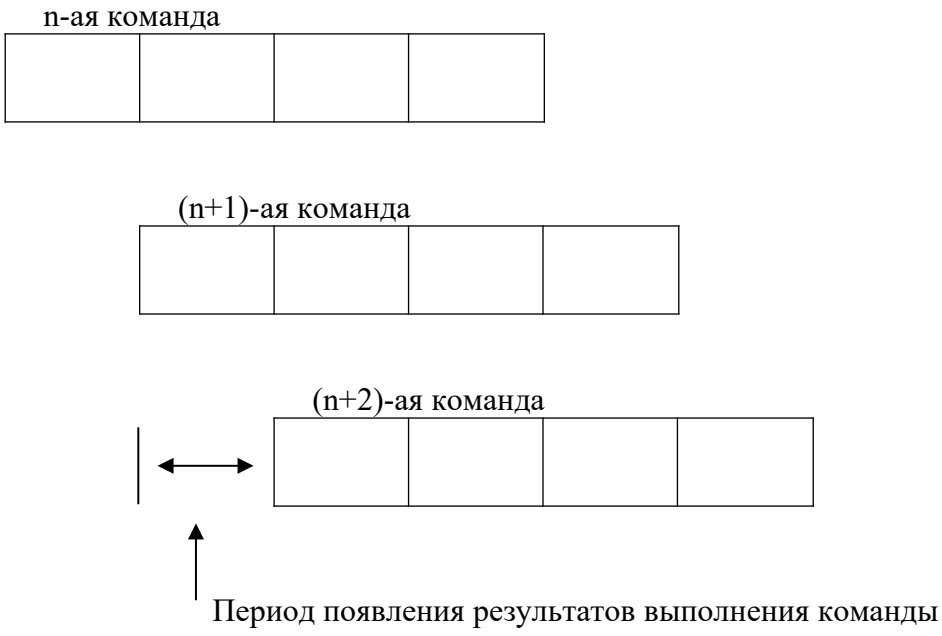
2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Процесс выполнения команды во всех процессорах разбивается на несколько этапов. Конвейерный принцип выполнения команд состоит в том, что различные этапы разных команд выполняются одновременно.

Время выполнения одного этапа команды называют командным циклом работы процессора (иногда тактом работы процессора). Время командного цикла, как правило равно периоду внутренней тактовой частоты работы процессора.

Схема выполнения команды во времени и обозначение этапов при разделении ее на четыре этапа.

