МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» 12 2017

РАЗРАБОТКА ПРОГРАММНОГО КОДА ДЛЯ ПРОЦЕССОРА NEUROMATRIX NM6403

Методические указания к выполнению лабораторных работ для студентов направления подготовки 09.03.01 по дисциплине «Микропроцессорные системы в системах передачи и обработки данных»

Курск 2017

УДК 004.383.8.032.26

Составитель: В.С. Панищев

Рецензент
Кандидат технических наук *Халин Ю.А.*

Разработка программного кода для процессора NeuroMatrix NM6403: методические указания к выполнению лабораторной работы / Юго-Западный гос. ун-т; сост.: В.С. Панищев. – Курск, 2017. – 23 с.

Рассмотрены возможности микропроцессора NEUROMATRIX NM6403 (Л1879ВМ1). Изложены рекомендации по написанию, отладке программного кода и работе в среде разработки и в программе-эмуляторе процессора.

Методические указания соответствуют Федеральному государственному образовательному стандарту высшего образования направления подготовки 09.03.01 Информатика и вычислительная техника, учебному плану направления подготовки 09.03.01 Информатика и вычислительная техника, одобренному Ученым советом университета (протокол № 7 «29» февраля 2016 г.).

Предназначены для студентов направления подготовки 09.03.01 очной и заочной формы обучения

Текст печатается в авторской редакции

Подписано в печать 15.12.17 . Формат 60*84 1/16.
Усл. печ.л. 1,1 . Уч.-изд.л. 1,0 Тираж 100 экз. Заказ 4789 Бесплатно.
Юго-Западный государственный университет
305040 Курск, ул. 50 лет Октября, 94.

3

РАЗРАБОТКА И ОТЛАДКА ПРОГРАММНОГО КОДА ДЛЯ ПРОЦЕССОРА NEUROMATRIX NM6403

1. Цель работы

Изучение возможностей микропроцессора NEUROMATRIX NM6403 (Л1879ВМ1), способов написания и отладки программного кода, среды разработки и эмулятора процессора.

2. Структура нейропроцессора NM6403

Рассмотрим архитектурные особенности процессора NM6403.

Разработанный в НТЦ “Модуль” процессор NM6403 представляет собой высокопроизводительный микропроцессор с элементами VLIW/SIMD архитектурой. В его состав входят устройства управления, вычисления адреса и обработки скаляров, а также узел для поддержки операций над векторами с элементами переменной разрядности. Кроме того, имеются два идентичных программируемых интерфейса, которые выходят на внешние шины (глобальную и локальную), по которым процессор может обращаться к двум внешним памятям, каждая из которых может содержать до двух банков, различающихся типом и временными параметрами. При обращении во внешнюю память процессор использует 32-разрядный вычисляемый адрес. Разрядность шин данных - 64, однако обмен может происходить как по 64, так и по 32 разряда.

RISC-ядро процессора содержит восемь 32-х разрядных адресных регистра, восемь 32-х разрядных регистров общего назначения, счетчик адреса программы и слово состояния программы, два функциональных устройства адресных вычислений, функциональное устройство для выполнения операций над скалярами, два таймера, а также регистры управления и состояния.

Векторный узел включает в себя три блока внутренней памяти , каждый из которых содержит тридцать два 64-х разрядных слова, набор специальных регистров управления, а также функциональное устройство с настраиваемой на разрядность операндов структурой для выполнения матричных операций.

Глобальная и локальная шина используются для доступа к внешней памяти. Память, которая доступна через глобальную шину, называется **глобальной памятью**. Память, доступная через локальную шину, называется **локальной** [1, 2].

Для обмена данными с ВУ процессор NM6403 имеет 4 канала. Внешний интерфейс NM6403 представлен на рисунке 1.

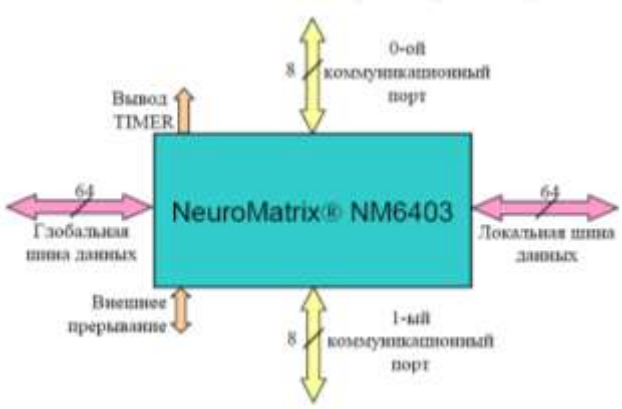

Рис. 1. Внешний интерфейс процессора NM6403

Помимо работы с внешней памятью процессор может принимать и передавать данные через коммуникационные порты.

Процессор имеет два коммуникационных порта связи (линка), аппаратно совместимых с портами TMS320C4x, работающие с темпом до 20 Мбайт/сек каждый. Коммуникационные порты связывают данный процессор с другими такими же, или с процессорами TMS320C4x, которые имеют аналогичный интерфейс обмена данными и, таким образом, соединение процессоров по линкам не требует дополнительной логики. Внутренней памяти в обычном понимании в процессоре нет [1, 2].

Процессора NM6403 состоит из следующих основных блоков:

1. скалярный процессор (СП);
2. векторный процессор (ВП);
3. два DMA-сопроцессора, управляющие работой коммуникационных портов;
4. два таймера;
5. регистры управления интерфейсом доступа к внешней памяти.

Скалярный процессор представляет собой RISC ядро, отвечающее за подготовку данных для векторного процессора. Он также может использоваться и как самостоятельный вычислительный блок. СП имеет 8 адресных регистров и 8 регистров общего назначения, а также набор регистров специального назначения.

СП позволяет осуществлять следующие операции:

• различные виды адресации с модификацией адресных регистров;

• чтение/запись в память как 32-х разрядных слов, так и пар слов, образующих 64-х разрядное число.

• все виды арифметических и логических операций над регистрами общего назначения с модификацией и без модификации флагов состояния;

• различные типы сдвигов, в том числе на произвольное количество битов;

• условные и безусловные переходы, в том числе отложенные переходы;

• вызовы функций с записью в стек адреса возврата, в том числе и

отложенные вызовы функций;

• пошаговое умножение;

• управление таймерами;

• настройка регистров управления доступом к внешней памяти на тип, используемый в конкретном устройстве;

• управление векторным процессором путем задания его конфигурации.

Внутренняя структура процессора представлена на рисунке 2.

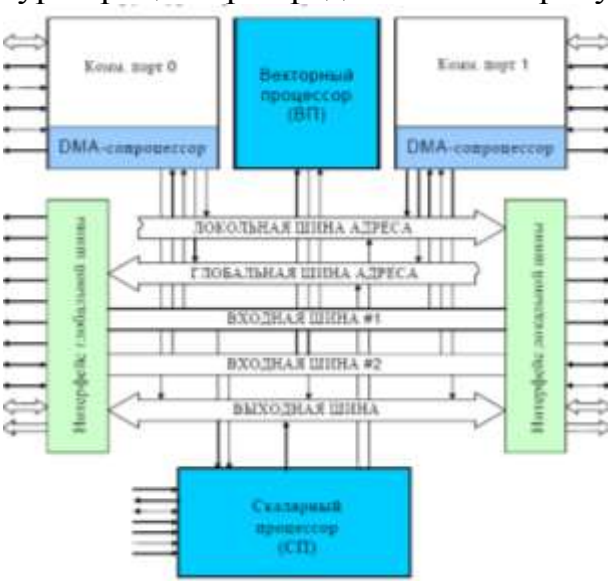

Рис. 2. Внутренняя структура процессора NM6403

Векторный процессор является основной особенностью процессора NM6403, он работает совместно со скалярным процессором и двумя DMA-сопроцессорами. Данный 64-х разрядный процессор предназначен для выполнения параллельных операций и способен выполнять одновременно до 2048 операций за один такт. Его архитектура позволяет менять отношение производительность/точность, изменяя разбиение данных (векторов) на элементы с одного бита до 64-х. Векторный процессор выполняет множественные действия над данными за одну инструкцию, так называемая SIMD (single instruction, multiple data) параллельная обработка. Процессор NM6403 имеет 64-х разрядный интерфейс работы с внешней памятью. За одно обращение к памяти он позволяет записать или прочитать одно 64-х разрядное число по каждой из шин [1, 2].

