

**Параллельный
перепрограммируемый вычислитель.
Возможность применения для обработки
изображений и программное обеспечение.**

*Аряшев С.И., Бобков С.Г., Сидоров Е.А.,
Юдин И.В.*

- 1 Введение.
- 2 Алгоритмы обработки изображений.
- 3 Структурная схема и краткое описание вычислителя.
- 4 Управляющий процессор.
 - 4.1 Структура и краткое описание процессора.
 - 4.2 Система команд.
- 5 Программное обеспечение
 - 5.1 Среда разработки.
 - 5.2 Транслятор.
 - 5.3 Трассировщик.
 - 5.4 Пример программы.
- 6 Заключение.
- Литература.

1. Введение

Стремительное развитие информационных технологий интенсивно стимулирует исследования и разработку методов и аппаратных средств для таких областей, как обработка изображений, распознавание образов и моделирование нейронных сетей. Отличительной чертой многих задач, свойственных данным областям, является большое количество простых базовых операций и возможность параллельного выполнения этих операций. Так как решение задач такого класса на стандартных компьютерах требует больших временных затрат, для решения этой проблемы предлагается множество вариантов параллельных высокопроизводительных вычислительных систем: от универсальных многопроцессорных систем до специализированных ускорительных плат, предназначенных для решения конкретной задачи.

В разработанном вычислителе обработка информации распределяется между базовыми вычислительными элементами, которые благодаря перепрограммируемости могут не только работать с операндами переменной разрядности, но и менять пути прохождения данных внутри микросхемы.

Преимуществом перепрограммируемых систем при решении задач, связанных с обработкой изображений и сигналов и моделирования нейронных сетей, является возможность настройки системы на решаемую задачу. Возможность изменения архитектуры позволяет расширить класс решаемых задач без существенного увеличения стоимости устройства.

2. Алгоритмы обработки изображений

Вычислитель может использоваться в системах обработки изображений, работающих в стандартном режиме кадровой развертки порядка нескольких десятков кадров в секунду и выше, где время отклика должно составлять несколько миллисекунд. В таких задачах требуется выполнять большое количество однотипных базовых операций над большим массивом данных, обычно это матричные и векторные операции.

Изображение поступает в цифровой форме с приемника в виде двумерной матрицы, элементами которой являются значения относительных яркостей соответствующих точек

изображения. Значения яркости могут лежать в диапазоне от 0 до 255, т. е. укладываться в восемь двоичных разрядов или один байт. В общем случае обычно необходимо выяснить, есть ли на изображении какой-либо из объектов, представляющих интерес, и если есть, то какой именно объект, и в каком месте он находится. В качестве частного случая общей задачи может стоять задача улучшения качества изображения или подчеркивания определенных признаков.

Условия получения изображения в рабочем режиме могут сильно отличаться от условий, при которых были получены эталонные объекты. Изображение может быть слабоосвещенным, зашумленным, низкоконтрастным, размытым и т. д. Для избавления от искажений такого рода обычно применяются различные виды пространственной фильтрации. Пространственные процессы используют группы элементов изображения для извлечения информации об изображении. Группа элементов изображения, используемых в пространственных процессах, называется областью примыкания. Преобразуемые точки (точки, старое значение которых заменено на новое как результат алгоритмических вычислений) находятся в центре области примыкания.

Рассмотрение набора точек в непосредственной близости от преобразуемой точки дает информацию о локальном распределении яркости, которая используется большинством пространственных процессов. Информация о распределении яркости обычно выражается в терминах пространственной частоты. Пространственная частота определяется как скорость изменения яркости элементов изображения или интенсивности, деленная на расстояние, на котором происходит это изменение. Пространственная частота имеет компоненты как горизонтального, так и вертикального направлений в изображении. Изображение с высокой пространственной частотой содержит резкие близкорасположенные изменения в значениях соседних элементов. Изображения с низкой пространственной частотой содержат большие области постоянных или медленно меняющихся значений элементов изображения. Информация о пространственной частоте позволяет использовать пространственные процессы в качестве фильтров путем удаления или усиления компонентов определенной частоты, обнаруженных в изображении.

В простейшем случае линейной фильтрации необходимо вычислить свертку исходного сигнала, в данном случае – двумерного, с импульсной характеристикой соответствующего фильтра. Свертка - это алгоритм очень широкого применения, который можно использовать как для предобработки изображения, так и для распознавания и идентификации объектов. Пусть изображение задается двумерной матрицей яркостей A' , а импульсная характеристика матрицей K . Математически свертку матрицы A с ядром K можно определить следующей формулой:

$$b_{ij} = \sum_{n=0}^{N_2-M_2-1} \sum_{m=0}^{M_2-1} a_{i+n, j+m} \cdot k_{nm} , \quad 0 \leq i \leq N_1-1 , \quad 0 \leq j \leq M_1-1 , \quad (1)$$

где $N_2 \times M_2$ - размер матрицы ядра свертки. Размер матрицы A равен $(N_1+N_2-1) \times (M_1+M_2-1)$, где $N_1 \times M_1$ - размер исходной матрицы A' . Матрица A получается из исходной путем добавления элементов на краях матрицы по некоторому правилу с тем, чтобы привести ее к необходимому размеру. Обычно исходная матрица на краях дополняется нулями на половину ширины матрицы K влево и вправо и соответственно на половину высоты вверх и на столько же вниз. Тогда размер полученной матрицы B будет таким же, как и у матрицы A' (рис. 1).

Свертку можно вычислять непосредственно "пробеганием" одной матрицы по другой, как уже было показано выше. Другой широко практикуемый метод - это выполнение циклической свертки с использованием интегральных фурье-подобных преобразований. Используя данный метод можно строить так называемые *согласованные фильтры*. Согласованный фильтр в изображении усиливает лишь те частоты, которые присутствуют в изображении объекта, на который он настроен, остальные частоты ослабляются, т.е. частотная характеристика согласованного фильтра повторяет частотную характеристику полезного сигнала. Результатом действия такого фильтра должно стать выделение искомого объекта на изображении и отсечение фоновых сигналов.

В основе вычисления свертки с использованием интегральных преобразований типа Фурье лежит теорема о свертке :

$$K \otimes M = F^{-1} (F(K) \times F(M)) , \quad (2)$$

где символами F и F^{-1} обозначены соответственно прямое и обратное фурье-подобное интегральное преобразование. В выражении справа подразумевается почленное произведение элементов матриц-образов. Матрицы K и M должны иметь одинаковую размерность, что достигается путем добавления нулевых элементов к матрице, имеющей меньший размер.

Алгоритм свертки с использованием интегральных преобразований может использоваться как на этапе предобработки, так и на более поздних этапах обнаружения и идентификации [5].