Конвейерный способ выполнения команд

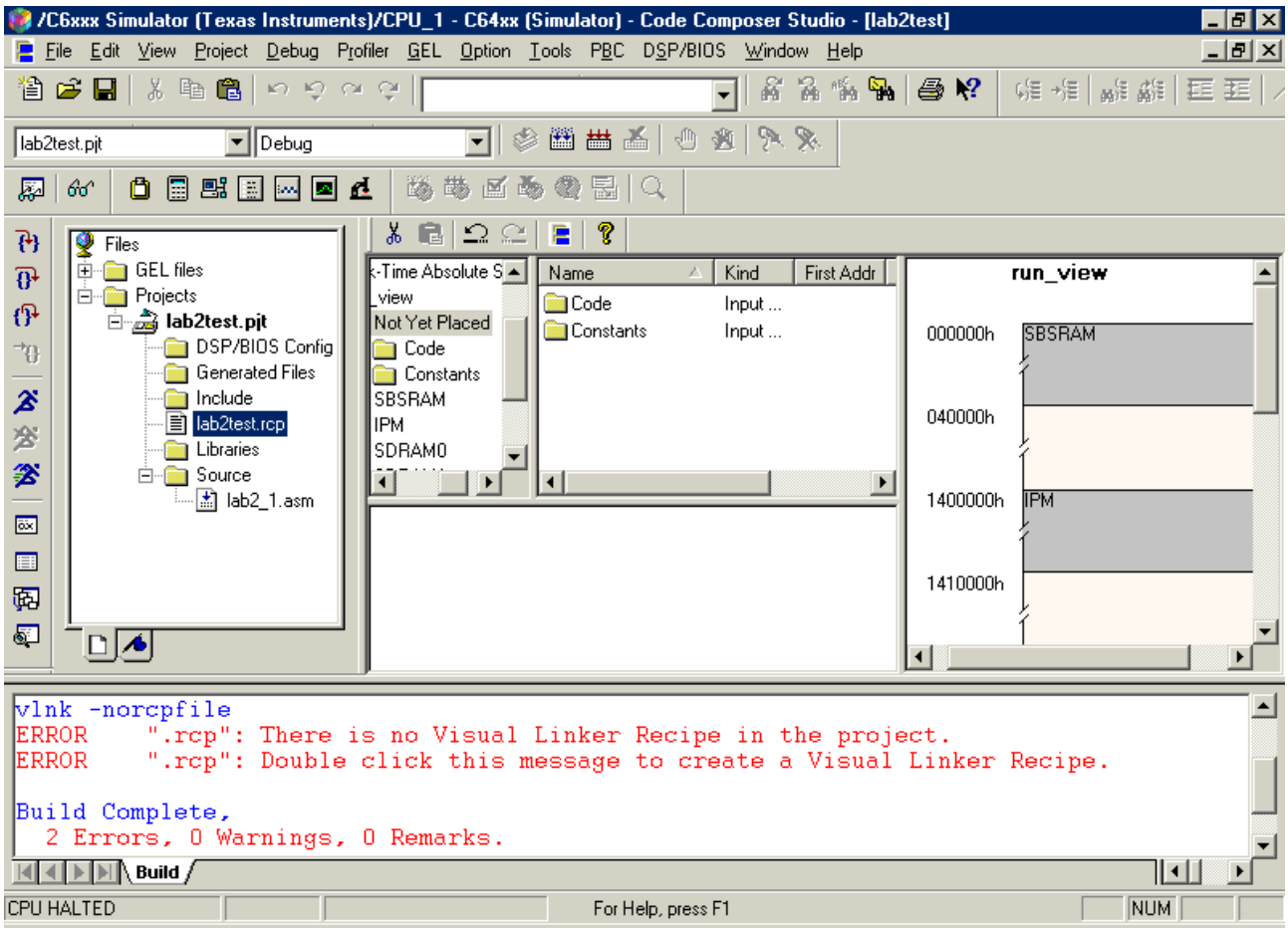


Конвейерное выполнение команд возможно при условии, что в процессоре имеются несколько функциональных узлов для выполнения одновременно различных этапов с разными данными, относящимися к разным командам.

3. ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ:

Для запуска тестового примера необходимо оформить его в виде файла с расширением *.asm, скопировать в папку созданного проекта.
После проведения всех необходимых установок выполнить команду Project\Rebuild All.
В результате в окне сообщений о результатах компиляции появится предложение двойным щелчком мыши создать проект Visual Linker.

Затем на все предложенные запросы следует нажимать клавишу Далее, до тех пор, пока не будет предложено указать memory file. Для этого следует нажать клавишу Browse и путем перемещения по папкам достигнуть следующего пути:
C6x\c6201\c6xEVM\map0\c6201EVM_map0.mem, затем щелкнуть открыть и продолжать нажимать далее пока ввод запросов не закончится. После проведенных действий в списке файлов проекта должен появиться файл с расширением *.rcp. После щелчка мыши по этому файлу запуститься мастер распределения памяти для программы.
В программе присутствует раздел кода .text и раздел данных .data.



Затем следует выбрать папку Code, раскрыть ее и перетащить содержащийся в ней файл на первую представленную в правой части окна область памяти. Затем следует выбрать

папку Constants, раскрыть ее и перетащить содержащийся в ней файл на первую представленную в правой части окна область памяти.
В результате область памяти будет заштрихована, данные распределены.

После проведенных действий следует повторить команду Project\Rebuild All, закрыть Visual Linker Recipe и продолжать работу с программой.

Задание: Дано два массива, необходимо поэлементно их перемножить и сложить полученные произведения.

