Организация ЭВМ и систем.

Конспект лекций. Рабочая версия.

Кустарев П.В., Довгий П.С., Скорубский В.И.
Кафедра Вычислительной Техники
СПбГУ ИТМО
Санкт-Петербург 2010-2011г.

ОГЛАВЛЕНИЕ.

ОСНОВНЫЕ ПОНЯТИЯ.

1.1 Понятие ЭВМ

Вычислительная машина (ВМ, компьютер) – это искусственная техническая система, предназначенная для обработки данных под управлением наборов команд.

Непосредственно с понятием ЭВМ связаны, а иногда заменяют его, другие понятия. Наиболее часто употребляемые из них:

Процессор – центральный аппаратный блок ЭВМ, непосредственно осуществляющий обработку данных. В настоящее время основным типом процессоров являются цифровые программно управляемые процессоры. Такие процессоры построены на базе цифровых электронных схем и используют для обработки данных методы булевой алгебры и двоичной арифметики.

Вычислительный комплекс (ВК) – несколько ЭВМ, имеющих общие (разделяемые структурные блоки, например, общую память.

Вычислительная система (ВС) – аппаратно-программная система из нескольких ЭВМ, жестко взаимосвязанных аппаратно и логически, и, обычно, пространственно локализованных. ЭВМ в составе ВС являются неотъемлемыми частями для решения единой задачи.

Вычислительная сеть (ВСТ) – система из нескольких ЭВМ, функционирующих автономно, но связанных между собой каналами передачи данных, которые могут использоваться для обмена данными, для синхронизации работы ЭВМ и для организации совместного решения единой задачи (подобно ВС). ВСТ могут быть пространственно децентрализованными, но также могут быть локализованными, например, в рамках одной электронной платы (Network on a Board) или даже в одной микросхеме (Network on a Chip).

1.2 Типы ВМ.

1) ВМ может иметь различную физическую природу:

– *Электронные ВМ (ЭВМ)* – ВМ, имеющая в физической основе электронную схему – это на сегодня наиболее распространенные типы ВМ. Они в свою очередь могут делиться на *цифровые ЭВМ* и *аналоговые ЭВМ*.

– *Механические, оптические, пневматические, биологические и другие*, в том числе комбинированные типы ВМ – данные типы ВМ в *настоящее время* менее распространены, чем ЭВМ в силу неразвитости или неэффективности технологии их производства на современном этапе. Однако могут рассматриваться как варианты перспективных типов ВМ (например, ведутся активные исследования в области опто-электронных ВМ).

2) ВМ «настраиваются» на исполнение требуемого алгоритма путём исполнения поданных им команд. Множество команд, которое ВМ может понять и выполнить называют *набором команд ВМ (instruction set).*
Существуют различные типы ВМ, для которых понятие «команда ВМ» интерпретируется по разному:

a) Команды представлены в виде некоторых кодов, которые последовательно исполняются ВМ. В процессе исполнения команд внутренние структура и правила функционирования ВМ не меняются – можно говорить, что команды для неё – это нечто «внешнее». ВМ с таким принципом работы называют *«программно-управляемой ВМ»*. На сегодняшний момент - это наиболее распространенный тип ВМ. Важно, что программа и алгоритм это не одно и то же: один и тот же алгоритм может быть реализован в виде нескольких различных программ.

b) У ЭВМ, работающих по принципу реконфигурирования, команды – это конфигурационные параметры, которые перенастраивают структуру и режимы функционирования физических средств ВМ на выполнение той или иной функции или алгоритма по обработке данных. Например, можно изменять соединения между блоками ВМ, можно настраивать блоки на реализацию различных функций, можно менять коэффициенты передачи (усиления) электронных каскадов. Данная интерпретация понятия «команда» менее распространено, характерно в основном для специализированных ВМ - для ВМ встраиваемых в приборы, в бортовую аппаратуру и т.п. Однако, в связи с совершенствованием и ростом популярности цифровой аппаратуры с перепрограммируемой структурой (например, программируемых логических интегральных схем - ПЛИС), данный тип ВМ также набирает популярность.

c) вариант комбинации двух первых вариантов.

В дальнейшем, если не будет особых пометок, речь пойдет об цифровых ЭВМ с последовательной или комбинированной (с приоритетом последовательной) интерпретацией команд.

Примечание:

В литературе предлагается множество различных формулировок понятия ЭВМ (электронная вычислительная машина)1, компьютер и связанных с ними, от достаточно простых и понятных до вычурных, многие из которых, однако, схожи по своей сути. Например:

Компьютер – это прибор, способный производить вычисления и принимать решения в миллионы или даже в миллиарды раз быстрее человека. Компьютеры обрабатывают данные под управлением наборов команд, называемых компьютерными программами [13].

Компьютер – это машина, которая может решать задачи, выполняя данные ей команды. Последовательность команд, описывающих решение определенной задачи, называется программой [16].

ЭВМ – комплекс электронного оборудования, выполняющий интерпретацию программ в виде физических процессов, назначением которых является реализация математических операций над информацией, представляемой в цифровой форме [2].

ЭВМ – искусственная (инженерная) система, предназначенная для выполнения вычислений на основе алгоритмов. Принципы построения ЭВМ определяются с одной стороны назначением ЭВМ и с другой – элементной базой (набором элементов, которые используются для создания ЭВМ). Основным назначением ЭВМ является выполнение вычислений на основе алгоритмов, и поэтому свойства алгоритмов предопределяют принципы построения ЭВМ или, точнее, ее архитектуру (организацию) [4].

[1] В дальнейшем изложении термины «ЭВМ» и «компьютер» используются как синонимы.

1.3 Понятие архитектуры и организации ЭВМ

В компьютерной литературе связанно (иногда как синонимы, иногда как взаимодополняющие термины) применяются понятия *архитектура ЭВМ (computer architecture), организация ЭВМ (computer organization). В ассоциации с этими* понятиями используются также термины *дизайн (проект) ЭВМ (computer design), аппаратное обеспечение (hardware), программное обеспечение различных уровней (software, firmware, middleware) и другие.*

Единого признанного определения для понятий *архитектура и организация ЭВМ* не существует. Многие специалисты используют их как синонимы. Например, одним из сторонников такого подхода является Э. Таненбаум. В других источниках эти термины имеют различающуюся трактовку, что позволяет более детально определять аспекты, проблемы и задачи, решаемые в рамках создания ЭВМ. Сторонниками такого подхода являются Д. Паттерсон и Д.Хеннеси, В. Столлингс и многие другие. Кроме того, трактовка одного и того же понятия может различаться в зависимости от того, на каком этапе проектирования ЭВМ используется (разработка архитектуры, моделирование, реализация аппаратуры и программного обеспечения).

В данном курсе эти понятия рассматриваются как различные.

1.3.1 Архитектура и организация ЭВМ в «широком» и «узком» смыслах.

Можно рассматривать две трактовки понятий архитектура и организация ЭВМ (а также связанных с ними терминов): широкая и узкая.

Понятия *в широком смысле* используются, чтобы описать и регламентировать процесс проектирования ЭВМ, указывают те обязательные аспекты, этапы и задачи, которые должны быть выделены и проработаны при создании ЭВМ/ВС. Это очень важно, чтобы в процессе создания сложных вычислительных систем были выполнены все фазы, в правильном порядке и с правильными приоритетами и ничего не было упущено. В конечном счёте это определяет качество проектирования.

С другой стороны каждое из понятий в широком смысле затрагивает ВСЕ подсистемы и блоки ЭВМ (хотя и на различном уровне детализации), но не фиксируют перечень элементов ЭВМ/ВС и состав проектных документов, которые относятся к спецификациям архитектуры, организации ЭВМ соответственно. То есть они не позволяют сказать: «Описание архитектуры (организации) ЭВМ должно включать то-то, то-то и то-то». Соответственно их сложно применять в качестве обобщающего регламента для формирования описания ЭВМ. Поэтому для «практических нужд» предложены определения в «узком» смысле.

Понятия *в узком смысле* нацелены на описание устройства ЭВМ и ее свойств.

1.3.2 Понятие архитектуры и организации ЭВМ в «широком» смысле.

ЭВМ может быть описана и проектироваться на различных уровнях:

- На уровне системы команд – это так называемый уровень *Архитектуры системы команд (Instruction Set Architecture, ISA);*
- На уровне (крупных) функциональных блоков и подсистем ЭВМ – уровень *организации ЭВМ*;
- На уровне реализации функциональных блоков в виде программы или аппаратуры – уровень программной, аппаратной или комбинированной (программно-аппаратной) *реализации ЭВМ (implementation).*

Архитектура системы команд (ISA) – совокупность аспектов ЭВМ, видимых программисту. К ней относится организация адресного пространства памяти и

периферийных устройств ЭВМ, режимы адресации, типы и форматы операндов, типы и форматы команд, режимы исключений (прерываний), специальные режимы (защищённый, прямого доступа к памяти и т.п.) и другие аспекты. Архитектура системы команд – «внешний вид» ЭВМ, не показывает внутреннего устройства.

Организация ЭВМ рассматривает устройство и проектирование ЭВМ на уровне крупных функциональных боков и подсистем ЭВМ. Например, рассматриваются функции и принципы работы и взаимодействия процессора, блоков памяти, периферийных устройств, шин и т.п. Причём эти принципы (принципы организации ЭВМ) будут похожими как для простейших ЭВМ, так и для высокопроизводительных компьютеров – разница в количественных параметрах. Например, модули памяти ЭВМ, встраиваемой в телефон, и высокопроизводительного сервера различаются количеством запоминающих ячеек, шириной, скоростью передачи данных для шины данных и некоторыми иными параметрами, определяющими ее быстродействие, но внутренняя структура и основные принципы функционирования будут очень похожими.

Кроме аппаратных блоков на уровне организации рассматривают и программные функциональные блоки ЭВМ: супервизоры, операционные системы, драйверы и т.п., но также на высоком уровне, не опускаясь до деталей внутренней организации программного обеспечения.

На уровне организации ЭВМ *не рассматривают* цифровые электрические схемы (аппаратуру) и программный код (программное обеспечение).

Реализация ЭВМ - воплощение функциональных блоков ЭВМ и связей между ними в виде аппаратуры и/или программного обеспечения.

Архитектура ЭВМ в широком смысле рассматривает общие аспекты, способы объединения процессов и результатов всех уровней проектирования ЭВМ: архитектуры системы команд, организации и реализации.

Конечная цель архитектурного проектирования в широком смысле - объединение компонент и подсистем ЭВМ таким образом, чтобы достигнуть в целом для ЭВМ требуемой функциональности, производительности, энергопотребления, других функциональных, технических, конструктивных и коммерческих параметров, вплоть до маркетинговой привлекательности.

То есть *архитектура в широком смысле* – это совокупность согласованных проектных решений на уровнях системы команд, организации, программно-аппаратной реализации.

Из приведенных выше определений следует понятие *Технического проекта (дизайна)* ЭВМ в широком смысле – это совокупность спецификаций всех уровней (ISA, организации, реализации), необходимых и достаточных для понимания устройства и функционирования ЭВМ, для ее использования и производства.

1.3.3 Понятие архитектуры и организации ЭВМ в «узком» смысле.

Организация ЭВМ в «узком» смысле – все аспекты внутреннего устройства ЭВМ (как аппаратуры, так и программных компонент), которые «не видимы» для пользователя ЭВМ (в большинстве случаев – для программиста, но также и для конечного пользователя) и не оказывают прямого влияния на возможности и правила использования (программирования) ЭВМ, если сравнивать их с альтернативными вариантами организации ЭВМ. Например, использование x86-совместимых процессоров одного класса от фирмы Intel или от фирмы AMD не влияет сколько-нибудь ощутимо на правила создания программного обеспечения и на характеристики его функционирования.

Архитектура ЭВМ в «узком» смысле определяет представление и описание возможностей ЭВМ с точки зрения пользователя, например, для программиста разрабатывающего программу на машинно-ориентированном языке, но также, возможно, и для прикладного программиста, и для конечного пользователя, выбирающего настройки программного обеспечения. В этом смысле архитектура в «узком» смысле близка к понятию архитектуры системы команд (ISA).

Архитектура отображает те аспекты структуры и принципы функционирования ЭВМ, которые являются видимыми и/или ощутимыми для пользователя. Например, видимой для программиста будет регистровая модель, ощутимой но не видимой для программиста может быть организация конвейера обработки команд, который будет более эффективен, например, в предсказании переходов, если соблюдать определенные правила построения последовательности машинных команд (на практике эти правила заложены в компилятор программ). Также к архитектуре могут быть отнесены принципы организации вычислений (вычислительного процесса), которые будут более эффективны на ЭВМ определенного типа, с определенной организацией.

Технический проект (*дизайн) ЭВМ* в узком смысле – техническая спецификация конкретных электронных схем и компонентов программного обеспечения различного уровня (встроенного (firmware), системного и прикладного (software)), из которых построена конкретная ЭВМ или ВС, набор правил проведения инженерной разработки, которые в комплексе позволяют получить изделие (ЭВМ, ВС) определенного качества, за определенные сроки и деньги. Технический проект в узком смысле не должен в деталях определять высокоуровневые принципы устройства ЭВМ, но должен быть достаточным для ее производства и использования.

1.4 Виды организации и архитектуры ЭВМ.

1.4.1 Виды организации ЭВМ.

Любую техническую систему, в том числе ЭВМ, можно охарактеризовать ее структурой и функцией. В связи с этим организацию систем, в частности организацию ЭВМ, принято рассматривать в двух аспектах:

1) Структурная организация
2) Функциональная организация.

Структурная и функциональная организация ЭВМ – это различный взгляд на одно и то же: на блоки/компоненты в составе ЭВМ и связи между ними.

Структурная организация ЭВМ определяет состав ЭВМ на уровне блоков (устройств, компонентов), из которых она состоит, и организацию связей между ними, на уровне интерфейсов. Не так давно в этой связи рассматривались только аппаратные блоки, но на современном этапе сюда относят и программные компоненты, как минимум компоненты системного программного обеспечения.

Функциональная организация ЭВМ – рассматривает ЭВМ с точки зрения функционирования блоков/компонент в составе ЭВМ, то есть определяет какую функцию выполняет каждый блок/компонент, как и в каком порядке он взаимодействует с другими блоками/компонентами в рамках выполнения общей задачи на ЭВМ.

1.4.2 Виды архитектуры ЭВМ.

1.4.2.1 Уровни архитектурных описаний.

Как говорилось выше, понятие архитектуры может интерпретироваться по разному, в зависимости от решаемых разработчиком ЭВМ или пользователем ЭВМ задач. В первую очередь будет различаться уровень архитектурного представления ЭВМ.

Высоким уровнем представления (описания) ЭВМ будет считаться описание структуры и функций без деталей устройства аппаратуры и организации программного обеспечения.

По мере снижения уровня описания будут исследоваться детали внутреннего устройства ЭВМ вплоть до составления технической и технологической документации для ее производства.

Самым высоким уровнем описания архитектуры ЭВМ считают описание *организации вычислительных процессов*, реализующих заданные/выбранные алгоритмы.

На более низком уровне рассматривают архитектуру на уровнях *прикладной и системной «программных моделей ЭВМ»*, «видимых» прикладному и системному программисту.

На еще более низких уровнях рассматривается на уровнях встроенного конфигурационного кода или микропрограммного обеспечения (firmware) и на уровне схемотехники электронных схем и физической конструкции ЭВМ.

Наконец на самом низком уровне рассматривают технологию производства электронных и конструктивных компонент.

1.4.2.2 Программная и аппаратная архитектуры.

Существует достаточно много классификаций архитектурных представлений.

Естественным, часто применяемым на практике подходом к классификации архитектур ЭВМ является выделение двух видов архитектур ЭВМ (одним из сторонников подобного подхода к классификации архитектуры является В. Л. Григорьев.):

1) *программная архитектура*, которая включает в себя аспекты, видимые и важные для программиста.
2) *аппаратная архитектура*, которая включает в себя аспекты, связанные с аппаратными средствами, **не**видимые программистом (прозрачные как для программиста, так и для программ), но оказывающие влияние на способ и эффективность использования ЭВМ, прежде всего на организацию программного обеспечения (см. п.1.3.3).

Программную архитектуру также можно разделить на два уровня: *прикладную* и *системную*, что связано со сложившимся различием проблем и приоритетов в работе системных и прикладных программистов.

В соответствии с рассмотренным выше принципом разделения понятий *архитектура ЭВМ* и *организация ЭВМ*, а также с разделением архитектуры ЭВМ на программную и аппаратную, можно сопоставить понятия *аппаратная архитектура ЭВМ* и *структурная организация ЭВМ*.

1.4.2.3 Instruction Set Architecture, ISA – архитектура системы команд.

Архитектура системы команд (Instruction Set Architecture, ISA) относится к классу программных архитектур. Она представляет собой описание средств ЭВМ (команд. регистров, памяти и других), которые доступны программисту на уровне машинных команд (самый низкий уровень программирования). ISA является спецификацией аппаратуры для программиста (можно сказать. что ISA задает границу между аппаратурой и программными средствами ЭВМ) и соответственно ISA – обязательная часть описания любого процессора, определяющая, как его использовать. Более подробно ISA будет рассмотрена в последующих разделах.

1.4.2.4 Системная архитектура.

В настоящее время в силу возросшей сложности устройства ЭВМ/ВС аспекты программной и аппаратной архитектур тесно переплелись между собой и с трудом могут быть отделены друг от друга. Это касается прежде всего специализированных систем. Например:

- в системах обработки аудио- и видеоинформации функции, описываемые на уровне прикладных программ, могут иметь полностью аппаратную реализацию и при разработке программ программист должен учитывать особенности доступа к данным аппаратным функциональным блокам и доступность этих блоков в каждый момент времени.
- в быстродействующем телекоммуникационном оборудовании в зависимости от необходимоти поддержки того или иного сетевого протокола могут генерироваться/подключаться соответствующие специализированные аппаратные процессоры. Если какого либо протокола не требуется, то соответствующие процессоры «стираются» из аппаратуры, а на их месте создаются процессоры другого типа. Такое стало возможным только в последнее время после появления мощной динамически реконфигурируемой аппаратуры – ПЛИС.

Таким образом, выделяют не только аппаратную и программную архитектуры, но и *системную архитектуру*, которая располагается над аппаратной и программной, описывает, из каких функциональных блоков состоит ЭВМ, как они объединены, но не указывает способ их реализации – аппаратный или программный.

1.4.3 Обобщенное представление архитектуры и организации ЭВМ и их аспектов.

В соответствии классификацией, приведенной в предыдущих разделах, можно предложить структуру аспектов организации и архитектуры ЭВМ, изображенную на рисунке (см. Рисунок 1).

К основным элементам (аспектам) *прикладной программной архитектуры* (прежде всего архитектуры уровня системы команд - ISA), как правило, относят:

- типы, форматы и способы представления данных, аппаратно поддерживаемые в ЭВМ;
- регистровая структура процессора;
- адресная структура основной памяти и принципы размещения информации в ней, принципы формирования физического адреса;
- режимы адресации;
- структуры и форматы машинных команд;
- система команд.

Все аспекты прикладной архитектуры входят и в системную архитектуру.

К дополнительным аспектам *системной программной архитектуры*, как правило, относятся:

- организация прерываний;
- организация ввода/вывода;
- организация виртуальной памяти (сегментная и страничная), принципы преобразования логического (виртуального) адреса в физический;
- организация защиты памяти;
- организация многозадачного (многопрограммного) режима работы ЭВМ, организация переключения задач (программ);
- поддержка механизмов отладки программ на аппаратном уровне;
- поддержка механизмов проверки (тестирования) отдельных блоков процессора на аппаратном уровне.

К основным аспектам *аппаратной архитектуры*, как правило, относятся:

- структурная организация ЭВМ, включающая в себя номенклатуру устройств, входящих в состав ЭВМ, и организацию связей между устройствами на уровне аппаратных интерфейсов;

- структурная организация процессора, включающая в себя реализацию конвейера команд и арифметико-логического устройства и принципы построения блока микропрограммного управления;
- организация кэш−памяти;
- организация основной памяти на физическом уровне и, в частности, принципы построения многомодульной памяти с расслоением обращений (чередованием адресов);

представление аппаратного интерфейса на физическом уровне.

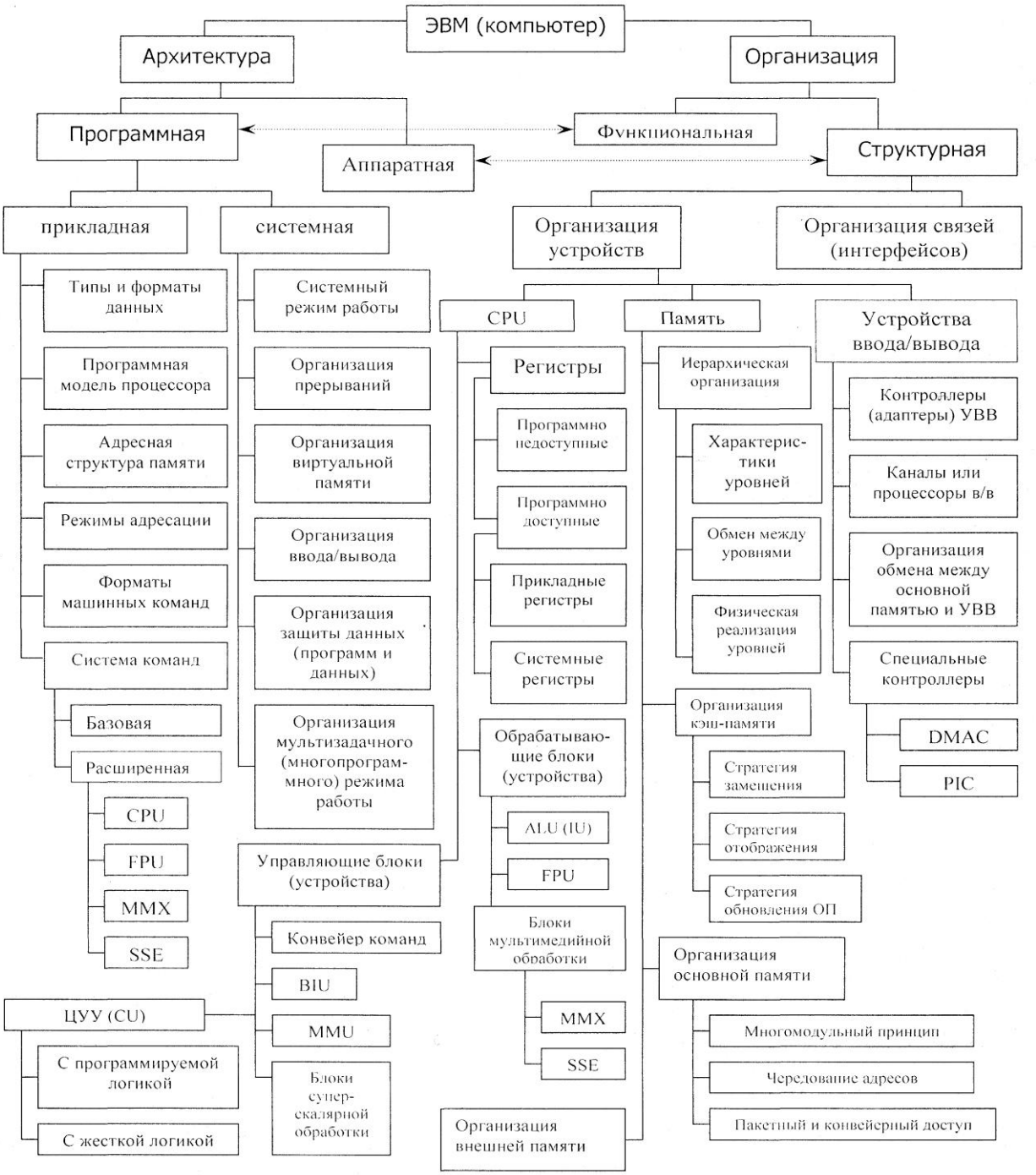

Рисунок 1 Обобщенное представление архитектуры и организации ЭВМ.

CPU - *Central Processing Unit* - Центральное устройство обработки, Центральный процессор
FPU - *Floating Point Unit* - Устройство обработки с плавающей точкой
MMX - *Multi Media Extension* - Мультимедийное расширение
SSE - *Streaming SIMD Extension* - Потоковое SIMD расширение
SIMD - *Single Instruction Multiple Data* - Одиночный поток команд и многопотоковых данных

ALU - *Arifmetic and logic Unit* - Арифметическое и логическое устройство
IU - *Integer Unit* - Целочисленное устройство
MMU - *Memory Management Unit* - Устройство (блок) управления памятью
BIU - *Bus Interface Unit* - Интерфейсный блок (устройство)
DM AC - *Direct Memory Access Controller* - Контроллер прямого доступа памяти
PIC - *Programmable Interrupt Controller* - Программируемый контроллер прерывания
CU - *Central Unit* - Устройство (блок) управления

2 Организация ЭВМ

2.1 Логическая организация ЭВМ

Логическая организация ЭВМ (т.е. представление связанных структурно-функциональных компонент) представлена на рисунке (см. Рисунок 2).

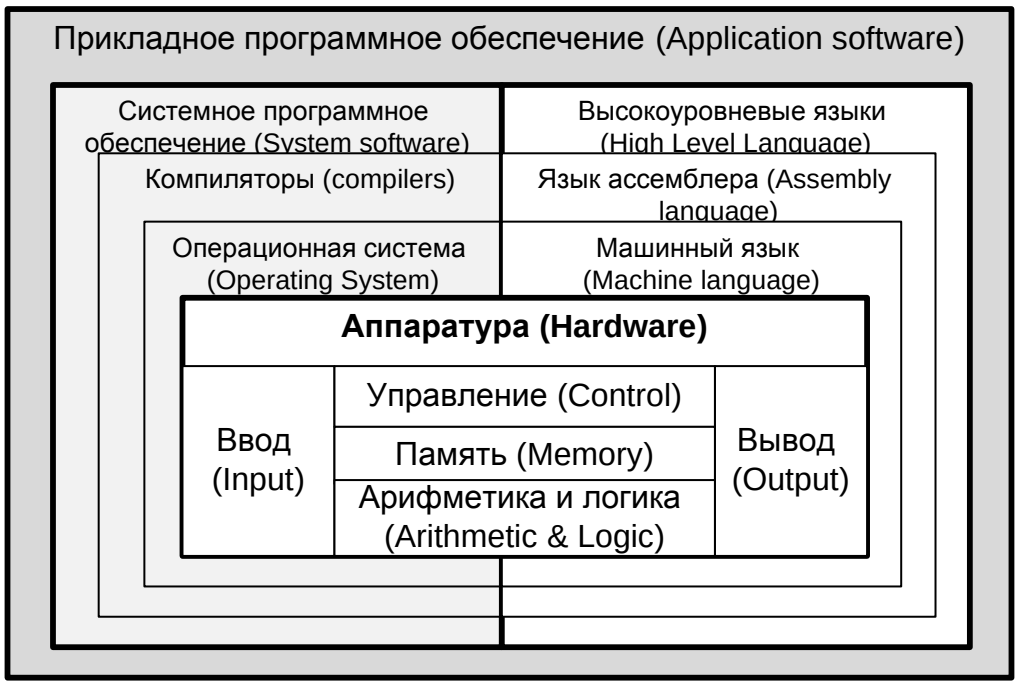

Рисунок 2 Логическая организация ЭВМ

Можно рассматривать два варианта логического представления ЭВМ: как иерархию программных и аппаратных подсистем (см. левую часть рисунка) и как иерархию виртуальных машин, взаимнопреобразующихся друг в друга (см. правую часть рисунка). Так как эти представления тесно связаны и зависят друг от друга, то они изображены на одном рисунке.

