4. АРХИТЕКТУРА МИКРОПРОЦЕССОРНЫХ СИСТЕМ

4.1. Организация компьютерных систем

4.1.1. Основные понятия

**Электронная система** – в данном случае это любой электронный узел, блок, прибор или комплекс, производящий обработку информации.

**Задача** – это набор функций, выполнение которых требуется от электронной системы.

**Быстродействие** – это показатель скорости выполнения электронной системой ее функций.

**Гибкость** – это способность системы подстраиваться под различные задачи.

**Избыточность** – это показатель степени соответствия возможностей системы решаемой данной системой задаче.

**Интерфейс** – соглашение об обмене информацией, правила обмена информацией, подразумевающие электрическую, логическую и конструктивную совместимость устройств, участвующих в обмене. Другое название – **сопряжение**.

**Микропроцессорная система** может рассматриваться как частный случай электронной системы, предназначенной для обработки входных сигналов и выдачи выходных сигналов ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.1)). В качестве входных и выходных сигналов при этом могут использоваться аналоговые сигналы, одиночные цифровые сигналы, цифровые коды, последовательности цифровых кодов. Внутри системы может производиться хранение, накопление сигналов (или информации), но суть от этого не меняется. Если система цифровая (а *микропроцессорные системы* относятся к разряду цифровых), то входные аналоговые сигналы преобразуются в последовательности кодов выборок с помощью аналого-цифрового преобразователя (*АЦП*), а выходные аналоговые сигналы формируются из последовательности кодов выборок с помощью цифро-аналогового преобразователя (*ЦАП*). Обработка и хранение информации производятся в цифровом виде.

Характерная особенность традиционной цифровой системы состоит в том, что алгоритмы обработки и хранения информации в ней *жестко связаны* со схемотехникой системы, т. е. изменение этих алгоритмов возможно только путем изменения структуры системы, замены электронных узлов, входящих в систему, и/или связей между ними. Например, если нам нужна дополнительная операция суммирования, то необходимо добавить в структуру системы лишний *сумматор*. Или если нужна дополнительная функция хранения кода в течение одного такта, то мы должны добавить в структуру еще один регистр. Естественно, это практически невозможно сделать в процессе эксплуатации, обязательно нужен новый производственный цикл проектирования, изготовления, отладки всей системы. Именно поэтому традиционная цифровая система часто называется системой на «жесткой логике».

Аналоговые

сигналы

Цифровые коды

Цифровые

сигналы

Электронная система (обработка, хранение)

Аналоговые

сигналы

Цифровые

сигналы

Цифровые коды

Рис. 4.1. Электронная система

Любая система на «жесткой логике» обязательно представляет собой специализированную систему, настроенную исключительно на одну задачу или (реже) на несколько близких, заранее известных задач. Это имеет свои бесспорные преимущества.

Во-первых, специализированная система (в отличие от универсальной) никогда не имеет аппаратурной избыточности, т. е. каждый ее элемент обязательно работает в полную силу (конечно, если эта система грамотно спроектирована).

Во-вторых, именно специализированная система может обеспечить максимально высокое быстродействие, так как скорость выполнения алгоритмов обработки информации определяется в ней только быстродействием отдельных логических элементов и выбранной схемой путей прохождения информации. А именно логические элементы всегда обладают максимальным на данный момент быстродействием.

Но в то же время большим недостатком цифровой системы на «жесткой логике» является то, что для каждой новой задачи ее надо проектировать и изготавливать заново. Это процесс длительный, дорогостоящий, требующий высокой квалификации исполнителей. А если решаемая задача вдруг изменяется, то вся аппаратура должна быть полностью заменена. В нашем быстро меняющемся мире это довольно расточительно.

Путь преодоления этого недостатка очевиден: надо построить такую систему, которая могла бы легко адаптироваться под любую задачу, перестраиваться с одного алгоритма работы на другой без изменения аппаратуры. И задавать тот или иной алгоритм мы тогда могли бы путем ввода в систему некой дополнительной управляющей информации, **программы** работы системы ([рис. 4.2](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.2)). Тогда система станет универсальной, или **программируемой**, не жесткой, а гибкой. Именно это и обеспечивает *микропроцессорная система*.

Аналоговые

сигналы

Цифровые коды

Цифровые

сигналы

Электронная система

Аналоговые

сигналы

Цифровые

сигналы

Цифровые коды

Управляющая

информация

(программа)

Рис. 4.2. Программируемая (она же универсальная) электронная система

Но любая универсальность обязательно приводит к избыточности. Ведь решение максимально трудной задачи требует гораздо больше средств, чем решение максимально простой задачи. Поэтому сложность универсальной системы должна быть такой, чтобы обеспечивать решение самой трудной задачи, а при решении простой задачи система будет работать далеко не в полную силу, будет использовать не все свои ресурсы. И чем проще решаемая задача, тем больше избыточность и тем менее оправданной становится универсальность. Избыточность ведет к увеличению стоимости системы, снижению ее надежности, увеличению потребляемой мощности и т. д.

Кроме того, универсальность, как правило, приводит к существенному снижению быстродействия. Оптимизировать универсальную систему так, чтобы каждая новая задача решалась максимально быстро, попросту невозможно. Общее правило таково: чем больше универсальность, гибкость, тем меньше быстродействие. Более того, для универсальных систем не существует таких задач (пусть даже и самых простых), которые бы они решали с максимально возможным быстродействием. За все приходится платить.

Таким образом, можно сделать следующий вывод. Системы на «жесткой логике» хороши там, где решаемая задача не меняется длительное время, где требуется самое высокое быстродействие, где алгоритмы обработки информации предельно просты. А универсальные, программируемые системы хороши там, где часто меняются решаемые задачи, где высокое быстродействие не слишком важно, где алгоритмы обработки информации сложные. Таким образом, любая система хороша на своем месте.

Однако за последние десятилетия быстродействие универсальных (*микропроцессорных*) систем сильно выросло (на несколько порядков). К тому же большой объем выпуска микросхем для этих систем привел к резкому снижению их стоимости. В результате область применения систем на «жесткой логике» резко сузилась. Более того, высокими темпами развиваются сейчас программируемые системы, предназначенные для решения одной задачи или нескольких близких задач. Они удачно совмещают в себе как достоинства систем на «жесткой логике», так и программируемых систем, обеспечивая сочетание достаточно высокого быстродействия и необходимой гибкости. Так что *вытеснение* «жесткой логики» продолжается.

4.1.2. Что такое микропроцессор

Ядром любой *микропроцессорной системы* является *микропроцессор* или просто процессор (от англ. *processor*). Перевести на русский язык это слово правильнее всего как «обработчик», так как именно ***микропроцессор*** – это тот узел, блок, который производит всю обработку информации внутри *микропроцессорной системы*. Остальные узлы выполняют всего лишь *вспомогательные функции*: хранение информации (в том числе и управляющей информации, т. е. программы), связи с внешними устройствами, связи с пользователем и т.д. Процессор заменяет практически всю «жесткую логику», которая понадобилась бы в случае традиционной цифровой системы. Он выполняет *арифметические функции* (сложение, умножение и т. д.), *логические* функции (сдвиг, сравнение, маскирование кодов и т. д.), временное хранение кодов (во внутренних регистрах), пересылку кодов между узлами *микропроцессорной системы* и многое другое. Количество таких элементарных операций, выполняемых процессором, может достигать нескольких сотен. Процессор можно сравнить с мозгом системы.

Но при этом надо учитывать, что все свои операции процессор выполняет **последовательно**, т. е. одну за другой, по очереди. Конечно, существуют процессоры с параллельным выполнением некоторых операций, встречаются также *микропроцессорные системы*, в которых несколько процессоров работают над одной задачей параллельно, но это редкие исключения. С одной стороны, последовательное выполнение операций – несомненное достоинство, так как позволяет с помощью всего одного процессора выполнять любые, самые сложные алгоритмы обработки информации. Но с другой стороны, последовательное выполнение операций приводит к тому, что время выполнения алгоритма зависит от его сложности. Простые алгоритмы выполняются быстрее сложных. То есть *микропроцессорная система* способна сделать все, но работает под управлением программы (рис. 4.3).

Процессор

Управляющая

информационная

(программа)

Рис. 4.3. Информационные потоки в микропроцессорной системе

Итак, *микропроцессор* способен выполнять множество операций. Но откуда он узнает, какую операцию ему надо выполнять в данный момент? Именно это определяется управляющей информацией, программой. Программа представляет собой набор **команд (инструкций)**, т. е. цифровых кодов, расшифровав которые, процессор узнает, что ему надо делать. Программа от начала и до конца составляется человеком, программистом, а процессор выступает в роли послушного исполнителя этой программы, никакой инициативы он не проявляет (если, конечно, исправен). Поэтому сравнение процессора с мозгом не слишком корректно. Он всего лишь исполнитель того алгоритма, который заранее составил для него человек. Любое отклонение от этого алгоритма может быть вызвано только неисправностью процессора или каких-нибудь других узлов *микропроцессорной системы*.

Все команды, выполняемые процессором, образуют **систему команд** процессора. Структура и объем системы команд процессора определяют его быстродействие, гибкость, *удобство использования*. Всего команд у процессора может быть от нескольких десятков до нескольких сотен. Система команд может быть рассчитана на узкий круг решаемых задач (у специализированных процессоров) или на максимально широкий круг задач (у универсальных процессоров). Коды команд могут иметь различное количество разрядов (занимать от одного до нескольких байт). Каждая команда имеет свое время выполнения, поэтому время выполнения всей программы зависит не только от количества команд в программе, но и от того, какие именно команды используются.

По числу больших интегральных схем (БИС) в микропроцессорном комплекте различают микропроцессоры однокристальные, многокристальные и многокристальные секционные.

Процессоры даже самых простых ЭВМ имеют сложную функциональную структуру, содержат большое количество электронных элементов и множество разветвленных связей. Изменять структуру процессора необходимо так, чтобы полная принципиальная схема или ее части имели количество элементов и связей, совместимое с возможностями БИС. При этом микропроцессоры приобретают внутреннюю магистральную архитектуру, т. е. в них к единой внутренней информационной магистрали подключаются все основные функциональные блоки (арифметико-логический, рабочих регистров, стека, прерываний, интерфейса, управления и синхронизации и др.).

Для обоснования классификации микропроцессоров по числу БИС надо распределить все аппаратные блоки процессора между основными тремя функциональными частями: операционной, управляющей и интерфейсной. Сложность операционной и управляющей частей процессора определяется их разрядностью, системой команд и требованиями к системе прерываний; сложность интерфейсной части — разрядностью и возможностями подключения других устройств ЭВМ (памяти, внешних устройств, датчиков и исполнительных механизмов и др.). Интерфейс процессора содержит несколько десятков информационных шин данных (ШД), адресов (ША) и управления (ШУ).

Однокристальные микропроцессоры получаются при реализации всех аппаратных средств процессора в виде одной БИС или СБИС (сверхбольшой интегральной схемы). По мере увеличения степени интеграции элементов в кристалле и числа выводов корпуса параметры однокристальных микропроцессоров улучшаются. Однако возможности однокристальных микропроцессоров ограничены аппаратными ресурсами кристалла и корпуса. Для получения многокристального микропроцессора необходимо провести разбиение его логической структуры на функционально законченные части и реализовать их в виде БИС (СБИС). Функциональная законченность БИС многокристального микропроцессора означает, что его части выполняют заранее определенные функции и могут работать автономно.

На рис. 4.4, *а* показано функциональное разбиение структуры процессора при создании трехкристального микропроцессора (пунктирные линии), содержащего БИС операционного (ОП), управляющего (УП) и интерфейсного (ИП) процессоров. Обмен информацией производится по магистрали адреса (МА), магистрали данных (МД) и магистрали управления (МД).

УП

ОП

ИП

УП

ОП

ИП

МА

МД

МУ

*а* *б*

Рис. 4.4. Процессор:

*а* – функциональная структура процессора; *б* – разбиение структуры процессора для реализации процессора в виде комплекта секционных БИС

Операционный процессор служит для обработки данных, управляющий процессор выполняет функции выборки, декодирования и вычисления адресов операндов и также генерирует последовательности микрокоманд. Автономность работы и большое быстродействие БИС УП позволяет выбирать команды из памяти с большей скоростью, чем скорость их исполнения БИС ОП. При этом в УП образуется очередь еще не исполненных команд, а также заранее подготавливаются те данные, которые потребуются ОП в следующих циклах работы. Такая опережающая выборка команд экономит время ОП на ожидание операндов, необходимых для выполнения команд программ. Интерфейсный процессор позволяет подключить память и периферийные средства к микропроцессору; он, по существу, является сложным контроллером для устройств ввода/вывода информации. БИС ИП выполняет также функции канала прямого доступа к памяти.

Выбираемые из памяти команды распознаются и выполняются каждой частью микропроцессора автономно и поэтому может быть обеспечен режим одновременной работы всех БИС МП, т. е. конвейерный поточный режим исполнения последовательности команд программы (выполнение последовательности с небольшим временным сдвигом). Такой режим работы значительно повышает производительность микропроцессора.

Многокристальные секционные микропроцессоры получаются в том случае, когда в виде БИС реализуются части (секции) логической структуры процессора при функциональном разбиении ее вертикальными плоскостями (рис. 4.4, *б*). Для построения многоразрядных микропроцессоров при параллельном включении секций БИС в них добавляются средства «стыковки».

Для создания высокопроизводительных многоразрядных микропроцессоров требуется столь много аппаратных средств, не реализуемых в доступных БИС, что может возникнуть необходимость еще и в функциональном разбиении структуры микропроцессора горизонтальными плоскостями. В результате рассмотренного функционального разделения структуры микропроцессора на функционально и конструктивно законченные части создаются условия реализации каждой из них в виде БИС. Все они образуют комплект секционных БИС МП.

Таким образом, микропроцессорная секция – это БИС, предназначенная для обработки нескольких разрядов данных или выполнения определенных управляющих операций. Секционность БИС МП определяет возможность «наращивания» разрядности обрабатываемых данных или усложнения устройств управления микропроцессора при «параллельном» включении большего числа БИС.

Однокристальные и трехкристальные БИС МП, как правило, изготовляют на основе микроэлектронных технологий униполярных полупроводниковых приборов, а многокристальные секционные БИС МП — на основе технологии биполярных полупроводниковых приборов. Использование многокристальных микропроцессорных высокоскоростных биполярных БИС, имеющих функциональную законченность при малой физической разрядности обрабатываемых данных и монтируемых в корпус с большим числом выводов, позволяет организовать разветвление связи в процессоре, а также осуществить конвейерные принципы обработки информации для повышения его производительности.

По назначению различают универсальные и специализированные микропроцессоры.

Универсальные микропроцессоры могут быть применены для решения широкого круга разнообразных задач. При этом их эффективная производительность слабо зависит от проблемной специфики решаемых задач. Специализация МП, т. е. его проблемная ориентация на ускоренное выполнение определенных функций, позволяет резко увеличить эффективную производительность при решении только определенных задач.

Среди специализированных микропроцессоров можно выделить различные микроконтроллеры, ориентированные на выполнение сложных последовательностей логических операций, математические МП, предназначенные для повышения производительности при выполнении арифметических операций за счет, например, матричных методов их выполнения, МП для обработки данных в различных областях применений и т. д. С помощью специализированных МП можно эффективно решать новые сложные задачи параллельной обработки данных. Например, конволюция позволяет осуществить более сложную математическую обработку сигналов, чем широко используемые методы корреляции. Последние в основном сводятся к сравнению и определению подобия всего двух серий данных: входных, передаваемых формой сигнала, и фиксированных опорных. Конволюция дает возможность в реальном масштабе времени находить соответствие для сигналов изменяющейся формы путем сравнения их с различными эталонными сигналами, что, например, может позволить эффективно выделить полезный сигнал на фоне шума.

Разработанные однокристальные конвольверы используются в устройствах опознавания образов в тех случаях, когда возможности сбора данных превосходят способности системы обрабатывать эти данные.

По виду обрабатываемых входных сигналов различают цифровые и аналоговые микропроцессоры. Сами микропроцессоры – цифровые устройства, однако они могут иметь встроенные аналого-цифровые и цифро-аналоговые преобразователи. Поэтому входные аналоговые сигналы передаются в МП через преобразователь в цифровой форме, обрабатываются и после обратного преобразования в аналоговую форму поступают на выход. С архитектурной точки зрения такие микропроцессоры представляют собой аналоговые функциональные преобразователи сигналов и называются аналоговыми микропроцессорами. Они выполняют функции любой аналоговой схемы (например, производят генерацию колебаний, модуляцию, смещение, фильтрацию, кодирование и декодирование сигналов в реальном масштабе времени и т. д., заменяя сложные схемы, состоящие из операционных усилителей, катушек индуктивности, конденсаторов и т. д.). При этом применение аналогового микропроцессора значительно повышает точность обработки аналоговых сигналов и их воспроизводимость, а также расширяет функциональные возможности за счет программной «настройки» цифровой части микропроцессора на различные алгоритмы обработки сигналов.

Обычно в составе однокристальных аналоговых МП имеется несколько каналов аналого-цифрового и цифро-аналогового преобразования. В аналоговом микропроцессоре разрядность обрабатываемых данных достигает 24 бит и более, большое значение уделяется увеличению скорости выполнения арифметических операций.

Отличительная черта аналоговых микропроцессоров – способность к переработке большого объема числовых данных, т. е. к выполнению операций сложения и умножения с большой скоростью при необходимости даже за счет отказа от операций прерываний и переходов. Аналоговый сигнал, преобразованный в цифровую форму, обрабатывается в реальном масштабе времени и передается на выход обычно в аналоговой форме через цифро-аналоговый преобразователь. При этом согласно теореме Котельникова частота квантования аналогового сигнала должна вдвое превышать верхнюю частоту сигнала.

Сравнение цифровых микропроцессоров производится сопоставлением времени выполнения ими списков операций. Сравнение же аналоговых микропроцессоров производится по количеству эквивалентных звеньев аналого-цифровых фильтров рекурсивных фильтров второго порядка. Производительность аналогового микропроцессора определяется его способностью быстро выполнять операции умножения: чем быстрее осуществляется умножение, тем больше эквивалентное количество звеньев фильтра в аналоговом преобразователе и тем более сложный алгоритм преобразования цифровых сигналов можно задавать в микропроцессоре.

Одним из направлений дальнейшего совершенствования аналоговых микропроцессоров является повышение их универсальности и гибкости. Поэтому вместе с повышением скорости обработки большого объема цифровых данных будут развиваться средства обеспечения развитых вычислительных процессов обработки цифровой информации за счет реализации аппаратных блоков прерывания программ и программных переходов.

По характеру временной организации работы микропроцессоры делят на синхронные и асинхронные.

Синхронные микропроцессоры – микропроцессоры, в которых начало и конец выполнения операций задаются устройством управления (время выполнения операций в этом случае не зависит от вида выполняемых команд и величин операндов).

Асинхронные микропроцессоры позволяют начало выполнения каждой следующей операции определить по сигналу фактического окончания выполнения предыдущей операции. Для более эффективного использования каждого устройства микропроцессорной системы в состав асинхронно работающих устройств вводят электронные цепи, обеспечивающие автономное функционирование устройств. Закончив работу над какой-либо операцией, устройство вырабатывает сигнал запроса, означающий его готовность к выполнению следующей операции. При этом роль естественного распределителя работ принимает на себя память, которая в соответствии с заранее установленным приоритетом выполняет запросы остальных устройств по обеспечению их командной информацией и данными.

По организации структуры микропроцессорных систем различают микроЭВМ одно- и многомагистральные.

В одномагистральных микроЭВМ все устройства имеют одинаковый интерфейс и подключены к единой информационной магистрали, по которой передаются коды данных, адресов и управляющих сигналов.

В многомагистральных микроЭВМ устройства группами подключаются к своей информационной магистрали. Это позволяет осуществить одновременную передачу информационных сигналов по нескольким (или всем) магистралям. Такая организация систем усложняет их конструкцию, однако увеличивает производительность.

По количеству выполняемых программ различают одно- и многопрограммные микропроцессоры.

В однопрограммных микропроцессорах выполняется только одна программа. Переход к выполнению другой программы происходит после завершения текущей программы.

В много- или мультипрограммных микропроцессорах одновременно выполняется несколько (обычно несколько десятков) программ. Организация мультипрограммной работы микропроцессорных управляющих систем позволяет осуществить контроль за состоянием и управлением большим числом источников или приемников информации.

**4.1.3. Структура типового микропроцессора**

Архитектура типичной небольшой вычислительной системы на основе микроЭВМ показана на рис. 4.5. Такая микроЭВМ содержит все пять основных блоков цифровой машины: устройство ввода информации, управляющее устройство (УУ), арифметико-логическое устройство (АЛУ) (входящее в состав микропроцессора), запоминающие устройства (ЗУ) и устройство вывода информации.

Порты

Входы

Память данных ЗУПВ

Память программ ПЗУ

Микропроцессор

Выходы

Порты

От периферийных устройств

Шина

управления

Шина

адреса

16 линий

Шина

данных

(8 линий)

Рис. 4.5. Архитектура типового микропроцессора

Микропроцессор координирует работу всех устройств цифровой системы с помощью шины управления (ШУ). Помимо ШУ имеется 16-разрядная адресная шина (ША), которая служит для выбора определенной ячейки памяти, порта ввода или порта вывода. По 8-разрядной информационной шине или шине данных (ШД) осуществляется двунаправленная пересылка данных к микропроцессору и от микропроцессора. Важно отметить, что МП может посылать информацию в память микроЭВМ или к одному из портов вывода, а также получать информацию из памяти или от одного из портов ввода.

Постоянное запоминающее устройство (ПЗУ) в микроЭВМ содержит некоторую программу (на практике программу инициализации ЭВМ). Программы могут быть загружены в запоминающее устройство с произвольной выборкой (ЗУПВ) и из внешнего запоминающего устройства (ВЗУ). Это программы пользователя.

В качестве примера, иллюстрирующего работу микроЭВМ, рассмотрим процедуру, для реализации которой нужно выполнить следующую последовательность элементарных операций:

1) нажать клавишу с буквой «А» на клавиатуре;

2) поместить букву «А» в память микроЭВМ;

3) вывести букву «А» на экран дисплея.

Это типичная процедура ввода-запоминания-вывода, рассмотрение которой дает возможность пояснить принципы использования некоторых устройств, входящих в микроЭВМ.

На рис. 4.6 приведена подробная диаграмма выполнения процедуры ввода-запоминания-вывода. Обратите внимание, что команды уже загружены в первые шесть ячеек памяти. Хранимая программа содержит следующую цепочку команд:

1) ввести данные из порта ввода 1;

2) запомнить данные в ячейке памяти 200;

3) переслать данные в порт вывода 10.

В данной программе всего три команды, хотя на рис. 4.6 может показаться, что в памяти программ записано шесть команд. Это связано с тем, что команда обычно разбивается на части. Первая часть команды 1 в приведенной выше программе – команда ввода данных. Во второй части команды 1 указывается, откуда нужно ввести данные (из порта 1). Первая часть команды, предписывающая конкретное действие, называется кодом операции (КОП), а вторая часть – операндом. Коды операции и операнд размещаются в отдельных ячейках памяти программ. На рис. 4.6 КОП хранится в ячейке 100, а код операнда – в ячейке 101 (порт 1); последний указывает, откуда нужно взять информацию.

В МП на рис. 4.6 выделены еще два новых блока – регистры: аккумулятор и регистр команд.

ПОРТ1

ВХОД

Аккумулятор

МП

Регистр команд

Адрес

200

202

203

Содержимое

201

«А»

11

16

ВЫХОД

Порт 10

Адрес

100

101

102

103

104

105

106

Ввести данные

Из порта 1

Запомнить данные

В ячейки памяти 200

Вывести данные

В порт 10

Содержимое

1

3

6

8

12

14

16

2

4

7

9

13

15

5

16

5

Код символа «А» с

Клавиатуры

Код символа «А» к монитору

Память программ

ЗУПВ

10

Рис. 4.6. Диаграмма выполнения процедуры ввода-запоминания-вывода

Рассмотрим прохождение команд и данных внутри микроЭВМ с помощью занумерованных кружков на диаграмме. Напомним, что микропроцессор – это центральный узел, управляющий перемещением всех данных и выполнением операций.

Итак, при выполнении типичной процедуры ввода-запоминания-вывода в микроЭВМ происходит следующая последовательность действий.

1. МП выдает адрес 100 на шину адреса. По шине управления поступает сигнал, устанавливающий память программ (конкретную микросхему) в режим считывания.

2. ЗУ программ пересылает первую команду («Ввести данные») по шине данных, и МП получает это закодированное сообщение. Команда помещается в регистр команд. МП декодирует (интерпретирует) полученную команду и определяет, что для команды нужен операнд.

3. МП выдает адрес 101 на ША; ШУ используется для перевода памяти программ в режим считывания.

4. Из памяти программ на ШД пересылается операнд «Из порта 1». Этот операнд находится в программной памяти в ячейке 101. Код операнда (содержащий адрес порта 1) передается по ШД к МП и направляется в регистр команд. МП теперь декодирует полную команду («Ввести данные из порта 1»).

5. МП, используя ША и ШУ, связывающие его с устройством ввода, открывает порт 1. Цифровой код буквы «А» передается в аккумулятор внутри МП и запоминается. Важно отметить, что при обработке каждой программной команды МП действует согласно микропроцедуре выборки-декодирования-исполнения.

6. МП обращается к ячейке 102 по ША. ШУ используется для перевода памяти программ в режим считывания.

7. Код команды «Запомнить данные» подается на ШД и пересылается в МП, где помещается в регистр команд.

8. МП дешифрирует эту команду и определяет, что для нее нужен операнд. МП обращается к ячейке памяти 103 и приводит в активное состояние вход считывания микросхем памяти программ.

9. Из памяти программ на ШД пересылается код сообщения «В ячейке памяти 200». МП воспринимает этот операнд и помещает его в регистр команд. Полная команда «Запомнить данные в ячейке памяти 200» выбрана из памяти программ и декодирована.

10. Теперь начинается процесс выполнения команды. МП пересылает адрес 200 на ША и активизирует вход записи, относящийся к памяти данных.

11. МП направляет хранящуюся в аккумуляторе информацию в память данных. Код буквы «А» передается по ШД и записывается в ячейку 200 этой памяти. Выполнена вторая команда. Процесс запоминания не разрушает содержимого аккумулятора. В нем по-прежнему находится код буквы «А».

12. МП обращается к ячейке памяти 104 для выбора очередной команды и переводит память программ в режим считывания.

13. Код команды вывода данных пересылается по ШД к МП, который помещает ее в регистр команд, дешифрирует и определяет, что нужен операнд.

14. МП выдает адрес 105 на ША и устанавливает память программ в режим считывания.

15. Из памяти программ по ШД к МП поступает код операнда «В порт 10», который далее помещается в регистр команд.

16. МП дешифрирует полную команду «Вывести данные в порт 10». С помощью ША и ШУ, связывающих его с устройством вывода, МП открывает порт 10, пересылает код буквы «А» (все еще находящийся в аккумуляторе) по ШД. Буква «А» выводится через порт 10 на экран дисплея.

В большинстве микропроцессорных систем (МПС) передача информации осуществляется способом, аналогичным рассмотренному выше. Наиболее существенные различия возможны в блоках ввода и вывода информации.

Подчеркнем еще раз, что именно микропроцессор является ядром системы и осуществляет управление всеми операциями. Его работа представляет последовательную реализацию микропроцедур выборки-дешифрации-исполнения. Однако фактическая последовательность операций в МПС определяется командами, записанными в памяти программ.

Таким образом, в МПС микропроцессор выполняет следующие функции:

— выборку команд программы из основной памяти;

— дешифрацию команд;

— выполнение арифметических, логических и других операций, закодированных в командах;

— управление пересылкой информации между регистрами и основной памятью, между устройствами ввода/вывода;

— отработку сигналов от устройств ввода/вывода, в том числе реализацию прерываний с этих устройств;

— управление и координацию работы основных узлов МП.

**4.1.4. Логическая структура микропроцессора**

Логическая структура микропроцессора, т. е. конфигурация составляющих микропроцессор логических схем и связей между ними, определяется функциональным назначением. Именно структура задает состав логических блоков микропроцессора и то, как эти блоки должны быть связаны между собой, чтобы полностью отвечать архитектурным требованиям. Срабатывание электронных блоков микропроцессора в определенной последовательности приводит к выполнению заданных архитектурой микропроцессора функций, т. е. к реализации вычислительных алгоритмов. Одни и те же функции можно выполнить в микропроцессорах со структурой, отличающейся набором, количеством и порядком срабатывания логических блоков. Различные структуры микропроцессоров, как правило, обеспечивают их различные возможности, в том числе и различную скорость обработки данных. Логические блоки микропроцессора с развитой архитектурой показаны на рис. 4.7.

БС

БУВВ

БУВП

БУПК

БУФКА

БУПРПр

БУВОп

БЗП

РгСОЗУ

АЛБ

БДА1

БДА*n*

II

I

Рис. 4.7. Общая логическая структура микропроцессора:

I – управляющая часть; II – операционная часть; БС – блок синхронизации; БУВВ – блок управления вводом/выводом; БУВП – блок управления виртуальной памятью; БУПК – блок управления последовательно-стью команд; БУВОп – блок управления выполнением операций; БУФКА – блок управления формированием кодов адресов; БЗП – блок защиты памяти; БУПРПр – блок управления прерыванием работы процессора; РгСОЗУ –­ регистровое сверхоперативное запоминающее устройство; АЛБ – арифметико-логический блок; БДА – блок дополнительной арифметики.

При проектировании логической структуры микропроцессоров необходимо рассмотреть:

1) номенклатуру электронных блоков, необходимую и достаточную для реализации архитектурных требований;

2) способы и средства реализации связей между электронными блоками;   
 3) методы отбора наиболее рациональных вариантов логических структур из возможного числа структур с отличающимся составом блоков и конфигурацией связей между ними.

При проектировании микропроцессора приводятся в соответствие внутренняя сложность кристалла и количество выводов корпуса. Относительный рост числа элементов по мере развития микроэлектронной технологии во много раз превышает относительное увеличение числа выводов корпуса, поэтому проектирование БИС в виде конечного автомата, а не в виде набора схем, реализующих некоторый набор логических переключательных функций и схем памяти, дает возможность получить функционально законченные блоки и устройства ЭВМ.

Использование микропроцессорных комплектов БИС позволяет создать микроЭВМ для широких областей применения вследствие программной адаптации микропроцессора к конкретной области применения: изменяя программу работы микропроцессора, изменяют функции информационно-управляющей системы. Поэтому за счет составления программы работы микропроцессоров в конкретных условиях работы определенной системы можно получить оптимальные характеристики последней.

Если уровень только программной «настройки» микропроцессоров не позволит получить эффективную систему, доступен следующий уровень проектирования – микропрограммный. За счет изменения содержимого ПЗУ или программируемой логической матрицы (ПЛМ) можно «настроиться» на более специфичные черты системы обработки информации. В этом случае частично за счет изменения микропрограмм затрагивается аппаратный уровень системы. Технико-экономические последствия здесь связаны лишь с ограниченным вмешательством в технологию изготовления управляющих блоков микроЭВМ.

Изменение аппаратного уровня информационно-управляющей микропроцессорной системы, включающего в себя функциональные БИС комплекта, одновременно с конкретизацией микропрограммного и программного уровней позволяет наилучшим образом удовлетворить требованиям, предъявляемым к системе.

Решение задач управления в конкретной системе чисто аппаратными средствами (аппаратная логика) дает выигрыш в быстродействии, однако приводит к сложностям при модификации системы. Микропроцессорное решение (программная логика) является более медленным, но более гибким решением, позволяющим развивать и модифицировать систему. Изменение технических требований к информационно-управляющей микропроцессорной системе ведет лишь к необходимости перепрограммирования работы микропроцессора. Именно это качество обеспечивает высокую логическую гибкость микропроцессоров, определяет возможность их широкого использования, а значит и крупносерийного производства.

**4.1.5. Типы архитектур**

Существует несколько подходов к классификации микропроцессоров по типу архитектуры. Так, выделяют МП с CISC (Complete Instruction Set Computer) архитектурой, характеризуемой полным набором команд, и RISC (Reduce Instruction Set Computer) архитектурой, которая определяет систему с сокращенным набором команд одинакового формата, выполняемых за один такт МП.

Определяя в качестве основной характеристики МП разрядность, выделяют следующие типы МП архитектуры:

* с фиксированной разрядностью и списком команд (однокристальные);
* с наращиваемой разрядностью (секционные) и микропрограммным управлением.

Анализируя адресные пространства программ и данных, определяют МП с архитектурой фон Неймана (память программ и память данных находятся в едином пространстве и нет никаких признаков, указывающих на тип информации в ячейке памяти) и МП с архитектурой Гарвардской лаборатории (память программ и память данных разделены, имеют свои адресные пространства и способы доступа к ним).

Для выполнения команд в структуру процессора входят внутренние регистры, *арифметико-логическое устройство* (*ALU* – *Arithmetic Logic Unit*), мультиплексоры, буферы, регистры и другие узлы. Работа всех узлов синхронизируется общим внешним тактовым сигналом процессора, т. е. процессор представляет собой довольно сложное цифровое устройство ([рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/#image.1.4)8).

Микропроцессор

Арифметическо-логическое устройство (АЛУ)

Схема управления выборкой команд

Схема управления прерываниями

RG

RG

**.**

**.**

**.**

Регистр признаков

Схема управления прямым доступом памяти (ПДП)

Логика

управления

RG

RG

Рис. 4.8. Пример структуры простейшего процессора

Впрочем, для разработчика *микропроцессорных систем* информация о тонкостях внутренней структуры процессора не слишком важна. Разработчик должен рассматривать процессор как «*черный ящик*», который в ответ на входные и *управляющие коды* производит ту или иную операцию и выдает выходные сигналы. Разработчику необходимо знать систему команд, режимы работы процессора, а также правила взаимодействия процессора с внешним миром или, как их еще называют, протоколы обмена информацией. О внутренней структуре процессора надо знать только то, что необходимо для выбора той или иной команды, того или иного режима работы.

**4.2. Обмен данными в микропроцессорной системе**

**4.2.1. Шинная структура связей**

Для достижения максимальной универсальности и упрощения *протоколов обмена* информацией в *микропроцессорных системах* применяется так называемая шинная структура связей между отдельными устройствами, входящими в систему. Суть шинной структуры связей сводится к следующему.

При классической структуре связей ([рис. 4.9](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.5)) все сигналы и коды между устройствами передаются по отдельным линиям связи. Каждое устройство, входящее в систему, передает свои сигналы и коды независимо от других устройств. При этом в системе получается очень много линий связи и разных *протоколов обмена* информацией.