Тип данных: 32 бита со знаком (signed int)

Пример:
A: {1,2,3}
B: {1,2,3}
1x1+2x2+3x3=1+4+9=14;

```
;поэлементное произведение элементов 2 массивов
;регистр A0 содержит конечную сумму произведений
;параллельные вычисления не используются

 .ref _c_int00     ;точка входа
_c_int00:

;////////////////////////////////////////
 .data   ;секция данных

array1:     .int 1,2,3    ;создаем массив 32 разрядных чисел
array2:     .int 1,2,3    ;создаем массив 32 разрядных чисел
size    .set 3            ;размер массива(>1)(препроцессорная константа)


;////////////////////////////////////////
 .text   ;секция кода



    ;Инициализация:
        MVKL .S1 array1,A3    ;загружаем адрес массива1 в A3
        MVKH .S1 array1,A3

        MVKL .S1 array2,A7    ;загружаем адрес массива2 в A7
        MVKH .S1 array2,A7

        MVK .S1 size,A2    ;загружаем колво элементов массива в A2
        MVK .S1 0,A4          ;ПРОИЗВЕДЕНИЕ ТЕКУЩИХ элементов массива
        MVK .S1 0,A0          ;сумма произведений
        MVK .S2 0,B2          ; тек. элемент выбираемый из массива 1
        MVK .S1 0,A5          ; тек. элемент выбираемый из массива 1
```

```
LOOP:
        SUB .L1 A2,1,A2      ;A2 := A2 - 1
        LDW   .D1  *A3[A2], B2     ;загружаем текущий элемент в B2
        NOP 4                      ;4х тактовая задержка загрузки
        LDW   .D1  *A7[A2], A5     ;загружаем текущий элемент в A2
        NOP 4                      ;4х тактовая задержка загрузки
        MPY .M1X A5,B2,A4          ;умножение элементов массивов
        NOP
        ADD .L1 A0, A4,A0          ;сумма результатов умножения
        [A2] B .S1 LOOP            ;переход если A2 <> 0
        NOP 5
```

Описание команд:

ЦПУ имеет восемь операционных модулей L, S, M, D разбитые на две идентичные группы 1 и 2. Источниками операндов и получателями результатов являются два набора 32- разрядных регистров A и B соответственно для операционных модулей группы 1 и 2.

L1, S1, M1, D1 – регистры общего назначения A0-A15
L2, S2, M2, D2 – регистры общего назначения B0-B15

Модули L (L1 , L2)
- 32/40 – разрядные арифметические операции и операции сравнения;
- 32–разрядные логические операции;
- операции нормализации

Модули S (S1 , S2)
- 32–разрядные арифметические операции;
- 32/40 – разрядные операции сдвига и операции с отдельными битами;
- 32–разрядные логические операции;

Модули M (M1 , M2)
- Операции умножения 32x32 с фиксированной точкой;
- Операции умножения 32x32 с плавающей точкой;

Модули D (D1 , D2)

32–разрядные операции по вычислению адресов, в том числе адресов циклических и линейных буферов.

Описание команд:

Записать 16 разрядную константу в регистр

MVK (.unit) cst, dst ; где cst – константа; dst – регистр

unit = .S1 или .S2

Пример:
```
    MVK .S1 0,A4            ;Запишем 0 в регистр A4;
```

Количество тактов : 1

Записать 16 разрядную константу в верхние биты регистра

MVKH (.unit) cst, dst ; где cst – константа; dst – регистр
unit = .S1 или .S2

Количество тактов : 1

Записать 16 разрядную константу в нижние биты регистра

MVKL (.unit) cst, dst ; где cst – константа; dst – регистр
unit = .S1 или .S2

Количество тактов : 1

Знаковое вычитание

SUB (.unit) src1, src2, dst
src1 вычитает src2, результат размещается в dst.

Формат данных 32 бита

.unit = .L1, .L2, .S1, .S2

Количество тактов : 1

Знаковое сложение

ADD (.unit) src1, src2, dst
src1 складывается с src2, результат размещается в dst.

Формат данных 32 бита

.unit = .L1, .L2, .S1, .S2

Количество тактов : 1

Знаковое умножение

MPY (.unit) src1, src2, dst
src1 умножается на src2, результат в dst

Формат данных 16 бит

.unit = .M1 или .M2

Количество тактов : 2; 1 такт на саму команду, 1 такт задержка выполнения. Команда считается выполненной после того, как произошла запись значения в dst.

Чтение из памяти

LDW (.unit) *+baseR[offsetR], dst ;
Команда считывет из памяти значение и размещает его в dst.

baseR - начальный адрес памяти, offsetR – смещение относительно начального адреса памяти.