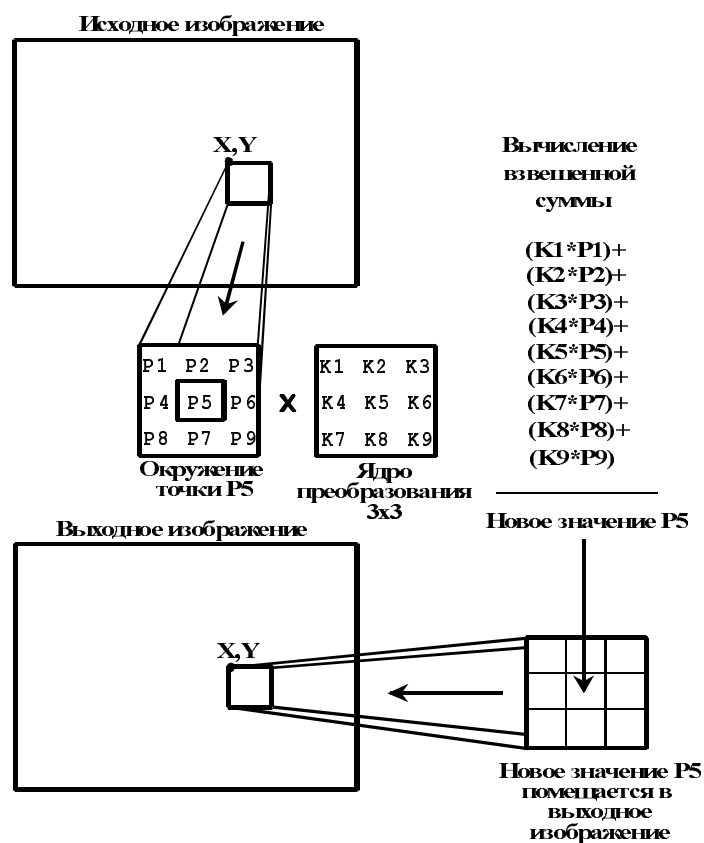


Рис.1. Операция свертки

Для избавления от точечных и импульсных помех широко используется метод медианной (усредненной) фильтрации [7]. Данный алгоритм выстраивает элементы в области примыкания в возрастающем порядке и отбирает средние значения как новые значения рассматриваемых элементов. Результатом усредненного фильтрования является то, что любой случайный шум, содержащийся в изображении, будет эффективно устранен. Это происходит потому, что любое случайное резкое изменение в интенсивности элемента в пределах области примыкания будет сортироваться, т.е. оно будет помещаться либо на вершину, либо на нижнюю часть сортированных значений области примыкания и не будет учитываться, так как для нового значения элементов всегда отбирается усредненное значение (рис.3).

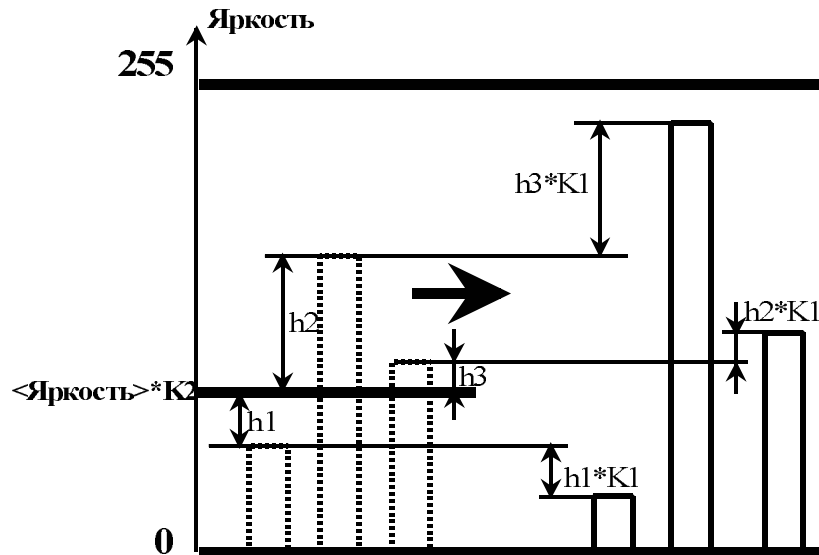


Рис.2. Алгоритм повышения контрастности

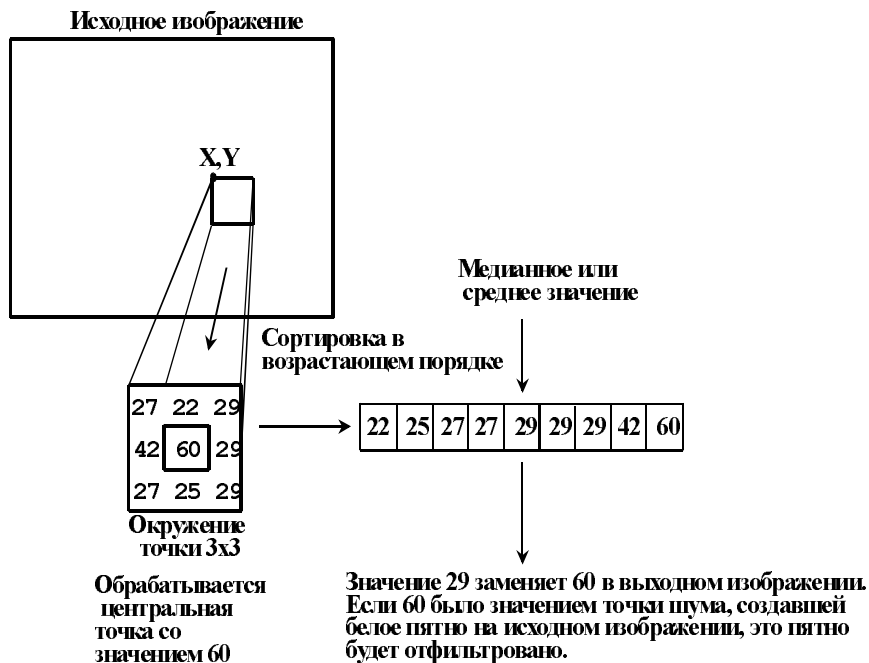


Рис.3. Медианный фильтр

Для улучшения визуального восприятия изображения кроме избавления от помех и искажений иногда необходимо также усилить контрастность объектов: темные объекты сделать еще более темными, а светлые - более светлыми. Изображения с высоким контрастом используют весь диапазон яркости, соответствующей полутоновым изображениям. Для низкоконтрастных изображений значения яркостей всех точек лежат лишь в некотором узком диапазоне, т. е. близки друг к другу. Существуют различные методы повышения контрастности изображения. В используемом нами методе для каждой точки изображения вычисляется среднее значение яркости изображения в прилегающей области, и затем значение яркости данной точки переопределяется по формуле:

$$P_{new} = P_{old} + K_1 \cdot (P_{old} - K_2 \cdot \langle Average \rangle) , \quad (3)$$

где P_{new} , P_{old} – новое и старое значения яркости точки, K_2 определяет порог в долях от значения средней яркости, от которого будет вестись пересчет, K_1 определяет значение, которое необходимо прибавить к текущей яркости, в долях разности предыдущего значения яркости и порога (рис.2). Если новое значение яркости ниже нуля, то будем присваивать ему