Устройство 1

Устройство 3

Устройство 2

Устройство 4

Рис. 4.9. Классическая структура связей

При шинной структуре связей ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.6)0) все сигналы между устройствами передаются по одним и тем же линиям связи, но в разное время (это называется мультиплексированной передачей). Эти линии связи называются ***шинами*** (англ. *bus*). Причем передача по всем линиям связи может осуществляться в обоих направлениях (так называемая двунаправленная передача). В результате количество линий связи существенно сокращается, а правила обмена (протоколы) упрощаются.

Устройство 1

Устройство 2

Устройство 3

Устройство 4

Рис. 4.10. Шинная структура связей

Понятно, что при шинной структуре связей легко осуществляется пересылка всех информационных потоков в нужном направлении, например, их можно пропустить через один процессор, что очень важно для *микропроцессорной системы*. Однако при шинной структуре связей вся информация передается по линиям связи последовательно во времени, по очереди, что снижает быстродействие системы по сравнению с классической структурой связей.

Большое достоинство шинной структуры связей состоит в том, что все устройства, подключенные к *шине*, должны принимать и передавать информацию по одним и тем же правилам (протоколам обмена информацией по *шине*). Соответственно, все узлы, отвечающие за обмен с *шиной* в этих устройствах, должны быть единообразны, унифицированы.

Существенный недостаток шинной структуры связан с тем, что все устройства подключаются к каждой линии связи параллельно. Поэтому любая неисправность любого устройства может вывести из строя всю систему, если она портит линию связи. По этой же причине отладка системы с шинной структурой связей довольно сложна и обычно требует специального оборудования.

В системах с шинной структурой связей применяют все три существующие разновидности выходных каскадов цифровых микросхем:

— стандартный выход или выход с двумя состояниями (обозначается 2С, 2S, реже ТТЛ, TTL);

— выход с открытым коллектором (обозначается ОК, OC);

— выход с тремя состояниями или (что-то же самое) с возможностью отключения (обозначается 3С, 3S).

Упрощенно эти три типа выходных каскадов могут быть представлены в виде схем на [рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.7)11.

Выход 2С

Выход 3С

Выход ОК

Рис. 4.11. Три типа выходов цифровых микросхем

У выхода 2С два ключа замыкаются по очереди, что соответствует уровням логической «1» (верхний ключ замкнут) и логического «0» (нижний ключ замкнут). У выхода ОК замкнутый ключ формирует уровень логического «0», разомкнутый – логической «1». У выхода 3С ключи могут замыкаться по очереди (как в случае 2С), а могут размыкаться одновременно, образуя третье, высокоимпедансное состояние. Переход в третье состояние (Z-состояние) управляется сигналом на специальном входе EZ.

Выходные каскады типов 3С и ОК позволяют объединять несколько выходов микросхем для получения мультиплексированных (рис. 4.12) или двунаправленных (рис. 4.13) линий.

◊

Выход

Вход

◊

Выход

◊

Выход

Рис. 4.12. Мультиплексированная линия

◊

Вход Выход

Вход

◊

Выход

◊

Вход Выход

Рис. 4.13. Двунаправленная линия

При этом в случае выходов 3С необходимо обеспечить, чтобы на линии всегда работал только один активный выход, а все остальные выходы находились бы в это время в третьем состоянии, иначе возможны конфликты. Объединенные выходы ОК могут работать все одновременно, без всяких конфликтов.

Типичная структура *микропроцессорной системы* приведена на [рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/2.html#image.1.10)4. Она включает в себя три основных типа устройств:

* процессор;
* *память*, включающую оперативную ***память*** (ОЗУ, *Random Access Memory*) и постоянную *память*(ПЗУ, *Read Only Memory*), которая служит для хранения данных и программ;
* ***устройства ввода/вывода*** (УВВ, I/O – *Input*/*Output Devices*), служащие для связи *микропроцессорной системы* с внешними устройствами, для приема (ввода, чтения, *Read*) входных сигналов и выдачи (вывода, записи, *Write*) выходных сигналов.

Выходные

сигналы

Входные

сигналы

Устройство ввода/вывода

Память

Процессор

Шина

питания

Системная шина

(магистраль)

Шина данных

Шина

управления

Шина адреса

Рис. 4.14. Структура микропроцессорной системы

Все устройства *микропроцессорной системы* объединяются общей системной ***шиной*** (она же называется еще **системной магистралью** или **каналом**). Системная *магистраль* включает в себя четыре основные *шины* нижнего уровня:

*— шина* *адреса* (*Address Bus*);

*— шина данных* (*Data Bus*);

*— шина* *управления* (*Control Bus*);

*— шина* *питания* (*Power Bus*).

*Шина* *адреса* служит для определения адреса (номера) устройства, с которым процессор обменивается информацией в данный момент. Каждому устройству (кроме процессора), каждой ячейке *памяти* в *микропроцессорной системе* присваивается собственный адрес. Когда код какого-то адреса выставляется процессором на шине адреса, устройство, которому этот адрес приписан, понимает, что ему предстоит обмен информацией. Шина адреса может быть однонаправленной или двунаправленной.

*Шина* *данных* – это основная шина, которая используется для передачи информационных кодов между всеми устройствами *микропроцессорной системы*. Обычно в пересылке информации участвует процессор, который передает код данных в какое-то устройство или в ячейку *памяти* или же принимает код данных из какого-то устройства или из ячейки *памяти*. Но возможна также и передача информации между устройствами без участия процессора. Шина данных всегда двунаправленная.

*Шина* *управления* в отличие от шины адреса и шины данных состоит из отдельных управляющих сигналов. Каждый из этих сигналов во время обмена информацией имеет свою функцию. Некоторые сигналы служат для стробирования передаваемых или принимаемых данных (т. е. определяют моменты времени, когда информационный код выставлен на шину данных). Другие управляющие сигналы могут использоваться для подтверждения приема данных, для сброса всех устройств в исходное состояние, для тактирования всех устройств и т. д. Линии шины управления могут быть однонаправленными или двунаправленными.

Наконец, *шина* *питания* предназначена не для пересылки информационных сигналов, а для питания системы. Она состоит из линий питания и общего провода. В *микропроцессорной системе* может быть один источник питания (чаще +5 В) или несколько источников питания (обычно еще –5 В, +12 В и –12 В). Каждому напряжению питания соответствует своя линия связи. Все устройства подключены к этим линиям параллельно.

Если в *микропроцессорную систему* надо ввести входной код (или входной сигнал), то процессор по шине адреса обращается к нужному *устройству ввода/вывода* и принимает по *шине* данных *входную информацию*. Если из *микропроцессорной системы* надо вывести выходной код (или выходной сигнал), то процессор обращается по шине адреса к нужному *устройству ввода/вывода* и передает ему по шине данных выходную информацию.

Если информация должна пройти сложную многоступенчатую обработку, то процессор может хранить промежуточные результаты в системной оперативной памяти. Для обращения к любой ячейке *памяти* процессор выставляет ее адрес на шину адреса и передает в нее информационный код по шине данных или же принимает из нее информационный код по шине данных. В *памяти* (оперативной и постоянной) находятся также и *управляющие коды* (команды выполняемой процессором программы), которые процессор также читает по шине данных с адресацией по шине адреса. Постоянная память используется в основном для хранения программы начального пуска *микропроцессорной системы*, которая выполняется каждый раз после включения питания. Информация в нее заносится изготовителем раз и навсегда.

Таким образом, в *микропроцессорной системе* все информационные коды и коды команд передаются по *шинам* последовательно, по очереди. Это определяет сравнительно невысокое быстродействие *микропроцессорной системы*. Оно ограничено обычно даже не быстродействием процессора (которое тоже очень важно) и не скоростью обмена по системной шине (магистрали), а именно последовательным характером передачи информации по системной шине (магистрали).

Важно учитывать, что *устройства ввода/вывода* чаще всего представляют собой устройства на «жесткой логике». На них может быть возложена часть функций, выполняемых *микропроцессорной системой*. Поэтому у разработчика всегда имеется возможность перераспределять функции системы между аппаратной и программной реализациями оптимальным образом. Аппаратная реализация ускоряет выполнение функции, но имеет недостаточную гибкость. Программная реализация значительно медленнее, но обеспечивает высокую гибкость. Аппаратная реализация функций увеличивает стоимость системы и ее *энергопотребление*, программная – не увеличивает. Чаще всего применяется комбинирование аппаратных и программных функций.

Иногда *устройства ввода/вывода* имеют в своем составе процессор, т. е. представляют собой небольшую специализированную *микропроцессорную систему*. Это позволяет переложить часть программных функций на *устройства ввода/вывода*, разгрузив центральный процессор системы.

**4.2.2. Режимы работы микропроцессорной системы**

Как уже отмечалось, *микропроцессорная система* обеспечивает большую гибкость работы, она способна настраиваться на любую задачу. Гибкость эта обусловлена прежде всего тем, что функции, выполняемые системой, определяются программой (программным обеспечением, software), которую выполняет процессор. Аппаратура (аппаратное обеспечение, hardware) остается неизменной при любой задаче. Записывая в *память* системы программу, можно заставить *микропроцессорную систему* выполнять любую задачу, поддерживаемую данной аппаратурой. К тому же шинная *организация связей микропроцессорной системы* позволяет довольно легко заменять аппаратные модули, например, заменять *память* на новую большего объема или более высокого быстродействия, добавлять или модернизировать *устройства ввода/вывода*, наконец, заменять процессор на более мощный. Это также позволяет увеличить гибкость системы, продлить ее жизнь при любом изменении требований к ней.

Но гибкость *микропроцессорной системы* определяется не только этим. Настраиваться на задачу помогает еще и выбор режима работы системы, то есть режима обмена информацией по системной магистрали (*шине*).

Практически любая развитая *микропроцессорная система* (в том числе и компьютер) поддерживает три основных режима обмена по магистрали:

— программный обмен информацией;

— обмен с использованием прерываний (*Interrupts*);

— обмен с использованием прямого доступа к *памяти* (ПДП, DMA – *Direct Memory Access*).

**Программный обмен информацией** является основным в любой *микропроцессорной системе*. Он предусмотрен всегда, без него невозможны другие режимы обмена. В этом режиме процессор является единоличным хозяином (или задатчиком, Master) системной магистрали. Все операции (циклы) обмена информацией в данном случае инициируются только процессором, все они выполняются строго в порядке, предписанном исполняемой программой.

Процессор читает (выбирает) из *памяти* коды команд и исполняет их, читая данные из *памяти* или из *устройства ввода/вывода*, обрабатывая их, записывая данные в *память* или передавая их в *устройство ввода/вывода*. Путь процессора по программе может быть линейным, циклическим, может содержать переходы (прыжки), но он всегда непрерывен и полностью находится под контролем процессора. Ни на какие внешние события, не связанные с программой, процессор не реагирует (рис. 4.15). Все сигналы на магистрали в данном случае контролируются процессором.

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N+1

Команда N+2

Команда N+3

Команда N+4

●

●

●

Выполнение команд

Программа

Последовательно

Цикл

Прыжок

**Рис. 4.15.** Программный обмен информацией

**Обмен по прерываниям** используется тогда, когда необходима реакция *микропроцессорной системы* на какое-то внешнее событие, на приход внешнего сигнала. В случае компьютера внешним событием может быть, например, нажатие на клавишу клавиатуры или приход по локальной сети пакета данных. Компьютер должен реагировать на это соответственно выводом символа на экран или же чтением и обработкой принятого по сети пакета.

В общем случае организовать реакцию на внешнее событие можно тремя различными путями:

— с помощью постоянного программного контроля факта наступления события (так называемый метод опроса флага или *polling*);

— с помощью прерывания, т. е. насильственного перевода процессора с выполнения текущей программы на выполнение экстренно необходимой программы;

— с помощью прямого доступа к *памяти*, т. е. без участия процессора при его отключении от системной магистрали.

Проиллюстрировать эти три способа можно следующим простым примером. Допустим, вы готовите себе завтрак, поставив на плиту кипятиться молоко. Естественно, на закипание молока надо реагировать, причем срочно. Как это организовать? Первый путь – постоянно следить за молоком, но тогда вы ничего другого не сможете делать. Правильнее будет регулярно поглядывать на молоко, делая одновременно что-то другое. Это программный режим с опросом флага. Второй путь – установить на кастрюлю с молоком датчик, который подаст звуковой сигнал при закипании молока, и спокойно заниматься другими делами. Услышав сигнал, вы выключите молоко. Правда, возможно, вам придется сначала закончить то, что вы начали делать, так что ваша реакция будет медленнее, чем в первом случае. Наконец, третий путь состоит в том, чтобы соединить датчик на кастрюле с управлением плитой так, чтобы при закипании молока горелка была выключена без вашего участия (правда, аналогия с ПДП здесь не очень точная, так как в данном случае на момент выполнения действия вас не отвлекают от работы).

Первый случай с опросом флага реализуется в *микропроцессорной системе* постоянным чтением информации процессором из *устройства ввода/вывода*, связанного с тем внешним устройством, на поведение которого необходимо срочно реагировать.

Во втором случае в режиме прерывания процессор, получив запрос прерывания от внешнего устройства (часто называемый *IRQ* — *Interrupt ReQuest*), заканчивает выполнение текущей команды и переходит к программе обработки прерывания. Закончив выполнение программы обработки прерывания, он возвращается к прерванной программе с той точки, где его прервали ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.12)6).

Здесь важно то, что вся работа, как и в случае программного режима, осуществляется самим процессором, внешнее событие просто временно отвлекает его. Реакция на внешнее событие по прерыванию в общем случае медленнее, чем при программном режиме. Как и в случае программного обмена, здесь все сигналы на магистрали выставляются процессором, т. е. он полностью контролирует магистраль. Для обслуживания прерываний в систему иногда вводится специальный модуль *контроллера прерываний*, но он в обмене информацией не участвует. Его задача состоит в том, чтобы упростить работу процессора с внешними запросами прерываний. Этот контроллер обычно программно управляется процессором по системной магистрали.

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N+1

Команда N+2

●

●

●

Команда 1

Команда 2

Команда 3

●

●

●

Команда M

Программа обработки прерывания

Запрос прерывания

Выполнение команд

Выполнение команд

Основная программа

Рис. 4.16. Обслуживание прерывания

Естественно, никакого ускорения работы системы прерывание не дает. Его применение позволяет только отказаться от постоянного опроса флага внешнего события и временно, до наступления внешнего события, занять процессор выполнением каких-то других задач.

**Прямой доступ к памяти (ПДП, DMA)** – это режим, принципиально отличающийся от двух ранее рассмотренных режимов тем, что обмен по системной шине идет без участия процессора. Внешнее устройство, требующее обслуживания, сигнализирует процессору, что режим ПДП необходим, в ответ на это процессор заканчивает выполнение текущей команды и отключается от всех шин, сигнализируя запросившему устройству, что обмен в режиме ПДП можно начинать.

Операция ПДП сводится к пересылке информации из *устройства ввода/вывода* в *память* или же из *памяти* в *устройство ввода/вывода*. Когда пересылка информации будет закончена, процессор вновь возвращается к прерванной программе, продолжая ее с той точки, где его прервали ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.13)7). Это похоже на режим обслуживания прерываний, но в данном случае процессор не участвует в обмене. Как и в случае прерываний, реакция на внешнее событие при ПДП существенно медленнее, чем при программном режиме.

Понятно, что в этом случае требуется введение в систему дополнительного устройства (контроллера ПДП), которое будет осуществлять полноценный обмен по системной магистрали без всякого участия процессора. Причем процессор предварительно должен сообщить этому контроллеру ПДП, откуда ему следует брать информацию и/или куда ее следует помещать. Контроллер ПДП может считаться специализированным процессором, который отличается тем, что сам не участвует в обмене, не принимает в себя информацию и не выдает ее ([рис. 4.1](http://www.intuit.ru/department/hardware/mpbasics/1/3.html#image.1.14)8).

Рис. 4.17. Обслуживание ПДП

Команда 1

Команда 2

Команда 3

●

●

●

Команда N

Команда N + 1

Команда N + 2

Команда N + 3

Команда N + 4

●

●

●

Выполнение команд

Запрос ПДП

Точка остановки на время ПДП

Продолжение

выполнения команд

Программа

Процессор

Контроллер

ПДП

Память

Устройство

ввода/вывода

Системная магистраль

Рис. 4.18**.** Информационные потоки в режиме ПДП

Контроллер ПДП может входить в состав *устройства ввода/вывода*, которому необходим режим ПДП или даже в состав нескольких *устройств ввода/вывода*. Теоретически обмен с помощью прямого доступа к *памяти* может обеспечить более высокую скорость передачи информации, чем программный обмен, так как процессор передает данные медленнее, чем специализированный контроллер ПДП. Однако на практике это преимущество реализуется далеко не всегда. Скорость обмена в режиме ПДП обычно ограничена возможностями магистрали. К тому же необходимость программного задания режимов контроллера ПДП может свести на нет выигрыш от более высокой скорости пересылки данных в режиме ПДП. Поэтому режим ПДП применяется редко.

Если в системе уже имеется самостоятельный контроллер ПДП, то это может в ряде случаев существенно упростить аппаратуру *устройств ввода/вывода*, работающих в режиме ПДП. В этом, пожалуй, состоит единственное бесспорное преимущество режима ПДП.

**4.2.3 Архитектура микропроцессорных систем**

До сих пор мы рассматривали только один тип *архитектуры* *микропроцессорных систем* – *архитектуру* с общей, единой *шиной* для данных и команд (одношинную, или **принстонскую**, **фон-неймановскую** *архитектуру* ). Соответственно, в составе системы в этом случае присутствует одна общая *память* как для данных, так и для команд ([рис. 4.](http://www.intuit.ru/department/hardware/mpbasics/1/4.html#image.1.15)19).

Процессор

Устройства

ввода/вывода

Память

Команды

Данные

Общая шина данных и команд

Рис. 4.19**.** Архитектура с общей шиной данных и команд

Но существует также и альтернативный тип *архитектуры микропроцессорной системы* – это *архитектура* с раздельными *шинами* данных и команд (двухшинная, или гарвардская, *архитектура*). Эта *архитектура* предполагает наличие в системе отдельной *памяти* для данных и отдельной *памяти* для команд ([рис.](http://www.intuit.ru/department/hardware/mpbasics/1/4.html#image.1.16) 4.19). Обмен процессора с каждым из двух типов *памяти* происходит по своей шине.

*Архитектура* с общей шиной распространена гораздо больше, она применяется, например, в персональных компьютерах и в сложных *микрокомпьютерах*. *Архитектура* с раздельными шинами применяется в основном в однокристальных *микроконтроллерах*.

Рассмотрим некоторые достоинства и недостатки обоих архитектурных решений.

*Архитектура* с общей шиной (принстонская, фон-неймановская) проще, она не требует от процессора одновременного обслуживания двух *шин*, контроля обмена по двум шинам сразу. Наличие единой *памяти* данных и команд позволяет гибко распределять ее объем между кодами данных и команд. Например, в некоторых случаях нужна большая и сложная программа, а данных в *памяти* надо хранить не слишком много. В других случаях, наоборот, программа требуется простая, но необходимы большие объемы хранимых данных. Перераспределение *памяти* не вызывает никаких проблем, главное – чтобы программа и данные вместе помещались в *памяти* системы. Как правило, в системах с такой *архитектурой память* бывает довольно большого объема (до десятков и сотен мегабайт). Это позволяет решать самые сложные задачи.

Шина команд

Память команд

Процессор

Устройства

ввода/вывода

Память данных

Шина данных

Рис. 4.20**.** Архитектура с раздельными шинами данных и команд

*Архитектура* с раздельными шинами данных и команд сложнее, она заставляет процессор работать одновременно с двумя потоками кодов, обслуживать обмен по двум шинам одновременно (рис. 4.20). Программа может размещаться только в *памяти* команд, данные — только в *памяти* данных. Такая узкая специализация ограничивает круг задач, решаемых системой, так как не дает возможности гибкого перераспределения памяти. *Память* данных и *память* команд в этом случае имеют не слишком большой объем, поэтому применение систем с данной *архитектурой* ограничивается обычно не слишком сложными задачами.

В чем же преимущество *архитектуры* с двумя шинами (гарвардской)? В первую очередь, в быстродействии. Дело в том, что при единственной шине команд и данных процессор вынужден по одной этой шине принимать данные (из *памяти* или *устройства ввода/вывода*) и передавать данные (в *память* или в *устройство ввода/вывода*), а также читать команды из *памяти*. Естественно, одновременно эти пересылки кодов по магистрали происходить не могут, они должны производиться по очереди. Современные процессоры способны совместить во времени выполнение команд и проведение циклов обмена по системной шине. Использование конвейерных технологий и быстрой кэш-памяти позволяет им ускорить процесс взаимодействия со сравнительно медленной системной *памятью*. Повышение тактовой частоты и совершенствование структуры процессоров дают возможность сократить время выполнения команд. Но дальнейшее увеличение быстродействия системы возможно только при совмещении пересылки данных и чтения команд, т. е. при переходе к *архитектуре* с двумя шинами.

В случае двухшинной *архитектуры* обмен по обеим *шинам* может быть независимым, параллельным во времени. Соответственно, структуры шин (количество разрядов кода адреса и кода данных, порядок и скорость обмена информацией и т. д.) могут быть выбраны оптимально для той задачи, которая решается каждой шиной. Поэтому при прочих равных условиях переход на двухшинную *архитектуру* ускоряет работу *микропроцессорной системы*, хотя и требует дополнительных затрат на аппаратуру, усложнения структуры процессора. *Память* данных в этом случае имеет свое распределение адресов, а *память* команд – свое.

Проще всего преимущества двухшинной *архитектуры* реализуются внутри одной микросхемы. В этом случае можно также существенно уменьшить влияние недостатков этой *архитектуры*. Поэтому основное ее применение – в *микроконтроллерах*, от которых не требуется решения слишком сложных задач, но зато необходимо максимальное быстродействие при заданной тактовой частоте.

4.2.4. Архитектура 16-тиразрядного микропроцессора

Мы рассмотрим более подробно основные типы архитектурных решений, выделяя связь со способами адресации памяти. Регистровая архитектура определяется наличием достаточно большого регистрового файла внутри МП. Команды получают возможность обратиться к операндам, расположенным в одной из двух запоминающих сред: оперативной памяти или регистрах. Размер регистра обычно фиксирован и совпадает с размером слова, физически реализованного в оперативной памяти. К любому регистру можно обратиться непосредственно, поскольку регистры представлены в виде массива запоминающих элементов – регистрового файла. Типичным является выполнение арифметических операций только в регистре, при этом команда содержит два операнда (оба операнда в регистре или один операнд в регистре, а второй – в оперативной памяти).

4.1. Структура микропроцессора Intel8080

Intel 8080 является однокристальным микропроцессором, работающим с 8-разрядной шиной данных и 16-разрядной шиной адреса. Управляющие сигналы передаются по шине управления. Шины отделены друг от друга. Структура Intel 8080 приведена на рис. 4.21.

Буфер данных

Аккумулятор

Буфер аккумулятора

Арифметико-

Логическое устройство

Регистр

флагов

16

Дешифратор выбора регистров

Схема инкремента

JP(16)

SP(16)

B(8) C(8)

D(8) E(8)

H(8) L(8)

W(8) Z(8)

Буфер адреса

Внутренняя шина данных

Дешифратор

команд

Регистр

команд

Intel 8080

Буфер данных

8

8

Схема десятичной коррекции

Аккумулятор

Буфер аккумулятора

Арифметико-

логическое устройство

Регистр

флагов

8

TR

RC

Устройство

управления

Управление обменом информацией

Рис. 4.21. Внутренняя структура микропроцессора Intel 8080

Микропроцессор содержит внутреннюю шину данных, посредством которой происходит обмен информацией между внутренними регистрами, арифметико-логическим устройством, обрабатывающим 8-разрядные данные и передающим их через буфер на внешнюю шину данных. Кроме того, в состав Intel 8080 входит устройство управления, буфер адресной шины, связанный с регистром команд, и блок регистров. В общих чертах работа микропроцессора выглядит следующим образом: в регистре, называемом программным счетчиком, хранится адрес следующей команды, которую необходимо выполнить. Устройство управления подключает этот регистр к шине адреса (конечно, через буфер) и выдает управляющие сигналы, необходимые для чтения кода команды из памяти. На этом завершается первый такт.

В следующем такте микропроцессор проверяет состояние сигналов на входе готовности и запрос останова. При их наличии микропроцессор переходит в соответствующее состояние. В противном случае после появления на шине управления сигналов, подтверждающих выдачу кода команды на шину данных, устройство управления подключает к ней регистр команды и записывает в нее полученный код. Это требуется потому, что команда передается только в первом машинном цикле, а сохранить ее нужно на все время выполнения команды. Из регистра команды ее код поступает в дешифратор команды и затем в устройство управления, которое в зависимости от поступившей команды либо сразу переходит к ее выполнению, либо считывает данные или адрес, расположенные сразу после кода команды и необходимые для ее выполнения. На это тратится третий такт и, если это необходимо, четвертый и пятый такты. Таким образом, вся команда выполняется за 3–5 тактов. При тактовой частоте 2 МГц это составляет 1,5–2,5 мкс.

Перед выполнением команды проверяется состояние сигнала на входе захвата шины HLD (этот сигнал отключает микропроцессор от шины, давая внешним устройствам возможность прямого доступа в память по общей шине). При его обнаружении микропроцессор переходит в состояние захвата и вырабатывает сигнал подтверждения захвата шины. После снятия сигнала HLD процессор продолжает выполнение команды. В конце машинного цикла вновь анализируется состояние входа захвата, а затем проверяется, завершено ли выполнение команды. Если нет, то микропроцессор переходит к выполнению следующего цикла команды. Это может быть цикл работы с оперативной памятью или с внешним устройством.

После выполнения каждой команды проверяется состояние входа запроса прерывания. Если этот сигнал присутствует, то текущая программа приостанавливается и на шину данных выдается сигнал «подтверждение прерывания». Затем внешний контроллер прерываний передает по шине данных команду и адрес перехода к подпрограмме обработки прерывания. По окончании обработки прерывания происходит возврат к выполнению прерванной программы.

Сигнал готовности позволяет синхронизировать работу микропроцессора с работой более медленных устройств, и, кроме того, используется для пошагового выполнения программ.

Следующим важным узлом микропроцессора является блок регистров. Он включает в себя 16-разрядный регистр для временного хранения данных WZ, шесть 8-разрядных регистров общего назначения B, C, D, E, H, L, которые могут использоваться парами в качестве 16-разрядных – BC, DE, HL (это сделано прежде всего для удобной работы с адресами). Кроме того, блок регистров содержит 16-разрядный регистр адреса команды IP (программный счетчик), 16-разрядный регистр указателя стека SP, а также 16-разрядную схему инкремента-декремента. С помощью последней изменяется, например, состояние программного счетчика после выполнения каждой следующей команды.

Еще один важный узел – регистр результата (аккумулятор), связанный с АЛУ и используемый для хранения одного из исходных операндов или результата выполнения команды.

Последний регистр – это регистр флажков. В нем записан байт, каждый бит которого содержит информацию о результате выполнения последней команды.

4.3. Архитектура микропроцессора 8086

Микропроцессор Intel 8086 существенно отличается от i8080. В нем применена новая, значительно более мощная и гибкая система команд, есть возможность адресации 1 Мбайта памяти, обращения к 65536 устройствам ввода и такому же количеству устройств вывода информации.

В Intel 8086 имеется возможность изменения внутренней аппаратной конфигурации с помощью специального управляющего сигнала. В более простом режиме 8086 ориентирован на использование в простых вычислительных и управляющих устройствах. При этом микропроцессор сам вырабатывает сигналы управления шиной и обеспечивает прямой доступ к ней посредством контроллера Intel 8257. В режиме полной конфигурации обеспечивается работа с контроллером шины Intel 8288, который декодирует три сигнала состояния процессора и в зависимости от них выдает семь сигналов управления шиной. Такой режим используется в мультипроцессорных системах и в сложных вычислительных устройствах, в частности в компьютере IBM PC/XT.

Интересно организована память: хранение 16-разрядных слов осуществляется в виде отдельных байтов, причем байты, передающиеся по восьми младшим линиям шины данных (D7-D0), собраны в банк 0, а передаваемые по восьми старшим линиям – в банк 1. Объем каждого банка составляет 512 Кбайт. Таким образом, нечетные байты хранятся в банке 1, а четные – в банке 0. Выбор банка осуществляется с помощью младшего адреса и сигнала управления старшими разрядами шины данных.

Еще одна важная особенность – возможность обработки 256 типов прерываний (от 0 до 255), в том числе есть прерывания, определяемые пользователем, и пошаговые прерывания.

Микропроцессор Intel 8086 приспособлен для работы с несколькими процессорами в одной системе, причем возможно использование как независимых процессоров, так и сопроцессоров. Отличие заключается в том, что независимый процессор выполняет свою собственную последовательность команд, а сопроцессор следит за потоком команд центрального процессора и выделяет из него «свои» команды, расширяя набор команд основного процессора и улучшая таким образом характеристики системы. Для поддержки этих режимов используются команды *ESC*, *LOCK* и *XCHG*, а также специальные управляющие сигналы, позволяющие разрешать конфликты доступа к общим ресурсам.

Внешние шины адреса и данных в Intel 8086 объединены, и поэтому наличие на шине в данный момент времени информации или адреса определяется порядковым номером такта внутри цикла. Процессор ориентирован на параллельное выполнение команды и выборки следующей команды. В целом выполнение команды происходит примерно так же, как и в Intel 8080. Команда выбирается из памяти и принимается микропроцессором в свободный регистр очереди команд, причем в то же самое время выполняется предыдущая команда. Конвейеризация команд позволяет значительно повысить быстродействие системы. При выполнении команд проверяются состояния входов запросов прерываний и захвата шины, и при необходимости выполняются соответствующие действия.

Микропроцессор Intel 8086 состоит из трех основных частей: устройства сопряжения шины, устройства обработки и устройства управления и синхронизации.

Устройство сопряжения шины состоит из шести 8-разрядных регистров очереди команд, четырех 16-разрядных регистров адреса команды, 16-разрядного регистра команды и 16-разрядного сумматора адреса (рис. 4.22). Оно выполняет следующие функции: выбирает команды из памяти и записывает их в регистр очереди команд, вычисляет и формирует физический адрес, читает операнды из памяти или из регистров и записывает результат выполнения команд в память или в регистры.

А0/D0-A15/D15

16

А16-А19

4

Устройство управления и синхронизации

Буфер шины адреса/данных

Внутренняя шина команд

Внутренняя шина

Сумматор адреса

CS

DS

SS

ES

IP

Регистр обмена

Регистры очереди команд (8-разрядные)

1

2

3

4

5

6

16

16

Внутренняя шина данных

Микропрограммное устройство управления

Арифметико-

логическое устройство

Регистр флагов

AH AL

BH BL

CH CL

DH DL

SP

BP

SI

DI

A

B

C

D

Рис. 4.22. Внутренняя структура микропроцессора Intel 8086

Устройство обработки преобразует данные. Команда из очереди команд по запросу устройства обработки поступает на внутреннюю шину команд, а с нее на микропрограммное устройство управления, декодирующее ее и генерирующее соответствующие последовательности микрокоманд, необходимые для выполнения текущей операции. В отличие от первых микропроцессоров, устройство обработки в Intel 8086 не связано с внешней шиной, а обменивается с ней информацией через регистр обмена устройства сопряжения шины.

Устройство обработки содержит 16-разрядное арифметико-логическое устройство, восемь 16-разрядных регистров общего назначения и 16-разрядный регистр флагов. Регистры могут использоваться как 16-разрядные или как пары 8-разрядных (при этом их количество удваивается).

**4.3. Архитектура микропроцессора Intel80286**

Кристалл Intel 80286 представляет для читателя интерес прежде всего потому, что является, пожалуй, наиболее распространенным микропроцессором из применяющихся в персональных компьютерах. Как и его предшественник – Intel 8086 – он имеет 16-разрядные шины данных и адреса и самым характерным его отличием можно считать, помимо большей тактовой частоты, возможность работы в режиме виртуальной адресации (адресация памяти объемом более 1 Мбайта), речь о котором пойдет ниже.

Как и любой процессор, Intel 80286 содержит некоторое количество ячеек памяти быстрого доступа, называемых регистрами. В состав Intel 80286 входят три набора по четыре регистра и один специальный регистр – указатель команды.

***Регистры общего* назначения.** Первый набор включает в себя регистры общего назначения, или РОН, необходимые для временного хранения тех операндов и результатов вычислений, доступ к которым постоянно повторяется в процессе выполнения программы. Использование РОН в подобных случаях существенно ускоряет работу системы за счет сокращения времени чтения/записи и пересылки данных из ОЗУ. Всего регистров общего назначения четыре, они, разумеется, 16-разрядные, но могут использоваться и как 8-разрядные (однобайтные), при этом их количество удваивается.

Функции всех РОН, в основном, идентичны, но в некоторых случаях архитектура предполагает их строгую специализацию. Например, при выполнении команд обработки строк и циклов в одном из регистров должно храниться число, равное количеству итераций. Этот регистр выполняет роль счетчика (counter) и носит название CX. Остальные регистры выполняют функции аккумулятора (AX), базы (BX) и ячейки временного хранения данных (DX). Как мы уже знаем, каждый регистр из числа РОН может быть разделен на два однобайтных, один из которых (0–7) называется младшим (*Low*), а другой (8–15) – старшим (*High*). В соответствии с этим каждый 8-разрядный регистр получил свое название: младшие именуются AL, BL, CL, DL, а старшие – AН, BН, CН и DH (рис. 4.23).

AH AL

BH BL

CH CL

DH DL

AX

BX

CX

DX

Аккумулятор

База

Счетчик

Данные

15

7

8

0

Рис. 4.23. Регистры общего назначения

Перед тем как познакомиться с назначением и функциями остальных наборов регистров, разберемся, каким образом процессору с 16-разрядной шиной адреса удается работать с памятью объемом в 1 Мбайт.

***Режим реального адреса***. Адресная шина процессора Intel 80286 имеет ширину 16 бит, к тому же известно, что максимальное двоичное число длиной в два байта равно 216, или 64 Кбайт и если адрес задается таким числом, то, вроде бы, пространство ОЗУ, с которым может работать процессор, не должно превышать 64 Кбайт. С другой стороны, 1 Мбайт памяти можно адресовать с помощью двоичного числа длиной 20 бит (220). Как быть?

