МОСКОВСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ ПРИБОРОСТРОЕНИЯ И ИНФОРМАТИКИ

Кафедра
“Персональные компьютеры и сети”

Ульянов М.В.

АРХИТЕКТУРЫ ПРОЦЕССОРОВ

Учебное пособие

Москва
2002

УДК 004.31

М. В. Ульянов. Архитектуры процессоров. - М.: МГАПИ, 2002. - 68 с.

ISBN 5-8068-02 68 - X

Рекомендовано Ученым Советом МГАПИ в качестве учебного пособия для специальности 2201.

Рецензенты: к.т.н., проф. Зеленко Г.В.
к.т.н., проф. Рощин А.В.

Предлагаемое издание рекомендуется в качестве учебного пособия для подготовки студентов различных специальностей, изучающих программные и аппаратные методы организации вычислительного процесса.

Для специальности 2201 «Вычислительные машины, комплексы, системы и сети» это издание может быть использовано в качестве учебного пособия по дисциплинам «Теория вычислительных процессов» и «Организация ЭВМ и систем» студентами первого и второго курсов.

В учебном пособии рассмотрены как основные архитектуры процессоров (фон Неймановская архитектура, стековая и конвейерная архитектура, машины потоков данных, процессор пересылок и RISC - процессоры), так и архитектуры оперативной памяти (адресная память с использованием кэш и чередования адресов, ассоциативный подход к выборке данных), а так же решения по организации ввода/вывода данных.

Л $\frac{240\ 402\ 0000}{ЛР020418-97}$

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 5
1. ПОНЯТИЕ АРХИТЕКТУРЫ ПРОЦЕССОРА И ЭЛЕМЕНТНАЯ БАЗА 6
1.1 Понятие архитектуры ЭВМ и архитектуры процессора 6
1.2 Элементная база (физическая база) процессора 7
1.3 Пути совершенствования элементной базы 9
1.3.1 Молекулярные компьютеры 10
1.3.2 Биокомпьютеры или нейрокомпьютеры. 10
1.3.3 Квантовые компьютеры 11
1.3.4 Оптические компьютеры 11
2. ПРОЦЕССОРЫ С КЛАССИЧЕСКОЙ АРХИТЕКТУРОЙ 12
2.1. Основные принципы классической архитектуры: 12
2.2 Организация памяти в классической архитектуре 14
2.3 Набор команд фон - Неймановского процессора 14
2.4 Обработка особых ситуаций и прерывания 15
3. СТЕКОВЫЕ ПРОЦЕССОРЫ 17
3.1 Этапы выполнения команды в фон Неймановском процессоре 17
3.2 Архитектурные особенности стекового процессора 18
3.3 Операции с оперативной памятью 20
3.4 Программирование на стековом процессоре 21
3.5 Замечания по реализации 21
4. КОНВЕЙЕРНЫЕ ПРОЦЕССОРЫ 22
4.1 Предпосылки создания конвейера данных 22
4.2 Структура конвейера данных 23
4.3 Сокращение времени при использовании конвейера данных 23
4.4 Конвейер команд 24
4.5 Многооперационные конвейеры 24
4.6 Проблемы конвейерных процессоров: 26
4.7 Особенности программирования конвейерных процессоров. 26
4.8 Замечания по реализации 28
5. CRAY - ПРОЦЕССОР 29
5.1 Предпосылки создания суперкомпьютеров 29
5.2 Недостатки фон Неймановской архитектуры 29
5.3 Идеи, лежащие в основе CRAY процессора. 29
5.4 Общая структура и состав процессора CRAY. 33
5.5 Производительность и области применения 34
6. ПРОЦЕССОР ПЕРЕСЫЛОК 35
6.1 Иерархия памяти в классической архитектуре 35
6.2 Организация памяти в процессоре пересылок 35
6.3 Организация процессора пересылок 36
6.3.1 Адресная фиксация схем исполнения машинных команд 36
6.3.2 Механизм запуска машинной команды 37

6.4 Пример программы в процессоре пересылок ... 38
6.5 Реализация перехода по адресу и сравнения ... 38
6.6 Замечания по реализации процессора пересылок ... 39
7. АРХИТЕКТУРЫ ПРОЦЕССОРОВ ... 40
И ФОРМАТЫ ДАННЫХ ... 40
7.1. Процессоры с универсальным набором команд ... 40
7.2 RISC – процессоры ... 40
7.3 Теговые машины ... 41
7.4 Гарвардская архитектура ... 43
8. ПОДХОДЫ К ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА И ПОТОКОВЫЕ МАШИНЫ ... 44
8.1 Процедурное программирование. ... 44
8.2 Функциональное программирование ... 45
8.3 Потоковое программирование ... 45
9. АРХИТЕКТУРЫ ПАМЯТИ ... 49
9.1 Классификация архитектур памяти ... 49
9.2 Память с чередование адресов ... 50
9.3 Кэш память ... 51
9.4. Ассоциативная память (безадресная память) ... 52
10. АРХИТЕКТУРНЫЕ РЕШЕНИЯ ... 54
ВВОДА/ВЫВОДА ДАННЫХ ... 54
10.1 Проблемы организации и управления вводом/выводом ... 54
10.2 Основные архитектурные решения ... 54
10.3 Канальный ввод/вывод ... 55
10.4 Архитектура с общей шиной ... 56
10.5 Архитектура ввода/вывода с общей памятью ... 57
11. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ... 58
11.1 Мультипрограммирование (многозадачность) ... 58
11.1.1 Однопроцессорная обработка ... 58
11.1.2 Многопроцессорная обработка ... 60
11.2 Параллелизм независимых ветвей ... 61
11.3 Параллелизм объектов ... 62
12. МАТРИЧНЫЕ СИСТЕМЫ ... 63
12.1 Однородные системы - параллелизм этапов задач ... 63
12.2 Матрицы волнового фронта данных - параллелизм команд ... 65
12.3 Классические матричные системы - параллелизм объектов ... 66
ЛИТЕРАТУРА ... 67

В В Е Д Е Н И Е

Начиная с середины XX века - времени появления первых реальных вычислительных машин - научные идеи и технические решения в области электронно-вычислительной техники получили стремительное развитие. На первом этапе развития ЭВМ предполагалось, что увеличение быстродействия связано, прежде всего, с развитием элементной базы процессоров. Действительно переход на транзисторы (в 50-е годы) и интегральные схемы (в конце 60-х годов) оправдывал эти предположения. Однако ряд научных идей, сформулированных еще в начале 60-х годов (стек, конвейер), показал разработчикам ЭВМ, что и организационные решения могут во многом определять характеристики вычислительных машин. Тем не менее, совершенствование элементной базы успешно продолжается и в настоящее время, достаточно проследить рост тактовой частоты процессоров персональных компьютеров за последние 10 лет.

Одновременно с совершенствованием элементной базы развивались и научные идеи, связанные с логическим построением процессоров, способов организации выполнения последовательности операций, принципами управления ЭВМ - всего того, что впоследствии получило название архитектуры ЭВМ.

В рамках данного учебного пособия мы остановимся на изложении основных принципов и подходов к организации процессоров, оперативной памяти и системы ввода/вывода, ставящих своей целью повышение наблюдаемой производительности ЭВМ при фиксированной элементной базе.

Будут рассмотрены: фон Неймановская архитектура, стековые процессоры, конвейерная обработка команд и данных, машины потоков данных, процессор пересылок, RISC - процессоры и основы многопроцессорных систем.

Архитектуры оперативной памяти представлены в этом учебном пособии адресной организацией с использованием кэш-памяти и чередования адресов, и ассоциативным подходом к выборке данных.

Решения по организации ввода/вывода представлены классической канальной архитектурой, шинной организацией и идеей сквозной адресации памяти.

1. ПОНЯТИЕ АРХИТЕКТУРЫ ПРОЦЕССОРА И ЭЛЕМЕНТНАЯ БАЗА

1.1 Понятие архитектуры ЭВМ и архитектуры процессора

Архитектура ЭВМ - это многоуровневая иерархия аппаратно-программных средств, из которых строится ЭВМ. Каждый из уровней допускает многовариантное построение и применение. Конкретная реализация уровней определяет особенности структурного построения ЭВМ.

Детализацией архитектурного и структурного построения ЭВМ занимаются различные категории специалистов. Инженеры - схемотехники проектируют отдельные технические устройства и разрабатывают методы их сопряжения друг с другом. Системные программисты создают программы управления техническими средствами, информационного взаимодействия между уровнями, организации вычислительного процесса. Программисты разрабатывают пакеты программ более высокого уровня, которые обеспечивают взаимодействие пользователей с ЭВМ и необходимый сервис при решении ими своих задач.

Важнейшими характеристиками ЭВМ являются быстродействие и производительность. И хотя эти характеристики тесно связаны, тем не менее, их не следует смешивать. Быстродействие характеризует технические параметры ЭВМ, а производительность - наблюдаемую скоростью выполнения программ. При этом в основном быстродействие определяется характеристиками элементной базы, на которой построены процессор и оперативная память, а производительность - совокупностью структурных и организационных решений процессора.

Архитектура процессора - совокупность научных идей, структурных, организационных и технических решений, определяющих основные принципы функционирования процессора, его наблюдаемые характеристики и области практического применения.

Помимо архитектурных решений на характеристики процессора и ЭВМ в целом влияют также принятые схемные решения, выбранные или разработанные микропрограммные или схемно реализованные алгоритмы выполнения машинных команд и элементная база процессора. Особо следует отметить влияние на наблюдаемые характеристики процессора решение о границе между программными и аппаратными средствами, которая может быть существенно варьируема между элементарными машинными командами и машинными ко-

мандами, реализующими целые программные фрагменты, например умножение матриц, что схематично показано на рис 1.1.

Программно - аппаратная варьируемость

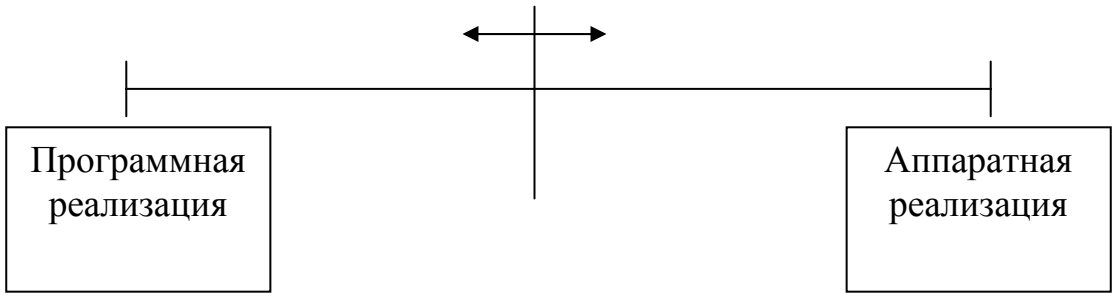


Рис 1.1

1.2 Элементная база (физическая база) процессора

Начиная с первого поколения ЭВМ, и до настоящего времени, все процессоры разрабатываются на основе двоичной системы счисления - это означает, что для реализации процессора необходимо любое физическое устройство, поддерживающее два устойчивых состояния. Основой современной элементной базы является твердотельный планарный транзистор на кремниевой подложке, изготавливаемый по диффузионной технологии.

Технологический процесс представляет собой поэтапную (по областям транзистора) диффузию примесей в кристаллическую структуру кремниевой подложки при температурах порядка 800° с очень жесткими ограничениями на градиент температуры. Для получения приемлемых характеристик градиент должен иметь порядок ± 0,1°/час.

Тем не менее, как физические процессы в твердотельной структуре, так и «шумы», вносимые технологией определяют следующие недостатки p-n-p транзистора, влияющие на его характеристики - паразитные ёмкости p-n перехода и наведённые токи.

Дополнительно к этому сам кремний обладает малой подвижностью электронов, что не позволяет существенно уменьшить время изменения сигнала на выходе, даже при микро миниатюризации размеров транзистора.

Структура p-n-p планарного транзистора в разрезе по подложке приведена на рис 1.2:

Разрез твердотельного планарного транзистора

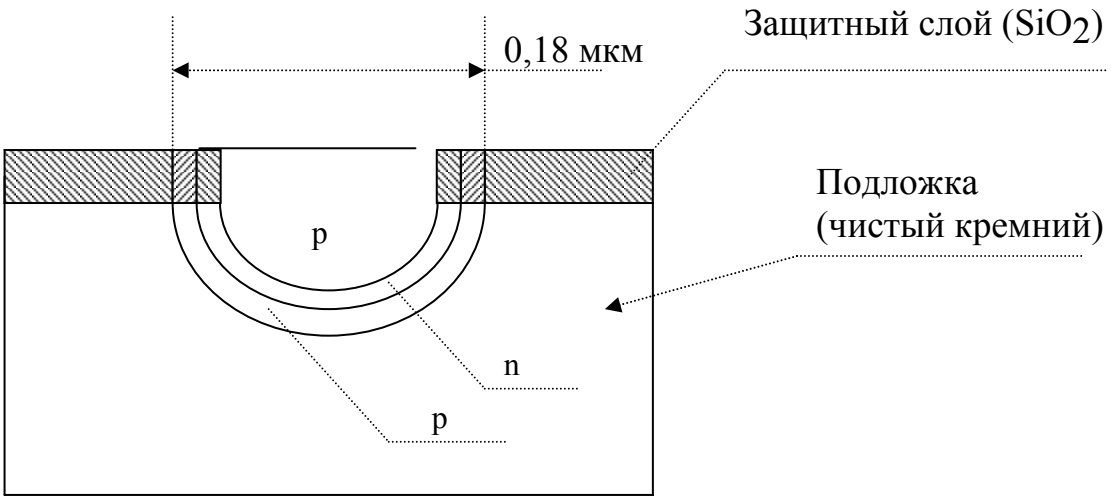


Рис 1.2

С точки зрения быстродействия процессора можно выделить следующие основные характеристики элементной базы:

а) *Время фронта на вентиле.* Логические значения «0» и «1» представляются двумя значениями напряжения, при переключении элемента из одного состояния в другое напряжение не изменяется скачкообразно, а нарастает во времени, приблизительно так, как это показано на рис 1.3. Время фронта определяется такими параметрами твердотельного транзистора, как паразитные емкости переходов и подвижность электронов.

Изменение напряжения на элементе во времени

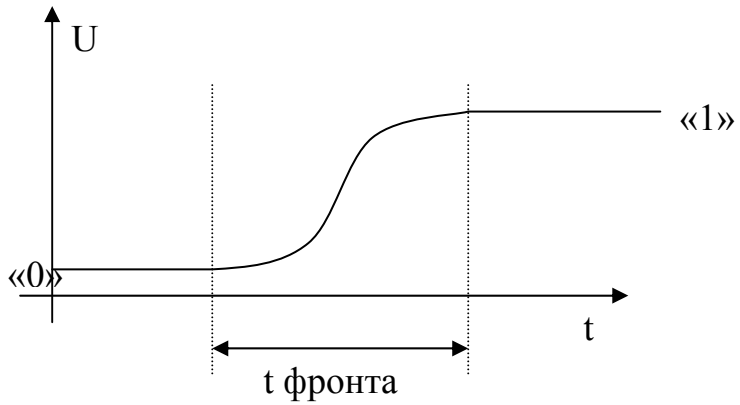


Рис 1.3

Очевидно, что временной такт процессора, определяемый тактовой частотой, должен для обеспечения устойчивой работы элемента в несколько раз превышать время фронта.

б) *Характеристики тепловыделения*

Тепловыделение элементной базы процессоров принято измерять в пикоджоулях на переключение одного бита (1 ПкДж/Бит =10^{-12} Дж/Бит).

Это энергия, выделяемая при переключении вентиля. При современных тактовых частотах и плотностях интеграции на кристалле суммарные тепловыделения имеют порядок нескольких ватт на площади в 1 квадратный сантиметр. В связи с этим достаточно остро стоит проблема отвода тепла от процессора для обеспечения необходимого температурного режима.

в) *Фундаментальное ограничение по скорости света*

При тактовой частоте в 1 Ггц сигнал на проводе проходит 30 см за 1 такт, а при такте в 10 Ггц - 3 см - это означает, что все соединительные линии между элементами, работающими при таких частотах не должны вносить ощутимую задержку при распространении сигнала.

1.3 Пути совершенствования элементной базы

Указанные выше ограничения, связанные с особенностями твердотельных транзисторов на основе кремния не позволяют безгранично увеличивать тактовую частоту, поэтому сегодня рассматриваются как альтернативы в рамках твердотельных технологий, так и альтернативные элементные базы.

Альтернативных путей развития в рамках твердотельных технологий просматривается не очень много.

Большие исследования проводятся в области использования явления сверхпроводимости и туннельного эффекта - эффекта Джозефсона. Работа микросхем при температурах, близких к абсолютному нулю (-273°С), позволяет достигнуть максимальных тактовых частот с минимальным тепловыделением. Очень интересны результаты по использованию «теплой сверхпроводимости». Оказывается, что для некоторых материалов, в частности для солей бария, явление сверхпроводимости наступает уже при температурах около -150°С. Тематика исследовательских работ и их результаты в этом направлении являются закрытыми. Однако с уверенностью можно сказать, что появление таких элементов знаменовало бы революцию в развитии элементной базы процессоров.

Другой альтернативой, в рамках твердотельной технологии, является использование AsGa (арсенид галия), имеющего более высокую подвижность электронов, чем у кремния. Проблема технологии арсенида галлия связана с

созданием защитного слоя на поверхности, необходимого для выделения площадей диффузии.

Внедрение новых технологий производства микросхем связано и с экономическими проблемами. Например, строительство нового завода по производству микросхем с 0,13-микронной технологией обходится от 2 до 4 млрд. долл. Это заставляет искать новые альтернативные пути в элементной базе. Интенсивные поиски идут по многим направлениям. Наиболее перспективными из них следует считать создание молекулярных и биокомпьютеров (нейрокомпьютеров), разработку квантовых компьютеров и разработку оптических компьютеров.

1.3.1 Молекулярные компьютеры.

Во многих странах проводятся опыты по синтезу молекул на основе их стереохимического генетического кода, способных менять ориентацию и реагировать на воздействия током, светом и т.п. Например, ученые фирмы Hewlett - Packard и Калифорнийского университета (UCLA) доказали принципиальную возможность создания молекулярной памяти ЭВМ на основе молекул роксана (http://www.zdnet.ru/prmtreviews.asp?ID=89). Продолжаются работы по созданию логических схем, узлов и блоков. По оценкам ученых, подобный компьютер в 100 млрд. раз будет экономичнее современных микропроцессоров.

1.3.2 Биокомпьютеры или нейрокомпьютеры.

Идея создания подобных компьютеров базируется на основе теории перцептрона — искусственной нейронной сети, способной обучаться. Автором этих идей был Ф. Розенблат. Он указал, что структуры, обладающие свойствами мозга и нервной системы, позволяют получить целый ряд преимуществ:

1) параллельность обработки информационных потоков;
2) способность к обучению и настройке;
3) способность к автоматической классификации;
4) более высокую надежность;
5) ассоциативность.

Компьютеры, состоящие из нейроподобных элементов, могут искать нужные решения посредством самопрограммирования, на основе соответствия множеств входных и выходных данных. В настоящее время уже созданы и ис-

пользуются программные нейропакеты, которые доказывают возможность построения подобных машин на СБИС.