Рассмотрим более подробно структуру векторного процессора (см. рис. 3).

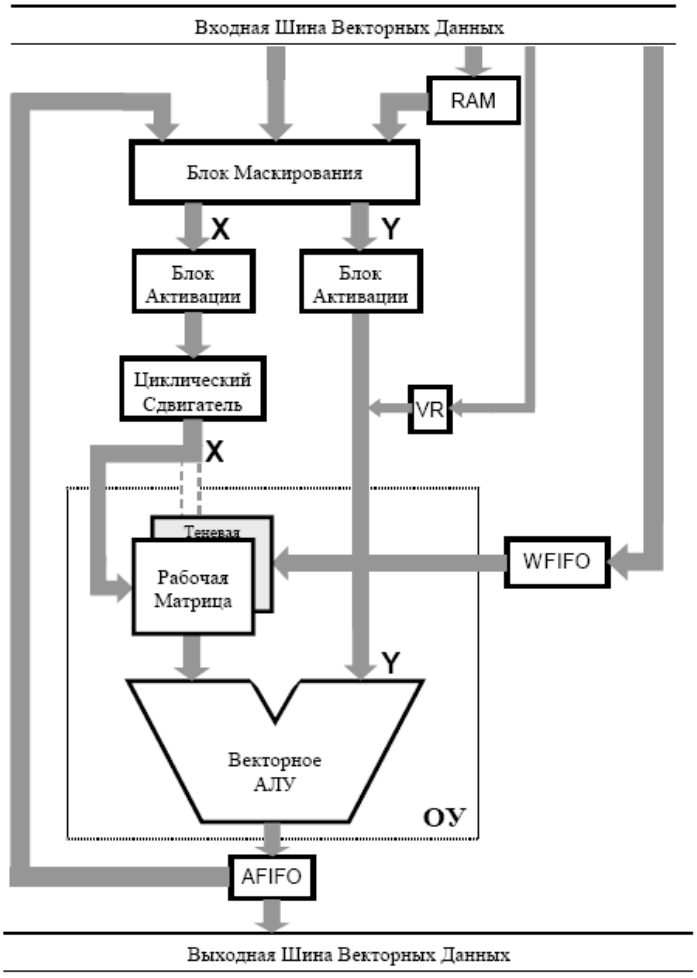

Рис. 3. Внутренняя структура векторного процессора

Входная Шина Векторных Данных, изображенная на Рис. 3, соответствует на Рис. 2 Входной шине #1 и Входной Шине #2, Выходная Шина Векторных Данных отражена как Выходная Шина. Центральным узлом Векторного Процессора является операционное устройство (ОУ). Оно состоит из Рабочего/Теневого Матричного блока и Векторного АЛУ. Операционное Устройство (ОУ) используется для выполнения операций умножения с накоплением, арифметических и логических операций, произвольной перестановки битов в векторах упакованных данных.

ВП состоит из следующих узлов:

• Рабочая матрица - операционный узел, в котором осуществляются операции умножения с накоплением. С рабочей матрицей связана пара регистров, которые определяют ее разбиение на столбцы и строки.

• Теневая матрица - устройство, используемое для ускорения закачки весовых коэффициентов в рабочую матрицу. В то время, как рабочая матрица участвует в операции умножении с накоплением, в теневую может параллельно подкачиваться новая порция весовых коэффициентов. После того, как теневая матрица загружена, она в течение одного процессорного такта может быть

перегружена в рабочую. С теневой матрицей связана своя пара регистров,

определяющая ее разбиение на столбцы и строки. Это разбиение может быть отличным от того, которое использовалось в рабочей матрице на предыдущем этапе;

• Векторное АЛУ - устройство, позволяющее совершать стандартный набор арифметических и логических операций над парами 64-х разрядных слов, каждое из которых разделено на малоразрядные элементы. При арифметических операциях в случае переполнения внутри диапазона, отведенного под один малоразрядный элемент, блокируется перенос битов в соседний элемент.

• Буфер весовых коэффициентов (wfifo) - очередь глубиной в 32 64-х разрядных слова, организованная по принципу FIFO. В нее подгружаются весовые коэффициенты, и в ней они хранятся, прежде чем происходит их загрузка в теневую матрицу. Загрузка данных и их выгрузка из wfifo может осуществляться по частям, то есть, например, в wfifo можно загрузить сначала 8 слов, а затем еще 24, но так, чтобы не произошло переполнения.

• Буфер внутренней памяти (ram) - очередь глубиной в 32 64-х разрядных слова, организованная по принципу FIFO. Используется, как один из аргументов в операциях умножения с накоплением, а также в операциях на векторном АЛУ. В ram может быть загружено от 1 до 32 слов. Буфер может использоваться многократно, однако в операциях должны участвовать все данные, хранящиеся в ram. Не допускается использование только части хранящихся там данных.

• Псевдобуфер шины данных (data) используется для обозначения данных, находящихся на шине данных непосредственно в процессе их загрузки из памяти в ВП. Позволяет обрабатывать данные на проходе. Псевдобуфер имеет глубину в 32 64-х разрядных слов и организован по принципу FIFO. Используется, как один из аргументов в операциях умножения с накоплением, а также в операциях на векторном АЛУ.

• Буфер накопления результатов (afifo) – очередь глубиной в 32 64-х разрядных слова, организованная по принципу FIFO. Результат любой операции на ВП сохраняется в afifo. Может также использоваться, как один из аргументов в операциях умножения с накоплением и в операциях на векторном АЛУ. Для того, чтобы получить доступ к результатам вычислений на векторном процессоре, хранящимся в afifo, необходимо выгрузить их в память.

• Векторный регистр (vr) – 64-х разрядный регистр, используемый в качестве определенного операнда в операции умножения с накоплением. Можно представить его, как буфер, состоящий из заданного количество одинаковых слов. Может быть загружен из СП. Доступен только на запись.

• Устройства, обеспечивающие выполнение функции активации над входными векторами. В ВП содержится два устройства, работающих независимо. Они позволяют активировать входные данные перед выполнением операции умножения с накоплением или перед подачей их на векторное АЛУ.

• Устройства, обеспечивающие выполнение операции маскирования над

входными векторами.

• Устройство, обеспечивающее циклический сдвиг вправо на один бит слова данных, подаваемого на вход X рабочей матрицы в операции взвешенного суммирования.

3. Формат данных и система команд процессора NM6403

ВП осуществляет обработку векторов. Под векторами понимаются одномерные массивы однородных данных, расположенные в памяти в виде непрерывного блока. Матрица - это массив векторов. Разрядность всех узлов ВП составляет 64 бита. ВП осуществляет обработку целочисленных данных, которые упакованы в 64-х разрядные слова с помощью простой конкатенации (см. рис. 4). В общем случае слово упакованных данных представляет собой вектор D = {Dk...D1}, содержащий k элементов, суммарная разрядность которых составляет 64 бита. Причем в одном слове D могут быть упакованы данные, имеющие разную разрядность. Количество элементов k, упакованных в одном слове, зависит от их разрядностей и может принимать целочисленное значение в диапазоне от 1 до 64.

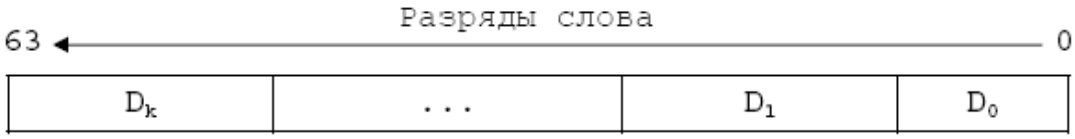

Рис. 4. Формат упакованных целочисленных данных

Все команды процессора делятся на две группы: скалярные и векторные.

Скалярные команды осуществляют загрузку/чтение всех регистров (доступных по чтению/записи) за исключением векторных регистров, образующих очереди FIFO.

Векторные команды управляют работой векторного процессора.

Все команды процессора состоят из двух частей, называемых условно «левой» и «правой». В левой части команды записываются только адресные операции, в правой все арифметическо-логические, не связанные с вычислением адресов и обращением к памяти. Обе части команды выполняются процессором параллельно.