Можно, например, воспользоваться двойным адресом, ведь в повседневной жизни нам приходится постоянно сталкиваться с многоступенчатыми адресами: мы пишем на почтовом конверте сначала название города, потом – улицы, дома и т. д. (Предположим на мгновенье, что все квартиры в Беларуси пронумерованы последовательно, каково придется почте в такой ситуации?) Разработчики Intel 80286 решили проблему подобным же образом: полный адрес ячейки памяти состоит из комбинации двух 16-разрядных чисел, причем одно из них предназначили для адресации внутри некоторой области ОЗУ размером 64 Кбайта, а второе – для локализации этой области во всем пространстве ОЗУ. Область, внутри которой происходит адресация, называется сегментом, а адрес внутри сегмента – внутрисегментным смещением. Адрес, локализующий положение сегмента в оперативной памяти, содержится в одном из специальных сегментных регистров процессора, но он тоже 16-разрядный. Для того чтобы при помощи этого адреса можно было перекрыть все пространство ОЗУ, со стороны младшего байта его дополняют четырьмя нулями. Например, если содержимое сегментного регистра: 0001.1101.1000.1111 (или 1D8F16), то адрес начала соответствующего сегмента будет равен: 0001.1101.1000.1111.0000 (или 1D8F016). Таким образом можно искусственно разделить всю память на сегменты, начинающиеся по адресам, кратным 1610. Предположим, что внутрисегментное смещение нашей ячейки задано числом 1001.1011.0010.0101 (или 9В2516), в этом случае ее реальный адрес будет равен сумме адреса сегмента и внутрисегментного смещения: 1D8F016 + 9B2516 = 2701516. Схема такого формирования реального адреса будет рассмотрена в следующем пункте.

Выполняемая программа может обращаться к любому из четырех сегментов, именуемых текущим сегментом кода (т. е. программы), текущим сегментом данных, текущим сегментом стека и текущим дополнительным сегментом.

Теперь вернемся к регистрам.

***Указательные и индексные регистры.*** Второй набор состоит из четырех 16-разрядных регистров, которые, в основном, используются для хранения внутрисегментных смещений. При выполнении многих команд функции каждого из данных регистров строго определены.

Регистры SI (индекс источника) и DI (индекс приемника) называются указательными и содержат смещения в текущем сегменте данных. Регистры SP (указатель стека) и ВР (указатель базы) называются базовыми и содержат смещения в текущем сегменте стека. Для тех, кто забыл или не знает, следует напомнить, что стек – это способ организации работы с оперативной памятью по принципу LIFO (LastIn-FirstOut), что в переводе означает: «последним вошел – первым вышел», т. е. слово данных, помещенное в стек последним, будет извлечено оттуда в первую очередь. Само слово стек произошло от английского *stack*–скирда. И действительно, область памяти, организованную в виде стека, можно сравнить со скирдой сена: последние, уложенные сверху снопы, будут использованы первыми – никому не придет в голову выгребать сено из середины. Для работы со стеком необходимо знать две величины: адрес дна и адрес вершины стека. Если адрес дна – число фиксированное, то адрес вершины зависит от того, сколько байтов данных содержится в стеке. В нашем случае адрес вершины находится в регистре SP.

Регистры SP и BP содержат указатели стека и базы соответственно, регистры SI и DI – смещения соответственно источника и приемника при выполнении команд обработки строк *MOVSB*, *MOVSW*, *LODSB*, *LODSW*, *STOSB*, *STOSW* и команды *LOOP*.

***Сегментные регистры***

Если мы вспомним, что программа в любой момент может обратиться к одному из четырех сегментов: к текущему сегменту кода, данных, стека или к дополнительному (сегменту данных), то нас вряд ли удивит, что в состав процессора входят четыре 16-разрядных регистра, являющихся указателями адресов текущих сегментов. Их функции строго дифференцированы, а потому каждый регистр имеет свою «профессию»: CS определяет сегмент кода, DS – сегмент данных, SS – сегмент стека и ES – дополнительный сегмент.

Теперь для того чтобы, к примеру, произвести выборку слова данных из стека, программе достаточно обратиться к регистрам SS и SP, сложить находящиеся в них числа по уже известному нам правилу и в качестве результата получить реальный адрес вершины стека.

***Флажки***

Совокупность значений флажков и определяет состояние процессора во время его работы. В самом общем случае слово состояния – это двоичное число, каждый бит которого отражает строго определенный параметр состояния устройства. Что касается Intel 80286, то здесь биты слова состояния называются флажками, всего их девять, причем шесть из них регистрируют состояние процессора, а три применяются для управления его работой.

Верхний регистр IP является указателем команды, нижний – регистром флажков.

C

T

S

NT

IOPL

O

D

I

Z

A

P

IP

Флажки

Указатель команды

15

0

Рис.4.24. Указатель команды и флажки

Верхний регистр IP является указателем команды, нижний – регистром флажков.К флажкам состояния относятся: флажок переноса CF (имеет значение равное 1 при переносе из старшего бита), флажок вспомогательного переноса AF (индицирует перенос из младших 4-х бит), флажок переполнения OF (устанавливается равным единице при выходе знакового результата за границу диапазона), флажок нуля ZF (фиксирует нулевой результат выполнения команды), флажок знака SF (фиксирует отрицательный результат выполнения команды) флажок четности PF (фиксирует четное число единиц в последнем байте, полученном в результате выполнения команды).

К флажкам управления относятся: флажок направления DF (указывает направление прохождения строк в строковых командах), флажок разрешения прерывания IF (разрешает или запрещает прерывание по входу INTR), флажок трассировки TF (переводит процессор в пошаговый режим).

Размещение флажков в разрядах регистра слова состояния будет также рассмотрено в следующей лекции.

5. УРОВЕНЬ АРХИТЕКТУРЫ НАБОРА КОМАНД

5.1. Система команд

5.1.1. Классификация команд

Проектирование системы команд оказывает влияние на структуру ЭВМ. Оптимальную систему команд иногда определяют как совокупность команд, которая удовлетворяет требованиям проблемно-ориентированных применений таким образом, что избыточность аппаратных и аппаратно-программных средств на реализацию редко используемых команд оказывается минимальной. В различных программах ЭВМ частота появления команд различна; например, по данным фирмы DEC в программах для ЭВМ семейства PDP-11 наиболее часто встречается команда передачи MOV(B), на ее долю приходится приблизительно 32% всех команд в типичных программах. Систему команд следует выбирать таким образом, чтобы затраты на редко используемые команды были минимальными.

При наличии статистических данных можно разработать (выбрать) ЭВМ с эффективной системой команд. Одним из подходов к достижению данной цели является разработка команд длиной в одно слово и кодирование их таким образом, чтобы разряды таких коротких команд использовать оптимально, что позволит сократить время реализации программы и ее длину.

Другим подходом к оптимизации системы команд является использование микроинструкций. В этом случае отдельные биты или группы бит команды используются для кодирования нескольких элементарных операций, которые выполняются в одном командном цикле. Эти элементарные операции не требуют обращения к памяти, а последовательность их реализации определяется аппаратной логикой.

Сокращение времени выполнения программ и емкости памяти достигается за счет увеличения сложности логики управления.

Важной характеристикой команды является ее формат, определяющий структурные элементы команды, каждый из которых интерпретируется определенным образом при ее выполнении. Среди таких элементов (полей) команды выделяют следующие: код операции, определяющий выполняемое действие; адрес ячейки памяти, регистра процессора, внешнего устройства; режим адресации; операнд при использовании непосредственной адресации; код анализируемых признаков для команд условного перехода.

Классификация команд по основным признакам представлена на рис. 5.1. Важнейшим структурным элементом формата любой команды является код операции (КОП), определяющий действие, которое должно быть выполнено. Большое число КОП в процессоре очень важно, так как аппаратная реализация команд экономит память и время. Но при выборе ЭВМ необходимо концентрировать внимание на полноте операций с конкретными типами данных, а не только на числе команд и на доступных режимах адресации. Число бит, отводимое под КОП, является функцией полного набора реализуемых команд.

Команды

По функциональному числу

По числу

адресов

По длине команды

По способу кодирования

команд

По способу

адресации

Передача данных

Обработка данных

Передача управления

Дополнительные

Безадресные

(нуль адресные)

Одноадресные

Двухадресные

С фиксированным полем КОП

С расширяющимся полем КОП

Длиной в одно слово (байт)

Длиной в два слова (байта)

Длиной в три слова (байта)

Рис. 5.1. Классификация команд

При использовании фиксированного числа бит под КОП для кодирования всех *m* команд необходимо в поле КОП выделить *n* двоичных разрядов, чтобы 2*n* = *m*. Однако, учитывая ограниченную длину слова мини- и микроЭВМ, различное функциональное назначение команд, источники и приемники результатов операций, а также то, что не все команды содержат адресную часть для обращения к памяти и периферийным устройствам, в малых ЭВМ для кодирования команд широко используется принцип кодирования с переменным числом бит под поле КОП для различных групп команд.

В некоторых командах необходим только один операнд и они называются однооперандными (или одноадресными) командами в отличие от двухоперандных (или двухадресных), в которых требуются два операнда. При наличии двух операндов командой обычно изменяется только один из них. Так как информация берется только из одной ячейки, эту ячейку называют источником; ячейку, содержимое которой изменяется, называют приемником.

По форматам команд можно судить о возможностях ЭВМ.

5.1.2. Режимы адресации

Для взаимодействия с различными модулями в ЭВМ должны быть средства идентификации ячеек внешней памяти, ячеек внутренней памяти, регистров МП и регистров устройств ввода/вывода. Поэтому каждой из запоминающих ячеек присваивается адрес, т. е. однозначная комбинация бит. Количество бит определяет число идентифицируемых ячеек. Обычно ЭВМ имеет различные адресные пространства памяти и регистров МП, а иногда – отдельные адресные пространства регистров устройств ввода/вывода и внутренней памяти. Кроме того, память хранит как данные, так и команды. Поэтому для ЭВМ разработано множество способов обращения к памяти, называемых режимами адресации.

Режим адресации памяти – это процедура или схема преобразования адресной информации об операнде в его исполнительный адрес.

Все способы адресации памяти можно разделить на два типа:

1) прямой, когда исполнительный адрес берется непосредственно из команды или вычисляется с использованием значения, указанного в команде, и содержимого какого-либо регистра (прямая адресация, регистровая, базовая, индексная и т. д.);   
2) косвенный, который предполагает, что в команде содержится значение косвенного адреса, т. е. адреса ячейки памяти, в которой находится окончательный исполнительный адрес (косвенная адресация).

5.1.3. Адресация операндов

Большая часть команд процессора работает с кодами данных (операндами). Одни команды требуют входных операндов (одного или двух), другие выдают выходные операнды (чаще один операнд). Входные операнды называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды операндов (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения операндов производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной операнд и куда помещать выходной операнд. Эти методы называются **методами адресации**. Эффективность выбранных *методов адресации* во многом определяет эффективность работы всего процессора в целом.

5.1.4. Методы адресации

Количество *методов адресации* в различных процессорах может быть от 4 до 16. Рассмотрим несколько типичных *методов адресации операндов*, используемых сейчас в большинстве микропроцессоров.

***Непосредственная адресация*** (рис. 5.2) предполагает, что *операнд* (входной) находится в памяти непосредственно за кодом команды. *Операнд* обычно представляет собой константу, которую надо куда-то переслать, к чему-то прибавить и т. д. Например, команда может состоять в том, чтобы прибавить число 6 к содержимому какого-то внутреннего регистра процессора. Это число 6 будет располагаться в памяти, внутри программы в адресе, следующем за кодом данной команды сложения.

Программа

Команда

Операнд

Память

Рис.5.2. Непосредственная адресация

***Прямая (абсолютная) адресация*** (рис. 5.3) предполагает, что *операнд* (входной или выходной) находится в памяти по адресу, код которого находится внутри программы сразу же за кодом команды. Например, команда может состоять в том, чтобы очистить (сделать нулевым) содержимое ячейки памяти с адресом 1000000. Код этого адреса 1000000 будет располагаться в памяти, внутри программы в следующем адресе за кодом данной команды очистки.

Память

Команда

Операнд

Программа

Операнд

Данные

Рис. 5.3.  Прямая адресация.

***Регистровая адресация*** (рис. 5.4) предполагает, что *операнд* (входной или выходной) находится во внутреннем регистре процессора. Например, команда может состоять в том, чтобы переслать число из нулевого регистра в первый. Номера обоих регистров (0 и 1) будут определяться кодом *команды пересылки*.

Косвенно-регистровая (она же косвенная) адресация предполагает, что во внутреннем регистре процессора находится не сам *операнд*, а его адрес в памяти (рис. 3.5). Например, команда может состоять в том, чтобы очистить ячейку памяти с адресом, находящимся в нулевом регистре. Номер этого регистра (0) будет определяться кодом команды очистки.

Память

Регистр

Команда

Программа

Операнд

Рис. 5.4.  Регистровая адресация.

Память

Регистр

Команда

Операнд

Программа

Операнд

Данные

Рис. 5.5.  Косвенная адресация.

Реже встречаются еще два *метода адресации*.

***Автоинкрементная адресация*** очень близка к косвенной адресации, но отличается от нее тем, что после выполнения команды содержимое используемого регистра увеличивается на единицу или на два. Этот *метод адресации* очень удобен, например, при последовательной обработке кодов из массива данных, находящегося в памяти. После обработки какого-то кода адрес в регистре будет указывать уже на следующий код из массива. При использовании косвенной адресации в данном случае пришлось бы увеличивать содержимое этого регистра отдельной командой.

***Автодекрементная адресация*** работает похоже на автоинкрементную, но только содержимое выбранного регистра уменьшается на единицу или на два перед выполнением команды. Эта адресация также удобна при обработке массивов данных. Совместное использование автоинкрементной и автодекрементной адресаций позволяет организовать память стекового типа.

Из других распространенных *методов адресации* можно упомянуть об индексных методах, которые предполагают для вычисления адреса *операнда* прибавление к содержимому регистра заданной константы (индекса). Код этой константы располагается в памяти непосредственно за кодом команды.

Отметим, что выбор того или иного *метода адресации* в значительной степени определяет время выполнения команды. Самая быстрая адресация – это регистровая, так как она не требует дополнительных циклов обмена по магистрали. Если же адресация требует обращения к памяти, то время выполнения команды будет увеличиваться за счет длительности необходимых циклов обращения к памяти. Понятно, что чем больше внутренних регистров у процессора, тем чаще и свободнее можно применять регистровую адресацию, и тем быстрее будет работать система в целом.

5.1.5. Сегментирование памяти

Говоря об адресации, нельзя обойти вопрос о *сегментировании* памяти, применяемой в некоторых процессорах, например в процессорах IBM PC-совместимых персональных компьютеров.

В процессоре Intel 8086 *сегментирование* памяти организовано следующим образом.

Вся память системы представляется не в виде непрерывного пространства, а в виде нескольких кусков – сегментов заданного размера (по 64 Кбайта), положение которых в пространстве памяти можно изменять программным путем.

Для хранения кодов адресов памяти используются не отдельные регистры, а пары регистров:

* сегментный регистр определяет адрес начала сегмента (т. е. положение сегмента в памяти);
* регистр указателя (регистр смещения) определяет положение рабочего адреса внутри сегмента.

При этом физический 20-разрядный адрес памяти, выставляемый на внешнюю *шину адреса*, образуется так, как показано на рис. 5.6, то есть путем сложения смещения и адреса сегмента со сдвигом на 4 бита. Положение этого адреса в памяти показано на рис. 5.7.

Сегмент может начинаться только на 16-байтной границе памяти (так как адрес начала сегмента, по сути, имеет четыре младших нулевых разряда, как видно из рис. 5.6), т. е. с адреса, кратного 16.

Эти допустимые границы сегментов называются границами параграфов.

Отметим, что введение *сегментирования*, прежде всего, связано с тем, что внутренние регистры процессора 16-разрядные, а физический адрес памяти 20-разрядный (16-разрядный адрес позволяет использовать память только в 64 Кбайт, что явно недостаточно). В появившемся в то же время процессоре MC68000 фирмы Motorola внутренние регистры 32-разрядные, поэтому там проблемы *сегментирования* памяти не возникает.

16 бит

20 бит

4 бита

16 бит

Исполнительный адрес

(смещение)

Адрес сегмента

Физический адрес памяти

Рис. 5.6.  Формирование физического адреса памяти из адреса сегмента и смещения

15АВЕ

Смещение (ЗАВЕ)

12000

.

.

.

.

.

.

Память

Адрес сегмента

Размер сегмента

(10000 байт = 64 Кбайт)

Физический адрес

Адреса 00000

21FFF

22000

Рис. 5.7.  Физический адрес в сегменте (все коды – шестнадцатеричные)

Применяются и более сложные методы *сегментирования* памяти. Например, в процессоре *Intel 80286* в так называемом защищенном режиме адрес памяти вычисляется в соответствии с [рис. 5.](http://www.intuit.ru/department/hardware/mpbasics/4/#image.3.7)8.

В *сегментном регистре* в данном случае хранится не базовый (начальный) адрес сегментов, а коды селекторов, определяющие адреса в памяти, по которым хранятся дескрипторы (т. е. описатели) сегментов. Область памяти с дескрипторами называется таблицей дескрипторов. Каждый дескриптор сегмента содержит базовый адрес сегмента, размер сегмента (от 1 до 64 Кбайт) и его атрибуты. Базовый адрес сегмента имеет разрядность 24 бит, что обеспечивает адресацию 16 Мбайт физической памяти.

Процессор

Сегментный регистр

15

0

Селектор

15

0

Смещение

Сумматор физического адреса

23

0

Базовый адрес сегмента

Память

Операнд

Операнд

Сегмент

Дескриптор

сегмента

Таблица

дескрипторов

сегмента

Рис. 5.8.  Адресация памяти в защищенном режиме процессора Intel 80286

Таким образом, на *сумматор*, вычисляющий физический адрес памяти, подается не содержимое сегментного регистра, как в предыдущем случае, а базовый адрес сегмента из таблицы дескрипторов.

Еще более сложный *метод адресации* памяти с *сегментированием* использован в процессоре *Intel 80386* и в более поздних моделях процессоров фирмы Intel. Этот метод иллюстрируется рис. 5.9.

Индекс

дескриптора

32

Эффективный

адрес

Вычисление эффективного адреса

INDEX

SCALE (1,2,4,8)

BASE

DISP

32

Линейный

адрес

32

Физический

адрес

Память

Блок страничной переадресации

Блок сегментации

Сегментный регистр

Рис. 5.9.  Формирование физического адреса памяти процессора Intel 80386 в защищенном режиме

Адрес памяти (физический адрес) вычисляется в три этапа. Сначала вычисляется так называемый **эффективный адрес** (32-разрядный) путем суммирования трех компонентов: базы, индекса и смещения (Base, Index, Displacement), причем возможно умножение индекса на масштаб (Scale). Эти компоненты имеют следующий смысл:

* смещение – это 8-, 16- или 32-разрядное число, включенное в команду;
* база – это содержимое базового регистра процессора. Обычно оно используется для указания на начало некоторого массива;
* индекс – это содержимое индексного регистра процессора. Обычно оно используется для выбора одного из элементов массива;
* масштаб – это множитель (он может быть равен 1, 2, 4 или 8), указанный в коде команды, на который перед суммированием с другими компонентами умножается индекс. Он используется для указания размера элемента массива.

Затем специальный блок *сегментации* вычисляет 32-разрядный линейный адрес, который представляет собой сумму базового адреса сегмента из сегментного регистра с эффективным адресом. Наконец, физический 32-битный адрес памяти образуется путем преобразования линейного адреса блоком страничной переадресации, который осуществляет перевод линейного адреса в *физический страницами* по 4 Кбайта.

В любом случае *сегментирование* позволяет выделить в памяти один или несколько сегментов для данных и один или несколько сегментов для программ. Переход от одного сегмента к другому сводится всего лишь к изменению содержимого сегментного регистра. Иногда это бывает очень удобно. Но для программиста работать с сегментированной памятью обычно сложнее, чем с непрерывной, несегментированной памятью, так как приходится следить за границами сегментов, за их описанием, переключением и т. д.

5.1.6. Адресация байтов и слов

Многие процессоры, имеющие разрядность 16 или 32, способны адресовать не только целое слово в памяти (16-разрядное или 32-разрядное), но и отдельные байты. Каждому байту в каждом слове при этом отводится свой адрес.

Так, в случае 16-разрядных процессоров все слова в памяти (16-разрядные) имеют четные адреса. А байты, входящие в эти слова, могут иметь как четные адреса, так и нечетные.

Например, пусть 16-разрядная ячейка памяти имеет адрес 23420, и в ней хранится код 2А5Е (рис. 5.10).

.

.

16 бит

Память

.

.

2A 5E

48 7F

Старший байт

(адрес 23421)

Старший байт

(адрес 23423)

Младший байт

(адрес 23420)

Младший байт

(адрес 23422)

Слово

(адрес 23422)

Слово

(адрес 23420)

Рис. 5.10. Адресация слов и байтов

При обращении к целому слову (с содержимым 2А5Е) процессор выставляет адрес 23420. При обращении к младшему байту этой ячейки (с содержимым 5Е) процессор выставляет тот же самый адрес 23420, но использует команду, адресующую байт, а не слово. При обращении к старшему байту этой же ячейки (с содержимым 2А) процессор выставляет адрес 23421 и использует команду, адресующую байт. Следующая по порядку 16-разрядная ячейка памяти с содержимым 487F будет иметь адрес 23422, т. е. опять же четный. Ее байты будут иметь адреса 23422 и 23423.

Для различия байтовых и словных циклов обмена на магистрали в шине управления предусматривается специальный сигнал байтового обмена. Для работы с байтами в систему команд процессора вводятся специальные команды или предусматриваются методы байтовой адресации.

5.2. Регистры процессора

Как уже упоминалось, внутренние регистры процессора представляют собой сверхоперативную память небольшого размера, которая предназначена для временного хранения служебной информации или данных. Количество регистров в разных процессорах может быть от 6–8 до нескольких десятков. Регистры могут быть универсальными и специализированными. Специализированные регистры, которые присутствуют в большинстве процессоров, – это *регистр*-*счетчик команд, регистр состояния* (**PSW**), *регистр указателя стека*. Остальные регистры процессора могут быть как универсальными, так и специализированными.

Например, в 16-разрядном процессоре Т-11 фирмы DEC было 8 регистров общего назначения (РОН) и один *регистр состояния*. Все регистры имели по 16 разрядов. Из регистров общего назначения один отводился под *счетчик команд*, другой – под указатель стека. Все остальные регистры общего назначения полностью взаимозаменяемы, т. е. имеют универсальное назначение, могут хранить как данные, так и адреса (указатели), индексы и т. д. Максимально допустимый объем памяти для данного процессора составлял 64 Кбайт (адрес памяти 16-разрядный).

В 16-разрядном процессоре MC68000 фирмы Motorola было 19 регистров: 16-разрядный *регистр состояния*, 32-разрядный регистр *счетчика команд*, 9 регистров адреса (32-разрядных) и 8 регистров данных (32-разрядных). Два регистра адреса отведены под указатели стека. Максимально допустимый объем адресуемой памяти – 16 Мбайт (внешняя шина адреса 24-разрядная). Все 8 регистров данных взаимозаменяемы, 7 регистров адреса тоже взаимозаменяемы.

В 16-разрядном процессоре Intel 8086, который стал базовым в линии процессоров, используемых в персональных компьютерах, реализован принципиально другой подход. Каждый регистр этого процессора имеет свое особое назначение, и заменять друг друга регистры могут только частично или же не могут вообще. Остановимся на особенностях этого процессора подробнее.

Процессор Intel 8086 имеет 14 регистров разрядностью по 16 бит. Из них четыре регистра (AX, BX, CX, DX) – это регистры данных, каждый из которых помимо хранения операндов и результатов операций имеет еще и свое специфическое назначение:

• регистр AX – умножение, деление, обмен с устройствами ввода/вывода (команды ввода и вывода);

• регистр BX – базовый регистр в вычислениях адреса;

• регистр CX – счетчик циклов;

• регистр DX – определение адреса ввода/вывода.

Для регистров данных существует возможность раздельного использования обоих байтов (например, для регистра AX они имеют обозначения AL – младший байт и AH – старший байт).

Следующие четыре внутренних регистра процессора – это сегментные регистры, каждый из которых определяет положение одного из рабочих сегментов (рис. 5.11):

• CS (CodeSegment) соответствует сегменту команд, исполняемых в данный момент;

• DS (DataSegment) соответствует сегменту данных, с которыми работает процессор;

• ES (ExtraSegment) соответствует дополнительному сегменту данных;

• SS (StackSegment) соответствует сегменту стека.

**.**

**.**

**.**

(CS)

Сегмент команд

(DS)

**.**

**.**

**.**

Сегмент данных

Дополнительный сегмент данных

(ES)

**.**

**.**

**.**

(SS)

**.**

**.**

**.**

Стек

**.**

**.**

**.**

Рис. 5.11. Сегменты команд, данных и стека в памяти

В принципе, все эти сегменты могут и перекрываться для оптимального использования пространства памяти. Например, если программа занимает только часть сегмента, то сегмент данных может начинаться сразу после завершения работы программы (с точностью 16 байт), а не после окончания всего сегмента программы.

Следующие пять регистров процессора (SP – StackPointer, BP –BasePointer, SI – SourceIndex, DI – DestinationIndex, IP – InstructionPointer) служат указателями (т. е. определяют смещение в пределах сегмента). Например, *счетчик команд* процессора образуется парой регистров CS и IP, а указатель стека – парой регистров SP и SS. Регистры SI, DI используются в строковых операциях, т. е. при последовательной обработке нескольких ячеек памяти одной командой.

Последний регистр FLAGS – это регистр состояния процессора (PSW). Из его 16 разрядов используются только девять (рис. 5.12): CF (CarryFlag) – флаг переноса при арифметических операциях, PF (ParityFlag) – флаг четности результата, AF (AuxiliaryFlag) – флаг дополнительного переноса, ZF (ZeroFlag) — флаг нулевого результата, SF (SignFlag) – флаг знака (совпадает со старшим битом результата), TF (TrapFlag) – флаг пошагового режима (используется при отладке), IF (Interrupt-enableFlag) – флаг разрешения аппаратных прерываний, DF (DirectionFlag) – флаг направления при строковых операциях, OF (OverflowFlag) – флаг переполнения.



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | OF | DF | IF | TF | SF | ZF |  | AF |  | PF |  | CF |

Рис. 5.12. Регистр состояния процессора Intel 8086

Биты регистра состояния устанавливаются или очищаются в зависимости от результата исполнения предыдущей команды и используются некоторыми командами процессора. Биты регистра состояния могут также устанавливаться и очищаться специальными командами процессора (о системе команд процессора будет рассказано в следующем разделе).

Во многих процессорах выделяется специальный регистр, называемый **аккумулятором** (то есть накопителем). При этом, как правило, только этот регистр-аккумулятор может участвовать во всех операциях, только через него может производиться взаимодействие с устройствами ввода/вывода. Иногда в него же помещается результат любой выполненной команды (в этом случае говорят даже об «аккумуляторной» архитектуре процессора). Например, в процессоре 8086 регистр данных АХ можно считать своеобразным аккумулятором, так как именно он обязательно участвует в командах умножения и деления, а также только через него можно пересылать данные в устройство ввода/вывода и из устройства ввода/вывода. Выделение специального регистра-аккумулятора упрощает структуру процессора и ускоряет пересылки кодов внутри процессора, но в некоторых случаях замедляет работу системы в целом, так как весь поток информации должен пройти через один регистр-аккумулятор. В случае, когда несколько регистров процессора полностью взаимозаменяемы, таких проблем не возникает.

К данному типу архитектуры относится микропроцессор фирмы Zilog. Процессор Z80 – детище фирмы Zilog помимо расширенной системы команд, одного номинала питания и способности исполнять программы, написанные для i8080, имел архитектурные «изюминки».

В дополнение к основному набору РОН, в кристалле был реализован второй комплект аналогичных регистров. Это значительно упрощало работу при вызове подпрограмм или процедур обслуживания прерываний, поскольку программист мог использовать для них альтернативный набор регистров, избегая сохранения в стеке содержимого РОНов для основной программы с помощью операций PUSH. Кроме того, в систему команд был включен ряд специальных инструкций, ориентированных на обработку отдельных битов, а для поддержки регенерации динамической памяти в схему процессора введены соответствующие аппаратные средства. Z80 применялся в машинахSinclairZX, SinclairSpectrum, TandyTRS80.

Предельный вариант – архитектура с адресацией посредством аккумуляторов (меньший набор команд).

МП фирмы Motorola имел ряд существенных преимуществ. Прежде всего, кристалл МС6800 требовал для работы одного номинала питания, а система команд оказалась весьма прозрачной для программиста. Архитектура МП также имела ряд особенностей.

Шина адреса (16 бит)

Шина данных (8бит)

Управление (13 линий)

Регистр команды

8 бит

Индексный регистр 1Х

Индексный регистр 1Y

Указатель стека SP

Программный

счетчик PC

16 бит

Основной набор регистров

Аккумулятор A

B

D

H

S

Z

H

PV

N

C

C

E

L

Аккумулятор A

B`

D`

H`

S

Z

H

PV

N

C

C`

E`

L`

АЛУ

8 бит

8 бит

8 бит

8 бит

PSW

Дополнительный набор регистров

Управление шиной

Вектор прерывания 1

Регенерация памяти - R

8 бит

8 бит

Рис. 5.13. Микропроцессор Z80 фирмы Zilog

Адрес(16 бит)

Данные (8бит)

Управление (9 линий)

8 бит

АЛУ

Управление шиной

16 бит

16 бит

16 бит

Программный счетчик

Указатель стека

Индексный регистр

H

I

N

Z

V

C

Регистр команды

Состояние процессора

А (8 бит)

А (8 бит)

Аккумуляторы

Рис 5.14. Микропроцессор МС6800 фирмы Motorola.

Микропроцессор МС 6800 содержал два аккумулятора, и результат операции АЛУ мог быть помещен в любой из них. Но самым ценным качеством структуры МС 6800 было автоматическое сохранение в стеке содержимого всех регистров процессора при обработке прерываний (Z80 требовалось для этого несколько команд PUSH). Процедура восстановления РОН из стека тоже выполнялась аппаратно.

1. Стековая архитектура дает возможность создать поле памяти с упорядоченной последовательностью записи и выборки информации.
2. В общем случае команды неявно адресуются к элементу стека, расположенному на его вершине, или к двум верхним элементам стека.
3. Архитектура МП, ориентированная на оперативную память (типа «память-память»), обеспечивает высокую скорость работы и большую информационную емкость рабочих регистров и стека при их организации в оперативной памяти.

Архитектура этого типа не предполагает явного определения аккумулятора, регистров общего назначения или стека; все операнды команд адресуются к области основной памяти.

С точки зрения важности для пользователя-программиста под архитектурой в общем случае понимают совокупность следующих компонентов и характеристик:

* разрядности адресов и данных;
* состава, имен и назначения программно-доступных регистров;
* форматов и системы команд;
* режимов адресации памяти;
* способов машинного представления данных разного типа;
* структуры адресного пространства;
* способа адресации внешних устройств и средств выполнения операций ввода/вывода;
* классов прерываний, особенностей инициирования и обработки прерываний.

5.2.1. Система команд процессора

В общем случае система команд процессора включает в себя следующие четыре основные группы команд:

— команды пересылки данных;

— арифметические;

— логические;

— команды переходов.

**Команды пересылки** данных не требуют выполнения никаких операций над операндами. Операнды просто пересылаются (точнее, копируются) из источника (Source) в приемник (Destination). Источником и приемником могут быть внутренние регистры процессора, ячейки памяти или устройства ввода/вывода. АЛУ в данном случае не используется.

**Арифметические команды** выполняют операции сложения, вычитания, умножения, деления, увеличения на единицу (инкрементирования), уменьшения на единицу (декрементирования) и т. д. Этим командам требуется один или два входных операнда. Формируют команды один выходной операнд.

**Логические команды** производят над операндами логические операции, например, логическое «И», *логическое* «ИЛИ», исключающее «ИЛИ», очистку, инверсию, разнообразные сдвиги (вправо, влево, арифметический сдвиг, циклический сдвиг). Этим командам, как и *арифметическим*, требуется один или два входных операнда, и формируют они один выходной операнд.

Наконец, **команды переходов** предназначены для изменения обычного порядка последовательного выполнения команд. С их помощью организуются переходы на подпрограммы и возвраты из них, всевозможные циклы, ветвления программ, пропуски фрагментов программ и т. д. Команды переходов всегда меняют содержимое счетчика команд. Переходы могут быть условными и безусловными. Именно эти команды позволяют строить сложные алгоритмы обработки информации.

В соответствии с результатом каждой выполненной команды устанавливаются или очищаются биты регистра состояния процессора (PSW). Но надо помнить, что не все команды изменяют все имеющиеся в PSW флаги. Это определяется особенностями каждого конкретного процессора.

У разных процессоров системы команд существенно различаются, но в основе своей они очень похожи. Количество команд у процессоров также различно. Например, у упоминавшегося уже процессора МС68000 всего 61 команда, а у процессора 8086 – 133 команды. У современных мощных процессоров количество команд достигает нескольких сотен. В то же время существуют процессоры с *сокращенным набором команд* (так называемые RISC-процессоры), в которых за счет максимального сокращения количества команд достигается увеличение эффективности и скорости их выполнения.

Рассмотрим теперь особенности четырех выделенных групп команд процессора более подробно.

5.2.2. Команды пересылки данных

*Команды пересылки* данных занимают очень важное место в системе команд любого процессора. Они выполняют следующие важнейшие функции:

— загрузка (запись) содержимого во внутренние регистры процессора;

— сохранение в памяти содержимого внутренних регистров процессора;

— копирование содержимого из одной области памяти в другую;

— запись в устройства ввода/вывода и чтение из устройств ввода/вывода.

В некоторых процессорах (например, Т-11) все эти функции выполняются одной единственной командой MOV (для байтовых пересылок – MOVB), но с различными *методами адресации* операндов.

В других процессорах помимо команды MOV имеется еще несколько команд для выполнения перечисленных функций. Например, для загрузки регистров могут использоваться команды загрузки, причем для разных регистров – разные команды (их обозначения обычно строятся с использованием слова LOAD – загрузка). Часто выделяются специальные команды для сохранения в стеке и для извлечения из стека (PUSH – сохранить в стеке, POP – извлечь из стека). Эти команды выполняют пересылку с автоинкрементной и с автодекрементной адресацией (даже если эти режимы адресации не предусмотрены в процессоре в явном виде).

Иногда в систему команд вводится специальная команда MOVS для строчной (или цепочечной) пересылки данных (например, в процессоре Intel 8086). Эта команда пересылает не одно слово или байт, а заданное количество слов или байтов (MOVSB), то есть инициирует не один цикл обмена по магистрали, а несколько. При этом адрес памяти, с которым происходит взаимодействие, увеличивается на 1 или на 2 после каждого обращения или же уменьшается на 1 или на 2 после каждого обращения. То есть в неявном виде применяется автоинкрементная или автодекрементная адресация.