значение ноль, если выше 255, то присваивается значение 255. В простейшем случае $K_1=K_2=1$, тогда $P_{new} = 2 \cdot P_{old} - \langle Average \rangle$, что соответствует увеличению текущего значения яркости точки на величину превышения им среднего значения. Если текущее значение яркости ниже среднего, то яркость точки уменьшится на величину превышения среднего значения над значением яркости точки. Другими словами, разность между яркостью точки и локальной средней яркостью удваивается. Если определить α и β как $\alpha = 1 + K_1$ и $\beta = K_1 \cdot K_2$, то формулу (3) можно переписать в виде

$$P_{new} = \alpha \cdot P_{old} - \beta \cdot \langle Average \rangle . (4)$$

В задачах машинного зрения, распознавания образов и т.д. очень часто в качестве рабочей информации о характеристиках объектов и изображения служит не сама яркость точек, а градиенты яркости в различных областях изображения. Полезной информацией здесь является информация о скорости и направлении перепадов яркости в изображении, на основе которой формируются наборы признаков, используемые на последующих этапах распознавания и идентификации. Такую информацию в ряде случаев можно получить путем подчеркивания областей, в которых величина изменения яркости выше некоторой пороговой, и удаления из изображения всех прочих участков, т. е. фактически используя алгоритмы определения краев. Меняя порог перепада интенсивности можно строить иерархические наборы признаков, т. е. выделять перепады яркости на разных уровнях размытости. Выделение краев можно проводить, например, выполняя операцию свертки с ядрами с соответствующими коэффициентами и размерами. Существуют также и нелинейные алгоритмы определения края.

Одним из наиболее известных алгоритмов вычисления градиентов интенсивности является алгоритм Собеля [7]. В данном методе вычисляются 2 различные свертки с ядрами:

$$\text{ядро X} \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \text{ядро Y} \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

и, исходя из этих сверток, вычисляется величина и направление перепадов:

$$\text{Величина} = \sqrt{X^2 + Y^2}, \text{ направление} = \arctg\left(\frac{Y}{X}\right), \quad (5)$$

Полученные таким образом матрицы можно далее использовать в задачах распознавания и идентификации объектов.

За этапом предварительной обработки изображения следует непосредственно этап поиска и идентификации объектов. Иногда эти этапы объединены, и одна операция выполняет сразу несколько функций – и предобработки, и обнаружения, и распознавания. По-видимому, наиболее эффективный подход к построению алгоритма распознавания – это разбиение общей задачи обнаружения и идентификации на ряд последовательных подзадач. Результатом обработки на каждом этапе должна являться все более упорядоченная информация об изображении и свойствах объектов, присутствующих на нем.

Самый общий метод, который применим в подавляющем большинстве случаев – это прямое поточечное сравнение яркостей эталонов и участков изображения соответствующего размера. Эталон как бы “скользит” по изображению с шагом в одну точку, и в каждом положении подсчитывается сумма абсолютных значений разностей яркостей соответствующих точек эталона и изображения. Если сумма ниже определенного порога, то считается, что в данном месте обнаружен искомый объект. Вместо операции разности яркостей могут использоваться и другие операции – например, умножение, как в свертке, либо какие-то логические функции. Главным недостатком данного метода является необходимость очень большого числа вычислений, а следовательно – большие затраты времени на обработку одного изображения. При повороте объекта, его увеличении или уменьшении необходимо формировать новые эталоны и производить сравнение также и с этими эталонами, что приводит к многократному увеличению количества операций. Но если ориентация и размеры объектов на изображении предполагаются всегда строго фиксированными, и размеры изображения и искомого объекта соизмеримы, данный метод может эффективно применяться. Результат обработки может быть представлен в виде матрицы, каждый элемент которой есть итог сравнения точек эталона и изображения в каждой позиции, т.е. в двоичном виде определяться как: 0, если порог превышен, и 1, если порог не превышен (если 0 – “объект не найден” и если 1 – “объект найден”).

В случае, если искомыми объектами являются сплошные фигуры, то поиск объектов можно начинать с поиска всех объектов с площадью, соответствующей площади объекта, т.е. провести предварительную сегментацию изображения с учетом неоднородностей. После этого, для дальнейшей обработки можно применить, например, метод прямого сравнения, описанный выше. Количество операций при этом существенно уменьшается. Алгоритм выделения участков с пятнами яркости заданной площади заключается в следующем. Пусть ширина искомого объекта равна n точек, а высота m точек. Тогда все изображение разбивается на участки шириной $2n$ и высотой $2m$ точек, данные участки накладываются друг на друга с шагом n по горизонтали и m по вертикали. Такое разбиение проводится для того, чтобы для любого объекта размером $n \times m$ нашлся бы как минимум один участок $2n \times 2m$, в который данный объект укладывался бы полностью. Для каждого такого участка находится средняя яркость, и затем подсчитывается число точек, яркость которых превышает среднюю более, чем на некоторое заданное число. Таким образом подсчитывается площадь фигуры, яркость точек которой выше средней. Затем производится сравнение полученной площади с площадью объекта, и если она приблизительно равна или больше площади объекта, то данный участок помечается как “подозрительный” на существование объекта в нем – формируется гипотеза. Далее этот участок можно исследовать другими методами.

Наиболее распространенными методами распознавания образов являются методы, в которых производится предварительное отображение матриц интенсивности эталонных объектов и исследуемых изображений в некоторое пространство образов или признаков. Фактическое распознавание и идентификация проводятся уже в этом пространстве. Понятие “пространство образов” здесь определяется достаточно широко и не строго. Это может быть и просто набор некоторых интегральных характеристик изображения (средняя яркость, координаты “центра тяжести” фигуры, дисперсия, моменты и т. д.) - фазовое пространство признаков, и пространство фурье-образов, и определяться еще каким-либо иным путем. Выбор способа отображения обычно обуславливается типом задачи, требованиями к различным видам инвариантности, свойствами распознаваемых объектов и т.д. Например, для идентификации объекта, который на изображении может появляться под

произвольным углом к осям, можно использовать преобразования, инвариантные по отношению к операциям поворота, и т.д.

В настоящее время реализован ряд алгоритмов векторно-матричных и матрично-матричных вычислений, а также реализованы специальные алгоритмы работы с двумерными изображениями.

В таблице 1 приведена оценка производительности вычислителя при выполнении некоторых алгоритмов работы с двумерными изображениями. Программы для тестирования времени работы ЭВМ на базе процессоров Pentium и Sparc были написаны на языке Си, измерялось “чистое” время вычислений без учета инициализаций, формирования исходных массивов в памяти и т.д. Возможно, при использовании специальных ассемблерных библиотек, можно было бы оптимизировать данные алгоритмы, уменьшив тем самым временные затраты. Время указано в секундах.

Таблица 1.

Название алгоритма	Pentium -100, с	PentiumII-333, с	Ultra SPARC, с	Вычислитель, с
Свертка с ядром 4×4 ³⁾	0.65	0.11	0.76	0.02
Медианный Фильтр	1.97	0.49	0.75	0.001
Повышение контрастности	0.51	0.13	1.31	0.004
Прямое поточечное сравнение с маской 32×32 ⁴⁾	43.78	7.14	58.89	0.142
Поиск локальных неоднородностей 32×32	0.120	0.028	0.146	0.032
Умножение матрицы на матрицу	8.61	0.60	12.31	0.011

Примечания:

1) Оценки приведены для:

Pentium-100 при частоте 100 МГц, объем ОЗУ 16 Мбайт;

Pentium-333 при частоте 330 МГц, объем ОЗУ 128 Мбайт;

UltraSPARC при частоте 200 МГц, объем ОЗУ 64 Мбайт;

Вычислителя при частоте 33 МГц.