Левая и правая части команды соединяются воедино при помощи ключевого слова **with**. Пример инструкции процессора:

gr0 = [ar0++]	with	gr1 = gr2 and gr3;
левая часть,		правая часть,
адресная операция		логическая операция

В языке ассемблера левая или правая часть инструкции может быть опущена, однако поскольку процессор не может выполнить только левую или только правую часть команды, вместо опускаемой части при компиляции автоматически добавляется пустая операция nul.

Например, команда gr0 = gr0 + 1 трактуется ассемблером как nul with gr0 = gr0 + 1 или команды gr0 = [ar0]; трактуется как gr0 = [ar0] with nul.

Векторные команды процессора также как и скалярные разделены на левую и правую части. Помимо этого все векторные команды, за исключением команд загрузки теневой и рабочей матрицы - **ftw** и **wtw,** имеют поле префикс повторения **rep n,** где n – количество повторений векторной команды.

Например rep 1 data = [ar0] with data + afifo.

4. Примеры разработки и откладки программного кода

Базовое программное обеспечение процессора NM6403.

В состав базового программного обеспечения (БПО) процессора NM6403 входят следующие компоненты:

nmcc.exe - компилятор Си++ (драйвер компонент);

asm.exe – ассемблер;

linker.exe – редактор связей;

libr.exe – библиотекарь;

dump.exe – декодер и исполняемых файлов;

emurun.exe – эмулятор процессора на уровне инструкций;

teme.exe – точный эмулятор процессора;

emudbg.exe – многоцелевой подключаемый отладчик.

Подробное описание компонент приводится в [3, 4].

БПО NM6403 поддерживает разработку программ как на языке низкого уровня - ассемблере так и на языке высокого уровня – Си++.

Специальных редакторов для разработки программ в составе БПО нет, поэтому для написания программ можно воспользоваться любым текстовым редактором, например **notepad.exe**, входящим в состав Windows.

Рассмотрим на конкретном примере процесс разработки и откладки программного кода для процессора NM6403.

В листинге 1 представлена программа, осуществляющая сложение двух одномерных массивов типа word.

Листинг 1

```
// объявление глобальной метки
global __main: label;
// секция инициализированных данных
data ".MyData"
        A : word[16] = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16);
        B : word[16] = (17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32);
end ".MyData";
// секция неинициализированных данных
nobits ".MyData1"
      C : word[16];
end ".MyData1";
// секция кода
begin ".textMyCode"
<__main>
```

```
        // адрес массива A
        ar0 = A;
        // адрес массива B
        ar1 = B;
        // адрес массива C
        ar2 = C;
        // счетчик цикла
        gr0 = 16;

<loop>
        // i-й элемент массива A в gr1
        gr1 = [ar0++];
        // i-й элемент массива B в gr1
        gr2 = [ar1++];
        // gr3 = A[i] + B[i]
        gr3 = gr1 + gr2;
        // C[i] = gr3
        [ar2++] = gr3;
        // уменьшаем счетчик цикла
        gr0--;
        // если gr0 > 0 переход на метку loop
        if > goto loop;
        return;
end ".textMyCode";
```

Пример начинается с объявления глобальной метки __main, которая определяет адрес команды, с которой начинается тело основной программы. Метки можно объявлять в любом месте ассемблерного файла, однако для лучшей читаемости кода рекомендуется выносить его за пределы секций.

Метка __main особая, так как она является меткой начала пользовательской программы. Функция с этим именем автоматически добавляется к любой пользовательской программе при компиляции.

За объявлением глобальной метки следует секция инициализированных данных. В этой секции содержатся объявления и инициализация переменных, используемых программой. Секция данного типа начинается с открывающейся скобки data и заканчивается словом end (закрывающаяся скобка). В нашем примере в секции с именем «.MyData» объявлено два массива A и B размером в 16 слов.

После секции инициализированных данных следует секция неинициализированных данных. В этой секции содержатся объявления переменных используемых программой, без их инициализации. Секция данного типа начинается с открывающейся скобки nobits и заканчивается словом end. В нашем примере в секции неинициализированных данных с именем «.MyData1» объявлен массив C размером в 16 слов для результата

сложения.

Далее идет секция кода. Секция кода начинается с открывающей скобки begin и заканчивается закрывающей скобкой end. Рекомендуется имя секции кода начинать с префикса text, как в нашем примере «.textMyCode». Дизассемблер, разбирая первые символы имени секции, поймет, что это код программы и представит ее содержимое в виде дизассемблированных инструкций. В противном случае он оставит содержимое секции в виде бинарного кода.

Алгоритм работы программы заключается в следующем. В адресные регистры скалярного процессора ar0, ar1, ar2 загружаем адреса массивов A, B, C соответственно. В регистр общего назначения gr0 загружаем счетчик циклов. Затем в цикле за 16 итераций складываем поэлементно содержимое массивов A и B, результат сохраняем в массив C

Для компиляции данного примера необходимо в командной строке ввести команду:

nmcc -g primer_1.asm libc.lib –m

Программа nmcc представляет собой специальную оболочку (shell), упрощающую процесс запуска компилятора. Она автоматически определяет, какой набор системных компонент необходимо вызвать для сборки исполняемого файла.

Параметры nmcc могут располагаться в произвольном порядке. Параметр –g включает отладочную информацию в выходной файл. Параметр –m включает порождение файла-карты памяти, где показано, какой объём памяти отведён под те или иные секции кода и данных, каковы их адреса, приводит список глобальных переменных и т.д.

Для того чтобы компиляция примера primer_1.asm прошла успешно, в командную строку добавлен файл libc.lib. Это библиотека времени выполнения Си. В библиотеке содержится стартовый код программы, определена точка входа start. Стартовый код позволяет выполнять и отлаживать программу при помощи набора утилит, входящих в состав SDK. В частности, там содержится код останова, куда программа приходит по окончании выполнения. Именно по выполнению этого кода утилиты, запустившие программу на исполнение, понимают, что программа завершена.

Если исходный файл программы написан на языке Си++ то при компиляции нет необходимости включать библиотеку libc.lib так как она будет подключена автоматически.

После успешного выполнения компиляции будут созданы следующие файлы primer_1.elf – объектный файл, primer_1.map – файл карты-памяти, primer_1.abs – исполняемый файл.

Проследим за выполнением программы с помощью многоцелевого подключаемого отладчика **emudbg.exe**. Данный отладчик позволяет вести отладку на программном эмуляторе NM6403 уровня команды, а так же на процессорах PCI-платы МЦ4.01.

После запуска программы необходимо в появившемся диалоговом окне выбрать отладочную цель (см. рис. 4).

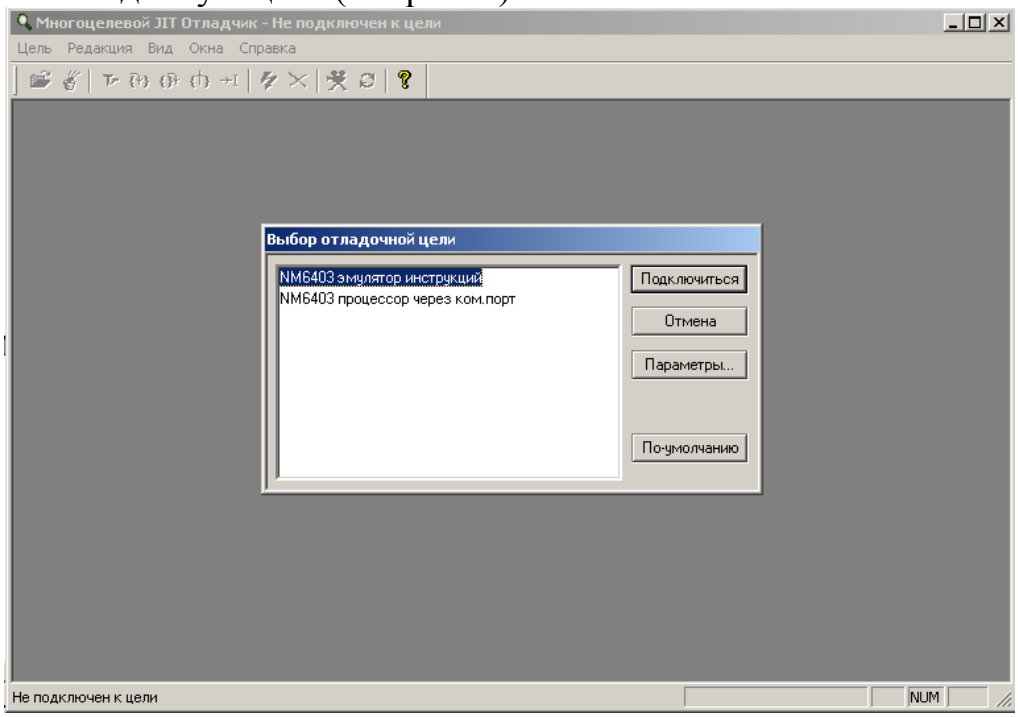

Рис. 4. Выбор отладочной цели

Выбираем эмулятор инструкций и нажимаем кнопку «Подключиться».