В некоторых процессорах (например, в процессоре Intel 8086) специально выделяются функции обмена с устройствами ввода/вывода. Команда IN используется для ввода (чтения) информации из устройства ввода/вывода, а команда OUT используется для вывода (записи) в устройство ввода/вывода. Обмен информацией в этом случае производится между регистром-аккумулятором и устройством ввода/вывода. В более продвинутых процессорах этого же семейства (начиная с процессора Intel 80286) добавлены команды строчного (цепочечного) ввода (команда INS) и строчного вывода (команда OUTS). Эти команды позволяют пересылать целый массив (строку) данных из памяти в устройство ввода/вывода (OUTS) или из устройства ввода/вывода в память (INS). Адрес памяти после каждого обращения увеличивается или уменьшается (как и в случае с командой MOVS).

Также к *командам пересылки* данных относятся команды обмена информацией (их обозначение строится на основе слова *Exchange*). Может быть предусмотрен обмен информацией между внутренними регистрами, между двумя половинами одного регистра (SWAP) или между регистром и ячейкой памяти.

5.2.3. Арифметические команды

*Арифметические команды*рассматривают коды операндов как числовые двоичные или двоично-десятичные коды. Эти команды могут быть разделены на пять основных групп:

— команды операций с *фиксированной запятой* (сложение, вычитание, умножение, деление);

— операций с *плавающей запятой* (сложение, вычитание, умножение, деление);

— очистки;

— инкремента и декремента;

— сравнения.

Команды операций с *фиксированной запятой* работают с кодами в регистрах процессора или в памяти как с обычными двоичными кодами. Команда сложения (ADD) вычисляет сумму двух кодов. Команда вычитания (SUB) вычисляет разность двух кодов. Команда умножения (MUL) вычисляет произведение двух кодов (разрядность результата вдвое больше разрядности сомножителей). Команда деления (DIV) вычисляет частное от деления одного кода на другой. Причем все эти команды могут работать как с числами со знаком, так и с числами без знака.

Команды операций с *плавающей запятой* (точкой) используют формат представления чисел с порядком и *мантиссой* (обычно эти числа занимают две последовательные ячейки памяти). В современных мощных процессорах набор команд с *плавающей запятой* не ограничивается только четырьмя арифметическими действиями, а содержит и множество других более сложных команд, например вычисление тригонометрических функций, логарифмических функций, а также сложных функций, необходимых при обработке звука и изображения.

Команды очистки (CLR) предназначены для записи нулевого кода в регистр или ячейку памяти. Эти команды могут быть заменены *командами пересылки* нулевого кода, но специальные команды очистки обычно выполняются быстрее, чем *команды пересылки*. Команды очистки иногда относят к группе *логических команд*, но суть их от этого не меняется.

Команды инкремента (увеличения на единицу, INC) и декремента (уменьшения на единицу, DEC) также бывают очень удобны. Их можно заменить командами суммирования с единицей или вычитания единицы, но инкремент и декремент выполняются быстрее, чем суммирование и вычитание. Эти команды требуют одного входного операнда, который одновременно является и выходным операндом.

Наконец, команда сравнения (обозначается CMP) предназначена для сравнения двух входных операндов. По сути, она вычисляет разность этих двух операндов, но выходного операнда не формирует, а всего лишь изменяет биты в регистре состояния процессора (PSW) по результату этого вычитания. Следующая за командой сравнения команда (обычно это *команда перехода*) будет анализировать биты в регистре состояния процессора и выполнять действия в зависимости от их значений. В некоторых процессорах предусмотрены команды цепочечного сравнения двух последовательностей операндов, находящихся в памяти.

5.2.4. Логические команды

*Логические команды* выполняют над операндами логические (побитовые) операции, т. е. они рассматривают коды операндов не как единое число, а как набор отдельных битов. Этим они отличаются от *арифметических команд*. *Логические команды* выполняют следующие основные операции:

— логическое «И», логическое «ИЛИ», сложение по модулю 2 (исключающее «ИЛИ»);

— логические, арифметические и циклические сдвиги;

— проверка битов и операндов;

— установка и очистка битов (флагов) регистра состояния процессора (PSW).

Команды логических операций позволяют побитно вычислять основные логические функции от двух входных операндов. Кроме того, операция «И» (AND) используется для принудительной очистки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие очистки, установлены в нуль). Операция «ИЛИ» (OR) применяется для принудительной установки заданных битов (в качестве одного из операндов при этом используется код маски, в котором разряды, требующие установки в единицу, равны единице). Операция «исключающее ИЛИ» (XOR) используется для инверсии заданных битов (в качестве одного из операндов при этом применяется код маски, в котором биты, подлежащие инверсии, установлены в единицу). Команды требуют двух входных операндов и формируют один выходной операнд.

Команды сдвигов позволяют побитно сдвигать код операнда вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Тип сдвига (логический, арифметический или циклический) определяет, каково будет новое значение старшего бита (при сдвиге вправо) или младшего бита (при сдвиге влево), а также определяет, будет ли где-то сохранено прежнее значение старшего бита (при сдвиге влево) или младшего бита (при сдвиге вправо). Например, при логическом сдвиге вправо в старшем разряде кода операнда устанавливается нуль, а младший разряд записывается в качестве флага переноса в регистр состояния процессора. А при арифметическом сдвиге вправо значение старшего разряда сохраняется прежним (нулем или единицей), младший разряд также записывается в качестве флага переноса.

Циклические сдвиги позволяют сдвигать биты кода операнда по кругу (по часовой стрелке при сдвиге вправо или против часовой стрелки при сдвиге влево). При этом в кольцо сдвига может входить или не входить флаг переноса. В бит флага переноса (если он используется) записывается значение старшего бита при циклическом сдвиге влево и младшего бита при циклическом сдвиге вправо. Соответственно, значение бита флага переноса будет переписываться в младший разряд при циклическом сдвиге влево и в старший разряд при циклическом сдвиге вправо.

Для примера на рис. 5.15 показаны действия, выполняемые командами сдвигов вправо.

Команды проверки битов и операндов предназначены для установки или очистки битов *регистра состояния* процессора в зависимости от значения выбранных битов или всего операнда в целом. Выходного операнда команды не формируют. Команда проверки операнда (TST) проверяет весь код операнда в целом на равенство нулю и на знак (на значение старшего бита), она требует только одного входного операнда. Команда проверки бита (BIT) проверяет только отдельные биты, для выбора которых в качестве второго операнда используется код маски. В коде маски проверяемым битам основного операнда должны соответствовать единичные разряды.

Циклический сдвиг вправо

через перенос

Циклический сдвиг вправо

Арифметический сдвиг вправо

Логический сдвиг вправо

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Флаг переноса

Мл.

Ст.

Операнд

Рис. 5.15. Команды сдвигов вправо

Наконец, команды установки и очистки битов *регистра состояния* процессора (т. е. флагов) позволяют установить или очистить любой флаг, что бывает очень удобно. Каждому флагу обычно соответствуют две команды, одна из которых устанавливает его в единицу, а другая сбрасывает в нуль. Например, флагу переноса C (от Carry) будут соответствовать команды CLC (очистка) и SEC или STC (установка).

5.2.5. Команды переходов.

*Команды переходов*предназначены для организации всевозможных циклов, *ветвлений*, вызовов *подпрограмм* и т. д., т. е. они нарушают последовательный ход выполнения программы. Эти команды записывают в регистр-*счетчик команд* новое значение и тем самым вызывают переход процессора не к следующей по порядку команде, а к любой другой команде в памяти программ. Некоторые *команды переходов* предусматривают в дальнейшем возврат назад, в точку, из которой был сделан переход, другие не предусматривают этого. Если возврат предусмотрен, то текущие параметры процессора сохраняются в стеке. Если возврат не предусмотрен, то текущие параметры процессора не сохраняются.

*Команды переходов* без возврата делятся на две группы:

— команды *безусловных переходов*;

— команды *условных переходов.*

В обозначениях этих команд используются слова *Branch* (ветвление) и *Jump* (прыжок).

Команды безусловных переходов вызывают переход в новый адрес независимо ни от чего. Они могут вызывать переход на указанную величину смещения (вперед или назад) или же на указанный адрес памяти. Величина смещения или новое значение адреса указываются в качестве входного операнда.

Команды условных переходов вызывают переход не всегда, а только при выполнении заданных условий. В качестве таких условий обычно выступают значения флагов в регистре состояния процессора (PSW). То есть условием перехода является результат предыдущей операции, меняющей значения флагов. Всего таких условий перехода может быть от 4 до 16. Несколько примеров команд условных переходов:

— переход, если равно нулю;

— если не равно нулю;

— если есть переполнение;

— если нет переполнения;

— если больше нуля;

— если меньше или равно нулю.

Если условие перехода выполняется, то производится загрузка в регистр-счетчик команд нового значения. Если же условие перехода не выполняется, счетчик команд просто наращивается, и процессор выбирает и выполняет следующую по порядку команду.

Специально для проверки условий перехода применяется команда сравнения (CMP), предшествующая команде условного перехода (или даже нескольким командам условных переходов). Но флаги могут устанавливаться и любой другой командой, например командой пересылки данных, любой арифметической или логической командой. Отметим, что сами команды переходов флаги не меняют, что как раз и позволяет ставить несколько команд переходов одну за другой.

Совместное использование нескольких команд условных и безусловных переходов позволяет процессору выполнять разветвленные алгоритмы любой сложности. Для примера на рис. 5.15 показано разветвление программы на две ветки с последующим соединением, а на рис. 5.16 — разветвление на три ветки с последующим соединением.

Команды переходов с дальнейшим возвратом в точку, из которой был произведен переход, применяются для выполнения подпрограмм, т. е. вспомогательных программ. Эти команды называются также командами вызова подпрограмм (распространенное название – CALL). Использование подпрограмм позволяет упростить структуру основной программы, сделать ее более логичной, гибкой, легкой для написания и отладки. В то же время надо учитывать, что широкое использование подпрограмм, как правило, увеличивает время выполнения программы.

нет

да

Программа 1

А + В А

Программа 3

Программа 2

А>0

Память

Сложение

Переход если больше 0

Начало программы 1

.

.

Безусловный переход

Начало программы 2

.

.

Конец программы 2

Начало программы 3

.

.

Рис. 5.16. Реализация разветвления на две ветки

Память

Сложение

Переход, если равно 0

Переход если больше 0

Начало программы 3

.

.

Безусловный переход

Начало программы 1

.

.

Безусловный переход

Начало программы 2

.

.

Конец программы 2

Начало программы 4

нет

да

нет

Программа 1

А + В А

Программа 3

Программа 2

А=0

А>0

Программа 4

Рис. 5.17. Реализация разветвления на три ветки

Все команды переходов с возвратом предполагают безусловный переход (они не проверяют никаких флагов). При этом они требуют одного входного операнда, который может указывать как абсолютное значение нового адреса, так и смещение, складываемое с текущим значением адреса. Текущее значение счетчика команд (текущий адрес) сохраняется перед выполнением перехода в стеке.

Для обратного возврата в точку вызова подпрограммы (точку перехода) используется специальная команда возврата (RET или RTS). Эта команда извлекает из стека значение адреса команды перехода и записывает его в регистр-счетчик команд.

Особое место среди команд перехода с возвратом занимают команды прерываний (распространенное название – INT). Эти команды в качестве входного операнда требуют номер прерывания (адрес вектора). Обслуживание таких переходов осуществляется точно так же, как и аппаратных прерываний. Т. е. для выполнения данного перехода процессор обращается к таблице векторов прерываний и получает из нее по номеру прерывания адрес памяти, в который ему необходимо перейти. Адрес вызова прерывания и содержимое регистра состояния процессора (PSW) сохраняются в стеке. Сохранение PSW – важное отличие команд прерывания от команд переходов с возвратом.

Команды прерываний во многих случаях оказываются удобнее, чем обычные команды переходов с возвратом. Сформировать таблицу векторов прерываний можно один раз, а потом уже обращаться к ней по мере необходимости. Номер прерывания соответствует номеру подпрограммы, т. е. номеру функции, выполняемой подпрограммой. Поэтому команды прерывания гораздо чаще включаются в системы команд процессоров, чем обычные команды переходов с возвратом.

Для возврата из подпрограммы, вызванной командой прерывания, используется команда возврата из прерывания (IRET или RTI). Эта команда извлекает из стека сохраненное там значение счетчика команд и регистра состояния процессора (PSW).

Отметим, что у некоторых процессоров предусмотрены также команды условных прерываний, например команда прерывания при переполнении.

Конечно, в данном разделе мы рассмотрели только основные команды, наиболее часто встречающиеся в процессорах. У конкретных процессоров могут быть и многие другие команды, не относящиеся к перечисленным группам команд. Но изучать их надо уже после того, как выбран тип процессора, подходящий для задачи, решаемой данной микропроцессорной системой.

5.3. Система команд микропроцессора Intel 8086

5.3.1. Способы адресации и форматы команд микропроцессора I 8086

Микропроцессор *Intel 8086* (К1810ВМ80) имеет двухадресную систему команд. Ее особенностью является отсутствие команд, использующих оба операнда из оперативной памяти. Исключение составляют лишь команды пересылки и сравнения цепочек байт или слов, которые в данном пособии рассматриваться не будут. Таким образом, в командах допустимы следующие сочетания операндов: RR, RS, RI, SI. Здесь R обозначает операнд, находящийся в одном из регистров регистровой памяти микропроцессора, S – операнд, находящийся в оперативной памяти, адрес которого формируется по одному из допустимых способов адресации, I – непосредственный операнд, закодированный в адресном поле самой команды. Формат команды во многом определяется способом адресации операнда, находящего в оперативной памяти, длиной используемого непосредственного операнда, а также наличием и длиной смещения, используемого при относительных режимах адресации.

Микропроцессор имеет все режимы адресации, общая схема которых была рассмотрена выше. Естественно, они имеют определенные особенности, присущие данному процессору.

Непосредственная адресация предполагает, что операнд занимает одно из полей команды и, следовательно, выбирается из оперативной памяти одновременно с ней. В зависимости от форматов, обрабатываемых процессором данных непосредственный операнд может иметь длину 8 или 16 бит, что в дальнейшем будем обозначать data8 и data16 соответственно.

Механизмы адресации операндов, находящихся в регистровой памяти и в оперативной памяти, существенно различаются. К регистровой памяти допускается лишь прямая регистровая адресация. При этом в команде указывается номер регистра, содержащего операнд. 16-разрядный операнд может находиться в регистрах AX, BX, CX, DX, DI, SI, SP, BP, а 8-разрядный – в регистрах AL, AH, BL, BH, CL, CH, DL, DH.

Адресация оперативной памяти имеет свои особенности, связанные с ее разбиением на сегменты и использованием сегментной группы регистров для указания начального адреса сегмента. 16-разрядный адрес, получаемый в блоке формирования адреса операнда на основе указанного режима адресации, называется эффективным адресом (ЭА). Иногда эффективный адрес обозначается как ЕА (*effective address*). 20-разрядный адрес, который получается сложением эффективного адреса и увеличенного в 16 раз значения соответствующего сегментного регистра, называется физическим адресом (ФА).

Именно физический адрес передается из микропроцессора по 20 адресным линиям, входящим в состав системной шины, в оперативную память и используется при обращении к ее ячейке на физическом уровне. При получении эффективного адреса могут использоваться все

основные режимы адресации, рассмотренные выше, а также некоторые их комбинации.

Прямая адресация предполагает, что эффективный адрес является частью команды. Так как ЭА состоит из 16 разрядов, то и соответствующее поле команды должно иметь такую же длину.

При регистровой косвенной адресации эффективный адрес операнда находится в базовом регистре BX или в одном из индексных регистров DI либо SI:

ЭА=

[BX]

[DI]

[SI]

**.**

Обозначение имени регистра в квадратных скобках указывает на содержимое соответствующего регистра. Фигурные скобки – символ выбора одной из нескольких возможных альтернатив.

При регистровой относительной адресации эффективный адрес равен сумме содержимого базового или индексного регистра и смещения:

ЭА=

[BX]

[BP]

[DI]

[SI]

+

disp8

disp16

**.**

Обозначения disp8 и disp16 здесь и далее указывают на 8- или 16-разрядное смещение соответственно.

Эффективный адрес при базово-индексной адресации равен сумме содержимого базового и индексного регистров, определяемых командой:

ЭА=

[BX]

[BP]

+

[DI]

[SI]

**.**

Наиболее сложен механизм относительной базово-индексной адресации. Эффективный адрес в этом случае равен сумме 8- или 16-разрядного смещения и базово-индексного адреса:

**.**

+

+

ЭА=

[BX]

[BP]

[DI]

[SI]

disp8

disp16

Форматы двухоперандных команд представлены на рис. 5.18. Пунктиром показаны поля, которые в зависимости от режима адресации могут отсутствовать в команде.

формат командтипа RR і RS

1 байт

1 байт

1 байт

1 байт

КАП *d w*

*mdreg r/m*

*disp L*

*disp H*

1 байт

1 байт

1 байт

1 байт

1 байт

1 байт

КОП *s w*

*md* КОП*r/m*

*disp L*

*disp H*

*data L*

*data H*

формат командснепосредственнымоперандом

Рис. 5.18. Форматы двухоперандных команд микропроцессора I808:

Поле КОП содержит код выполняемой операции. Признак *w* указывает на длину операндов. При *w* = 1 операция проводится над словами, а при *w* = 0 – над байтами. Признак *d* указывает положение приемника результата. Признак *d* = 1, если результат записывается на место операнда, закодированного в поле reg, и *d* = 0, если результат записывается по адресу, закодированному полями (*md*, *r*/*m*).

Второй байт команды, называемый постбайтом, определяет операнды, участвующие в операции. Поле reg указывает регистр регистровой памяти согласно табл. 5.1

|  |  |  |
| --- | --- | --- |
| Таблица 5.1  **Использование поля reg** | | |
| reg | Регистр | |
| *w* = 1 | *w* = 0 |
| 000 | AX | AL |
| 001 | CX | CL |
| 010 | DX | DL |
| 011 | BX | BL |
| 100 | SP | AH |
| 101 | BP | CH |
| 110 | SI | DH |
| 111 | DI | BH |

Поля *md* и *r*/*m* задают режим адресации второго операнда согласно табл. 5.2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 5.2  **Использование полей *md* и *r*/*m*** | | | | | | | |
| *r*/*m* | md | | | | | | |
| 00 | | 01 | 10 | | 11 | |
| *w* = 1 | *w* = 0 |
| 000 | (BX)+(SI)  (DS) | | (BX)+(SI)+disp 8  (DS) | (BX)+(SI)+disp 16  (DS) | | AX | AL |
| 001 | (BX)+(DI)  (DS) | | (BX)+(DI)+disp 8  (DS) | (BX)+(DI)+disp 16  (DS) | | CX | CL |
| 010 | (BP)+(SI)  (SS) | | (BP)+(SI)+disp 8  (SS) | (BP)+(SI)+disp 16  (SS) | | DX | DL |
| 011 | (BP)+(DI)  (SS) | | (BP)+(DI)+disp 8  (SS) | (BP)+(DI)+disp 16  (SS) | | BX | BL |
| 100 | (SI)  (DS) | | (SI)+disp 8  (DS) | (SI)+disp 16  (DS) | | SP | AH |
| 101 | (DI)  (DS) | | (DI)+disp 8  (DS) | (DI)+disp 16  (DS) | | BP | CH |
| 110 | disp16  (DS) | | (BP)+disp 8  (SS) | (BP)+disp 16  (SS) | | SI | DH |
| 111 | | (BX)  (DS) | (BX)+disp 8  (DS) | (BX)+disp 16  (DS) | DI | | BH |

В этой таблице помимо определения режима адресации оперативной памяти указан также сегментный регистр, используемый по умолчанию для получения физического адреса. Использование другого сегментного регистра возможно введением специального префикса (дополнительного байта, который записывается перед командой).

В командах, использующих непосредственный операнд, признак *s* вместе с признаком *w* определяет разрядность непосредственного операнда, записываемого в команде, и разрядность выполняемой операции согласно [табл. 5.3](http://www.intuit.ru/department/hardware/archhard2/6/2.html#table.6.3).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 5.3  **Использование признака *s*** | | | |
| ***w*** | ***s*** | Операция | Непосредственный операнд |
| 0 | 0 | 8-разрядная | 8-разрядный |
| 0 | 1 | Не используется | |
| 1 | 0 | 16-разрядная | 16-разрядный |
| 1 | 1 | 8-разрядный, расширяемый знаком до 16 разрядов при выполнении операции |

Изменение естественного порядка выполнения команд программы осуществляется с помощью команд передачи управления. К ним относятся команды переходов, циклов, вызова подпрограммы и возврата из нее, а также некоторые другие. Мы рассмотрим лишь первые две группы команд.

Классификация команд переходов в персональной ЭВМ представлена на рис. 5.19.

Команды переходов

Условные

Прямые

Косвенные

Внутрисегментные

Безусловные

Прямые

Косвенные

Межсегментные

Рис. 5.19. Классификация команд переходов IBM PC

Физический адрес выполняемой команды определяется содержимым указателя команд IP и сегментного регистра команд CS. Команды, меняющие значение обоих этих регистров, называются командами межсегментных переходов, а меняющие только значение IP, – командами внутрисегментных переходов.

Команды безусловных переходов производят модификацию регистра IP или регистров IP и CS без предварительного анализа каких-либо условий. Существует пять команд безусловных переходов. Все они имеют одинаковую мнемонику JMP и содержат один операнд. Конкретный формат команды определяется соответствующим префиксом и приведен в общей таблице машинного представления команд (табл. 5.4).

Таблица 5.4

**Машинные коды некоторых команд**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Команда | Байты кода команды | | | Схема операции |
| байт 1 | байт 2 | байты 3–6 |
| ADD | 000000dw | *md*reg r/m | (disp8/16) | r(r/m) = r + r/m |
| 100000sw | *md* 000 r/m | (disp8/disp16)d8/16 | r/m = r/m + d8/16 |
|  | 0000010w | data L | (data H) | ac = ac + d8/16 |
|  | | | | |
| SUB | 001010dw | *md*reg r/m | (disp8/16) | r(r/m) = r(r/m) – (r/m)r |
| 100000sw | *md* 101 r/m | (disp8/16)d8/16 | r/m = r/m – d8/16 |
|  | 0010110w | data L | (data H) | ac = ac - d8/16 |
| AND | 001000dw | *md*reg r/m | (disp8/16) | r(r/m) = r & r/m |
|  | 100000sw | *md* 100 r/m | (disp8/16)d8/16 | r/m = r/m & d8/16 |
|  | 0010010w | data L | (data H) | ac = ac& d8/16 |
| OR | 000010dw | *md*reg r/m | (disp8/16) | r(r/m) = r V r/m |
|  | 100000sw | *md* 001 r/m | (disp8/16)d8/16 | r/m = r/m V d8/16 |
|  | 0000110w | data L | (data H) | ac = ac V d8/16 |
| XOR | 001100dw | *md*reg r/m | (disp8/16) | r(r/m) = r + r/m |
|  | 100000sw | *md* 110 r/m | (disp8/16)d8/16 | r/m = r/m + d8/16 |
|  | 0011010w | data L | (data H) | ac = ac + d8/16 |
| MOV | 100010dw | *md*reg r/m | (disp8/16) | r = r/m, r/m = r |
| 1100011w | *md* 000 r/m | (disp8/16)d8/16 | r/m = d8/16 |
|  | 1011wreg | data L | (data H) | reg = d8/16 |
|  | 1010000w | disp L | disp H | ac = m; прямой адрес |
|  | 1010001w | disp L | disp H | m = ac; прямой адрес |
| CMP | 0011101w | *md*reg r/m | (disp8/16) | r – r/m |
|  | 0011100w | *md*reg r/m | (disp8/16) | r/m – r |
|  | 100000sw | *md* 111 r/m | (disp8/16)d8/16 | r/m – d8/16 |
|  | 0011110w | data L | (data H) | ac – d8/16 |
| INC | 1111111w | *md* 000 r/m | (disp8/16) | r/m = r/m+1 |
|  | 01000reg |  |  | reg = reg+1 |
| DEC | 1111111w | *md* 001 r/m | (disp8/16) | r/m = r/m – 1 |
|  | 01001reg |  |  | reg = reg – 1 |
| TEST | 1000010w | *md*regr/m | (disp8/16) | r & r/m |
|  | 1111011w | *md* 000 r/m | (disp8/16)d8/16 | r/m & d8/16 |
|  | 1010100w | data L | (data H) | ac& d8/16 |
| XCHG | 10010reg |  |  | reg↔AX |
|  | 1000011w | *md*reg r/m | (disp8/16) | reg↔r/m |
| JMP short | 11101011 | disp L |  | IP = IP + dispL |
| nearptr | 11101001 | disp L | disp H | IP = IP + dispH,L |
| wordptr | 11111111 | *md* 100 r/m | (disp8/16) | IP = (*EA*) |
| farptr | 11101010 | IP-L | IP-H,CS-L,CS-H | IP = IPH,L, CS =*CSH*,L |
| dwordptr | 11111111 | *md* 101 r/m | (disp8/16) | IP = (*EA*), CS = (*EA*+2) |
| *Условный переход* | \_\_\_\_\_\_ | \_\_\_\_\_\_ | «нет данных» | IP = IP + dispL, если условие выполнено, иначе к следующей команде |
| JZ (JE) | 01110100 | disp L | «нет данных» | ноль (равно) |
| JNZ (JNE) | 01110101 | disp L | «нет данных» | не ноль (не равно) |
| JS | 01111000 | disp L | «нет данных» | минус |
| JNS | 01111001 | disp L | «нет данных» | плюс |
| JO | 01110000 | disp L | «нет данных» | переполнение |
| JNO | 01110001 | disp L | «нет данных» | нет переполнения |
| JL (JNGE) | 01111100 | disp L | «нет данных» | меньше для чисел |
| JNL (JGE) | 01111101 | disp L | «нет данных» | не меньше для чисел |
| JG (JNLE) | 01111100 | disp L | «нет данных» | больше для чисел |
| JNG (JLE) | 01111101 | disp L | «нет данных» | не больше для чисел |
| JB (JNAE,JC) | 01110010 | disp L | «нет данных» | меньше для кодов |
| JNB (*JAE*, JNC) | 01110011 | disp L | «нет данных» | не меньше для кодов |
| JA (JNBE) | 01110010 | disp L | «нет данных» | больше для кодов |
| JNA (JBE) | 01110011 | disp L | «нет данных» | не больше для кодов |
| JP (JPE) | 01111010 | disp L | «нет данных» | четное число «1» |
| JNP (JPO) | 01111011 | disp L | «нет данных» | нечетное число «1» |

*Примечание*. в столбце «Схема операции» ac означает регистр-аккумулятор, в качестве которого используется регистр AX при *w* = 1 и регистр AL при *w* = 0.

При безусловном прямом внутрисегментном переходе новое значение указателя команд IP равно сумме смещения, закодированного в соответствующем поле команды, и текущего значения IP, в качестве которого используется адрес команды, записанной вслед за командой перехода. Команды прямых межсегментных переходов содержат в сeбе помимо нового значения IP и новое значение сегментного регистра CS.

Команды косвенных переходов (внутрисегментных и межсегментных) передают управление на команду, адрес которой определяется содержимым регистра или ячеек оперативной памяти, на которые указывает закодированный в команде перехода постбайт.

Команды условных переходов являются только внутрисегментными. По своему формату и способу формирования нового значения IP они полностью аналогичны команде внутрисегментного прямого перехода с 8-разрядным смещением. Отличие их заключается в том, что в командах условного перехода механизм формирования нового значения IP включается лишь при выполнении определенных условий, а именно, при определенном состоянии регистра флагов. При невыполнении проверяемого условия в IP остается его текущее значение, т. е. адрес команды, следующей за командой условного перехода.

Ниже приведены примеры команд переходов различных типов.

1. Команды условного перехода.

Формат:

КОП

disp L

IP = IP + 2, если условие не выполнено;

IP = IP +2 + disp L, если условие выполнено;

Пример:

JZ MARK; переход на метку MARK, если ZF = 1.

2. Команды прямого внутрисегментного перехода

Формат:

КОП

disp L

disp

IP = IP + ∆ + disp, где ∆ – длина команды перехода (2 или 3 в зависимости от длины смещения).

Примеры:

JMP shortptr MARK; переход на метку MARK, с использованием 8-разрядного смещения;

JMP nearptr MARK; переход на метку MARK, с использованием 16-разрядного смещения.

3. Команды прямого межсегментного перехода

Формат:

КОП

IP\_L

IP\_H

CS\_H

CS\_H

IP = IP\_H, IP\_L,

CS = CS\_H, CS\_L.

Пример:

JMP farptr MARK; переход на метку MARK к команде, находящейся в другом сегменте.

4. Команды косвенного внутрисегментного перехода.

Формат:

КОП

md

disp

disp

IP = [EA + 1, EA]; или

IP = <регистр>, если в постбайте задано обращение к регистровой памяти.

Пример:

JMP wordptr [BX + SI]; новое значение IP берется из двух последовательных байт памяти, эффективный адрес первого из которых определяется суммой регистров BX и SI.

5. Команды косвенного межсегментного перехода

Формат:

КОП

md

dispL

disp

IP = [EA + 1, EA],

CS = [EA + 3, EA + 2].

Пример:

JMP dwordptr [BX + SI]; сумма регистров BX и SI определяет эффективный адрес области памяти, первые два байта которой содержат новое значение IP, а следующие два байта – новое значение CS.

Команды циклов идентичны по формату и очень близки по вы-полняемым действиям командам условных переходов. Однако по сравнению с последними они имеют ряд особенностей, позволяющих эффективно использовать их при программировании циклических участков алгоритмов. Один из наиболее распространенных видов циклического участка программы представлен на рис. 5.20.

да

нет

СЧ = 0

СЧ=<число повторений>

тело цикла

СЧ = СЧ – 1

Рис. 5.20. Структура счетного цикла с постпроверкой

Команды циклов предназначены для упрощения действий декремента (уменьшения на 1) счетчика цикла, проверки условия выхода из цикла и перехода.

Некоторые команды цикла реализуют выход из цикла не только по значению счетчика, но и при выполнении некоторых других условий.

Описание команд цикла сведено в табл. 5.5. За исключением команды JCXZ, которая не изменяет значения регистра CX, при выполнении команд циклов производятся следующие действия:

CX = (CX) – 1.

Затем, если проверяемое условие выполнено, то IP = (IP) + disp8 с расширением смещения знаком до 16 разрядов, в противном случае IP не изменяется, и программа продолжает выполнение в естественном порядке.

Таблица 5.5

**Команды циклов**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Мнемоника | Альтернативная мнемоника | КОП | Проверяемое условие |
| Зациклить | LOOP | — | 11100010 | (CX) = 0 |
| Зациклить пока ноль (равно) | LOOPZ | LOOPE | 11100001 | (ZF = 1)&(CX) = 0) |
| Зациклить пока не ноль (неравно) | LOOPNZ | LOOPNE | 11100000 | (ZF = 0)&((CX) = 0) |
| Перейти по (CX) | JCXZ | — | 11100011 | (CX) = 0 |

6. Архитектура параллельных компьютерных систем

Развитие элементно-конструкторской базы, основанное на физических принципах электроники, всегда (и это диалектически правильно) отстает от требований конкретно решаемых задач. Быстродействия одного компьютера недостаточно для, например, массового обслуживания группы перехватчиков, стартующих против налета баллистических целей. Своевременно пришло понимание того, что только *структурными* методами можно добиться необходимой производительности вычислительных средств. Таким основным структурным методом является *распараллеливание вычислений*.

Сначала стало ясно, что вместо одной ЭВМ в систему управления необходимо встраивать несколько, решающих единый набор функционально связанных задач. Такой коллектив ЭВМ получил название вычислительного комплекса (ВК). Информационная взаимосвязанность задач потребовала применения средств оперативного обмена данными внутри ВК и средств синхронизации. Однако временные затраты на организацию взаимодействия ЭВМ в ВК (накладные расходы) оказались весьма высоки. Возникла идея использования взамен *разделенной* оперативной памяти (ОП), реализованной в ВК, *общей (разделяемой)* ОП, которая связывает несколько центральных процессоров. Проблема затрат времени на обмен оказалась решенной. Так зародилось понятие *параллельной вычислительной системы* (ВС).

Впоследствии принципы распараллеливания стали применяться на всех уровнях разработки ВС: и отдельных устройств, и процессоров. Отметим, что такой способ распараллеливания, как конвейерный, использовался на раннем этапе создания ЭВМ и используется сейчас.

Приоритет в создании ВС на общей памяти принадлежит С. Крею, признанному «отцу супер-ЭВМ», главному конструктору системы CDC 6600, разработанной в 1963 году. Однако условия соперничества и сокрытия информации определили независимость отечественных разработок.

Необходимо отметить тех выдающихся ученых, чьими усилиями осуществлялась не только разработка, но и обоснование эффективности и целесообразности пути создания многопроцессорных ВС, ибо своевременно принятая техническая политика определяет эффективное развитие государства. (К сожалению, известна и негативная политика в области вычислительной техники, отвлекшая значительные технологические ресурсы.) Такими учеными являются: В. М. Бахарев, настойчиво пропагандировавший, продвигавший и руководивший исследованиями в области ВС на общей памяти; В. С. Бурцев и Б. А. Бабаян – конструкторы и ведущие разработчики семейства «Эльбрус», продолжающие борьбу за честь Отечества и поныне; М. А. Карцев – теоретик ВС и разработчик семейства, по его словам, «универсальных спецпроцессоров», объединивших в одном вычислительном устройстве скалярные и векторные операции; Б. А. Головкин, исследовавший эффективность внедрения ВС в сложные управляющие системы.

Какова же цель распараллеливания?

Универсальным критерием, используемым при решении задач распараллеливания, является минимум времени выполнения совокупности работ, распределяемых между процессорами.

Если несколько процессоров составляют ВС, то важной характеристикой ее эффективности (основные составляющие эффективности – производительность, надежность, стоимость) при специализированном использовании (например, в составе автоматизированной системы управления, АСУ) является коэффициент полезной загрузки *k*з. Для его определения находят коэффициенты загрузки процессоров:

**,

где *Ti*, *i* = 1, ..., *n*, – суммарное время занятости каждого процессора решением задачи на всем отрезке полного решения задачи длиной *T*реш([рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/0/#image.B.1)). Тогда

.

Если *P*0 – производительность одного процессора, то реальная производительность *P*BC ВС, состоящей из n процессоров, при решении данной задачи находится следующим образом:

,

где *P*0 определяется классом решаемых задач.

Т3

Т2

Т1

Рис. 6.1. Пример распараллеливания вычислительного процесса

**6.1. Два уровня распараллеливания**

Развитие вычислительной техники характеризуется тем, что на каждом этапе новых разработок требования к производительности значительно превышают возможности элементной базы.

Это обусловлено задачами сложных систем управления в реальном времени, централизованным решением задач в сетях, имитационным моделированием сложных процессов (например, в ядерной физике), оперативным планированием и управлением и решением других задач исследования операций, преодолевающих «проклятие размерности». Такие задачи требуют концентрации вычислительных мощностей, постоянно поддерживая высокую актуальность проблемы создания супер-ЭВМ.