2.1.1 ЭВМ как иерархия программных и аппаратных компонент.

Аппаратура (аппаратные средства, аппаратное обеспечение, hardware[2]) является физической основой ЭВМ. Это цифровая электрическая схема, состоящая из нескольких подсистем. Говорят, что аппаратные компоненты находятся *на нижнем уровне* организации ЭВМ.

Типовыми аппаратными подсистемами ЭВМ являются:

- процессор, включающий устройство выполнения арифметических и логических операций над данными и устройство управления, осуществляющее декодирование машинных команд и управление работой арифметическо-логического устройства;
- память, в которой хранятся программы и данные;

[2] *Строго говоря, термин HARDWARE обозначает не только аппаратуру, а все неизменяемые пользователем части ЭВМ, например, аппаратные средства с «зашитой» микропрограммой.*

- подсистемы ввода и вывода, отвечающие за взаимодействие процессора с «внешним» миром: с человеком (посредством монитора, звуковой системы, клавиатуры и других устройств человеко-машинного интерфейса (Human-Machine Interface, HMI)), с окружающей физической средой (посредством датчиков и исполнительных устройств – двигателей, ламп, реле и т.п.)

Аппаратные блоки ЭВМ (процессор, сопроцессоры) могут исполнять только машинные команды. Кроме того, их устройство и управление ими очень сложно. Поэтому реализовывать прикладные алгоритмы (например, графический редактор на ПК) в виде программ из машинных команд фактически невозможно для человека. Поэтому между прикладными программами и аппаратурой ЭВМ имеется специальная «прослойка» - *системное программное обеспечение*.

В различных ЭВМ используются много разных типов системного программного обеспечения, но два типа являются центральными:

1) операционная систем (operating system);
2) компилятор (compiler).

Операционная система (ОС) – это системное программное обеспечение, которое обеспечивает интерфейс между пользовательскими программами и аппаратурой, а также отвечает за управление работой аппаратуры и распределение ее ресурсов между программами. Типовыми функциями ОС являются:

- поддержка операций ввода и вывода;
- распределение системной и внешней памяти между программами;
- управление одновременным исполнением на ЭВМ нескольких программ (задач);
- организация взаимодействия нескольких ЭВМ по сети.

Компилятор – отвечает за преобразование (трансляцию) программ на языках высокого уровня (пригодных для написания программ человеком) в программы из машинных команд (в машинные коды). Кроме того, компиляторы преобразуют программные операции в команды операционных систем: поэтому на рисунке они изображены «над» операционной системой. Трансляция обычно выполняется в несколько этапов: программы на высокоуровневом языке (например, на C++) преобразуются в программы на языке ассемблера, а они в свою очередь конвертируются в машинные коды. Или в три этапа: Java – Java Byte Code – Ассемблер – машинные коды.

Компиляторы могут выполнять преобразование предварительно – перед запуском программ на исполнение. Такой режим называют *offline-компиляция* или просто компиляция. Другой вариант компиляции – преобразование «на лету», в процессе исполнения высокоуровневой программы. Такой способ называется *online-компиляцией или трансляцией.*

В любом случае компилятор является обязательным (явным - online или неявным - offline) компонентом системного программного обеспечения, от которого зависит каким образом прикладной алгоритм (в виде программы) будет понят ЭВМ.

Прикладное программное обеспечение (application software) – это программа на языке высокого уровня, реализующая прикладной алгоритм (задачу). Для прикладной программы скрыты особенности внутренней организации ЭВМ (аппаратуры, системного ПО); она использует операции и форматы данных, определенные спецификацией высокоуровневого языка.

2.1.2 ЭВМ как иерархия виртуальных машин.

Другой взгляд на ЭВМ – как на иерархию *программируемых ВМ* с различной системой команд (языками) и организацией вычислительных средств (памяти, регистров и т.п.) В

процессе поэтапной компиляции программ осуществляется переход от более высокоуровневой ВМ к низкоуровневой и так до аппаратуры - до физической ВМ.

На верхнем уровне языком ВМ является высокоуровневый язык программирования (C, C++, Java и т.п.) плюс набор команд (вызовов) операционной системы. Форматы данных и наборы операций также определяются спецификацией высокоуровневого языка. Программы пишет человек в соответствии с реализуемым им прикладным алгоритмом.

Высокоуровневых языковых ВМ может использоваться одна или несколько.

Ниже находится ВМ с языком ассемблера. Организация этой ВМ максимально близка к организации физической ВМ: такой же набор регистров, памяти. Основные отличия: использование символьных идентификаторов для команд и данных, поддерживается модульная организация программ и т.п.

Под ВМ в иерархии ассемблера располагается ВМ с системой машинных команд. Это физическая ВМ, как она доступна для программирования.

Ниже располагаются аппаратные средства ЭВМ. Однако (хоть это и «не видно» программисту), они «внутри» так же организованы как иерархия ВМ, имеющая несколько НЕОБЯЗАТЕЛЬНЫХ (кроме нижнего) уровней:

- уровень настроек для аппаратных функциональных блоков (таймеров, контроллеров и т.п.);
- уровень микропрограмм;
- уровень конфигураций («прошивок») для аппаратной платформы (например, для ПЛИС);
- **уровень физической цифровой схемы - НЕИЗМЕНЯЕМЫЙ**.

Так как все уровни выше уровня машинных команд не имеют в основе реальных аппаратных средств, воспринимающих соответствующий язык команд, то их называют *виртуальными ВМ (ВВМ)*.

Уровни включая и ниже уровня машинных команд относятся к классу *реальных ВМ (РВМ)*. Для большинства пользователей – программистов единственной реальной ВМ является ВМ уровня машинных команд[3].

2.1.3 Связь вариантов представления ЭВМ.

Рассмотренные выше варианты представления ЭВМ имеют жесткую взаимозависимость:

1) Взаимное преобразование виртуальных ВМ выполняется при помощи системного программного обеспечения: компиляторов (в процессе трансляции или компиляции) и операционных систем (в процессе обработки команды для ОС и управления ресурсами ЭВМ для прикладных программ).
2) Уровень РВМ обычно ассоциируется с уровнем аппаратуры ЭВМ, как физической (реальной) основы ЭВМ.

Исходя из этого при разработке ЭВМ эти два представления организации ЭВМ следует рассматривать взаимосвязано.

2.2 Структурно-функциональный базис ЭВМ

Основой большинства современных ЭВМ является цифровая электронная аппаратура – цифровые схемы. Данные схемы строятся на *элементной базе* - элементарных аппаратных

[3] Если оценивать понятия реальной и виртуальной ВМ в более широком смысле, то мы можем установить, что нет четко определенных уровней ВВМ и РВМ – все должно рассматриваться относительно доступных возможностей программирования ВМ.

Если при разработке программного обеспечения на данном уровне программный код ВМ изменяется, то это уровень ВВМ. Если не изменяется – РВМ. Например: Если мы берем ПЛИС, записываем в нее прошивку «микропроцессор» (т.е. она работает как микропроцессор) и отдаем программисту для разработки программ, то для него уровнем РВМ будет уровень машинных операций микропроцессора. Если же пытаться делать обработку прикладных данных аппаратно путем разработки специальных аппаратных блоков на ПЛИС, то уровнем РВМ станет уровень конфигурирования ПЛИС.

функциональных элементах, выполненные в современной интегральной технологии (в виде интегральных микросхем, ИМС).

В соответствии с иерархическим подходом, принятым при проектировании сложных систем, аппаратная элементная база также может быть разделена на уровни:

1) Нижний уровень – *вентильный*: логические элементы (вентили) И, ИЛИ, НЕ и т.п., и запоминающие элементы – триггеры.
2) Средний уровень – *базовых операционных элементов*: элементы выполнения арифметических операций – сумматоры, АЛУ; многоразрядные запоминающие элементы - регистры, адресуемая регистровая память, счетчики; элементы коммутации и управления коммутацией – мультиплексоры, шины, шифраторы, дешифраторы.
3) Высокий уровень – *функциональных блоков*: РАЛУ, умножители, массивы памяти, порты ввода-вывода.
4) Сверхвысокий уровень – *подсистем*: процессор, блоки и устройства памяти, контроллеры интерфейсов и т.п.

В реальном проектировании нет четких границ между уровнями, на любом этапе могут возникнуть задачи, которые требуют более детальной проработки на более низких уровнях, или доступны уже существующие высокоуровневые решения.

2.2.1 Вентильный уровень.

…

2.2.2 Уровень базовых операционных элементов.

Базовые операционные элементы (БОЭ) – функционально завершенные цифровые схемы, которые выполняют обработку многоразрядных цифровых данных (слов). БОЭ в составе структурно-функциональных и электрических принципиальных схем ЭВМ выступают как законченные элементы, т.е. не представляется их внутреннее устройство.

БОЭ бывают *комбинационными*: в каждый момент времени значения на цифровых выходах БОЭ зависят только от значений на их входах в это же время, и *последовательностными:* значения на цифровых выходах БОЭ зависят от значений на их входах и от состояния схемы в предыдущие моменты времени. Для запоминания предыдущего состояния в составе последовательностных БОЭ имеются запоминающие элементы – триггеры.

Основными типами комбинационных БОЭ, применяющимися в составе процессоров и ЭВМ являются:

2.2.2.1 Сумматор (Adder)

Сумматор выполняет арифметическое сложение и вычитание (в дополнительном коде) чисел, представленных в двоичном коде. В составе ЭВМ сумматор является ядром арифметико-логического устройства (АЛУ). На базе сумматора строятся также схемы умножения и деления (обычного и ускоренного).

Полусумматор (half-adder) – складывает два однобитовых операнда (A и B), выдает однобитовую сумму (S) и однобитовый перенос (C).

Полный сумматор (full-adder) – складывает два однобитовых операнда (A и B) и бит входящего переноса (C), выдает однобитовую сумму (S) и однобитовый перенос (C). Полный сумматор строится из двух полусумматоров.

Многоразрядный сумматор со сквозным переносом (ripple-carry adder) – складывает многоразрядные числа, стоится из полных сумматоров и полусумматоров.

Многоразрядные сумматоры также могут быть построены по иным схемам (кроме сквозного переноса).

2.2.2.2 Арифметико-логическое устройство (АЛУ)

АЛУ выполняет различные логические и математические операции над несколькими операндами. Обычно у АЛУ два операнда, представленных в двоичном коде. Типовыми операциями являются логические И, ИЛИ, НЕ; арифметические сложение, вычитание, умножение, деление; различные типы сдвигов. Выбор выполняемой операции осуществляется подачей кода на специальные управляющие входы. По результатам выполнения операций формируются специальные признаки – флаги – характеризующие результат. Например, флаги Zero (нулевой результат), Carry (перенос), Parity (четность) и другие.

АЛУ является составным БОЭ: включает сумматор, дешифраторы, мультиплексоры, комбинационные схемы установки флагов и другие составные части.

2.2.2.3 Дешифратор (Decoder)

Дешифратор - схема, которая преобразует входной многоразрядный код в отличный от него выходной многоразрядный код с равным или большим количеством разрядов.

В вычислительных устройствах наиболее применяем *двоичный дешифратор* «N в 2^N», преобразующий двоичный код в унитарный код «1 из N».

2.2.2.4 Шифратор (Encoder)

Шифратор – схема с функцией обратной дешифратору, которая преобразует входной многоразрядный код в отличный от него выходной многоразрядный код с меньшим количеством разрядов.

В вычислительных устройствах наиболее применяем *двоичный приоритетный шифратор* «2^N в N», преобразующий унитарный код «1 из N» в двоичный код.

2.2.2.5 Схемы преобразования специальных кодов

Это варианты дешифраторов и шифраторов, работающие со специальными типами кодов. Например, шифратор дешифратор кода Хемминга используются для хранения данных памяти с возможностью обнаружения и восстановления ошибки битов.

2.2.2.6 Мультиплексор (multiplexer)

Осуществляет коммутацию одного из нескольких цифровых входов на один выход. Номер подключенного входа задается двоичным кодом на специальных адресных входах мультиплексора. Входы и выходы могут быть одноразрядными или многоразрядными, т.е. коммутируется или «один бит» или многоразрядная шина.

2.2.2.7 Демультиплексор (demultiplexer)

Реализует функцию обратную мультиплексору: коммутацию одного входа на один из нескольких выходов. Номер подключенного выхода задается двоичным кодом на специальных адресных входах демультиплексора. Входы и выходы могут быть одноразрядными или многоразрядными, т.е. коммутируется или «один бит» или многоразрядная шина.

2.2.2.8 Компаратор (Comparator)

Компаратор определяет отношение между двумя цифровыми значениями (кодами). Различают компараторы равенства и компараторы отношения.

Компараторы равенства вырабатывают значение ИСТИНА (в случае ПОЗИТИВНОГО кодирования значений кодируется логической «1») на выходе в случае равенства входных кодов. Если коды не равны – на выходе ЛОЖЬ (в случае ПОЗИТИВНОГО кодирования значений кодируется логическим «0»).

Компараторы отношения вырабатывают три выходных сигнала:

- «больше» - ИСТИНА если операнд А больше операнда В;
- «меньше» - ИСТИНА если операнд А меньше операнда В;
- «равно» - ИСТИНА если операнд А равен операнду В.

Основными элементами последовательностного типа являются:

2.2.2.9 Регистры хранения (запоминающие)

Регистр хранения – это массив из нескольких триггеров с объединенными между ними управляющим входами. Соответственно, операции записи и чтения со всеми триггерами в составе регистра можно проводить одновременно (синхронно) путем подачи соответствующих сигналов на объединенные управляющие входы.

Регистры используются для хранения многоразрядных кодов. Регистры – основной тип многоразрядных ячеек памяти в составе процессоров.

2.2.2.10 Регистры сдвига

Регистры сдвига – модификация регистров хранения, которые позволяют сдвигать хранимые коды на заданное число разрядов. Сдвиг осуществляется при появлении на специальном управляющем входе импульса на один разряд. Если нужно сдвинуть на несколько разрядов – нужно подать несколько импульсов. Другой вариант – на специальном управляющем входе устанавливается значение, на которое нужно сдвинуть код, и сдвиг осуществляется сразу на нужное число разрядов. Такой способ обычно используется в составе АЛУ.

Существуют регистры сдвига вправо, влево или комбинированные. Направление сдвига выбирается отдельными управляющими сигналами.

Регистры сдвига могут использоваться в составе АЛУ (но в АЛУ чаще используются иные способы и схемы сдвига операндов). Основным применением регистров сдвига – последовательная передача данных: сохраненный в регистр код последовательно передается по одноразрядной линии путем побитового сдвига из регистра.

2.2.2.11 Блоки адресуемых регистров

Блоки адресуемых регистров представляют собой массив регистров, объединенный с управляющей логикой – дешифратором адреса. Дешифратор адреса регистра позволяет обращаться по выбору к одному из регистров в массиве через общие входные и выходные линии. Выбор регистра осуществляется посредством подачи номера регистра – адреса – на специальные управляющие входы.

Блоки адресуемых регистров основной тип блоков хранения данных в составе процессоров.

2.2.2.12 Счетчики

Счетчик – тип регистра, который увеличивает или уменьшает свое значение при поступлении на специальный вход импульсов. Т.е. счетчик считает импульсы. Счетчики построены на триггерах и дополнительном блоке управляющей логики.

Счетчики классифицируются по следующим признакам:

1) По типу кода, в котором хранится цифровое значение:
   - двоичные;
   - двоично-десятичные.
2) По коэффициенту пересчета - числовому значению, после достижения которого значение счетчика обнуляется и счет начинается с нуля.
3) По направлению счета:
   - инкрементные;
   - декрементные;
   - реверсивные.
4) По внутренней организации – несколько вариантов классификации, например, по способу переноса между разрядами хранимого кода.

Счетчики широко используются в вычислительных системах, как формирователи последовательностей синхронизирующих импульсов с определенной частотой, как

счетчики команд в процессорах, как указатели адресов при обращениях к последовательным ячейкам памяти и т.п.

2.3 Базовые принципы организации ЭВМ фон Неймана.

Джон фон Нейман – американский ученый венгерского происхождения. На основе критического анализа одной из первых ЭВМ ENIAC (к моменту его присоединения к проекту архитектура ENIAC уже была выбрана) им и его коллегами были сформулированы базовые принципы построения ЭВМ, обеспечивавшие ее эффективность, универсальность, развиваемость и удобство. В 1946 году фон Нейман вместе с Г.Гольдстейном и А.Берксом написал и выпустил отчет "Предварительное обсуждение логической конструкции электронной вычислительной машины", в котором были представлены данные принципы и базовая структура ЭВМ им соответствующая.

В дальнейшем, в работах фон Неймана и других специалистов эти принципы дополнялись, уточнялись, однако основные идеи не изменились до нашего времени под наименованием «принципы фон Неймана». На этих основах строятся большинство современных компьютеров, и даже в случае создания так называемых «не фон Неймановских ЭВМ» видоизменяются не все принципы, а только их часть, т.е. идейная база во много сохраняется.

2.3.1 Принципы фон Неймана.

Существует много вариантов «принципов фон-Неймана», полученных путем анализа комплекса его работ и работ других специалистов. Данные варианты обладают различной степенью полноты охвата аспектов организации ЭВМ. Ниже приводится, как кажется, в значительной мере полный вариант. Разделение принципов на основные и дополнительные выполнялось не по критерию времени опубликования и фактического авторства, а по принципу значимости для организации ЭВМ.

К основным принципам относим:
1. Принцип хранимой программы.
2. Принцип линейности памяти.
3. Принцип последовательного выполнения программы и возможности перехода.
4. Принцип отсутствия различий между командами и данными.
5. Принцип отсутствия различий в семантике данных.

К дополнительным принципам относим:
6. Принцип программного управления.
7. Принцип двоичного кодирования.
8. Принцип иерархической памяти.
9. Принцип низкоуровневости машинного языка.
10. Принцип микропрограммирования.

2.3.2 Каноническая структура ЭВМ.

Реализация принципов Фон-Неймана базируется на использовании ЭВМ с некоторой канонической (типовой) структурой, которая также была предложена группой Фон-Неймана. Подсистемы канонической структуры являются обязательными для любых современных ЭВМ, отвечающих (целиком или частично) принципам организации Фон-Неймана. При этом реальные современные ЭВМ имеют гораздо более сложную, многоуровневую внутреннюю организацию подсистем ЭВМ и организацию связей между ними, чем в канонической структуре.

Каноническая структура ЭВМ (см.рисунок) содержит следующие подсистемы:

- центральный процессор (ЦП), включая:
  - устройство управления (УУ);
  - операционной устройство – арифметико-логическое устройство (АЛУ);
- память, включая:
  - основную память (ОП);
  - вторичную (внешнюю) память (ВП);
- подсистему ввода-вывода, включая:
  - устройства ввода;
  - устройства вывода.

В целом принципы Фон-Неймана задают распределение функций между данными подсистемами и определяют правила взаимодействия подсистем между собой.

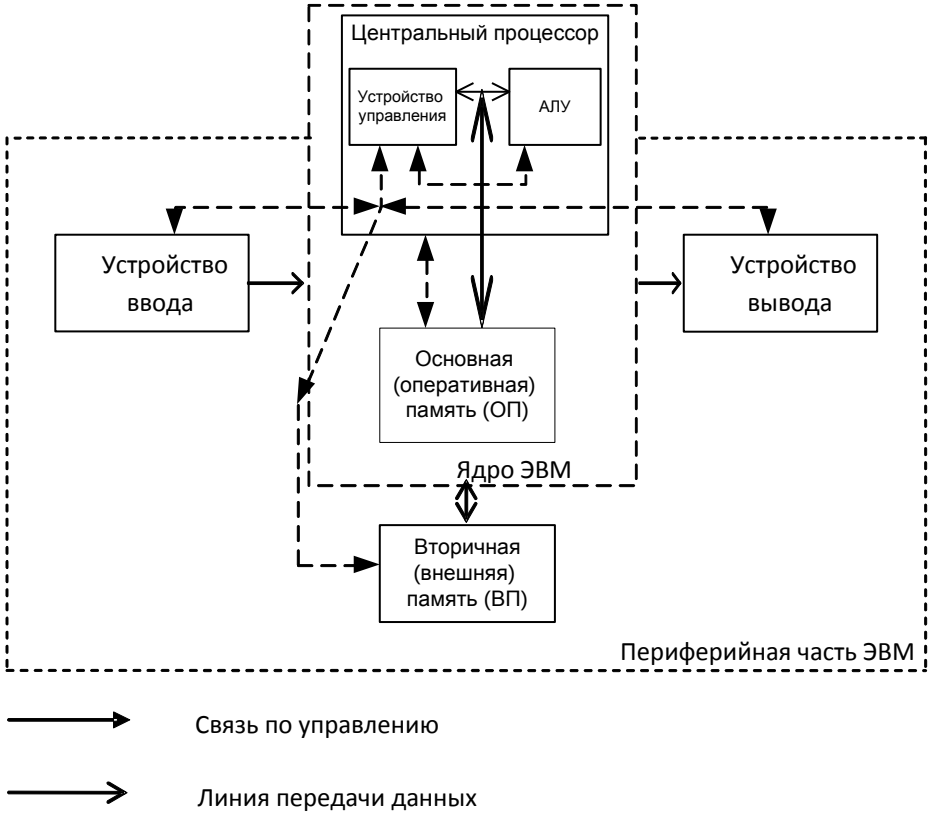

Рисунок 3 Каноническая структура ЭВМ

2.3.2.1 Ядро ЭВМ.

ЦП и ОП образуют центральную часть ЭВМ – *ядро ЭВМ*. Иногда ядро ЭВМ называют PMS-подсистемой (Processor-Memory-Switch). Ядро ЭВМ объединяет минимальный и обязательный набор компонентов, чтобы хранить и исполнять программу, хранить и предоставлять данные для программной обработки.

Основным центральный процессор (ЦП, CPU) является основным устройством ЭВМ, которое выполняет двойную функцию: с одной стороны ЦП является обрабатывающим устройством, т.к. выполняет функции по обработке данных в соответствии с заданной программой; с другой стороны ЦП является управляющим устройством, в связи с тем, что на него возлагаются функции: во-первых, по управлению программой, во-вторых, по управлению периферийными устройствами ЭВМ.

Управление программой подразумевает ее запуск, остановку, прерывание, обнаружение ошибок в процессе исполнения программы и реакцию на них. Управление периферийными устройствами (ПУ) со стороны ЦП, как правило, сводится к организации обмена между ПУ и ядром ЭВМ и к обеспечению реакции на запросы ПУ.

Основными устройствами (блоками) ЦП являются, во-первых, *АЛУ (ALU),* во-вторых, *устройство управления (CU).*

АЛУ (или, более широко, операционное устройство) реализует функции ЦП по обработке данных и предназначено для выполнения арифметических и логических операций над целыми числами, логическими значениями и символьными данными.

Функцией устройства управления (УУ) является выработка сигналов управления, с помощью которых осуществляется выполнение элементарных операций (которые называются *микрооперациями*) в АЛУ, ОП или блоках периферийной подсистемы ЭВМ.

УУ, во-первых, обеспечивает **автоматическое** выполнение команд программы, реализуя выборку команд из памяти, их декодирование, формирование адресов операндов и их выборку из памяти, настройку АЛУ на выполнение заданной операции и запись результата операции в память. Настройка и управление работой АЛУ, управление записью и чтением ОП реализуется посредством специальных электрических сигналов, которые генерирует УУ.

Второй функцией УУ является управление периферийными подсистемами ЭВМ, координация (синхронизация) их взаимодействия ядра и периферийных подсистем ЭВМ между собой. Для этого УУ также вырабатывает специальные управляющие электрические сигналы, подаваемые в устройства периферийной подсистемы.

2.3.2.2 Периферийные подсистемы ЭВМ.

Подсистемы ЭВМ, не относящиеся к ядру, называют *периферийными (под)системами*. Для исполнения программ периферийные подсистемы не являются обязательными. Их функция – обеспечить обмен программами и данными между ЭВМ и «внешним» миром.

Периферийные подсистемы обеспечивают подключение к ядру различных *периферийных устройств*, таких как:

1) Устройств ввода-вывода: для связи человека и ЭВМ (монитор, клавиатура, принтер и т.п.), для связи ЭВМ с физическими объектами (датчиков, управляемых электроприводов, реле и т.п.),
2) Устройств вторичной (внешней) памяти (НЖМД, НГМД и т.п.),
3) Устройств передачи данных (по каналам и сетям передачи данных).

В канонической структуре ЭВМ периферийные подсистемы и устройства делятся по направлению на подсистемы/устройства ввода и подсистемы/устройства вывода, однако на практике это в основном комбинированные периферийные подсистемы.

Подключение периферийных устройств ЭВМ к периферийным подсистемам осуществляется через *порты ввода-вывода* (порт – множество электрических сигналов, электронных каскадов и связанных с ними процедур соединения ЭВМ и периферийных устройств).

2.3.3 Принцип хранимой программы.

Суть данного принципа состоит в том, что команды управления процессом вычисления (в совокупности составляющие программу) могут быть закодированы в виде числовых значений (машинных команд) и сохраняться в памяти ЭВМ подобно данным. Из этого вытекает, что организация памяти для хранения команд и данных не имеет различий, т.е. и коды команд (программа) и данные могут храниться в одной памяти.

Отметим, что единая память для хранения программ и данных НЕ является ОСНОВНЫМ отличительным признаком архитектуры фон Неймана, как это часто упоминают. Можно говорить о двух разновидностях машин фон Неймана:

1) Принстонская архитектура - с общей памятью программ и данных, с единым каналом доступа к этой памяти как при считывании (выборке) команд, так и при работе с данными, и с возможностью автоматической модификации программы (т.е. «из самой исполняемой программы»). Принстонская архитектура является базовой для большинства современных универсальных процессоров (x86, Power, ARM).
2) Гарвардская архитектура – с раздельной памятью для хранения программ и данных и, соответственно, с раздельными каналами доступа к этим областям памяти, с недоступностью автоматической модификации программ.

   Конечно, строго говоря, в гарвардской архитектуре есть различия для программных кодов и данных с точки зрения их хранения, что формально не соответствует принципам Фон-Неймана. Но, с другой стороны, принцип *хранения программы в памяти* соблюден.

   В современных ЭВМ с гарвардской архитектурой программная недоступность памяти программ выполняется в ограниченном объеме: с памятью данных можно работать посредством «обычных» машинных команд, а доступ к памяти программ также может осуществляться, но в специальных режимах работы процессора или с помощью специальных ПОСЛЕДОВАТЕЛЬНОСТЕЙ команд.

   Принципы гарвардской архитектуры памяти заложены:
   - в организацию КЭШ-памяти многих высокопроизводительных процессоров (например, в процессорах Intel x86 начиная с Pentium существуют блоки с независимым доступом: КЭШ-команд (Instruction Cache) КЭШ-данных (Data Cache));
   - в организацию основной памяти многих специализированных процессоров, прежде всего – микроконтроллеров для систем управления. Это обеспечивает высокий уровень надежности/безопасности (защита от само модификации кода) и повышение производительности при ограниченных аппаратных ресурсах.

2.3.4 Принцип линейности памяти.