Далее в главном меню выбираем Цель→Загрузить программу. В появившемся диалоговом окне выбираем файл primer_1.abs. На рисунке 5 представлено окно отладчика после загрузки программы primer_1.abs.

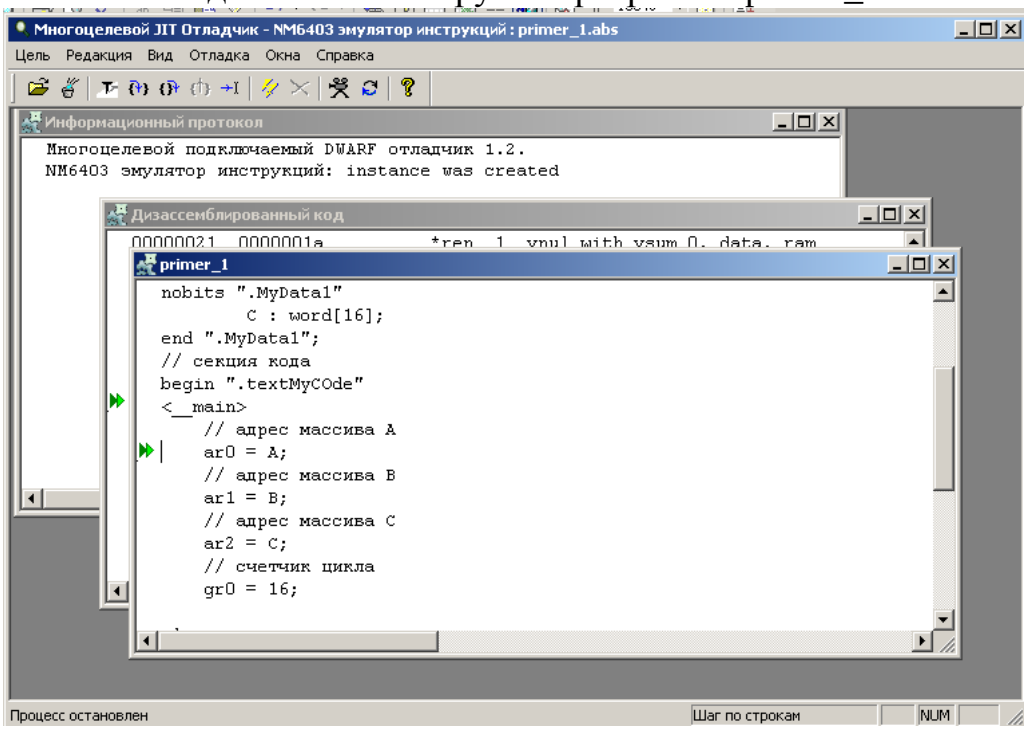

Рис. 5. Окно отладчика после загрузки программы

Затем выполняем команды главного меню Вид→Регистры и

Вид→Память (см. рис. 6).

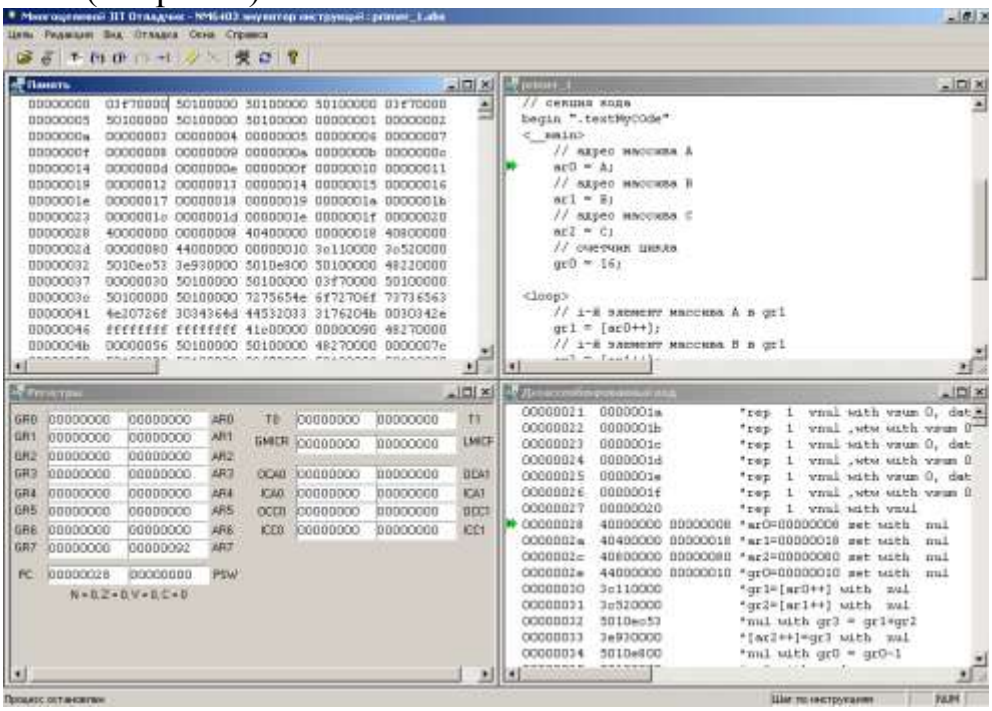

Рис. 6. Окна «Память» и «Регистры»

Команды, управляющие ходом выполнения отлаживаемой программы, сгруппированы в меню Отладка. Сюда входят различного вида шаги по программе, установка с снятие точек останова, запуск, останов, рестарт, а также специальные действия – запуск программы с отсоединением от отладочной цели. На рисунке 7 представлена оперативная панель управления режимами отладки.

Рис. 7. Оперативная панель управления режимами отладки

На рисунке 8 показано окно состояния памяти после выполнения программы primer_1.abs. Результат сложения массивов

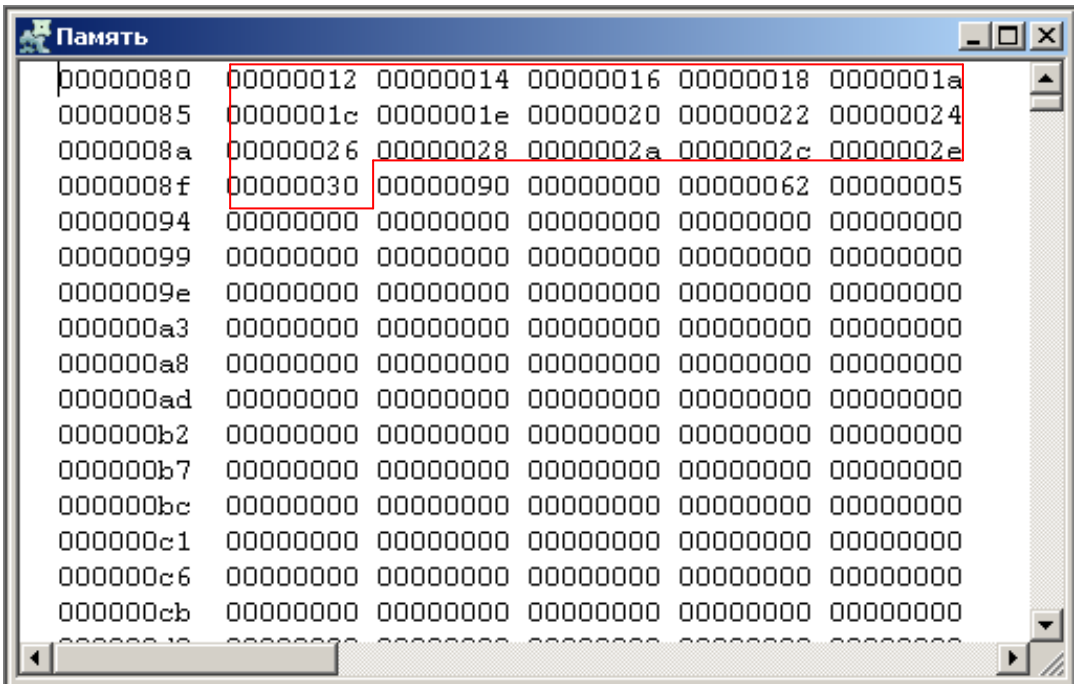

Рис. 8. Состояние памяти после выполнения программы

Значения результирующего массива C - это выделенная область на рисунке 8.