Уже давно стало ясно, что только структурными методами можно уравнять возможности вычислительных средств и требуемые скорости решения на них задач. Под структурными понимают методы распараллеливания работ. К распараллеливанию прибегают при проектировании отдельных устройств ЭВМ – устройств управления, буферов команд, каналов обращения к памяти и модулей памяти, многофункциональных арифметическо-логических устройств (АЛУ), повсеместно применяемых конвейеров и т. д. Но к распараллеливанию же прибегают и в проектировании совместной работы многих процессоров при параллельной или распределенной обработке информации, вводя в обращение термин «вычислительная система (ВС)».

Технический прогресс, несомненно, сказывается на росте частоты работы элементной (элементно-конструкторской) базы, на повышении степени интеграции, но благодаря ему появляются все новые задачи, требующие еще более значительного роста производительности вычислительных средств. Это можно считать законом, приводящим к новым уловкам при совмещении работы устройств ВС, при увеличении их количества в системе, при увеличении их эффективности в процессе решения задач.

Под ***эффективностью работы устройства*** в составе ВС понимают степень его участия в общей работе ВС при решении конкретной задачи – коэффициент загрузки устройства. Распараллеливание работ оправдано, если приводит к существенному росту усредненного по всем устройствам коэффициента загрузки оборудования при решении задач. Это непосредственно сказывается на времени решения. Сегодня говорят не о специальном классе задач, а о задачах, ориентирующих ВС на универсальность, что обусловлено современными областями применения.

Важным революционизирующим моментом стал переход на микропроцессорную элементно-конструкторскую базу, обусловившую построение мультимикропроцессорных ВС.

Сложилось представление о двух основных уровнях, на которых в ВС применяются практические методы распараллеливания:

— на уровне программ, процессов, процедур (первый уровень распараллеливания);

— на уровне команд и операций (второй уровень распараллеливания).

Эти уровни обусловили уровни структуризации ВС на пути превращения ее в супер-ЭВМ. Современным практическим воплощением первого уровня структуризации являются однородные многопроцессорные ВС на общей (разделяемой) оперативной памяти. Они получили название симметричных ВС за обеспечение «равноправия» составляющих модулей. Окончательное признание симметричных ВС положило конец поиску «экзотических» архитектур, эффективных лишь при решении определенных классов задач. Универсальность симметричных ВС, возможность реализации на них любых вычислительных процессов с высокой эффективностью оборудования иллюстрируются многими применениями и анализируются ниже.

Уровень команд и операций наиболее ярко представлен многофункциональными АЛУ и их обобщением – решающими полями, представляющими разделяемый вычислительный ресурс многих процессоров. Некоторые современные проекты ВС в разной степени предполагают такой ресурс. Здесь основная проблема – полная загрузка отдельных исполнительных устройств (ИУ) в процессе выполнения программы.

Различают два способа реализации такой загрузки: динамический и статический.

Динамическая загрузка осуществляется аппаратурой в процессе выполнения программы. Она использует упрощенные алгоритмы распараллеливания.

Статическая загрузка предусматривается при трансляции программы. Транслятор оптимизирует использование оборудования, также решая задачи распараллеливания. Это выражается в формировании «длинных» командных слов, задающих работы устройствам АЛУ в каждом машинном такте.

Основная сложность распараллеливания заключается в соблюдении частичной упорядоченности распределяемых работ. Поэтому решение задач синхронизации параллельного вычислительного процесса сопровождает все усилия по организации совместной работы устройств. Это сказывается и при решении всех задач эффективного использования расслоенной многоуровневой памяти ВС.

Отечественный опыт создания семейства МВК (многопроцессорных вычислительных комплексов) «Эльбрус», модели которого относятся к симметричным ВС, и последующее проектирование позволили проанализировать, разработать и применить ряд существенно новых, важных и перспективных решений в распараллеливании как самого вычислительного процесса, так и работы отдельных устройств. Разработка пронизана такими решениями, они ложатся в основу проектирования развития семейства, являются основой обобщений и дальнейшего исследования возможности применения.

**6.2. Классификация параллельных ВС.**

**6.2.1. Потоки команд и потоки данных**

Общепринятую удачную классификацию ВС предложил в 1970 г. Г. Флин (США). Основным определяющим архитектурным параметром он выбрал взаимодействие *потока команд* и *потока данных* (операндов и результатов).

В ЭВМ классической архитектуры ведется последовательная обработка команд и данных. Команды поступают одна за другой (за исключением точек ветвления программы), и для них из ОЗУ или регистров так же последовательно поступают операнды. Одной команде (операции) соответствует один необходимый ей набор операндов (как правило, два для бинарных операций). Этот тип архитектуры – «***один поток команд – один поток данных», ОКОД (SISD  – "SingleInstruction, SingleData")*** условно изображен на рис. 6.2.

Память

Поток данных после обработки

Поток команд

Поток данных для обработки

Процессор

Рис.6.2.  Вычислительная система типа ОКОД (SISD)

Тип ***ОКМД – «один поток команд – много потоков данных» (SIMD – "Single Instruction – Multiplе Data")*** охватывает ВС, в которых одной командой обрабатывается набор данных, множество данных, вектор, и вырабатывается множество результатов. Это векторные и матричные системы, в которых по одной команде выполняется одна и та же операция над всеми элементами массива – вектора или матрицы, распределенными между *процессорными (обрабатывающими) элементами ПЭ или процессорами.* Принцип обработки показан на [рис.](http://www.intuit.ru/department/hardware/paralltech/1/#image.1.2) 6.3.

ПЭ1

ПЭ2

ПЭ*n*

Память данных

Память программ

Устройство управления

Поток данных для обработки

Поток данных после обработки

Поток команд

……………

Рис. 6.3.  Вычислительная система типа ОКМД (SIMD)

Отечественные векторные ВС – ПС-2000, ПС-2100 допускают организацию матричной обработки. Классический пример матричной архитектуры – ILLIAC-1V (США).

К типу ***МКОД – «много потоков команд – один поток данных» (MISD – "Multiple Instruction – Single Data")*** принято относить векторный конвейер (обычно в составе ВС, чтобы подчеркнуть основной используемый принцип вычислений), например в составе ВС *Crey*-1, «Электроника ССБИС». На векторном конвейере производится последовательная обработка одного потока данных многими обрабатывающими устройствами (ступенями, станциями) конвейера.

К такому же типу относится ВС, реализующая *макроконвейер* (ВС «Украина»). В ней задача, решаемая циклически, «разрезается» на последовательные этапы, закрепляемые за отдельными процессорами. Запускается конвейер многократного выполнения цикла, составляющего задачу.

Принцип обработки показан на рис. 6.4*.*

……………

Память

Потоки команд

Поток данных для обработки

Поток данных после обработки

УУ

1

2

*n*

Рис. 6.4.  Вычислительная система типа МКОД (MISD)

Тип ***МКМД – «много потоков команд – много потоков данных» (MIMD – "Multiple Instruction – Multiple Data")***cоответствует более полному и независимому распараллеливанию. К этому типу относятся, например, все многопроцессорные вычислительные комплексы (МВК) семейства «Эльбрус».

**6.1.2.** «**Фон-Неймановские» и «не-Фон-Неймановские» архитектуры**

Первую ЭВМ создал в 1939 г. в США профессор Джон Атанасов, болгарин, со своим аспирантом К. Берри. Две малые ЭВМ, созданные ими в период 1937–1942 гг., были прототипами большой ЭВМ АВС для решения систем линейных уравнений, которая в 1942 г. доводилась по устройствам ввода-вывода и должна была войти в строй в 1943 г., но призыв Атанасова в армию в 1942 г. воспрепятствовал этому. Проект электронной ЭВМ «Эниак» (*Electronics Numerical Integratorand Computer*) был сделан в 1942 г. Д. Моучли и Д. Эккертом и осуществлен в 1945 г. в Муровской электротехнической лаборатории Пенсильванского университета. В 1946 г. «Эниак» был публично продемонстрирован в работе. В нем впервые были применены триггеры. Рождение Эниак считают началом компьютерной эры, посвящая ему научные симпозиумы и другие торжественные мероприятия (Международный симпозиум, посвященный 50-летию первой ЭВМ, был проведен и в Москве в июне 1996 г.).

Однако еще в начале 40-х г. XX в. Атанасов поделился с Моучли информацией о принципах, заложенных в ЭВМ АВС. Хотя Моучли впоследствии утверждал, что он не воспользовался этой информацией в патенте на Эниак, суд не согласился с этим. Вернувшись из армии после войны, Атанасов узнал, что более мощная ЭВМ «Эниак» уже создана, и потерял интерес к этой теме, не поинтересовавшись, насколько Эниак похож на его ЭВМ АВС.

Известный английский математик Алан Тьюринг был не только теоретиком по информации и теории алгоритмов, автором теоретического автомата «машины Тьюринга», но и талантливым инженером, создавшим в начале 1940-х г. первую работающую специализированную ЭВМ. Эта ЭВМ под названием «Колосс» была сконструирована и сделана им совместно с Х. А. Ньюменом в Блетчи (Англия) и начала работать в 1943 г. Сообщения о ней своевременно не публиковались, так как она использовалась для расшифровки секретных германских кодов во время войны.

Основные архитектурно-функциональные принципы построения ЦВМ были разработаны и опубликованы в 1946 г. венгерским математиком и физиком Джоном фон Нейманом и его коллегами Г. Голдстайном и А. Берксом в ставшем классическим отчете «Предварительное обсуждение логического конструирования электронного вычислительного устройства». Основополагающими принципами ЭВМ на основании этого отчета являются: 1) принцип программного управления выполнением программы; 2) принцип хранимой в памяти программы. Они легли в основу понятия **«*фон-Неймановской*» *архитектуры***, широко использующей *счетчик команд*.

Вернемся к настоящему. Счетчик команд отражает «узкое горло», которое ограничивает поток команд, поступающих на исполнение, их последовательным анализом.

Альтернативной архитектурой является «***не-фон-Неймановская» архитектура***, допускающая одновременный анализ более одной команды. Поиски ее обусловлены необходимостью распараллеливания выполнения программы между несколькими исполнительными устройствами – процессорами. Счетчик команд при этом не нужен. Порядок выполнения команд определяется наличием исходной информации для выполнения каждой из них. Если несколько команд готовы к выполнению, то принципиально возможно их назначение для выполнения таким же количеством свободных процессоров. Говорят, что такие ВС управляются *потоком данных (dataflow).*

В командах проверки условия возможно альтернативное задание адреса результата (ИЛИ – ИЛИ).

Адреса результатов являются адресами ПК, т. е. результаты выполнения одних команд в качестве операндов могут поступать в текст других команд. Команда не готова к выполнению, если в ее тексте отсутствует хотя бы один операнд. Ячейка, обладающая полным набором операндов, переходит в возбужденное состояние и передает в селекторную сеть информационный пакет (токен), содержащий код операции и необходимую числовую и связную информацию. Он поступает по сети на одно из исполнительных устройств.

Там же операция выполняется, и в распределительную сеть выдается результирующий пакет, содержащий результат вычислений и адреса назначения в ПК (возможно, за счет выбора альтернативы, т. е. такой выбор – тоже результат). По этим адресам в ПК результат и поступает, создавая возможность активизации новых ячеек. После выдачи токена в селекторную сеть операнды в тексте команды уничтожаются, что обеспечивает повторное выполнение команды в цикле, если это необходимо.

Общая схема потоковых ВС представлена на рис. 6.5.

ИСПОЛНИТЕЛЬНЫЕ

УСТРОЙСТВА

**. . . . . . . . . . . . .**

КОММУНИКАЦИОННАЯ

СЕТЬ

Распределительная

сеть

Селекторная сеть

ПАМЯТЬ КОМАНД (ПК)

ТОКЕНЫ

**. . . . . . . . . . . . .**

Рис. 6.5. «Идеальная» потоковая ВС

Программа или ее часть (сегмент) размещается в памяти команд ПК, состоящей из ячеек команд. Команды имеют следующую структуру:

{код операции, операнд 1, ..., операнд L,

адрес результата 1, ..., адрес результата M}

Селекторная и распределительная сети образуют ***коммуникационную сеть*** ВС.

Ожидаемая сверхвысокая производительность такой системы может быть достигнута за счет одновременной и независимой активизации большого числа готовых команд при проблематичной бесконфликтной передаче пакетов по сетям и параллельной работы многих исполнительных устройств.

Существует ряд трудностей, в силу которых «не-фон-Неймановские» архитектуры не обрели технического воплощения для массового применения в «классическом», отраженном выше, исполнении. Однако многие устройства используют данный принцип, но чаще всего взаимодействие процессоров при совместном решении общей задачи и их синхронизация при использовании общих данных основаны на анализе готовности данных для их обработки. Это дает основание многим конструкторам заявлять, что в своих моделях они реализовали принцип *dataflow*.

**6.1.3. Системы с общей и распределенной памятью**

***Системы с общей (разделяемой) оперативной памятью*** образуют современный класс ВС – многопроцессорных супер-ЭВМ. Одинаковый доступ всех процессоров к программам и данным представляет широкие возможности организации параллельного вычислительного процесса *(параллельных вычислений).* Отсутствуют потери реальной производительности на межпроцессорный (между задачами, процессами и т. д.) обмен данными ([рис. 6.6, *a*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.5)).

***Системы с распределенной памятью*** образуют вычислительные комплексы (ВК) – коллективы ЭВМ с межмашинным обменом для совместного решения задач ([рис. 6.6, *б*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.5)).

П

С

*n*

1

. . . . . . . .

ЛОП1

Система обмена

1

*n*

Процессоры

1

ОП1

ОП1

ОП*n*

*а*

*б*

Рис. 6.6.  Вычислительная система с общей (*а*) и распределённой (*б*) памятью

В ВК объединяются вычислительные средства систем управления, решающие специальные наборы задач, взаимосвязанных по данным. Принято говорить, что такие ВК выполняют *распределенные вычисления*, а сами ВК называют *распределенными ВК.*

Другое, противоположное воплощение принципа МИМД – *масспроцессорные* или *высокопараллельные* архитектуры, объединяющие сотни, тысячи, десятки тысяч процессоров.

В современных супер-ЭВМ наметилась тенденция объединения двух принципов: общей (распределяемой) и распределенной (локальной) оперативной памяти (ЛОП). Такая структура используется в проекте МВК «Эльбрус-3» и «Эльбрус-3М» ([рис.](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.6) 6.7).

1

*n*

**. . . . . . .**

Коммутатор

**. . . . . . . . . . . . . . .** .

Общая ОП

ЛОП*1*

ЛОП*n*

М1

М*n*

Рис. 6.7.  Схема ВС с модулями локальной памяти

**6.2. Способы межмодульного соединения (комплексирования)**

Различают два противоположных способа комплексирования: с *общей шиной (шинная архитектура)* и с *перекрестной (матричной) коммутацией* модулей ВС (процессоров, модулей памяти, периферии).

На рис. 6.8. представлена система с общей шиной. Шина состоит из линий, по которым передаются информационные и управляющие сигналы.

1

Общая шина

**. . . . . . . . . . . . . . . . . . . . .**

Периферийные устройства

Процессоры

Модули памяти

*n*

1

*m*

Рис. 6.8.  Схема ВС с общей шиной

Шина используется в режиме разделения времени, при котором лишь один модуль в данный момент работает на передачу. Принимать принципиально могут все модули, хотя преимущественно информация при выдаче в нее адресуется. Применяется в микро- и мини-ЭВМ при сравнительно небольшом числе модулей. Практически производится разделение шины на управляющую, адресную и шину данных.

В высокопроизводительных ВС для возможности одновременного обмена многими парами абонентов используется перекрестная или матричная коммутация.

***Матричный коммутатор*** можно представить (прямоугольной) сеткой шин. К одному концу каждой подсоединен источник-потребитель информации ([рис.](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8) 6.9). Точки пересечения – *узлы* этой сетки –представляют собой *управляющие ключи*, которые соединяют или разъединяют соответствующие шины, устанавливая или прекращая связь между модулями. Реализуется связь «каждый с каждым». Одновременно могут связываться многие (до *n*/2) пары модулей.

1

2

*n*

Коммутатор

Управляющие

сигналы

.

.

Процессоры

.

. . .

. . .

*а*

1

2

…

n

1 2 *m*

Процессоры

Коммутатор

Управляющие

сигналы

Модули памяти

*б*

Рис. 6.9.  Матричные коммутаторы:

*а*– перекрестная коммутация процессоров, *б*– коммутация процессоров и модулей памяти

На [рис. 6.9, *а*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8)показана перекрестная связь между процессорами в ВС с распределенной памятью, на [рис. 6.9, *б*](http://www.intuit.ru/department/hardware/paralltech/1/2.html#image.1.8)– между *n* процессорами и *m* модулями ОП.

|  |  |  |
| --- | --- | --- |
| mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания2.mht!http://www.intuit.ru/img/empty.gif | 6.3. МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ И СПОСОБЫ РАСПАРАЛЛЕЛИВАНИЯ   |  | | --- | | **6.3.1. Мультимикропроцессорные вычислительные системы**  В настоящее время выбор сделан в пользу многопроцессорных симметричных ВС типа *MIMD*, обеспечивающих виртуализацию вычислительных ресурсов. Основу такой ВС составляет *суперскалер*, сосредоточивший в себе все способы достижения максимального быстродействия при выполнении одиночной программы. Векторные и векторно-конвейерные процессоры и системы получили свое место. Их эффективность как самостоятельных установок могла быть достаточно высокой только при решении специальных задач и тестов. Поэтому достаточно быстро выяснилось, что эти установки могут выполнять функции интеллектуальных терминалов при решении основной задачи на другом универсальном вычислительном средстве и выполнять лишь отдельные его заявки. Сегодня стало окончательно ясно, что первые эффективны лишь в роли специализированных вычислительных устройств для решения специальных задач. Вторые твердо заняли место в составе многофункциональных арифметическо-логических устройств (АЛУ) суперскалеров, ибо без конвейеров мы не мыслим себе выполнение всех операций ВС.  Складывается и структура памяти ВС, которая может совмещать в одной установке все способы доступа: от разделяемой (общей) до распределенной оперативной памяти. Однако ограниченные возможности эффективной работы с общей памятью часто диктуют иерархическую структуру ВС, где уровни иерархии (кластеры) отличаются или способом доступа к оперативной памяти, или тем, что каждый кластер имеет свою собственную физическую память в общем адресном пространстве. При этом принцип буферизации, основанный на многоуровневой по быстродействию (и, конечно, различной по технологии) памяти, на активном использовании кэш-памяти, продолжает развиваться. Кэш-память, как память самого высокого уровня, претерпевает функциональное разбиение в зависимости от типа данных, для хранения которых она предназначена, либо, в зависимости от вида обработки, – программ или данных.  Все сказанное выше подтверждает перспективность структурных решений при проектировании многопроцессорного комплекса «Эльбрус-3» и его микропроцессорного развития «Эльбрус-3М», «Эльбрус-2К». Таким образом, структура «длинного командного слова» (архитектура *VLIW*, лежащая в основе *EPIC*) попадает в разряд классических.  Сейчас микропроцессор, сконцентрировавший все достижения микроэлектроники, является основной составляющей элементно-конструкторской базы ВС. Поэтому понятие «мультимикропроцессорные ВС» пришло на смену понятию «микропроцессорные ВС».  Анализ современных мультимикропроцессорных ВС позволяет выделить те развиваемые характерные решения, которые в условиях микроминиатюризации и снижения энергоемкости, «экономного» логического развития обеспечивают необходимые свойства универсального применения.  Такими решениями являются следующие.  **6.3.2. Многопроцессорные кристаллы**.  Воспроизведение многопроцессорной ВС на одном кристалле в значительной степени характерно для сигнальных вычислительных средств, специализирующихся на обработке двух- и трехмерных изображений, которые применяются в цифровом телевидении и радиовещании, при передаче изображений по каналам связи и др. Такие средства эффективно используются в качестве нейрокомпьютеров.  Например, на одном кристалле *MVP (MultimediaVideoProcessor)* семейства *TMS 320 C80* (фирма *TexasInstrument*) расположены 4 32-разрядных цифровых сигнальных процессора (*DSP*– DigitalSignalProcessor) с фиксированной запятой (*ADSP-0 – ADSP-3*). Их особенность – высокая степень конвейеризации и до 64 бит длина командного слова для параллельного выполнения нескольких операций. Система команд содержит команды над битовыми полями и структурами данных, несущими графическую информацию. Такая специализация обусловила понятие *DSP-архитектуры*.  Процессоры работают независимо, т. е. ВС типа *MIMD* (Multiple-Instruction, Multiple-Data) программируются отдельно на ассемблере или ЯВУ, данными обмениваются через общую внутрикристальную память.  Каждый из *ADSP* содержит кэш-память команд (2 Кбайта), и через матричный коммутатор *Crossbar* получает доступ к 32 из имеющихся 50 Кбайт быстродействующей статической внутренней памяти. Память расслоенная – поделена на сегменты. Если два и более процессора в одном цикле попытаются обратиться к одному сегменту, аппаратная система управления доступом с циклическим изменением приоритета (*roundrobinprioritization*) позволит сделать это только одному процессору.  32-разрядное АЛУ *ADSP* может работать как два 16-разрядных или четыре 8-разрядных АЛУ. Этого достаточно для обработки видеоизображений. Специальные блоки ускоряют обработку графики. Блоки генерации адресов формируют кольцевые (бесконечные) буферы. Аппаратно поддержаны три вложенных цикла.  *RISC*-процессор управляет четырьмя *ADSP* с помощью диспетчера. Диспетчер и планировщик заданий тесно взаимодействуют с контроллером пересылок. Кроме того, управляющий процессор самостоятельно выполняет вычисления и обеспечивает обмен с внешними устройствами. Он содержит встроенный блок плавающей арифметики и набор векторных операций с плавающей запятой, оптимизированных для обработки изображений, звука и трехмерной графики.  **6.3.3. Транспьютерная технология.**  Представленная выше архитектура обладает такой конструктивной законченностью, которая позволяет как встраивать ее в некоторую систему, так и организовать взаимодействие нескольких кристаллов. Это обеспечивается развитыми средствами связи и обмена данными.  Возможность комплексирования привлекла внимание еще на раннем этапе развития микропроцессоров (в середине 1980-х годов) и привела к построению *транспьютеров* – микропроцессоров, снабженных развитыми средствами комплексирования. Таким образом, создавались «кирпичики», на основе которых можно было создавать сложные структуры. Эта тенденция не только сохранилась, но является необходимым средством построения мультимикропроцессорных ВС.  Преследуя многофункциональность средств обмена, не обязательно требовать их размещения на одном кристалле с центральным процессором. Так, фирма *AnalogDevices* предлагает микропроцессоры *ADSP-21060/62 SHARC* («АКУЛА») для цифровой обработки сигналов, специально предназначенные для комплексирования.  Средства комплексирования «АКУЛЫ»:   * + магистраль для подключения 6 «АКУЛ» и одного ХОСТ-процессора (управляющего, с привилегированным доступом к магистрали, а также к памяти каждого процессора – через специальный порт);   + сигнальные регистры в составе каждого процессора, непосредственно связанные (одной ножкой) с каждым из других процессоров – для контроля их состояния;   + ЛИНКи – каждый процессор имеет 6 выходов (ЛИНКов) для непосредственной связи «процессор – процессор».   ***Общее адресное пространство*** комплексируемых микропроцессоров «АКУЛА» обеспечивает псевдообщую память и исключает необходимость программной организации обмена данными. Если адрес физически принадлежит ОП другого процессора, то обмен организуется автоматически, без вмешательства пользователя (т. е. программно не предусматривается).  ***Межпроцессорный (магистральный) обмен*** инициируется в том случае, если адрес считывания или записи принадлежит адресному пространству другого процессора (единичный обмен). Аналогично возникают групповые пересылки данных с использованием «чужого» адресного пространства.  Пользователь не составляет программу обмена, даже для контроллера обмена данных. Достаточно указать «чужие» адреса.  Процессоры обмениваются сигналами состояния. Поэтому каждый процессор знает, кто является «хозяином» магистрали, т. е. ведет обмен, и свой приоритет в очереди к магистрали. По завершении каждого обмена производится циклическая смена приоритетов процессоров, которым нужна магистраль. Процессор с максимальным приоритетом становится «хозяином». Обмен может прерываться только ХОСТ-процессором.  Микропроцессор утверждается в роли основы элементно-конструкторской базы ВС, и это поняли ведущие разработчики.  В этом смысле привлекает внимание трансформация интересов «отца суперкомпьютеров» С. Крея, который признал определяющую роль принципа *MIMD* при построении ВС *CraySuperserver 6400 System* (*CS640*), выпускаемой корпорацией *CrayResearch* в сотрудничестве с компанией *SUN Microsystems* (сотрудничество с фирмой *SUN* ныне характерно и для ведущих российских разработчиков).  Система предполагает наращиваемую конфигурацию от 4 до 64 процессоров *SuperSPARC*. Принято компромиссное решение на основе классической схемы разделения (общей) ОП при многопроцессорной обработке и распределенной памяти при параллельной обработке массивов. Чтобы работать с частично распределенной памятью в ОЗУ, ВС имеет в любой конфигурации 4 шины. Шина использует сетевую технологию «коммутации пакетов». Это позволяет находить путь обмена единицами информации в соответствии с занятостью или освобождением шин.  В целом, архитектуру следует считать шинной, хотя наличие нескольких шин делает ее промежуточной между шинной и использующей матричный коммутатор.  **6.4. Направление «мини-супер» как поддержка персональнального компьютера**  То, что говорилось выше, «по умолчанию» соответствует разработке супер-ЭВМ, предназначенных для решения особо сложных задач в составе систем управления в реальном времени, моделирования сложнейших физических процессов, решения задач исследования операций, задач искусственного интеллекта, выполнения роли майнфреймов и серверов в локальных, корпоративных и глобальных сетях.  Супер-ЭВМ уникальна, мало тиражируема, цена ее высока.  С другой стороны, ничто уже не может остановить «победного шествия» персональных компьютеров. Область применения их стала всеобъемлющей. Они используются и там, где могут справиться с задачами, и там, где уже не справляются, несмотря на применение современных суперскалеров.  Тогда целесообразно поставить следующую проблему.  Введем в состав персонального компьютера (*РС*), как его внешнее устройство, мультимикропроцессорную систему (мультипроцессор), использование которого в монопольном и однозадачном режиме может обеспечить успешное решение задач повышенной сложности.  Действительно, разрешение этой проблемы позволило бы заполнить определенную нишу между супер-ЭВМ и *PC*, вывести персональный компьютер на уровень мини-супер-ЭВМ. Применение мультипроцессора *РС* в однопрограммном режиме, при жестком распределении памяти, использование (см. далее) прогрессивной технологии «одна программа – много потоков данных» позволяют существенно снизить издержки производительности на работу ОС, легко «врезать» их в современные операционные системы компьютеров. Сборка такой системы должна производиться на основе существующей микропроцессорной элементно-конструкторской базы, с минимальным использованием вновь разрабатываемых элементов.  Здесь воспроизводится упомянутая выше идея о наличии мониторной системы, на которой решается основная задача, и о наличии интеллектуального терминала, который берет на себя функции, обеспечивающие общую эффективность системы.  Общая схема такой установки показана на [рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/2/2.html#image.2.1)0. Выбраны конкретные значения параметров.  Мониторная  система – PC  Память команд  Память команд  ЛОП  ЛОП  КЭШ команд  КЭШ команд  Процессор 2  Процессор 1  Общая шина или коммутатор  Средства синхронизации (семафоры, закрытые адреса, предикаты)  Синхронизатор  Общая разделяемая память  Модуль 1  Модуль 16  Рис. 6.10.  Схема ВС для персонального компьютера  Мультимикропроцессорную приставку к персональному компьютеру целесообразно разработать на основе исследования принципов построения локально-асинхронной архитектуры (*SPMD*-технологии). Важным достоинством архитектуры является сведение традиционных функций ОС на уровень команд. Т. е. система команд мультипроцессора такова, что позволяет реализовать функции управления параллельным процессом, не требуя запуска процедур ОС. Способствует простоте управления параллельным процессом также монопольный и однозадачный режим использования мультипроцессора. Ниже мы подробнее остановимся на принципах *SPMD*-технологии. Предполагая первоначальное знакомство с этими принципами, отметим следующее.  Известно (см. далее), что семафоры – универсальное средство синхронизации. Однако семафоры традиционно используют ОС. Чтобы этого избежать, семафоры следует реализовать с помощью *предикатного механизма*, т. е. с использованием памяти предикатов.  Семафорный механизм может быть эффективно реализован с помощью *механизма закрытия адресов* (памяти закрытых адресов).  Тогда, в общем случае применения семафоров, должны быть введены команды следующего вида.  ***Считать по семафору (Сч(С) А***). Считывание по адресу производится в случае, если указанный семафор (реализованный в памяти регистрового типа, наряду с индексными и базовыми регистрами) открыт. Если семафор закрыт, реализуется ожидание данного ПЭ без прерывания (т. е. в данном применении пользователь может быть допущен к операциям над семафорами типа «жужжать»).  ***Записать по семафору (Зап(С) А***). Запись по адресу производится аналогично предыдущей команде.  При использовании памяти закрытых адресов необходима лишь команда *Закрыть адрес*. Любое последующее считывание по этому адресу циклически возобновляется (в режиме «жужжания») до тех пор, пока по этому же адресу другой процессор не произведет запись.  В случае использования механизма предикатов адрес некоторой булевой переменной записывается в специальные разряды командного слова. Команда, для которой указанный в ней предикат имеет значение 0, выполняется, в соответствии с кодом операции, в *спекулятивном режиме* в двух модификациях:   * ожидается присвоение данному предикату значения 1 (в режиме «жужжания»); * пропускается выполнение данной команды.   ПЭ реализует идею *RISC*-архитектуры и представляет собой функционально законченное устройство, состоящее из микропроцессора, схем обрамления и локальной оперативной памяти (ЛОП). Локальная память процессора содержит область для хранения стеков вычислительного процесса, в том числе стеков подпрограмм и вложенных циклов. В других областях этой памяти хранятся модификаторы, дескрипторы массивов и локальные величины. Здесь же находятся микропрограммы, реализующие систему команд ВС.  Общая (разделяемая) память (ОП) содержит *M* модулей с общим адресным пространством и реализует принцип ***интерливинга***, предполагающий, что смежные ячейки памяти находятся в разных модулях.  ***Синхронизатор*** предназначен для обеспечения одновременного пуска программ или их модулей.  Возможно использование простейших коммутаторов для обмена ПЭ с модулями памяти. | |

**6.5. Распределенный и разделяемый вычислительный ресурс второго уровня. Решающие поля**

Произведем некоторые обобщения.

Итак, второй уровень распараллеливания предполагает распределение команд, инструкций, операций, элементарных функций и других несложных процедур для выполнения исполнительными устройствами процессоров или в общем вычислительном ресурсе симметричной ВС. Здесь существуют свои проблемы, связанные с «элементарным» характером операций, небольшим объемом содержащихся в них работ, с их еще большей критичностью по отношению к «накладным расходам» на организацию и синхронизацию. Мы предполагаем, что исполнительные устройства ВС образуют вычислительный ресурс второго уровня (распараллеливания).

Сложились традиции построения этого ресурса, где основное внимание уделяется построению многофункциональных АЛУ. Однако в ряде архитектур пока еще робко пробивает себе дорогу объединение АЛУ в единый разделяемый ресурс системы – построение решающих полей.

Проработка этой идеи проводилась неоднократно в отечественной практике разработки ВС. Она проявлялась во включении в состав ВС специализированных процессоров на правах интеллектуальных терминалов для эффективного выполнения определенных операций. Это были векторные процессоры с доступом от нескольких ЦП (ПС-3100), векторный сопроцессор в ПС-2100. Эта же идея фактически воплощена в семействе «Эльбрус», допускающем включение в свой состав спецпроцессоров – эмуляции других систем, векторно-конвейерных модулей и др.

При реализации идеи решающего поля проблема выбора и развития вычислительного ресурса неотделима от выбора вариантов архитектуры системы вообще. Единственным средством обоснования и исследования этого выбора является моделирование. Построение детерминированных имитаторов позволяет с любой детализацией выявить целесообразные технические решения и обосновать язык системы. Целью стохастических моделей является оценка эффективности различных архитектур на основе полноты полезной загрузки оборудования.

На основе традиций разработки многопроцессорных симметричных вычислительных систем можно сделать вывод о практике и тенденции развития вычислительного ресурса второго уровня.

Все модели семейства МВК «Эльбрус» предполагают наличие в составе АЛУ процессора нескольких исполнительных устройств (ИУ), специализированных по типам операций. Тогда в целом для ВС можно сказать, что вычислительный ресурс второго уровня является *распределенным и неоднородным* ([рис. 6.1](http://www.intuit.ru/department/hardware/paralltech/2/3.html#image.2.2)1).

УУ

ИУ1

ЦП 1

. . . . .

ЦП N

УУ

ИУ*n*

ИУ1

ИУ*n*

Рис. 6.11. ВС с распределенным решающим полем

Выше говорилось, что использование ресурса второго уровня неотделимо от общих идей функционирования процессоров ВС – от их архитектуры и от архитектуры ВС в целом. Поэтому сказанного о распределенном ресурсе недостаточно, надо говорить и о способе его использования.

Так, в МВК «Эльбрус-2» применена динамическая загрузка ИУ в процессе выполнения последовательности безадресных команд программы, которую подробно рассмотрим в лекции 3. Обобщенный алгоритм такой загрузки основан на промежуточном переводе безадресных команд в трехадресные. Появление адресов аргументов и результатов в каждой команде позволяет на основе совместного анализа нескольких команд, представленных в «окне просмотра», выявлять их частичную упорядоченность и выделять независимые команды для одновременного выполнения. Это адресный метод распараллеливания, который рассматривался в архитектуре *dataflow*. Команды проходят стадии обработки, как показано на рис. 6.12..

Однако проект МВК «Эльбрус-3», породивший микропроцессорное воплощение – МВК «Эльбрус-3М», основан на использовании идеи «длинного» командного слова и управления каждым тактом системы. Динамическое распределение работ между ИУ заменено статическим – предписанием каждому ИУ, что он должен начать делать в данном такте. В «длинном» командном слове в соответствующих позициях записаны инструкции каждому ИУ.

Дополнение снизу «окна просмотра» новой командой

Исключение из «окна просмотра» выполненных команд

Формирование для новой команды признака возможности выполнения на основе совместного анализа адресов команды и адресов выше находящихся команд

Выборка из «окна просмотра» команд, готовых к выполнению, и назначение их на свободные ИУ

Рис. 6.12. Схема оптимизатора-компоновщика «длинных» командных слов

Это означает, что решение проблем оптимального использования ИУ, их синхронизации при выполнении данного алгоритма возлагаются на оптимизирующий транслятор. Он фактически производит диспетчирование, оптимальное планирование параллельного вычислительного процесса на одном процессоре.