1.3.3 Квантовые компьютеры

Принцип работы элементов квантового компьютера основан на способности электрона в атоме иметь различные уровни энергии: E0, E1,...,En. Переход электрона с нижнего энергетического уровня на более высокий связан с поглощением кванта электромагнитной энергии - фотона. При излучении фотона осуществляется обратный переход. Всеми подобными переходами можно управлять, используя действие электромагнитного поля от атомного или молекулярного генератора. Этим исключаются спонтанные переходы с одного уровня на другой.

Основным же строительным блоком квантового компьютера служит qubit — Quantum Bit, который может иметь большое число состояний. Для таких блоков определен логически полный набор элементарных функций. Известны эксперименты по созданию RISC-процессора на RSFQ-логике (Rapid Single Flux Quantum) и проекты создания петафлопных (1000 триллионов операций/с) компьютеров (http://www.submarme.ru/prmt.cfm?Id=42).

Особо следует отметить реализацию квантового канала на основе разнесенного интерферометра Маха - Циммерманна.

1.3.4 Оптические компьютеры

Идея построения оптического компьютера давно волнует исследователей. Многие устройства ЭВМ используют оптику в своем составе: сканеры, дисплеи, лазерные принтеры, оптические диски CD-ROM и DVD-ROM. Появились и успешно работают оптоволоконные линии связи. Остается создать устройство обработки информации с использованием световых потоков. Способность света параллельно распространяться в пространстве даёт возможность создавать параллельные устройства обработки. Это позволило бы на много порядков ускорить быстродействие ЭВМ.

Пока отсутствуют проекты создания чисто оптических процессоров, но уже проводятся эксперименты по проектированию оптоэлектронных и оптонейронных отдельных устройств.

2. ПРОЦЕССОРЫ С КЛАССИЧЕСКОЙ АРХИТЕКТУРОЙ

2.1. Основные принципы классической архитектуры:

Большинство вычислительных машин первого, второго и третьего поколений строились на принципах сформулированных Дж. фон - Нейманом в 1949 году. Поэтому термин «фон - Неймановская архитектура» является синонимом термина «классическая архитектура». Эти принципы легли в основу большого количества процессоров, да и в настоящее время влияние классической архитектуры в современных процессорах достаточно велико.

Естественно, что понятие классической архитектуры шире, чем принципы Дж. фон-Неймана, и сегодня под ЭВМ с классической архитектурой понимается вычислительная машина, построенная на следующих принципах:

1. Принцип загружаемой программы. Программа и данные, обрабатываемые этой программой должны быть загружены в оперативную память для обработки их процессором. Это означает, что процессор выбирает очередную команду программы и данные, обрабатываемые этой командой из оперативной памяти ЭВМ. Загрузка в оперативную память с внешних устройств выполняется посредством операций ввода/вывода.
2. Принцип микропрограммного управления. Подразумевает наличие универсального АЛУ и специальной памяти для хранения микропрограмм, которые описывают работу процессора при выполнении команды по тактам;
3. Универсальный набор команд. Процессор с точки зрения программирования обеспечивает разработчиков универсальным по типам и операциям набором машинных команд.
4. Обработка особых ситуаций по прерываниям. Особые ситуации, возникающие при работе процессора (ввод/вывод, таймер) или при выполнении команды (деление на ноль, переполнение порядка) обрабатываются с использованием особого аппаратно-программного механизма, называемого прерыванием. При возникновении особой ситуации происходит аппаратная смена содержимого регистра адреса команды, посредством чего процессор начинает выполнять специальную программу операционной системы - обработчик прерываний;

5. Принцип последовательного выполнения команд. Процессор выбирает очередную команду из оперативной памяти по адресу, расположенному а специальном регистре устройства управления процессором - регистре адреса команды. После выборки команды в регистр команд регистр адреса увеличивается на длину машинной команды. Таким образом процессор последовательно выполняет команды загруженной программы, до тех пор пока очередная выбранная команда не будет командой перехода по адресу. В этом случае адрес, указанный в команде перехода замещает текущий адрес в регистре адреса команды УУ процессора, и тем самым следующая выбираемая процессором команда - это та, на которую указывала команда перехода.

Этим принципам соответствует структура процессора, показанная на рис 2.1, регистры общего назначения программно адресуемы и предназначены для целей пользовательского программирования.

Общая структура фон - Неймановского процессора

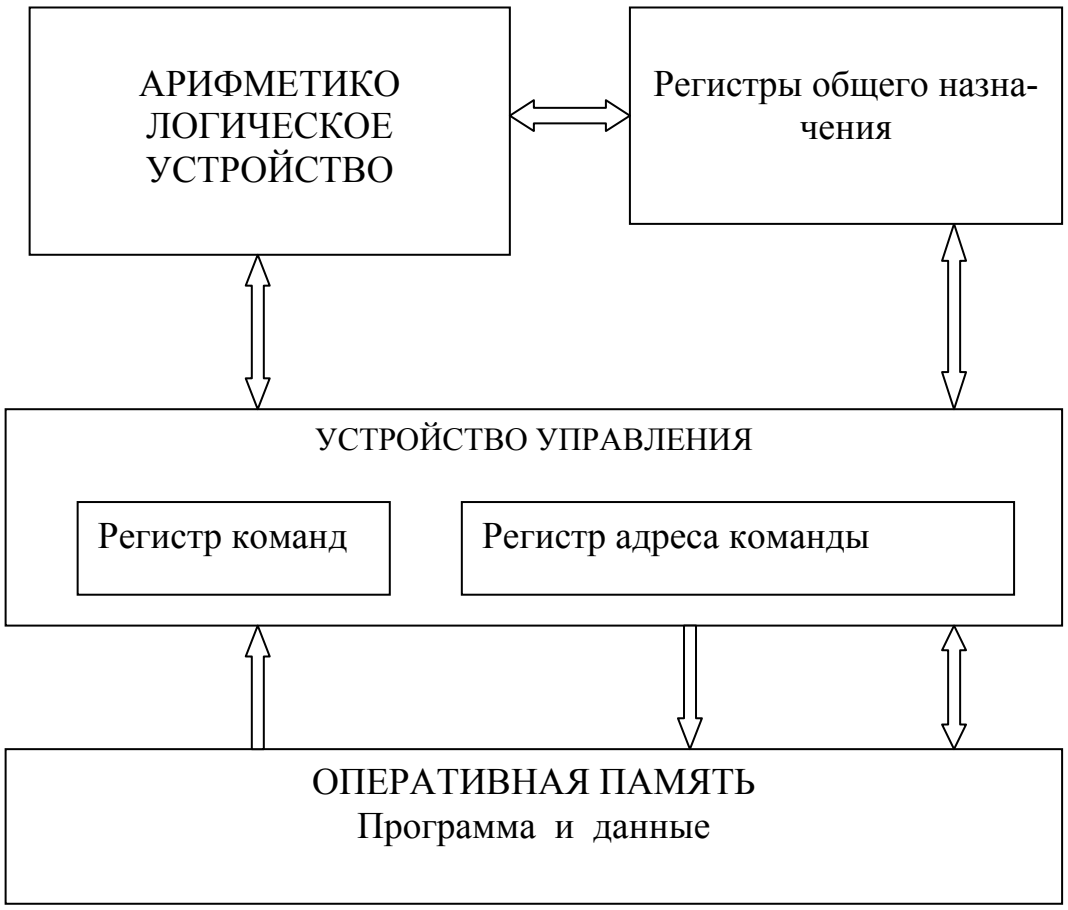


Рис 2.1

2.2 Организация памяти в классической архитектуре

Классическая архитектура предполагает развитую иерархию памяти, связанную с существенно различными временами доступа для разных компонент памяти Эта иерархия включает в себя следующие компоненты:

1. Быстрая регистровая память процессора, включающая программно адресуемые регистры общего назначения процессора, внутренние регистры АЛУ и специальные регистры УУ. В состав специальной памяти процессора может входить и память микропрограмм, как это показано на рис 2.2;

Микропрограммная память в структуре процессора

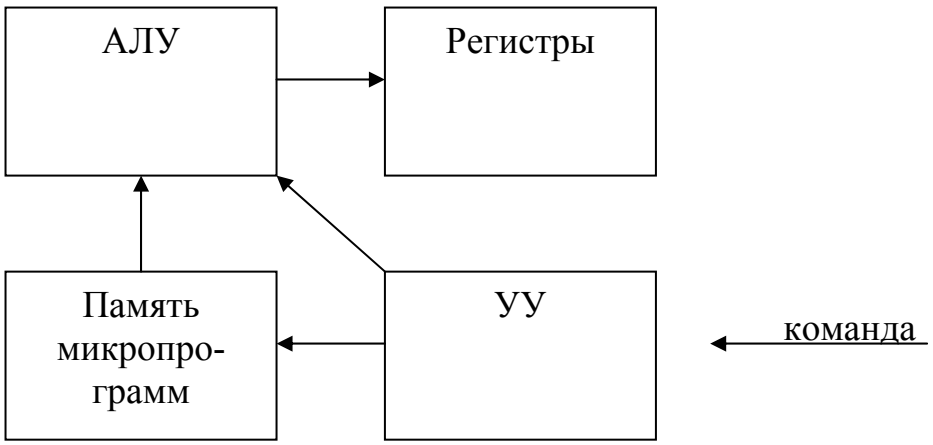


Рис 2.2

2. Более медленная, но зато имеющая значительно большую емкость оперативная память со своим собственным блоком управления;

3. Внешние запоминающие устройства, предназначенные для хранения программ и данных при выключенном процессоре, имеющие существенно большие времена доступа, совместно со специальной аппаратурой обслуживания операций ввода/вывода.

2.3 Набор команд фон - Неймановского процессора

Система команд процессора отражала принятую иерархию памяти и систему адресации байтов оперативной памяти. Дополнительным принципом, реализация которого обеспечивала возможность выполнения программы, начиная с любого адреса оперативной памяти, был принцип перемещаемой программы. После загрузки программы в ОП операционная система в одном из регистров общего назначения передавала адрес загрузки, относительно которого могли быть базированы символические имена операндов.

Принятая система адресации в команде предусматривала вычисление исполнительного адреса в виде суммы смещения, указанного непосредственно в коде команды и содержимого двух регистров общего назначения - базового и индексного, как это представлено на рис 2.3.

Схема формирования исполнительного адреса

[Адрес в ОП] = [Смещение] +[Регистр базы] +[Регистр индекса]
содержимое содержимое

Рис 2.3

Универсальный набор команд явно отражал как иерархию, так и адресацию памяти, включая в себя набор функционально одинаковых команд с различным расположением операндов. Существовали команды формата «регистр-регистр», «регистр-память», и «память-память», где для адресации операнда в памяти необходимо было указать явное смещение и номера двух регистров общего назначения. Наличие разноформатных команд существенно увеличивало общий набор команд процессора (150-200 машинных команд).

Управление процессором осуществлялось через регистр адреса команды, который содержал ряд дополнительных полей. Например в процессоре IBM 370/165 этот регистр назывался словом состояния программы (PSW) и имел длину в 64 бита. Регистр содержал следующую информацию:

- адрес следующей команды в оперативной памяти;
- результат последней команды сравнения;
- коды состояний процессора;
- ключ защиты памяти.

2.4 Обработка особых ситуаций и прерывания

Особые ситуации, возникающие при работе процессора, требуют специальной обработки со стороны операционной системы, т.е. некоторым образом выполнение текущей программы должно быть прервано и ресурс процессора должен быть предоставлен специальным программам операционной системы. Для обработки таких ситуаций в классической архитектуре был предложен специальный механизм, который получил название прерываний. Для иллюст-

рации разделения прерываний по типам и механизма их обработки рассмотрим обработку особых ситуаций, принятую в семействе ЭВМ IBM 360/370.

При разработке этого семейства ЭВМ были выделены следующие типы прерываний:

- программные прерывания - особые ситуации при выполнении команд (деление на ноль, переполнение порядка, потеря значимости и т.д.);

- прерывания ввода/вывода - особые ситуации, возникающие при нормальном или ненормальном завершении операций ввода/вывода;

-прерывания от часов и интервального таймера;

-прерывания от схем контроля - особые ситуации, когда специальные схемы, контролирующие работу процессора, обнаруживали ошибки аппаратуры;

- прерывания по обращению к операционной системе - прерывания, инициируемые обрабатываемой программой, для выполнения функций, находящихся в ведении операционной системы (прерывания по обращению к супервизору).

Механизм обработки прерываний включал в себя загрузку пар PSW для каждого типа прерываний при загрузке операционной системы. Новое PSW прерывания содержало адрес обработчика прерываний данного типа внутри операционной системы. При возникновении прерывания аппаратно производилась смена PSW, как это показано на рис 2.4, что и приводило к запуску обработчика прерываний. Старое PSW было необходимо для возврата из обработчика прерываний в прерванную программу.

Схема обработки прерываний в IBM 360/370

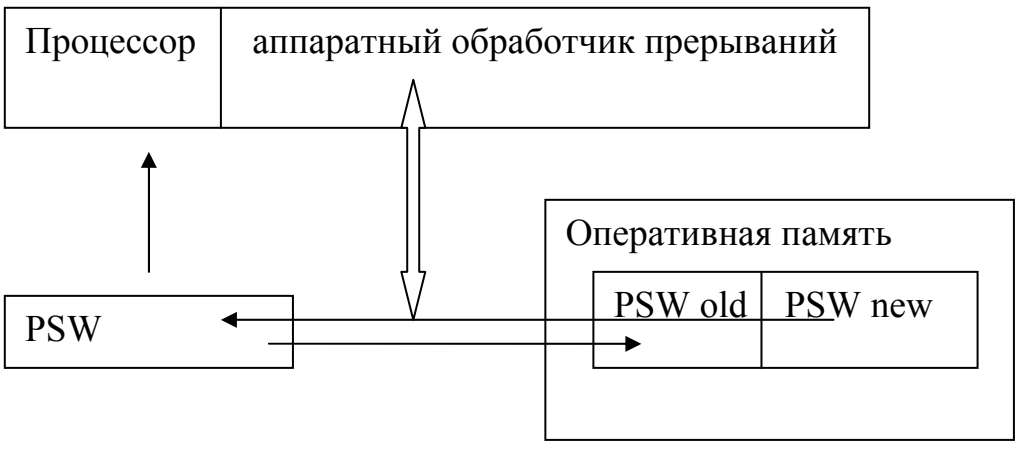


Рис 2.4

3. СТЕКОВЫЕ ПРОЦЕССОРЫ

3.1 Этапы выполнения команды в фон Неймановском процессоре

Для понимания принципов, на которых основывается идеология стекового процессора необходимо более подробно рассмотреть этапы выполнения команд в классической фон - Неймановской архитектуре и ряд узких мест, приводящих к определенной потере времени при выполнении последовательности операций.

Можно выделить следующие этапы выполнения команды в классической фон - Неймановской архитектуре:

1) выборка устройством управления команды из ОП (или из кэш памяти) в регистр команд;
2) модификация адреса в регистре команд на длину выбранной команды;
3) обработка кода операции - коммутация АЛУ на соответствующую микропрограмму или операционную схему;
4) коммутация регистров и АЛУ в соответствии с информацией команды;
5) вычисление адреса операнда в ОП;
6) выборка операнда из ОП в АЛУ;
7) выполнение команды процессором (АЛУ);
8) обработка результата выполнения команды - запись результата.

Схема взаимодействия регистров процессора и АЛУ представлена на рис 3.1:

Схема взаимодействия регистров процессора и АЛУ

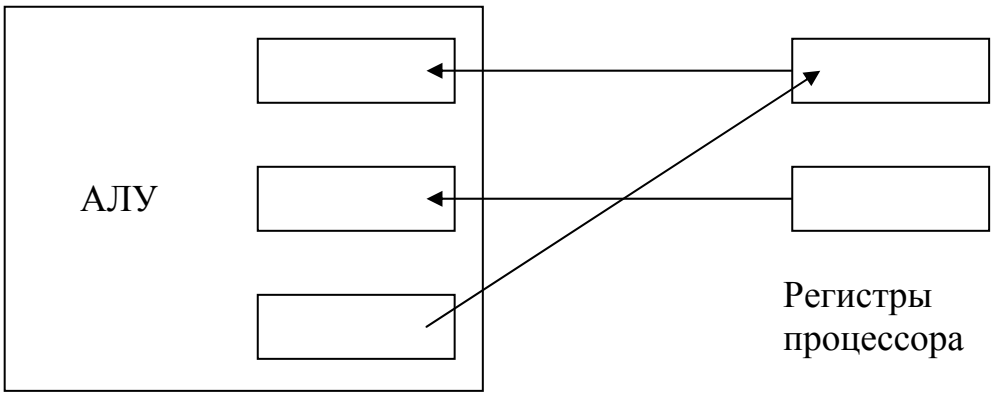


Рис 3.1

3.2 Архитектурные особенности стекового процессора

Основной идеей разработчиков стекового процессора был отказ от программно адресуемых регистров в пользу аппаратного стека. Стек представляет собой блок памяти с двумя фиксированными операциями - операцией помещения информации в стек, при этом новая информация размещается вверху стека, а все ранее хранимые элементы проталкиваются вниз, и операцией выборки из стека, при которой верхний элемент стека выталкивается и передается на обработку, а все остальные элементы продвигаются на единицу вверх. Таким образом, непосредственно доступным является только верхний элемент стека. Схема стека приведена на рис 3.2.

Упрощенная схема стека

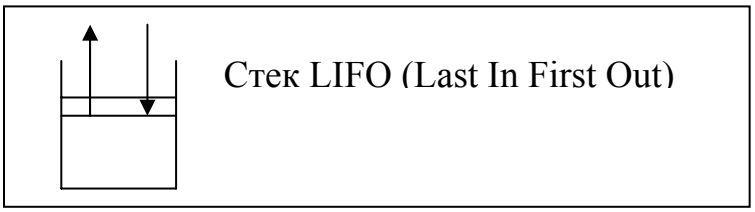


Рис 3.2

Прямое использование стека вместо регистров приводит к структуре, в которой остаются внутренние регистры АЛУ. Эта структура обладает тем недостатком, что остаются операции пересылки операндов в АЛУ, приводящие к увеличению времени выполнения операции. Структура простого стекового процессора приведена на рис 3.3

Структура простого стекового процессора

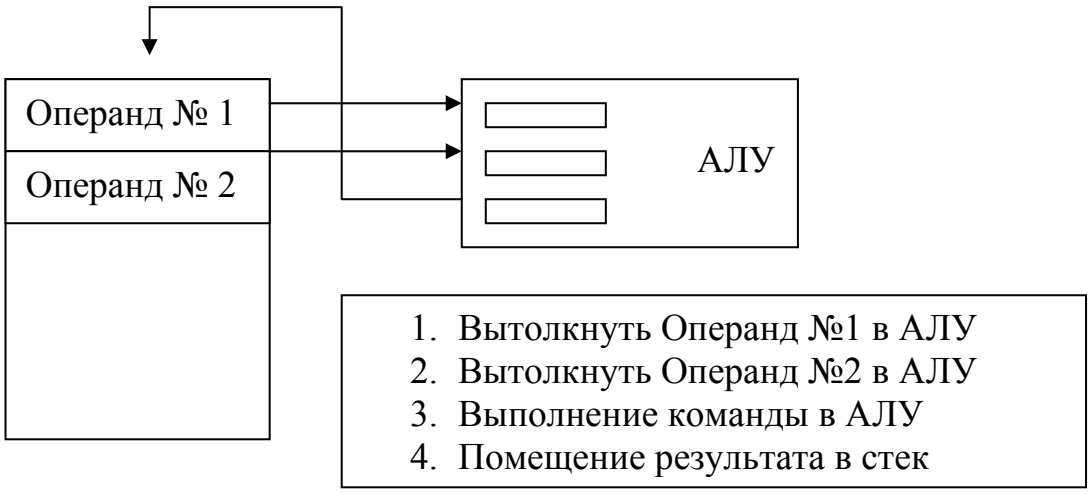


Рис 3.3

Очевидное усовершенствование такой структуры связано с использованием верхних элементов стека в качестве регистров АЛУ. Таким образом, мы приходим к структуре стекового процессора с прямой коммутацией, работающего по следующему принципу:

1) выполнение команды, операндами которой всегда являются верхний и непосредственно следующий за ним элементы стека (для бинарных операций);
2) формируемый в АЛУ результат пересылается по месту операнда №2 в стеке;
3) стек продвигается на один элемент вверх - тем самым результат предыдущей команды автоматически становится операндом следующей команды.