Сложение двух массивов в primer_1.asm было выполнено с помощью команд скалярного процессора. Для обработки массивов данных, как упоминалось ранее, в состав NM6403 входит векторный процессор, поэтому обработку массивов данных предпочтительнее выполнять на ВП.

В листинге 2 представлено решение той же задачи с помощью команд векторного процессора.

Листинг 2

```
// объявление глобальной метки
global __main: label;
// секция инициализированных данных
data ".MyData"
   A : word[16] = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16);
   B : word[16] = (17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32);
end ".MyData";
// секция неинициализированных данных
nobits ".MyData1"
        C : word[16];
end ".MyData1";
// секция кода
begin ".textMyCode"
<__main>
    // адрес массива A
    ar0 = A;
```

```
    // адрес массива B
    ar1 = B;
    // адрес массива C
    ar2 = C;
    // разбиение по два 32-разрядных элемента в векторе
    nb1 = 80000000h;
    // копирование содержимого теневого регистра nb1 в рабочий nb2
    wtw;
    // Загружаем в ram массив B
    rep 8 ram = [ar1++];
    //  afifo = A[i] + B[i]
    rep 8 data = [ar0++] with data + ram;
    // C[i] = afifo
    rep 8 [ar2++] = afifo;
    return;
end ".textMyCode";
```

Аналогично скалярной векторная инструкция состоит из левой и правой частей. В левой части содержится команда обращения к памяти на чтение/запись, а в правой операции на векторном процессоре.

Как и в предыдущем примере в адресные регистра скалярного процессора ar0, ar1, ar2 загружаем адреса массивов A, B, C соответственно.

Команда rep 8 ram = [ar1++]; осуществляет чтение из внешней памяти массива B в регистр-контейнер ram. Префикс повторения в данной команде равен 8 так как данный на ВП поступают по 64 бита.

При выполнении арифметических и логических операций, необходимо определить разбиение 64-разрядных векторов, поступающих на вход векторного АЛУ, на элементы. Это действие осуществляется с помощью регистра nb1.

В данной программе перед векторными инструкциями следуют команды

```
nb1 = 80000000h;
wtw;
```

При выполнении логических операций на векторном АЛУ данные команды можно опустить, так как не осуществляется межразрядный перенос.

Команда rep 8 data = [ar0++] with data + ram; осуществляет сложения содержимого логического регистра-контейнера data, в который загружается массив A, с содержимым регистра-контейнера ram. Результат сложения будет храниться в регистре-контейнере afifo.

Команда rep 8 [ar2++] = afifo; записывает результат в память по адресу в регистре ar2.

Загрузим программу primer_2.abs в отладчик и пронаблюдаем за состоянием основных узлов ВП (см. рис. 9).

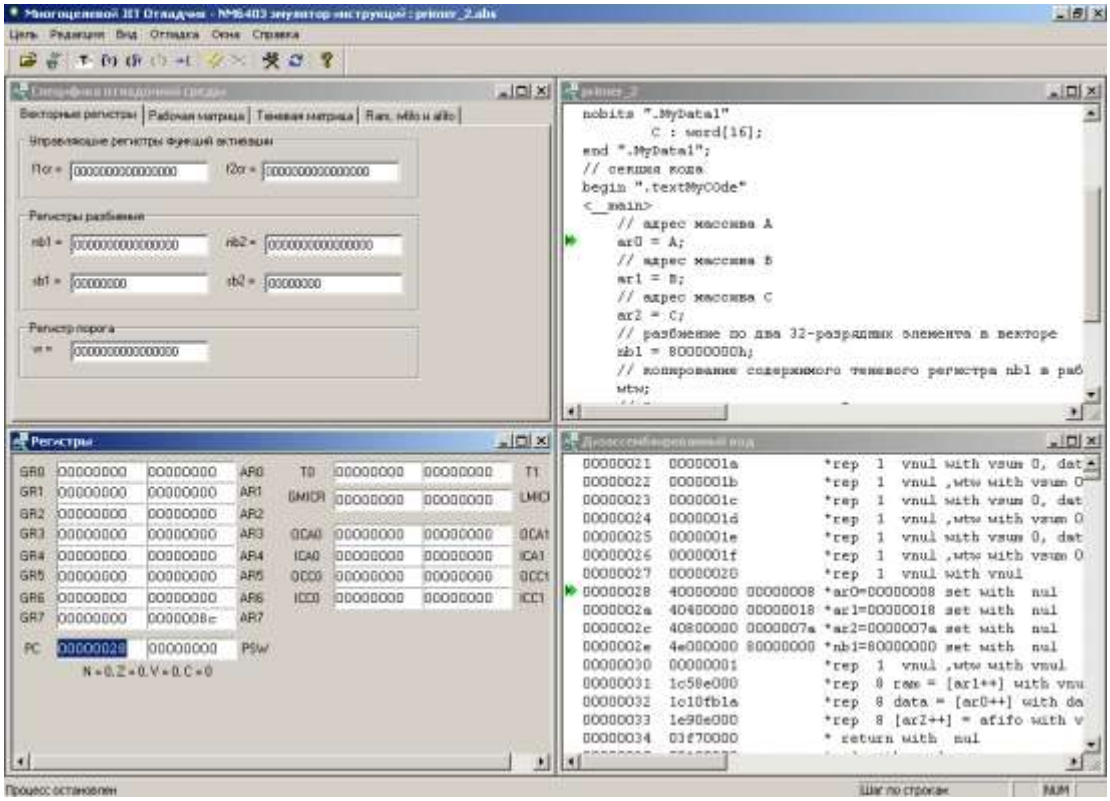

Рис. 9. Окна отладчика для primer_2.abs

Для просмотра состояния основных узлов ВП предназначено рабочее окно отладчика «Специфика отладочной среды», которое вызывается командой главного меню Вид→Специфика отладочной среды (см. рис. 10).

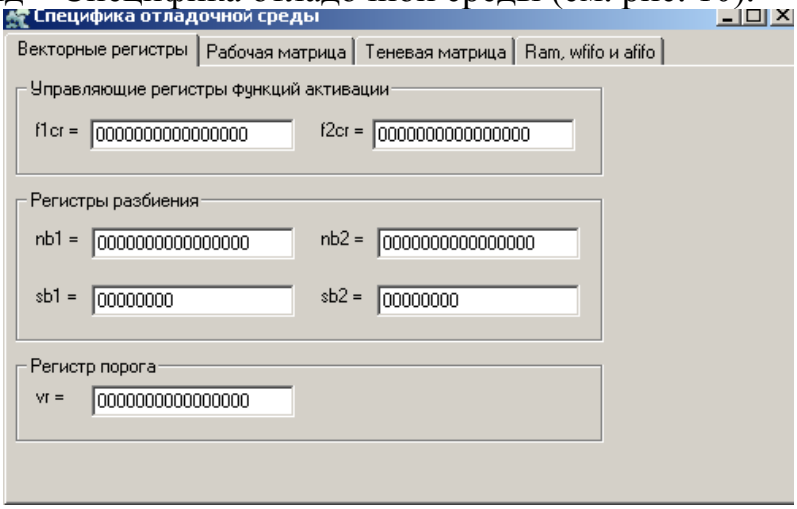

Рис. 10. Рабочее окно отладчика «Специфика отладочной среды»

На рисунке 11 показано состояние регистра-контейнера ram после выполнения команды загрузки.