**6.6. Способы распараллеливания**

Различают два основных способа распараллеливания: *по управлению* и *по информации*.

**Первый способ** – представление алгоритма задачи в виде частично-упорядоченной последовательности выполняемых работ. Затем в результате диспетчирования реализуется оптимальный план выполнения работ в ВС при ограничениях на время выполнения всего алгоритма или за минимальное время.

Основой является представление алгоритма граф-схемой G, отражающей информационные связи между работами (задачами, процессами, процедурами, операторами, макрокомандами и т. д.), на которые разбит алгоритм. Граф G – взвешенный, ориентированный, без контуров.

Для исследования графа и диспетчирования используют матрицы следования S; их дополняют столбцом T весов — получают расширенные матрицы следования S\* (рис. 6.13).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | T |
| 1 |  |  |  |  |  |  | 2 |
| 2 |  |  |  |  |  |  | 1 |
| 3 | 1 |  |  |  |  |  | 2 |
| 4 |  | 1 | 1 |  |  |  | 3 |
| 5 |  | 1 |  |  |  |  | 2 |
| 6 | 1 |  |  |  |  |  | 2 |

Рис. 6.13.  Исходная информация для распараллеливания

Здесь предполагаем, что ВС – однородная, с общей (разделяемой) памятью, т. е. потерями времени на обмен между работами можно пренебречь.

Пусть ВС содержит два процессора (*n* = 2). Тогда в результате оптимального распределения получим план (рис. 6.14).

0

2

5

6

1

1

3

4

2

1

2

3

4

5

6

7

*t*

Рис.6.14.  Временная диаграмма параллельного выполнения работ

План действительно совпадает с оптимальным, так как длина расписания T = 7, что совпадает с длиной критического пути в графе, Tкр = 7 (путь 1 mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания4.mht!http://www.intuit.ru/img/symbols/srarr.gif3 mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№2%20Микропроцессорные%20системы%20и%20способы%20распараллеливания4.mht!http://www.intuit.ru/img/symbols/srarr.gif4).

В общей схеме организации параллельного вычислительного процесса мы не полностью раскрыли содержание блока 3 – интерпретации потока макроинструкций в виде, удобном для работы диспетчера. Сейчас мы определили, что такой вид – это матрица следования. Значит, в случае необходимости автоматического формирования матрицы следования надо определять информационную взаимосвязь макроинструкций в пределах видимости, т. е. в «окне просмотра». Таким образом, по текущему содержимому «окна просмотра» надо формировать текущий вид матрицы следования.

Вспомним, что мы уже в упрощенном виде решали подобную задачу, например, когда по формируемому потоку трехадресных команд определяли их информационную взаимосвязь и возможность одновременного выполнения этих команд.

Обобщим эту задачу.

Возвращаясь к названной схеме, представим себе, что поток макроинструкций (блок 2) следует через «окно просмотра» так, что для планирования оптимальной загрузки процессоров диспетчер может анализировать некоторое множество этих макроинструкций и из них выбирать вариант назначения их на процессоры для выполнения. Каждая макроинструкция может интерпретироваться и как процедура, где можно выделить имя θμ множество {} входных параметров, множество {βμ} выходных параметров. На рис. 6.15 отображено «окно просмотра», через которое следует поток макроинструкций.

*m*



«окно просмотра»

Рис. 6.15.  Обработка «окна просмотра»

Составим по его содержимому соответствующую матрицу следования размерности *m* × *m*:





По матрице следования *S* диспетчер производит назначение.

После выполнения макроинструкций они исключаются из «окна прросмотра», оставшиеся макроинструкции уплотняются вверх, а снизу «окно просмотра» пополняется новыми макроинструкциями. С учетом вновь поступивших макроинструкций уточняется текущий вид матрицы следования *S* и процесс диспетчирования продолжается.

По такой же схеме, а именно на основе первого способа распараллеливания – по управлению, решается другая важная задача распараллеливания: *компоновки длинных командных слов в оптимизирующем трансляторе*. Назначение работы на ИУ осуществляется здесь в виде записи соответствующей инструкции в позицию длинного командного слова, соответствующую ИУ. т. е. план параллельного выполнения работ (команд, операций) фиксируется в длинных командных словах, в которых предусмотрены инструкции каждому ИУ, которые они должны начать выполнять с данного такта.

**Второй способ распараллеливания** – по информации – используется тогда, когда можно распределить обрабатываемую информацию между процессорами для обработки по идентичным алгоритмам (по одному алгоритму).

**1. Рассмотрим задачу умножения матриц:**





Развернем матрицу — результат *С* — в линейный (одномерный) массив, переименуем ее элементы и заменим два индекса на один:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *c*11 | *c*1*m* | … | *c*1*m* | *c*21 | ... | *c*2*m* | *c*31 | … | *cmn* |
| *d*1 | *d*2 | … | *dm* | *dm*+1 | ... | *d*2*m* | *d*2*m*+1 | … | *dm* |

Пусть ВС содержит *n* процессоров. Выберем следующий план счета элементов матрицы *C*:

процессор 1 считает элементы *d*1, *d*1+*n*, *d*1+2*n*, ...

процессор 2 считает элементы *d*2, *d*2+*n*, *d*2+2*n*, ...

…………………………………………………………..

процессор *n* считает элементы *dn*, *d*2*n*, *d*3*n*, ...

По-видимому, все они будут выполнять одну и ту же программу, но обрабатывать разные наборы данных. (Мы снова столкнулись с целесообразностью *SKMD*-технологии.)

Здесь не потребовалась какая-либо синхронизация параллельного вычислительного процесса.

|  |  |
| --- | --- |
| Рассмотрим задачу счета способом «Пирамиды».  Эту задачу мы исследовали при рассмотрении ВС типа *SKMD*. Посмотрим еще раз, какая синхронизация нам здесь потребуется.  Пусть необходимо перемножить все элементы некоторого массива {*a*1, *a*2, ..., *a*10}. Каждый элемент занимает одну ячейку памяти. Пусть число процессоров в ВС *n* = 4. Чтобы распараллелить этот процесс, примем схему счета «пирамидой» (рис. 6.16) по алгоритму «свертки»  Рис. 6.16.  Граф-схема выполнения операции «свертки»  Количество уровней операций в ней ]log2 *m*[=]log210[=4 (]*x*[ – ближайшее целое, не меньшее *x*).  Расширим массив, дополнив его ячейками, в которых будем хранить промежуточные частные произведения. Тогда весь план счета примем таким, как показано на [рис.](http://www.intuit.ru/department/hardware/paralltech/2/5.html#image.2.8) 6.17. Отмечены процессоры, выполняющие указанную операцию.  Рис. 6.17.  Схема выполнения операции «свертки» четырьмя процессорами  Следовательно, надо так написать программу, одну для всех процессоров, предусмотрев необходимую переадресацию для выборки и вычисления «своих» данных, чтобы по ней выбирались два соседних элемента этого удлиненного массива, а результат их умножения отправлялся в очередную ячейку этого «удлинения».  Возникает только одна трудность: для первых пяти произведений данные есть, а вот последующие произведения должны выполняться тогда, когда для них будут найдены исходные данные.  Значит, процессоры, которым выпало произвести такие умножения, должны «уметь» обнаруживать отсутствие данных и дожидаться их появления. Т. е. требуется синхронизация процессоров по использованию общих данных.  Здесь распараллеливание по данным смыкается с распараллеливанием по управлению.  Возможная схема общей для всех процессоров программы дана на [рис.](http://www.intuit.ru/department/hardware/paralltech/2/5.html#image.2.9) 6.18. Она представлена в примере для типа *SKMD.*  Общая синхронизация ВС  Запрет отображения по считыванию к ячейке, в которую будет помещен результат операции  Выполнение операции, запись результата в «закрытую» ячейку, снятие запрета на использование этой ячейки  Переадресация или выход  ВС   |  | | --- | | Рис. 6.18. Конвейеры операций  Выполнение любой операции складывается из нескольких последовательных этапов, каждый из которых может выполняться своим функциональным узлом. Это легко показать на операциях сложения и умножения. Выполнение деления мантисс (порядки вычитаются) чаще всего производятся с помощью вычитания из делимого делителя, сдвига влево полученного остатка, нового вычитания делителя из результата сдвига и т. д. В некоторых ВС находится обратная величина делителя с помощью аппроксимирующих полиномов. Затем делимое умножается на эту величину.  Пусть задана операция, выполнение которой разбито на *n* последовательных этапов. Пусть *ti* – время выполнения *i*-го этапа. При последовательном их выполнении операция выполняется за время    Выберем *время такта* – величину *t*T = max*ti* и потребуем при разбиении на этапы, чтобы для любого *i* = 1, ..., *n* выполнялось условие *ti* + *t*(*i*+1) mod *n* > *t*T. Т. е. чтобы никакие два последовательных этапа (включая конец и новое начало операции) не могли быть выполнены за время одного такта.  Функциональные узлы, выполняющие последовательные этапы одной операции, целесообразно выстроить в единую конвейерную линию, где устройство, выполняющее некоторый этап, закончив его для операции над одним набором данных, переходило бы в следующем такте к выполнению этого же этапа той же операции для другого набора исходных данных.  Например, на [рис.](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.1) 6.19. представлен конвейер выполнения операции сложения.  Рис. 6.19.  Выполнение операции сложения на конвейере  Загрузка операндов (этап 1)  Выравнивание порядков (этап 2)  Сложение мантисс (этап 3)  Загрузка операндов (этап 4)  Запись результата (этап 5)  Пусть реализуется поток команд одного процессора или существует доступ к этому устройству нескольких процессоров так, что в каждом такте возможно задание на выполнение сложения новой пары чисел. Тогда временная диаграмма работы конвейера может иметь вид, представленный на [рис.](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.2) 6.20.  2  1  3  4  5  1  2  3  1  2  3  4  уровень 1  уровень 2  уровень 3  уровень 4  уровень 5  0  *t*T  2*t*T  3*t*T  4*t*T  5*t*T  6*t*T  …  *t*  1  2  3  4  5  6  2  3  1  2  Рис. 6.20.  Схема заполнения конвейера  Максимальное быстродействие процессора при полной загрузке конвейера составляет    Число *n* – *количество уровней конвейера*, или *глубина перекрытия*, так как каждый такт на конвейере параллельно выполняются *n* операций. Чем больше число уровней (станций), тем больший выигрыш в быстродействии может быть получен.  В проекте МВК «Эльбрус-3» АЛУ его ЦП имеет конвейерные ИУ сложения (*n* = 5), умножения (*n* = 5), деления (*n* = 8 для полусловного формата – 32 разряда, *n* = 16 для словного формата). Логические операции также выполняются на конвейере с *n* = 2.  Известна оценка, т. е. выигрыш в быстродействии получается в *n* - раз.  Реальный выигрыш в быстродействии оказывается всегда меньше, чем указанный выше, поскольку:   1. некоторые операции, например над целыми, могут выполняться за меньшее количество этапов, чем другие арифметические операции. Тогда отдельные станции конвейера будут простаивать; 2. при выполнении некоторых операций на определенных этапах могут требоваться результаты более поздних, еще не выполненных этапов предыдущих операций. Приходится приостанавливать конвейер; 3. поток команд порождает недостаточное количество операций для полной загрузки конвейера.   **6.7. Векторные конвейеры. «Зацепление» векторов**  Наряду с использованием конвейеров для обработки единичных (скалярных) данных используют так называемые *векторные конвейеры*, единичной информацией для которых являются векторы – массивы данных. Применение векторных конвейеров определило класс ВС – *векторно-конвейерных ВС*, сегодня еще являющихся основой построения некоторых супер-ЭВМ – ВС сверхвысокой производительности.  Для эффективности векторно-конвейерных ВС (например, для подготовки алгоритмов решения задач на ВС «Электроника-ССБИС») необходима векторизация задач. Это такое преобразование алгоритма, при котором максимально выделяются (если не вся задача сводится к этому) элементы обработки массивов данных одинаковыми операциями. Сюда входят все задачи, основанные на матричных преобразованиях, обработка изображений, сигналов, моделирование поведения среды и т. д.  В основе векторного конвейера лежит то же самое разбиение операции на уровни или этапы выполнения, но он дополняется средствами аппаратной поддержки, позволяющими по информации о векторах организовать последовательную загрузку конвейера элементами векторов, учитывая их длину.  Пусть необходимо выполнить операцию *C* = *A* × *B*, т. е. *cj* = *aj* + *bj*, *j* = 1, ..., *N*. Пусть на регистрах СОЗУ записаны векторы *A* и *B*. Группа регистров отведена для результатов *C*. Для управления этим процессом известны дескрипторы векторов *DA* , *DB* , *DC*, где *D*α = {*a*α, *h*α, *N*}, α = *A*, *B*, *C*, *h*α – шаг переадресации. Если загрузка векторов производится всегда в одни и те же регистры АЛУ, то достаточно знать значение *N*. Может задаваться маска *M* длиной *N*, состоящая из нулей и единиц. Каждый элемент *M* соответствует элементу вектора-результата *C*. Если элемент *mj* = 1 (логическая переменная), то операция получения *cj* производится, в противном случае соответствующие элементы векторов пропускаются. Это применимо для альтернативного счета в соответствии со значением логических переменных.  Пусть операция умножения выполняется за три этапа. Тогда можно представить временную диаграмму получения *N* результатов при предположении о назначении функциональных устройств (рис. 6.21)*.*  В составе АЛУ может быть два и более конвейерных устройств, специализированных каждое для выполнения некоторой операции. Тогда возможно и эффективно ***«зацепление» векторов***, иллюстрируемое примером на [рис.](http://www.intuit.ru/department/hardware/paralltech/3/#image.3.4) 6.22 для выполнения сложной операции над векторами: *D* = *A* × *B* + *C*.  Рис.6.21.  Умножение векторов на конвейере  *N*–3  *N*–2  *N*–1  *N*–2  *N*–1  *N*  *N*  1  2  3  Загрузка очередной пары (*a*, *b*), переадресация или остановка  Первый уровень сумматоров  Второй уровень сумматоров  Третий уровень сумматоров  Запись *c*j  1  2  3  4  5  6  1  2  3  4  5  1  2  3  4  1  2  1  2  3  4  5  6  …………  *N*  *N*+1  *N*+2  *N*+3  *N*+4  *N*  *N*–1  *N*  Рис. 6.22. «Зацепление» векторов  1  2  *n*x  …….  1  2  *n*x  …….  A  B  C  Здесь два конвейера образовали один с глубиной перекрытия *n*  =  *n*× + *n*+. Очередной результат умножения немедленно направляется на конвейер сложения, куда параллельно направляется необходимый сомножитель. |   mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств1.mht!http://www.intuit.ru/img/empty.gif |

|  |
| --- |
| **6.8. Выполнение операций на стеке**  В вычислительной технике повсеместно используется такая структура данных, как стек. Его применение при выполнении арифметических и логических операций в арифметическо-логическом устройстве (АЛУ) позволяет реализовать безадресную систему команд, что, в свою очередь, дает возможность минимизировать число обращений к оперативной памяти, осуществить буферизацию при многоуровневой памяти, кодировать большое число команд в одном слове.  Рассмотрим механизмы обработки информации на стеке и формирования безадресных команд.  Анализируя структуру команд, мы можем отметить, что практически команды бывают одно-, двух-, трехадресные. Трехадресная команда, как правило, по законченности соответствует одному оператору, так как отображает действие над двумя операндами и результат. Двухадресная команда – это такая команда, где один из адресов подразумевается (например, аккумулятор или сумматор), или результат направляется по одному из указанных адресов. То же касательно одноадресной команды: по числу адресов – число обращений к памяти, не считая считывания самой команды.  Возникает вопрос: можно ли отделить загрузку регистров сверхоперативного запоминающего устройства (СОЗУ) от собственно выполнения команды, сделать эту загрузку опережающей, выполнение команды производить только с использованием подразумеваемых адресов СОЗУ и при этом в целом минимизировать количество обращений к ОП? Ответ приводит к структуре безадресных команд и к выполнению операций на стеке.  ***Стек*** представляет собой множество последовательно пронумерованных регистров СОЗУ или ячеек ОП, снабженное ***указателем вершины стека УВС***, в котором автоматически при записи и считывании устанавливается номер (адрес) последнего занятого регистра – ***вершины стека***. При записи в стек (загрузка стека) слово пишется в следующий по номеру регистр, а УВС увеличивается на единицу. При считывании извлекается слово с адресом, указанным в УВС. Затем УВС уменьшается на единицу. Таким образом, в стеке реализуется правило «последний пришел – первый ушел».  Адреса памяти фигурируют только в командах загрузки стека и записи из стека в память.  Значения выражений можно вычислять полностью безадресным способом.  Команда, состоящая только из кода операции, извлекает из стека один или два операнда, выполняет операцию и заносит результат в стек. На рис. 6.23 приведен пример преобразования стека при выполнении трех команд программы.    Рис. 6.23.  Выполнение команд на стеке  2  2  2  ……  17  3  25  3  ……  4  17  3  2  ……  21  3  25  1  ……  3  25  25  УВС:  УВС:  УВС:  УВС:  2  1  0  0  1  1  3  0  1  0  Команда: Загрузка стека из R, (R)=4  «+»  Записать в A  Вычисления с использованием стеков удобно задавать, описывать и программировать с помощью ***польской инверсной (бесскобочной) записи*** арифметических выражений ***ПОЛИЗ*** (предложил польский математик Я. Лукашевич в 1929 г.). Запись производится по правилу «читаем арифметическое выражение слева направо и последовательно друг за другом выписываем встречающиеся операнды». Как только окажется, что все операнды некоторой операции выписаны, записываем знак этой операции, условно учитываем вместо операндов и знака операции потенциальное имя ее результата. Это может породить возможность записи новой операции; или же будем продолжать выписывать операнды, если условно учтенный результат не позволяет записать знак операции. Запись на ПОЛИЗ состоит из цепочек имен и цепочек операций.  Например, y := (*k* + (*l* – *mn*))(*a* – *b*) mhtml:file://F:\INTUIT_ru%20Курс%20Архитектура%20__%20Лекция%20№3%20Распараллеливание%20в%20ВС%20на%20уровне%20исполнительных%20устройств2.mht!http://www.intuit.ru/img/symbols/srarr.gify*klmn* × – + *ab* – ×  Последовательность символов в ПОЛИЗ может рассматриваться как программа вычисления значения данного выражения, если под именами понимать команды загрузки стека (кроме случая, когда имени следует знак операции :=, тогда это имя определяет запись из стека), а под знаками операций – безадресные команды, содержащие только коды операций. Удобно ввести команду вида Зп Y– записи из стека по адресу Y. Тогда мы получим окончательный вид программы  *klmn* × – + *ab* - × Зп Y  Не будем отображать УВС, интерпретируя стек магазинной памятью. Тогда схема выполнения данной программы сводится к представленной на рис.  6. 24.  *n*  *m*  *l*  *k*  *m*x*n*  *l*  *k*  ……..  *l*– *mn*  *k*  …….  *k*+ (*l*– *mn*)  ……..  *b*  *a*  *k*+ (*l*– *mn*)  ……….  *a* – *b*  *k*+ (l - *mn*)  ………..  результат  ………..  A  Рис. 6.24.  Выполнение программы на стеке |

|  |
| --- |
| 6.9. Распараллеливание в BC на уровне исполнительных устройств |

Очевидно, что одновременно или с перекрытием во времени могут выполняться такие две команды, каждая из которых не использует в качестве операнда результат выполнения другой. Если же команда меняет значение некоторой величины, то предшествующие команды в «окне просмотра» должны раньше выполнения данной команды успеть использовать предыдущее значение. При этом должна соблюдаться последовательность присваивания значений одной величине.

Пусть α*j* — логическая переменная (признак), соответствующая возможности назначения *j*-й команды для выполнения. Тогда в процессе назначения команд на ИУ *j*-я команда может быть назначена для выполнения в соответствии со значением истинности предикатов.

При наличии незанятых ИУ необходимой специализации или при наличии свободных регистров в буферах этих ИУ, команды с признаком α*j*=1 назначаются для выполнения. Так, в АЛУ процессора МВК «Эльбрус-2» каждое специализированное конвейерное ИУ имеет небольшой буфер, в которые записываются назначенные команды.

Назначенные команды снабжаются признаком назначения β*j* и не исключаются из «окна просмотра» до сигнала ИУ о завершении выполнения. Это необходимо для исключения преждевременного начала выполнения других команд, использующих результат данной.

Легко видеть, что в каждой из одновременно выполняющихся команд, на разных ИУ, в данном случае используются адреса СОЗУ, которых нет ни в одной из других команд. Т. е. одновременно работающие ИУ используют непересекающиеся множества адресов. Это исключает конфликты обращения к регистрам СОЗУ, ускоряет работу. В таком случае говорят, что ИУ *полностью сегментированы*. Хотя в общем случае работы на общих вычислительных ресурсах (решающих полях) это необязательно.

**Пример.** Пусть «окно просмотра» вместило в себя весь фрагмент ранее рассмотренной программы. АЛУ содержит необходимые устройства сложения, умножения и деления, а также два одновременно работающих канала обращения к ОП. Последовательное преобразование содержимого «окна просмотра» показано на [рис. 6.](http://www.intuit.ru/department/hardware/paralltech/3/4.html#image.3.9)25.

Рис. 7.1.  Распределение работ между исполнительными устройствами

+

Сч

Сч

×

:

3n

r1

<*m*>

<v>

r4

r3

r1

r2

r5

r6

r3

r4

r5

r6

r1

<*A*>

1

1

1

0

0

0

1

1

1

0

0

0

θ*j*

Сч

Сч

+

Сч

Сч

×

:

3n

A*j*1

<*a*>

<*b*>

r1

<m>

<*n*>

r4

r3

r1

A*j*2

r2

r5

r6

A*j*3

r1

r2

r3

r4

r5

r6

rl

3n

 α

1

1

0

1

1

0

0

0

β

1

1

0

0

0

0

0

0

x

:

3n

r4

r3

r1

r5

r6

r6

r1

<*A*>

1

0

0

1

0

0

:

3n

r3

r1

r6

Q

r1

<*A*>

1

0

1

0

Данный способ распределения команд, потребовавший промежуточного перевода безадресных команд в трехадресные, эффективен в случае многофункционального АЛУ. Универсальные исполнительные устройства, образующие АЛУ, — решающие поля, могут потребовать разработки других способов распределения работ.

Рассматривается проблема параллельного выполнения программы, которая представлена в безадресной системе команд, имитирующей выполнение операций на стеке. Параллельная обработка стека применима к программам счёта арифметических операторов, содержащих условия. Это позволяет значительно сократить количество условных переходов.

6.10. Параллельная обработка стека и статическое распараллеливание в решающем поле

6.10.1. Подстеки и их взаимодействие

В процессорах супер-ЭВМ используются многофункциональные АЛУ, состоящие из специализированных по операциям исполнительных устройств. Широкое применение микропроцессоров позволяет реализовать в составе АЛУ *решающие поля* на основе универсальных исполнительных устройств, которые могут выполнять последовательности команд или целые процедуры. Рассмотрим возможность динамического распараллеливания в таких АЛУ при выполнении арифметических операторов программы в безадресной системе команд процессора, воспроизводящей выполнение работ на стеке.

Существуют пути обобщения такой структуры ВС на основе комплектации многопроцессорных ВС на общем вычислительном ресурсе, – решающем поле. Такая структура в наибольшей степени адекватна концепции двух основных уровней распараллеливания: уровня программ и уровня команд.

Процессор в обычном смысле выполняет функции устройства управления – управления выполнением программы. Исполнительные команды выполняются процессорными элементами (ПЭ) решающего поля РП, которые составляют общий вычислительный ресурс ВС. Т. е. РП выполняет заявки процессоров на выполнение операций. При этом порядок использования ПЭ определяется динамически в соответствии с потоком команд программы и с возможностями их параллельного выполнения.

Однако при такой структуре и универсальности ПЭ целесообразно распределять между ними не отдельные команды, а группы команд или законченные операторы. И принимать решение об этом можно на основе анализа бесскобочной записи (польской инверсной записи – ПОЛИЗ) арифметических операторов программы, или, что то же самое, на основе потока безадресных команд программы. Выясняется, что производить распределение (распараллеливание) можно, минуя промежуточный перевод команд из *безадресной формы* в трехадресную, – непосредственно по тексту программы.

Тогда если представить, что программа в ПОЛИЗ предполагает ее непосредственное выполнение на стеке, то, следовательно, производится распараллеливание обработки стека. Работа одного стека сводится к параллельной работе нескольких взаимосвязанных *подстеков* данного стека. Каждый подстек реализуется на стеке выделенного для этого ПЭ. Если предположить, что данные находятся в СОЗУ и не подлежат перемещению при организации совместной работы ПЭ, то для их обработки целесообразно организовать в локальной регистровой памяти каждого ПЭ адресные стеки, как это рассматривалось выше.

Процессорный элемент и его адресный стек со своим окружением показан на рис. 6.26.

Рассмотрим арифметическое выражение

*A* ꞉꞊ *a* - (*b* ∙ (*c* + *d*) - *e* / *f*) / (*g* ∙ *h* ∙ *i*)

Его бесскобочная запись

*Aabcd* + ∙ *ef* : - *gh* ∙ *i* ∙ : - :=.

Указатель вершины стека

Ссылка на левый (или два левых) подстек

Цепочка (список) операций

Адресный стек

Рис. 6.26. Подстек

Запишем программу, произведя очевидное оптимизирующее преобразование, сокращающее количество цепочек имен и операций:

*abcd* + × *ef* : - *ghi* × × : - Зп*А*.

Составим информационный граф *G*, соответствующий порядку выполнения операций на стеке при счете значения этого выражения (рис. 6.27).

Строить этот граф будем в порядке выполнения операций. Сначала изобразим вершины *a*, *b*, *c*, *d* в соответствии с вызовом их в стек. Затем изобразим вершину +, соответствующую сложению *c* и *d*. Затем вершину ×, соответствующую умножению *b* на *c*+ *d*. Так как цепочка операций закончилась, изобразим вершины *e* и *f* и т. д. В результате последовательных действий развернется граф *G*, иллюстрирующий параллельную структуру алгоритма счета значения данного выражения. Выделены подструктуры, которые могут выполняться параллельно.

Важный вывод на основе анализа графа: можно независимо (и параллельно) извлекать из памяти величины по всем именам, составляющим цепочки имен в ПОЛИЗ. Т. е. можно их заблаговременно считывать в любой последовательности, в том числе и параллельно. Однако при возможности последовательного обращения к памяти предпочтительным является анализ цепочек имен слева направо, а каждой цепочки – справа налево, так как использование величин в операциях производится справа налево (в нижеследующих примерах загрузка стеков производилась традиционно.)

Рис. 6.27.  Граф-схема счета арифметического выражения

Тогда разовьем стековый механизм, позволяющий производить распараллеливание вычислений.

По числу цепочек имен (и операций) в ПОЛИЗ сформируем несколько подстеков, каждый из которых имеет свою вершину. Подстеки, сформированные для счета одного выражения, упорядочены так, что каждый из них может обладать *ссылкой* на свой *правый* подстек (кроме последнего). Левый по отношению к данному подстек образуется цепочкой имен, находящейся в бесскобочной записи непосредственно левее цепочки имен, соответствующей данному подстеку. Каждому подстеку ставится в соответствие следующая за цепочкой имен цепочка операций и разрешается ее выполнение. На [рис. 6.28, *а*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3) показаны сформированные подстеки и их начальная загрузка (не по стековому принципу – последовательно через вершину, а по возможности параллельно или с перекрытием во времени).

Рис. 6.28.  Взаимодействие подстеков

*d*

*c*

*b*

*a*

Ссылка 2

*f*

*e*

Ссылка 3

*i*

*h*

*g*

Ссылки нет

*c*+*d*

*b*

*a*

Ссылка 2

*e*:*f*

Ссылка 3

*h*×*i*

*g*

Ссылки нет

*b*(*c*+*d*)

*a*

Ссылка 2

*e*:*f*

Ссылка 3

*g*×*h*×*i*

Ссылки нет

*b*(*c*+*d*)-*e*:*f*

*a*

Ссылка 3

*g*×*h*×*i*

Ссылки нет

*a*

*б*

*в*

*г*

1

2

3

1

2

3

1

2

3

1

3

+×

:-

× ×:-

×:-

-

×

:-

Определим возможный вариант реализации: подстеки могут заполняться не соответствующими величинами, а их адресами в СОЗУ. Т. е. традиционные приемы использования кэш-памяти должны быть применены здесь.

Взаимосвязь подстеков следует из неравенства длин цепочек имен и соответствующих им цепочек операций. Возможны следующие ситуации:

1. *подстек выродился* в свою вершину, но цепочка операций оказалась не исчерпанной при том, что первая из этих операций – двуместная; такой подстек назовем *не полностью вырожденным*;
2. цепочка операций исчерпалась, но на подстеке содержится одна или более величин (или их имен, адресов), – такой подстек назовем *вырожденным*; если на подстеке находится единственная величина, составляющая вершину, то такой стек назовем *полностью вырожденным.*

Если данный подстек не полностью вырожденный, то последующая операция возможна над вершиной этого подстека и вершиной левого подстека, если тот является вырожденным или полностью вырожденным.

Если левый подстек также не полностью вырожденный или выполнение на нем возможных операций не закончилось, то данный подстек находится в состоянии ожидания. Для выполнения операции над вершинами двух подстеков (если правый подстек является вырожденным) правый подстек переводится на данный подстек так, что его вершина становится новой вершиной данного левого подстека. Ссылка правого подстека заменяет собой ссылку левого подстека. Таким образом, на основе двух подстеков формируется один и появляется возможность дальнейшего выполнения операций из цепочки.

При правильной бесскобочной записи каждый не полностью вырожденный подстек дождется возможности продолжения выполнения операций за счет вырожденного или полностью вырожденного левого подстека. Он, в свою очередь, превратится в вырожденный или полностью вырожденный и пополнит левый не полностью вырожденный, если такой имеется. Окончательное значение данного выражения образуется в вершине единственного оставшегося полностью вырожденного подстека, не содержащего ссылку на правый подстек.

Продолжим рассмотрение примера, для простоты считая одинаковым время выполнения всех операций.

На первом шаге (в первом такте) выполняются операции на всех подстеках ([рис. 6.28, *б*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)). После этого подстек 2 оказывается не полностью вырожденным; в дальнейшем он будет ждать вырождения левого — подстека 1.

После выполнения операций на втором шаге ([рис. 6.28, *в*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)) подстек 1 оказывается вырожденным, его дополняет подстек 2 со своей ссылкой на третий подстек, который становится правым для первого. Подстек 3 оказывается не полностью вырожденным, он ждет вырождения своего левого подстека.

После выполнения операций на третьем шаге ([рис. 6.28, *г*](http://www.intuit.ru/department/hardware/paralltech/4/#image.4.3)) подстек 1 оказывается вырожденным; он дополняется подстеком 3.

После выполнения операций на подстеке 1 на четвертом и пятом шагах получается окончательный результат.

При произвольных временах выполнения операций динамическая картина меняется, но правила взаимодействия подстеков те же.

Таким образом, все подстеки образуют очередь заданий, которые распределяются между операционными или исполнительными универсальными устройствами – ПЭ для выполнения. ПЭ действительно образуют распределяемый ресурс системы, в общем случае –многопроцессорной, и назначаются динамически по необходимости. Реализуется концепция виртуальных исполнительных устройств или процессорных элементов решающего поля.

Усложним пример, введя в рассмотрение арифметические операторы, содержащие условие, а также образующие сложные конструкции с использованием условий.

Пусть необходимо распараллелить счет арифметического выражения

Y:=(*a*+*e*:*f*)× if *a*×l≤*c* then if *l* <*g*×*h* then A else A+*l* else *c*×(*h*+16).

Предварительно необходимо распространить правила формирования ПОЛИЗ для отображения условий. Целесообразно оставлять на месте разграничителиif, then, else. Знак операции, в которой участвует условное выражение, необходимо предпосылать каждому альтернативному оператору.

Процессор, как указывалось ранее, выполняя функции лишь устройства управления, производит последовательный анализ символов программы. Обнаружив цепочку имен, он выделяет ПЭ и настраивает его на обработку этой цепочки и последующей цепочки операций. При нахождении каждой следующей цепочки имен, если анализ производится вне условного оператора, он формирует ссылку «левого» подстека на свой «правый». Однако после формирования последнего (а возможно – единственного) подстека оператора if он формирует ему ссылку на два подстека: первого подстека оператора then и первого подстека оператора else – после их анализа.

**7. Практикум**

**7.1. Исследование логических элементов в среде схемотехнического моделирования Multisim**

**7.1.1. Среда схемотехнического моделирования Multisim**

Наряду с физическими (реальными) экспериментами в настоящее время широкое распространение получило компьютерное проектирование и анализ цифровых устройств в среде схемотехнического моделирования NI Multisim, разработанной группой Electronics Workbench (входящей в корпорацию National Instruments). Особенностью программной среды Multisim является наличие в ее библиотеке более 16 000 электронных компонентов, а также наличие виртуальных контрольно-измерительных приборов, которые по характеристикам приближены к их промышленным аналогам.

Multisim состоит из редактора схем и подсистемы моделирования, базирующейся на интеграции вычислительных ядер SPICE3F5 (BSpice) и XSpice. Пакет MCU позволяет включать в эмуляцию смешанной схемы определенные микроконтроллеры.

Программа Multisim имитирует реальное рабочее место в исследовательской лаборатории, которое оборудовано измерительными приборами: генераторами, мультиметрами, осциллографами, анализатором спектра, измерителем АЧХ и ФЧХ, измерителем нелинейных искажений, преобразователем и анализатором логических сигналов и др.

Multisim является программой с многооконным графическим интерфейсом, позволяющим строить и редактировать схемы, модели и изображения компонентов, а также представлять результаты расчетов в удобном графическом виде.

Пользовательский интерфейс программы показан на рис. 7.1 и состоит из следующих элементов: строка меню, панель инструментов, панель разработки, окно редактирования, приборная панель.

Пользовательский интерфейс Multisim можно настроить на свой вкус, изменения зависят друг от друга. Панели инструментов можно закрепить в любом месте и изменить их форму. Инструменты всех панелей также можно изменять и создавать новые панели. Система меню также полностью настраивается, вплоть до контекстных меню разных объектов.

Рассмотрим некоторые модели контрольно-измерительных приборов, вынесенные на приборную панель.