2) Размер изображения 256x256 точек, диапазон яркости лежит в пределах от 0 до 255.

3) Одновременно можно производить свертку с 6 фильтрами - один фильтр на каждый ВЭ, производительность улучшается тогда в 6 раз.

4) Данные приведены для поиска одного объекта, одновременно может вестись поиск $2*6=12$ объектов, что приведет к повышению производительности в 12 раз.

3. Структурная схема и краткое описание вычислителя

Параллельный перепрограммируемый вычислитель (ППВ) разработан в стандарте VME и реализован на базе перепрограммируемых микросхем семейства 10K фирмы Altera. Вычислитель предназначается для работы в качестве аппаратного ускорителя и является ведомым устройством на шине VME. Он всегда должен включаться в систему как подчиненное устройство основной управляющей ЭВМ (host-машины) с универсальным процессором. Тактовая частота вычислителя 33 МГц.

Разработанное устройство используется для построения систем распознавания образов на основе обработки телевизионной, тепловизионной и другой информации, а также систем, основанных на реализации алгоритмов с пороговыми функциями и простейшими арифметическими операциями и позволяет добиться значительной скорости вычислений при решении задач следующих типов:

- предобработка изображения;
- реализация разработанной нейронной сети на формально-логических нейронах;
- реализация базовых нейронных операций (умножение, сложение с накоплением, пороговые

функции) для нейронов с размером веса и входного вектора от двух до восьми бит;

- другие типы задач связанные с обработкой изображения и принятием решений.

В процессе разработки проводилось изучение возможности аппаратной реализации и применения в поставленной задаче как нейронных сетей, так и классических способов обработки изображения. Исследовалась возможность представления алгоритмов обработки изображения и нейронных сетей на уровне векторно-матричной и матрично-матричной математики с добавлением пороговых функций.

Структурная схема вычислителя представлена на рис. 4.

Вычислитель состоит из следующих функциональных блоков:

- схема управления (Сх Упр);
- базовые вычислительные элементы (БВЭ1-БВЭ6);
- контроллер внешней шины (Контроллер E-bus);
- контроллер системной шины (Контроллер VME);
- два массива статической памяти (ОЗУ0, ОЗУ1);
- блок высокоскоростных приемников/передатчиков.

Схема управления используется для управления БВЭ и потоками данных в вычислителе и представляет собой простейший RISC процессор. Структура и набор команд процессора могут изменяться в зависимости от типа решаемой задачи.

БВЭ используются для выполнения простейших арифметических операций типа суммирования, вычитания, умножения и вычисления пороговых функций. Так как БВЭ реализованы на перепрограммируемых микросхемах, их архитектура может изменяться. Архитектура БВЭ для различных алгоритмов может отличаться, но обычно легко реализуются путем комбинации библиотечных функций, компиляции их при помощи САПР (типа MaxPlus) и загрузки файла конфигурации в выбранный БВЭ.

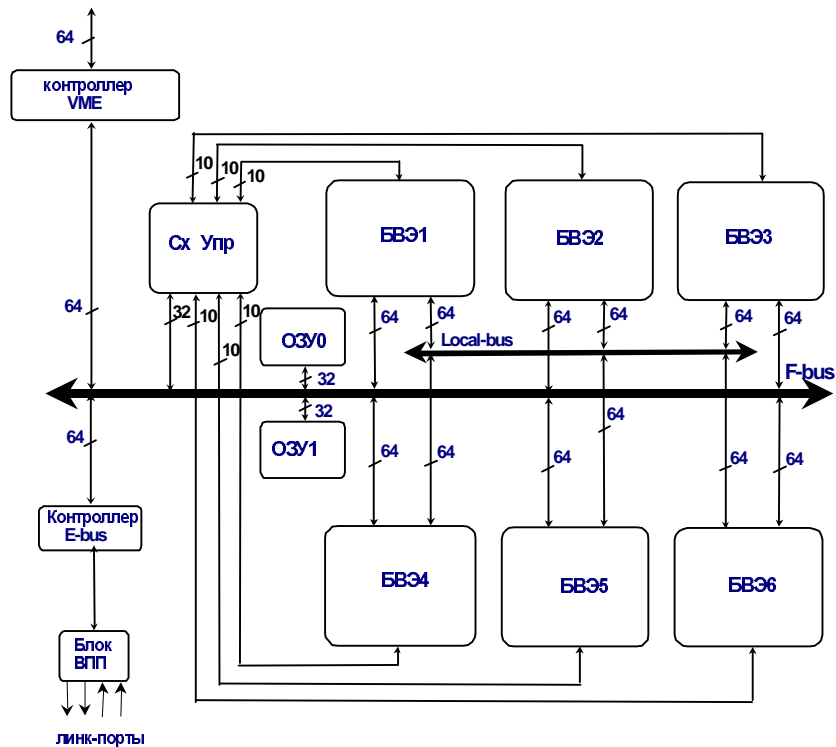


Рис4. Структурная схема вычислителя

Например, БВЭ для перемножения векторов и матриц содержит: набор регистров и 4 вычислительных элемента, состоящих из 8-разрядного конвейерного умножителя, 24-разрядного сумматора с накоплением результата, схемы логического умножения, компаратора. Наличие таких ресурсов БВЭ позволяет аппаратно выполнять векторно-матричные и матрично-матричные операции, которые являются базовыми при обработке изображения и реализации нейронных сетей. Выбор необходимой арифметической операции проводимой над входными данными и путь прохождения данных через любой вычислительный элемент определяется установкой схемой управления соответствующей команды, набора битов на шине. Внутренние регистры БВЭ используются для хранения значений входных коэффициентов, порогов и результатов вычислений.

БВЭ для реализации различного типа фильтров, в дополнение к регистрам и 4 вычислительным элементам, содержит внутреннюю память коэффициентов.

Два массива локальной статической памяти собраны из 8 микросхем статической памяти емкостью 0,5 Мбайт, имеют размер 4Мбайт и организованы как массив 512К 8-байтовых слов. Массивы памяти связаны со схемой управления отдельными адресными шинами и могут функционировать независимо друг от друга. Память предназначена для хранения общих коэффициентов, а также промежуточных результатов вычислений или окончательных результатов, подготовленных к передаче через контроллер системной шины в центральный процессор или через контроллер E-bus на линк-порты.

Связь нескольких вычислителей между собой или вычислителя с устройством оцифровки изображения, при наличии у устройства оцифровки соответствующего интерфейса, осуществляется посредством последовательного канала приемников/передатчиков HOTLink фирмы CYPRESS. Управление передачей данных выполняет контроллер внешней шины, который представляет из себя набор 4-х стандартных FIFO и регистров управления и данных.

Контроллер шины VME выполняет функцию интерфейса с центральным процессором и является стандартным устройством [8].

С точки зрения программиста вычислитель можно представить как RISC-процессор (схема управления или управляющий процессор) и шесть векторных процессоров (вычислительных элементов), обрабатывающих SIMD-команды (одна команда для многих данных) - рис. 5.