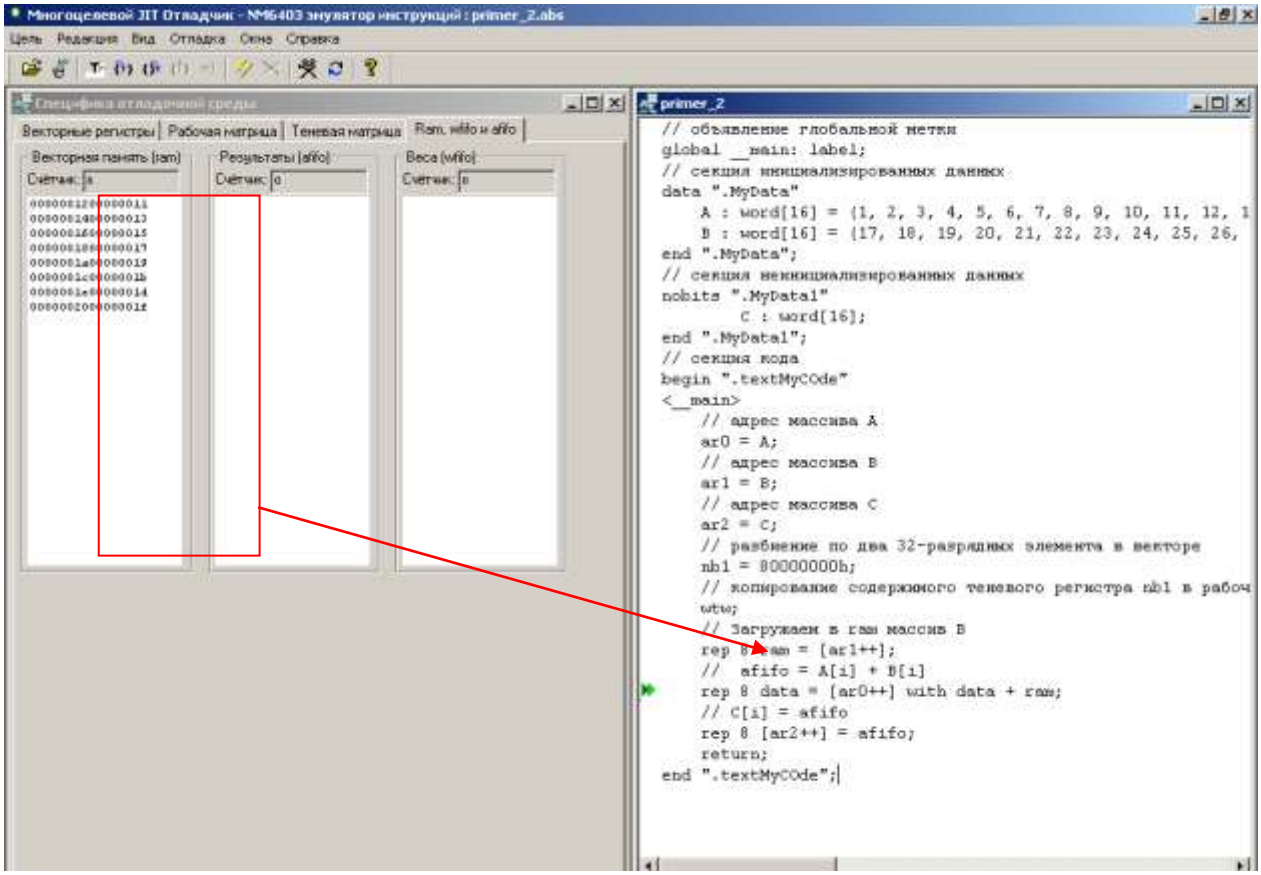

Рис. 11. Содержимое регистра-контейнера ram

На рисунке 12 показано состояние регистра-контейнера afifo после выполнения операции сложения.

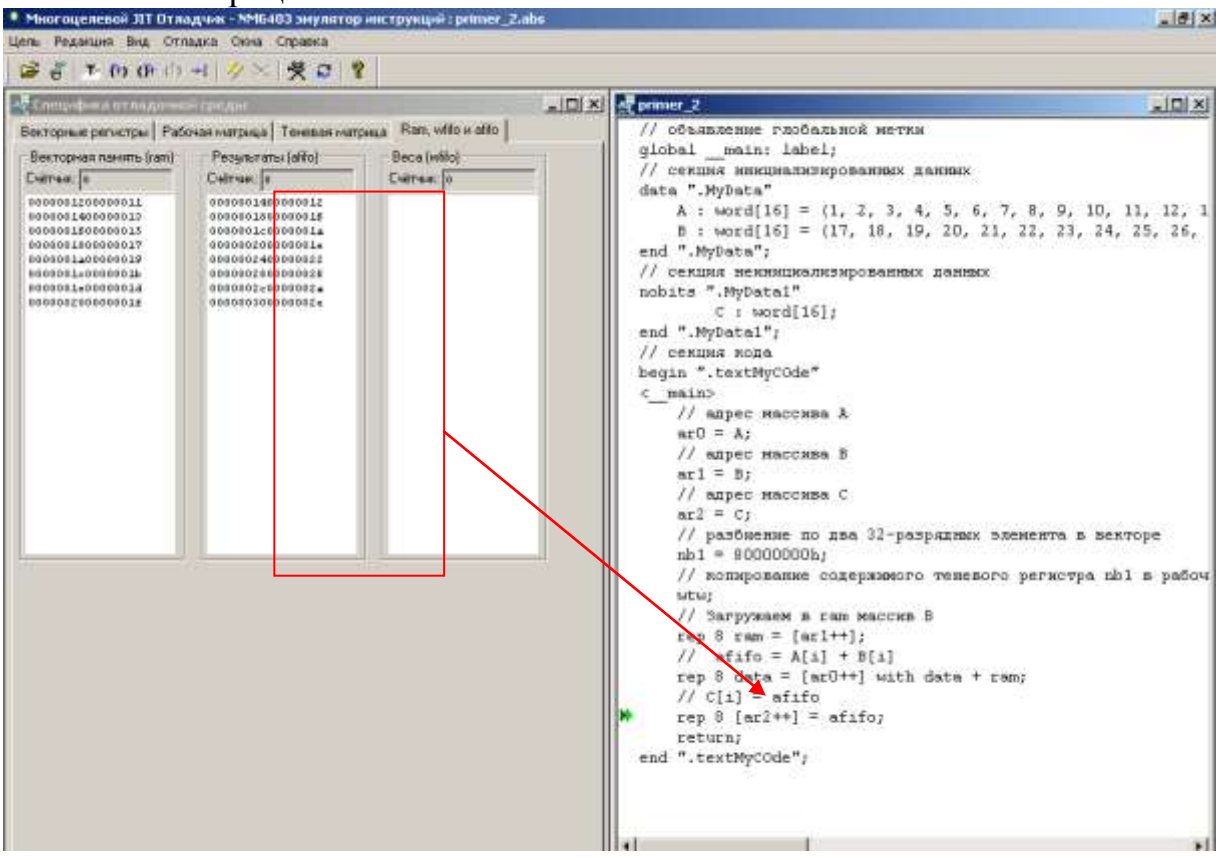

Рис. 12. Содержимое регистра-контейнера afifo

На рисунке 13 показано содержимое памяти после выполнения операции записи в память содержимого регистра-контейнера afifo.