Отметим, что в этом случае АЛУ работает не с чисто стековой структурой, так как доступными (коммутированными) являются два элемента стека.

Отметим положительные особенности данной архитектуры:

- отсутствие этапа коммутации АЛУ с операндами приводит к сокращению времени выполнения команды;

- прямая передача результатов операций от одной к другой через верхний элемент стека позволяет упростить подготовку следующей команды;

- короткие команды без операндов, так как положение операндов фиксировано в двух верхних элементах стека, приводят к более короткому машинному коду программы.

Схема стекового процессора с прямой коммутацией приведена на рис 3.4

Схема стекового процессора с прямой коммутацией

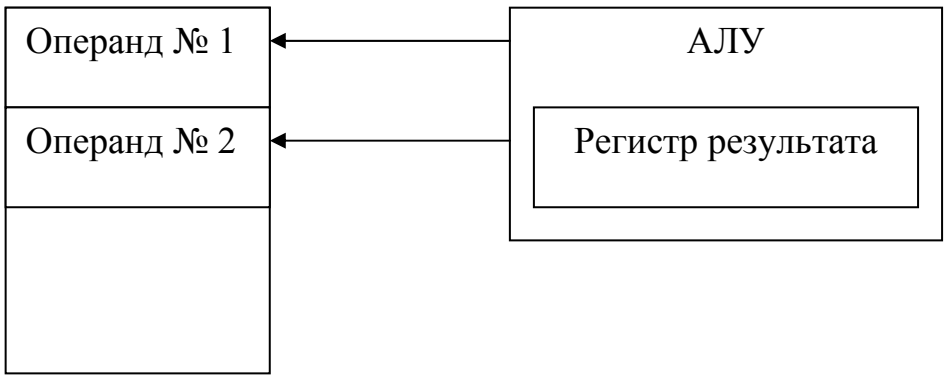


Рис 3.4

Специфика стекового процессора заключается, прежде всего, в необходимости специальных подходов к программированию, связанных с представлением арифметических выражений в так называемой польской постфиксной записи и проблеме хранения промежуточных результатов, которая решается введением специальной команды, дублирующей содержимое верхнего элемента стека вниз.

3.3 Операции с оперативной памятью

В системе команд стекового процессора должны быть предусмотрены специальные команды, осуществляющие загрузку стека содержимым по адресу ОП и выталкивание элемента стека в ОП. Такие команды будут содержать в той или иной форме адреса ОП, которые в дальнейшем будут для простоты представляться символическими именами.

Будем обозначать операции «из ОП в стек» и «из стека в ОП» с учетом символических адресов следующим образом:

- A↓ - содержимое по адресу A помещается в стек, все остальные элементы стека проталкиваются вниз;

- Y↑ - содержимое верхнего элемента стека помещается в память по адресу Y, стек продвигается на один элемент вверх.

Реализация операций перемещения элементов стека должна быть достаточна быстрой по времени, при этом поэлементная перезапись, как это показано слева на рис 3.5 не обеспечивает временных характеристик. Для реализации быстрого стека применяется идея регистров параллельного переноса, как это показано справа на рис 3.5:

Схема проталкивания элементов стека

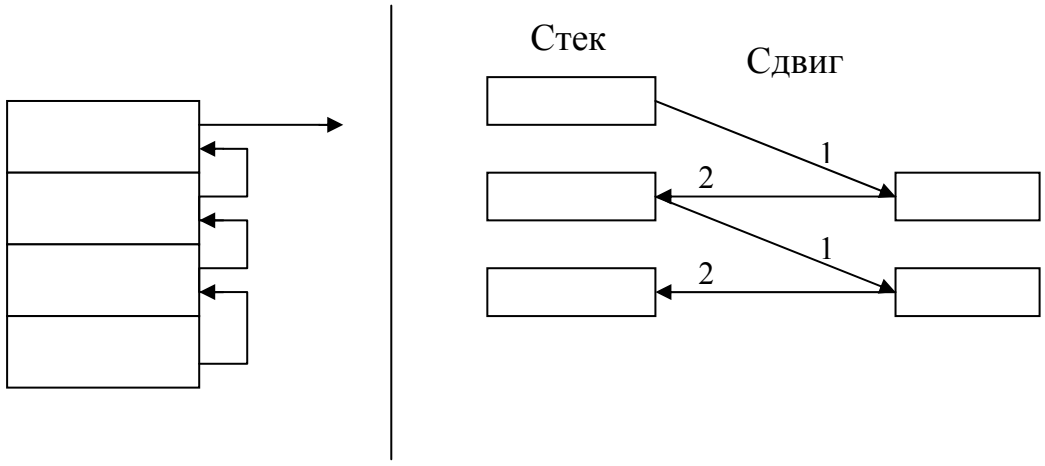


Рис 3.5

3.4 Программирование на стековом процессоре

Рассмотрим несколько простых примеров вычисления арифметических выражений с учетом особенностей архитектуры стекового процессора. В дополнение к командам работы с памятью введем обозначение команд арифметических операций в виде знаков операций и унарную операцию 1/α для обозначения операции вычисления обратной величины.

1. Выражение C=A+B

Фрагмент программы A↓, B↓ ,+ ,C↑

2. Выражение D=A+B+C

Фрагмент программы A↓, B↓, +, C↓, +, D↑

Или A↓, B↓, D↓, +, +, C↑

3. Выражение Y=((A+B)*E)/X

Фрагмент программы A↓, B↓, +, E↓, *, X↓, 1/α, *, Y↑

Или X↓, E↓, B↓, A↓, +, *, /, Y↑

3.5 Замечания по реализации

Первые реализации стековых процессоров относятся ко второму поколению ЭВМ, т.е. к началу 1960-х годов. Стековую идею впервые широко использовала фирма Burroughs, начиная с машины B 5000 и далее в машинах серии B 6000 и B 6600, из отечественных разработок следует упомянуть машину БЭСМ - 6 с одними из лучших показателей производительности в своем классе.

В настоящее время идеи стековой архитектуры используются для построения сопроцессоров с плавающей точкой (математических сопроцессоров) в процессорах Intel и AMD.

Отметим еще одну идею, связанную с использованием стека - стековую выборку команд. Идея предполагает реализацию быстрого буфера команд в устройстве управления в виде стека, в этом случае в УУ явно отсутствует регистр адреса команды, так как очередная команда выбирается из верхнего элемента стека. Возникающая очевидно при этом проблема выполнения команды перехода по адресу может быть решена с использованием циклического стека с адресацией.

4. КОНВЕЙЕРНЫЕ ПРОЦЕССОРЫ

4.1 Предпосылки создания конвейера данных

Общая идея конвейера связана с разбиением некоторого процесса обработки объектов на независимые этапы и организацией параллельного выполнения во времени различных этапов обработки различных объектов, передвигающихся по конвейеру от одного этапа к другому. Поэтому основой разработки конвейера является разбиение процесса на независимые этапы. Рассмотрим такое разбиение на примере машинной команды умножения чисел с плавающей точкой. Формат хранения действительных чисел - чисел с плавающей точкой (FP) представлен на рис 4.1:

Формат хранения действительных чисел (FP)

Рис 4.1

Этапы выполнения машинной команды умножения:

1) Сложение порядков;
2) Нормализация мантисс (приведение мантисс к виду 0,1xxxx);
3) Умножение мантисс;
4) Нормализация результата.

Таким образом, команда умножения может быть разделена на четыре этапа, которые могут быть реализованы аппаратно в виде четырех операционных блоков (сегментов), как это показано на рис 4.2

Операционные блоки для машинной команды умножения действительных чисел

Рис 4.2

4.2 Структура конвейера данных

Создание конвейера предполагает выполнение следующих действий:

1) Деление машиной команды на этапы;
2) Аппаратная реализация этапов в виде конвейерных блоков (сегментов);
3) Создание входных / выходных регистров блоков для передачи результатов.

Таким образом каждый сегмент конвейера имеет структуру, показанную на рис 4.3, где ОР1 и ОР 2 - входные и выходные операнды блока (регистры), а R - поле результата

Структура конвейерного сегмента

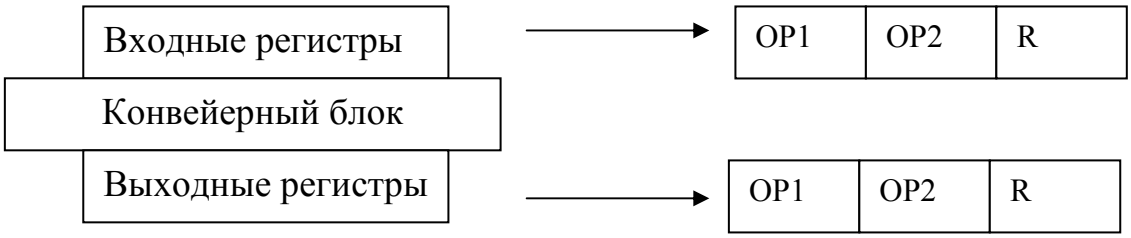


Рис 4.3

Последовательно соединяя конвейерные блоки (сегменты) в порядке следования этапов выполнения машинной команды мы получаем конвейер данной машинной операции для обработки потока данных, в связи с чем такие конвейеры получили название конвейеров данных.

4.3 Сокращение времени при использовании конвейера данных

Пусть при конвейерной обработке машинная команда, как показано на рис. 4.2, разбивается на несколько (допустим, **n**) блоков, и каждая пара операндов последовательно обрабатывается в каждом таком блоке, начиная с первого по **n**-й, причем, как только одна пара операндов заканчивает обрабатываться на некотором операционном блоке, то этот блок начинает обрабатывать пару другую пару операндов, переданных с предыдущего блока. Таким образом, все блоки операционного конвейера работают одновременно и выполняют **n** разных этапов обработки для **n** разных пар операндов. Допустим, что конвейер спроектирован таким образом, что время прохождения одной пары операндов в обычном АЛУ и конвейере одинаково и равно T, а число пар операндов, которые можно последовательно пропускать через конвейер, достаточно велико и

равно N. В таком случае в исходном операционном блоке АЛУ на выполнение одной операции потребуется время T, но если используется n-звенный конвейер, то на обработку N операций потребуется время **(n + N) • (T/n)**.

При этом на одну операцию потребуется время **((n + N)*T)/(n*N)**. Если отношение **N/n** достаточно велико, то это время приближается к **T/n**, т. е. скорость вычислений возрастает приблизительно в **n** раз, где **n -** количество блоков или сегментов конвейера.

4.4 Конвейер команд

В случае, если процессор содержит конвейер данных, то скорость подачи очередной команды из УУ на конвейер должна быть согласована со скоростью конвейера данных. Таким образом, необходима синхронизация конвейера данных и устройства управления процессором.

Реально при N сегментном конвейере необходимо подавать данные на первый блок конвейера данных в N раз быстрее, чем при обычной реализации процессора. При фиксированной тактовой частоте реальным единственным решением данной проблемы является построение конвейера команд. По аналогии с конвейером данных в устройстве управления выделяются самостоятельные этапы подготовки команды к выполнению, они реализуются аппаратно в виде конвейерных блоков, и через эти блоки пропускается поток команд.

Например, в устройстве управления можно выделить следующие конвейерные блоки (сегменты):

1) выборка команды по адресу из ОП (или из КЭШ памяти);
2) дешифрация и обработка кода операции;
3) выборка первого операнда;
4) выборка второго операнда.

Таким образом, мы получаем конвейер команд, позволяющий при определенных условиях согласовать скорость конвейера данных и устройства управления.

4.5 Многооперационные конвейеры

При реализации идеи конвейерной обработки для различных операций процессора возможны два следующих подхода, которые называются конвейером в ширину и конвейером в глубину.

Конвейер в ширину предполагает аппаратную реализацию каждой операции в виде набора конвейерных сегментов. Поскольку некоторые этапы выполнения разных машинных команд совпадают (например - нормализация), то такая реализация является аппаратно избыточной, но позволяет повысить наблюдаемую скорость процессора для рядя специфических задач. Структура конвейера в ширину приведена на рис 4.4

Структура конвейера в ширину

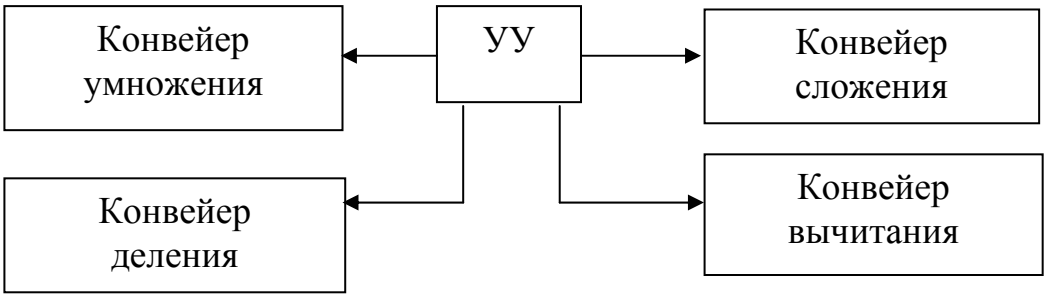


Рис 4.4

Конвейер в глубину предполагает последовательное соединение конвейерных сегментов в один «глубокий» конвейер, при этом вместе с операндами некоторой команды устройство управления передает на такой конвейер значение регистра маски, содержащего единицы в тех позициях, которые соответствуют необходимым для данной операции конвейерными сегментам.

Пример конвейера в глубину фирмы Texas Instruments в системе ASC приведен на рис 4.5. В этой структуре отсутствует дублирование конвейерных блоков, но затрачивается время на передачу операндов через блоки, неиспользуемые в данной операции.

Конвейер в глубину

Рис. 4.5

4.6 Проблемы конвейерных процессоров:

a) *Стоимость аппаратных средств.* Стоимость аппаратной реализации растёт за счёт конвейерных блоков, входных и выходных регистров блоков, особенно для конвейеров в ширину и конвейера команд;
b) *Обработка особых ситуаций.* На одном из конвейерных сегментов возникает ситуация, аналогичная программному прерыванию в фон - Неймановском процессоре - например - результат сложения порядков превышает разрядную сетку числа. При этом возникает проблема остановки конвейера, и, следовательно, обработка особых ситуаций в конвейерном процессоре будет более сложной.
c) *Синхронизация*. Устройство управления должно выполнять синхронизацию конвейерных блоков, конвейера команд и конвейера данных.

4.7 Особенности программирования конвейерных процессоров.

Собственно наличие аппаратно реализованного конвейера данных и конвейера команд еще не означает значительного увеличения наблюдаемой скорости работы процессора из-за наличия ряда условий, при которых конвейер данных может быть нормально загружен - это отсутствие связей по данным и связей по управлению в потоке конвейеризируемых команд

Связанные по данным операции - это ситуация, при которой результат предыдущей (не обязательно непосредственно, но в рамках глубины конвейера) команды является операндом следующей:

Y ← D*C;
.....
Z ← **Y***E:

В этой ситуации команда умножения **Y***E должна быть задержана до выхода значения **Y** из конвейера умножения, как результата умножения D*C.

Таким образом, необходимы особые подходы к программированию конвейерных процессоров, которые мы продемонстрируем на следующих двух задачах:

1. Задача суммирования элементов массива:

Приемлемый для конвейера алгоритм должен обеспечить поток несвязанных операндов сложения, что приводит к алгоритму попарного сложения, схема которого в виде бинарного дерева сложений глубиной $\log_2 N$ приведена на рис 4.6 справа.

Схема попарного сложения элементов массива

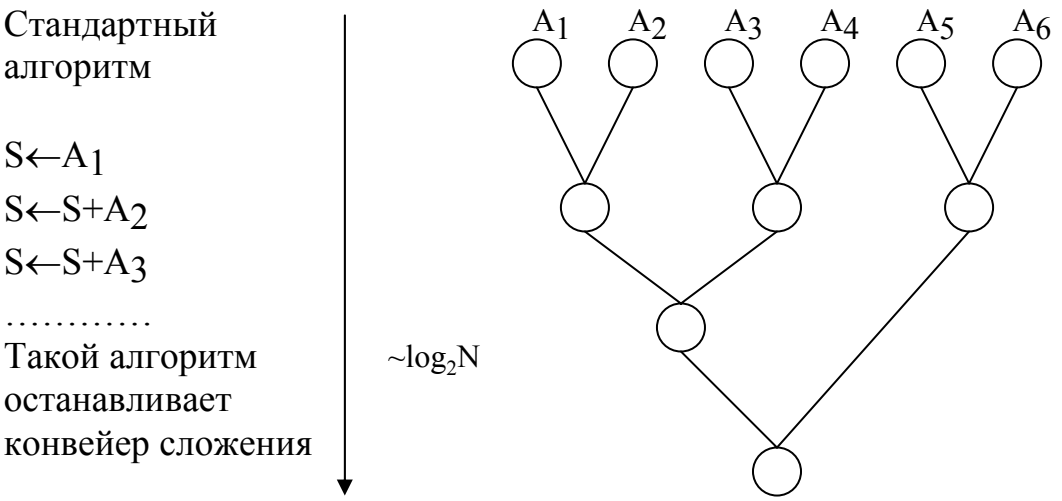


Рис 4.6

2. Задача умножения матрицы на вектор

Умножение $\bar{A} * \bar{X} = B$

Прямая схема умножения :

b1=x1*a11+x2*a12+x3*a13+…..

b2=x1*a21+x2*a22+x3*a23+…..

b3=x1*a31+x2*a32+x3*a33+…..

Если мы реализуем алгоритм умножения по вышеприведенной схеме, то, несмотря на наличие независимых пар по умножению, мы сталкиваемся с необходимостью добавления очередного результата умножения в B[i], что приводит к задержке конвейера сложения.