Принцип состоит в том, что память ЭВМ представляет совокупность упорядоченных ячеек, каждая их которых идентифицируется (адресуется) индивидуальным адресом.

Совокупность адресов всех ячеек образует линейное – неразрывное – адресное пространство, где ячейки памяти упорядочены по возрастанию их адресов. Соответственно память канонической Нейманновской архитектуры можно рассматривать как вектор ячеек.

Принцип линейности памяти является ключевым для организации последовательного исполнения программы с помощью счетчика команд (см. ниже) и для организации структур данных без явных указателей элементов.

Понятие «линейности» относится к способу адресации ячеек памяти, а не к способу их «физического расположения»: ячейки с соседними адресами могут располагаться «в различных углах» кристалла микросхемы памяти или даже в различных миросхемах памяти.

Само понятие «ячейки памяти» не является однозначным: с одной стороны под ним понимается совокупность запоминающих элементов (один элемент обычно хранит один бит данных), идентифицируемых одним и тем же адресом. Так как минимальной адресуемой единицей данных в большинстве современных вычислительных систем является байт из 8-ми бит (но могут быть и другие варианты), то в этом смысле байт будет рассматриваться как ячейка памяти. Такой взгляд ближе к логической организации ЭВМ.

С другой стороны, под ячейкой памяти иногда понимают совокупность запоминающих элементов, параллельно участвующих в одной и той же операции с памятью: записи или чтения. В этом смысле в ячейку будут образовывать биты, одновременно передаваемые по шине данных от или к памяти. Соответственно, если шина 16-разрядов, то и ячейка 16-разрядов, если шина памяти 32-разряда – то и ячейка 32-разряда. Такой взгляд – ближе к аппаратной организации ЭВМ.

Наконец, третий взгляд: ячейка – это элемент данных в памяти адресуемый командой. В этом смысле понятие ячейки совпадет с понятием слова памяти. Такой взгляд на понятие ячейки – функциональный.

2.3.5 Принцип последовательного выполнения программы и возможности перехода.

Этот принцип означает, что выполнение любой программы в ЭВМ сводится к выполнению ее (программы) команд в порядке возрастания их номера – адреса расположения кода команды в памяти. Это избавляет от применения в составе команд указателей следующей команды, чем упрощает устройство выборки команды в составе процессора и экономит память на хранение программного кода.

Данный принцип управления исполнением программы также называют принципом *управления потоком команд*. Альтернативные принципы – управление потоком данных и потоком запросов (см. ниже).

Аппаратная поддержка принципа последовательного выполнения программы реализуется в виде счетчика команд (для его обозначения используют английские аббревиатуры IP (Instruction Pointer) или PC (Program Counter)). Счетчик команд – аппаратный двоичный счетчик, увеличивающий свое значение с каждым очередным периодом сигнала синхронизации. Это значение используется как адрес следующей команды в линейно организованной памяти.

Линейная последовательность выполнения команд может быть нарушена при необходимости путем исполнения команд условного (JA, JBE и т.п.) или безусловного (JMP) перехода, команд циклов (LOOP и т.п.), команд вызова и возврата из процедур (CALL, RET). Все эти команды изменяют (переписывают) значение счетчика команд, а значит и адрес следующей исполняемой команды.

Еще одним случаем нарушения линейной последовательности исполнения команд в программе является обработка прерываний: по сути это исполнение команды вызова процедуры инициированное устройством управления процессора в специальных случаях: по аппаратным сигналам запроса прерывания, по специальным командам вызова «программных прерываний», при возникновении ошибки в работе процессора и т.п.

В современных процессорах широко используется механизм выполнение команд программы не в порядке их следования – Out of Order Execution. Однако, это не означает, что нарушен неймановский принцип последовательного выполнения команд: результаты выполненных не по порядку команд сохраняются в памяти в соответствии с заданной программой последовательностью. Для этого используется специальный блок In Order Retirement Unit. Пример такого механизма можно найти в составе процессоров серии Intel P6, начиная с модели Intel Pentium Pro.

2.3.5.1 Не Нейманновские принципы управления программами.

Альтернативой Нейманновского принципа управления потоком команд являются:
- Принцип управления потоком данных, лежащий в основе потоковых ЭВМ;
- Принцип управления потоком запросов, лежащий в основе редукционных ЭВМ.

2.3.5.2 Мультитредовая и гипертредовая организация выполнения программ.

3 Организация процессора

3.1 Классификация процессоров.

Существуют различные варианты классификации процессоров по различным критериям: по реализуемой модели вычислений (model of computation: управляемые потоком команд (Command Driver - фон-Неймановская архитектура), потоком данных (Data Driver), потоком запросов (Demand Driver) и др.), по составу и типу системы команд (CISC, RISC, VLIW), по уровням распараллеливания программ (параллелилизм на уровне машинных команд (ILP), параллелилизм на уровне потоков (MultiThreading), параллелилизм на уровне памяти (MPL) и др.), по месту хранения операндов.

Способ хранения операндов – базовая характеристика, непосредственно определяющая:

- прикладную архитектуру (архитектуру системы команд);
- аппаратную структуру процессоров.

Данная характеристика применима к процессорным архитектурам любых типов по иным вариантам классификации.

По способу хранения операндов выделяют следующие архитектуры процессоров [1]:

1) аккумуляторную;
2) регистровую;
3) с выделенным доступом к памяти;
4) стековую.

3.1.1 Аккумуляторная архитектура

Имеется специальный регистр *аккумулятор*, используемый в большинстве машинных команд как источник одного из операндов и место сохранения результата.

За счет этого в команде нужно хранить только адрес одного из операндов – адрес второго операнда и результата предопределены (это аккумулятор). Код машинной команды за счет этого становится короче, его проще декодировать. Аппаратная структура таких процессоров проста: используется только один канал для считывания данных из основной памяти, а остальные пересылки выполняются между аккумулятором и АЛУ.

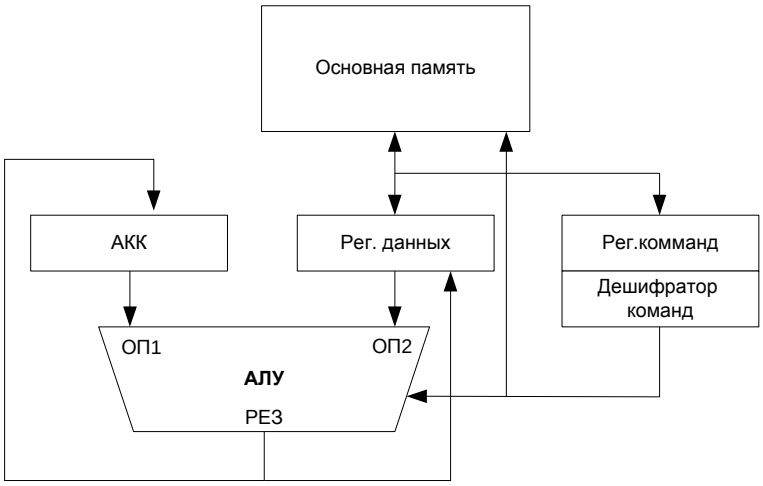

Рисунок 4 Обобщенная структура процессора с аккумуляторной организацией

Недостатком аккумуляторной архитектуры является большое количество дополнительных команда передачи данных из памяти в аккумулятор: прежде чем выполнить операцию аргумент нужно загрузить в аккумулятор.

3.1.2 Регистровая архитектура.

Процессор включает в себя массив регистров – регистровый файл или регистры общего назначения. Операнды и результаты могут располагаться в любом из этих регистров. Так как количество регистров относительно невелико (8-32 для CISC-процессоров, до нескольких сотен для RISC-процессоров), то адресное поле также имеет небольшой размер (порядка 3-8 бит), что позволяет в одной команде указывать адреса 2-3 операндов/результатов. Как и в аккумуляторной архитектуре, прежде чем обрабатывать операнды, их нужно загрузить в регистровый файл из основной памяти. Однако многие процессоры поддерживают арифметические и логические операции не только типа «регистр-регистр», но также «регистр-память» и даже «память-память». Такие операции выполняются медленнее (требуется загрузка/сохранение операндов из/в памяти), но позволяют получить более компактный код (одна команда вместо нескольких).

Аппаратная структура регистровых процессоров более сложная чем для аккумуляторной архитектуры: кроме самого регистрового файла требуется более сложный декодер команд, устройство формирования физических адресов для регистрового файла, более многочисленными и разветвленными становятся каналы (шины) передачи данных между блоками внутри процессора. Например, между АЛУ и регистровым файлом должно быть минимум три шины: две для чтения операндов и одна для записи результатов.

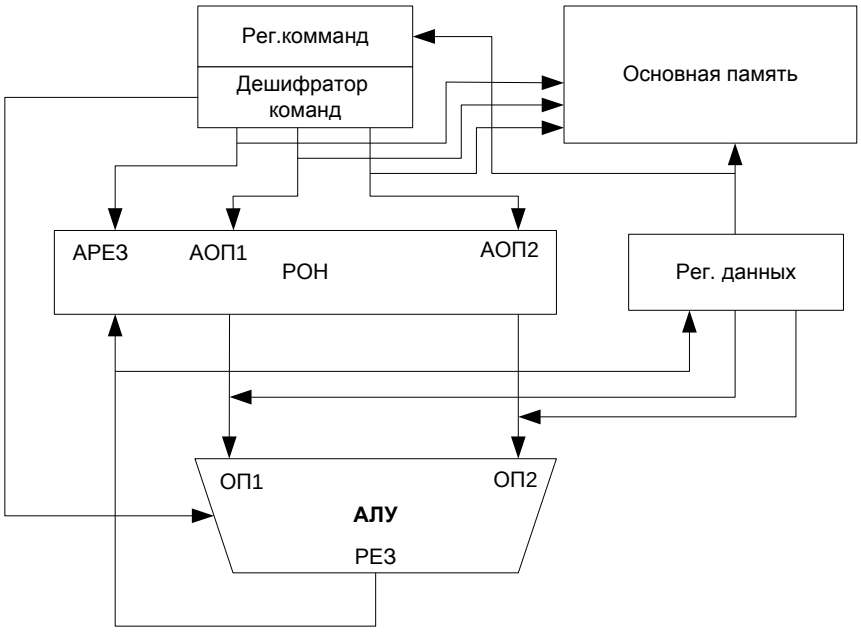

Рисунок 5 Обобщенная структура процессора с регистровой организацией

На практике наиболее часто используется именно регистровая архитектура, а также смешанная регистрово-аккумуляторная архитектура.

3.1.3 Архитектура с выделенным доступом к памяти.

Данная архитектура является развитием регистровой архитектуры, которая обеспечивает упрощение внутренней структуры процессора, для уменьшения размера и унификации формата команд. Команды обработки данных могут работать ТОЛЬКО с регистрами. Обрабатывающие команды типа регистр-память и память-память не допускаются. Перед/после обработкой(-и) данные должны быть загружены из памяти с помощью специальных команд LOAD и STORE, подобно аккумуляторной архитектуре. Так как в данных командах максимум два (вместо трех) операнда, то на адрес памяти можно выделить в два раза большее поле, не наращивая размер команды. Чтобы минимизировать число пересылок с памятью, делают регистровый файл относительно большого объема (несколько сот байт), так что операнды и результаты пересылаются из(в) память только в начале и в конце крупных процедур.

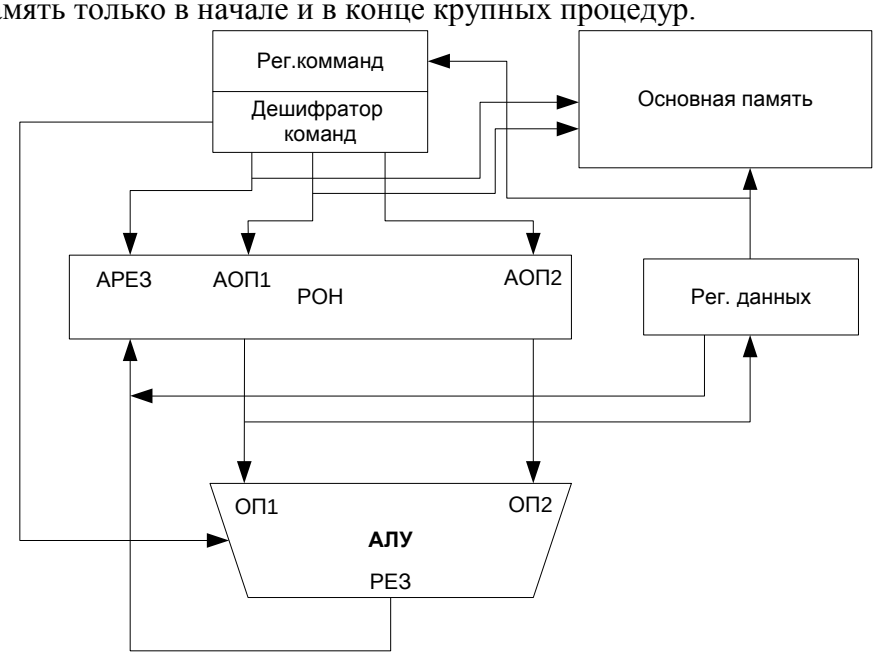

Рисунок 6 Обобщенная структура процессора с выделенным доступом к памяти

Архитектура память-память (или Load/Store architecture) реализована во многих современных RISC-процессорах (SPARC, MIPS и др.)

3.1.4 Стековая архитектура.

Данная архитектура стоит несколько особняком. Стек – регистровый файл (блок памяти) работающий по принципу LIFO (последний вошел – первый вышел). В архитектуре процессора стек используется в качестве регистрового файла, подобно иным типам процессоров. При этом данные в стек записываются/считываются из основной памяти с помощью команд push и pop.

Особенностью стековых машин является то, что для выполнения операций в стековых машинах требует сохранение данных в стеке в специальном порядке, в соответствии с дисциплиной доступа LIFO. Данный порядок сохранения данных и выполнения операций называется обратная польская запись (разработана польским математиком Я.Лукашевичем). Например, выражение a=a+b+a*c будет записано как a=ab+ac*+ и в таком же порядке выполняются загрузки в стек и выполнение операций.

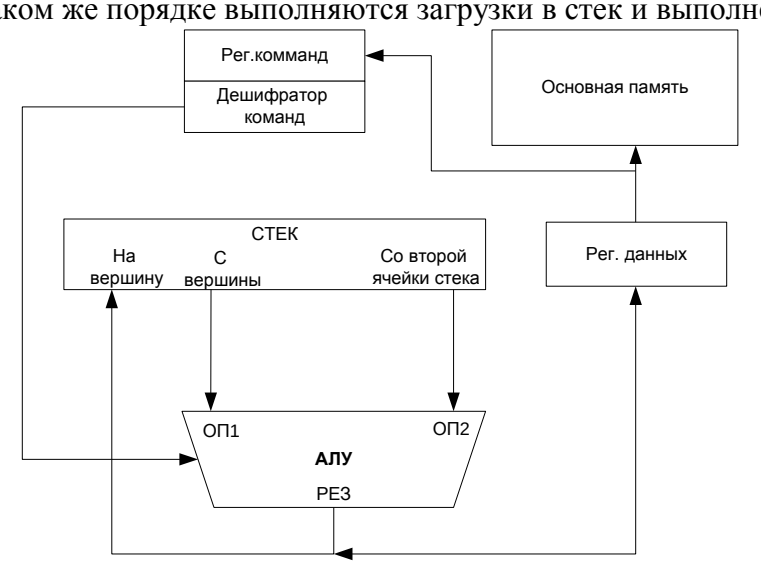

Рисунок 7 Обобщенная структура стекового процессора

Достоинством стековой архитектуры является малый размер кода команды (нет адресной части, кроме команд обмена между стеком и памятью), наличие формальных методов компиляции программ (через аппарат обратной польской записи). Также стековая машина имеет простую аппаратную структуру.

Недостатком стековой машины является наличие узкого места – стека, отсутствие произвольного доступа к памяти, сложность оптимизации кода.

Стековые архитектуры имеют давнюю историю и последнее время вновь «возродились» в связи ростом популярности языков Java и Forth и созданием соответствующих процессоров.

3.2 Схема тракта данных (datapath) процессора.

Для общего понимания функциональной и аппаратной организации процессоров и алгоритмов исполнения машинных команд процессором, удобно использовать схему (модель) тракта данных (datapath) процессора.

Схема тракта данных – это структурно-функциональная схема, на которой показаны:

- Совокупность функциональных блоков, последовательно обрабатывающих данные. К данным относятся и обрабатываемые данные, и коды команд и адреса.
- Каналы, по которым слова данных ПОСЛЕДОВАТЕЛЬНО передаются внутри процессора от одного обрабатывающего блока к другому.
- Коммутационные блоки и сигналы, управляющие направлением и порядком передачи данных между обрабатывающими блоками.

Отличительной особенностью схем трактов данных является то, что явно показана ПОСЛЕДОВАТЕЛЬНОСТЬ обработки и передачи данных от блока к блоку и по такой схеме можно составить алгоритм исполнения машинных команд. С другой стороны. модель явно показывает аппаратную структуру процессора.

Пример обобщенной (без всех деталей организации процессора) схемы тракта данных процессора приведен на рисунке.

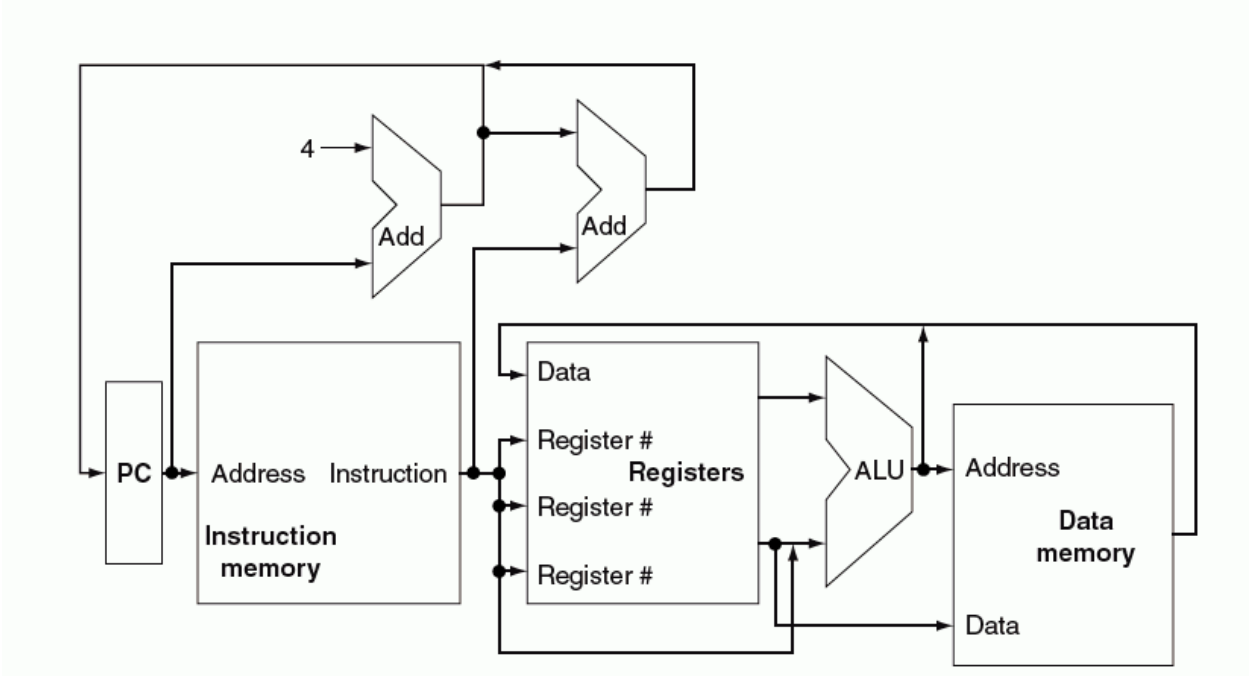

Рисунок 8 Обобщенная схема тракта данных процессора

В этой схеме алгоритм исполнения команд определен направленными связями между блоками и будет следующим:

1) Исполнение всех команд начинается с подачи адреса команды из программного счетчика PC на вход адреса (Adress) блока памяти команд (Instruction memory);
2) Из памяти команд через выход Instruction считывается команда.
3) Исполнение команды:
    a. Если выбрана команда, исполняющая операцию со значениями регистров, то из полей адресов операндов и результата адреса соответствующих регистров подаются на адресные входы (Register#) блока регистров (Registers). Далее операнды, считанные из блока регистров, подаются на вход АЛУ, выполняется требуемая операция и результат сохраняется в регистре результата через вход Data.
    b. Если выбрана команда с «непосредственным» операндом, то его значение подается на вход АЛУ «в обход» блока регистров и далее команда выполняется как в п.3.
    c. Если выбрана команда записи или чтения из памяти данных (Data Memory), то адрес для памяти данных вычисляется сложением значения базового регистра (считывается из блока регистров) и абсолютного смещения (указывается в непосредственно команде). Сложение выполняется с помощью АЛУ, полный адрес подается на вход адреса (Adress) блока памяти данных (Data Memory). Данные записываются в память данных через вход Data блока Data Memory или считываются из внешней памяти через выход блока Data Memory и записываются в регистр через вход Data блока Registers.
    d. Если выбрана команда перехода по относительному адресу, то смещение адреса подается на блок сумматора (Add) для вычисления адреса перехода.

4) Адрес следующей команды записывается в программный счетчик:
   a. Если выполняется следующая команда, то ее адрес рассчитывается как адрес предыдущей команды +4 (если считать, что одна команда занимает 4-е байта; если размер команды иной, то прибавляться будет другое число). Суммирование выполняется с помощью блока сумматора Add;
   b. Если выполняется переход, то записывается адрес перехода, вычисленный в соответствии с п.3.d.

Приведенная выше схема использована как сильно упрощенный пример и не определяет следующие важные аспекты организации процессора:

1) На один вход блока могут подаваться данные от нескольких источников (выходов других блоков). Например, на нижний вход АЛУ данные поступают с выхода блока регистров и с выхода блока памяти команд. Реальные цифровые компоненты не допускают такого подключения, так как напряжения на двух соединенных выходах могут оказаться различными и это приведет к короткому замыканию потенциалов и выходу схемы из строя.
2) Не определены управляющие сигналы, которые должны указывать, по какому из возможных путей выполняется передача данных в конкретный момент (это зависит от выполняемой команды), и должны настраивать реконфигурируемые операционные блоки (например, АЛУ) на выполнение требуемой в данный момент операции (например, для АЛУ: сложение, вычитание, умножение, деление или сдвиг).

Более детальный вариант тракта данных, в котором устранены эти недостатки, приведен на рисунке ниже. Здесь выбор источников для входов функциональных блоков выполняется при помощи мультиплексоров (MUX), а также приведены управляющие сигналы для мультиплексоров, АЛУ, блоков регистров и памяти. В целом за корректную последовательность подачи управляющих сигналов отвечает специальный блок управления Control. Данный вариант будет подробно рассмотрен ниже.

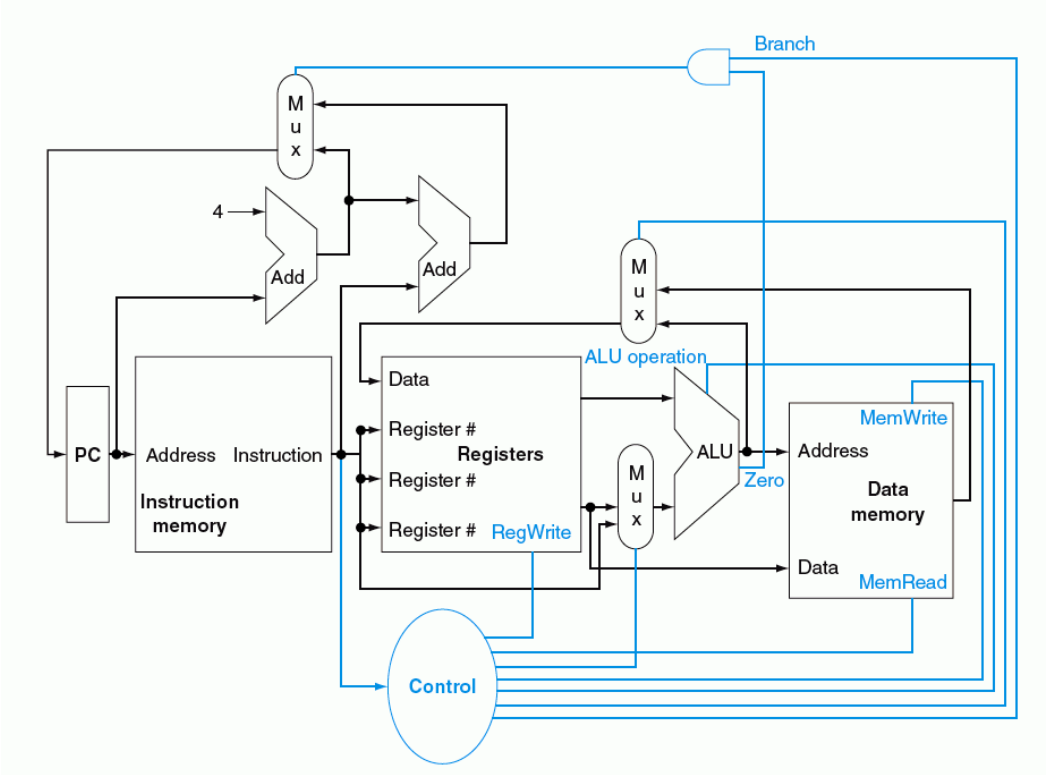

Рисунок 9 Детализированная схема тракта данных процессора.

3.3 Компоненты тракта данных процессора.

Далее будут рассмотрены основные компоненты тракта данных процессора - функциональные блоки в составе процессоров. Данный набор блоков не является полным: могут использоваться иные типы блоков, но представленные являются ключевыми, применяемыми фактически в любом процессорном ядре.

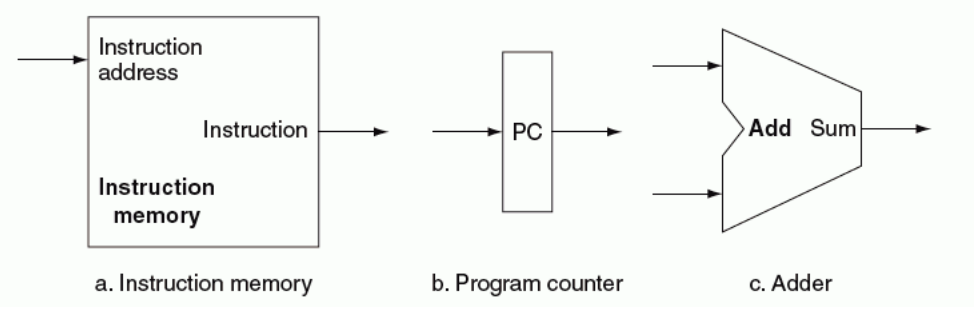

Рисунок 10 Функциональные блоки процессора: память команд (a), счетчик команд (b), сумматор (c).

3.3.1 Память команд (Instruction memory).

Память команд (см. Рисунок 13) предназначена для хранения кодов команд исполняемых процессором. Имеет один многоразрядный адресный вход (входной порт), на который подается адрес расположения команды в памяти, и выход, через который считывается команда. Процесс определения адреса команды и считывания команды из памяти называется *выборка команды*.