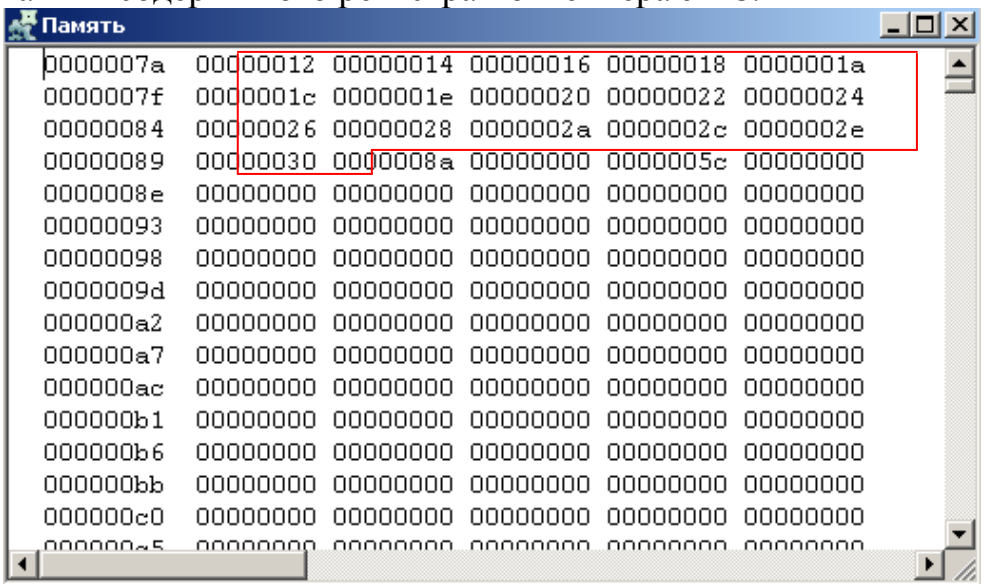

Рис. 13. Содержимое памяти

Рассмотрим еще один пример. Пусть необходимо написать программу, осуществляющую умножения матриц A[2][2] * B[2][2], результат - матрица C[2][2]. Условимся, что переполнение при умножении не возникает.

В листинге 3 представлено решение поставленной задачи.

Листинг 3

```
global __main: label;
data ".MyData"
        // матрица A[2][2]
        A: word[4] = (1, 2, 3, 4);
        // матрица B[2][2]
        B: word[4] = (5, 6, 7, 8);
end ".MyData";
nobits ".MyData1"
        // матрица для результатов C[2][2]
        C: word[4];
end ".MyData1";
begin ".textMyCode"
<__main>
        // адрес массива A
        ar0 = A;
        // адрес массива B
        ar1 = B;
        // кол-во столбцов рабочей матрицы
        nb1 = 80000000h;
```

```
        // кол-во строк рабочей матрицы
        sb = 2h;
        // загружаем матрицу B в wfifo
        rep 2 wfifo = [ar1++];
        // копируем весовые коэффициенты из wfifo в теневую матрицу
        ftw;
        // копирование из теневой матрицы в рабочую
        wtw;
        // взешанное суммирование
        rep 2 data = [ar0++] with vsum, data, 0;
        // сохраняем результат в память
        ar2 = C;
        rep 2 [ar2++] = afifo;
        return;
end ".textMyCode";
```

В данном примере умножение матриц основано на операции взвешенного суммирования, которая поясняется следующим рисунком (см. рис. 14).

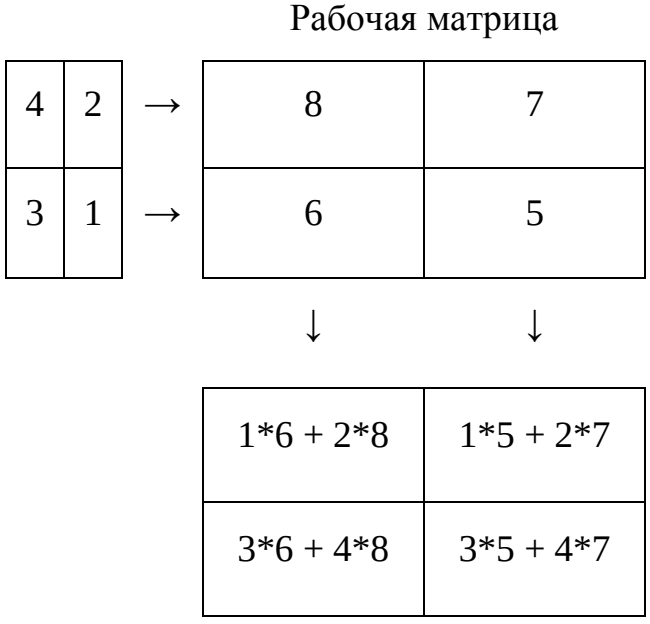

Рис. 14. Умножения матриц с помощью команды взвешенное суммирование

На рисунке 15 показано состояния регистра-контейнера wfifo после выполнения команды rep 2 wfifo = [ar1++];

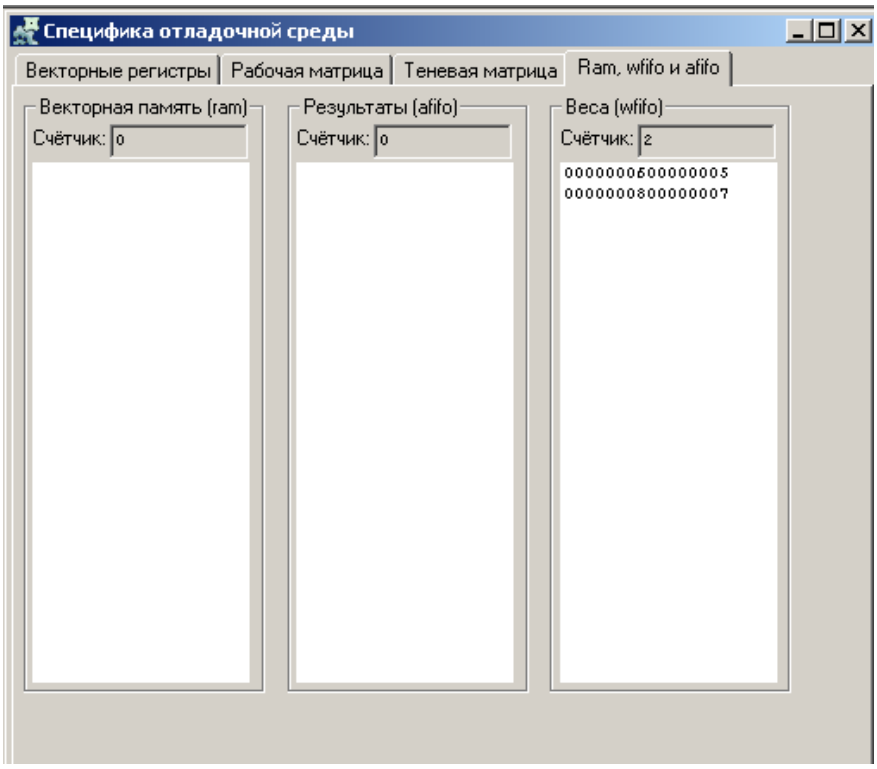

Рис. 15. Состояние регистра-контейнера wfifo

На рисунке 16 показано состояния теневой матрицы после выполнения команды ftw;

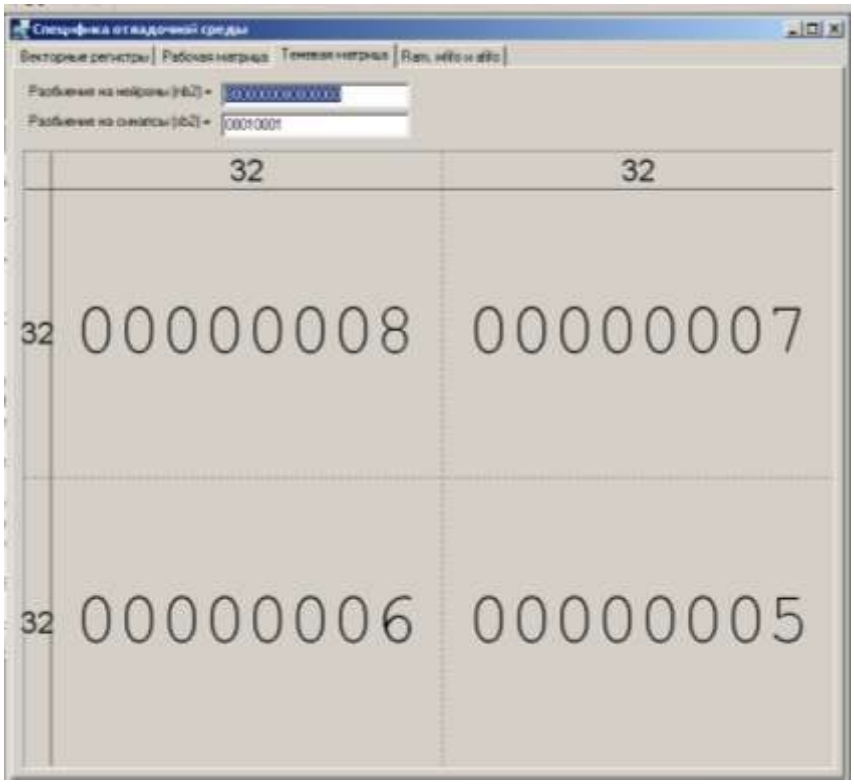

Рис. 16. Состояние теневой матрицы

Команда wtw; выполняет копирование содержимого теневой матрицы в рабочую. Команда rep 2 data = [ar0++] with vsum, data, 0; выполняет умножение

матриц, в результате регистр-контейнер afifo будет содержать следующий результат (см. рис. 17).