Оптимальный для конвейера алгоритм связан с переворотом порядка умножения на 90° - т.е. мы умножаем вектор **X** не на строки матрицы **A**, а на столбцы матрицы и добавляем полученные произведения ко всем элементам вектора ***B***. При этом добавление происходит параллельно во все элементы столбца, и следовательно может быть успешно конвейеризировано. Схема такого умножения приведена на рис 4.7

Схема конвейерно - оптимального умножения матрицы на вектор

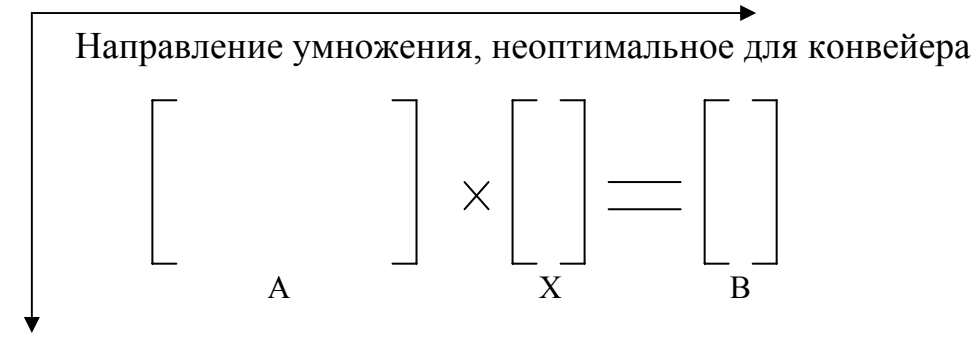


Рис. 4.7

4.8 Замечания по реализации

Необходимость решать крупные вычислительные задачи с более высокой скоростью привела в 60-е годы к необходимости разработки ЭВМ с высокоскоростной параллельной обработкой данных. К первым машинам такого типа относятся LARC фирмы UNIVAC и IBM 7030, в которых впервые был применен принцип конвейера команд.

Что касается области конвейерных суперкомпьютеров, то здесь одной из первых была фирма Control Dada Corporation (CDC). В. 1964 г. была создана ЭВМ CDC - 6600, а в 1969 г. - CDC - 7600, которые объединились в семейство CYBER. Конвейерные принципы были использованы так же в машинах STAR 100 фирмы CDC и ASC формы Texas Instuments, наиболее полно конвейерные принципы нашли свое отражение в архитектуре CRAY процессора.

5. CRAY - ПРОЦЕССОР

5.1 Предпосылки создания суперкомпьютеров

a) задачи вычислительной математики и в особенности векторные и матричные задачи большой размерности с трудоемкостью порядка нескольких MY (1MY=1 миллион операций в секунду в течении года);

b) задачи реального времени с большой трудоёмкостью (например - обсчет информации, поступающей за 1 оборот локатора), требующие повышенной надежности аппаратных средств;

c) задачи моделирования сложных объектов и систем (например - задачи моделирования в области ядерной физики).

5.2 Недостатки фон Неймановской архитектуры

a) Смешанная адресация в большинстве команд:

Команда «Сложить содержимое регистра и содержимое по адресу оперативной памяти» приводит к:

- выборке команды из ОП;
- выборке операндов из ОП.

b) Векторные операции - реализация циклами:

```
Цикл I от 1 до N
   C[i]←A[i]+B[i];
кц;
```

Эта структура приводит к необходимости индексировать элементы массива:

```
A[i]=A[0]+i*L;
      i←i+1;
```

Таким образом, для доступа к элементам массива выполняется много дополнительных операций.

5.3 Идеи, лежащие в основе CRAY процессора.

В 7О-х годах бывший сотрудник и один из руководителей фирмы CDC Seymour Cray (Сеймур Крей) организовал собственную фирму, которая занялась проектированием сверхбыстродействующей ЭВМ, известной под названием Cray -1 с быстродействием, превосходящим 150 млн. операций в секунду и с широким использованием новой интегральной технологии

В основу нового процессора были положены следующие идеи:

5.3.1 Блок (сегмент) векторных регистров

Помимо обычных регистров в процессоре предусматривался блок векторных регистров, предназначенных для обработки массивов за одну машинную команду с широким использованием конвейеров арифметических операций:

Векторный блок состоял из восьми 64-элементных векторных регистров, которые предназначались для хранения восьми операндов-векторов. Каждый такой операнд состоял из 64 компонент (элементов). В свою очередь каждый компонент представлял собой 64-разрядное слово, в котором хранилось число с плавающей или фиксированной точкой. В системе команд были предусмотрены специальные операции, в качестве операндов которых выступали многокомпонентные векторы. Не во всех задачах требуется обрабатывать 64-элементные векторы. Специальный управляющий регистр центрального процессора позволял указывать требуемую размерность. Этот регистр был программно-управляемым, что позволяло в процессе вычислений изменять размерность обрабатываемых векторов. Кроме того, в центральном процессоре предусматривался регистр маски, с помощью которого можно было блокировать выполнение арифметических действий над некоторыми компонентами вектора.

5.3.2 Регистровый трёхадресный ассемблер

Для получения лучших временных характеристик большинство машинных команд было реализовано как команды с регистровыми операндами.

Система команд машины Сгау-1 прямо отражает регистровую структуру центрального процессора, своеобразие связей функциональных модулей с операционными регистрами и связи их с главной памятью. Команды машины Сгау-1 двух форматов: короткие команды - 16 разрядов и длинные - 32 разряда.

Семь первых разрядов определяют код операции, затем следуют трехразрядные поля i, j, k, определяющие соответственно номер регистра результата и номера регистров исходных операндов.

5.3.3 Регистровая буферизация и «прозрачная память»

Следуя регистровой идеологии, разработчики CRAY организовали буфера операндов в виде групп регистров общего назначения.

В состав регистровой памяти центрального процессора входят две группы вспомогательных буферных регистров, сокращающих число обращений к главной памяти. В первую группу входят 64 так называемых В - регистра, которые служат для накопления операндов, поступающих из А - регистров или направляемых в А - регистры из главной памяти. Во вторую группу входят 64 буферных регистра операндов, связанных с S-регистрами. Они называются Т-регистрами и служат тем же целям в отношении главной памяти, что и В - регистры. Совместно В- и Т- регистры можно рассматривать как единый буфер для хранения часто используемых операндов и их адресов.

Для объяснения механизма «прозрачной памяти» рассмотрим фрагмент программы для сложения двух чисел:

Загрузить 9,B;

Загрузить 11,C;

Сложить 7,9,11;

Команде «Сложить» предшествуют две команды загрузки содержимого из ОП в регистры - устранение этих команд из программы привело бы к значительному повышению эффективности процессора. Идея «прозрачной памяти» заключается в совмещении во времени этапа выборки команды из оперативной памяти центральным процессором и загрузки в регистр содержимого по указанному адресу процессором оперативной памяти.

Схема работы «прозрачной памяти» приведена на рис. 5.1

Схема работы процессора оперативной памяти CRAY

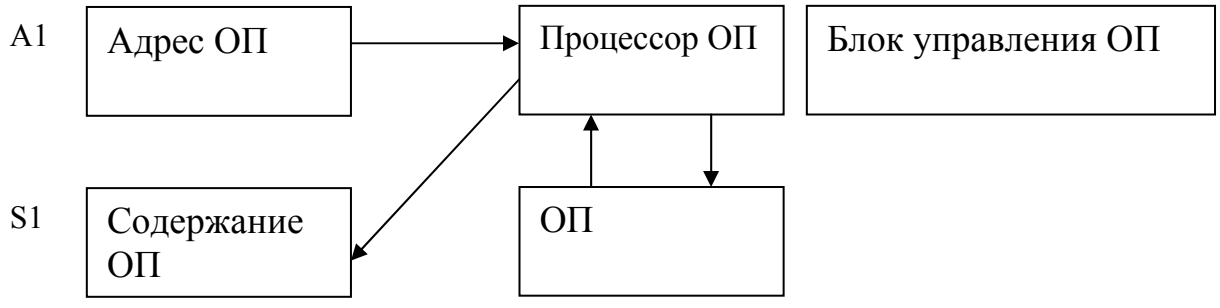


Рис 5.1

5.3.4 Функциональные конвейерные блоки операций

Двенадцать функциональных устройств машины Cray, играющие роль арифметико-логических преобразователей, не имеют непосредственной связи с

главной памятью. Так же как и в машинах семейства CDC-6000, они имеют доступ только к быстрым операционным регистрам, из которых выбираются операнды и на которые засылаются результаты после выполнения соответствующих действий.

В процессоре CRAY реализован конвейер «в ширину» - каждая операция представляет собой отдельный конвейер с различным количеством сегментов, однако такт любого сегмента любого конвейера фиксирован (Для CRAY-1 - 12,5 нс). Кроме того специальный механизм управления позволяет коммутировать конвейеры между собой- для передачи результатов одного конвейерного блока на вход другого.

В совокупности с векторными регистрами функционально - распределенный конвейер представляет собой мощный механизм увеличения наблюдаемой производительности процессора.

Схема конвейера арифметических операций приведена на рис 5.2

Схема конвейера «в ширину» CRAY - процессора

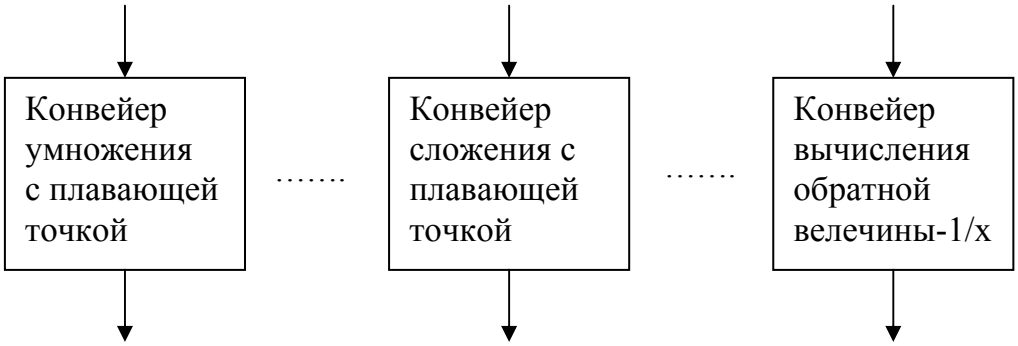


Рис 5.2

5.3.5 Буферизация команд

В состав центрального процессора машины Cray-1 входит регистровая буферная память значительного объема для промежуточного хранения команд программы, исполняемой в данный момент. Эта буферная память состоит из четырех секций, каждая по 16 слов. Последовательность команд программы предварительно поступает в этот буфер. Если она содержит условный переход, то в буфере накапливаются также команды, относящиеся к последовательности, на которую возможен этот условный переход. Буфер команд является средством ускорения работы устройства управления, минимизируя время ожидания чтения команд из главной памяти.

5.4 Общая структура и состав процессора CRAY.

В состав центрального процессора Cray-1 входят:

• главная (оперативная) память, разделенная на 16 независимых по обращению блоков;

• регистровая память, состоящая из пяти групп быстрых регистров, предназначенных для хранения и преобразования адресов и данных;

• функциональные модули АЛУ, в состав которых входят 12 параллельно работающих конвейерных блоков, служащих для выполнения арифметических и логических операций;

• устройство управления (УУ), выполняющее функции управления параллельной работой модулей, блоков и устройств центрального процессора.

Обобщенная структура процессора приведена на рис 5.3

Обобщенная структура Cray процессора

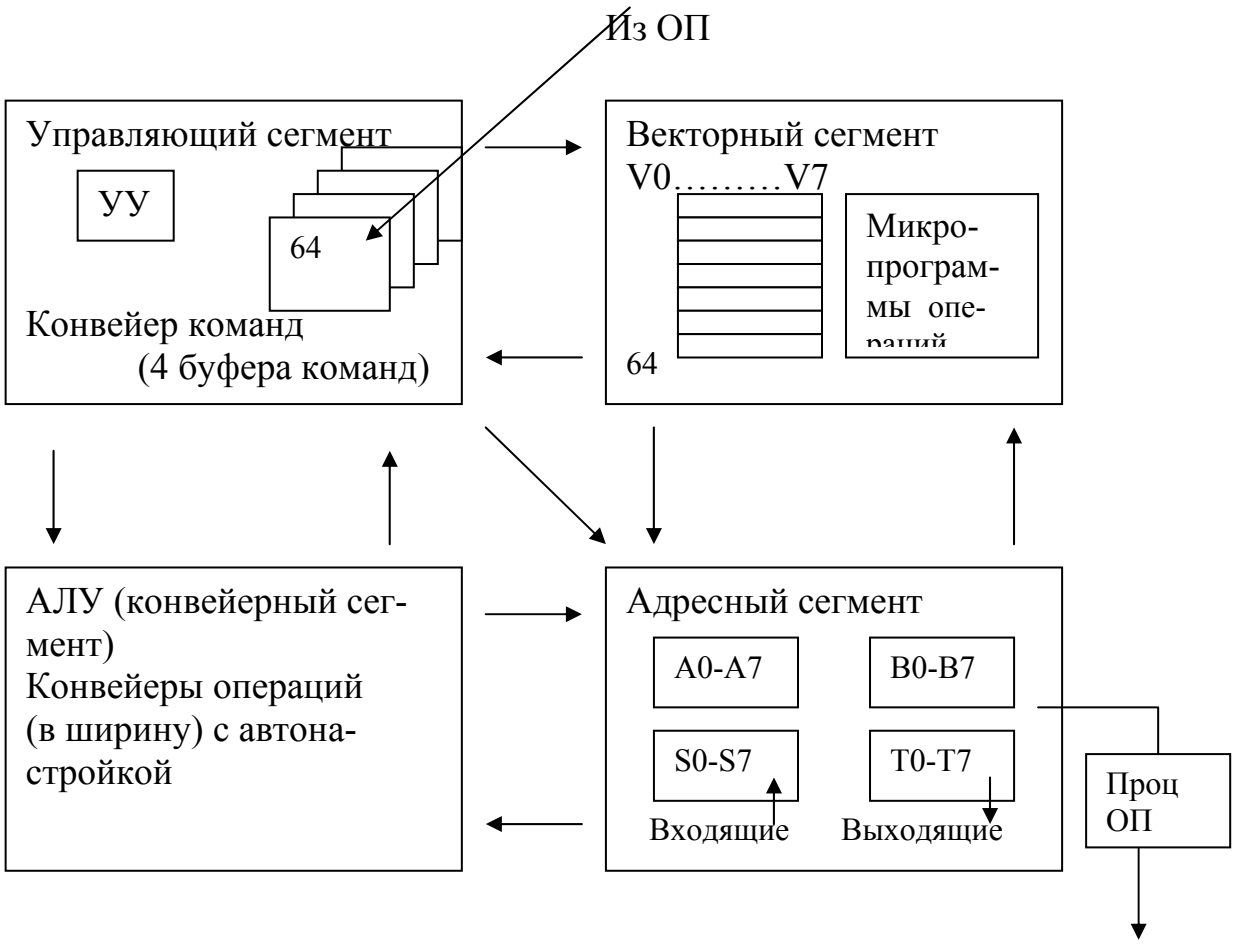


Рис 5.3

5.5 Производительность и области применения

CRAY процессор разрабатывался как специальный процессор с высокой производительностью для решения научно-технических задач. Это предопределило отсутствие развитых средств динамического перераспределения адресов, аппаратное базирование и наличие простых методов обработки прерываний и зашиты памяти.

Совокупность целого ряда оригинальных технических решений (регистровый ассемблер, прозрачная память, векторные регистры) и ориентация на максимальное повышение производительности позволили получить ощутимые результаты в области «научных» вычислений. В 1976 году процессор CRAY-1 был одним из наиболее быстрых процессоров - Cray-конвейерный сегмент FP имел цикл 12,5 нс. ~ 85 MFOPS или 250 MIPS

MIPS (Million Instructions Per Second)-Миллион операций в секунду

MFLOPS (Million Floating Point Operation Per Second)- миллион операций с действительными числами в секунду (плавающая точка)

6. ПРОЦЕССОР ПЕРЕСЫЛОК

6.1 Иерархия памяти в классической архитектуре

Классические принципы построения процессора предусматривают наличие развитой иерархии памяти, в которой внутренние регистры АЛУ не адресуются программой, быстрая память процессора представлена регистрами общего назначения с собственной адресацией, а выборка команд и данных происходит из оперативной памяти, имеющей сквозную адресацию. Схематично эта иерархия представлена на рис 6.1. В неймановских машинах пересылки данных между процессором и памятью выполняются довольно часто и поочередно, а так как ширина канала пересылки мала, то здесь образуется так называемое «узкое место фон-неймановской архитектуры».

Фон - Неймановская иерархия памяти

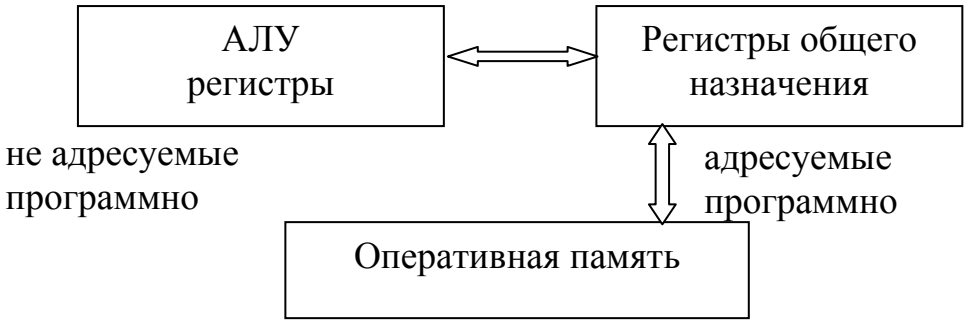


Рис 6.1

С другой стороны иерархия памяти приводит к тому, что в машинной команде применяется много разных способов адресации, при этом разные уровни иерархии рассматриваются логически как разные устройства.

6.2 Организация памяти в процессоре пересылок

Основной идеей разработчиков процессора пересылок был отказ от классической иерархии памяти в пользу логического объединения адресного пространства. Эта идея получила название сквозной адресации. При этом разная память может представлять собой физически различные устройства, но с точки зрения устройства управления процессором - это единое адресное пространство, имеющее единый способ адресации в машинной команде и каждый элемент такой сквозной памяти является программно доступным.

Таким образом, с точки зрения машинной команды можно единым образом адресовать и обращаться, как к специальным регистрам управления про-

цессором, внутренним регистрам схем выполнения машинных команд АЛУ, регистрам общего назначения (РОН), так и к словам оперативной памяти (ОП). Схема сквозного адресного пространства представлена на рис 6.2.

Сквозное адресное пространство в процессоре пересылок

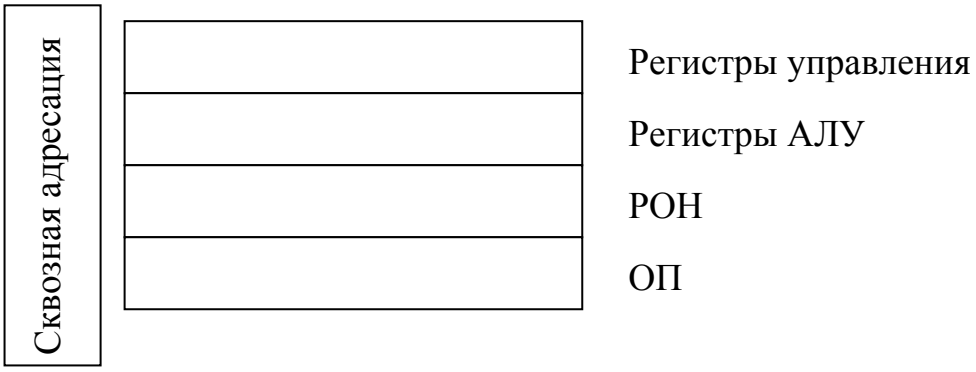


Рис 6.2

Такой подход к организации памяти порождает и специфический подход к организации работы процессора и механизму выполнения машинных команд.