Формат данных 32 бита

.unit = .D1 или .D2

Количество тактов : 5; 1 такт на саму команду, 4 такта задержка выполнения. Команда считается выполненной после того, как произошла запись значения в dst.

Команда перехода:

B (.unit) метка
.unit = .S1 или .S2

Перед командой можно указывать условие перехода. В квадратных скобках указывается регистр, если его значение не равно нулю, то происходит выполнение команды.

Количество тактов : 6; 1 такт на саму команду, 5 тактов задержка выполнения. Команда считается выполненной после того, как произошел переход.

В разделе .data производится объявление массивов с одновременной записью начальных значений.

В разделе .text загрузка адреса массива в регистр производится в два этапа, сначала в регистр загружаются младшие 16 бит адреса массива

```
MVKL .S1 array1,A3
```

, а затем производится загрузка старших 16 бит адреса массива

```
MVKH .S1 array1,A3
```

, это необходимо из-за того что адрес массива является 32 битным, а команда пересылки значения в регистр работает только с 16 разрядными числами, в целях сохранения корректности данных загрузка адреса массива в регистр производится в два этапа.

Алгоритм:

В начале работы программы производится инициализация используемых в работе регистров начальными значениями. Уменьшается счетчик количества элементов массива на 1, затем производится загрузка значения элемента первого и второго массива в

соответствующие регистры. Затем производится перемножение элементов массивов и полученное произведение суммируется с предыдущим. После происходит переход на метку при условии, что счетчик количества элементов массива не равен 0.

Каждая команда выполняется определенное количество тактов. Например, команда перехода [A2] B .S1 LOOP выполняется за 6 тактов работы процессора. Команда LDW .D1 *A3[A2], B2 выполняется за 5 тактов работы процессора. В программе пустые операторы NOP, необходимы для того, чтобы результат выполнения команды с ненулевой задержкой успел записаться в регистр - приемник до начала выполнения следующей команды программы.

Например, если убрать NOP после команды умножения MPY .M1X A5,B2,A4, то ее результаты работы, а именно содержимое регистра A4(значение произведения) примет значение равное произведению A5 и B2 к моменту выборки команды условного перехода [A2] B .S1 LOOP. В тот момент пока команда MPY еще будет выполняться, уже выполнится команда суммирования произведений ADD .L1 A0, A4,A0, так как она выполняется за 1 такт работы процессора.

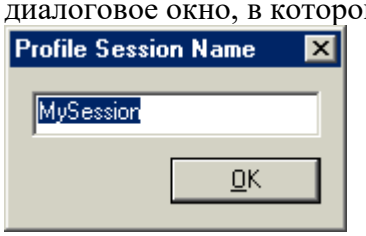
Таким образом, содержимое регистра A4 примет значение произведения после того, как команда ADD сложит предыдущие значение произведения с текущим произведением, данные будут потеряны и результаты вычислений будут неверными. Поэтому в работе программы необходимо получать значения расчетов до того времени, как следующая команда начнет выполняться.

С этой целью используются пустые операторы NOP. Если пользователь хочет, чтобы результаты работы команды были доступны следующей команде , то необходимо дать процессору время на обработку команды с ненулевой задержкой.