3.3.2 Сумматор (Adder).

Сумматор – осуществляет арифметическое сложение двух двоичных чисел. Сумматор – комбинационная цифровая схема. Сумматор можно рассматривать как упрощенное АЛУ, на управляющем порту которого постоянно установлен код операции сложения. В составе процессора сумматоры как отдельные (не в составе АЛУ) функциональные блоки используются, например, для вычисления физического адреса в памяти данных из базового адреса и смещения, для увеличения (инкремента) счетчика команд, особенно в CISC-процессорах с переменной длиной команды и в других случаях.

3.3.3 Счетчик команд

Счетчик команд (см.Рисунок 10), англ. аббревиатуры IP (Instruction Pointer) или PC (Program counter) - специальный регистр, содержащий адрес текущей команды (инструкции), по которому эта команда считывается (выбирается) из памяти команд (памяти программ). С точки зрения функции счетчик команд правильнее называть указателем команд (IP).

В режиме последовательной выборки команд счетчик команд после выборки очередной команды увеличивается на размер команды, чтобы при следующей выборке команды указывать на следующую команду. При выполнении команд перехода адрес перехода переписывается «поверх» текущего значения счетчика команд.

3.3.4 Регистровый файл.

Регистровый файл (см.Рисунок 11) – обязательный блок процессоров с регистровой организацией и их модификаций.

Регистровый файл – это блок, состоящий из нескольких регистров, предназначенных для хранения данных, обрабатываемых процессором. В каждый момент времени может совершаться операция чтения или записи только с одним регистром, адрес которого (двоичный номер регистра) подается на специальный адресный вход.

Для начального ознакомления с организацией процессора будет использован регистровый файл с несколькими параллельными трактами (под трактом понимаем адресный вход со связанным с ним входом/выходом данных) доступа к данным: два

тракта чтения данных (Read register1 + Read data1; Read register2 + Read data2) и тракт записи данных (Write register + Write data). Такая организация регистрового файла позволит выполнять операции с несколькими операндами из регистрового файла за один такт, т.е. за один такт работы процессора будут считаны два операнда из регистрового файла, обработаны (в АЛУ) и записан результат также в регистровый файл.

На практике - в реальных процессорах - регистровый файл может быть организован иным образом, например, с одним входом адреса, одним входом записываемых данных и одним выходом считываемых данных. Это значительно упрощает его электрическую схему, но потребует разделить выполнение машинных операций во времени на несколько шагов, что поведет к усложнению организации процессора.

С другой стороны, так как регистровый файл – внутренний блок процессора, участвующий в выполнении большинства машинных операций, то он должен обеспечивать высокую скорость доступа и чем меньше тактов на операцию у него будет, тем лучше.

3.3.5 Арифметико-логическое устройство (АЛУ).

АЛУ (см.Рисунок 11) – основной блок процессора, выполняющий арифметические и логические операции. Имеет два входа для ввода операндов, один выход ALU result для считывания результатов, многоразрядный конфигурационный вход ALU operation для выборки операции (сложение, умножение, логическое сложение, сдвиг или другой), которая выполняется АЛУ в данный момент. Кроме перечисленных АЛУ обычно имеют выходы признаков результатов, например, признак нулевого результата Zero, или признак четности результата Parity и другие.

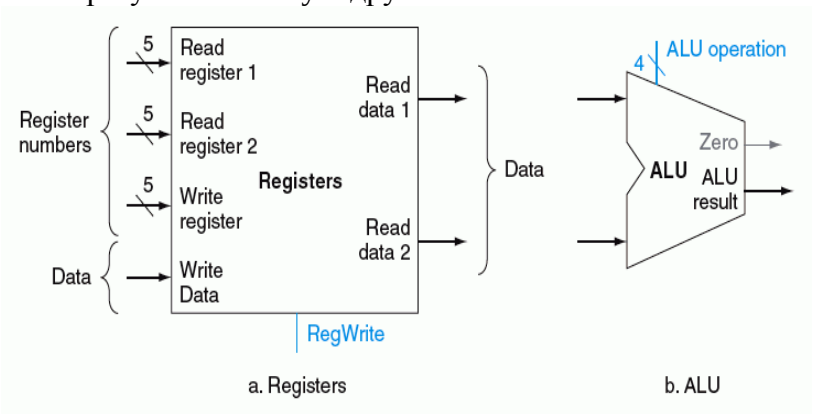

Рисунок 11 Функциональные блоки процессора: регистровый файл (a), АЛУ(b).

3.3.6 Память данных

Блок памяти данных (Data memory) (см. Рисунок 12) предназначен для хранения операндов и результатов. Его функции аналогичны регистровому файлу, но, так как память данных имеет значительный объем и соответственно большое количество разрядов шины адреса Adress, а также физически реализуется как внешний по отношению к процессору блок (отдельная микросхема или модуль), то память данных не может быть оснащена несколькими трактами данных – будет слишком много выводов у модуля памяти и слишком много связанных с ними проводников на печатной плате ЭВМ.

Поэтому память данных оснащают только совмещенным трактом данных: общий вход адреса (Adress) для записываемых и считываемых данных, один вход записываемых данных (Write data) и один вход считываемых данных (Read Data) (Такая организация портов данных также несколько упрощена: в реальных ЭВМ память обычно оснащается только одним общим портом записи/чтения данных).

Так как вход адреса общий для записи и чтения, то требуется использовать отдельные управляющие входы, показывающие, какую операцию выполняем в данный конкретный момент: запись – подается импульс на вход MemWrite или чтение – подается импульс на вход MemRead.

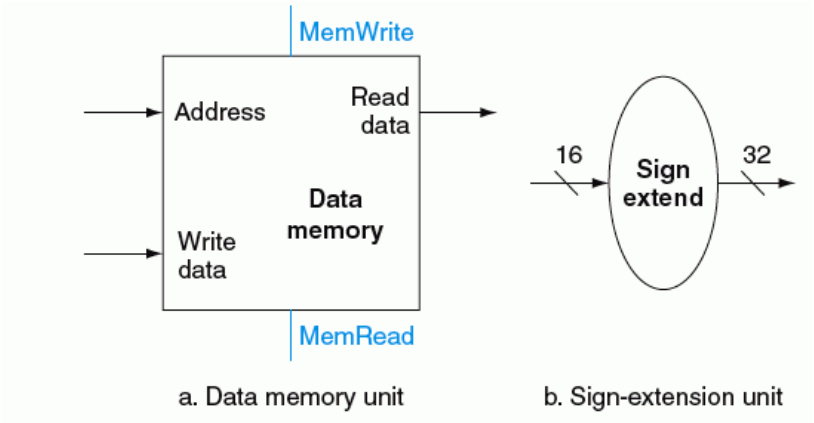

Рисунок 12 Функциональные блоки процессора: память данных (а), знаковый расширитель (б).

3.3.7 Знаковый расширитель.

Блок знакового расширителя (Sign extend) (см. Рисунок 12) предназначен для преобразования знакового двоичного числа с меньшей разрядностью в двоичное число с большей разрядностью. Так как числа хранятся в дополнительном коде, то преобразование заключается в дублирование старшего бита во все вновь добавляемые разряды. Например, если 16-разрядное число преобразуется в 32-х разрядное, то в случае положительного числа в старшие разряды записываются «0», а в случае отрицательного числа – «1».

Изменение разрядности числа необходимо, если требуется сложение двух чисел с различной разрядностью, например (см. 3.4.1), при вычислении адреса следующей команды при выполнении команды перехода (Branch).

3.3.8 Блоки сдвига (сдвигатели).

Блоки сдвига (Shift N) используются для выполнения операции сдвига на N-разрядов. Это может быть необходимо, например, в следующем случае (см. 3.4.1) (но также и в других случаях): если в команде перехода указан относительный адрес (смещение) перехода в машинных командах и при этом каждая команда состоит из нескольких байт, то сначала смещение в командах преобразуется в смещение в байтах путем перемножения размера команды на смещение в командах, а потом уже выполняется остальной расчет полного адреса перехода. Если размер команды кратен 2^k (степени числа 2), то данное умножение удобно делать посредством сдвига вправо на k-разрядов, выполняемого при помощи схемы сдвига.

3.4 Функциональная организация однотактового процессора.

Далее будет рассмотрена организация тракта обработки данных простейшего однотактового процессора применительно к различным классам машинных операций. За тем будет рассмотрен объединенный тракт данных, реализующий инструкции различных типов.

Данная реализация тракта обработки данных предполагает:

1) Выполнение всех машинных операций за один такт. Это означает, что в процессе выполнения операции не выделяется нескольких последовательных шагов – микроопераций.
2) В рамках одной машинной операции один (каждый) блок имеет только одну функцию: нельзя, например, АЛУ использовать в одной операции и для сложения операндов и для расчета физического адреса в памяти.
3) Один и тот же блок может иметь различное назначение при выполнении различных операций. Например, при операции сложения АЛУ используется для сложения

операндов, а при доступе к основной памяти данных - для расчета физического адреса в памяти.

4) При реализации различных операций функциональные блоки могут быть по разному соединены друг с другом. Для перекоммутации путей передачи данных к/от блоков используются блоки коммутации цифровых данных – мультипрексоры.
5) Память команд и память данных раздельные.

Для демонстрации организации процессора выбрана прикладная архитектура MIPS (*Microprocessor without Interlocked Pipeline Stages* — «микропроцессор без блокировок в конвейере, разработано компанией MIPS Technologies). Архитектура MIPS – классическая RISC, регистровая с раздельным доступом к памяти, имеет фиксированный размер команды, ограниченное число типов машинных команд, ориентирована на фиксированную длительность и максимально одинаковый порядок исполнения команд всех типов (так как данная архитектура строго ориентирована на конвейерную организацию). Архитектуру MIPS можно реализовывать поэтапно: в минимальном варианте – без конвейера и далее – добавляя конвейер и иные сложные механизмы (контроллер прерываний, контроллер памяти и т.п.).

Все это делает удобной архитектуру MIPS для поэтапного ознакомления с принципами организации процессоров.

Далее будет рассмотрена реализация подмножества MIPS-системы команд для работы *с целыми числами* в следующем объеме:

1) Команды чтения и записи слова в основную (не регистровую) память: load word (lw), store word (sw).
2) Арифметические и логические команды add - сложение, sub - вычитание, and – логическое умножение, or – логическое сложение, slt (set of less that) – сравнение регистра со значением.
3) Команды переходов: beq - переход если равно, безусловный переход – j.

3.4.1 Инкремент счетчика команд.

Инкремент значения счетчика команд для получения адреса очередной инструкции выполняется следующим образом: используется регистр и дополнительный блок сумматора, добавляющий к текущему значению счетчика команд длину команды (для архитектуры MIPS это +4). Результат записывается в счетчик команд (см.Рисунок 13). Данный вариант механизма инкремента позволяет добавлять любые длины команд, в том числе и изменяющиеся от команды к команде (характерно для CISC-процессоров). Однако этот механизм имеет достаточно сложную структуру (включает регистр и сумматор) и функционированию (несколько шагов считывание-инкремент-запись).

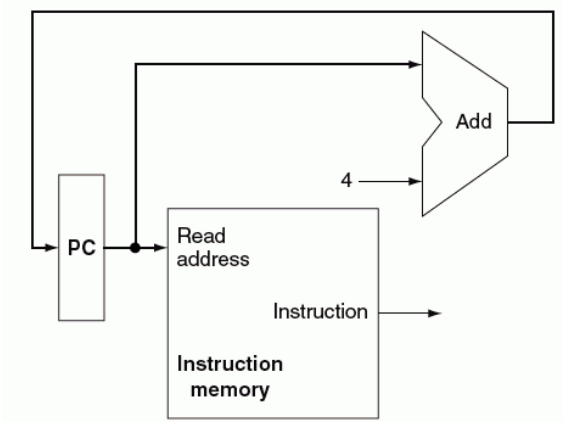

Рисунок 13 Организация счетчика команд на базе регистра и сумматора

Поэтому можно использовать альтернативный вариант - схему на основе двоичного счетчика. В данном случае инкремент счетчика будет выполняться «автоматически» при подаче очередного импульса синхросигнала на счетный вход двоичного счетчика.

3.4.2 Реализация переходов.

Команды переходов могут быть:

1) Условные. Будем рассматривать команду *beq $s1, $s2, L* – переход на L команд относительно очередного значения счетчика команд, если равно содержимое регистров s1 и s2. Признаком равенства операндов будет установка флага Zero.
2) Безусловные. Будем рассматривать команду *j L* – переход на L команд относительно очередного значения счетчика команд.

На рисунке (см.Рисунок 14) показан вариант тракта данных, реализующий команды переходов. Адреса сравниваемых операндов s1 и s2 подаются на адресные входы ReadRegister1 и ReadRegister2 регистрового файлаб а соответствующие значения для сравнения считываются с выходов ReadData1 и ReadData2. Эти значения подаются на вход АЛУ и сравниваются путем вычитания (или логической операции XOR). Соответствующий код операции подается на управляющий вход ALU operation. По результатам устанавливается флаг Zero (если операнды равны – «1», не равны – «0»). Полученное численное значение с выхода данных АЛУ не используется.

Переход выполняется: для команды *beq* если Zero = 1, для команды *j* – всегда. Расчет адреса перехода выполняется следующим образом. Значение L из команды является смещением следующей исполняемой команды относительно адреса текущей команды. L измеряется в командах (не в байтах), является 16-ти разрядным знаковым целым, в дополнительном коде.

Поэтому: для вычисления адреса следующей команды при выполнении команды перехода к 32-разрядному текущему значению счетчика команд нужно прибавить 16-ти разрядный относительный адрес перехода. Сначала 16-ти разрядное значение L преобразуется в 32-разрядный с помощью знакового расширителя. Далее, полученное значение приводится к байтовому адресу путем умножения на байтовый размер команды. равный 4 байта: эта операция выполняется при помощи сдвига на 2 бита влево посредством аппаратного блока сдвига Shift left 2. Наконец складываются два 32-разрядных числа (счетчик команд и смещение) и получаем полный адрес следующей команды (Branch Target), переписываемый в счетчик команд.

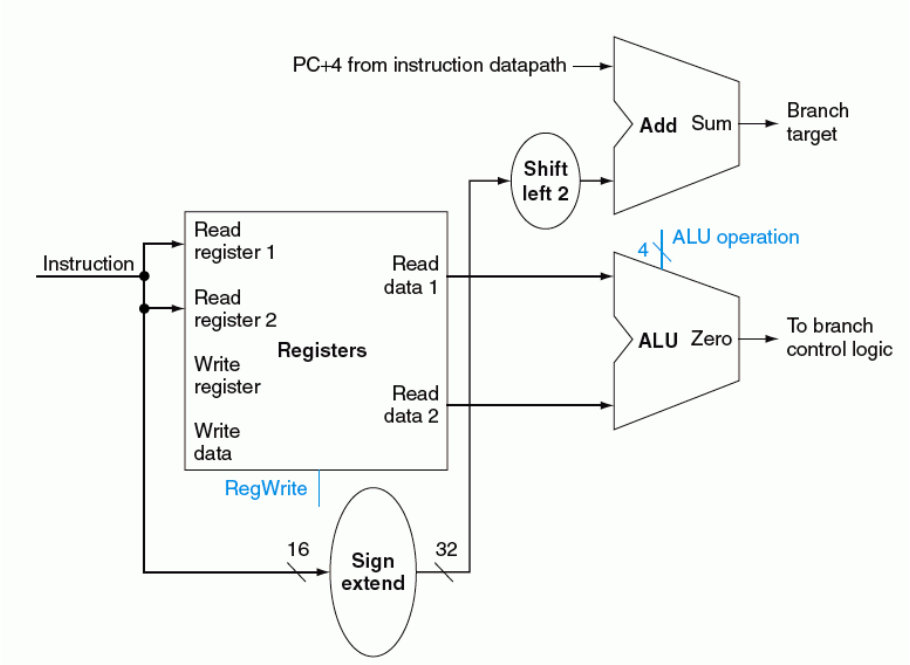

Рисунок 14 Пример тракта данных для реализации команд перехода.

3.4.3 Реализация операций с регистровым файлом и основной памятью.

Как было показано выше (см. п.3.2 и Рисунок 8), АЛУ может использоваться как для работы с операндами из регистрового файла при выполнении арифметических, логических команд и команд условного перехода, так и для расчета адреса основной памяти для команд lw и sw. Однако для такого «разделения между операциями» в тракт данных требуется ввод коммутирующих блоков – мультиплексоров. Пример подобного тракта показан на рисунке (см.Рисунок 15).

Здесь с помощью мультиплексора, управляемого сигналом ALUSrc выбирается операнд на втором входе АЛУ: либо содержимое регистра для арифметических/логических операций (ALUSrc = 0), либо значение смещения слова в памяти, взятое непосредственно из кода операции (ALUSrc = 1). На второй (верхний) вход АЛУ подается соответственно или второй операнд для арифметических/логических операций или базовый адрес для доступа к памяти. Так как и то и другое считывается из регистрового файла, то в данном случае мультиплексора не требуется.

Второй мультиплексор на схеме, управляемый сигналом MemtoReg, определяет какой результат будет записан в регистровый файл: если выполняется арифметическая/логическая операция, то записывается значение с выхода АЛУ (MemtoReg = 0), если считываются данные из памяти (команда *sw*), то записываются данные с выхода памяти ReadData (Memtoreg=1).

Применительно к рассмотренным командам сигналы ALUSrc и MemtoReg модут быть объединены, однако, учитывая возможность реализации иных команд, оставим их раздельными. Например, для арифметических операций с непосредственным (значение из кода операции) операндом и записью в регистровый файл потребуется ALUSrc = 1, а MemtoReg = 0.

Остальные управляющие сигналы на управляющих входах функциональных блоков были описаны ранее: MemWrite и MemRead выбираются, соответственно, для команд *sw* и *lw*. ALU operation – устанавливает тип арифметической или логической операции. RegWrite – тактирует запись рзультата в регистровый файл по его (регультата) готовности.

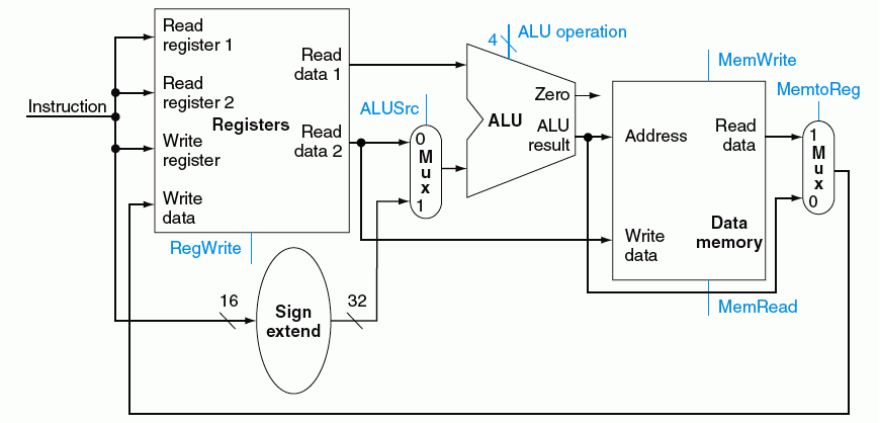

Рисунок 15 Тракт данных для операций с памятью и регистровым файлом.

3.4.4 Объединенный тракт обработки данных.

Тракт обработки данных, ориентированный на реализацию всех описанных выше команд и представляющий собой композицию описанных выше трактов данных, реализующих отдельные команды, представлен на рисунке (см. Рисунок 16). Функционирование отдельных его компонентов было описано в предыдущих параграфах.

Управляющие сигналы на управляющих входах всех блоков генерируются блоком (устройством) управления (см. Рисунок 9, не представлена на Рисунок 16).

Модель тракта данных является шаблоном для последующей реализации процессора в виде цифровой схемы. Аппаратная структура процессора в целом будет соответствовать модели тракта данных. Далее будут рассмотрены основные этапы реализации данного тракта в виде цифровой схемы.

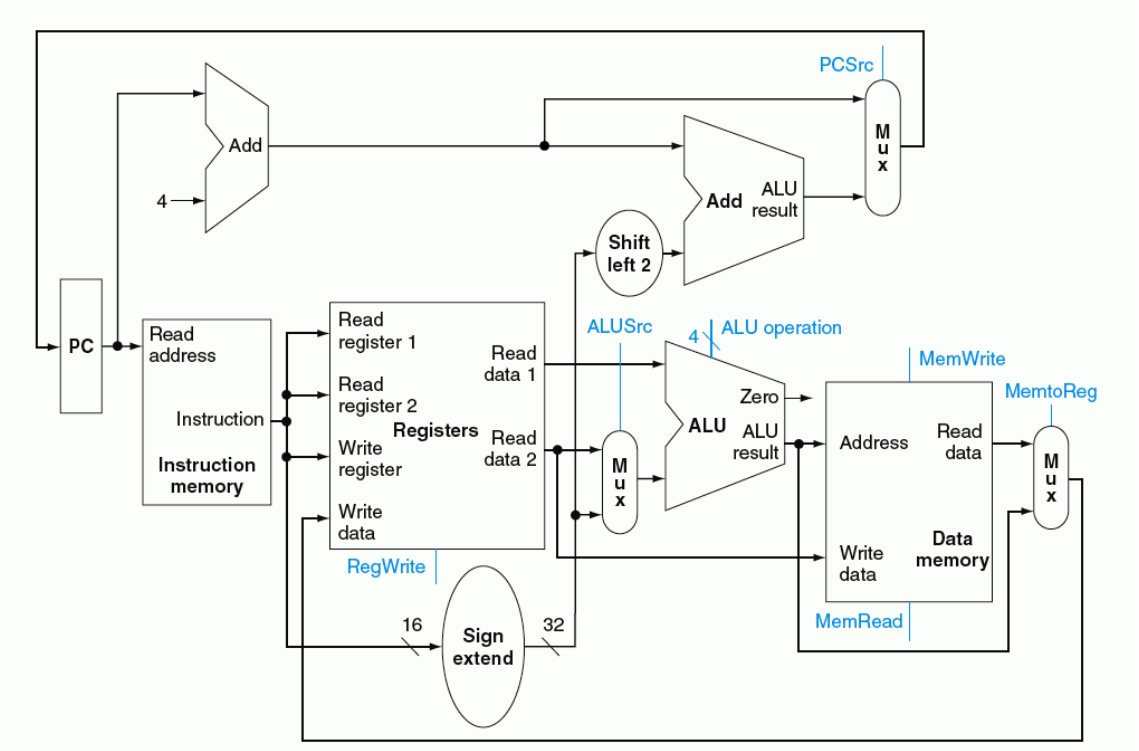

Рисунок 16 Объединенный тракт данных процессора.

3.5 Реализация однотактового процессора.

Рассмотренный ниже процессор соответствует модели тракта данных, описанной в предыдущем разделе. На первом этапе datapath-модель будет детализирована в части организации управления и структуры АЛУ, в части разрядности шин передачи данных и сигналов управления. Далее будет представлена разработка устройства управления. Синтез принципиальной электрической схемы в том или ином виде не входит в задачу данного этапа.

3.5.1 Организация управления АЛУ.

Как было сказано выше, АЛУ используется для нескольких задач: непосредственно для реализации машинных команд, арифметических или логических, и для выполнения вспомогательных операций – для вычисления адреса при обращении к основной памяти (команды *lw, sw*), для вычисления адреса перехода (команды *beq, j*).

Выбор операции реализуемой АЛУ осуществляется подачей определенного кода на управляющий вход ALU operation. Коды операций для типового АЛУ процессоров MIPS приведены в таблице: приведены только рассматриваемые в нашем примере операции, но, учитывая возможные расширения, используется 4-х разрядный код, который позволит кодировать всего до 16 различных операций. В нашем примере требуется поддержка 5-ти операций (add, sub, and, or, set on less than). Кроме того, сразу добавим операцию NOR для ближайших расширений: для функциональной полноты процессора должна быть операция инверсии.

Значение на входе ALU operation	**Реализуемая функция**
0000	AND
0001	OR
0010	ADD
0110	SUB
0111	SLT
1100	NOR

Как показано выше, АЛУ может быть использовано в нескольких вариантах:

1) Для выполнения арифметической или логической операции с операндами (команды add, sub, and, or);
2) Для вычисления физического адреса памяти путем сложения базового адреса и смещения (команды lw, sw);
3) Для сравнения двух значений путем вычитания (beq, slt).

Каждый из этих вариантов использования АЛУ кодируется специальным двухразрядным кодом ALUOp, генерируемым устройством управления.

В первом случае (арифметических и логических команд) АЛУ также может использоваться по разному, в зависимости от выполняемой команды. В данном случае выполняемая операция кодируется кодом Instruction, получаемым непосредственно из кода исполняемой команды.

Из значений ALUOp и Instruction при помощи специального блока ALU Control получаем четырехразрядный код, подаваемый непосредственно на вход ALU Operation управления АЛУ.

Такая двухступенчатая генерация ALU Operation - сначала из кода команды получаем ALUOp и Instruction, а на втором этапе из этих значений получаем ALU Operation – нужна, чтобы обеспечить ускоренную генерацию кода ALU Operation для команд lw, sw, beq: в данном случае фактически будет использоваться комбинационная декодера от двухразрядного кода ALUOp, которая работает быстрее, чем если бы она декодировала сразу 8-ми разрядный код ALUOp + Instruction.

В таблице (см. Рисунок 17) представлено кодирование рассмотренных выше значений и соответствующие операции АЛУ. Далее (см. Рисунок 18) показана таблица истинности для блока ALU Control, на основании которой может быть синтезирована цифровая логическая схема этого блока. Еще раз отметим, что для строчек 1 и 2 таблицы истинности выходные значения не зависят от значений на входах F0-F5, а только от ALUOp0 и ALUOp1, что ускоряет срабатывание схемы ALU Control в этих случаях.

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

Рисунок 17 Описание кодов ALUOp, Instruction (Func Field) и ALU Operation (ALU control input).

ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

Рисунок 18 Таблица истинности блока комбинационного декодера ALU Control

3.5.2 Анализ формата кодов команд процессора.

Устройство управления любого процессора декодирует поля машинной команды, анализирует различные признаки (флаги) состояния и на основе этих данных выполняет генерацию управляющих сигналов для блоков процессора, настраивая их на совместное выполнение заданной команды.

Первым шагом для синтеза схемы блока управления является определение формата машинной команды (состав, назначение, размер и кодирование полей команды). Форматы рассматриваемых нами команд процессора MIPS показаны на рисунке (см. Рисунок 19).

Рисунок 19 Форматы команд

Первое поле Op[5:0] (биты [31:26] кода операции) – 6-ти разрядный код операции. Для регистровых операций Op= 0, значения для остальных операций: lw – $(35)_{10}$; sw - $(43)_{10}$; beq - $(4)_{10}$; j - $(2)_{10}$.

Поле rs – адрес регистра для чтения, где хранится первый операнд для регистровых операций и условного перехода и базовый адрес для операций с памятью.

Поле rt – адрес регистра для чтения, где хранится второй операнд для регистровых операций и условного перехода, адрес регистра-источника для операции sw, адрес регистра назначения (адрес записи!!) данных для lw.

Поле rd – только для регистровых команд - адрес регистра назначения результата.

Поле funct – код регистровой операции (add, sub, and, or, nor и т.д.).

Поле address[15:0] для команд lw, sw – смещение в памяти относительно базового адреса, для команды beq – относительный адрес перехода (смещение от значения PC).