6.3 Организация процессора пересылок

6.3.1 Адресная фиксация схем исполнения машинных команд

С учетом механизма сквозной адресации следующей идеей разработчиков было распределение схем выполнения машинных команд АЛУ со своими собственными регистрами операндов и результатов по фиксированным адресам сквозной памяти.

Адресная фиксация схемы АЛУ

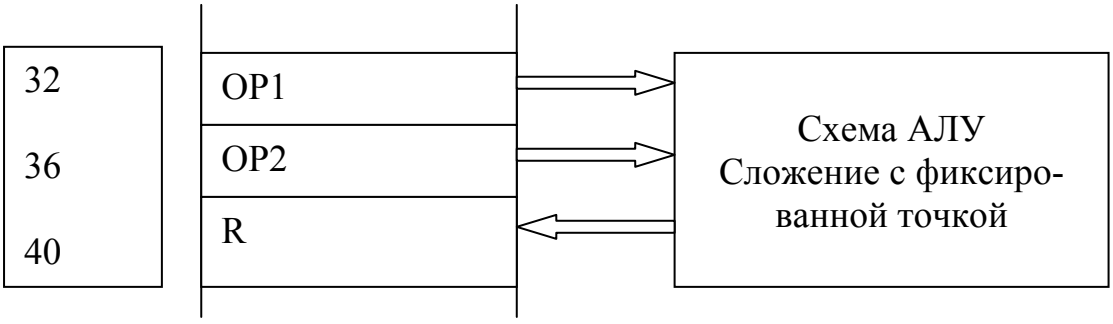


Рис 6.3

Таким образом, например, схема сложения чисел с фиксированной точкой, обладающая собственными регистрами двух операндов и регистром результата оказывалась фиксированной в определенных адресах сквозной памяти,

а именно в тех, которые назначались для регистров этой схемы. Вариант такой фиксации схемы АЛУ приведен на рис 6.3.

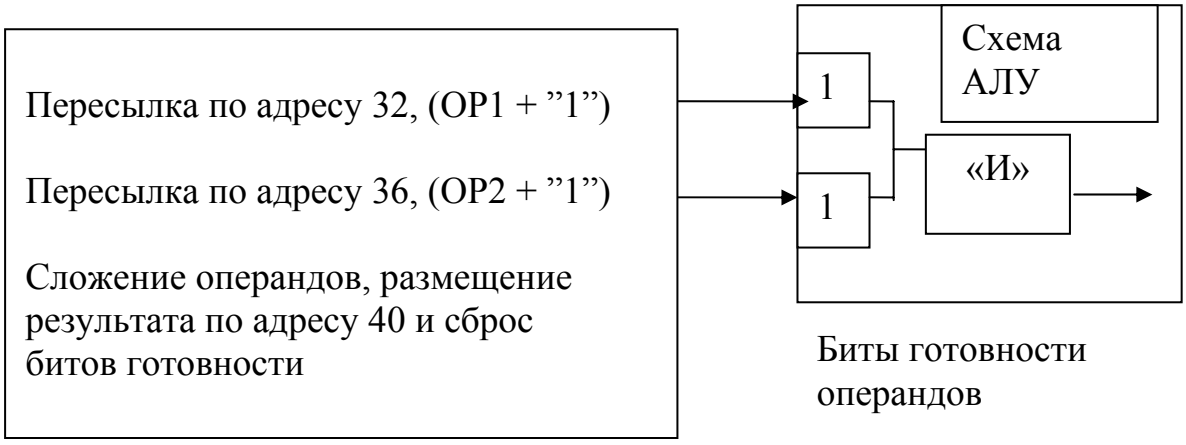
Таким образом область сквозного адресного пространства, соответствующая входным/выходным регистрам схем АЛУ оказывалась жестко разделена и закреплена за соответствующими схемами, и следовательно за машинными командами.

6.3.2 Механизм запуска машинной команды

Механизм запуска машинной команды предусматривал пересылку операндов в соответствующие входные регистры исполнительной схемы, что в принятой системе сквозной адресации реализовывалось пересылкой содержимого из одного слова памяти в другое. Именно этот механизм и дал название данной архитектуре - «процессор пересылок». Поскольку каждая машинная команда оказывалась жестко закрепленной за фиксированными адресами сквозной памяти, то запуск той, или иной операции был связан только с адресами расположения соответствующих входных регистров - процессор пересылок не имеет поля кода машинной команды, и следовательно устройство управления процессором выполняет только одну команду - команду пересылки слова. Механизм запуска машинной команды проиллюстрирован на рис 6.4

Запуск машинной команды в процессоре пересылок

Рис 6.4



Для синхронизации процесса выполнения машинных команд внутри схем АЛУ предусматривались биты готовности операндов, которые устанавливались в «1» после пересылки операнда команды. Наличие всех необходимых операндов запускало по схеме «И» выполнение команды, после чего результат помещался схемой в выходной регистр, а биты готовности операндов сбрасывались

в «0». Во время выполнения очередной команды УУ задерживало следующую команду пересылки до появления результата в выходном регистре.

6.4 Пример программы в процессоре пересылок

Рассмотрим фрагмент программы процессора пересылок для вычисления следующего арифметического выражения, в рамках стандартного понимания символических имен:

Y = ((a+b)*(c+d))

Будем считать, что схемы АЛУ для операций с плавающей точкой фиксированы на следующие адреса сквозной памяти (операнд 1, операнд 2, результат), а сами операнды имеют длину в 4 байта:

Сложение - 64, 68, 72;

Умножение - 76, 80, 84;

Фрагмент программы (операция пересылки справа на лево):

64, a; (пересылка первого операнда для сложения)

68, b; (пересылка второго операнда для сложения)
(выполнение команды сложения - результат по адресу 72)

64, 76; (пересылка результата, как операнда для умножения)

64, c; (пересылка первого операнда для сложения)

68, d; (пересылка второго операнда для сложения)
(выполнение команды сложения - результат по адресу 72)

64, 80; (пересылка результата, как операнда для умножения)
(выполнение команды умножения - результат по адресу 84)

Y, 84; (пересылка результата в оперативную память)

6.5 Реализация перехода по адресу и сравнения

Реализация операции сравнения в процессоре пересылок аналогично обычным арифметическим операциям - схемы сравнения фиксированы на определенные адреса сквозной памяти, но в поле результата схема помещает «0» или «1», в зависимости от результата сравнения операндов. Эта реализацию проиллюстрирована на рис 6.5 слева.

Более интересно реализован механизм выполнения команды перехода по адресу. Регистр адреса команды УУ так же является словом в едином адресном пространстве (и следовательно доступен программисту !) и расположен по адресу «0». По адресу «4» расположено поле смещения, которое или вычисляется

компилятором, и следовательно расположено в области загруженной программы, или вычисляется в программе. Со словами по адресам «0», «4» и «8» коммутирована схема целочисленного сложения. При установке битов готовности операндов, т. е. после пересылки смещения и результата сравнения, эта схема выполняет сложение смещения и текущего адреса команды, при условии, что слово по адресу «8» содержит «1», т.е. при истинности результата сравнения - рис 6.5 справа. Поскольку это приводит к модификации текущего адреса команды, то тем самым процессор выполняет переход на другую команду в программе.

Реализация сравнения и перехода по адресу в процессоре пересылок

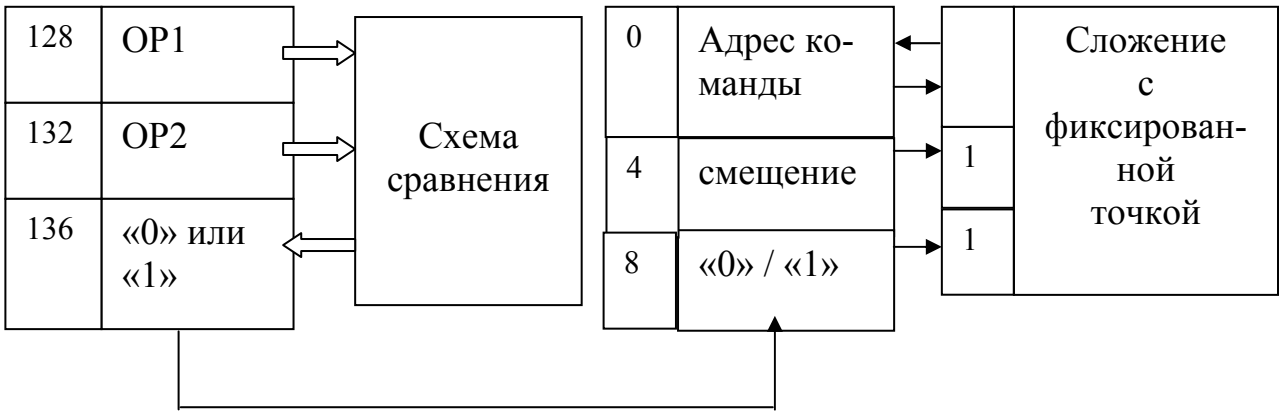


Рис 6.5

6.6 Замечания по реализации процессора пересылок

Основным достоинством данной архитектуры является независимость устройства управления от набора машинных команд и возможность универсально расширять этот набор путем включения регистров исполнительной схемы команды в сквозную память процессора. Недостатки архитектуры связаны с большим объемом пересылок данных, однако определенная часть этих пересылок - пересылки между регистрами схем АЛУ. Очевидно, что должны быть приняты определенные решения по гибкой адресации операндов ОП, что приводит к введению регистров базы и индекса.

Эта архитектура нашла свое применение в ряде специализированных процессоров, например в TMS 320 - процессоре обработки сигналов.

7. АРХИТЕКТУРЫ ПРОЦЕССОРОВ И ФОРМАТЫ ДАННЫХ

7.1. Процессоры с универсальным набором команд

В связи с необходимостью решения различных прикладных задач на ЭВМ общего назначения, уже начиная с машин второго поколения отчетливо наметилась тенденция к созданию универсального набора команд. Такой универсальный набор охватывал как разнообразные форматы данных, т.е. количество занимаемых объектом битов, так и различные типы данных, т.е. внутреннюю структуру формата данных. Наиболее широко используемые в рамках универсальных ЭВМ форматы и типы приведены в таб. 7.1.

Форматы	Типы
→ полуслово	числа с фиксированной точкой числа с плавающей точкой числа в двоично - десятичном представлении
→ слово	
→ двойное слово	
→ длинное слово	

Таблица 7.1 Форматы и типы данных

Идея универсального набора команд предполагала самостоятельную реализацию одинаковой обработки, например сложения, для разных форматов и типов в виде отдельных машинных команд. Таким образом возникало несколько машинных команд (до 10 и более) для одной операции обработки, в результате чего общий набор машинных команд имел порядок 150 - 200.

Такие процессоры получили название «CISC процессоры», т.е. процессоры с универсальным или общим набором команд.

7.2 RISC – процессоры

Для быстрого выполнения программы, написанной на языке высокого уровня, не нужны сложные машинные команды - гораздо более важно сократить время выполнения наиболее часто используемых команд. Этот принцип был положен в основу RISC-архитектуры, которая представляет собой улучшенный вариант неймановской архитектуры. Благодаря сокращению набора команд упрощаются аппаратные схемы, а значит, обеспечивается оптимизация выполнения часто используемых команд. Кроме того, за счет применения

большого числа регистров уменьшается частота (число) доступов к памяти, что также позволяет повысить скорость выполнения команды.

Таким образом основная идея RISC процессоров – малый фиксированный набор быстрых команды позволяет не только резко сократить набор машинных команд, отметим, что сокращение до 32 команд сокращает так же до 5 битов длину кода операции, но и сократить набор схем, реализующих команды, что позволяет при той же степени интеграции СБИС увеличить количество регистров и объем кэш-памяти.

Типичные представители этих машин: компьютер RISC Калифорнийского университета в Беркли, IBM 801, MIPS Станфордского университета, μ3L Университета шт. Юта, RIDGE 32 - фирмы Midge, Pyramid 90X фирмы Pyramid и др. RISC-архитектуру имеет и транспьютер фирмы «Инмос» - 32-разрядный процессор, спроектированный с оптимальным набором команд, позволяющим использовать язык высокого уровня Оккам.

7.3 Теговые машины

Одним из факторов, усложняющих разработку программного обеспечения, является наличие большого различия между понятиями операций и их объектов на языке программирования высокого уровня и понятиями операций и их объектов, определяемыми архитектурой компьютера. Это отличие носит название семантического разрыва. Иначе говоря, если на языке высокого уровня можно описать различные операции и типы данных, то в неймановской архитектуре разницы между программами и данными нет, как нет и разницы между типами данных. Это обстоятельство тяжелым бременем ложится на плечи программиста при составлении программы. И впоследствии оно является причиной усложнения отладки программы. Из-за отсутствия различий в типах данных и между программами и данными нельзя обнаружить, была ли ошибка связана с выполнением команды или с обращением к данным. Нельзя также обнаружить, выполняются ли данные в качестве команды или что к команде осуществляется обращение, как к данным.

Для решения этой проблемы Илифф предложил с помощью некоторого алгоритма добавлять ко всем данным информацию, необходимую для того, чтобы идентифицировать их как данные, использовать вместо линейного адресного пространства памяти структурированное пространство и добавлять к каждому элементу памяти информацию, показывающую атрибут этого элемента. Эта до-

полнительная информация получила название «тег». Машины, основанные на этом принципе, называют теговыми машинами. Так, различные типы данных характеризуются своими тегами, а однотипные команды, отличающиеся только типами операндов, никак не различаются. Например, программисту нет необходимости различать команды ADD (сложения) с плавающей и фиксированной запятой, как в CISC процессорах: машина это сделает автоматически, проверив типы операндов. В случае обращения с массивами данных добавляется такая теговая информация, как длина, ширина массива, индексы, а выход за пределы массива автоматически контролируется машиной при обращении к данным.

Майерс предложил проект SWARD - машины, в которой идеи теговых машин получили свое дальнейшее развитие.

В SWARD-машине данные представлены структурным элементом, называемым ячейкой. На рис. 7.1 приведен пример ячейки. Каждая ячейка состоит из поля тега и поля данных. Численные значения представлены в двоично-десятичном коде, и один разряд десятичного числа представлен четырьмя двоичными разрядами (битами). Память разбита на 4-разрядные единицы, которые называются признаками. Разрядность данных, указанных в поле тега, выражается числом признаков. Положительный знак числа выражается кодом 0000, отрицательный - 0001. Если используются массивы данных, то в поле тега указываются размерность массива, тип ячейки элемента массива, длина каждого измерения. Если элементом массива является целое число, то к типу ячейки после 1111 добавляется информация о длине числа. В SWARD-машине теговая информация используется не только применительно к данным, но и применительно к программным модулям.

Представление данных в теговых машинах

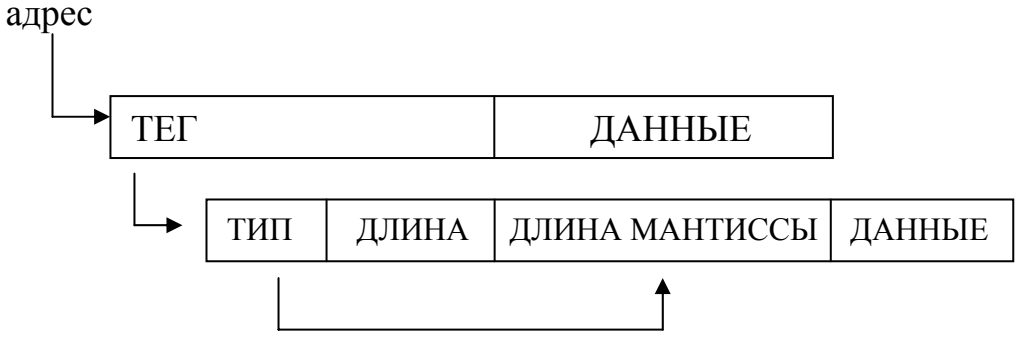


Рис 7.1

7.4 Гарвардская архитектура

Еще одна архитектурная идея, связанная с преодолением проблемы семантического разрыва, но теперь в части неразличимости программы и данных, основывается на физическом разделении оперативной памяти на два независимых блока с собственными устройствами управления. Предложенная в Гарвардском университете она получила название «Гарвардская архитектура». Схема такого процессора приведена на рис 7.2

Схема процессора с гарвардской архитектурой

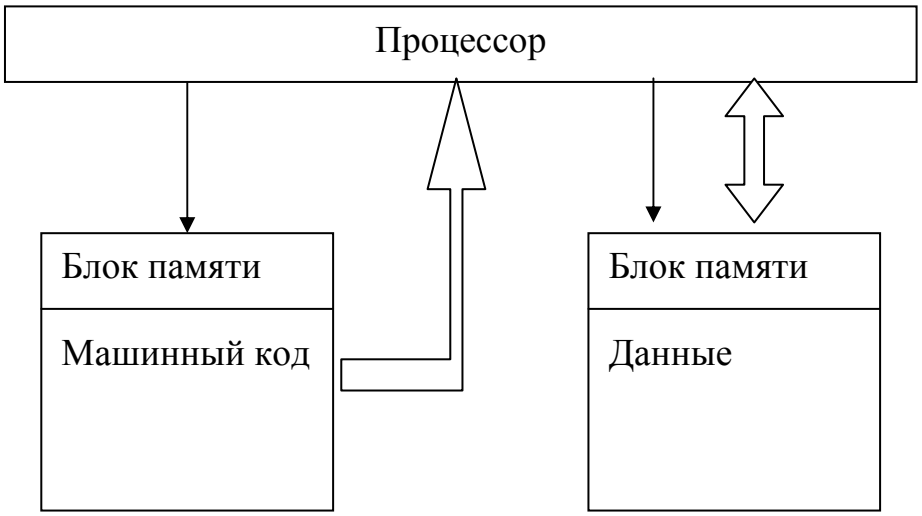


Рис 7.2

Два блока оперативной памяти для хранения программы и данных могут работать параллельно, что важно для конвейерной организации самого процессора. Для предотвращения возможности модификации программы во время выполнения (самомодифицируемые программы), что иногда активно использовалось при программировании в фон-неймановских процессорах, аппаратно запрещена операция записи в область машинного кода.

Такой подход к организации памяти широко используется в настоящее время в микропроцессорах, и в процессорах специального назначения, где чрезвычайно важно сохранить целостность программы, даже при возникновении аппаратной ошибки.

8. ПОДХОДЫ К ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА И ПОТОКОВЫЕ МАШИНЫ

Архитектура вычислительной машины во многом определяется принятой моделью обработки данных, т.е. подходом или принципом, в соответствии с которым организуется процесс вычислений. На современном этапе можно выделить следующие три основных подхода к организации вычислительного процесса:

8.1 Процедурное программирование.