Большое количество шин данных, возможность одновременной работы всех БВЭ и выполнение арифметических операций умножения и сложения за один такт позволяет эффективно распараллеливать процесс обработки информации.

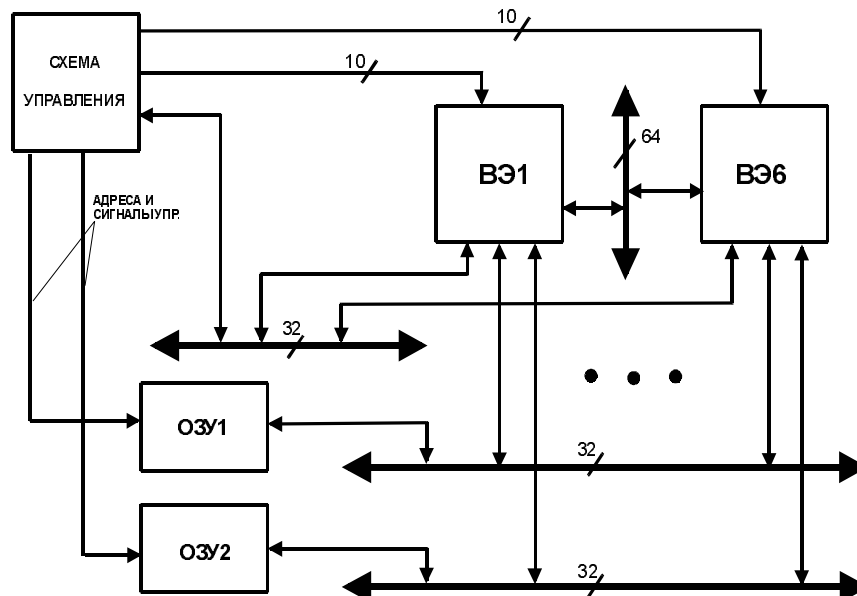


Рис.5. Схема вычислителя.

4 Управляющий процессор

4.1 Структура и краткое описание процессора

Структурная схема управляющего процессора представлена на рис. 6.

Схема управления состоит из следующих компонент:

- внутренней памяти программ;
- дешифратора команд;
- регистров данных и регистра констант;
- регистров адреса и схем инкремента;
- регистров ВЭ
- АЛУ
- регистра состояния и устройства управления
- счетчика команд и указателя стека
- контроллера внешнего ОЗУ.

Выполняемая программа хранится во внутренней памяти программ, выполненной на встроенных в микросхему блоках памяти. В настоящее время память программ имеет размер 512 32-разрядных слов. В дальнейшем, используя более мощные микросхемы, размер памяти программ может быть увеличен до 2048 32-разрядных слов.

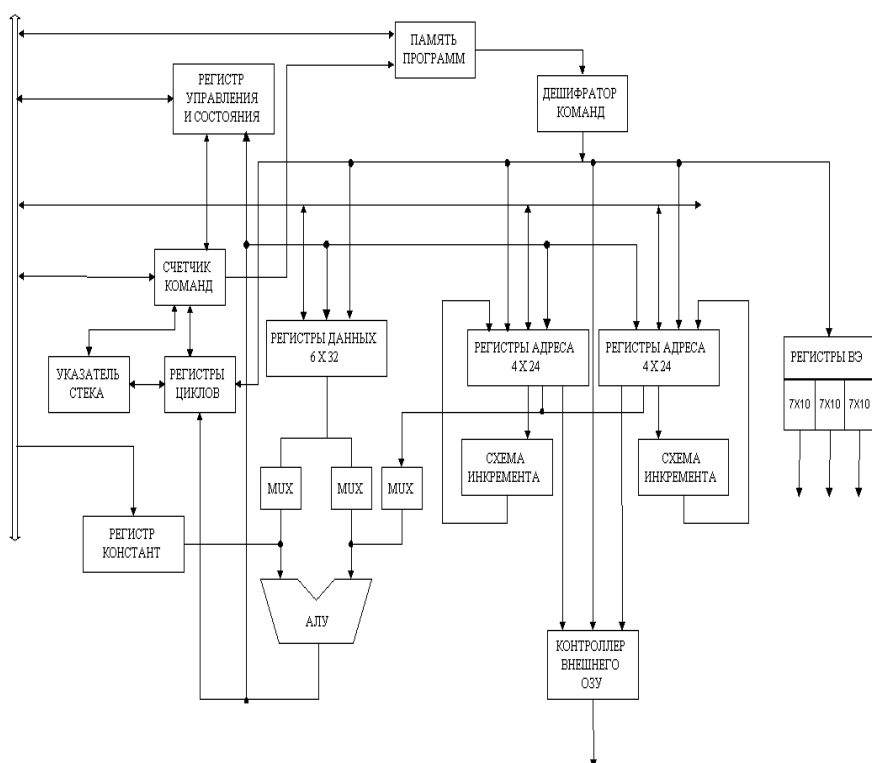


Рис. 6. Структурная схема управляющего процессора

Загрузка памяти программ может выполняться во время загрузки микросхемы, а также предусмотрена возможность загрузки программы во время работы через шину VME.

Память программ представляет собой набор подпрограмм (микропрограмм), каждая из которых реализует определенный алгоритм обработки информации. К примеру, 1-я микропрограмма, хранимая с 0-го адреса, выполняет умножение вектора на матрицу, 2-я микропрограмма, хранимая с 20-го адреса, умножает матрицу на матрицу, 3-я реализует сеть

Хопфилда, 4-я транспонирует матрицу и т.д. Параметры микропрограмм, размеры векторов или матриц, количество проходов, шаг инкрементации адреса задаются путем загрузки регистров с шины VME.

Устройство управления и регистр состояния управляют запуском, остановкой и выбором режима функционирования процессора. Регистр состояния адресует как младший регистр данных. Управление процессором заключается в установке, сбросе и проверке отдельных битов регистра состояния.

Регистр состояния состоит из трех блоков:

- блок состояния процессора (бит запуска/останова, бит загрузки счетчика команд, биты прерываний, биты порядка инкрементации адресных регистров)
- блок, содержащий значение счетчика команд
- блок флагов состояния АЛУ.

Выполнение процессором команды происходит за три такта. Наличие в процессоре трехуровневого конвейера позволяет каждый такт загружать новую команду. Подобно RISC процессору, для равномерного заполнения конвейера все команды выполняются за одинаковое число тактов.

Уровни конвейера:

- считывание и декодирование команды
- выборка данных из регистра и их загрузка в АЛУ
- загрузка результата в регистр

Процессор содержит 8 24-разрядных регистров адреса RA0-RA7, 21 10-разрядных регистров вычислительных элементов, 6 32-разрядных регистров данных RD2-RD7, 8 8-разрядных регистров констант, регистры ускоренного ветвления. Регистр состояния можно рассматривать как самый младший регистр данных RD0. Регистр RD1 разделен на два 16-ти разрядных блока и выделен под счетчики команд вычислительных элементов и ускоренного ветвления.