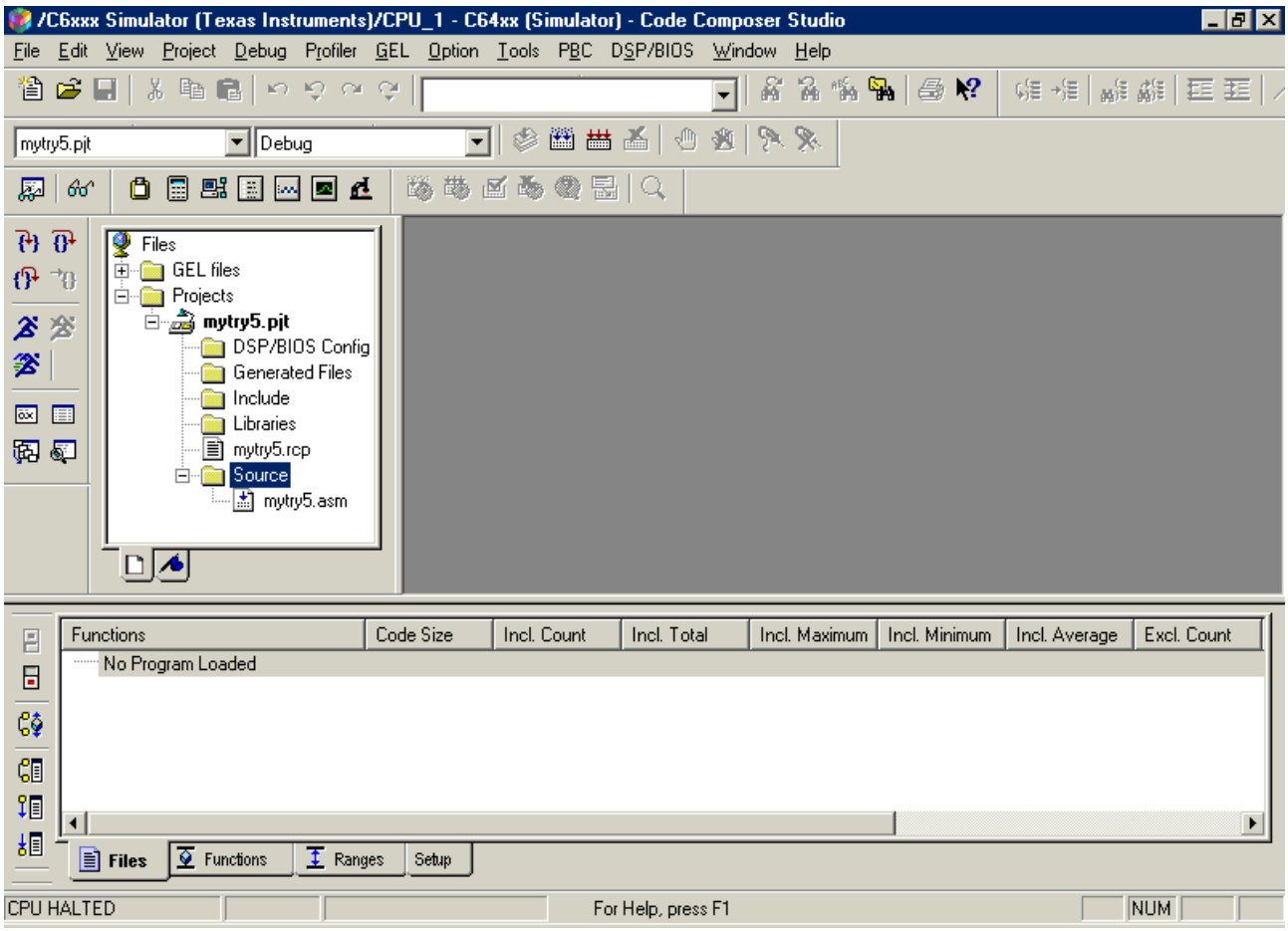
Подсчет тактов работы процессора:

Для того чтобы подсчитать количество тактов необходимо:

Выбрать пункт меню Profiler /Start New Session…, на экран будет выведено диалоговое окно, в котором пользователь может назначить имя сессии.