Большинство вычислительных машин, существующих в настоящее время, относятся к так называемым неймановским ЭВМ, т. е. вычисления выполняются на основе принципа, который определил Дж. фон Нейман, называемым принципом процедурного программирования. Этот принцип требует, чтобы в процессоре было устройство управления, содержащее программный счетчик, указывающий текущую команду, чтобы команды (указанные программным счетчиком) последовательно считывались и декодировались по заранее заданному в виде программы алгоритму вычислений, вычисления выполнялись в операционном устройстве и данные последовательно перезаписывались в запоминающее устройство.

Особенности принципа работы машин неймановского типа можно определить следующим образом: - последовательное выполнение программы при централизованном управлении с помощью программного счетчика и обработка данных с перезаписью содержимого памяти и регистров.

Прежде всего, при последовательной обработке в неймановской машине скорость обработки определялась быстродействием элементов, что ограничивало производительность ЭВМ. Поэтому для реализации высокой производительности при существующем ограничении на скорость обработки, обусловленной элементной базой, ничего не оставалось, как использовать параллельную обработку.

Кроме того, в программировании, основанном на концепции перезаписи памяти, соответствие между переменной и данными, которые являются ее значением, не обязательно определено однозначно и порядок перезаписи оказывает большое влияние на смысл программы. Это является причиной возникновения ошибок в программе, поэтому при составлении программы нужно быть предельно внимательным. В результате все это приводит к снижению произво-

дительности программного обеспечения. В этой связи возникла необходимость отказа от подобной концепции перезаписи памяти и соответствующего алгоритма организации вычислительного процесса.

8.2 Функциональное программирование

Вычислительная модель, в которой программа рассматривается как множество определений функций, называется функциональной моделью. Для описания функциональных моделей используются два метода: первый основан на использовании аппликативного языка, а второй - на использовании языка с однократным присваиванием. Отличительной чертой этих моделей является то, что в основу их положена простая и четкая математическая модель, называемая «лямбда-исчислением».

Рассмотрим основные понятия «лямбда - исчисления». В выражении *f(x),* которое используется обычно для представления функции, не ясно: то ли оно означает функцию *f,* то ли ее значение при заданном значении параметра *x.* Поэтому для четкого описания функции *f* было введено выражение *λxf(x).* То есть, когда выражение *M* хотят рассматривать как функцию от *x,* следует использовать запись *λxM.* Получение выражения *λxM* из выражения *M* называют «лямбда - абстракцией». Таким образом, выражение *λx(x+y)* является функцией от *x,* а не от *y.* При этом *x* называется связанной переменной, а *y* - свободной переменной. Если f = *λxM,* то подстановка выражения *A в x* внутри *M* называется применением *A* к f и записывается как *fA.* Вычисление выражения в этой модели носит название редукции.

Примерами языков программирования, реализующими вычисления на основе функциональных моделей являются для аппликативного языка «чистый» Лисп, предложенный Маккарти, FP Бэкуса и др., к языкам с однократным присваиванием относятся Id Арвинда , VAL Аккермана и Денниса др.

8.3 Потоковое программирование

Основная идея потокового программирования или модели вычислительного процесса по потоку данных основана на рассмотрении операции обработки данных, активируемой этими данными. Действительно реальное выполнение некоторой операции возможно только тогда, когда мы получаем исходные данные для этой операции. Тем самым данные, передаваемые из одной операции к другим, активируют соответствующие операции.

Потоковая обработка базируется на принципе выполнения программы, называемом управлением по данными. Принцип управляемости потоком данных гласит: «Все операции выполняются только при наличии всех операндов (данных), необходимых для их выполнения». В программе, используемой для потоковой обработки, описывается не поток сигналов управления, а поток данных.

Обработка, управляемая потоком данных, исходя из описанного выше принципа, отличается от обработки неймановского типа следующими моментами.

1) Операцию со всеми операндами (с имеющимися операндами) можно выполнять независимо от состояния других операций, т. е. появляется возможность одновременного выполнения множества операций (параллельная обработка).
2) Обмен данными между операциями четко определен, поэтому отношение зависимости между операциями обнаруживается легко (функциональная обработка).
3) Поскольку управление операциями осуществляется посредством передачи данных между ними, то нет необходимости в управлении последовательностью выполнения и, кроме того, нет необходимости в централизованном управлении (распределенная обработка).

Описание вычислительного процесса в машине потоков данных может быть представлено в виде графа, в котором вершины суть операции обработки данных, а дуги - процессы пересылки данных, полученных в результате обработки, с помощью которых происходит активация следующих вершин, как это показано на рис 8.1.

Граф вычислительного процесса в схеме потока данных

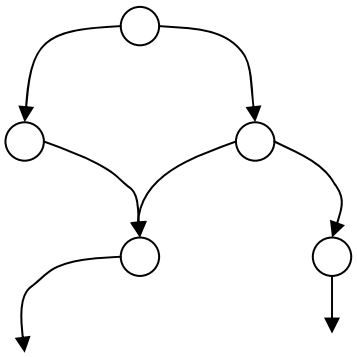


Рис 8.1

Реализация такой идеи приводит к появлению понятия командной ячейки, которая должна хранить код операции обработки, непосредственные операнды

операции с их битами готовности и адресные поля, указывающие поля операндов других командных ячеек, в которые отправляется результат данной операции. Схема командной ячейки приведена на рис 8.2.

Схема командной ячейки

Рис 8.2

Поскольку результат одной операции может активировать несколько других командных ячеек, то для фиксации длины командной ячейки вводится дополнительная операция пересылки. Она имеет один операнд и два адресных поля отсылок, что позволяет, комбинируя командные ячейки пересылок реализовывать отправку поля данных в несколько командных ячеек, как это показано на рис 8.3.

Множественная рассылка результата

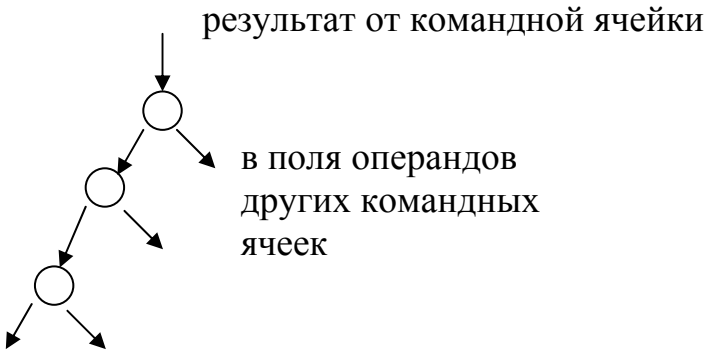


Рис 8.3

Для иллюстрации управления командными ячейками с помощью битов готовности рассмотрим вычисление следующего арифметического выражения:

Y = (a+b) / (a*b);

Командная ячейка, выполняющая операцию деления, будет активирована, когда в ее поля операндов поступят результаты от командных ячеек выполняющих операции сложения и умножения. Наличие двух битов готовности операндов приведет к установке бита готовности самой командной ячейки и, следовательно, она может быть выбрана для выполнения в процессоре. Эта ситуа-

ция приведена на рис 8.3

Активация командной ячейки

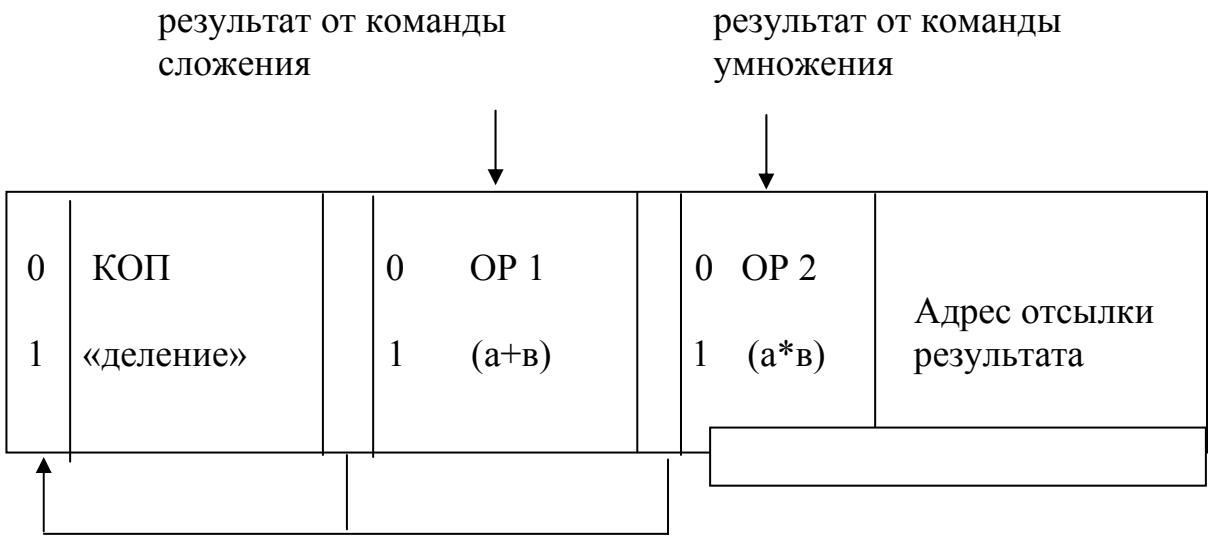


Рис 8.3

В общем виде машина потоков данных должна содержать память командных ячеек, процессор, выполняющий активированные командные ячейки и два специальных устройства, называемых арбитражной и распределительной сетью. Назначение арбитражной сети - отслеживать готовые (активированные) командные ячейки и передавать их на выполнение процессору. Назначение распределительной сети - размещение полученного результата в поля операндов других командных ячеек, включая выполнение команды пересылки. Структура машины потоков данных приведена на рис 8.4.

Структура машины потоков данных

Рис 8.4

9. АРХИТЕКТУРЫ ПАМЯТИ

9.1 Классификация архитектур памяти

В любой вычислительной машине с любой процессорной архитектурой программы и данные хранятся в памяти. Объем памяти и скорость доступа определяют размеры задач, которые можно решать на этой машине и скорость обработки данных, особенно для архитектур с большой частотой перезаписи данных.

Компромисс объема и скорости доступа достигается введением иерархии памяти, включающей запоминающие устройства разных типов. Однако такое решение по архитектуре памяти приводит к необходимости особого программирования различных запоминающих устройств.

Могут быть предложены различные архитектуры памяти, повышающие наблюдаемое быстродействие, без изменения элементной базы реализации запоминающих устройств, а именно:

a) архитектуры быстродействующей адресной оперативной памяти:
   i) - чередование адресов;
   ii) - иерархические структуры (кэш - память);
   iii) - сквозная адресация (процессор пересылок);
b) архитектура памяти большой емкости (дисковая память);
c) архитектура виртуальной памяти;
d) архитектура общей памяти (для многопроцессорного доступа)
e) архитектура интеллектуальной памяти (ассоциативная память).

Выбор той, или иной архитектуры обусловлен требованиями, предъявляемыми к вычислительной машине, однако в настоящее время одно из них - требование обеспечения высокой надежности становится достаточно общим.

Высокая надежность определяется двумя факторами - количеством ошибок чтения/записи и защитой информации в памяти от несанкционированного доступа. Снижение уровня ошибок чтения/записи достигается применением кодов, обнаруживающих и исправляющих ошибки. Защита информации может быть обеспечена либо применением криптографических систем, либо путем задания каждому вычислительному процессу условий, разрешающих обращение только к определенным данным. Такие условия носят названия мандата, а механизм разграничения доступа, использующий мандат, носит название механизма с мандатной адресацией.

9.2 Память с чередование адресов

Архитектура быстрой памяти с чередованием адресов возникла для сглаживания различия в скорости между конвейерным процессором с конвейером команд и конвейером данных и обычной адресной оперативной памятью. Анализ обращений в память, особенно при обработке массивов, показывает, что доля обращений с последовательно увеличивающимися адресами достаточно значительна. Для согласования с конвейером необходимо, что бы было реализовано упреждающее чтение в быструю регистровую память для последовательных адресов. Такое упреждающее чтение и реализовано в архитектуре с чередованием адресов.

Основная идея состоит в том, что адресное пространство разделяется между банками оперативной памяти так, что соседние слова располагаются в разных банках. При обращении по некоторому адресу все эквивалентно адресованные слова всех банков считываются в быструю специальную память, называемую фиксатором. При обращении к следующему слову содержимое извлекается из фиксатора без обращения к самой оперативной памяти. Схема такой памяти приведена на рис 9.1

Доступ к памяти с использованием чередования адресов

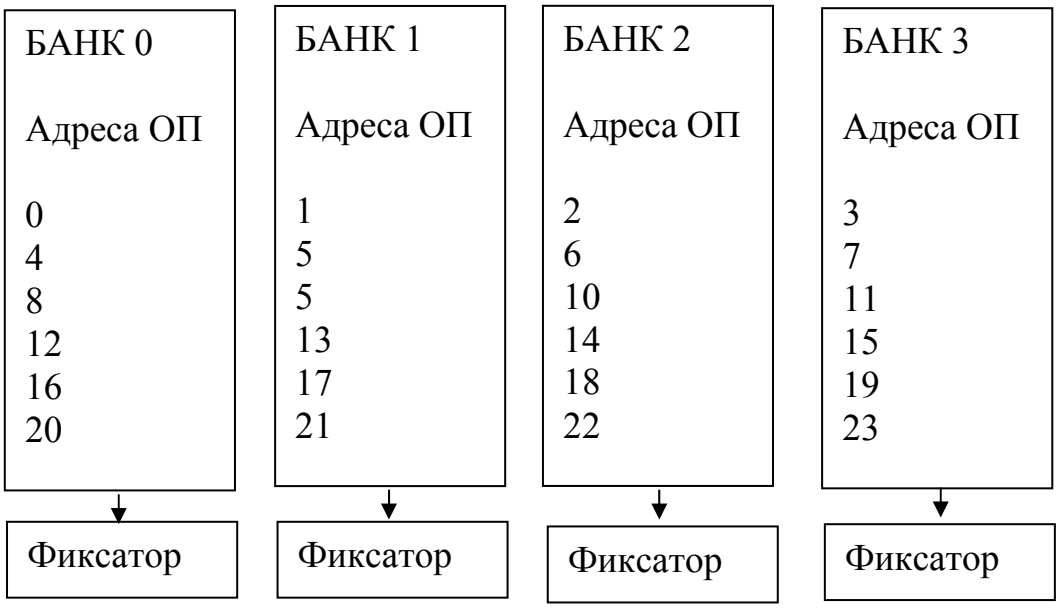


Рис 9.1

Рассмотрим более подробно реализацию архитектуры памяти с чередованием адресов: Разделим память на m банков $B_0, B_1, \ldots, B_{m-1}$ одинаковой емко-

сти и установим связь между банком B, содержащим адрес *i*, и адресом *d* внутри банка B_k, соответствующим адресу *i,* следующим равенством:

$$l = d * m + k, \quad d > 0, \quad 0 < k < m—1;$$

Распределение адресов между m банками памяти называется m-кратным чередованием адресов памяти. Таким образом, память, состоящая из m банков с распределенными адресами, называется памятью с m-кратным чередованием адресов (m — обычно степень числа 2). Очевидно, что при обращении в ОП по адресу 5 в фиксаторы будет считано из банков содержимое по адресам 4, 5, 6 и 7, и при следующем обращении к соседнему адресу содержимое будет считано из фиксатора.

Таким образом, эффективность памяти с чередованием адресов напрямую зависит от количества банков.

9.3 Кэш память

Кэш-память - это быстродействующая память, расположенная между ЦП и основной памятью (рис. 9.2). Вместе с основной памятью она образует иерархическую структуру, и ее действие эквивалентно быстрому доступу к основной памяти. В универсальных ЭВМ, основная память которых имеет емкость порядка 132 - 512 Мбайт, обычно используется кэш-память емкостью 64—512 Кбайт. То есть емкость кэш-памяти составляет порядка 1/1000—1/500 емкости основной памяти, а быстродействие в 5 - 10 раз выше быстродействия основной памяти.

Кэш - память

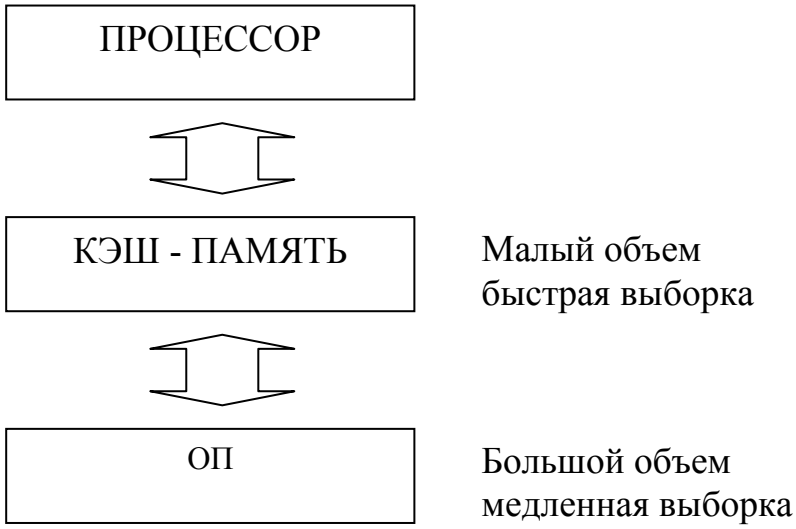


Рис 9.2

Кэш-память, состоящая из m слов, сохраняет копии не менее чем m слов из всех слов основной памяти. Если копия, к адресу которой был выполнен доступ, существует в кэш-памяти, то считывание завершается уже при доступе к кэш-памяти. Для записи в кэш-память существует несколько методов замещения старой информации. Эти методы называются стратегией обновления основной памяти. В случае пространственной локальности основная память разбивается на блоки с фиксированным числом слов и обмен данными между основной памятью и кэш-памятью осуществляется блоками. При доступе к некоторому адресу процессор должен сначала определить, имеется ли копия блока, содержащего этот адрес, в кэш-памяти, и если имеется, то определить, с какого адреса кэш-памяти начинается этот блок. Эту информацию процессор получает с помощью механизма преобразования адресов. На сложность этого механизма существенное влияние оказывает стратегия размещения, определяющая, в какое место кэш-памяти следует поместить каждый блок основной памяти.

На эффективность кэш-памяти большое влияние оказывают пространственная и временная локальности. Кроме того, программы и данные существенно отличаются по локальности? поэтому иногда для каждого типа данных, которые имеют различную локальность, используют различные кэш-памяти.

Стратегия замещения в кэш-памяти - это метод (алгоритм), определяющий для заполненной кэш-памяти, какой из блоков следует возвратить в ОП для освобождения блока кэша с целью помещения в него блока ОП, к которому сейчас выполняется обращение. Предложены различные стратегии замещения, использующие генератор случайных чисел, информацию о наименьшей частоте использования блока, информацию о временных параметрах обращения к блоку, а так же метод прямого наложения блоков.