Если счетчик команд ВЭ или счетчик ускоренного ветвления не загружен (в регистре RD1 содержится 0), то команда ВЭ или команда loop выполняется один раз, после чего счетчик адреса инкрементируется. Младший блок регистра RD1 соответствует счетчику команд ВЭ, а старший блок – счетчику ускоренного ветвления. Если в каком-то из блоков регистра RD1 содержится число, не равное 0, то после выполнения соответствующей этому блоку команды (команды ВЭ или ускоренного ветвления) данный блок регистра RD1

декрементируется, а содержимое счетчика команд не меняется. Команда повторяется до тех пор, пока соответствующий блок регистра RD1 не обнулится. Так как команда ВЭ - базовая, то наличие такого регистра заметно увеличивает скорость выполнения программ.

Шесть оставшихся регистров данных используются как регистры общего назначения.

Регистры ВЭ используются для хранения наиболее часто используемых 10-разрядных команд ВЭ. Общее число команд ВЭ может быть значительно большим, но обычно каждая отдельная микропрограмма не требует более семи команд и при ее запуске необходимые команды загружаются в регистры. Регистры разбиты на три группы по семь. Каждая группа закреплена за двумя ВЭ.

Регистры адреса используются для хранения и обработки адресов данных, хранимых в локальной памяти. Регистры разбиты на две группы по 4 регистра в каждой. Группа соответствует банку памяти. После выполнения команды ВЭ значение адресных регистров может быть увеличено на число, хранимое в регистрах инкрементации адреса.

Необходимость инкрементации определяется установкой соответствующих битов в слове команды, причем необходимость инкрементации для каждой группы устанавливается отдельно.

Все регистры, кроме регистра состояния, могут быть загружены во время выполнения программы. Регистры ВЭ используются только для хранения команды и над ними не могут производиться никакие операции.

Регистры адреса и данных также могут быть загружены или считаны центральным процессором по шине VME. Эта возможность введена для гибкости работы с микропрограммами и облегчения отладки, так как отладка процессора затруднена в связи с отсутствием соответствующего программного обеспечения.

Так как схема управления разрабатывалась для управления потоками данных, АЛУ процессора максимально упрощено. В соответствии с требованиями реализованных на сегодняшний день алгоритмов АЛУ выполняет знаковое и беззнаковое сложение и вычитание, сдвиговые и логические операции. Обрабатываемые операнды извлекаются из любых двух регистров-источников: или оба из регистров данных, или один из регистров данных, а другой из регистров адреса.

Результат вычислений может быть занесен в любой регистр.

По результату выполнения команды выставляются три флага состояния: нулевой или отрицательный результат и переполнение.

Контроллер внешнего ОЗУ используется для управления локальной памятью и представляет собой два отдельных контроллера статической памяти, с временем доступа не менее 20 нс.

Схема управления соединена отдельными линиями связи со всеми БВЭ. Особенностью схемы управления перепрограммируемого вычислителя для систем обработки информации является наличие рабочей команды, управляющей шестью базовыми вычислительными элементами. Команда позволяет одновременно, за один такт, задавать различные режимы функционирования шести базовым вычислительным элементам и инкрементировать адреса обоих массивов памяти на любое число от 0 до 255, хранимое в регистрах инкремента, причем каждому массиву соответствует свой регистр. Команда может повторяться любое количество раз в соответствии со значением, хранимым в специальном регистре. Это позволяет выполнять основную команду без потерь на организацию циклов и переходов.

Рабочая команда позволяет одновременно запускать оба контроллера локальной памяти, инкрементировать адресные регистры на требуемое значение, выставлять на адресные шины адреса из соответствующих регистров адреса, выставлять на шины управления БВЭ команды из соответствующих регистров БВЭ. Кроме того, рабочая команда осуществляет организацию обмена данными между контроллером внешней шины и локальной памятью.

4.2 Система команд

Регистры схемы управления делятся на три категории: регистры данных (RD), регистры адреса (RA) и байтовые регистры констант (RM).

На данный момент имеется 9 регистров данных, 8 регистров адреса и 8 байтовых регистров констант. Некоторые регистры данных являются специализированными. Это RD0, RD1 и RD9.

RD0 – регистр контроля и состояния.

RD1 – содержит количество повторений SIMD-команды.
Нулевое

значение соответствует 1.

RD9 – является декрементным счетчиком цикла команды loop.

Управляющий процессор имеет конвейер на три команды. Внутренняя память для хранения программ имеет объем два килобайта. Каждая команда занимает в оперативной памяти одно длинное слово (32 разряда), помещается в конвейер управляющего процессора за один такт и выполняется за два такта. Значения измененных регистров, соответственно, доступны после третьего такта (для RD9 – после второго такта).

Для удобства работы с циклами введена команда loop, соответствующая команде SOB ассемблера PDP-11.

Возможно ввести внутренний стек на четыре значения для регистра RD9, что позволит использовать один регистр счетчика циклов для вложенных циклов.

Виды команд управляющего процессора:

- Команды загрузки 24-разрядных чисел в регистры данных и адреса;
- Команды загрузки регистров вычислительного элемента;
- Команда пересылки данных (RD→RD, RA→RA);
- Команды сложения и вычитания регистров (RD из RD, RD из RA);
- Команды сложения с числом и вычитания числа (16-разрядного);
- Команды условного и безусловного ветвлений;
- Команда цикла loop;
- Команды пропуска такта (NOP) и останова (HALT);
- Команды вычислителям.

Для работы с 32-разрядными числами возможно последовательное использование двух команд, одна из которых работает с младшим словом регистра, а другая – со старшим. Это позволит не усложнять внутреннюю структуру процессора определением того, сколько слов нужно выбрать (одно или два). В любом случае, команда занесения в регистр длинного целого числа должна занимать два слова.

Пример команды вычислительным элементам (запись 4 байт результата) :

```
<< ra5 adr1 ra5 adr2++ wbank1 wbank2 1 e11 1  
e12 >>
```

Угловые скобки – признак команды вычислительным элементам.

Команда производит запись в оба банка по одному и тому же регистру адреса с его инкрементацией при выполнении команды. Регистр RD1 содержит количество повторений выставления команды вычислительному элементу. Если RD1 равен 0, то команда запишет по одному длинному слову в оба банка из внутренней памяти элемента. Если RD1 равен 1, – то по два длинных слова и т.д. "1 e11" и "1 e12" означают, что должна быть выполнена команда 1 первым и вторым вычислительными элементами.

Чтобы реализовать специализированный алгоритм или группу специализированных алгоритмов, необходимо определить набор микропроцессорных команд, позволяющий достичь при решении поставленных задач максимальной производительности. На языке проектирования микросхем типа verilog или VHDL, описывается архитектура процессора с требуемым набором команд, этот набор инструкций загружается в ППВ. Таким образом, мы получаем специализированную вычислительную систему, которая одинаково эффективно - в смысле использования команд - может решать принципиально различные задачи.

5. Программное обеспечение

Неотъемлемой частью разработки любого вычислительного устройства является разработка программного обеспечения. Под программным обеспечением обычно понимаются драйвера, языки программирования, библиотеки функций. В нашем случае основной задачей стала разработка гибкого языка программирования с легко изменяемым набором команд. Причем, требуется учитывать необходимость включения в команду процессора команды вычислительным элементов. Для простоты разработки был выбран язык типа ассемблера. Коды команд могут легко быть изменены. Так же легко могут быть добавлены новые или удалены существующие команды. Поскольку алгоритмы обработки данных имеют небольшие объемы, можно обойтись без компоновщика объектных модулей.