Поле address[25:0] для команды безусловного перехода j – абсолютный адрес перехода[4].

Отметим, что в качестве поля адреса регистра назначения может выступать либо rt (для команды lw) либо rd (для регистровых команд). Поэтому необходимо добавить в тракт данных дополнительный мультиплексор, который выбирает, какое поле (rt или rd) подать на вход адреса для записи в регистровый файл (см. Рисунок 20). Данный мультиплексор управляется сигналом RegDst.

[4] Полный адрес перехода при выполнении команды j рассчитывается как JumpAdress[31:0] = (PC+4)[31:26]·(address[25:0]*4)[25:0] – выполняется сдвиг address[25:0] на два разряда влево и конкатенация со старшими разрядами адреса следующей после выполняемой команды = PC+4.

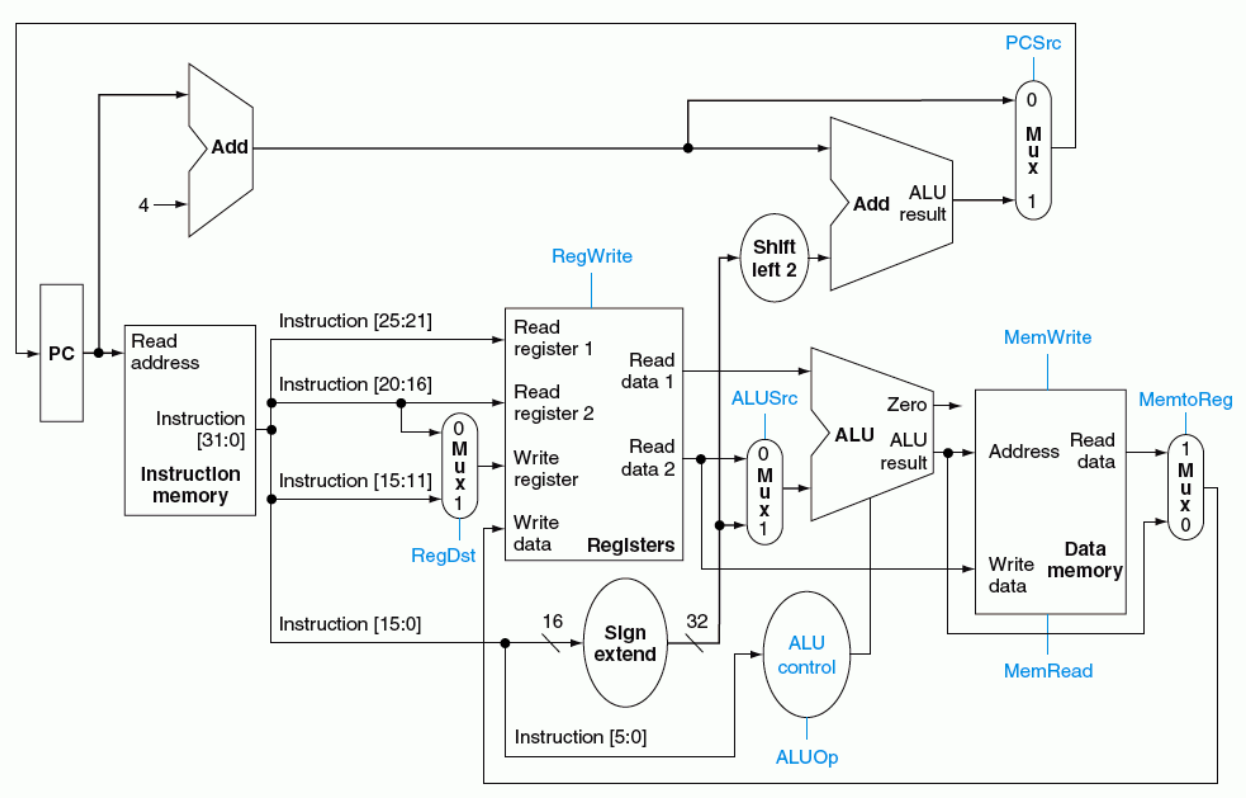

Рисунок 20 Потоковая модель процессора с выделением полей кода команды.

На рисунке (см. Рисунок 20) показана расширенная модель тракта данных (потоковая модель) процессора, содержащая блоки ALU Control и мультиплексор выбора поля адреса сохранения данных в регистровом файле. Также на этой схеме указаны, какие разряды (поля) кода операции подаются на входы блоков процессора.

3.5.3 Функционирование устройства управления.

Следующим шагом в разработке процессора является определение правил генерации управляющих сигналов для всех блоков процессора.

Функции генерации управляющих сигналов выполняет устройство управления, показанное на рисунке (Рисунок 21).

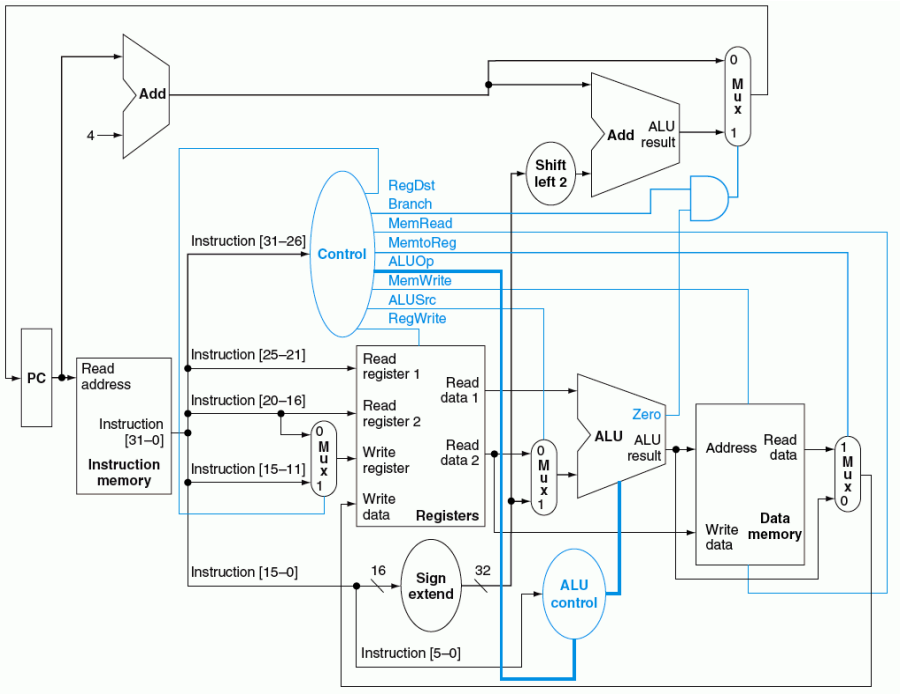

Рисунок 21 Генерация управляющих сигналов процессора

Как говорилось выше, для формирования сигналов управления АЛУ (на вход ALU operation), используется дополнительный – выделенный блок управления АЛУ ALU Control (см. Рисунок 21). На его вход подан код математической/логической функции из поля funct кода машинной команды (биты [5:0] кода операции) и двухбитный код ALUOp[1:0] от устройства управления.

Все управляющие сигналы формируются устройством управления исходя из значения кода операции – поля Op в машинной команде. Это значение – биты [31:26] машинной команды – подаются на вход устройства управления. Исключением является сигнал PCSrc, формируемый на входе мультиплексора для выбора источника адреса перехода при выполнении команды *beq*[5]. Сигнал PCSrc формируется при условии наличия кода операции beq и при установленном признаке Z нулевого результата на выходе АЛУ. Проверка данного условия выполняется при помощи элемента AND (логическое И), добавленного в схему (см. Рисунок 21), на входы которого подаются сигналы Branch от устройства управления и флаг ZERO от АЛУ.

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Рисунок 22 Формирование сигналов управления

В таблице (см. Рисунок 22) показаны состояния сигналов управления в зависимости от выполняемой машинной команды.

Значение управляющего сигнала «1» соответствует выполнению соответствующей операции. Например, RegWrite = 1 означает, что выполняется запись в регистр, а RegWrite=0, что запись в регистр не выполняется, даже если на соответствующих входах регистрового файла установлены какой-либо адрес и данные.

Если сигнал подается на управляющий вход мультиплексора, то существенными являются и значение «0» и значение «1». Например, при RegDest = 0 – адрес регистра берется из поля rt, а при RegDest=1 – из поля rd.

Если значение сигнала указано как «x», то это означает, что оно может быть любым – «0» или «1» и это не повлияет на результат выполнения команды. Например, значение сигнала RegDst не важно при выполнении команд sw и beq, т.к. в данном случае записи в регистровый файл не выполняется - сигнал RegWrite = 0 – и, соответственно, не важно. какое значение выставлено на адресный вход для записи.

Далее на общей потоковой схеме процессора показаны пути выполнения команд различных типов.

3.5.4 Выполнение регистровых команд.

Выполнение регистровых команд можно описать следующей последовательностью действий (реально все эти действия выполняются за один такт процессора).

Например: add $t1, $t2, $t3:

1) Код команды считывается из памяти команд, счетчик команд увеличивается на размер следующей команды PC=PC+4.
2) Значения операндов считываются из регистров $t1 и $t2 регистрового файла. Одновременно устройство управления декодирует код операции op и формирует все управляющие сигналы.
3) АЛУ выполняет операцию сложения, настроенное на нее управляющим сигналом ALU Operation от блока ФДГ Control.
4) Результат сложения записывается в регистровый файл по адресу $t1.

[5] Кроме того, добавится еще один мультиплексор и сигнал управления, выбирающий адрес абсолютного перехода для команды j, но на данном рисунке он пока не показан

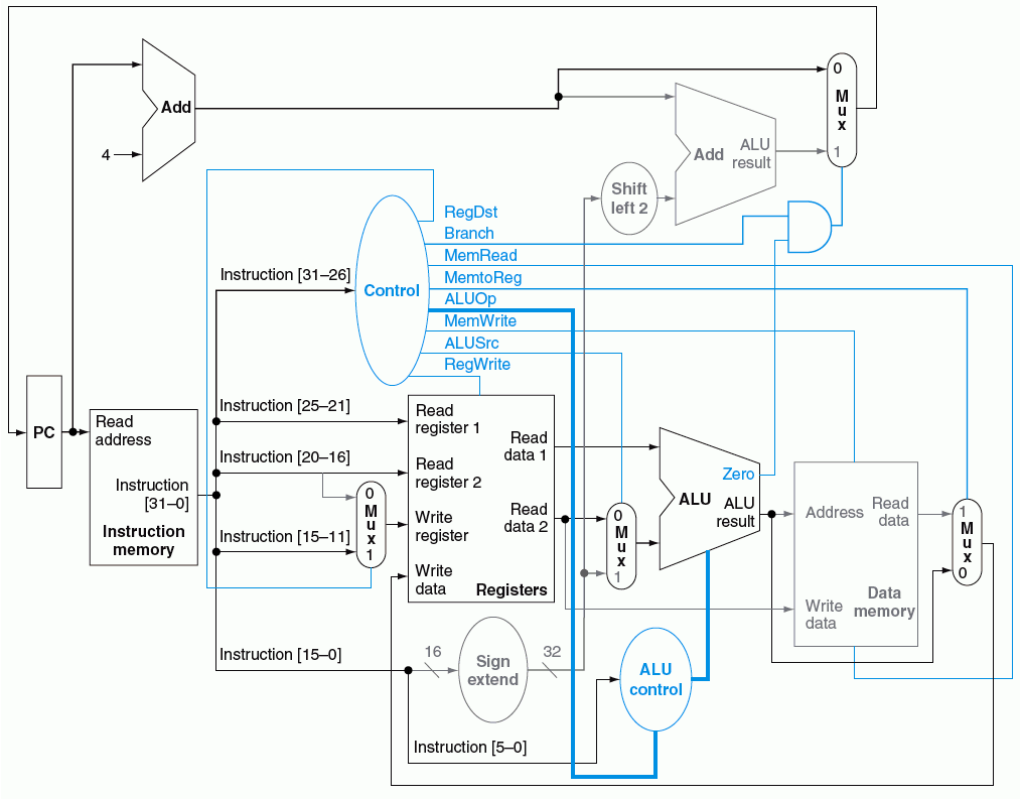

Рисунок 23 Схема выполнения регистровых команд (арифметических и логических)

3.5.5 Выполнение команд работы с основной памятью.

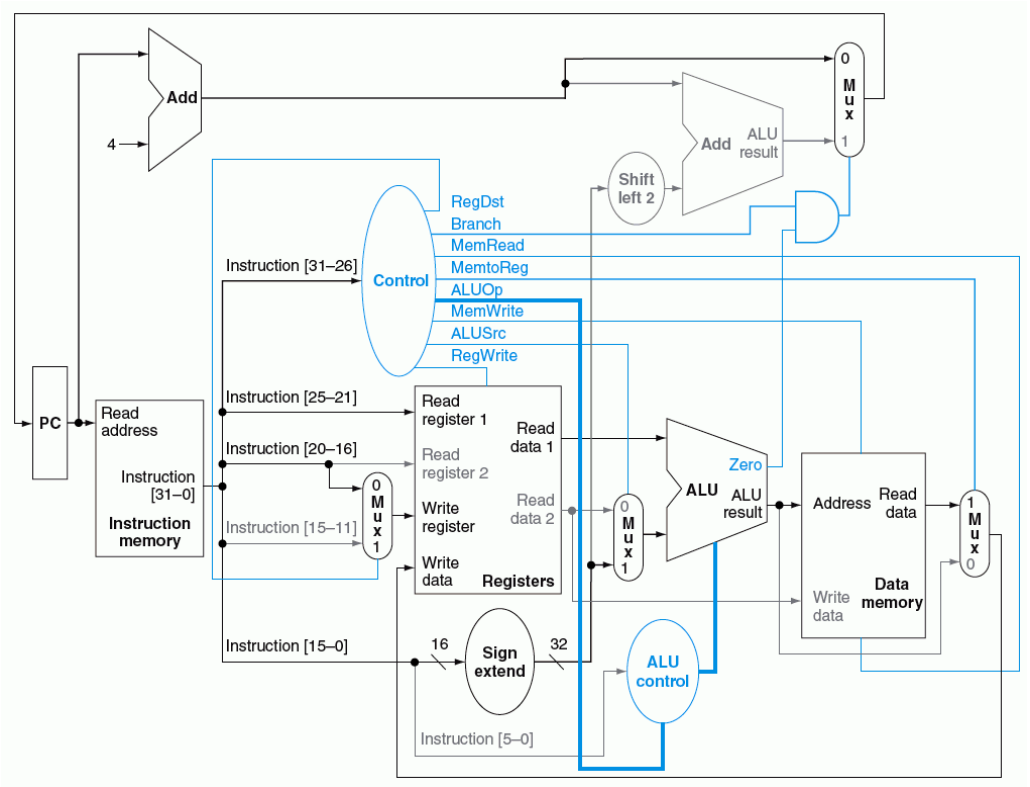

Рисунок 24 Схема выполнения команды lw загрузки из памяти

3.5.6 Выполнение команд условных переходов.

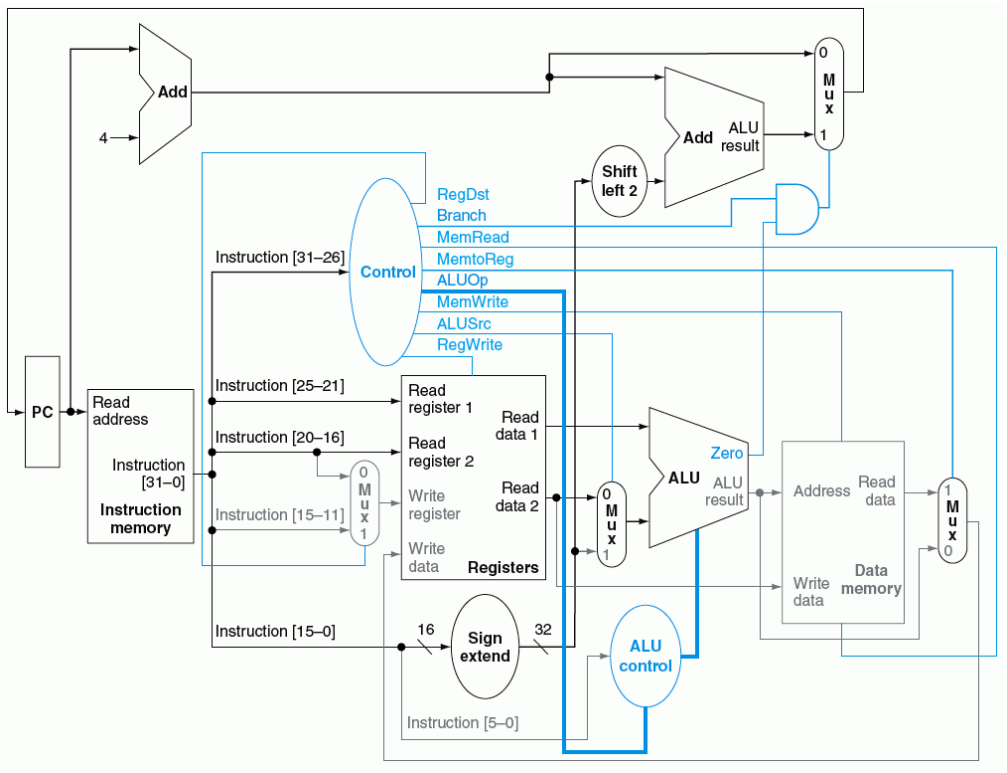

Рисунок 25 Схема выполнения команды BEQ условного перехода

3.5.7 Выполнение команд безусловных переходов.

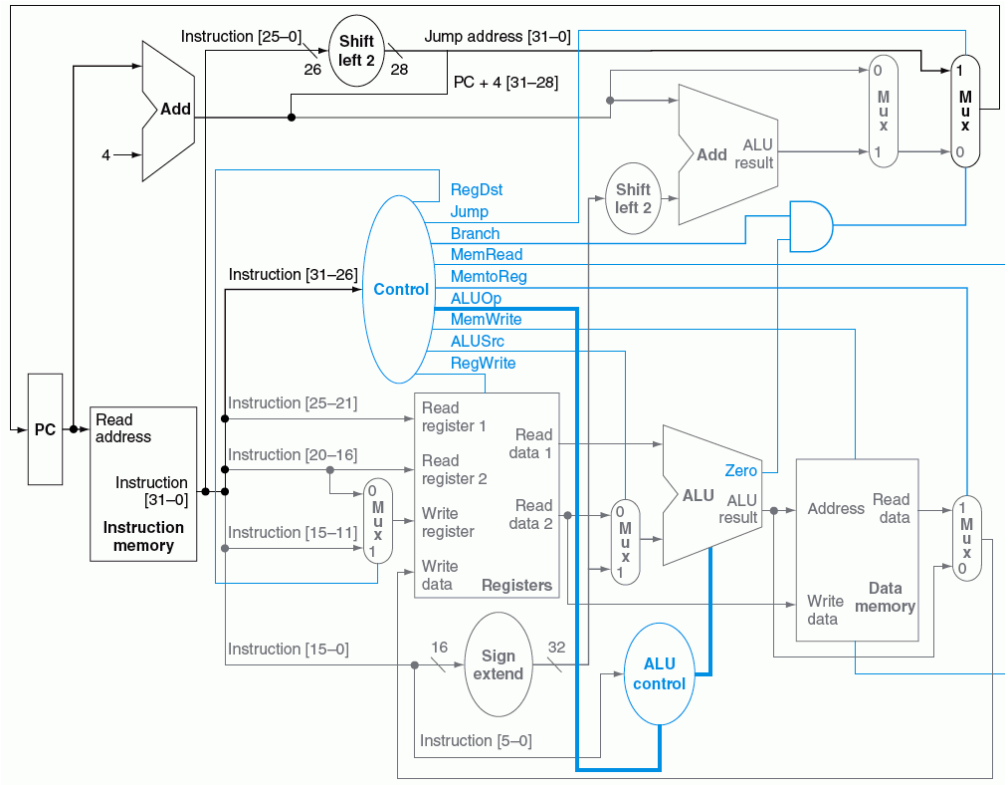

Рисунок 26 Схема выполнения команды J безусловного перехода

Синтез устройства управления однотактного процессора.

Ниже приведена таблица истинности (см.Рисунок 27), на основании которой может быть выполнен синтез устройства управления рассматриваемого однотактного процессора. Столбцы соответствуют исполнению команд различных типов: регистровых (R-format), обмена с внешней памятью (lw и sw) и перехода (beq). На вход устройства управления подается шестиразрядный код операции Op[5:0] из машинной команды, при этом на выходе формируются девять сигналов управления. При исполнении регистровых команд дополнительно к коду операции, декодируемым устройством управления Control, блоком управления ALU Control декодируется поле Instruction[5:0]. Управляющий код с ALU Control подается на вход выбора операции АЛУ.

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Рисунок 27 Таблица истинности для синтеза устройства управления

3.6 Использование однотактовых процессоров.

Однотактовые процессоры характеризуются простой комбинационной организацией - без последовательностных схем управления, однако имеют серьезные недостатки, ограничивающие эффективность их применения на практике:

1) Невозможность использования блоков процессора (АЛУ, регистровой памяти, основной памяти) более чем в одной микрооперации в рамках выполнения команды. Например, нельзя использовать память и для хранения исполняемых команд (как память команд) и для хранения данных (как основную память данных) (как это установлено принципами фон-Неймана). Или, при выполнении команд условных переходов нельзя использовать АЛУ и для сравнения операндов и для вычисления адреса перехода. В результате требуется дублировать некоторые блоки процессора, что значительно усложняет структуру и функционирование процессора, увеличивает схемотехническую и технологическую сложность оборудования - микросхемы процессора, а значит и его стоимость. Причем: чем более сложный набор команды реализуется, тем большее дублирование и более резкий рост структурной сложности и цены наблюдаются.
2) Использование нетиповых блоков многопортовой регистровой памяти, с достаточно сложной организацией (основная сложность – выполняемые одновременно чтение. Для этого требуется, на выбор:
   - применение специальных ячеек памяти с многотактовым доступом, обеспечивающих стабильность считываемых данных на выходе ячейки памяти при одновременной записи в данную ячейку памяти;
   - применение схем арбитража, предотвращающих запись пока не завершена операция чтения.

3) Большая средняя длительность исполнения команды, по сравнению с многотактовыми процессорами, и, соответственно, более низкая производительность.

Например, в таблице представлены задержки в блоках тракта обработки данных. В случае однотактового процессора инкремент счетчика команд PC и соответственно начало исполнения очередной команды должны происходить не чаще, чем через время исполнения максимально длительной команды – для рассмотренного процессора это 600 псек на команду LW. В случае остальных – менее длительных – команд, процессор будет простаивать остаток времени от окончания исполнения команды до окончания периода 600 псек. Реализовать переменную длительность периода выполнения команд является сложной задачей, существенно усложняющей организацию и стабильность работы процессора (этим занимается направление асинхронной цифровой схемотехники).

Вид команды	**Длительность обработки, псек.**					
	Выборка команды	**Чтение из блока регистров**	**Выполнение операции АЛУ**	**Чтение или запись в основную память**	**Запись в блок регистров**	**Итого на команду**
Регистровая	200	50	100		50	400
LW	200	50	100	200	50	600
SW	200	50	100	200		550
BEQ	200	50	100			350
J	200					200

Если же выделить процессорные такты более высокой частоты, с периодом равным длительности обработки в наиболее «быстром» блоке, то обработку в остальных блоках можно выполнять за несколько тактов, а время выполнения каждой команды процессора будет близко или равно реально требуемому на исполнение команды времени. Соответственно среднее время на исполнение команды будет ниже, чем для однотактового процессора (если только не все команды в программе будут «максимально длительными», т.е. LW). К примеру: в программе с 25% команд LW, 10% команд SW, 45% регистровых команд, 15% условных переходов и 5% безусловных переходов:

- однотактный процессор будет иметь среднее время исполнения команды T ком=600 псек;
- многотактный процессор будет иметь среднее время исполнения T ком=600*25%+550*10%+400*45%+350*15%+200*5% = 447.5псек.

То есть многотактный процессор будет иметь 1.34 раза большую производительность.

3.7 Организация мультитактового процессора.

3.7.1 Принципы построения мультитактового процессора.

Проблемы, описанные в предыдущем параграфе могут быть решены путем многотактовой (мультитактовой) организации процессора. Основными принципами проектирования таких процессоров являются:

1) В рамках тракта обработки данных должны быть выделены отдельные этапы – микрооперации, которые будут выполняться за один такт работы цифровой схемы процессора. Такт процессора характеризуется выполнением последовательности считывания данных-обработки данных-записи результатов. К данным в этом случае отнесены машинные команды, обрабатываемые этими командами прикладные данные, признаки выполнения операций обработки прикладных данных, специальные служебные данные.
2) В простом случае микрооперации должны иметь одинаковую (близкую) длительность исполнения, чтобы в рамках каждого такта выполнялась одна

микрооперация с минимальным простоем после завершения ее исполнения до начала исполнения следующей микрооперации. В более сложных мультитактовых процессорах микрооперации могут иметь ощутимо различающуюся длительность, но кратную длительности такта. При этом каждая из микроопераций выполняется различное число тактов и, соответственно, простой в рамках каждого такта будет минимальным. Данная схема позволяет достичь большей средней производительности для процессоров с широким разнообразием видов команд (CISC-процессоров), но ведет к росту сложности организации самого процессора и к сложности планирования исполнения программы во времени (последнее очень важно для систем реального времени. включая системы сигнальной обработки).

3) Набор микроопераций должен быть унифицирован для всех машинных команд: из этого набора должно быть возможно «составить» любую команду процессора. При этом нужно стремиться к сокращению числа микроопераций (это дает структурную и функциональную простоту устройства управления процессора), упрощению действий в рамках каждой микрооперации (это дает структурную и функциональную простоту операционного устройства процессора), к минимизации длительности исполнения микрооперации (дает скорость выполнения), к использованию одинаковой длины однотактовых микроопераций (достигаются скорость, простота устройства управления, возможность конвейеризации микроопераций).
4) Использование одних и тех же блоков несколько раз в рамках исполнения одной машинной команды. Это дает упрощение структуры процессора. Например, вместо АЛУ и двух дополнительных сумматоров в однотактовом процессоре, в мультитактовом используется только одно АЛУ; в качестве памяти команд и основной памяти данных используется один и тот же блок памяти.
5) Для возможности многократного использования блоков в рамках одной команды требуется установка дополнительных регистров хранения промежуточных данных, чтобы данные полученные от блока в рамках предыдущей мирооперации, не изменялись для следующих за ним блоков из за его использования в следующей микрооперации.
6) Так как некоторые промежуточные регистры обновляются каждый такт, а некоторые регистры и блоки регистров/памяти обновляются только на определенных этапах (тактах) исполнения команды, то для последних должны быть предусмотрены сигналы стробирования (подачи команды) записи, генерируемые устройством управления.
7) Так как многократно используемые блоки «объединяют» ранее «раздельные» микрооперации, то на входах и выходах таких блоков (например. АЛУ, памяти) должны быть предусмотрены дополнительные коммутирующие элементы (мультиплексоры/демультиплексоры), подающие/снимащие данные на блок по различным путям в зависимости от выполняемой микрооперации.
8) Устройство управления должно формировать последовательности выполняемых микроопераций для команды каждого типа и должно подсчитывать количество тактов для команды каждого типа, если команды имеют различное число тактов. Для реализации таких функций устройство управления должно быть построено как последовательностная схема (цифровой автомат), а не как комбинационная схема для однотактового процессора. При этом устройство управления может быть цифровым автоматом с жестко зафиксированной «схемной» логикой или микропрограммируемым.