9.4. Ассоциативная память (безадресная память)

Ассоциативная память представляет собой хранилище данных, в котором обращение к элементам (словам) происходит по полю ключа, хранящегося вместе с данными. Схема сравнения (компаратор) выполняет побитовое сравнение входного ключа со значениями ключей в словах ассоциативной памяти. В результате оказываются выбранными те слова памяти, которые имеют аналогичный ключ. Схема ассоциативной памяти приведена на рис 9.3

Схема ассоциативной памяти

Рис 9.3

Для выполнения операции поиска свободных слов и поиска по некоторым битам поля ключа в устройство управления ассоциативной памяти включается регистр маски, биты которого указывают, какие биты регистра ключа (регистра ассоциативных признаков) должны сравниваться компаратором с битами ключей слов ассоциативной памяти. Использование регистра маски проиллюстрировано на рис 9.4

Регистр маски в структуре ассоциативной памяти

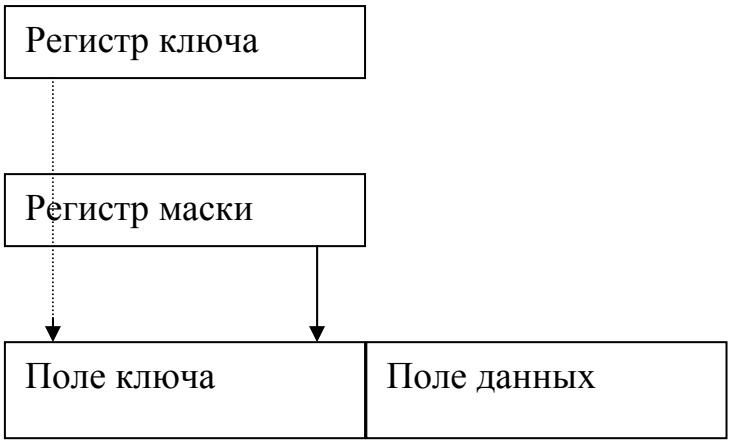


Рис 9.4

Идей ассоциативной памяти были использованы в процессоре STARAN. Это матричный процессор, способный выполнять ассоциативный поиск с параллельным сравнением разрядов и последовательным сравнением слов и наоборот, т.е. с параллельным сравнением слов и последовательным сравнением разрядов по отношению к 256 процессорам, объединенных в единую матрицу.

10. АРХИТЕКТУРНЫЕ РЕШЕНИЯ ВВОДА/ВЫВОДА ДАННЫХ

10.1 Проблемы организации и управления вводом/выводом

Проблемы организации ввода/вывода данных всегда занимали особое место в ряду проблем, решаемых разработчиками ЭВМ. Разнообразие самих устройств и их функционального назначения (долговременное хранение информации, визуализация, получение твердых копий, и т.д.) с одной стороны, и существенное различие устройств по скорости выполнения операций с другой стороны, были существенным препятствием на пути создания универсального механизма управления вводом/выводом.

Кроме того, в устройствах ввода/вывода, как ни в каких других устройствах ЭВМ, возникает значительное количество особых ситуаций, связанных с готовностью устройства, готовностью носителя и т.д., которые специфичны для данного конкретного устройства и требуют, следовательно, специальной обработки. Другой важной проблемой является синхронизация работы самих устройств, их контроллеров и устройства управления процессором. Возникают вопросы и плане взаимодействия процессора и контроллеров устройств при запуске, выполнении и завершении операций. Таким образом, можно выделить следующие проблемы, требующие решения в рамках архитектуры ввода/вывода:

1) общее управление;
2) запуск операции ввода/вывода;
3) идентификация и обработка особых ситуаций;
4) завершение операций ввода/вывода;
5) синхронизация;
6) подключение новых устройств.

10.2 Основные архитектурные решения

Исторически первым архитектурным решением по организации ввода/вывода была предложенная уже во втором поколении ЭВМ идея введение специализированных процессоров ввода/вывода (каналов), способных управлять различными по быстродействию внешними устройствами, что позволило освободить процессор для выполнения основной обработки. Такое решение получило название ввода/вывода с канальной архитектурой.

Стремление разработчиков получить универсальный механизм подключения устройств, совместно с наметившейся тенденцией увеличения разрыва между быстродействием процессора и скоростью устройств ввода/вывода, послужило толчком к разработке и внедрению универсального механизма подключения - обшей шины, обслуживаемого процессором.

Достаточно интересной является и архитектура, предусматривающая объединение в рамках сквозной адресации собственной памяти (буферов команд и буферов данных) устройств ввода/вывода - архитектура с общей памятью.

10.3 Канальный ввод/вывод

Основная идея канальной архитектуры - наличие специального процессора ввода/вывода – канала. В этой архитектуре предусматривается наличие в оперативной памяти специальной программы канала, которую канал, после активации центральным процессором выбирает из ОП в собственный буфер, что показано на рис 10.1.

Взаимодействие процессора и канала в канальной архитектуре

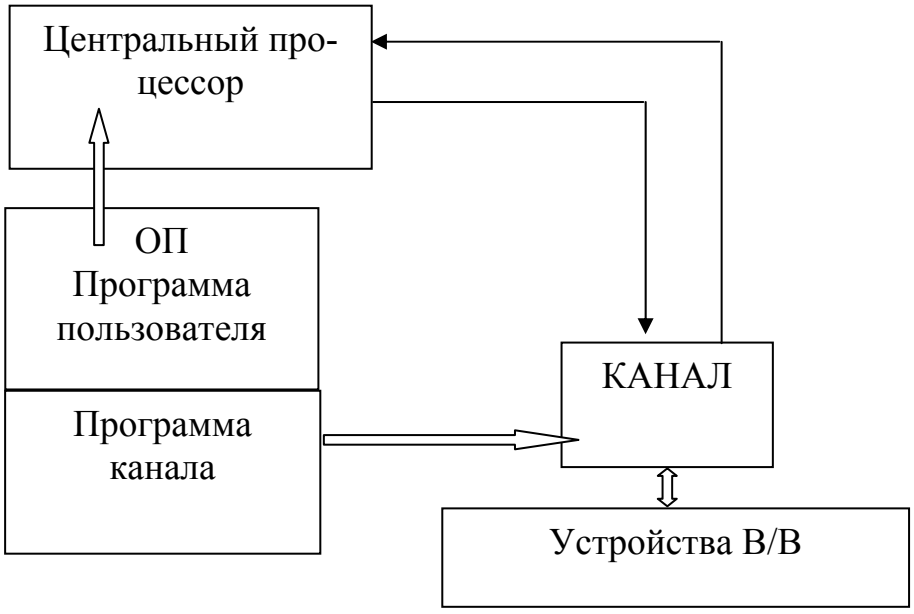


Рис 10.1

В момент выполнения каналом операции ввода/вывода центральный процессор может выполнять обработку данных, что повышает наблюдаемую производительность ЭВМ за счет совмещения во времени работы процессора и канала. По завершении операции канал с помощью механизма прерывания сигнализирует процессору о завершении канальной программы и передает слово состояния канала, содержащее информацию о условиях выполнения операции.

Обработка завершения операции ввода/вывода возлагается в канальной архитектуре на операционную систему.

К недостаткам данной архитектуры можно отнести необходимость особого канального программирования и синхронизации устройства управления процессором, нескольких каналов и специальных программ обработки особых ситуаций ввода/вывода в операционной системе - обработчиков канальных прерываний.

10.4 Архитектура с общей шиной

В ЭВМ четвертого поколения, особенно в персональных компьютерах, где проблема универсализации подключений выходит на первый план, используется архитектура с общей шиной. Соединение всех устройств обеспечивается с помощью общей шины, представляющей собой линии передачи данных, сигналов управления, адресов и питания. Эти магистрали получили название шины данных, шины адреса и шины управления, как это представлено на рис 10.2.

Архитектура с общей шиной

Рис 10.2

Единая система аппаратурных соединений значительно упростила структуру, сделав ее более децентрализованной. При этом все передачи данных по шине осуществляются процессором, управляемым сервисными программами. Подключение внешних устройств обеспечивается через соответствующие адаптеры или контроллеры - специальные устройства для согласования скоростей работы сопрягаемых устройств и управления периферийной аппаратурой.

В настоящее время наметилась тенденция к еще большей децентрализации, проявляющаяся в том, что контроллеры внешних устройств функционально приобретают черты специализированных процессоров ввода/вывода и снаб-

жаются значительными по объему буферами памяти. Следующая отмечаемая тенденция - появление иерархии и специализации шин - системная шина, локальная шина, периферийная шина.

Синхронизация в архитектуре с общей шиной может быть обеспечена как использованием аппарата прерываний, так и временным опросом контроллеров со стороны центрального процессора. Обработка особых ситуаций функционально возложена на операционную систему, получающую информацию об операции ввода/вывода по шинам управления и данных.

10.5 Архитектура ввода/вывода с общей памятью

Стремление универсально выполнять операции доступа, как к внешним, так и к внутренним устройствам привела в рамках архитектуры процессора пересылок к идее объединения адресного пространства не только внутри процессора и оперативной памяти, но и на уровне буферов устройств ввода/вывода - рис 10.3.

Архитектура ввода/вывода с общей памятью

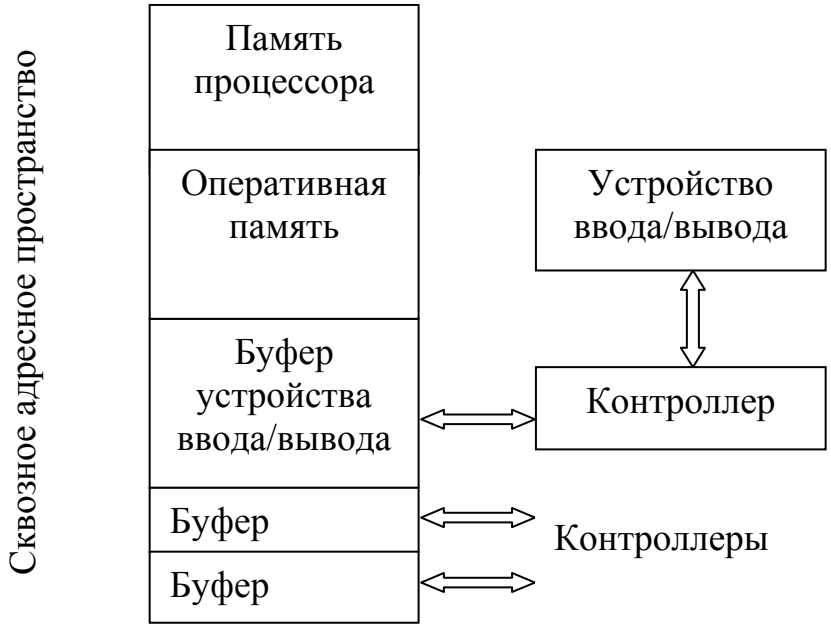


Рис 10.3

Таким образом, процессор обращается к устройству, просто пересылая данные в соответствующий буфер. Однако такая архитектура требует ряда специальных решений по синхронизации и обработке особых ситуаций.

11. ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА

Несмотря на существование различных архитектур процессоров, одной из самых привлекательных идей повышения наблюдаемой скорости обработки, при фиксированной элементной базе и тактовой частоте, была и остается идея параллельной обработки. Отметим сразу, что для эффективности параллельной обработки необходимо, что бы число одновременно поступающих в систему задач было достаточно велико. Более точно это означает, что коэффициент загрузки системы должен быть близок к единице.

В рамках этой идей предложено и реализовано много различных вариантов, обладающих разными характеристиками и имеющих различные области применения.

11.1 Мультипрограммирование (многозадачность)

Идея мультипрограммирования связана исторически прежде всего с совершенствованием операционных систем в направлении обеспечения «одновременного» выполнения на однопроцессорной ЭВМ потока разнородных задач. Исторически это было обусловлено большими ЭВМ второго и третьего поколений, работавших в основном в режиме пакетной обработки. Рассмотрим различные варианты обеспечения многозадачного режима:

11.1.1 Однопроцессорная обработка

Схемы обработки потока разнородных задач на ЭВМ с одним процессором различаются в зависимости от характера потока задач и принятых решений по организации ввода/вывода:

В ситуации, когда поток задач характеризуется превалирующими процессорными вычислениями (группа научно-технических задач), реальный параллелизм практически невозможен. Псевдопараллельность в этом случае может быть обеспечена путем реализации операционной системой принципа управления задачами методом квантования времени - рис 11.1. При этом каждая задача (программа) получает ресурс процессора на фиксированное время (квант), и после прерывания по интервальному таймеру операционная система передает управление следующей задаче, находящейся в оперативной памяти. Очевидно, что такой подход не приводит к наблюдаемому сокращению суммарного времени обработки.

Многозадачность в режиме квантования времени

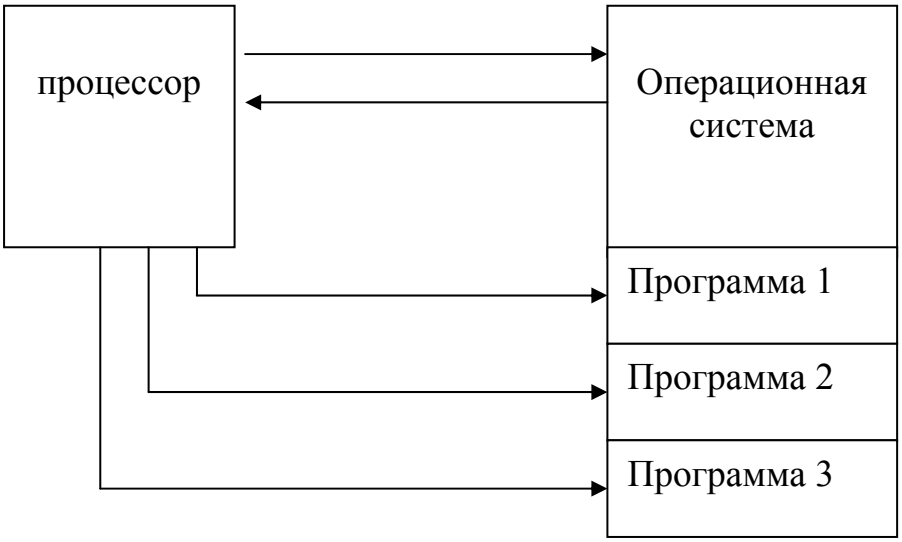


Рис 11.1

В ситуации, когда поток задач характеризуется как процессорной обработкой, так и значительным (по времени) вводом/выводом, и в ЭВМ реализована канальная архитектура, параллельность обработки обеспечивается совмещением во времени работы процессора и канала. Такая обработка пакета заданий носит название мультипрограммирования и существенно опирается на канальную архитектуру.

Совмещение во времени работы процессора и канала для двух программ иллюстрировано на рис 11.2

Мультипрограммирование в канальной архитектуре

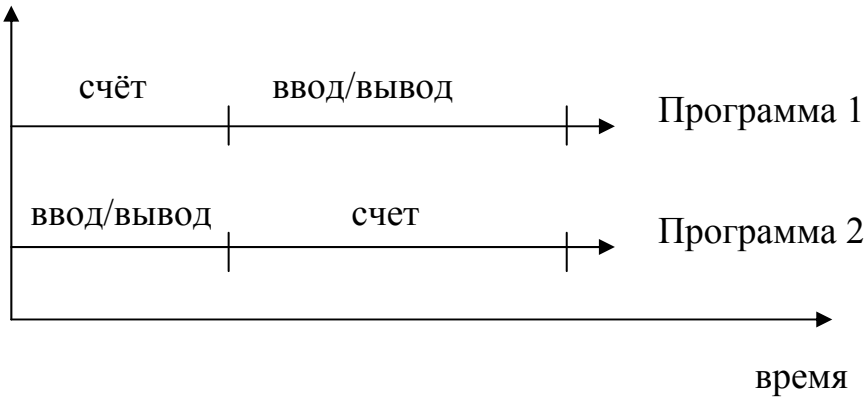


Рис 11.2

Реально при наличии нескольких каналов ввода/вывода и эффективного потока задач может быть получено значительное сокращение наблюдаемого со-

вокупного времени обработки пакета заданий по сравнению с суммой времен одиночного (вне пакета) выполнения задач.

11.1.2 Многопроцессорная обработка

Наличие нескольких обрабатывающих устройств (АЛУ или процессоров) позволяет реализовать «настоящую» параллельную обработку заданий, при этом можно рассматривать несколько вариантов:

Вариант с раздельной оперативной памятью подразумевает фактически многомашинную систему, когда несколько ЭВМ (процессор + ОП) находятся под общим управлением диспетчера задач, распределяющего поток заданий - рис 11.3

Многомашинная система с диспетчером задач

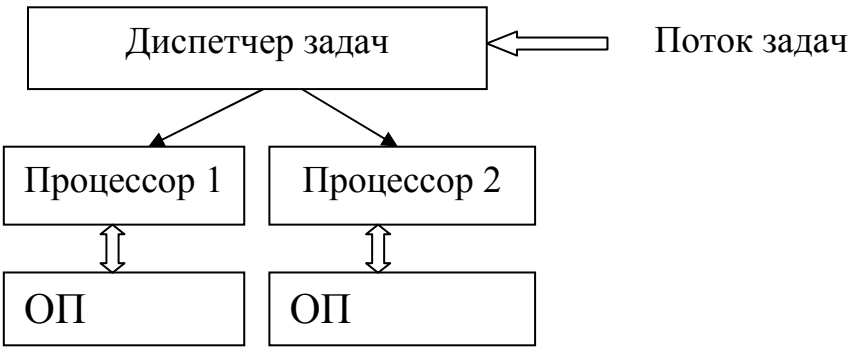


Рис 11.3

Вариант с общей оперативной памятью предусматривает наличие развитой операционной системы, выполняющей функции диспетчера задач, диспетчера оперативной памяти и диспетчера процессоров - рис 11.4.

Многопроцессорная система с общей памятью

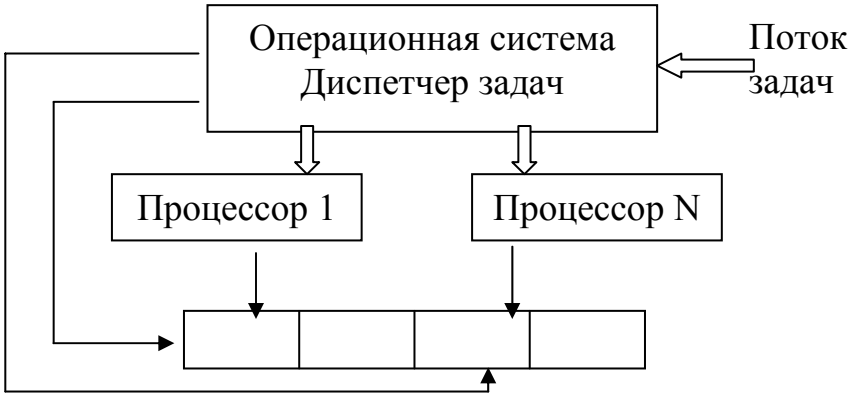


Рис 11.4

11.2 Параллелизм независимых ветвей

Если режим мультипрограммирования подразумевал наличие потока независимых задач, то параллелизм независимых ветвей - это совмещение во времени этапов выполнения *одной* задачи. Если представить функциональные независимые этапы решения задачи в виде графа вычислительного процесса, в котором вершинами являются этапы, а дугами - связи по управлению и данным, то очевидно, что не все задачи обладают свойством параллелизма ветвей - рис 11.5.