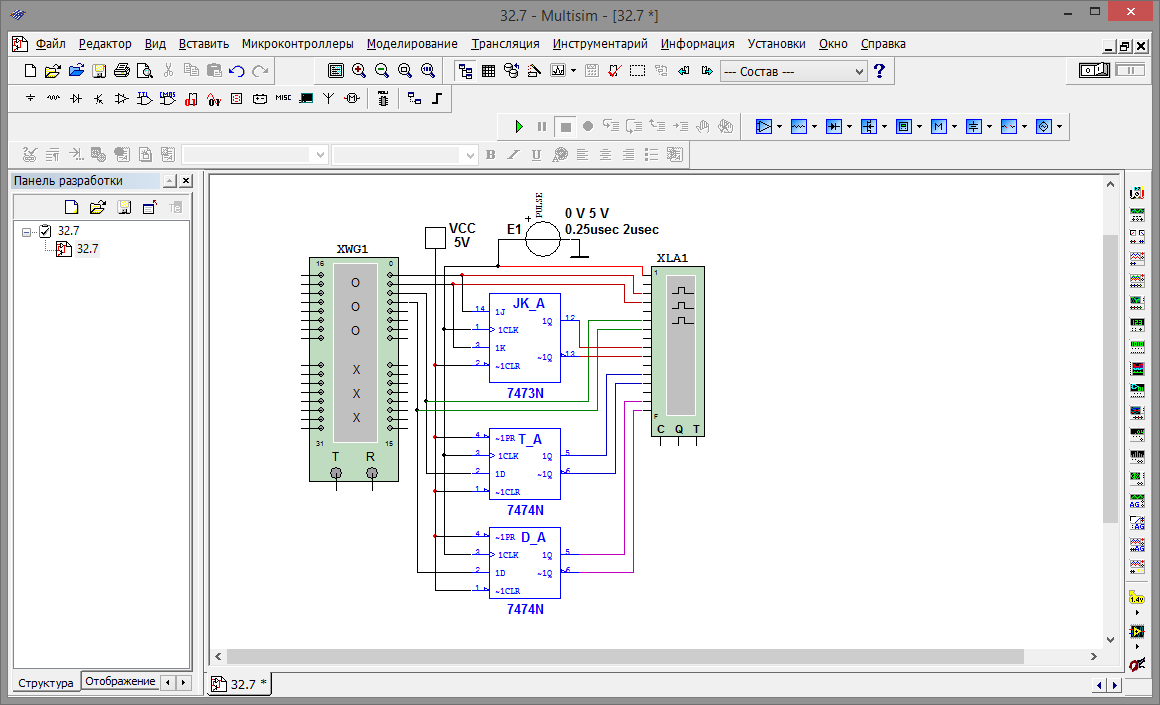


Рис. 7.1. Интерфейс программы Multisim

Панель инструментов

Панель разработки

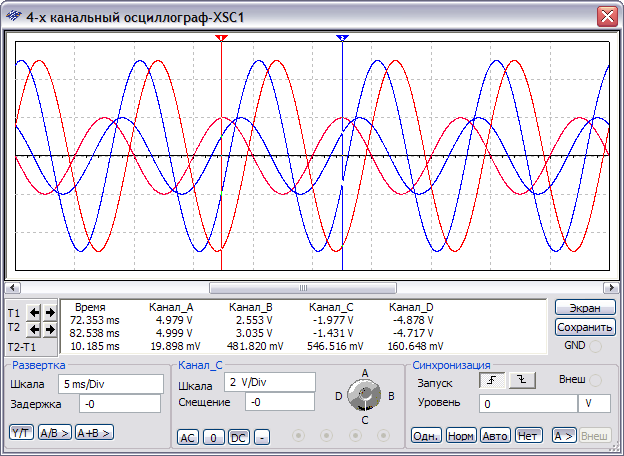
Окно редактирования

Приборная панель

Строка меню

**7.1.2. Контрольно-измерительные приборы Multisim**

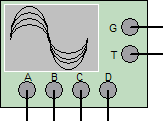
**Осциллограф (Oscilloscope)** позволяет проводить анализ сигналов во временной области. Внешний вид и лицевая панель осциллографа показаны на рис. 7.2.



Входные каналы

G – заземление

T ­– синхронизация



Панель управления

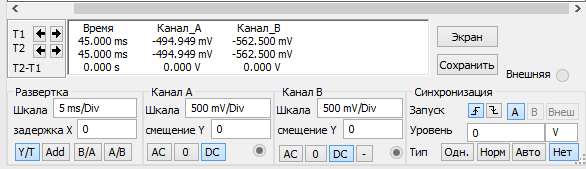
Рис. 7.2. Четырехканальный осциллограф

Модель имеет четыре канала A, B, С и D с раздельной регулировкой чувствительности в диапазоне от 10–15 до 1015 В/дел, а также регулировкой смещения по горизонтали и по вертикали.

Для настройки отображения измеряемого сигнала используется панель управления осциллографа (рис. 7.3). Выбор режима работы по входу осуществляется нажатием кнопок «AC», «0», «DC», «-» (инверсный режим – только для канала B). Режим AC равносилен введению емкостного фильтра в цепь сигнала, при этом отображается только переменная составляющая сигнала. В режиме 0 входной канал замыкается на землю. В режиме DC отображаются обе составляющие сигнала. В инверсном режиме сигнал инвертируется относительно положения нуля.

Для выбора режима развертки используются кнопки Y/T, A/B >, A+B >. В режиме Y/T реализуется временная развертка для каждого канала, т. е. горизонтальная ось представляет собой ось времени, а сигналы каналов A, B, С, D отображаются по вертикальной оси. Длительность развертки задается в поле Scale (Шкала) параметра Timebase и варьируется в диапазоне от 10–15 до 10+15 с/дел. В режиме A+B > отображается суммарный сигнал по двум выбранным каналам. Для построения передаточной характеристики исследуемой схемы используется режим A/B >.

|  |  |  |  |
| --- | --- | --- | --- |
| Временной сдвиг  между курсорами | Показания  курсора Т1 | Показания  курсора Т2 | Изменения фона |



Режим развертки

Амплитудный сдвиг между курсорами

Синхронизация

Рис. 7.3. Панель управления осциллографа

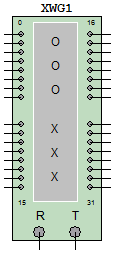
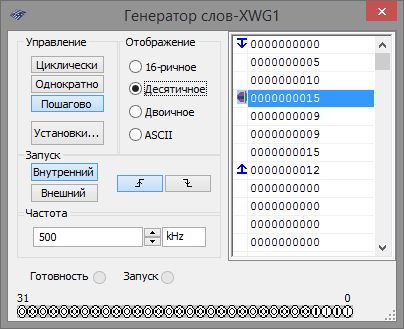
**Генератор слов (Word Generator)** предназначен для генерации до 8192 32-разрядных двоичных слов. Внешний вид и лицевая панель генератора показаны на рис. 7.4. Ввод генерируемых слов производится в буфере ввода. Формат отображения кодовых слов выбирается c помощью группы кнопок Display (Hex – шестнадцатеричный, Dec – десятичный, Binary – двоичный, ASCII – символьный код). Частота генерации кодовых слов задается в окне Frequency (Частота) и лежит в диапазоне от 1 Гц до 1000 МГц. В процессе работы на каждом выводе генератора появляется логический уровень согласно разряду двоичного кодового слова, при этом генератор работает в трех режимах:

* Step (Пошаговый) – каждый раз при подаче очередного слова на выход моделирование останавливается;
* Burst (Пакетный) – генерируется последовательность кодовых слов, начиная с начальной  позиции и заканчивая конечной позицией, моделирование останавливается при достижении конечной позиции;



* Cycle (Циклический) – на выводах генератора последовательно появляются логические уровни согласно комбинации слов, генерирование осуществляется до тех пор, пока не будет остановлено моделирование или достигнута точка прерывания (Breakpoint).

|  |  |  |
| --- | --- | --- |
| Старший 16 бит | Младший 16 бит | Буфер вывода |

** **

|  |  |  |
| --- | --- | --- |
| Внешняя  синхронизация | Сигнал готовности | Выводы генератора |

Рис. 7.4. Генератор слов

Во время моделирования курсор  в окне «Буфер вывода» указывает на текущее генерируемое слово. Остановив моделирование, можно изменить положение курсора, начальную позицию, конечную позицию, а также точку прерывания.

При нажатии кнопки «Установки» открывается диалоговое окно свойств буфера (рис. 7.5):

* Без изменений – оставить без изменений;
* Загрузить – загрузить кодовые слова из файла;
* Сохранить – сохранить кодовые слова в файл;
* Очистить буфер – обнулить содержимое буфера;
* Вверх – заполнить буфер кодовыми словами, начиная с кода, указанного в поле «Инициализировать конфигурацию» (по умолчанию 0×0000), с последующим увеличением на 1 в каждой следующей строке;
* Вниз – заполнить буфер кодовыми словами, начиная с кода, указанного в поле «Инициализировать конфигурацию» (по умолчанию 0×0400), с последующим уменьшением на 1 в каждой следующей строке;
* Вправо – заполнить буфер кодовыми словами, начиная с кода, указанного в поле «Инициализировать конфигурацию» (по умолчанию 0×80000000), с последующим двоичным сдвигом вправо на 1 разряд в каждой следующей строке;
* Влево – заполнить буфер кодовыми словами, начиная с кода, указанного в поле «Инициализировать конфигурацию» (по умолчанию 0×0001), с последующим двоичным сдвигом влево на 1 разряд в каждой следующей строке. Запуск генератора может синхронизироваться как внутренним (Internal), так и внешним (External) сигналом синхронизации. На вывод Ready подается сигнал готовности.

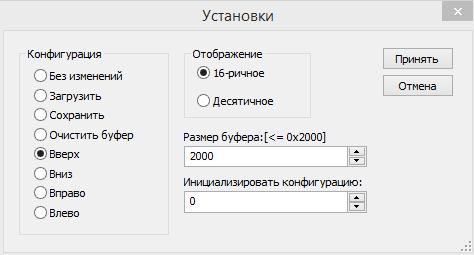
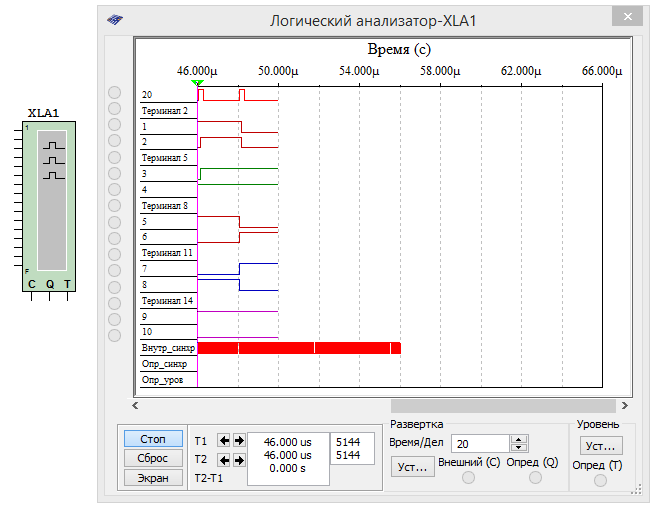


Рис. 7.5. Окно свойств буфера

**Логический анализатор (Logic Analyzer)** – устройство, предназначенное для диагностики цифровых схем. ЛА позволяет отслеживать и записывать состояния логических элементов цифровых электронных устройств, анализировать и визуализировать их. Внешний вид и лицевая панель логического анализатора показаны на рис. 7.6.

ЛА имеет 16 каналов для съема сигналов, а также несколько входов запуска. Кроме этого, прибор снабжен двумя курсорами, позволяющими проводить измерения во временной области.

Если вход 1 считать младшим разрядом, а вход 16 – старшим, то состояние всех входов может быть представлено 16-разрядным двоичным кодом. Код, соответствующий позиции курсора, отображается в поле «Входной код» (рис. 7.7).



Избирательный вход запуска

Маскированный вход запуска

Панель управления

Вход

запуска

Входы

Рис. 7.6. Логический анализатор

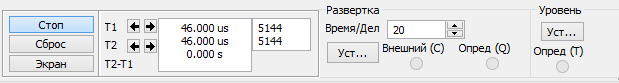
При нажатии кнопки «Установки» в группе «Развертка» (тактовый генератор) открывается диалоговое окно настройки параметров тактирования входных сигналов (рис. 7.8).

Тактирование сигналов осуществляется с использованием внешнего (External) или внутреннего (Internal) источника.

Показания курсоров Т1 и Т2

Остановить анализ

Очистить

****

Параметры запуска

Входной код

Инвертировать

цвет экрана

Рис. 7.7. Панель управления логического анализатора

В поле Clock Qualifier (Определитель) устанавливается активный уровень сигнала синхронизации. В поле Clock Rate (Тактовая частота) устанавливается частота выборки анализатора.

В группе Sampling Setting (Дискретизация) задаются параметры выборки сигналов:

* Pre-trigger Samples (До порога) – сбор данных производится до поступления импульса запуска;
* Post-trigger Samples (После порога) – сбор данных начинается после поступления импульса запуска и продолжается до тех пор, пока не будет набрано заданное количество отсчетов;
* Threshold Volt (Порог) – пороговая величина.

Дополнительные условия запуска анализатора осуществляются с помощью диалогового окна Trigger Settings.

В данном окне настраивается маска, по которой осуществляется фильтрация логических уровней и синхронизация входных каналов.

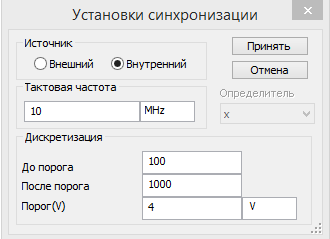


Рис. 7.8. Настройка параметров синхронизации

**7.1.3. Компоненты Multisim**

Компоненты – это основа любой схемы, т. е. элементная база, из которой состоит схема. В Multisim работа осуществляется с двумя категориями компонентов: виртуальными (virtual) (рис. 7.9, *а*) и реальными (real) (рис. 7.9, *б*).

Реальные компоненты являются полными аналогами компонентов, выпускаемых или выпущенных радиоэлектронной промышленностью. Виртуальные компоненты являются математическими моделями семейств (Family) компонентов (резисторы, конденсаторы и т. д.) с любыми произвольными параметрами, присущими данной категории.

**­**­­­­­­­ D:\Учеба\yflj\Безымянный-1.wmf

*б*

*а*

Рис. 7.9. Компоненты Multisim:

*а* – виртуальные; *б* – реальные

Добавление компонентов в схему осуществляется из меню **Place** либо панели инструментов **Components**.

Рассмотрим основные группы компонентов базы данных Multisim.

Группа **Sources** (Источники) содержит модели источников питания (однофазный источник питания постоянного (DC\_POWER) и переменного напряжения (AC\_POWER), трехфазные источники питания, источники питания постоянного тока (VCC1, VDD2, VEE3, VSS4), а также заземление (GROUND)), источников напряжения (источник прямоугольного сигнала, кусочно-линейного сигнала (PWL Voltage) и др.), источников тока и т. д.

Группа **Basic** (Базовые компоненты) включает модели резисторов, конденсаторов, индуктивностей, трансформаторов, виртуальных механических ключей и т. д.

Группа **Diodes** (Диоды) содержит модели таких компонентов, как диод, стабилитрон (zener), светодиод, диодный мост (FWB), диод Шоттки, тиристор и др.

Группа **Transistors** (Транзисторы) включает модели биполярных транзисторов (BJT), полевых транзисторов (JFET), МОП-транзисторов и др.

Группа **Analog** (Аналоговые компоненты) содержит модели операционных усилителей (OPAMP), компараторов (COMPARATOR) и др.

Группа **TTL** (цифровые микросхемы по технологии ТТЛ) содержит модели микросхем серий 74Sxx, 74LSxx, 74ALSxx и др.

Группа **CMOS** (цифровые микросхемы по технологии КМОП) содержит модели микросхем серии 74HCxx, NC7Sx (Tiny Logic) и др.

Группа **Misc** **Digital** (Цифровые устройства) включает виртуальные модели цифровых устройств (TIL) (логические элементы, триггеры, регистры, счетчики, мультиплексоры, декодеры, элементы арифметико-логических устройств и др.), микросхемы цифровой обработки сигналов (DSP), программируемые логические интегральные схемы, микросхемы памяти, микроконтроллеры и др.

Группа **Indicators** (Индикаторные устройства) включает следующие модели компонентов: индикаторы напряжения и тока, логические пробники, семисегментные индикаторы, звуковые индикаторы и др.

**7.1.4. Создание схем в Multisim**

Процесс создания схемы начинается с выбора компонентов схемы. Выбранный компонент автоматически прикрепляется к курсору мыши, после чего размещается в любом месте рабочего окна (рис. 7.10).

D:\Учеба\yflj\Безымянный-2.wmf

Рис. 7.10. Размещение компонентов схемы

Далее устанавливается ориентация компонентов (если это необходимо) с помощью команд всплывающего меню (нажатие правой кнопки мыши на выбранном компоненте) или комбинации клавиш *Alt +X*, *Alt +Y* и устанавливаются параметры элементов питания.

После размещения компонентов схемы производится соединение их выводов проводниками. При этом необходимо учитывать, что к выводу подключается один проводник. Для выполнения подключения курсор мыши подводится к выводу компонента и после изменения вида курсора  устанавливается соединение между выводами элементов или соединительной точкой (рис. 7.11). Для добавления соединительной точки необходимо нажать комбинацию клавиш *Ctrl +J* или выбрать соответствующий пункт во всплывающем меню нажатием правой кнопки мыши.

D:\Учеба\yflj\Безымянный-3.wmf

Соединительная точка

Рис. 7.11. Соединение элементов схемы

При изменении существующего соединения курсор отображается в виде перекрестия с двумя отрезками вдоль соединительной линии (рис. 7.12).

D:\Учеба\yflj\Безымянный-4.wmf

Рис. 7.12. Изменение соединения элементов схемы

**Практические задания**

**Задание 1**. Запустить среду разработки Multisim исобрать на рабочем поле среды Multisim схему для испытания *основных и базовых логических элементов* (рис. 7.13) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему на страницу отчета.

Схема (рис. 7.13) собрана на двоичных основных [**ОR** (ИЛИ), **AND** (И) и **NOT** (НЕ)] и универсальных (базовых) [**NAND** (И-НЕ) и **NOR** (ИЛИ-НЕ)] логических элементах, расположенных в библиотеке **Misc Digital/TIL (Цифровые микросхемы/TIL)** с уро­в­нем высокого постоянного напряжения 5 В. В схе­му включены ключи **1**, **2**, ..., **9**, пробники (лампочки) **Х1**, **Х2**, …, **Х5** с пороговыми напряжениями 5 В, генератор постоянных токов **Е1** с напряжением *Е* = 5 В и логический анализатор **XLA1.**

D:\Учеба\yflj\Безымянный-5.wmf

Рис. 7.13. Схема, собранная на двоичных основных и универсальных логических элементах

Для удобства измерения сигналов выходы логических элементов подключены к входам 2, 4, 6, 8 и 10 анализатора **XLA1** (рис. 7.14). При моделировании происходит медленная развертка временных диаграмм в окне анализатора. По достижении интервала времени, равного 70–80% ширины окна, следует посредством кнопки **Run/Stop** выключать процесс моделирования.

Оперируя ключами **1**, **2**, …, **9**, сформировать все возможные ком­бинации аргументов *х*1 и *х*2 (00, 10, 01 и 11) на входе дизъюнктора (**OR**), конъюнктора (**AND**), штриха Шеффера (**NAND**) и стрелки Пирса (**NOR**) и записать значения выходных логических функций *yk* (0 или 1) в табл. 7.1.

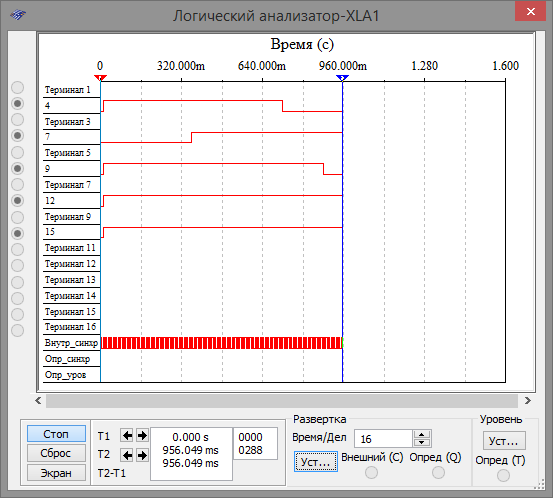
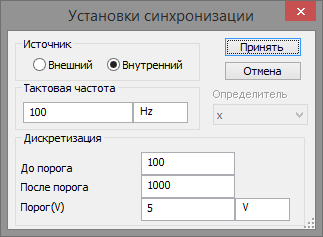


Рис. 7.14. Использование логического анализатора

Заметим, что если ключ замкнут, то на этот вход элемента будет подана логическая единица (положительный потенциал 5 В), а при разомкнутом ключе – логический ноль. Поскольку инвертор (**NOT**) имеет один вход, то для формирования двух значений входного сигнала (логической единицы или логического нуля) достаточно одного ключа **5**.

Значения функций исследуемых элементов можно контролироватьс помощью пробников **Х1**, **Х2**, …, **Х5**: если выходной сигнал элемента равен логической единице, то включенный на выходе этого элемента пробник светится. Так, при положении ключей схемы (рис. 7.2) функции элементов **OR**, **AND** и **NOR** равны логической единице.

Таблица 7.1

**Комбинации аргументов**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Дизъюнктор [ИЛИ (**OR**)] | | | Конъюнктор  [И (**AND**)] | | | Инвертор  [НЕ **NOT**)] | | Штрих Шеффера  [И-НЕ **NAND**)] | | | Стрелка Пирса  [ИЛИ-НЕ (**NOR**)] | | |
| *х*1 | *х*2 | *y* | *х*1 | *х*2 | *y* | *х* | *y* | *х*1 | *х*2 | *y* | *х*1 | *х*2 | *y* |
| 0 | 0 |  | 0 | 0 |  | 0 |  | 0 | 0 |  | 0 | 0 |  |
| 0 | 1 |  | 0 | 1 |  | 0 | 1 |  | 0 | 1 |  |
| 1 | 0 |  | 1 | 0 |  | 1 |  | 1 | 0 |  | 1 | 0 |  |
| 1 | 1 |  | 1 | 1 |  | 1 | 1 |  | 1 | 1 |  |

**Задание 2**. «Перетащить» из библиотеки **Misc Digi­tal\TIL** на рабочее поле среды Multisim необходимые логические элементы и собрать схему для реализации заданной в табл. 7.2 логической функции *у* с тремя аргументами *а*, *b* и *c*.Скопироватьсобранную логическую схему на страницу отчета.

Таблица 7.2

**Список заданий**

|  |  |
| --- | --- |
| Вариант | Логическая функция |
| 1, 5, 8 |  |
| 2, 6, 9 |  |
| 3, 7, 10 |  |
| 4, 8, 11 |  |

В качестве примера соберем схему для реализации логической функции:****

Анализ функции показывает, что для построения логической схемы нам потребуются три инвертора, три дизъюнктора, причем один дизъюнктор с двумя, а два − с тремя входами, и два конъюнктора, причем один с двумя, а другой с тремя входами.

«Перетащим» на рабочее поле среды Multisim необходимые модели логических элементов из библиотеки **Misc Digital\TIL**, располагая их начиная с входа, а именно:

− три инвертора NOT (**NOT1**, **NOT2** и **NOT3**) для получения инверсий  аргументов *a*, *b* и *с*;

− конъюнктор **AND1** с двумя входами для реализации функции *ab*;

− три дизъюнктора: **OR2** для реализации функции *y*1*= a* + *b* + *c*, **OR3** для реализации функции *y*2 =  и **OR1,** реализующий функцию*y*3 =  разместив их друг под другом (рис. 7.15).

­D:\Учеба\yflj\Безымянный-6.wmf

Рис. 7.15. Элементы для построения схемы

Для выполнения функции логического умножения *y*=*y*1*y*2*y*3 добавим в схему конъюнктор **AND2** c тремя входами, к выходу которого подключим логический пробник **Х1** (уро­вень высокого напряжения 5 В) для сигнализации появления ло­ги­ческой единицы на выходе схемы. «Перетащим» из соответствующих библиотек на рабочее поле источник прямоугольных сигналов **Е1** и ключ **1**, расположив их на входе схемы.

Соединив «проводниками» входы и выходы элементов в соответствии с логическими выражениями составляющих заданной функции и записав в отчете ожидаемые результаты выполнения операций на выходах элементов (рис. 7.16), приступим к моделированию.

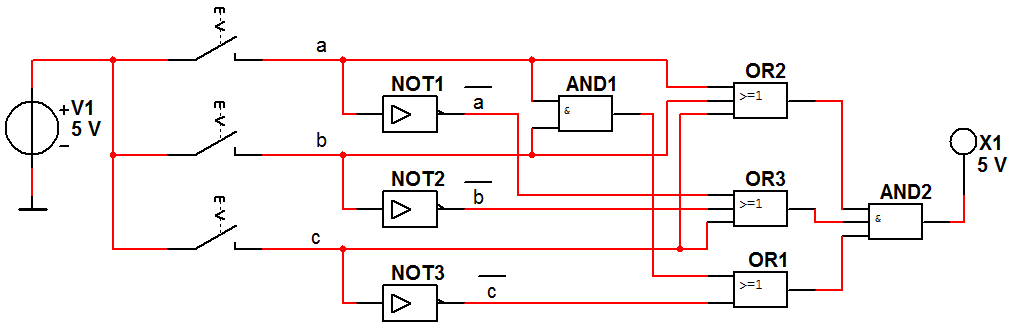


Рис. 7.16. Логическая схема

С этой целью вначале щелкнем мышью на кнопке **Run/Stop**, затем нажмем управляющую ключом клавишу с цифрой **1** клавиатуры. Если соединения элементов выполнены правильно, то пробник **Х1** засветится. При выключении ключа **1** пробник гаснет и т. д. По окончании моделирования щелкнем мышью на кнопке **Run/Stop**.

**Примечание**. Основным измерительным прибором для проверки цифровых электронных схем является логический пробник. После двойного щел­чка мышью на его изображении в открывшемся окне нужно задать уро­вень высокого напряжения, например 5 В (см. рис 7.4), при котором он светится. Если пробник не светится, то это обычно означает, что уровень проверяемого напряжения находится в промежутке между высоким и низким. Поиск неисправностей нужно начинать с проверки подачи сигналов высокого уровня генератором сигналов на входы элементов, затем про­верить правильность выполнения ими логических функций в схеме и проконтролировать появление сигналов на выходах.

**Содержание отчета**

1. Название и цель работы.

2. Перечень приборов, использованных в экспериментах, с их крат­кими характеристиками.

3. Изображения электрической схемы для испытания логических элементов и собранной схемы для реализации заданной логической функции.

4. Таблицы истинности, отображающие работу исследуемых логических элементов.

5. Выводы по работе.

**7.2. Запоминающие элементы. Триггеры**

**Практические задания**

**Задание 1**. Запустить среду разработки Multisim и собрать на рабочем поле среды Multisim схему для испытания *асинхронного RS-триггера* (рис. 7.17) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему на страницу отчета.

Схема (рис. 7.17) собрана на четырех логических эле­ментах И-НЕ (**NAND**). На входы *S* и *R* элементов **NAND1** и **NAND2** через ключи **1** и **2** подаются логические сигналы 1 или 0 от источника прямоугольных импульсов **Е1** с амплитудой 5 В. К выходам *Q* и  эле­ментов **NAND3** и **NAND4**, т. е. к выходам триггера, как и к его входам *S* и *R*, подключены пробники **Х1**, **Х2**, **Х3** и **Х4** с пороговым напряжением 5 В.

D:\Учеба\yflj\Безымянный-7.wmf

Рис. 7.17. Схема для испытания асинхронного *RS*-триггера

Воспользовавшись порядком засвечивания разноцветных пробников и задавая коды (00, 01, 10) состояния ключей 1 и 2 (входных сигналов), составить таблицу истинности *RS*-триггера. Например, сформировав с помощью ключей сигналы *S* = 1 и *R* = 0 и подав их на вход триггера, получить на его выходе сигналы *Q* = 1 и  = 0 (см. рис. 7.17). Убедиться, что при запрещенном коде 11 входных сигналов на выходе *RS*-триггера могут засветиться оба пробника, или оба не светятся.

**Задание 2**. Подключить к входам триггера логический генератор (генератор слова) **XWG1** (рис. 7.8), запрограммировав его первые три ячейки кодами 00, 10 и 01 и соединив входы и выходы триггера с входами логического анализатора **XLA2**.

В диалоговом окне генератора слова **XWG1 задать** частоту *f*г = 10 кГц и два цикла моделирования сигналов (в режиме **Burst)**, а в окне анализатора **XLA2** − частоту *fа*= 0,1 МГц таймера, уровень высокого напряжении *Um* = 5 В, число импульсов **Clocks/div** = 8 таймера, приходящихся на одно деление.

**D:\Учеба\yflj\Рисунки практикум\8.wmf**

Рис. 7.18. Схема для испытания асинхронного *RS*-триггера

Получить на экране анализатора **XLA2** временную диаграмму состояний *RS*-триггера (см. рис. 7.18, внизу). Скопировать схему испытания и временную диаграмму состояния *RS*-триггера на страницу отчета.

**Задание 3**. Собрать на рабочем поле среды Multisim схему для испытания *триггеров* **JK**,**Т** и**D**(рис. 7.19) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему на страницу отчета.

В схему (рис. 7.19) включены:генератор **XWG1** (частота *fг* = 500 кГц);логический анализатор **XLA1**;триггерыв интегральном исполнении: универсальный **JK**, счетный**Т** и задержки **D**.

На *-* и -входы триггеров подается постоянное напряжение 5 В (имитирующее сигнал 1) источника **VCC**, а на 1С-входы триггеров и на вход 20 анализатора **XLA1** поступают тактовые импульсы с амплитудой 5 В и частотой 500 кГц, сформированные генератором **Е1**.

С выходов 1 и 2 генератора **XWG1** сигналы подаются на управляющие входы **1J** и **1K** *JK*-триггера, с выхода 3 − на вход **1D** *Т*-триггера, а с выхода 4 − на вход **1D** *D*-триггера.

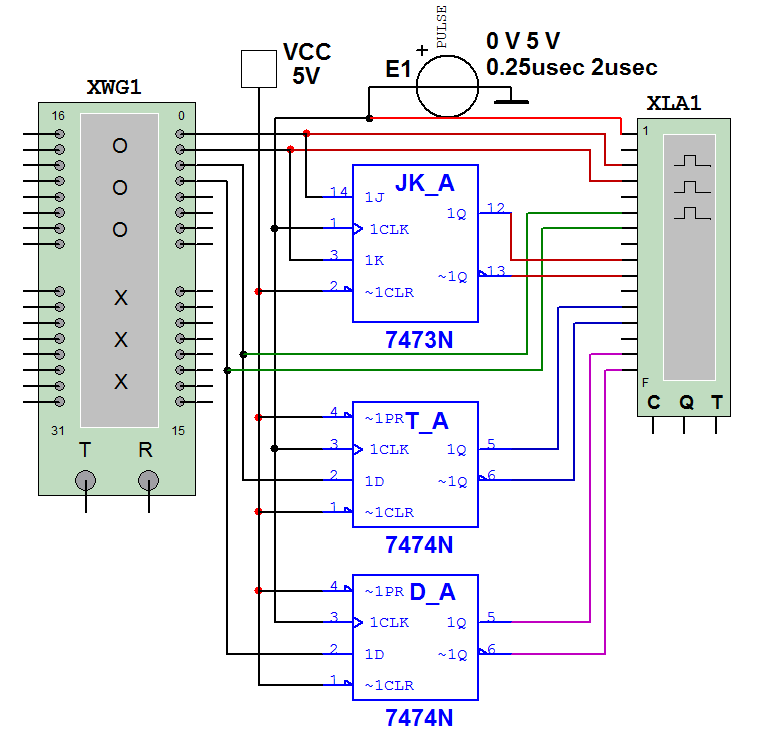


Рис. 7.19. Схема для испытания триггеров *JK*, *T* и *D*

Для формирования выходных сигналовгенератор **XWG1** нужно запрограммировать, т. е. ввести в ячейки памяти кодовые комбинации из единиц и нулей согласно варианту (таблица 7.3).

Таблица 7.3

**Список заданий**

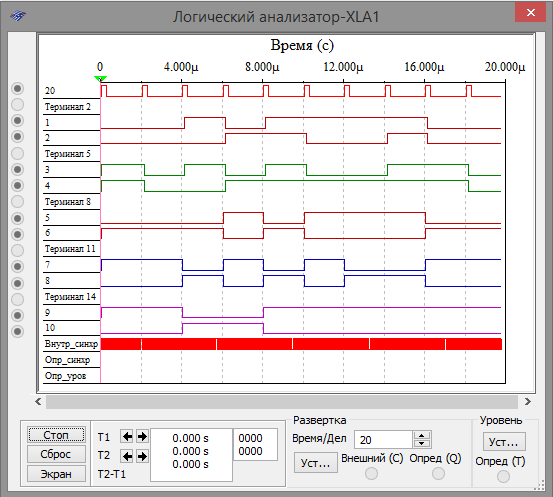
|  |  |
| --- | --- |
| Вариант | Содержимое ячеек памяти генератора слова **XWG1** |
| 1, 6, 11 | 0000, 1010, 1111, 1001, 1001, 1101, 1100, 0000 |
| 2, 7, 12 | 0000, 1100, 1010, 1011, 1001, 1111, 1110, 0000 |
| 3, 8, 13 | 0000, 1010, 1011, 1001, 1001, 1111, 1101, 0000 |
| 4, 9, 14, | 0000, 1111, 1101, 1001, 1011, 1011, 1100, 0000 |
| 5, 10, 15, | 0000, 1011, 1101, 1001, 1100, 1111, 1010, 0000 |

В качестве примера введем в первые восемь ячеек памяти генератора четырехразрядные кодовые комбинации (см. рис. 7.20):

0000, 0101, 1010, 1111, 1001, 1001, 1111, 1100.

При моделировании генератор последовательно и циклично выводит содержимое каждой ячейки памяти (от начальной до конечной) на выходы 1, 2, 3 и 4, формируя на них следующие коды сигналов: 01011110, 00110010, 01010011 и 00111111 (см. сигналы на каналах 1, 2, 3 и 4 логического анализатора **XLA1** (рис. 7.20)). Перед моделированием выделить в окне генератора **XWG1** ячейку с адресом 0 начала счета и вывода сигналов.

Провести моделирование работы триггеров в режиме «Пошагово» генератора **XWG1**,скопировать в отчет временные диаграммы, составить и заполнить таблицы истинности работы триггеров **JK**, **T** и **D** при заданном в таблице варианте входных кодовых комбинаций. В частности, описать состояния *JK*-триггера с приходом тактового сигнала *C* = 1, когда сигналы *J* = 1 и *K* = 1, а *Q* = 0 или *Q* = 1.