Инструментальное ПО вычислителя построено на базе Диалоговой Системы Структурированного Программирования (ДССП). Изменяемая архитектура вычислителя приводит к необходимости иметь транслятор, позволяющий быстро настраиваться на новую систему команд, и средства отладки, упрощающие разработку алгоритмов.

Инструментальное ПО состоит из упрощенного транслятора с ассемблера, пультового монитора и трассировщика. Пультовый монитор обеспечивает просмотр содержимого регистров и памяти и запуск программы с нужного адреса. Во всех компонентах ПО используется обратная польская запись.

5.1 Среда разработки

Необходимыми характеристиками, которыми должен был обладать инструмент разработки и отладки программ для ППВ, явились следующие:

- Низкоуровневый доступ к аппаратным ресурсам ППВ, в том числе в диалоговом режиме отладки
- Нетрудоемкое написание вспомогательных алгоритмов визуализации и программной обработки результатов
- Предоставление конечному пользователю удобного для него интерфейса при работе с программным обеспечением ППВ.

Программы на популярной в последнее время Java, обладают хорошей межплатформенной переносимостью на уровне исполняемых модулей, но в силу эффекта интерпретации Java-кода Java-машиной, проигрывают в производительности программам на Си. Си, практически повсеместно использующийся для разработки низкоуровневого программного обеспечения, создает исполняемый код, но создание диалогового режима для работы с ППВ, удовлетворяющего потребностям разработчика отлаживающего аппаратуру, является весьма трудоемким. Существуют языки программирования, удовлетворяющие приведенным выше характеристикам. Это Forth и ДССП.

Язык программирования Forth, разработанный Ч.Муром, стал в начале 80-х [S1] годов на короткое время самой распространенной в мире системой программирования. Причиной этого можно считать то, что Forth был первой

общедоступной интегрированной системой программирования, работоспособной на самых скромных в отношении вычислительных ресурсов архитектурах микроЭВМ. Удобство системы программирования некоторое время перевешивали непривычность языка.

В дальнейшем популярность Forth резко упала, что можно объяснить тем, что он допускает программирование только "снизу вверх", т.е. неудобен для коллективной разработки больших программ.

В мире в настоящее время Forth используется для разработки:

- а) тестового ПО;
- б) встроенного ПО;
- в) систем реального времени;
- г) бортового ПО космических аппаратов.

В нашей стране в начале 80-х годов была разработана Диалоговая Система Структурированного Программирования (ДССП), которая во многом превосходит язык Forth.

ДССП (Диалоговая система структурированного программирования) создана на ВМК МГУ в лаборатории Н.П.Брусенцова примерно в 1980 году. ДССП, как и Forth, можно отнести к классу систем программирования, занимающими промежуточное положение между языками программирования высокого и низкого уровня.

Общими характеристиками этих языков являются следующие:

Двухстековая архитектура

Передача входных и выходных параметров производится через стек данных. При вызове подпрограммы, адрес, на который нужно вернуться после выполнения подпрограммы, помещается в стек возвратов.

Обратная польская запись

Это способ записи арифметических выражений, при котором нет необходимости использовать скобки для определения приоритетов выполнения операций. При этом знак операции записывается после операндов. Например: $3*(4+5)$ записывается как $3\ 4\ 5\ +\ *$ или как $4\ 5\ +\ 3\ *$. Это естественный порядок вычислений, который применяет человек.

Словари

Языки Forth и ДССП допускают определение новых операторов на основе имеющихся. Новые операторы можно

сгруппировать по тематике, включив их в словарь. Это позволяет использовать операторы-омонимы.

Компактный код

Исходный текст транслируется в сшитый код, представляющий собой адреса вызываемых процедур. Кроме того, сами тексты программ достаточно компактны. На строке исходного текста размещается несколько операторов.

Гибкость, мобильность, наращиваемость

Архитектура функционального языка предоставляет механизм изменения поведения программы альтернативный механизму виртуальных функций языков объектно-ориентированного программирования. ДССП реализована в среде операционных систем RSX, RT-11, CPM, MSDOS, UNIX, OS9 и, возможно, некоторых других. Существует версия, ядро которой написано на языке Си. Как автономная система, ДССП реализована на процессорах Intel 8080, PDP-11, Intel 80x86, Motorola 68000, R3000.

Для каждой прикладной области можно создать свойственный ей операторный базис и писать программу практически на естественном языке. 1 ОТКРЫТЬ_КРАН 10 ПОДОЖДАТЬ_СЕК 1 ЗАКРЫТЬ_КРАН

Поддержка восходящего программирования (разработка снизу-вверх, от конечных процедур к управляющим системой в целом алгоритмам).

Дополнительные отличия ДССП от Forth:

Более простая запись базовых конструкций

Поддержка нисходящего программирования

Один оператор соответствует одному слову исполняемого кода

("слово есть слово"). Этот принцип позволяет реализовать отладку, рекомпиляцию и нисходящую разработку программ

Встроенный отладчик с рекомпиляцией

Благодаря рекомпиляции, система открыта для пользователя изнутри. Он может изучить работу ее алгоритмов вплоть до примитивных операций.

Высокоуровневый механизм программных прерываний и исключительных ситуаций

Это возможность предусмотреть реакцию на сбой аппаратуры (например, на ошибку чтения файла), сосредоточив средства обработки такой ситуации в отдельном месте программы.

Высокоуровневые структуры данных и операции

Есть возможность работать с элементами структур данных через имена полей, без знания о точном способе размещения структуры в памяти ЭВМ, создавать новые типы данных. Набор базовых операций (присвоение, сложение с числом) может переопределяться для новых типов. Таким образом можно построить тип SET (множество) и определить для него операции пересечения, добавления элемента в множество и другие операции.

Ниже приведена запись некоторых основных конструкций на языке высокого уровня (ЯВУ), на ДССП и Forth:

ЯВУ	ДССП	FORTH
IF (T=0) THEN A	IF0 A	NOT IF A THEN
IF (T<0) THEN A	IF- A	NEG IF A THEN
IF (T>0) THEN A ELSE B	BR+ A B	IF A THEN B ELSE

Прочитать значение переменной X и записать его в Y

Y:=X X ! Y X @ Y !

(THEN в Forth является правой скобкой условного оператора).

5.2 Транслятор

Традиционный подход в разработке трансляторов с ассемблеров состоит в том, что в поданной на вход транслятора очередной строке текста находится оператор ассемблера, который сравнивается с образцами таблицы команд транслятора. Затем, в зависимости от оператора определяются виды операндов. Таким образом, строка является пассивной компонентой синтаксического анализа. Язык ДССП позволяет определять новые операторы на основе имеющихся, а, значит, можно создать новый язык программирования в терминах исследуемой предметной области. В нашем случае это возможность определить оператор с именем команды. При таком подходе строка ассемблера является одновременно и строкой языка ДССП. Это позволяет не реализовывать алгоритмы разбора арифметических выражений, а пользоваться встроенными в ДССП средствами. Транслируемая программа является одновременно алгоритмом выполнения микропрограммы на языке ассемблера ППВ, и программой на языке ДССП, которая

транслирует текст микропрограммы в машинный код ППВ. То есть, транслируемая программа сама управляет своей компиляцией.