Определение параллелизма ветвей на основе графа вычислительного процесса задачи

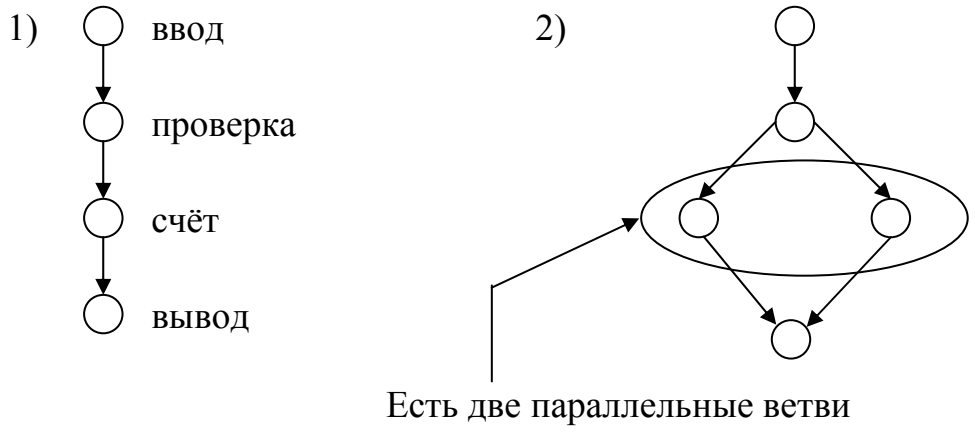


Рис 11.5

Более внимательное изучение проблемы параллелизма ветвей показывает, что необходимо выполнение следующих условий для параллельного выполнения этапов:

1) нет связи по данным - результаты одного этапа не являются входом другого;
2) нет связей по уравнению - один этап не передает управление другому;
3) нет общих ячеек памяти по записи - этапы не производят запись по одному и тому же адресу памяти.

Реализация параллелизма ветвей связана с решением проблемы программирования этапов и выделения ветвей, причем условия параллелизма должны быть обеспечены программистом. Решение второй проблемы - проблемы запуска параллельных ветвей на многопроцессорной системе возлагается на операционную систему.

Она должна распределять параллельные ветви по процессорам в соответствии с графом вычислительного процесса задачи.

11.3 Параллелизм объектов

Под параллелизмом объектов понимается ситуация, при которой различные по значениям, но однородные по структуре данные подвергаются одинаковой обработке (по одинаковой программе) на многопроцессорной системе. Такую обработку можно реализовать как в ширину, так и в глубину.

Параллелизм в ширину подразумевает использование каждого процессора для выполнения всех этапов решения задачи - рис 11.6

Параллелизм объектов - реализация «в ширину»

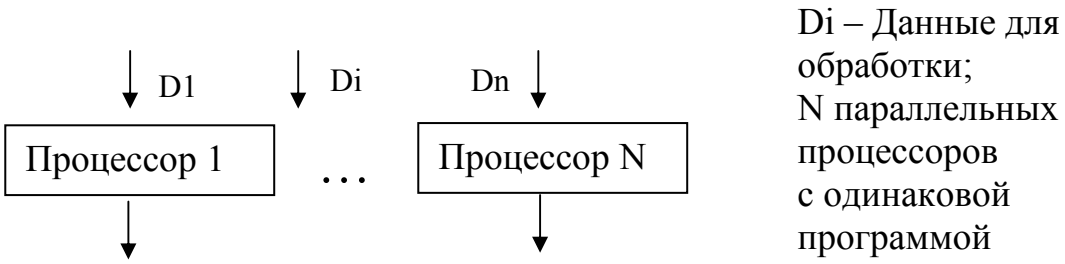


Рис 11.6

Параллелизм в глубину предполагает конвейерную обработку потока данных, при которой каждый процессор конвейера реализует определенный этап обработки данных, т.е. единая программа обработки распределена между процессорами, объединенными в конвейер данных - рис 11.7.

Параллелизм объектов - реализация «в глубину» - конвейер этапов

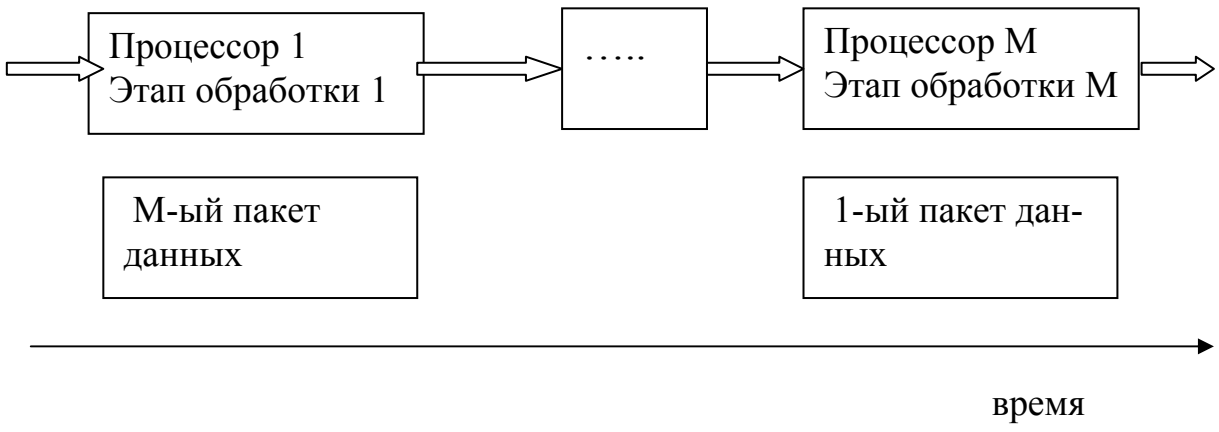


Рис 11.7

Этот механизм обработки представляет собой классический конвейер данных, в котором объектами являются крупные этапы решения задачи.

12. МАТРИЧНЫЕ СИСТЕМЫ

Под матричными системами или матричными процессорами обычно понимается многопроцессорная система, в которой процессоры с помощью той или иной сети связи объединены в матрицу. Задача устройства управления матричным процессором совместно с операционной системой - эффективная загрузка матрицы процессоров и эффективная (быстрая) передача промежуточных результатов. В качестве объектов параллелизма могут выступать этапы решения задачи, машинные команды или даже битовые операции, реализующие машинную команду.

Предложено и реализовано значительное разнообразие матричных систем, равно как и сетей связи в матричных процессорах. Рассмотрим более подробно предложения, относящиеся к разным уровням объектов параллелизма:

12.1 Однородные системы - параллелизм этапов задач

В этой матричной системе элементами матрицы являются полные процессоры с собственной оперативной памятью. Вершины графа вычислительного процесса (этапы) задачи распределяются между процессорами матрицы, а указанные графом связи по управлению и данным реализуются с помощью соединительной сети. Эффективность функционирования такой системы требует однородности этапов (подзадач) по времени выполнения и однородности этапов по требуемым ресурсам, что и объясняет название - однородные матричные системы. Т.е. однородность аппаратной реализации должна быть согласована с однородностью подзадач.

При условии, что на любом процессоре однородной матричной системы каждая подзадача выполняется за одинаковое время, параллельные ветви в графе вычислительного процесса будут завершены одновременно. Выполнение этого условия обеспечивает эффективность использования однородной системы, минимизируя время ожидания запуска следующего этапа. Очевидно, что такая идеология связана с избыточностью однородной матрицы, т.к. минимально необходимое количество процессорных элементов равно максимальному количеству параллельных этапов в графе вычислительного процесса. В однородной системе, однако, количество процессорных элементов должно быть больше или равно общему количеству вершин графа.

Отметим, что существенной для обеспечения эффективности является проблема разделения задачи на однородные этапы, которая ложится на плечи программистов.

В связи с тем, что разные задачи обладают разными графами вычислительного процесса, реально задействованная структура процессорных элементов и связей между ними существенно меняется для разных задач - в связи с этим такие системы получили название матричных систем с перестраиваемой структурой. Функционирование такой системы проиллюстрировано на рис 12.1

Однородная матричная система с перестраиваемой структурой

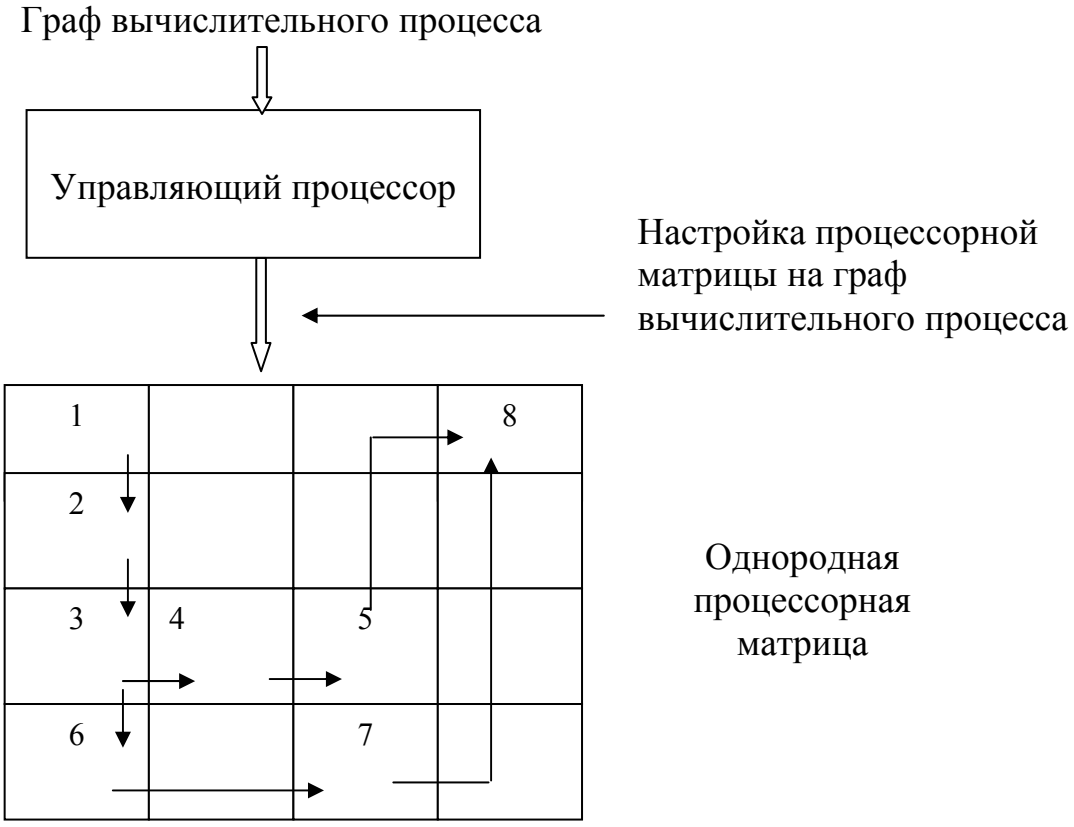


Рис 12.1

Идея однородных систем была предложена в начале 60-х годов Э.В. Евреиновым и Ю.Г. Косыревым в виде следующих трех принципов построения систем:

1) параллельность подзадач в алгоритмах (гипотеза параллельности - для сложной задачи можно предложить эффективный параллельный алгоритм решения);
2) переменность логической структуры;

3) конструктивная однородность элементов и связей между ними. Наиболее сложной задачей реализации однородных систем считается задача программирования связей.

12.2 Матрицы волнового фронта данных - параллелизм команд

Этот подход к организации матричных систем основан на принципе управления по потоку данных. В отличие от машин потока данных, где арбитражная сеть выбирает готовые к выполнению командные ячейки и отправляет их на выполнение в процессор, в матричном процессоре волнового фронта каждый элемент матрицы представляет собой самостоятельное АЛУ с назначенной командной ячейкой. Передавая результаты выполнения команд (поток данных), процессоры активируют друг друга, создавая динамическую по времени активную процессорную структуру - рис 12.2.

Матричная система с волновым фронтом данных

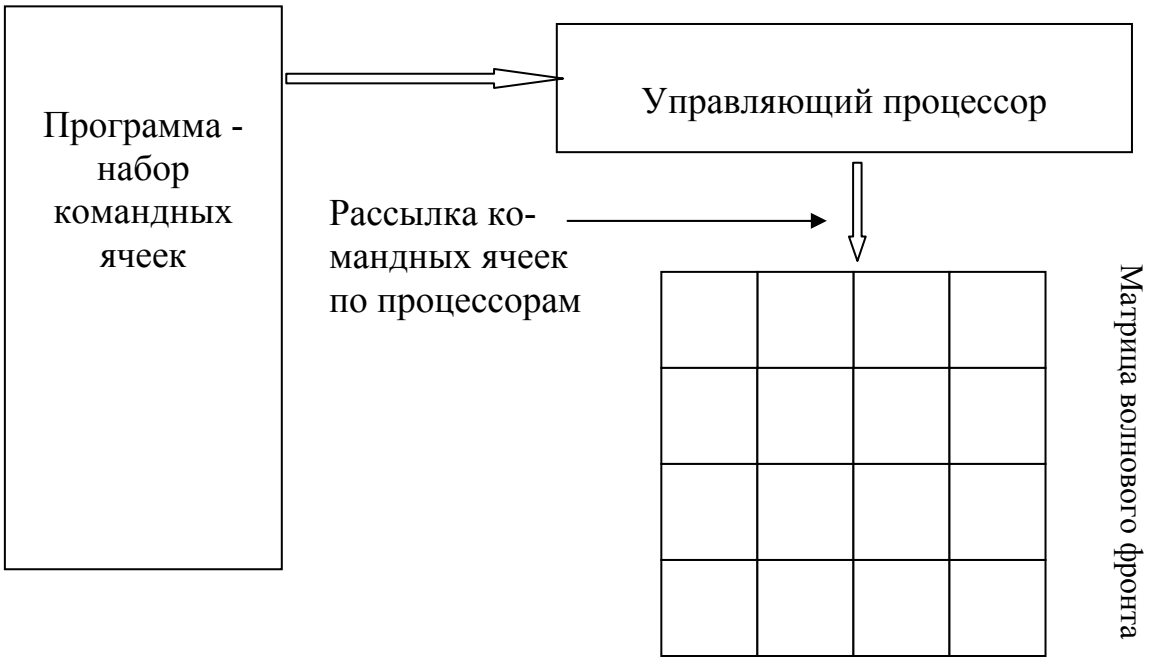


Рис 12.2

Однако реализация матрицы волнового фронта требует достаточно сложного алгоритма управления, т.к. управляющий процессор должен отслеживать динамическое состояние матрицы и при условии, что количество АЛУ в матрице меньше количества командных ячеек в программе, рассылать новые командные ячейки взамен уже выполненных. Еще одной проблемной задачей такой архитектуры является динамическое изменение связей АЛУ.

12.3 Классические матричные системы - параллелизм объектов

Классические матричные системы реализую принцип - «одиночный поток команд - множественный поток данных». Процессорная матрица состоит из множества процессорных элементов (ПЭ) и одного устройства управления (УУ). УУ одновременно передает всем ПЭ одну и ту же команду, поэтому на всех ПЭ одновременно выполняется одна и та же операция, но с разными данными. Для передачи данных между ПЭ используется синхронная сеть связи. Схема классической матричной системы приведена на рис 12.3. Такая архитектура ориентирована, прежде всего, на задачи обработки матриц и обработки изображений.

Схема классического матричного процессора

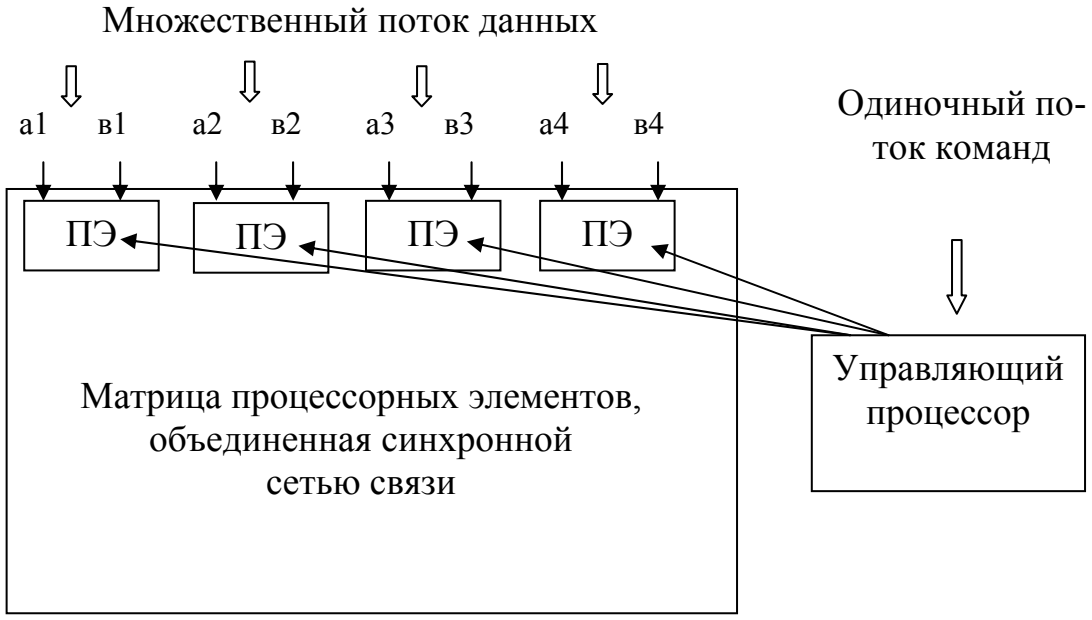


Рис 12.3

Одной из первых реализаций такой матричной архитектуры была машина ILLIAC - IV, разработанная во второй половине 60-х годов в Иллинойском университете и изготовленная фирмой «Барроуз». Другой пример - разрядно модульный матричный процессор - MPP (Massively Parallel Processor), разработанный фирмой Goodyear по заказу NASA для обработки изображений, передаваемых из космоса.

Л И Т Е Р А Т У Р А

1. Амамия М., Танака Ю. Архитектура ЭВМ и искусственный интеллект: Пер. с японск. - М.: Мир, 1993. - 400 с., ил.
2. Валиев К.А. и др. Развитие элементной базы высокопроизводительных ЭВМ // Информационные технологии и вычислительные системы № 1. - 1996.
3. Ларионов А.М. и др. Вычислительные комплексы, системы и сети / А.М. Ларионов, С.А. Майоров, Г.И. Новиков: Учебник для вузов. Л.: Энергоатомиздат. Ленингр. отд-ние, 1987. 288 с., ил.
4. Принципы работы системы IBM/370 : Пер с англ. под ред. Л.Д. Райкова. - М.: Мир, 1978. - 576 с.
5. Пятибратов А.П. и др. Вычислительные системы, сети и телекоммуникации: Учебник. - 2-е изд., перераб. и доп. / А.П. Пятибратов, Л.П. Гудыко, А.А. Кириченко: Под ред. А.П. Пятибратова. - М.: Финансы и статистика, 2001. - 512 с., ил.
6. Смирнов А.Д. Архитектура вычислительных систем. - М.: Наука, 1990.
7. Столлингс В. Структурная организация и архитектура компьютерных систем, 5-ое изд.: Пер. с англ. – М.: Издательский дом "Вильямс", 2002.– 896 с.: ил.

Учебное издание

Ульянов Михаил Васильевич

Архитектуры процессоров
Учебное пособие

Подписано в печать 02.10.2002
Формат 60 x 80 1/16
Объем 4,25 п.л. Тираж 300 экз. Заказ № 128

Отпечатано в типографии Московской государственной академии
приборостроения и информатики
107846, Москва, ул. Стромынка, 20