3.7.2 Организация мультитактового процессора.

На схеме (см. Рисунок 28) представлена обобщенная схема мультитактового процессора, показывающая субтракты для исполнения всех ранее перечисленных микроопераций: выборки команды, чтения из блока регистров, обработки данных АЛУ,

чтения-записи в основную память, записи в блок регистров. Каждый такой субтракт включает входные и выходные буферные регистры (регистры специального назначения PC, IR, MDR, A, B, ALUOut) и блок обработки (АЛУ, память, блок регистров). Данная схема соответствует приведенным выше принципам организации мультитактовых процессоров:

- один и тот же блок памяти используется для хранения команд и данных.
- АЛУ используется как для обработки операндов, а так же для расчета новых значений счетчика команд и для расчета полного адреса операндов в памяти.
- между субтрактами используются буферные регистры.

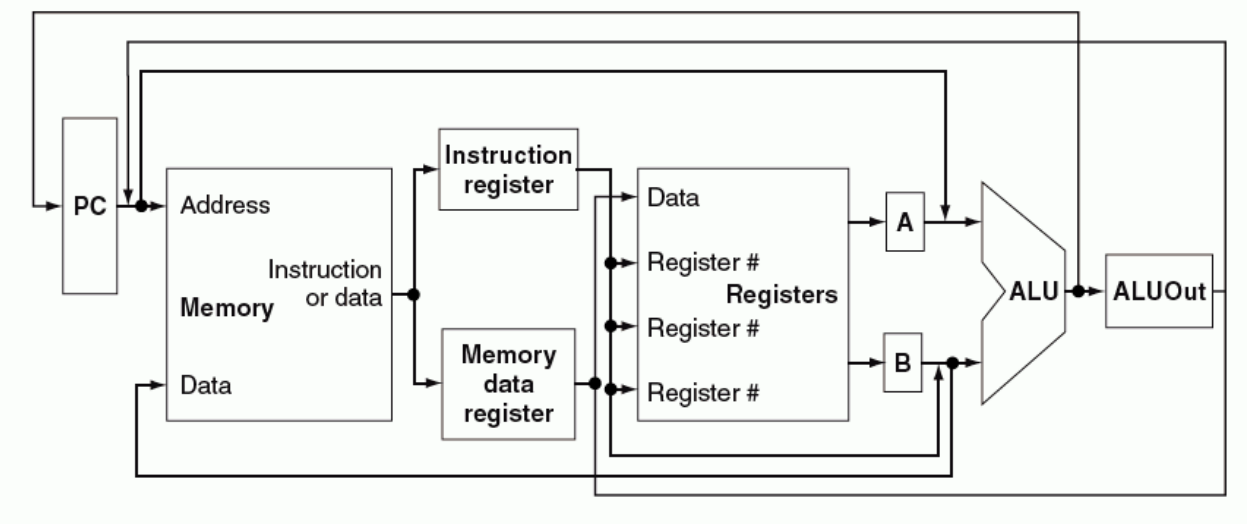

Рисунок 28 Обобщенная схема тракта данных мультитактового процессора

Далее (см.Рисунок 29) представлена эта же, но более детализированная схема, показывающая основные функциональные блоки, новые коммутационные элементы (мультиплексоры), обеспечивающие многократное использование блоков для выполнения нескольких микроопераций в рамках одной команды, а также данные (поля команды, адреса, операнды и результаты) передаваемые между ними. Также здесь показаны основные сигналы управления.

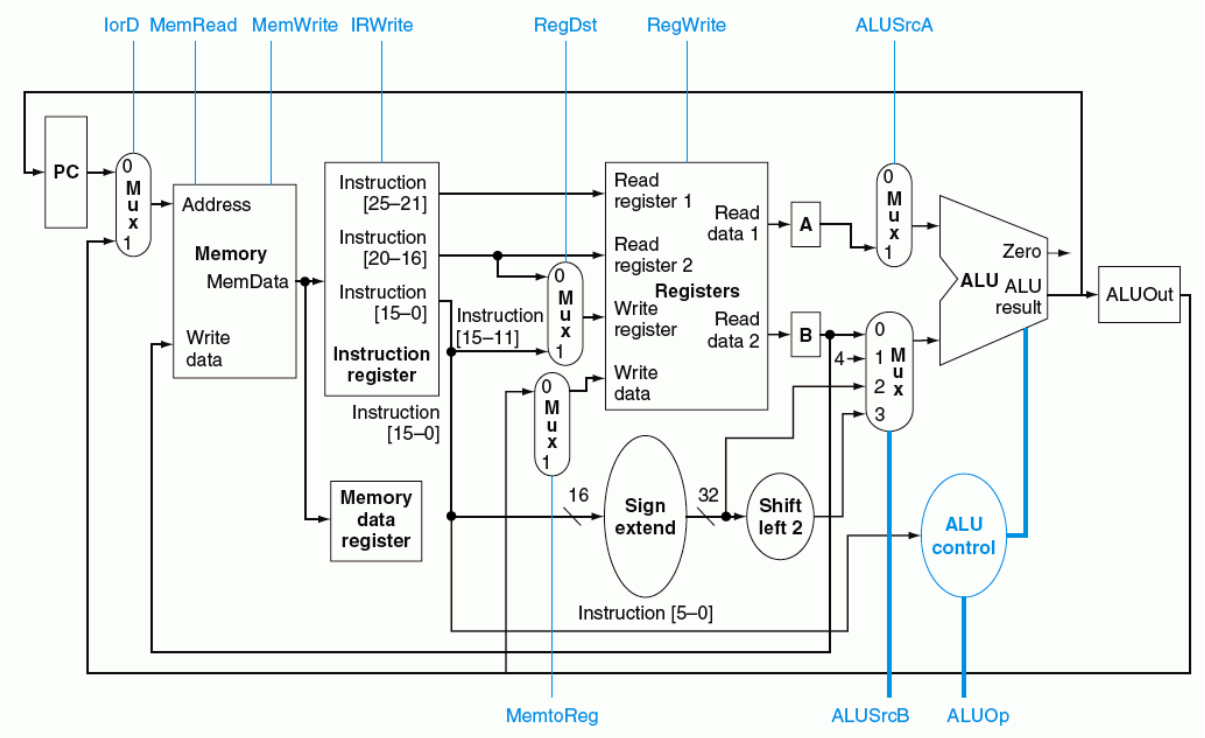

Рисунок 29 Детализированная схема тракта данных мультитактового процессора с сигналами управления

На следующем рисунке (Рисунок 30) представлена полная схема мультитактового процессора со всеми дополнительными блоками (Shift, SignExtend), а также с расширенным набором сигналов управления и с устройством управления, генерирующим эти сигналы.

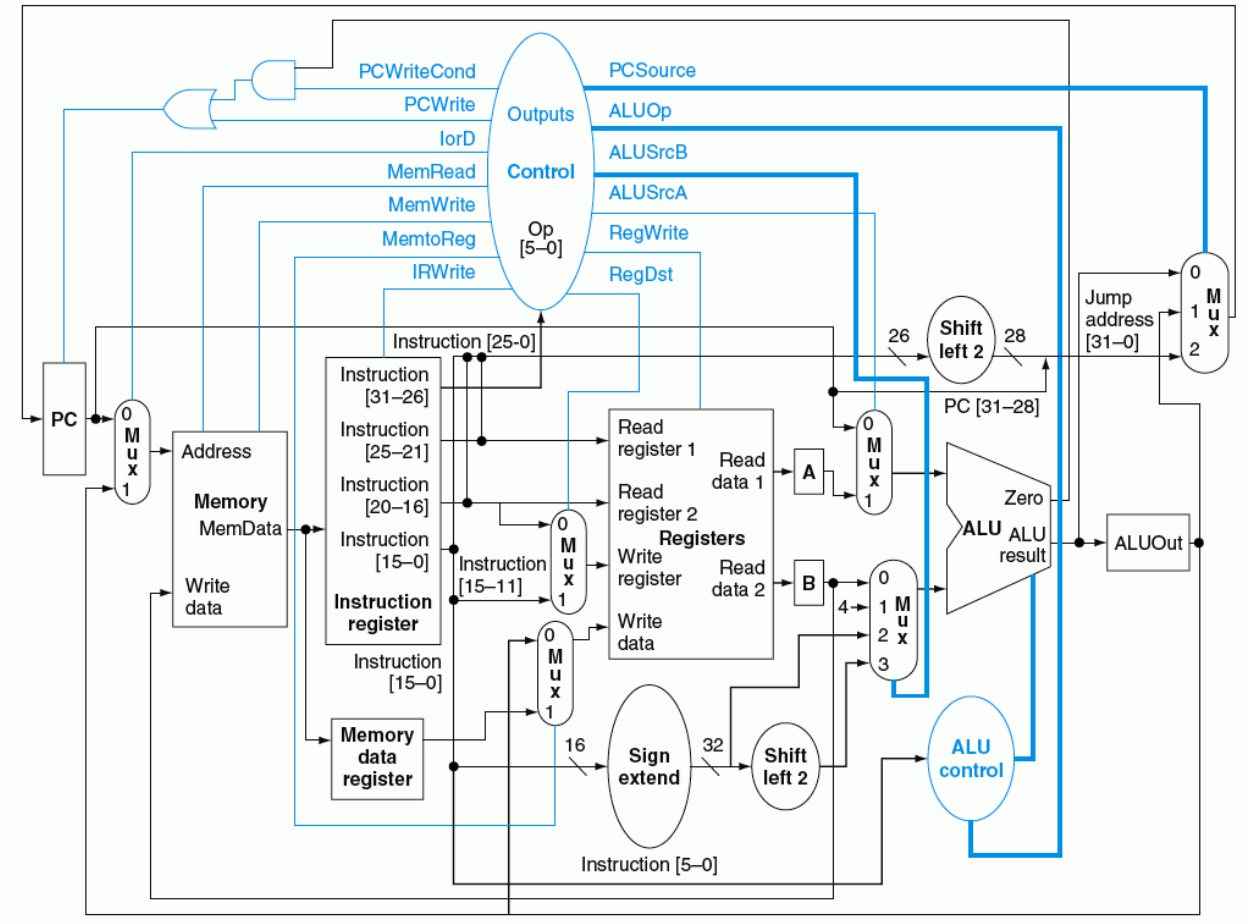

Рисунок 30 Детализированная схема тракта данных мультитактового процессора с устройством управления

Рассмотрим новые блоки, добавленные на этой схеме по сравнению со схемой однотактного процессора (см.Рисунок 21).

1) Мультиплексор на входе адреса блока памяти (Memory) позволяет подавать на адресный вход адрес команды (IorD=0 на этапе выборки команды) или адрес данных (IorD=1 на этапе считывания/записи операндов в память).
2) Регистр команд (Instruction register) хранит исполняемую команду на протяжении всего цикла исполнения команды. В отличие от других межблочных буферных регистров на этой схеме (Memory data, A, B, ALUOut), регистр команд не перезаписывается на каждом такте исполнения команды, поэтому запись в него не может тактироваться периодическими импульсами сигнала синхронизации, а ему требуется специальный сигнал управления записью IRWrite, вырабатываемый только на этапе выборки команды.
3) Регистр данных памяти (Memory Data Register) используется для промежуточного хранения данных считанных из памяти перед их записью в регистровый файл (при исполнении машинной команды LW). При записи в память (машинная команда SW) роль такого буфера выполняет регистр второго операнда АЛУ B.
4) Буферные регистры операндов A и B сохраняют операнды, считанные из регистрового файла, для их дальнейшей обработки на последующих тактах исполнения команды.
5) Мультиплексор на верхнем входе АЛУ позволяет выбрать для обработки первый «регистровый» операнд из буфера A (ALUSrcA=1) или счетчик команд для вычисления адреса следующего операнда (ALUSrcA=0).

6) Мультиплексор на нижнем входе АЛУ стал четырехвходовым и позволяет выбрать:
   - второй «регистровый» операнд из буферного регистра B (ALUSrcB=00);
   - константу 4 для суммирования ее со счетчиком команд (PC=PC+4) при расчете адреса следующей команды при последовательном (без переходов) исполнении команд (ALUSrcB=01);
   - значение младших 16-ти бит команды, расширенное до 32-х разрядного числа (с помощью блока SignExtend), которое используется в качестве смещения при вычисления полного адреса основной памяти (ALUSrcB=10);
   - значение младших 16-ти бит команды, расширенное до 32-х разрядного числа (с помощью блока SignExtend) и умноженное на 4 (сдвинутое на 2 бита влево). Это значение используется в качестве смещения при вычисления адреса перехода при выполнении команд условного перехода, например, BEQ (ALUSrcB=11).
7) Буферный регистр ALUOut, который используется для промежуточного хранения результата ALU перед его сохранением (в памяти, в регистровом файле или в счетчике команд).
8) Мультиплексор источников адреса следующей исполняемой команды, которая будет записана в счетчик команд PC. Это может быть значение (PC+4), вычисленное АЛУ (PCSource = 00), значение адреса условного перехода, сохраненное в буферном регистре ALUOut на время, пока вычисляется условие перехода с помощью того же АЛУ (PCSource = 01) и, наконец, это может быть адрес безусловного перехода, полученный путем конкатенации (соединения) старших 4-х битов счетчика команд и поля адреса перехода из кода команды, умноженного на 4 (сдвинутого на 2), чтобы указать смещение в командах, а не в байтах.

Ниже (см. Рисунок 31) приведено детальное описание сигналов управления мультитактовым процессором.

Actions of the 1-bit control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2-bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

Рисунок 31 Описание функций сигналов управления мультитактового процессора

3.7.3 Пошаговое исполнение команд в мультитактовом процессоре.

В таблице (см. Рисунок 32) показаны шаги исполнения команд всех классов и выполняемые на каждом шагу микрооперации.

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode/register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend (IR[15:0]) << 2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + sign-extend (IR[15:0])	if (A == B) PC <= ALUOut	PC <= {PC [31:28], (IR[25:0]),2'b00)}
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B		
Memory read completion		Load: Reg[IR[20:16]] <= MDR		

Рисунок 32 Пошаговое исполнение команд в мультитактовом процессоре.

3.7.4 Организация устройства управления.

По сравнению с устройством управления однотактового процессора, которое является *комбинационным*, описываемым таблицей истинности (см. Рисунок 27), устройство управления мультитактового процессора является *последовательностным*, т.е. должно описывать последовательность шагов, на каждом из которых устанавливаются определенные управляющие сигналы. Такие шаги называют *состояниями*, каждое состояние соответствует определенному этапу (такту) выполнения команды. Переход в то или иное состояние (т.е. установка определенного набора значений выходных сигналов) будет зависеть от значений на входах (для процессора входами является поле Op[5:0] из кода команды) и от предыдущего состояния (для процессора – от предыдущих завершенных тактов исполнения команды, которые будут различными для различных команд).

Последовательность этапов (тактов) исполнения команды процессором и, соответственно, состояний устройства управления при исполнении каждой команды показана на схеме ниже (см. Рисунок 33). Как видно, эта последовательность является циклической: после исполнения команды (второй шаг) переходим к выборке новой команды (первый шаг) и так по кругу.

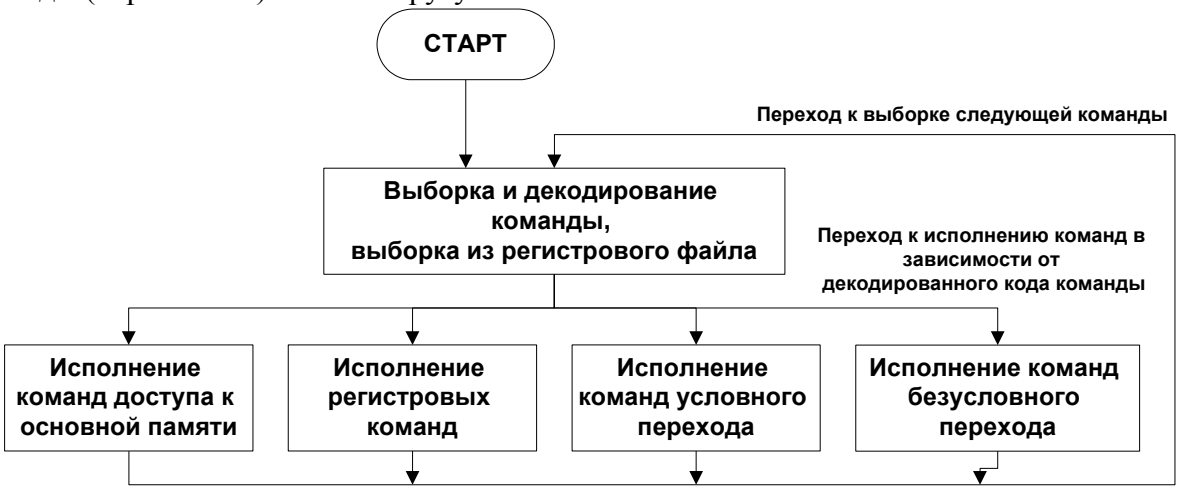

Рисунок 33 Функционирование устройства управления мультитактового процессора.

3.7.4.1 Конечные автоматы.

Для описания последовательностных схем, в том числе устройств управления процессора и других блоков ЭВМ (например, блоков расчета тригонометрических и иных сложных функций, контроллеров последовательных интерфейсов, контроллеров прямого доступа к памяти и т.п.), широко используется модель цифровых конечных автоматов (Finite State Machine, FSM). Конечный автомат – способ описания поведения

последовательностных систем как процесса перехода этих блоков из состояния в состояние в зависимости от входных сигналов управления и предыдущих состояний, в которых находилась система.

Описание в форме конечного автомата может иметь графический вид (граф конечного автомата), табличный вид (таблицы переходов и таблицы выходов), текстовый вид (программы, скрипты и т.п.). Чаще используется графический вид, как более наглядный: состояния обозначаются окружностями (иногда овалами или прямоугольниками), внутри каждого перечисляются значения, которые присваиваются выходным сигналам при попадании в это состояние, рядом (или внутри круга) указавается номер (или название) состояния. Переходы между состояниями указываются направленными дугами (стрелками), около которых указывается логическое условие, по которому выполняется этот переход из состояния в состояние; условием является определенное значение входных сигналов. Если переход из состояния в состояние безусловный, то условие у дуги не указывается. Чаще всего используются синхронные конечные автоматы: переходы из состояния в состояние выполняются только через определенные промежутки времени – по тактам задаваемым периодических сигналом синхронизации (на схемах автоматов сигнал синхронизации не показывается). Соответственно, безусловные переходы, даже если они идут друг за другом, будут выполняться не сразу все, а последовательно, по тактовым сигналам.

Конечные автоматы могут использоваться для описания как поведения (функционирования) аппаратуры, так и программы. Существуют формальные методы и автоматизированные средства синтеза аппаратных блоков и программного обеспечения из описаний в виде конечных автоматов.

Устройство управления процессора может реализовывать свои функции, описанные в форме конечного автомата, одним из двух способов:

1) Микропрограммным способом, когда устройство управления представляет собой самостоятельный простейший процессор, работающий по микропрограмме и выполняющий микрокоманды, каждая из которых формирует управляющие сигналы в соответствии с текущим этапом исполнения машинных команд.
2) В виде схемы с жесткой логикой. В данном случае определенные, жестко фиксированные функции конечного автомата заданы структурой цифровой схемы. Такая схема будет иметь вид, представленный на рисунке. Она включает блок комбинационной логики, ответственный за формирование выходных сигналов и номера следующего состояния в зависимости от входных и текущего состояния, а также блок регистров состояния, хранящий номер очередного состояния.

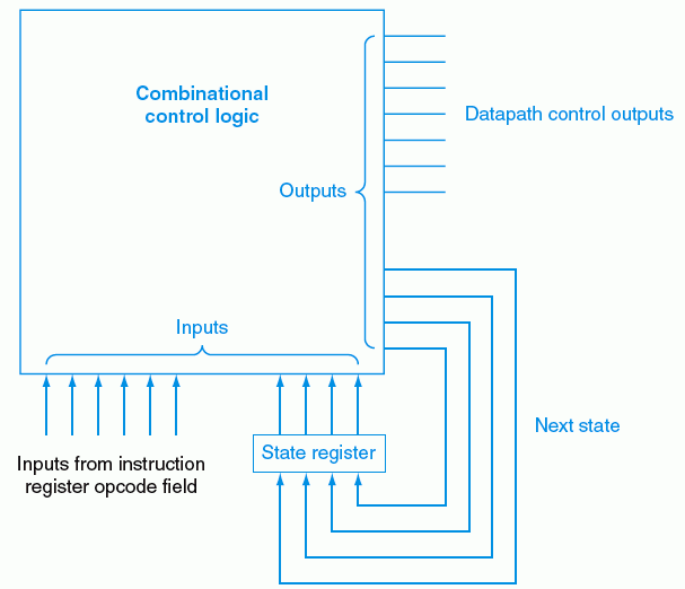

Рисунок 34 Структура схемы конечного автомата

3.7.5 Функционирование устройства управления мультитактового процессора.

Далее на графической диаграмме конечного автомата, построенной на основе диаграммы Рисунок 33, описывается поведение устройства управления на каждом этапе исполнения машинных команд. Как отмечалось в предыдущем параграфе, существуют формальные способы синтеза логических схем и микропрограмм из описаний поведения блоков в виде конечных автоматов, поэтому на базе предложенной обобщенной диаграммы функционирования устройства управления можно однозначно синтезировать аппаратную реализацию устройства управления (здесь не рассматривается).

3.7.5.1 Выборка и декодирование команд.

Состояния №0 и №1 одинаковы для всех команд.

В состоянии №0 выполняется чтение команды из памяти в регистр команд (IorD = 0, MemRead =1, IrWrite = 1); выполняется расчет следующего «нормального» значения счетчика команд PC=PC+4 (ALUSrcA=0, ALUSrcB=01, ALUOp = 00, PCSource = 00, PCWrite=1).

В состоянии №1 считываются операнды из регистрового файла и записываются в буферные регистры A и B. Адреса для чтения из регистрового файла установлены в состоянии №0, данные считываются и записываются в A и B каждый процессорный такт (т.е. по тактовому сигналу), поэтому специальных управляющих сигналов для этого формировать не нужно. Параллельно с этим рассчитывается и сохраняется в ALUOut адрес условного перехода (ALUSrcA=0, ALUSrcB=11, ALUOp = 00), который может быть использован или не использован далее.

Еще по выходе из состояния №0 поле Op[5:0] из регистра команд IR подается на входы устройства управления. Это значение показывает, какая команда должна исполняться, и будет использовано для определения перехода из состояния №3 в следующее состояние.

3.7.5.2 Доступ к основной памяти.

Если Op[5:0] = 0x23 (код LW) или = 0x2B (код SW), то выполняется переход к ветке работы с памятью – из состояния №1 переходим в состояние №2. В состоянии №2 выполняется расчет полного адреса памяти – базовый адрес из регистра A складывается со смещением из кода команды (ALUSrcA= 1, ALUSrcB= 10, ALUOp = 00) и результат записывается в ALUOut. Далее, в зависимости от кода операции в поле Op[5:0] выполняется чтение (состояние №3) или запись (состояние №5) в память. И в том и в другом случае на вход памяти подается ранее рассчитанный адрес из регистра ALUOut (IorD=1) и формируется соответствующий сигнал управления операции с памятью (MemRead или MemWrite).

При чтении добавляется еще одна операция – запись данных, считанных из памяти в регистровый файл (состояние №4, MemtoReg = 1, RegDst = 0, RegWrite = 1).

3.7.5.3 Исполнение «регистровых» команд.

Если Op[5:0] = 0x00 (код регистровых команд), то выполняется переход к ветке исполнения регистровых команд – из состояния №1 переходим в состояние №6. В состоянии №6 мультиплексоры на входе АЛУ подключают к АЛУ операнды из регистров A и B, ранее считанные из регистрового файла (ALUSrcA=1, ALUSrcB=00). Блок ALU Control перенастраивается на считывание кода операции из поля Instruction[5:0] из регистра команды IR (ALUOp = 10). В соответствии с этим генерируется код на вход управления АЛУ, а результат появляется на выходе АЛУ и записывается в регистр ALUOut. Далее, в состоянии №7, на входы регистрового файла подается результат операции из буфера ALUOut (MemtoReg = 0), адрес из поля ds команды (RegDst=1) и выполняется операция записи данных в регистровый файл (RegWrite = 1).

3.7.5.4 Условный переход.

Если Op[5:0] = 0x04 (код команды условного перехода BEQ), то выполняется переход к ветке исполнения команд условного перехода – из состояния №1 переходим в состояние

№8. Здесь на входы АЛУ подаем два операнда из регистрового файла, операция АЛУ – вычитание (ALUOp=01), чтобы сравнить два операнда, сигнал PCWriteCond=1 обеспечивает запись в счетчик команд значения ALUOut (PCSource=01), если установлен флаг ZERO, т.е. операнды равны. Напомним, что в ALUOut на этапе выборки и декодирования команды сохранен адрес перехода.

3.7.5.5 Безусловный переход.

Если Op[5:0] = 0x02 (код команды безусловного перехода J), то выполняется переход к ветке исполнения команды безусловного перехода – из состояния №1 переходим в состояние №9. Исполнение безусловного перехода наиболее простое: генерируется сигнал PCWrite, который подается непосредственно на вход разрешения записи регистра счетчика команд (PC). На вход записываемых данных счетчика команд подается адрес перехода равный (PC[31:28], (Instruction[25:0]*4)).

3.7.5.6 Обобщенная схема функционирования устройства управления.