Рассмотрим трансляцию команды “add.da” (добавить значение из регистра данных к регистру адреса).

```
: add.da [rd.src,rd.dst] .add.da OP REG2  
REG1[] ;
```

используемые слова определены так:

```
: OP          24 29 DEF ;  
: REG1        20 23 DEF ;  
: REG2        16 19 DEF ;
```

Они задают диапазоны размещения значений в машинном слове для кода оператора, регистра и др. Слово DEF, на котором они основаны, размещает число в диапазон бит оператора для записи числа. “.add.da” хранит код операции.

Таким образом компиляция строки “rd1 ra2 add.da ;;” происходит следующим образом:

- rd1 и ra2 – слова-константы, задающие номера регистра данных и регистра адреса. Когда исполнитель ДССП встречает их в тексте, он помещает соответствующие значения в стек.
- Далее выполняется слово “add.da”. В стек заносится код этой команды, который словом OP сдвигается в соответствующее ему место в машинном коде. При этом код команды извлекается из стека, значение машинного кода формируется в специально отведенной ячейке памяти.
- Аналогично номера регистров сдвигаются в отведенные для них позиции в машинном слове и дополняют собой значение машинного кода.
- Завершающее слово “;;” помещает команду в область выходных данных трансляции.

Определение новой команды укладывается в две строки: определение константы номера оператора (например, командой “38 VALUE .add.da”) и способа трансляции оператора (см. выше определение слова add.da). Такой подход позволяет очень быстро конфигурировать транслятор под новую систему команд.

Чтобы облегчить написание программ, содержащих операторы перехода, в язык ассемблера вводятся метки. При компиляции, адрес для перехода или смещение до него от текущей команды размещается в теле команды. Если метка еще

не встречалась, то адрес перехода заранее неизвестен. Есть два основных подхода решения этой проблемы: либо собирается информация о местах, в которых употребляется имя метки, адрес которой пока неизвестен, и при определении соответствующей метки по всем этим местам производится запись нужных значений, либо трансляция производится за два прохода. На первом определяются значения меток, а на втором производится трансляция текста с использованием этих значений. Второй подход более прост в реализации, хотя и занимает больше времени. Мы можем себе позволить именно его, поскольку программы для ППВ малы.

По причине малого объема программ также можно обойтись без компоновщика.

5.3 Трассировщик

Трассировщик является средством отладки и предназначен для отслеживания значений регистров и флагов управляющего процессора во время выполнения микропрограммы. Дизассемблированные команды при трассировке выглядят отлично от исходного текста, где используется префиксная нотация, но соответствуют стилю команд других ассемблеров.

В результате работы трассировщика с определенными начальными значениями регистров, создается текстовый файл, типичный вид строк которого таков:

```

15E) 26550001 00000000          ---- subq
    1,rd5
15F) 38310000 00000038 0000539B ---- add.da
    rd3,ra1
160) 20210000 000001E0 00000000 ---- mov.dd
    rd2,rd1
161) 0400015C                    N--- bpl      15C
162) 41000000 00000000          ---- ldrd
    0,rd1
163) 28130000 000053D3 00005053 ---- mov.aa
    ra1,ra3
164) 10000000                    ---- nop
165) 95E5BFFF 000006a0          -Z-- sync.r
    ra2++  EL1=3  EL2=3
166) 95E52FFF 000006a0          ---- sync.r
    ra2++  EL1=2  EL2=2

```

```

167) F9E52FFF          ---- sync._
EL1=2 EL2=2
168) ACBC9FFF 00000004 00000004 ---- sync.rw
ra5++, ra5++ EL1=1 EL2=1

```

Трассировщик реализован лишь для одного из вариантов вычислителя.

5.4 Пример программы

Этот фрагмент программы накладывает фильтр, хранящийся во внутренней памяти вычислительных элементов, на изображение в оперативной памяти.

```

rd5 – высота фильтра
ra1 – копия адреса начала подматрицы
rd9 – непройденная высота фильтра в переменной цикла
L: CUC14
rd2 rd1 mov.dd ;; [ширина фильтра/4
записывается в повторитель, т.к. производится
одновременная обработка четырех байт]
nop ;; [пропуск двух тактов для
доступности]
nop ;; [использования счетчика RD1
в команде для вычислительных элементов]
[команда скалярного произведения вектора
изображения и вектора фильтра]
<< ra1 adr1++ rbank1 4 e11 4 e12 >>
nop ;;
nop ;;
rd3 ra1 add.da ;; [перескочить на
следующую строку в подматрице]
CUC14 loop ;; [цикл по строкам подматрицы
(наложение фильтра)]
<< 4 e11 4 e12 >> [две строки для
очистки]
<< 4 e11 4 e12 >> [конвейера]
[сохраним 4 соседние горизонтальные
точки
длинным словом]
<< ra5 adr2++ wbank2 1 e11 1 e12 >>
Двойная точка с запятой является признаком конца
оператора.

```

6. Заключение

Разработанный вычислитель может эффективно использоваться в системах обработки изображений в реальном времени, где требуемые времена отклика порядка миллисекунд и ниже при объеме входных данных от сотен килобайт и выше. Отличительными особенностями разработанного вычислителя являются:

высокая параллельность архитектуры, позволяющая одновременно выполнять большое количество арифметико-логических операций;

аппаратная реализации арифметико-логических операций, что значительно сокращает время их проведения;

возможность гибкой перенастройки архитектуры вычисляющих элементов, что делает вычислитель универсальным устройством, которое можно эффективно использовать при решении широкого класса задач.

Получен положительный результат и накоплен значительный опыт в разработке аппаратных высокопроизводительных средств для решения задач по обработке информации и моделированию нейронных сетей.

Литература

2. Аряшев С.И., Бобков С.Г. Нейронные сети: основные типы, перспективы развития.//Вопросы кибернетики. Сб. статей под ред. В.Б. Бетелина .-Москва, 1997,с. 120-137.
3. Аряшев С.И., Бобков С.Г., Сидоров Е.А. Параллельный перепрограммируемый вычислитель для систем обработки информации и сигналов./Научная сессия МИФИ-99.Всероссийская научно-техническая конференция "Нейроинформатика-99". Сборник научных трудов. В 3 частях. Ч.2 М.:МИФИ, 1999. с.25-33.
4. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов: Пер. с англ. - М.: Мир, 1979.- 536с.

5. Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А. Диалоговая система структурированного программирования ДССП-80. - В кн.: Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ, 1986, с.3-21.
6. Грибков И.В. , Захаров А.В. и др. Предобработка и распознавание двумерных изображений.//Вопросы кибернетики. Сб. статей под ред. В.Б. Бетелина .-Москва, 1997,с. 3-72.
7. Келли М., Спайс Н. Язык программирования ФОРТ. - М.: Радио и связь 1993, 318 стр.
8. Линдли К. Практическая обработка изображений на языке Си : Пер. с англ.- М.: Мир, 1996.-512 с., ил.
9. Newbridge Corporation, SCV64 User Manual,1994.