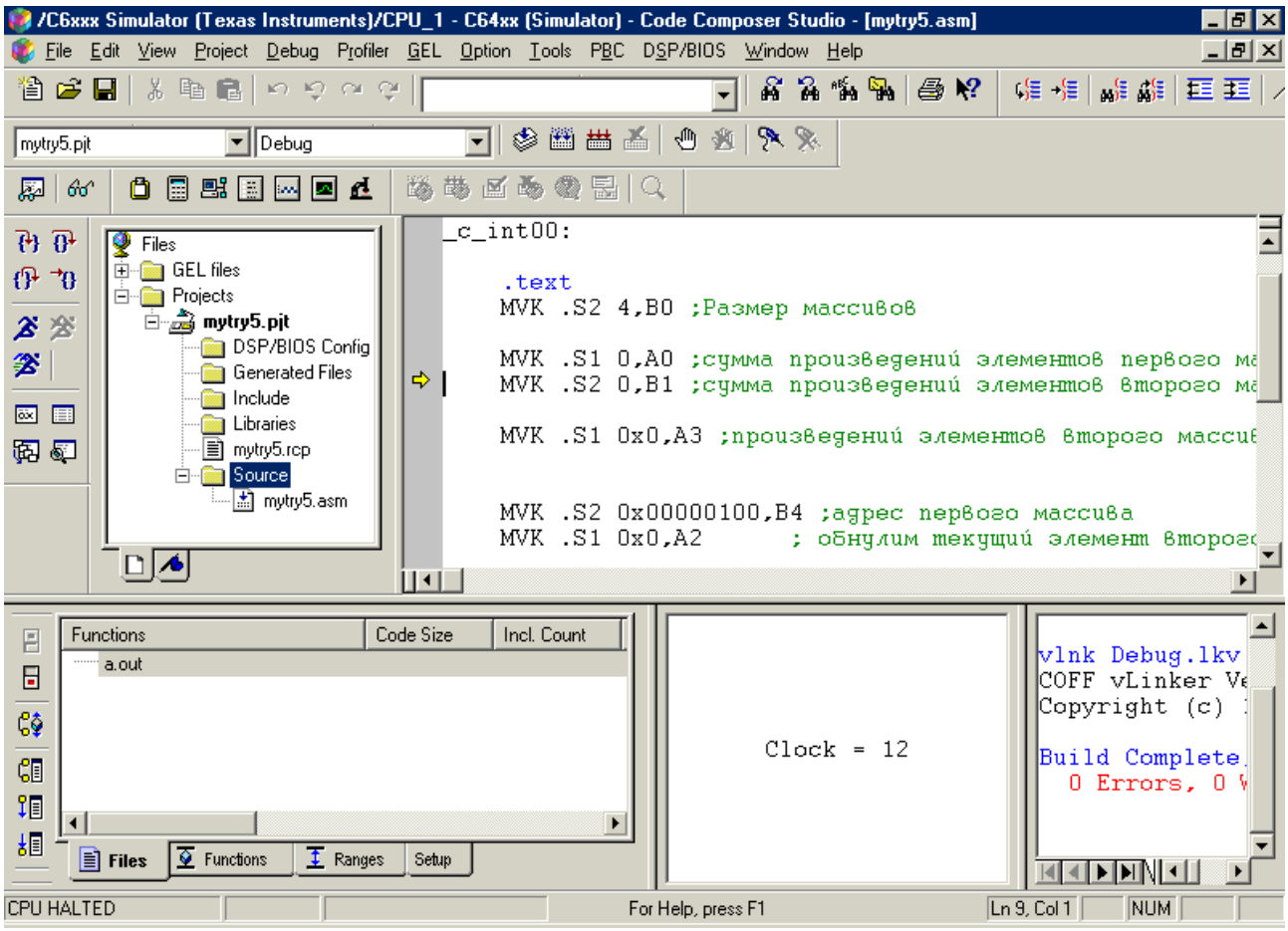


После того, как пользователь нажмет кнопку OK, на экран будет выведено диалоговое окно, где будут указаны исполнимые файлы программы для которых будет подсчитываться количество тактов.



В диалоговом окне будет надпись No Program loaded, поэтому следует запустить исполнимый файл программы. Для этого надо выбрать пункт меню File/Load Program …, затем в папке проекта выбрать файл *.out .

После запуска программы диалоговом окне сессии подсчета количества тактов будет отображен исполнимый файл.Затем чтобы просмотреть количество тактов надо выбрать пункт меню Profiler/View Clock, в результате на экране появится окно с указанием количества тактов.Clock = 0.Затем этого можно трассировать программу. После каждого шага трассировки в окне будет отображаться количество тактов команды или группы команд.



Для того, чтобы обнулить счетчик тактов необходимо два раза щелкнуть левой кнопкой мышки по надписи Clock=… В результате счетчик тактов обнулится.

4. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ:

В ходе выполнения лабораторной работы необходимо разработать программу, в соответствии с заданием. Отчет по лабораторной работе должен содержать описание индивидуального задания, граф схемы алгоритмов с их описанием, текст программы с соответствующими комментариями и пример результатов работы, а также количество тактов программы.