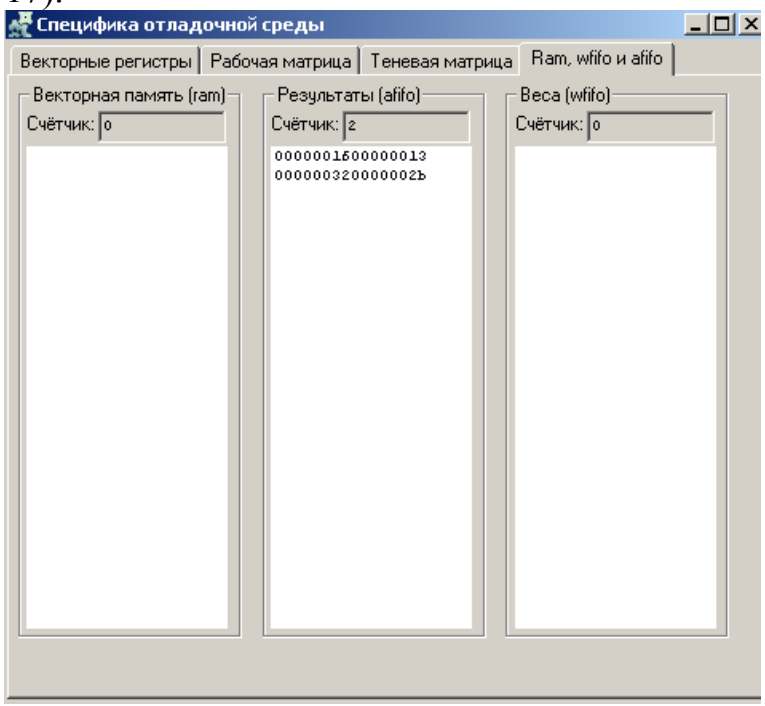

Рис. 17. Состояние регистра-контейнера afifo

После выполнения команды rep 2 [ar2++] = afifo; память будет содержать следующий результат, показанный на рисунке 18.

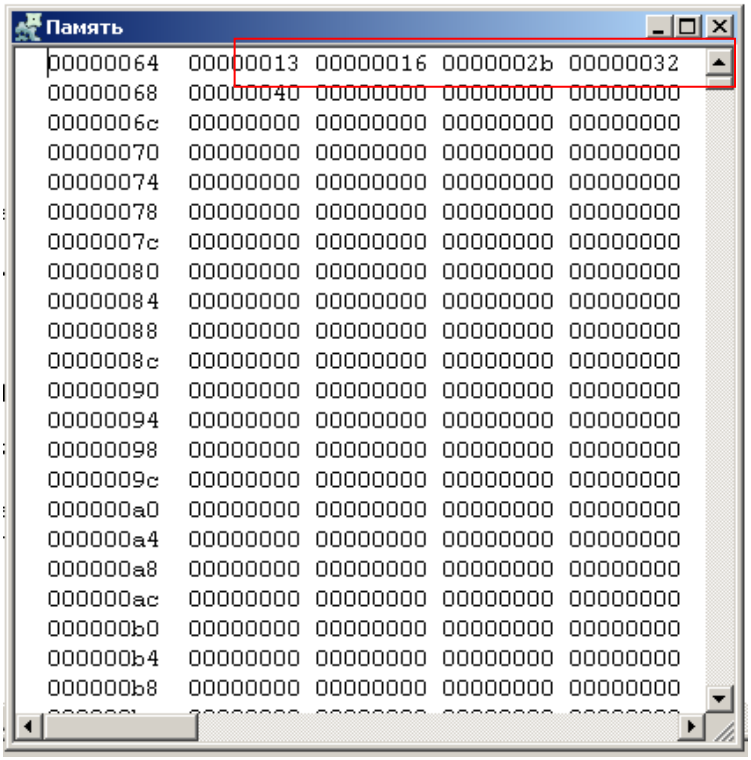

Рис. 18. Результат умножения матриц

5. Порядок выполнения работы

1. Изучить систему команд процессора NeuroMatrix NM6403 (Описание системы команд нейропроцессора находится в файле AsmOver(Rus).pdf. Файл необходимо взять у преподавателя).

2. Составить программы, в соответствии с вариантом задания, указанным преподавателем, в двух вариантах – с использованием команд скалярного процессора и с использованием команд векторного процессора.

3. Загрузить в программу-эмулятор подготовленную программы и произвести их отладку.

4. Продемонстрировать работу программ преподавателю на нескольких контрольных примерах.

6. Содержание отчёта

Отчёт должен содержать:

1) титульный лист;
2) наименование работы и цель исследований;
3) структурную схему скалярного и векторного процессора;
4) блок-схему программы, подготовленной в рамках домашнего задания;
5) текст программы.

7. Контрольные вопросы

1. Какие основные блоки входят в состав процессора NeuroMatrix?
2. Какие функции может выполнять скалярный процессор в составе NM6403?
3. Какие функции может выполнять векторный процессор в составе NM6403?
4. Опишите основные блоки векторного процессора в составе NM6403.
5. Чем отличаются команды скалярного и векторного процессоров?
6. Какие средства в составе NM6403 позволяют обеспечить работу с массивами данных?
7. Где можно размещать текущие адреса массивов при программировании?
8. Каким образом выполняется загрузка и отладка программы в эмуляторе?

8. Варианты заданий

1. Сравнить элементы трех массивов ($L_1=L_2=L_3$, L – длина массива) и подсчитать число совпадающих элементов с одинаковыми индексами.

2. Определить среднее арифметическое значений двумерного массива, выполнить умножение элементов массива на их среднее арифметическое.

3. Обработать кадр изображения размером n*m. Значение яркости каждого пикселя задается в параллельном p-разрядном коде. Разработать программу, преобразующую изображение в соответствии с заданной функцией f(i,j) (где f(i,j) – значение яркости пикселя с координатами (i,j), i=0..n-1, j=0..m-1, n=1..256, m=1..256). Варианты

a) f(i,j)=(f(i–1,j)+f(i+1,j))/2, p=1..8.

b) f(i,j)=(f(i–1,j)+f(i+1,j))/2, p=1..8.

c) f(i,j)=(f(i,j–1)+f(i,j+1))/2, p=1.

d) f(i,j)=(f(i,j–1)+f(i,j+1))/2, p=2.

e) f(i,j)=(f(i-1,j-1)+f(i-1,j)+f(i-1,j+1)+f(i,j-1)+f(i,j+1)+f(i+1,j-1)+f(i+1,j)+f(i+1,j+1))/8, p=1.

f) f(i,j)=(f(i-1,j-1)+f(i-1,j)+f(i-1,j+1)+f(i,j-1)+f(i,j+1)+f(i+1,j-1)+f(i+1,j)+f(i+1,j+1))/8, p = 2.

g) f(i,j)=f(i+1,j)+f(i-1,j)+f(i,j+1)+f(i,j-1)–4*f(i,j), p=1.

h) f(i,j)=f(i+1,j)+f(i-1,j)+f(i,j+1)+f(i,j-1)–4*f(i,j), p=2.

i) f(i,j)=f(i+1,j)+f(i–1,j)–2*f(i, j), p=1..8.

j) f(i,j)=f(i,j+1)+f(i,j–1)–2*f(i,j), p=2.

Библиографический список

1. Архитектура NeuroMatrix ® NM6403. Руководство пользователя. – НТЦ Модуль

2. Описание языка ассемблера NeuroMatrix ® NM6403. – НТЦ Модуль.

3. Базовое программное обеспечение NM6403. Справочное руководство. – НТЦ Модуль.

4. Многоцелевой подключаемый отладчик. Руководство пользователя. – НТЦ Модуль.