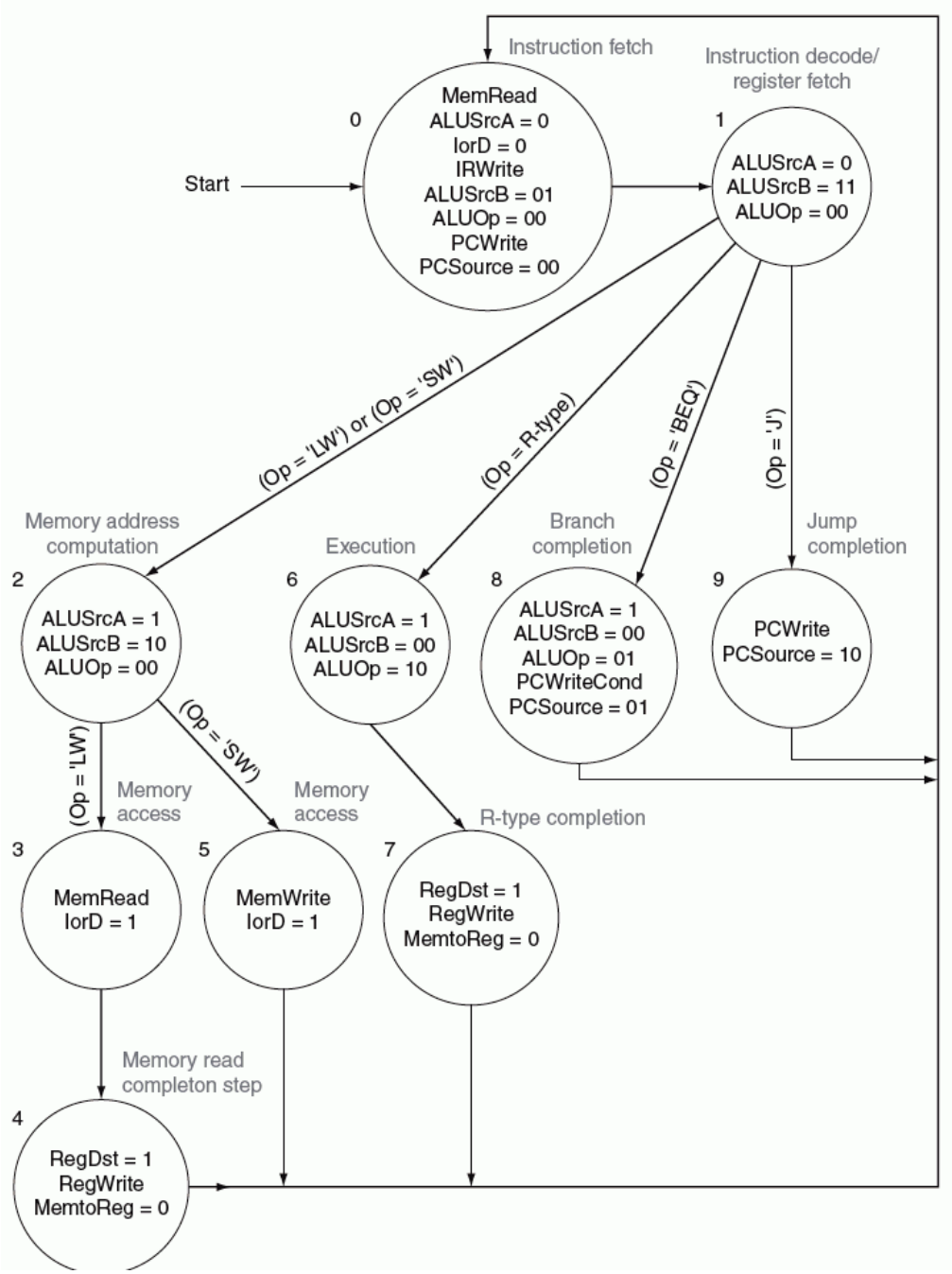

Рисунок 35 Алгоритм работы устройства управления мультитактового процессора

3.7.6 Организация обработки исключительных ситуаций в работе процессора: исключений, прерываний.

Исключения (exception) и прерывания (interrupt) – это события в работе процессора, связанные с изменением порядка выполнения машинных команд. В этом смысле исключения и прерывания можно рассматривать как исполнение перехода, но вызванного не соответствующей машинной командой, а аппаратурой процессора.

Причиной исключений (exception) являются внутренние события процессора, например, переполнение при выполнении арифметических операций или попытка доступа к защищенной ячейке памяти.

Причиной прерываний (interrupt) являются внешние для процессора события, например, запрос на обмен данными от контроллера ввода-вывода или предупреждение от температурного датчика о перегреве микросхемы процессора. Запросы прерываний передаются ядру процессора в форме перепадов цифровых сигналов на специальных входах процессора.

Часто понятия исключения и прерывания объединяются: для процессоров Intel все подобные события называют прерываниями (внешними и внутренними). Для процессоров MIPS – наоборот, все события называют исключениями.

3.7.6.1 Порядок обработки исключений/прерываний.

Для обработки исключений/прерываний требуется выполнить следующие действия:

1) Обнаружить (детектировать) исключение/прерывание. Способ детектирования зависит от типа исключения. Например, для прерываний от внешних источников достаточно обнаружить активный уровень сигнала на входе запроса прерывания. А для обнаружения переполнения нужно контролировать состояние флага переполнения от АЛУ. В целом, обнаружение исключения выполняется специальными аппаратными схемами, которые формируют соответствующие логические сигналы-признаки: ИСТИНА – есть ситуация исключения, ЛОЖЬ – нет. Эти «подготовленные» сигналы-признаки подаются на входы устройства управления процессора, а уже оно, в свою очередь, вызывает подпрограмму обработки исключений/прерываний. В некоторых случаях, связанных с работой самого устройства управления, сигналы-признаки исключений генерируются внутри устройства управления. Например, обнаружение неизвестного кода операции.
2) Сохранить адрес команды, при исполнении которой обнаружилось исключение/прерывание. Эта информация потребуется для анализа причины возникновения исключения/прерывания и/или для возврата к исполнению программы после обработки исключения/прерывания. Для сохранения может использоваться специальный регистр *EPC (Exception Program Counter)*. Иногда, например, при обработке прерывания, когда адрес исключения не несет информации о причине его возникновения, адрес исключения сохраняют в стеке просто как адрес возврата из подпрограммы (и для этого же – для возврата - используют).
3) Сохранить признаки, указывающие на причину возникновения (тип) исключения/прерывания. Эта информация нужна для анализа и обработки исключения/прерывания. Обычно используется специальный регистр признаков: это или выделенный регистр признаков возникновения исключения *Cause*, или общесистемный регистр статуса (признаков) *Status*.
4) Выполнить переход к специальной подпрограмме обработки исключения/прерывания, хранящейся по определенному известному адресу. Фактически это означает, что добавляется еще один вариант источника для записи счетчика команд PC (в добавок к инкрементированному адресу PC+4, адресу

условного перехода и к адресу безусловного перехода). Существует два варианта переходов при исключениях:

- по одному адресу для всех типов прерываний (для MIPS это адрес 0x8000 0180). Далее пути обработки определяются в зависимости от признаков, установленных в статусном регистре *Cause*.
- векторный – когда для каждого типа исключения/прерывания выполняется переход по специальному, отличному от других типов исключений адресу (вектору). При этом для каждого типа прерывания используется своя процедура обработки. Векторная обработка более производительная (не нужно тратить время на анализ признаков исключения), но более сложно реализуемая.

5) Если причина исключения/прерывания не привела к фатальной (неисправимой) ошибке, то после завершения обработки исключения/прерывания нужно вернуться к исполнению основной программы. Обычно это выполняется вызовом специальной машинной команды возврата из обработчика исключений/прерываний.

3.7.6.2 Потоковая модель процессора с элементами обработки исключений.

Далее (в качестве простого примера) рассмотрим вариант процессора, обеспечивающий обработку исключений двух типов – обнаружение неизвестной команды и возникновение арифметического переполнения. На схеме (см. Рисунок 36) показана структура мультитактового процессора с элементами обработки этих исключений: регистром EPC для хранения адреса исключения, регистром признаков Cause и мультиплексором на его входе, обеспечивающим выбор признака для одного из двух возможных исключений, с расширенным до 4-х входов мультиплексором выбора нового значения для счетчика команд PC (сигнал управления PCSource), который теперь позволяет записать в PC адрес обработчика исключений/прерываний равный 0x8000 0180.

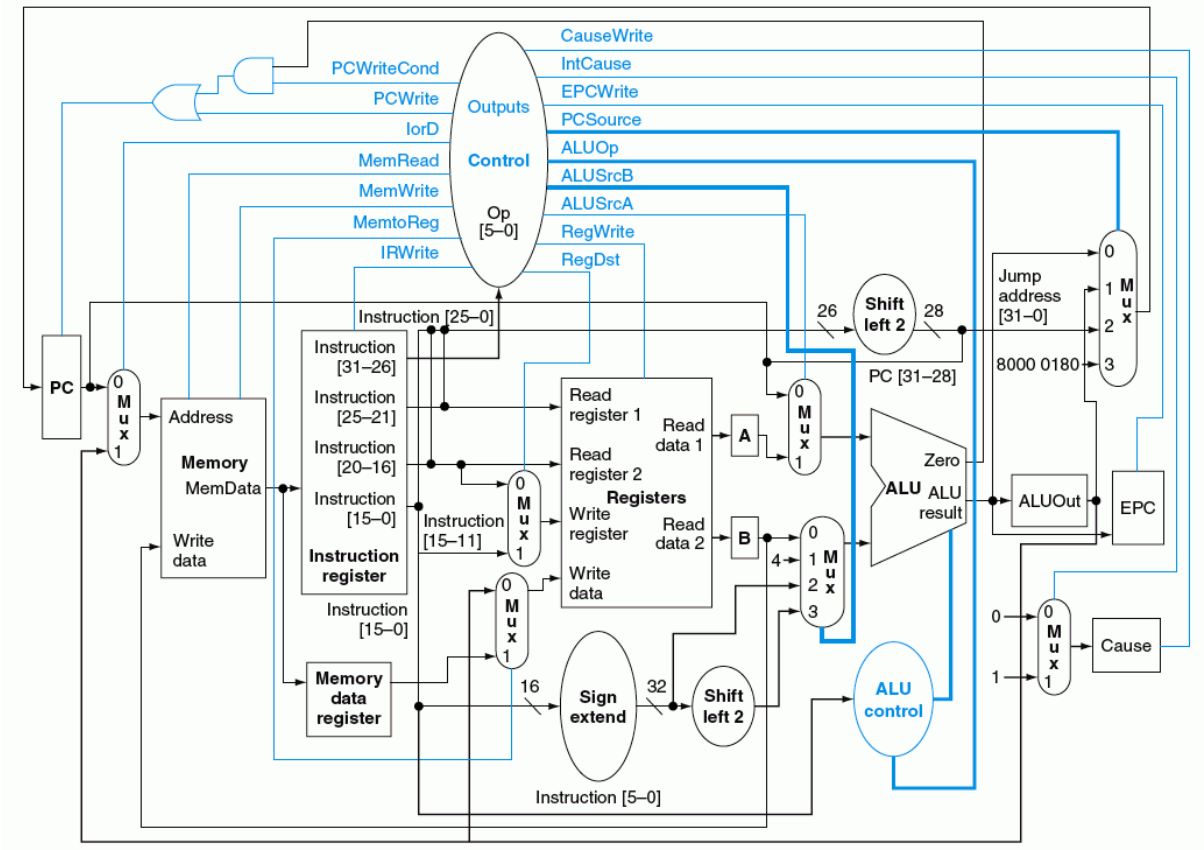

Рисунок 36 Модель процессора с обработкой исключений

3.7.6.3 Функционирование устройства управления.

Ниже показан автоматный алгоритм работы устройства управления, дополненный элементами обработки исключений. Конкретно: добавлены состояния 10 (обработка неизвестной команды) и 11 (обработка арифметического переполнения). Оба эти состояния идентичны, за исключением признака причины исключения IntCause, записываемого в регистр состояния Cause: для исключения по неизвестной команде IntCause = 0, для исключения по переполнению IntCause = 1.

Примечательной особенностью данного алгоритма (и процессора в целом) является формирование адреса исключения, записываемого в регистр EPC: т.к. обнаружение исключений выполняется на этапе декодирования команд или их исполнения, когда в регистре PC уже сохранен адрес следующей команды (PC+4), то значение для EPC вычисляется путем «обратного» вычитания числа 4 из уже инкрементированного счетчика команд PC. Для этого значение PC подается на вход «A» АЛУ (ALUSrcA=1), число 4 подается на вход B АЛУ (ALUSrcB=00), для АЛУ выбирается операция вычитания (ALUOp=01) и результат записывается в EPC (EPCWrite =1).

Данная модель не предусматривает специальных механизмов и команд для возврата из обработки исключений. Но они могут быть легко добавлены в виде команды, осуществляющей переход по адресу EPC+4, т.е. на следующую команду, после команды, на которой возникло исключение.

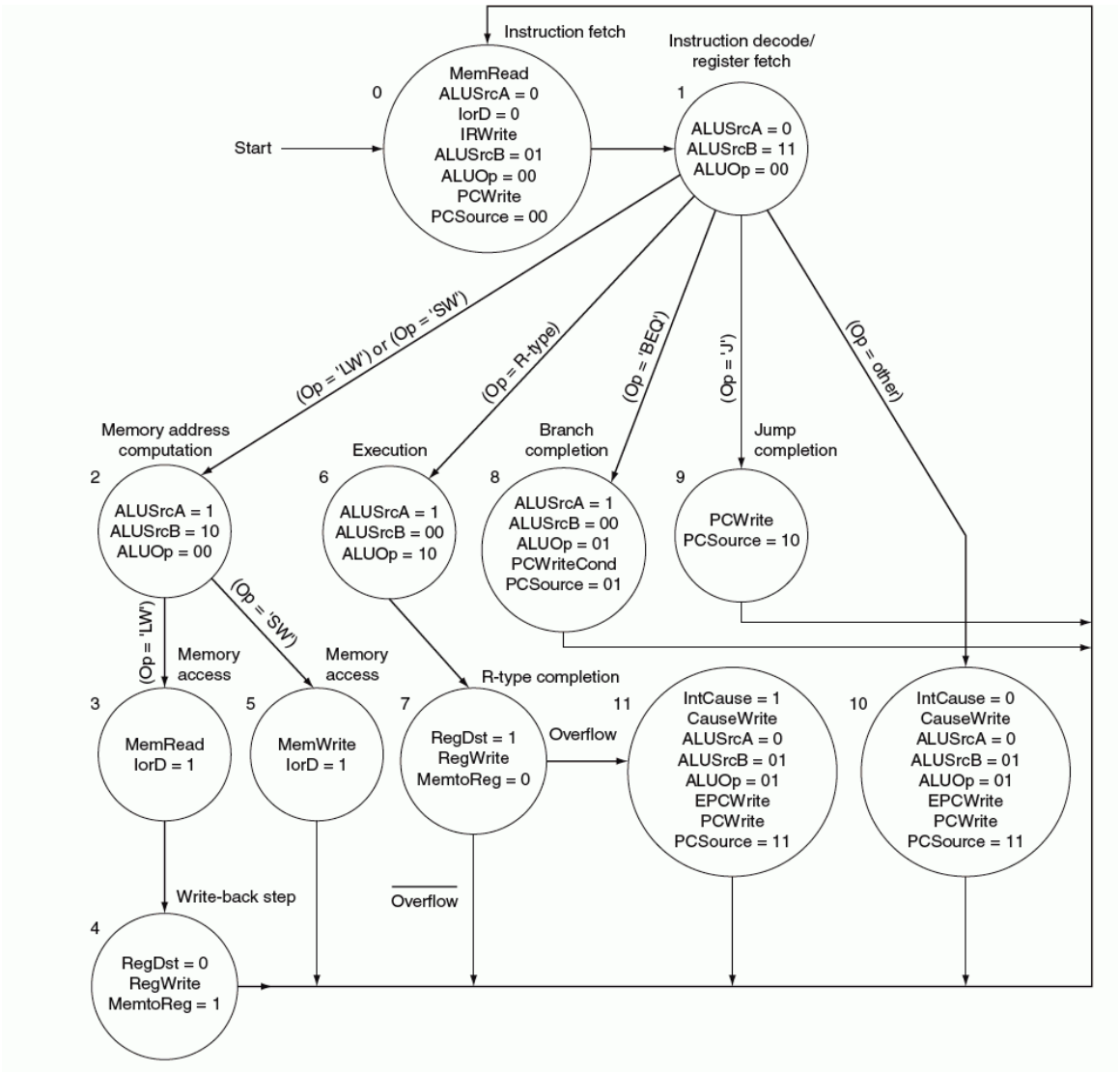

Рисунок 37 Алгоритм работы устройства управления с обработкой исключений

3.8 Организация конвейеров.

3.8.1 Принципы конвейеризации вычислений.

Следующим способом повышения производительности процессора (после введения мультитактовой схемы) является конвейеризация вычислений. Конвейеризация – это техника параллельного исполнения операций, идущих в последовательном потоке, базирующаяся на выделении у этих операций нескольких стадий исполнения и организации одновременного (параллельного) исполнения этих стадий.

Механизмы конвейеризации на сегодня широко применяются фактически во всех вычислительных систем (кроме простейших, используемых в частных случаях) и, считаются одним из основных способов повышения производительности вычислительных блоков различного типа, в.т.числе памяти, процессоров. Конвейеризация используется при исполнении машинных команд, при пакетной передаче данных, для ускорения доступа к блокам памяти (например, в памяти SDRAM/DDR) и во многих иных вариантах.

Принцип конвейеризации вычислений показан на рисунке (Рисунок 38).

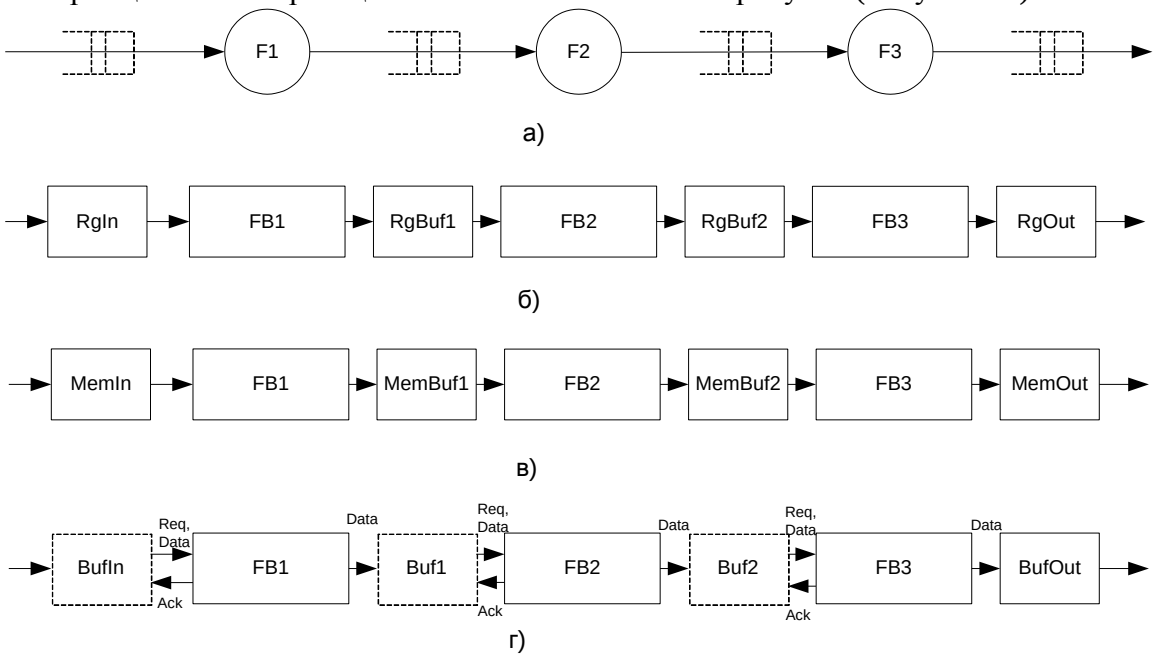

Рисунок 38 Конвейеризация вычислений

Алгоритм (процесс) исполнения операции разделяется на несколько последовательных стадий (Рисунок 38, а))

Каждая стадия выполняется независимым функциональным блоком: FB1, FB2, FB3 на Рисунок 38 б), в), г).

Чтобы в течение работы каждого функционального блока в цепочке данные на его входе оставались постоянными, учитывая что предыдущий блок уже выполняет уже соответствующую ему стадию следующей операции, то (подобно мультитактовому процессору), между блоками используются буферные регистры RgBuf (Рисунок 38, б)).

Исполнение своих стадий операции у всех функциональных блоков должны начинаться и завершаться одновременно: строго к окончанию очередного такта и к записи очередных результатов работы функционального блока в буферный регистр, предыдущие результаты были бы уже не нужны следующему блоку в цепочке.

Чтобы уйти от требования жесткой синхронности работы всех блоков, не заставляя одни блоки постоянно ждать другие, вместо буферных регистров используется буферная память типа FIFO. По готовности результатов предыдущего шага они записываются в эту память FIFO, а извлекаются оттуда следующим блоком в цепочке по его готовности (Рисунок 38, в)).

Все перечисленные варианты относятся к *синхронным конвейерам*, когда все ступени, включая функциональные блоки и буферные элементы, начинают и завершают

выполнение своей функции строго одновременно – по общему синхронизирующему сигналу. Так легче избежать коллизий по готовности-неготовности между блоками в цепочке, но неизбежно возникают потери времени, когда свободный блок ожидает очередного синхроимпульса, чтобы начать выполнение очередного действия.

Если хочется уйти от этого, то можно строить *асинхронные* конвейеры (Рисунок 38, г)), когда очередному функциональному блоку от предыдущего посылается запрос на готовность принять данные, а он, как только готов, это подтверждает, после чего данные продвигаются по цепочке. Между ступенями асинхронного конвейера могут добавляться буфера, но это не обязательный элемент. Асинхронные конвейеры более сложны в организации, но потенциально более производительны, т.к. не имеют задержек на прием новых данных.

Конвейер не всегда представляет собой линейную цепочку стадий. В соответствии с логикой исполнения операций отдельные стадии в конвейере могут пропускаться, а некоторые - повторяться. Такие конвейеры называются *нелинейными* (см. Рисунок 39: нелинейный конвейер позволяет вычислять два типа функций: X и Y). Нелинейные конвейеры позволяют выполнять большее разнообразие операций, однако имеют значительно более сложную организацию и более подвержены конфликтам при разделении одних и тех же функциональных блоков (или связанных с ними данных) при конвейерном исполнении операций.

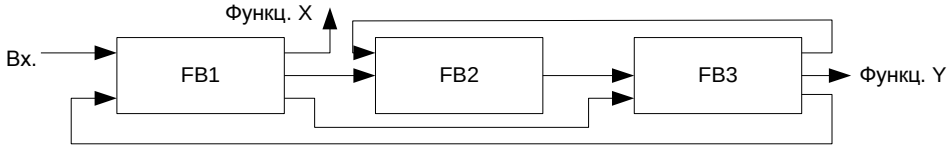

Рисунок 39 Нелинейный конвейер

3.8.2 Конвейер команд.

Наиболее часто в вычислительной технике конвейеризация применяется для организации исполнения процессорных команд. Каждая команда рассматривается как отдельная операция, у нее выделяются стадии, и организуется параллельное исполнение этих стадий. У рассмотренного в предыдущих разделах простого процессора (с архитектурой MIPS), можно выделить следующие стадии исполнения машинных команд (см. Рисунок 32):

1) Выборка команды (Instruction Fetch, IF);
2) Декодирование команды и чтение из регистрового файла (Instruction Decode, ID);
3) Исполнение или вычисления адреса памяти данных (Execution, EX);
4) Доступ к памяти данных (Data Memory Access, MEM);
5) Запись результата (Write Back, WB).

У других типов процессоров данный набор может отличаться, но не значительно – сохраняется общий принцип. Например, если поддержана работа с операндами из памяти данных, то стадии вычисления адреса памяти данных и доступа к памяти данных располагаются перед стадией исполнения.

3.8.3 Производительность конвейера.

Выше говорилось, что производительность однотактового процессора небольшая, так как на выполнение любой команды процессора выделяется время, равное времени, требующемуся для исполнения максимально длительной команды из всей системы команд.

В мультитактовом процессоре ситуация улучшилась: каждая команда исполняется ровно столько времени, сколько нужно на реализацию всей последовательности ее стадий, и не больше. Однако исполнение команд по прежнему последовательное.

При конвейерной организации исполнения команд стадии различных команд могут исполняться параллельно, как показано на рисунке (см. Рисунок 40). Различные стадии могут иметь различную длительность: например на рисунке, стадия Reg – 100 ps, а

остальные стадии – 200 ps. Но, так как любые из них могут исполняться параллельно, то период исполнения команды делится на равные части, каждая из которых имеет длительность равную продолжительности максимально долгой стадии, данном примере – 200 ps. В итоге получаем:

Минимально возможная длительность исполнения команды lw (для примера) составляет 800 ps., как у мультитактового процессора. Однако, при конвейерном исполнении она выполняется за 1000 ps., т.е. медленнее. С другой стороны, три команды исполняются за 1400 ps. против 2400 ps. у мультитактового процессора.

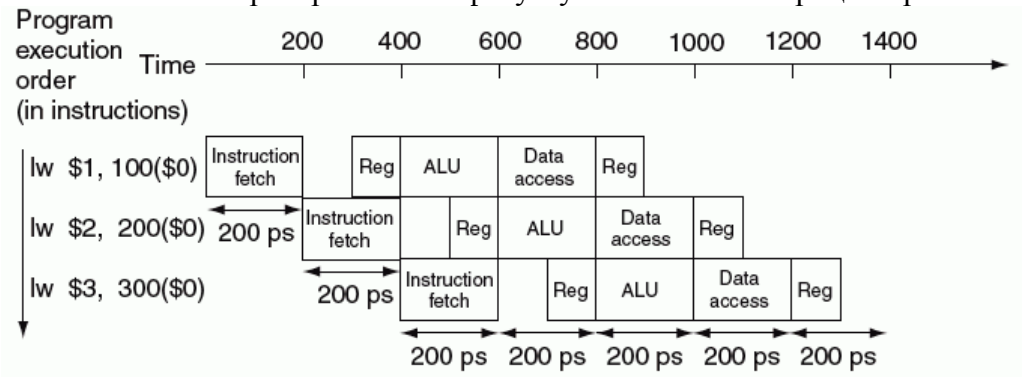

Рисунок 40 Конвейерное исполнение команд.

Таким образом, конвейер замедляет исполнение отдельно взятой команды, но ускоряет исполнение потока последовательных команд, т.е. увеличивает пропускную способность процессора.

В идеальном варианте конвейера, когда реальная требуемая длительность всех стадий одинакова и команды выбираются в естественном порядке - последовательно, рост производительности конвейерного процессора по сравнению с мультитактовым будет стремиться к значению, равному числу ступеней конвейера: выполнение каждой очередной команды будет завершаться через время, равное длительности одной стадии – ступени конвейера. Так как длительность стадии равна длительности команды деленной на количество стадий, то именно на количество стадий – ступеней конвейера и уменьшится период завершения команд в потоке. При этом дополнительные задержки на начальном этапе – при заполнении конвейера – будут иметь тем меньший вес, чем большее число команд будет выполнено.

3.8.4 Конфликты (риски) в конвейере.

Рост производительности за счет использования конвейера команд, описанный в предыдущем разделе, фактически не достижим на практике из-за возникновения конфликтов в конвейере, нарушающих жесткую временную периодичность стадий исполнения команд. Такие конфликты принято называть рисками конвейера (pipeline hazard). Выделяют три типа рисков:

1) Структурные риски – попытка нескольких команд, находящихся в различной стадии исполнения, обратиться к одному и тому же ресурсу вычислительной машины.
2) Риски по данным – обусловленные взаимосвязью конвейеризуемых команд по данным.
3) Риски по управлению – обусловлены неоднозначностью выбора следующей исполняемой команды, например, при выполнении команд условного перехода.

3.8.5 Структурные риски (Structural Hazards).

Такие риски в конвейере команд возникают, когда несколько команд, находящихся в различной стадии исполнения, пытаются использовать один и тот же ресурс вычислительной машины, чаще всего память (основную, регистровую), но возможно и операционные устройства (АЛУ, сумматоры, сдвигатели и т.п.) Структурные риски возникают в двух случаях:

- Если какой-нибудь функциональный блок не полностью «конвейеризован», т.е. не может выполнять свою операцию строго за один такт конвейера. Тогда последующие стадии конвейера будут «набегать» на предыдущие в части использования данного блока.

- Если на различных стадиях конвейера команды используют один и тот-же функциональный блок. Примером такого структурного конфликта может послужить использование одного и того же однопортового блока памяти и в качестве памяти инструкций и в качестве памяти данных. Тогда произойдет структурный конфликт впереди идущей команды на стадии MEM и команды, у которой в это время исполняется стадия IF.