Варианты заданий:

1. Разработать программу, подсчитывающую количество нулевых элементов массива.
2. Разработать программу вычисления суммы векторов. Сложить элементы двух массивов, результат записать в третий.
3. Разработать программу вычисления разности векторов. Произвести вычитание элементов двух массивов, результат записать в третий.
4. Разработать программу вычисления повторяемости заданного значения в массиве.
5. Подсчитать количество отрицательных элементов в массиве.
6. Разработать программу, переставляющую элементы массива в обратном порядке.
7. Вычисление факториала.
8. Проверить является ли число палиндромом. Палиндром – это число, которое читается слева направо так же как и справа налево (напр. число 121).
9. Подсчитать сумму всех элементов массива, имеющих положительные значения.
10. Заменить все отрицательные значения элементов массива на модули их значений.
11. Найти количество вхождений в массив заданной комбинации двух чисел.
12. Подсчитать сумму элементов массива следующий образом: пусть имеется массив состоящий из 5 элементов, необходимо просуммировать элементы в следующем порядке 1 элемент суммируется с 5, 2 суммируется с 4.
13. Разработать программу, вычисляющую сумму элементов массива между двумя заданными. Задаются индексы массива.
14. Разработать программу, сдвигающую элементы массива на две позиции влево. Освободившиеся ячейки левой части массива заполняются элементами правой части.
15. Разработать программу, переставляющую элементы массива в обратном порядке между двумя заданными индексами массива.
16. Проверить является ли массив упорядоченным по возрастанию. Результат проверки записать в регистр.
17. Проверить является ли последовательность элементов между двумя заданными индексами массива упорядоченной по убыванию. Результат проверки записать в регистр.
18. Разработать программу увеличения значения элементов массива в два раза, если элемент имеет нечетный индекс, если элемент имеет четный индекс, то к его значению следует прибавить число 10.
19. Разработать программу подсчета суммы элементов массива у которых нечетные индексы.
20. Проверить являются ли значения элементов массива имеющие нечетные индексы упорядоченными по возрастанию.
21. Проверить являются ли элементы массива имеющие нечетный индекс упорядоченными по возрастанию, а элементы массива имеющие четный индекс упорядоченными по убыванию.

Код варианта задания определяется тремя компонентами:

- Номер задания
- Формат данных (byte(b) – 1 байт; short(s) – 2 байта; int(i) – 4 байта;)
- Со знаком или без знака (signed(s) / unsigned(u))

Таблица вариантов

№ вар.	1	2	3	4	5	6	7	8
Код вар.	1bu	2ss	3is	4bs	5ss	6su	7iu	8bu

№ вар.	9	10	11	12	13	14	15	16
Код вар.	9ss	10is	11bu	12ss	13iu	14bu	15is	16bs

№ вар.	17	18	19	20	21	22	23	24
Код вар.	17su	18is	19bu	20ss	21iu	1iu	2is	3bu

№ вар.	25	26	27	28	29	30	31	32
Код вар.	4is	5ss	6bu	7bu	8su	9ss	10iu	11su

Список литературы:

1. **Общие теоретические сведения.**
   Перевод - файл “Процессор TMS 320C6000.doc”
   Английский вариант - файл “spru189f.pdf”

2. **Описание команд**
   Английский вариант - файл “spru189f.pdf”
   раздел TMS320C62x/C64x/C67x Fixed-Point Instruction Set
   Электронная документация - раздел Instruction Set Summary

3. **Структура ассемблерного кода**
   Перевод -раздел Structure of Assembly Code(***)

   Английский вариант -файл spru198f.pdf
   раздел Structure of Assembly Code

   Электронная документация - раздел
   TMS320C6000 Programmer’s Guide/ Structure of Assembly Code

4. **Директивы ассемблера**
   Перевод -раздел Assembler Directives(***)

   Английский вариант -файл spru186i.pdf
   раздел Assembler Directives

   Электронная документация - раздел
   Code Generation Tool OnLine Documentaion/ Using the Assembler/ Assembler Directives

5. **Описание ассемблера**
   Перевод -раздел Assembler Description(***)

   Английский вариант -файл spru186i.pdf
   раздел Assembler Description

   Электронная документация - раздел
   Code Generation Tool OnLine Documentaion/ Using the Assembler/ Assembler Description

6. **Руководство по среде программирования (***)**

7. **Приложение к методическому пособию (***)**