Структурные конфликты можно решать приостановкой конвейера для ожидания освобождения занятого функционального блока. Такие стадии простоя в конвейере называют «пузырьками в конвейере» (pipeline bubble). Возникновение «пузырьков» ведет к замедлению работы конвейера. Поэтому, если возможно, предпочтительнее использовать несколько функциональных блоков, например, выделение отдельной памяти программ и данных или использованием двухпортовой памяти или использование КЭШа данных и команд или использование буфера команд с предвыборкой.

В целом, разрешение структурных рисков наименее трудоемко и в меньшей степени влияет на производительность конвейера команд.

3.8.6 Риски по данным (Data Hazards).

Причина возникновения конфликтов в использовании данных лежит в самом принципе конвейеризации: распараллеливание этапов выполнения команд может привести к изменению порядка этапов записи и чтения данных, установленных порядком следования машинных команд. На рисунке (см. Рисунок 41) показан поток команд (в столбце слева), где результат первой команды DADD R1, R2, R3, сохраненный в R1? используется всеми последующими командами. Справа показано, как организуется конвейерное исполнение этой последовательности команд. Видно, что команды DSUB, AND и OR пытаются использовать данные из R1 еще до того, как они туда записаны командой DADD. Естественно, следствием таких попыток будет некорректный результат всех вычислений. В этом и состоит риск по данным.

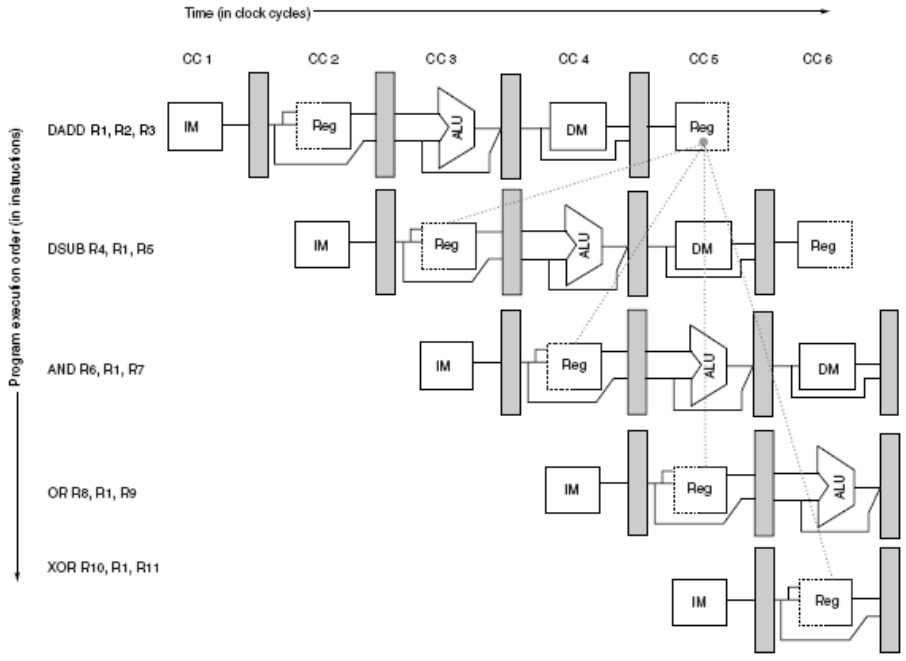

Рисунок 41 Пример конфликтов в использовании данных

В общем случае, если две команды в конвейере (i и j) предусматривают обращение к одной и той же переменной x, причем команда i предшествует команде j, ожидаемы три

типа нарушений порядка обращения к данным, приводящих к конфликтам (рискам) по данным:

1) «Чтение после записи» (ЧПЗ): команда j читает x до того, как команда i успела записать новое значение x, то есть j ошибочно получит старое значение x вместо нового.
2) «Запись после чтения» (ЗПЧ): команда; j записывает новое значение x до того, как команда i успела прочитать x:, то есть команда i ошибочно получит новое значение x вместо старого.
3) «Запись после записи» (ЗПЗ): команда j записывает новое значение x прежде, чем команда i успела записать в качестве x свое значение, то есть x ошибочно содержит i-е значение x вместо j- го.

Есть еще четвертый вариант нарушения порядка обращения к данным, который, однако, не приводит к конфликтам (рискам):

4) «Чтение после чтения» (ЧПЧ): команда j считывает новое значение x прежде, чем команда i успела считать значение x. Хотя порядок обращения к данным и нарушен, но данные считаны корректно и конфликта не произошло.

Из перечисленных типов рисков по данным наиболее вероятно возникновение ЧПЗ, так как именно данный порядок операций характерен для большинства конвейеров (сначала чтение операндов, а потом запись результатов). Иные типы конфликтов возможны в системах, где командам, приостановленным из-за конфликтов, разрешено догонять «ушедший вперед» поток команд, или в случаях, если реализован механизм изменения порядка выполнения команд

3.8.7 Разрешение конфликтов по данным.

Имеется два основных подхода к устранению конфликтов по данным:

1) Программный: возможность конфликта устраняется на этапе компиляции программ – компиляторы не допускают возникновения последовательностей конфликтующих команд или, если этого не избежать, вставляют между конфликтующими командами операции NOP, чтобы конвейер смог продвинуться на требуемое число шагов.
2) Аппаратный:
   а) Вводятся задержки конвейера – «пузырьки» - которые приостанавливают «последующие» стадии обращения к данным до окончания «предыдущих». Как говорилось в предыдущих параграфах, «пузырьки» в конвейере приводят к снижению его производительности, поэтому требуется «догонять» поток команд, что в свою очередь может привести к новым конфликтам.
   б) Используется механизм *ускоренного продвижения данных* (*forwarding* или bypassing или short-circuiting)

3.8.7.1 Ускоренное продвижение данных (forwarding).

Между двумя соседними функциональными блоками в конвейере располагается буферный регистр, через который предшествующая ступень передает результат своей работы на последующую ступень, то есть передача информации возможна лишь между соседними ступенями конвейера. При ускоренном продвижении, когда для выполнения команды требуется операнд, уже вычисленный предыдущей командой, но еще не сохраненный в памяти, этот операнд может быть получен непосредственно из промежуточного буферного регистра (например, на выходе АЛУ), минуя все промежуточные ступени конвейера. С данной целью в конвейере предусматриваются дополнительные тракты пересылки информации между промежуточными буферными регистрами (тракты опережения, тракты обхода), снабженные средствами мультиплексирования.

Например, на рисунке (см.Рисунок 42) показаны такие связи, позволяющие получить данные для команд DSUB и AND, прежде, чем они будут записаны в регистр R1.

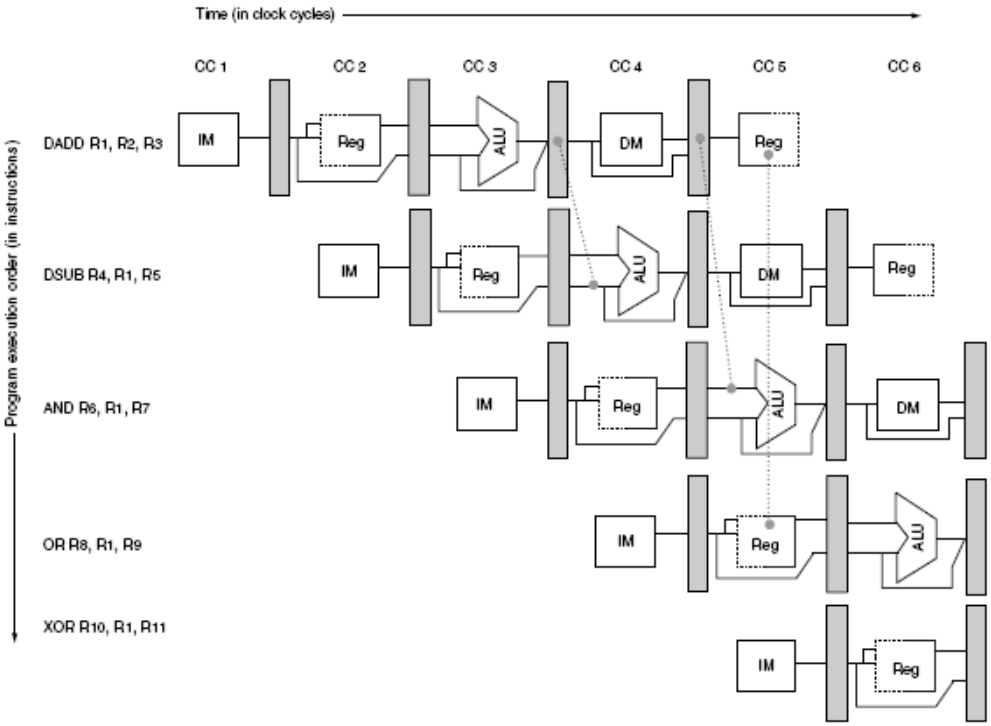

Рисунок 42 Схема ускоренного продвижения данных в конвейере

Здесь для команды DSUB данные берутся из буферного регистра, хранящего данные на выходе АЛУ, для команды AND – из буфера на выходе памяти данных, а при выполнении команды OR данные одновременно записываются и в регистр R1 и в буферный регистр на входе АЛУ.

Далее (см. Рисунок 43) показан еще один вариант ускоренного продвижения данных. Особенностью данного случая является передача данных с выхода функционального блока на вход этого же функционального блока: с выхода на вход АЛУ (команды DADD-LD), с выхода на вход памяти данных (команды LD-SD). При этом удается предотвратить конфликт ЧПЗ: данные в память сохраняемые командой SD берутся не из регистра R4 на этапе СС4 (на этом этапе они еще не обновлены командой LD), а непосредственно из буфера на выходе памяти данных между этапами СС5 и СС6.

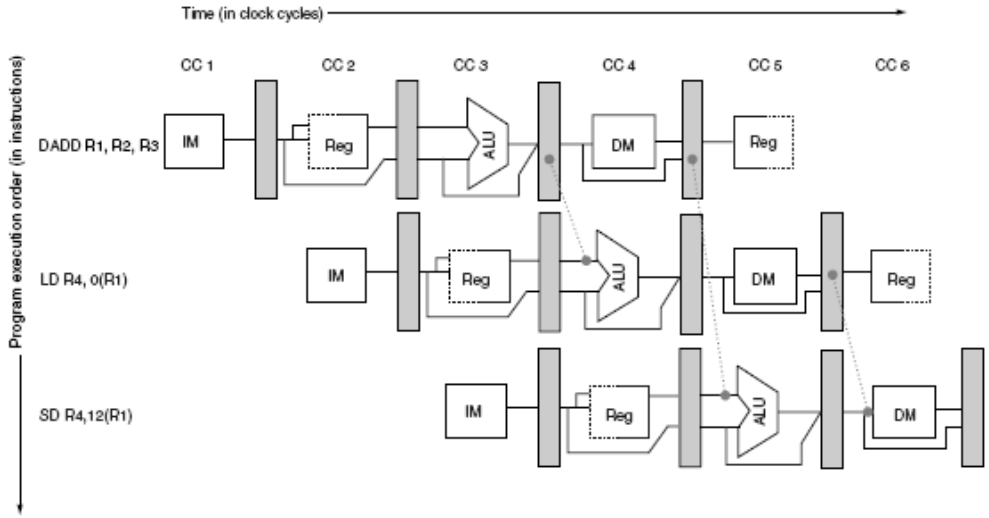

Рисунок 43 Схема №2 ускоренного продвижения данных в конвейере

3.8.8 Риски по управлению (Control Hazards)

Конфликты по управлению приводят к наибольшим потерям производительности и возникают в процессе исполнения команд, изменяющих естественный порядок вычислений[1]. Простейший конвейер ориентирован на линейные программы. В нем на стадии выборки команды извлекаются команды из последовательных ячеек памяти, используя для этого счетчик команд (PC). Адрес очередной команды в линейной программе формируется автоматически, за счет прибавления к содержимому СК числа, равного длине текущей команды (для MIPS: PC=PC+4). Однако в программах обязательно присутствуют команды переходов, возврата из процедур и т. п. Доля подобных команд в программе оценивается на уровне 10-20%. Выполнение команд перехода может приводить к приостановке конвейера на несколько тактов, из-за чего производительность процессора значительно снижается.

Выделяют два фактора, определяющих необходимость приостановки конвейера при выполнении команд перехода:

1) То, что команда относится к командам перехода, становится ясно только после завершения фазы декодирования кода команды (ID – Instruction Decoding), то есть по прошествии двух тактов с начала выполнения команды. За это время первые две ступени конвейера заполняются инструкциями, идущими за командой перехода в «естественном» порядке, т.е. последовательно. Если в процессе исполнения команды надо будет выполнить переход, то эти две первые ступени конвейера должны быть очищены и заполнены заново, что равносильно приостановке конвейера на два такта.
2) Второй фактор связан с командами условных переходов. Будет или не будет выполнен такой переход выясняется только на последней стадии исполнения таких команд (для MIPS - это 3-я стадия EXecution/Branch Completion), когда проверяется условие перехода. В случае невыполнения перехода конвейер работает без перерыва, без замедления. В случае же выполнения условия перехода, конвейер должен быть полностью очищен и заполнен заново, что равносильно приостановке его на время выполнения команды условного перехода.

3.8.8.1 Методы решения проблемы задержки на выборку команд перехода.

Для сокращений задержек конвейера, обусловленных выборкой команды из точки перехода, наиболее часто применяются следующие методы:

1) Вычисление исполнительного адреса перехода на ступени декодирования команды:

Как это было показано выше (см. Рисунок 32), на этапе декодирования команды (ID) осуществляется расчет исполнительного адреса для команд перехода и выборка этой команды осуществляется уже по окончании второй стадии исполнения команды перехода, на не третьей, как было бы, если бы адрес рассчитывался на последней стадии исполнения перехода, не одновременно, а после декодирования команды. Соответственно простой конвейера в таком случае сократится с двух тактов (IF + ID) до одного (IF). В данном случае могут возникать проблемы при выполнении команд условного перехода, до третьей стадии неизвестно, по какому пути пойдет исполнение команд (см. фактор 2 в предыдущем абзаце). Эта проблема решается использованием двух параллельных конвейеров команд, соответствующих путям для выполнения и невыполнения условия перехода.

2) Использование буфера адресов перехода:

Буфер адресов перехода (BTB, Branch Target Buffer) представляет собой кэш-память небольшой емкости, в которой хранятся АДРЕСА точек перехода для нескольких последних выполненных команд перехода. В роли тегов (заголовков каждой ячейки) для этой кэш-памяти выступают адреса соответствующих выполненных команд перехода. При исполнении программы перед выборкой очередной команды ее адрес (т.е. содержимое счетчика команд) сравнивается с тегами - адресами ранее выполненных команд перехода, представленных в BTB. Если адрес совпал, то для этой команды, адрес точки перехода не

вычисляется, а берется готовый (он был рассчитан ранее) из BTB. Благодаря этому выборка команды из точки перехода может быть начата одновременно с выборкой самой команды перехода. Если же исполняется команда, адрес которой в BTB отсутствует, то она обрабатывается стандартным образом: выборка(IF)-дешифрация/расчет адреса(ID)-исполнение(EX). Если это «новая», ранее не исполнявшаяся, команда перехода, то полученный при ее выполнении адрес точки перехода заносится в BTB. При замещении информации в BTB обычно применяется алгоритм LRU (из кэш-памяти вытесняется элемент, который не использовался дольше всего).

Применение BTB дает наибольший эффект, когда одни и те же команды перехода в программе выполняются многократно, например при выполнении программных циклов.

3) Использование кэш-памяти для хранения команд, расположенных в точке перехода:

Кэш-память команд, расположенных в точке перехода (BTIC, Branch Target Instruction Cache), - это усовершенствованный вариант BTB, где в ячейки кэш-памяти выполненных ранее команд перехода записывается не только адрес перехода, но также и код самой команды. За счет этого при повторном выполнении команды перехода можно исключить не только этап вычисления адреса точки перехода, но и этап выборки расположенной там команды. Преимущества BTIC в наибольшей степени проявляются при многократном исполнении одних и тех же команд перехода, например, при реализации программных циклов.

4) Использование буфера цикла:

Буфер цикла представляет собой маленькую быстродействующую память, входящую в состав первой ступени конвейера, где производится выборка команд. В буфере сохраняются коды «последних команд в той последовательности, в которой они выбирались. Когда имеет место переход, аппаратура сначала проверяет, нет ли нужной команды в буфере, и если это так, то команда извлекается из буфера. Этот механизм наиболее эффективен опять же при реализации циклов, чем и объясняется название буфера. Если буфер достаточно велик, чтобы охватить все тело цикла, выборку команд цикла из памяти достаточно выполнить только в первой итерации (один раз при первом исполнении тела цикла), а для последующих итераций команды уже находятся в буфере.

3.8.8.2 Методы решения проблемы задержки на исполнение условного перехода.

Основной проблемой при проектировании конвейеров команд являются проблемы условных переходов, поскольку именно они приводят к наибольшим задержкам в работе конвейера. Для устранения или частичного сокращения этих простоев были предложены различные способы, которые условно можно разделить на четыре группы:

1) Использование буферов предвыборки;
2) Использование множественных потоков конвейеров;
3) Задержанный переход;
4) Предсказание перехода:
   a. статическое;
   b. динамическое.

3.8.9 Потоковая модель процессора с конвейером.

Организация конвейерной обработки команд похожа на организацию мультитактовой обработки команд: процесс исполнения команды делится на несколько стадий, которые могут быть рассмотрены как независимые (суб)команды, которые выполняются последовательно друг за другом. Данные между этими стадиями передаются через специальные буферные регистры.

В отличие от мультитактовых процессоров, где каждая субкоманда (стадия) может использовать все аппаратные ресурсы процессора и, соответственно, в каждый момент

времени исполняется только одна субкоманда, при конвейерной обработке субкоманда (стадия) использует только часть аппаратных ресурсов процессора и одновременно исполняются другие стадии других команд, которые используют другие аппаратные ресурсы процессора. Если аппаратные ресурсы, используемые для различных стадий не пересекаются между собой (т.е. нет конфликтов-рисков), то они могут исполняться параллельно во времени.

Таким образом, аппаратный тракт исполнения команд делится на несколько независимых частей, каждая из которых предназначена для исполнения *четко определенной* стадии команд (например, стадии IF или EXE). После выполнения данной стадии для одной машинной команды, этот аппаратный блок начинает исполнять такую же стадию следующей команды и т.д. Параллельно другая часть тракта – другой аппаратный блок – исполняет другие стадии команд.

Так как на каждом такте каждый такой блок – ступень конвейера - получает полностью новый набор входных данных, включая новый код операции исполняемой команды, новый набор промежуточных результатов, полученных на предыдущих стадиях исполнения этой команды (другими ступенями конвейера), то между ступенями должны быть расположены буферные регистры, содержащие полностью или частично (в зависимости от необходимости для выполнения этой и последующих стадий) перечисленные данные: код операции, аргументы, промежуточные результаты, вычисленные адреса данных в памяти и переходов, признаки-флаги и т.п.

По завершению выполнения стадии, данные из входного буфера и новые значения, полученные на данной стадии выполнения команды, переписываются во входной буфер следующей ступени конвейера.

На рисунках (см. Рисунок 44 и Рисунок 45) для сравнения представлены тракты данных для мультитактового процессора и для конвейерного процессора.

Следует обратить внимание, что существуют линии обратной связи, по которым данные передаются «между» командами: например, адрес перехода передается в счетчик команд после стадии EX, когда на стадии IF находится уже третья команда после команды перехода. Другой пример, когда данные, записываемые в память после стадии WB. становятся доступными только команда, отстающей на три такта от текущей. Именно такие обратные связи приводят к возникновению конфликтам-рискам по данным и по управлению.

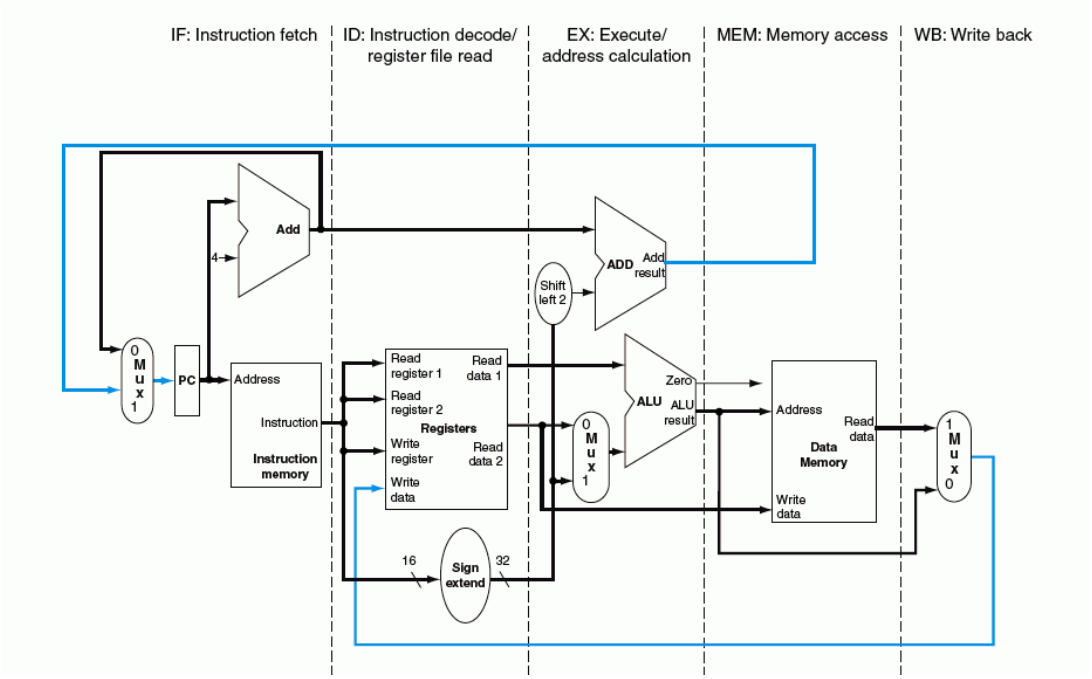

Рисунок 44 Потоковая модель мультитактового процессора.

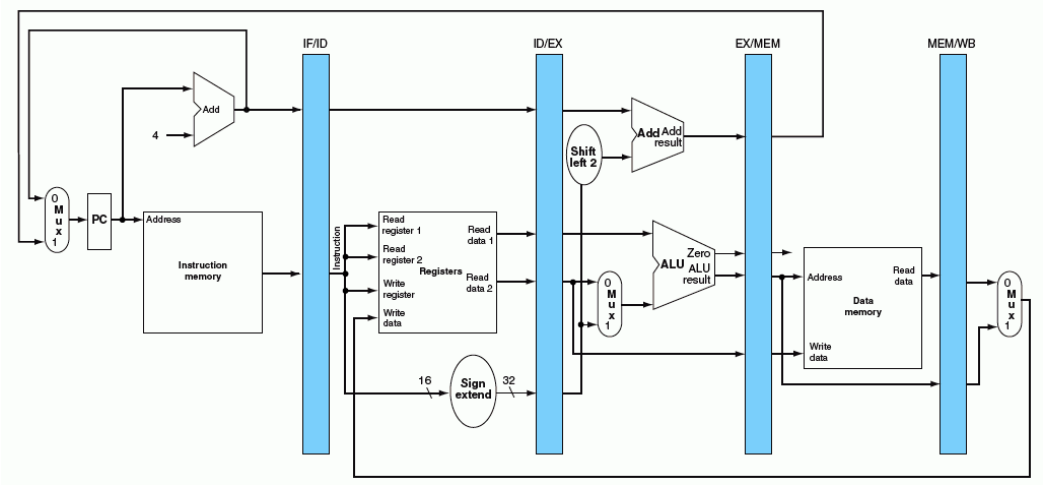

Рисунок 45 Потоковая модель конвейерного процессора с буферными регистрами.

ЛИТЕРАТУРА

Базовый учебник

1. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2007.

Базовое учебно-методическое пособие

2. Довгий П.С., Скорубский В.И. Проектирование ЭВМ / Учеб. пособие: cis.ifmo.ru.
3. Довгий П.С., Скорубский В.И. Лабораторный практикум по курсу «Организация ЭВМ и систем»: cis.ifmo.ru.

Основная литература

4. David A. Patterson, John L. Hennessy Computer Organization and Design. Hardware/Software Interface. 3rd edition. – San Francisco, CA94111: Morgan Kaufmann Publishers is an imprint of Elsevier, 2005.
5. David A. Patterson, John L. Hennessy Computer Architecture. A Quantitative Approach. 4rd edition. – San Francisco, CA94111: Morgan Kaufmann Publishers is an imprint of Elsevier, 2007.
6. Мелехин В.Ф., Павловский У.Г. Вычислительные машины, системы и сети: Учебник для студ. высш. учеб. заведений. М.: Изд. центр «Академия», 2006.
7. Таненбаум Э. Архитектура компьютера. 5-е изд. –СПб.: Питер, 2007.
8. Довгий П.С., Поляков В.И. Прикладная архитектура базовой модели процессора Intel. Часть 1 / Учеб. пособие: cis.ifmo.ru.
9. Довгий П.С., Поляков В.И. Прикладная архитектура базовой модели процессора Intel. Часть 2. Базовая система команд: cis.ifmo.ru.
10. Довгий П.С. Основные архитектурные принципы ЭВМ / Конспект лекций по курсу «Организация ЭВМ и систем»: cis.ifmo.ru.
11. Довгий П.С. Организация виртуальной памяти / Конспект лекций по курсу «Организация ЭВМ и систем»: cis.ifmo.ru.
12. Довгий П.С. Организация кэш-памяти / Конспект лекций по курсу «Организация ЭВМ и систем»: cis.ifmo.ru.
13. Довгий П.С. Организация прерываний / Конспект лекций по курсу «Организация ЭВМ и систем»: cis.ifmo.ru.

Дополнительная литература

12. Sajjan G. Shiva Computer Organization, Design, and Architecture. 4rd edition. – CRC Press, 2008.
13. Угрюмов Е.П. Цифровая схемотехника.Уч пособие для ВУЗов. 2-е изд. – СПб.: БХВ-Петербург,2007.
14. Столлингс В. Структурная организация и архитектура компьютерных систем. 5-е изд. –М.: Изд. дом «Вильямс», 2002.
15. Гук М.Ю. Аппаратные средства IBM PC. Энциклопедия. 3-е изд. –СПб.: Питер, 2006.
16. Микропроцессорные системы: Учеб. пособие для вузов / Е.К. Александров, Р.И. Грушевицкий, М.С. Куприянов и др.; под общ. ред. Д.В. Пузанкова. –СПб.: Политехника, 2002.
17. Микропроцессорный комплект К1810: структура, программирование, применение: Справочная книга / Ю.М. Казаринов, В.Н. Номоконов и др.; Под ред. Ю.М. Казаринова. –М.: Высш. шк., 1990.
18. Assembler / В. Юров. –СПб.: Питер, 2000.
19. Гук М. Процессоры Pentium II, Pentium Pro и просто Pentium –СПб.: Питер, 1999.
20. Гук М., Юров В. Процессоры Pentium Athlon и Diron. –СПб.: Питер, 2001.
21. Шагурин И.И., Бердышев Е.М. Процессоры семейства Intel P6. Архитектура, программирование, интерфейс. –М.: Горячая линия – Телеком, 2000.