*D*-триггер

*T*-триггер

*JK*-триггер

Генератор XWG1

Генератор Е1

Рис. 7.20. Восемь ячеек памяти генератора с четырехразрядными кодовыми комбинациями

**Содержание отчета**

1. Название и цель работы.

2. Перечень приборов, использованных в экспериментах, с их крат­кими характеристиками.

3. Изображения электрических схем для испытания триггеров *RS*, *JK*, *T* и *D* с помощью логических пробников и логического анализатора **XLA1**. Копии временных диаграмм и таблицы истинности, отображающие работу исследуемых триггеров.

4. Выводы по работе.

**7.3. Исследование интегральных преобразователей кодов (дешифратора, шифратора) и коммутационных узлов (демультиплексора и мультиплексора)**

**Практические задания**

**Задание 1**. Запустить среду разработки Multisim, собрать на рабочем поле среды Multisim схему для испытания *дешифратора* **DC** (рис. 7.21) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопировать схему на страницу отчета.

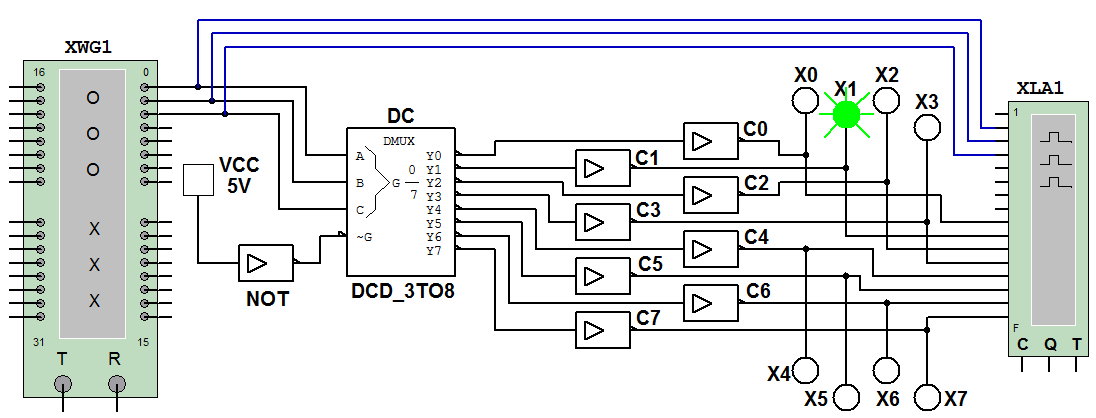


Рис. 7.21. Схема для испытаний дешифратора

Схема содержит интегральный *дешифратор* **DC** (*decoder*) 3×8, имеющий 3 информационных входа **A**, **B** и **C** (для кода 4−2−1), 8 выходов (**Y0**, …, **Y7**) и преобразующий позиционный 3-разрядный двоичный код в *унитарный* «1 из 8»: в выходной 8-разрядной кодовой комбинации только одна позиция занята единицей, а все остальные – нулевые (см. рис. 7.22, справа). В зависимости от входного двоичного кода, например 001, на выходе **DС** появляется сигнал 1 только на одной (второй, см. рис. 7.21) из 8 выходных линий, к которым подключены пробники **Х0**,…, **Х7**.

Данный тип шифратора относится к шифраторам с разным уровнем входных и выходных сигналов: активные входные уровни соответствуют уровню логической 1, а активные выходные сигналы – уровню логического 0. Для получения активных выходных уровней, равных 1, к выходам дешифратора подключено восемь инверторов **С0**, …, **С7**;

Логический генератор слова **XWG1** (*f*г = 500 кГц) с записанными логическими словами в ячейки памяти, которые эквивалентны десятичным числам от 0 до 7, представлен на рис. 7.22.

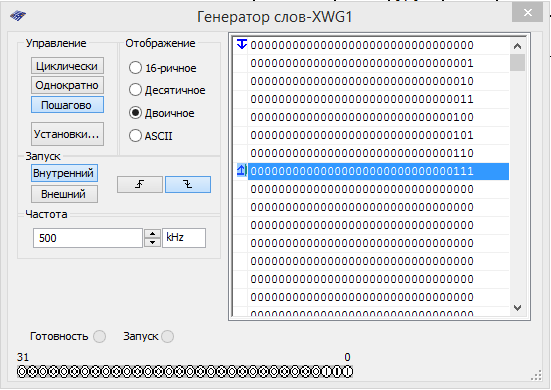


Рис. 7.22. Логический генератор

На экран логического анализатора **XLA1** (рис. 7.22) выводятся временные диаграммы как трех входных (**А**, **В,** **С**), так и восьми **(Y0**, **Y1**, …, **Y7**) выходных сигналов при пошаговом режиме генератора **XWG1**.

С выхода источник **VCC** напряжение 5 В подано на инвертор **NOT**. Логический 0 с инвертора подается на управляющий вход дешифратора **DC**: при дешифратор находится в активном состоянии.

Запустить программу моделирования дешифратора. Щелкая мышью на кнопке «Пошагово» генератора **XWG1**,последовательноподаватьна вход дешифратора логические слова.

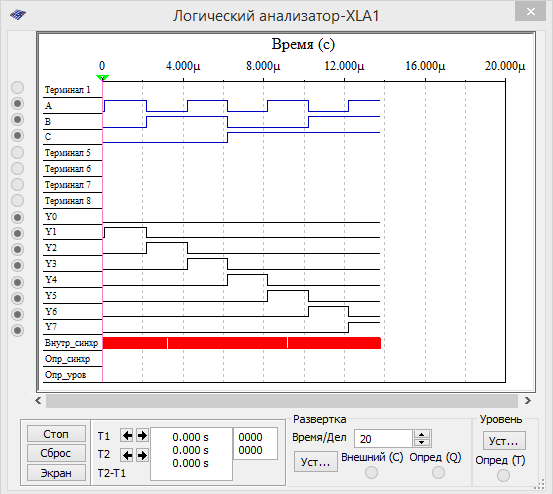


Рис. 7.23. Логический анализатор

Убедиться, что при подаче на вход дешифратора каждой новой двоичной кодовой комбинации засвечивается только один пробник, который «распознает» свой входной код.

Скопировать временные диаграммы входных и выходных сигналов дешифратора на страницу отчета. По результатам моделирования составить и заполнить табл. 7.4 переключений (функций  на выходах дешифратора **DC** 3×8.

Таблица 7.4

**Переключение на выходах дешифратора DC 3×8**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **А** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **В** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **С** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| **Y0** |  |  |  |  |  |  |  |  |  |
| **Y1** |  |  |  |  |  |  |  |  |  |
| **Y2** |  |  |  |  |  |  |  |  |  |
| **Y3** |  |  |  |  |  |  |  |  |  |
| **Y4** |  |  |  |  |  |  |  |  |  |
| **Y5** |  |  |  |  |  |  |  |  |  |
| **Y6** |  |  |  |  |  |  |  |  |  |
| **Y7** |  |  |  |  |  |  |  |  |  |

**Задание 2.** Собрать на рабочем поле среды Multisim схему для испытания *шифратора* **СD** (рис. 7.24) и установить в диалоговых окнах ком­понентов их параметры или режимы работы.

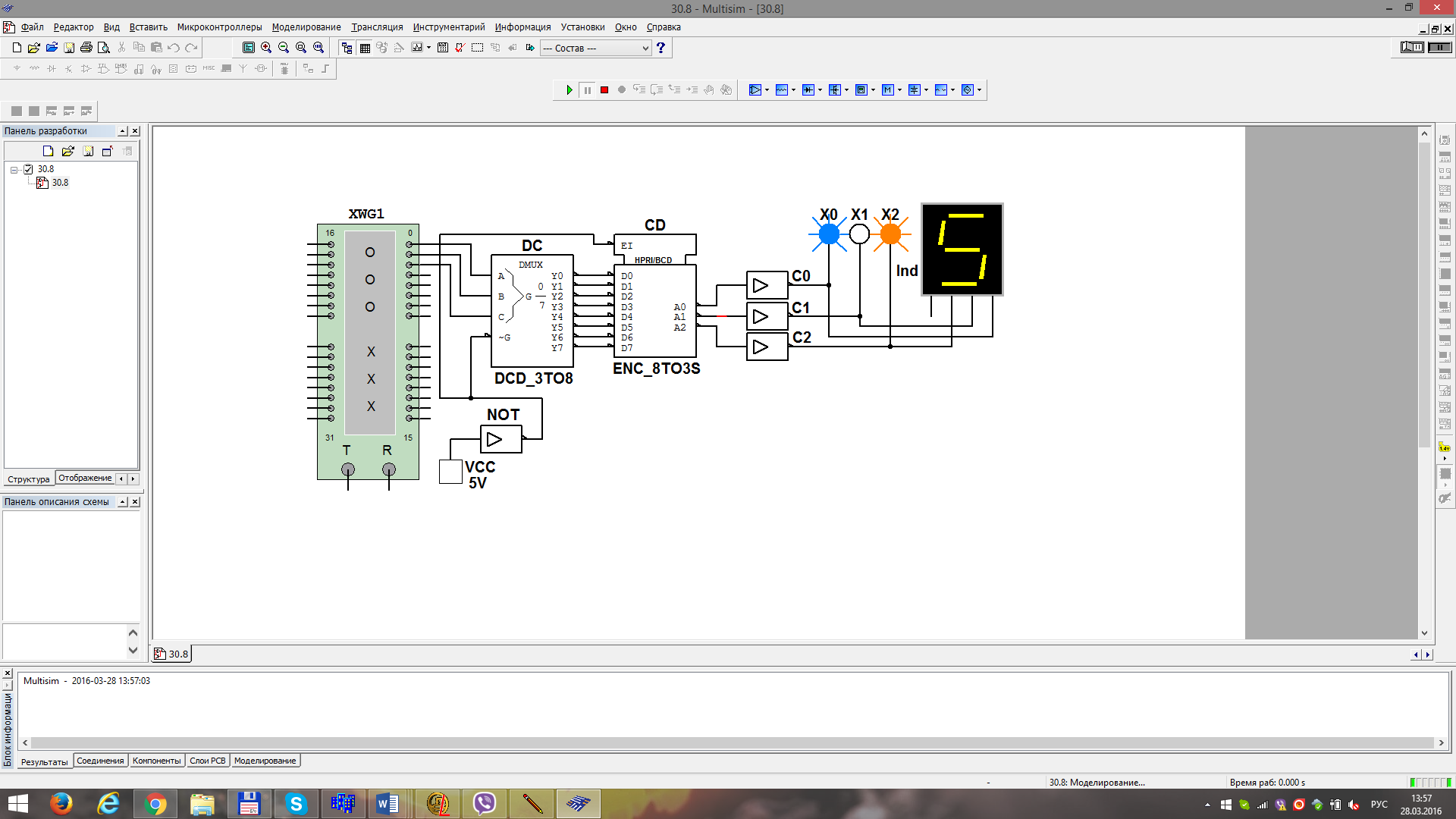
****

Рис. 7.24. Схема для испытаний шифратора

Скопироватьсхему на страницу отчета.

Запустить программу моделирования шифратора. Щелкая мышью на кнопке «Пошагово» генератора **XWG1**,последовательноподаватьна вход дешифратора логические слова. Убедиться, что при подаче с выхода **DC** на вход шифратора **СD** 8-разрядной последовательности, в которой только одна позиция занята единицей, а остальные − нулями, на выходе шифратора формируются 3-разрядные двоичные коды **A0A1A2**, где **А0 = А**, **А1 = В** и **А2 = С**, соответствующие двоичным кодовым комбинациям на входе дешифратора **DC**.

По результатам моделирования (позасвечиванию логических пробников **Х0**, **Х1**, **Х2** и показаниям индикатора **Ind**) составить и заполнить таблицу переключений на выходе шифратора **CD** 8×3 (табл. 7.5).

Таблица 7.5

**Переключение на выходах шифратора DC 3×8**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **B** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **C** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| **Y0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Y1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Y2** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Y3** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **Y4** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Y5** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **Y6** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **Y7** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **А0** |  |  |  |  |  |  |  |  |  |
| **А1** |  |  |  |  |  |  |  |  |  |
| **А2** |  |  |  |  |  |  |  |  |  |

Преобразоватьсхему дешифратора **DC** 3×8 и шифратора **CD** 8×3 (см. рис. 7.24) в схему **DC** 2×4 и шифратора **CD** 4×2, отсоединив провод **С**, подходящий к дешифратору, и провод **A2** с выхода шифратора, и составить таблицы переключений дешифратора 2×4 и шифратора 4×2.

**Задание 3**. С**обрать** на рабочем поле среды Multisim схему для испытания*демультиплексора* **DMS** (рис. 7.25)и установить в диалоговых окнах компонентов их параметры или режимы работы.

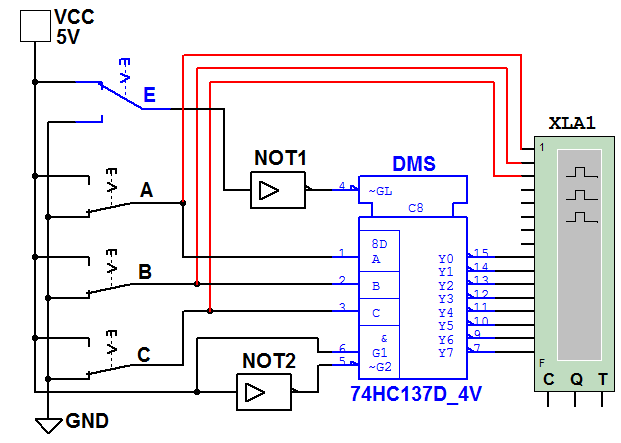


Рис. 7.25. Схема для испытаний демультиплексора

*Демультиплексор* **DMS (логические микросхемы CMOS\74HC\_4V – 74HC137D\_4V)** 1×8 (из 1 в 8) имеет один информационный вход (с активными высоким **G1** и низким **G2** уровнями), три адресных **А**, **В**, **С** входа, разрешающий **GL** вход с активным низким уровнем и восемь **Y0**, **Y1**, …, **Y7** инверсных выходов, соединенных с входами логического анализатора **XLA1**. На вход анализатора также подаются сигналы с адресных входов **А**, **В**, **С**. С помощью ключей **А**, **В** и **С** можно сформировать восемь трехразрядных двоичных адресных слов. При последовательной подаче формируемых ключами адресных слов от 111 до 000 на экран анализатора **XLA1** при моделировании выводятся 8-разрядные кодовые последовательности с одним активным (низким) уровнем.

Для обеспечения медленного перемещения лучей на экране анализатора **XLA1** установитьтактовую частоту его таймера *fa* = 500 Гц и число импульсов, приходящихся на одно деление, **Время/Дел** = 80.

Задать код ключей 111 и щелкнуть мышью на кнопке **Run/Stop**.Кривые адресных и выходных логических сигналов медленно разворачиваются во времени на экране анализатора.

Повторять перечисленные выше операции для спадающих счетных комбинаций адресных сигналов (с 110 до 000) до тех пор, пока не будет записан процесс моделирования при адресном слове 000.

Убедиться, что для каждой комбинации адресных сигналов демультиплексор формирует логический 0 на одном из восьми выходов, номер которого соответствует определенному кодовому слову на входе, т. е. демультиплексор подобен коммутатору, посредством которого поток цифровой информации разделяется на 8 выходных потоков.

Скопироватьсхему (рис. 7.25) и временные диаграммы входных и выходных сигналов на страницу отчета.

Если адресные входы **А**, **В** и **С** принять в качестве информационных входов, а вход **G1** (**G2**) в качестве входа разрешения работы, то мультиплексор превратится в дешифратор.

**Задание 4.** **Собрать** на рабочем поле среды Multisim схему(рис. 7.26) для испытания*мультиплексора* **MS 8×1 (из 8 в 1) (Логические микросхемы TTL\74STD\74151N)** и установить в диалоговых окнах компонентов их параметры или режимы работы.

Мультиплексор МS с разрешающим входом G осуществляет передачу сигнала с каждого информационного входа D0, D1, …, D7, заданного 3-разрядным кодом АВС – адресом выбираемого входа, на единственный выход Y. Разрядность (3) управляющего сигнала определяет количество входов (23 = 8), с которых мультиплексор может принимать информацию. Если предположить, что к входам D0, D1, …, D7 мультиплексора MS присоединено 8 источников цифровых сигналов – генераторов последовательных двоичных слов, то байты от любого из них можно передавать на выход Y.

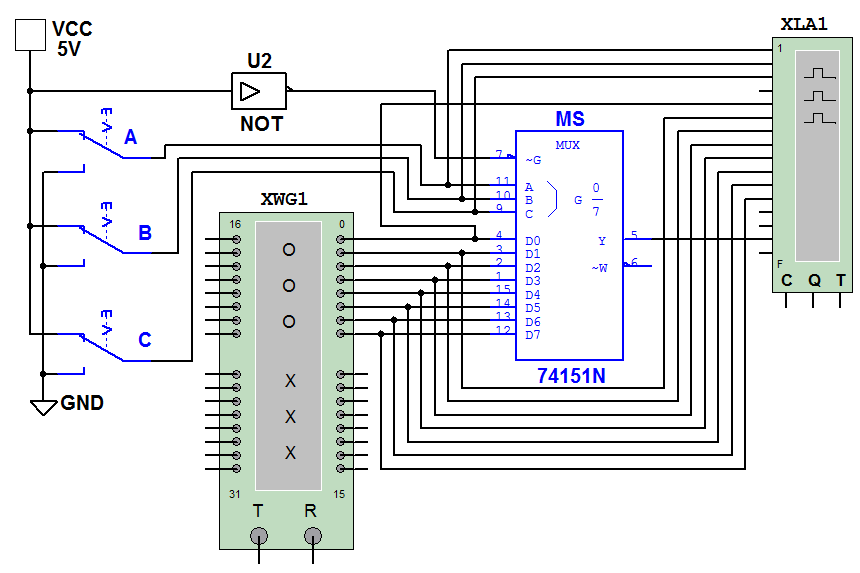


Рис. 7.26. Схема для испытаний мультиплексора

Скопировать схему (рис. 7.26) на страницу отчета.

Для иллюстрации работы мультиплексора MS запишем в ячейки памяти генератора XWG1 произвольные 8-разрядные кодовые слова (рис. 7.27), а с помощью ключей А, В, С сформируем управляющий сигнал **111**.

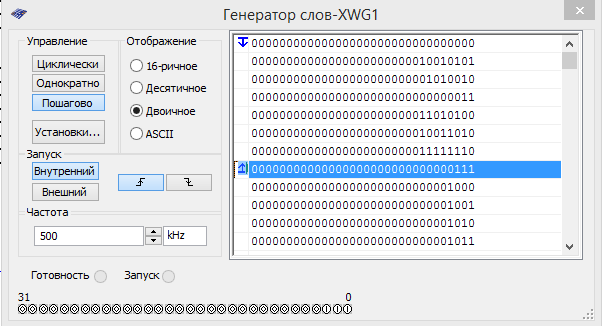


Рис. 7.27. Генератор XWG1 с произвольными 8-разрядными кодовыми словами

Последовательно щелкая мышью на кнопке «Пошагово» генератора XWG1 и при G = 1 поступающие на вход D7 мультиплексора байты (сигнал 01001110) с 8-го разряда (на рис. 7.27, слева 8-й разряд показан стрелкой) логических слов генератора XWG1 передаются на выход Y и на вход анализатора (см. рис. 7.28).

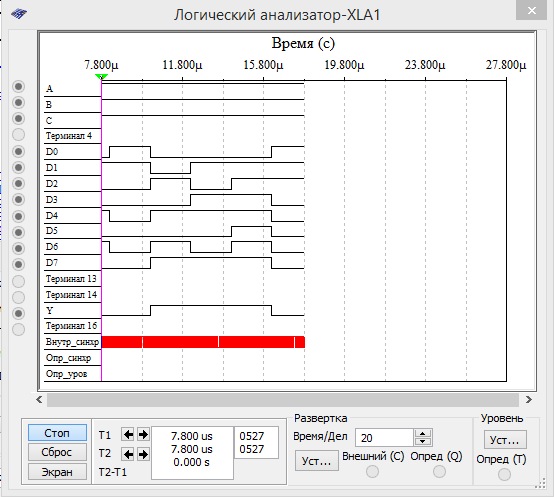


Рис. 7.28. Анализатор XLA1 с произвольными 8-разрядными кодовыми словами

Если ключ **А** установить в нижнее положение (сформировав тем самым адресный код 011), то с входа **D3** на выход **Y** мультиплексора будут поступать байты 4-го разряда логических слов, записанных в ячейки памяти генератора **XWG1**, и т. д.

Записатьв первые восемь ячеек памяти генератора **XWG1** произвольные 8-разрядные кодовые слова, **задать** частоту *f*г = 500 кГц и режим «Пошагово» его работы (см. рис. 7.27, слева).

Задатьчастоту *f*a = 20 МГц таймера логического анализатора **XLA1** и количество импульсов таймера **Clock/div** = 20, приходящихся на одно деление.

Установить с помощью ключей **А**, **В** и **С** адресный код, например 1002 (410) и запустить программу моделирования мультиплексора. Получить ископировать временные диаграммы входных сигналов **D0**, **D1**, …, **D7** и выходного сигнала **Y** мультиплексора на страницу отчета.

**Задание 4.** С**обрать** на рабочем поле среды Multisim схему для испытания*демультиплексора* **DMS 1×16 (из 1 в 16**) (рис. 7.29)и установитьв диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему (рис. 7.29) в отчет.

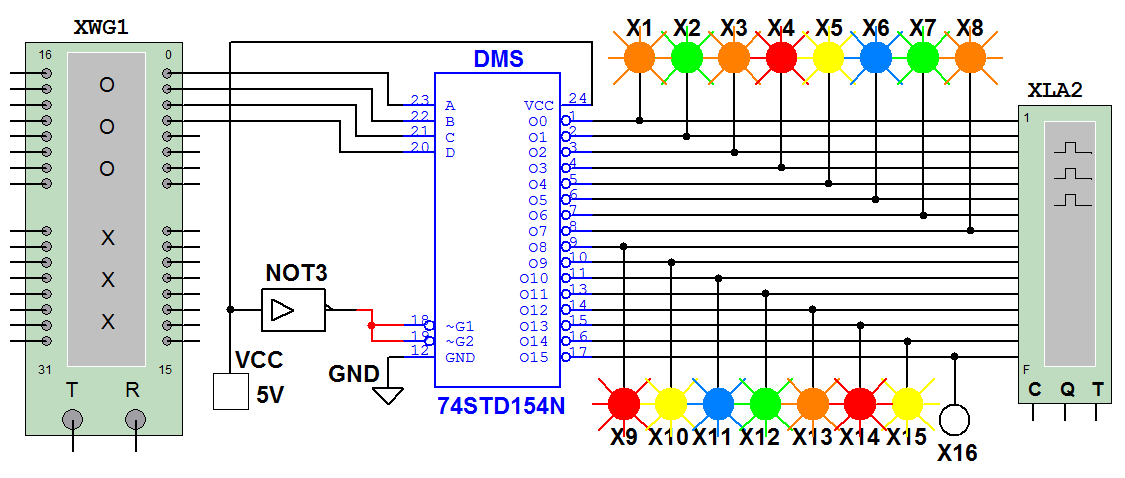


Рис. 7.29. Схема для испытаний демультиплексора

С целью автоматизации процесса моделирования к входу демультиплексора **DMS** подключен логический генератор **XWG1** сзаписанными в его ячейки памяти адресными кодами от 0000 до 1111, а для визуализации сигналов на выходах включены 16 логических пробников **Х1**, **Х2**, …, **Х16** и логический анализатор **XLA2**.

Запустить программу моделирования демультиплексора **DMS** 1×16. Последовательноподавать(щелкая мышью на кнопке «Пошагово» генератора **XWG1**) (рис. 7.30) на вход демультиплексора логические слова, начиная с комбинации 0000 адресного сигнала и заканчивая комбинацией 1111, инаблюдатьза изменениями выходных сигналов по показаниям индикаторов и в окне анализатора **XLA2** (рис. 7.30).

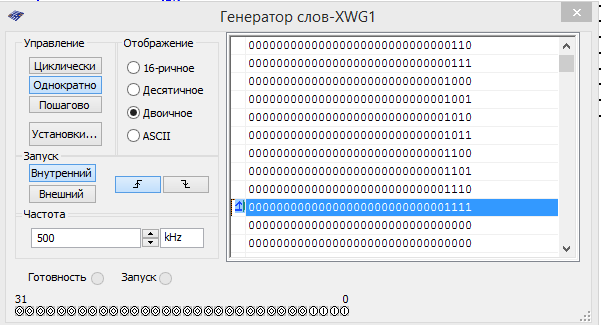


Рис. 7.30. Логический генератор с записанными в него ячейками памяти

В исследуемой модели демультиплексора соответствующий активный выход имеет низкий логический уровень (рис. 7.29), поэтому про­бник на этом выходе не светится. Так, при подаче последней кодовой комбинации 1111 на вход демультиплексора не светится пробник **Х16**, так как активным является выход **15** (см. рис. 7.31).

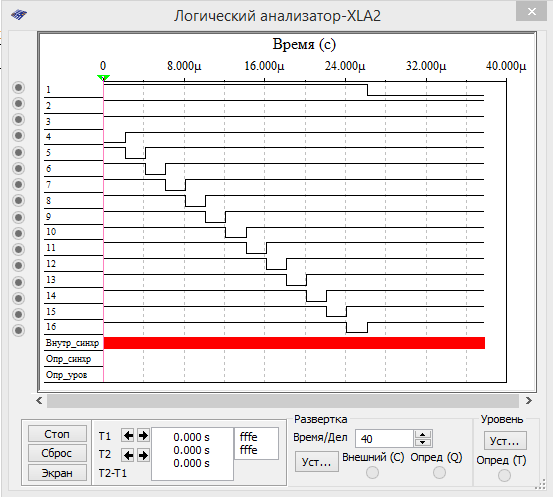


Рис. 7.31. Показания логического анализатора при исследовании демультиплексора

Скопировать на страницу отчета временные диаграммы выходных сигналов демультиплексора **DMS** 1×16.

**Содержание отчета**

1. Название и цель работы.

2. Перечень приборов, использованных в экспериментах, с их крат­кими характеристиками.

3. Изображения электрических схем для испытания дешифратора, шифратора, демультиплексора и мультиплексора.

4. Копии временных диаграмм и таблицы переключений, отображающие работу исследуемых преобразователей кодов.

5. Выводы по работе.

**7.4. Запоминающие узлы. Регистры**

**Практические задания**

**Задание 1.** Запустить среду разработки Multisim, собрать на рабочем поле среды Multisim схемудля испытания *универсального регистра сдвига* (рис. 7.32) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопировать схему на страницу отчета.

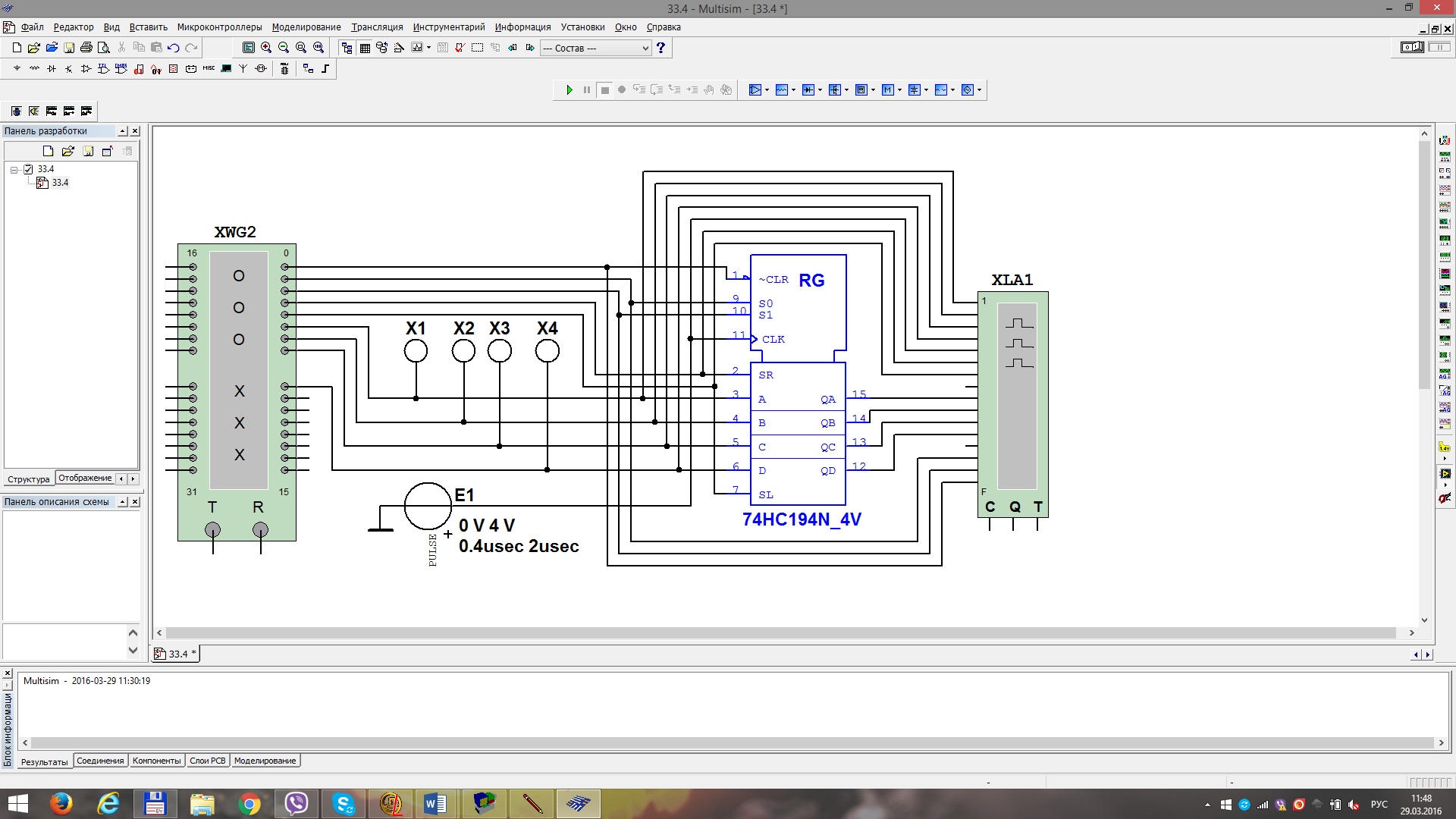


Рис. 7.32. Схема для испытания универсального регистра сдвига

Универсальный 4-разрядныйрегистр сдвига 74НС194N\_4V (отечественные аналоги-микросхемы К230ИР2, КМ155ИР1, К176ИР3) способен сдвигать информацию и вправо, и влево, возможна как параллельная, так и последовательная запись данных. Регистр имеет параллельные входы (А, В, С, D), параллельные выходы (QA, QB, QC, QD), последовательные входы (SR, SL), цепь прямой очистки регистра по входу  и управляющие входы (S0 и S1) – входы задания режима:

S0 = 1, S1 = 1 – запись данных в регистр по входам А, В, С, D;

S0 = 1, S1 = 0 – сдвиг данных влево в направлении от QА к QD;

S0 = 0, S1 = 1 – сдвиг данных вправо в направлении от QD к QА;

S0 = 0, S1 = 0 – входы регистра недоступны (блокировка).

**Задание 2.** Составить план исследования параллельного регистра сдвига, заполнив ячейки памяти генератора слова XWG1 на основе правил функционирования регистра 74НС194\_4V, отраженных в таблице 7.6.

Таблица 7.6

**План исследования параллельного регистра сдвига**

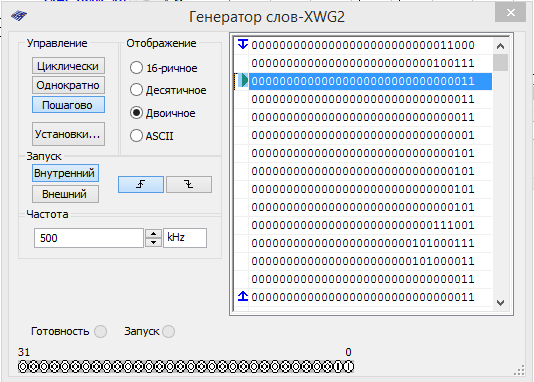
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входы | | | | | | | | | | Выходы | | | |
| Сброс | Старт | Режим | | Последовательный вход | | Параллельный вход | | | |
|  |  | S0 | S1 | SR | SL | А | В | C | D | QA | QB | QC | QD |
| 0 | × | × | × | × | × | × | × | × | × | 0 | 0 | 0 | 0 |
| 1 | 0 | × | × | × | × | × | × | × | × | QА0 | QВ0 | QС0 | QD0 |
| 1 | ↑ | 1 | 1 | × | × | А | В | C | D | А | В | C | D |
| 1 | ↑ | 1 | 0 | 1 | × | × | × | × | × | 1 | QА*п* | QВ*п* | QС*п* |
| 1 | ↑ | 1 | 0 | 0 | × | × | × | × | × | 0 | QА*п* | QВ*п* | QС*п* |
| 1 | ↑ | 0 | 1 | × | 1 | × | × | × | × | QВ*п* | QС*п* | QD*п* | 1 |
| 1 | ↑ | 0 | 1 | × | 0 | × | × | × | × | QВ*п* | QС*п* | QD*п* | 0 |
| 1 | × | 0 | 0 | × | × | × | × | × | × | QА0 | QВ0 | QС0 | QD0 |

*Примечание.* 0 – низкий уровень; 1 – высокий уровень; × – любое состояние; ↑ – по­ложительный перепад (с низкого уровня на высокий); QА0, QB0, QС0, QD0 – стационарные уровни А, В, С, D до установки указанных состояний на входах; QА*п*, QВ*п*, QС*п*, QD*п* – соответственно уровниА, В, С, D перед началом прохождения фронта самого последнего тактового импульса.

Запустить программу моделирования параллельного регистра, скопировать в отчет программу и временные диаграммы сигналов на входах и выходах регистра.

Ввод (запись) и вывод (считывание) информации производится параллель­ным кодом. Ввод обеспечивается тактовым импульсом, с приходом очередного тактового импульса записанная информация обновляется. Считывание информации происходит в прямом коде в интервале между синхроимпульсами, когда триггеры находятся в режиме хранения.

Руководствуясь схемой соединения генератора XWG1 с регистром (см. рис. 7.32), при записи чисел в ячейки памяти генератора в младший разряд 9-разрядных чисел нужно заносить значение сигнала : логический 0 для очистки регистра или логическая 1 − разрешение записи числа, сдвига данных и др.; в следующие два разряда – значения (1 или 0) сигналов S0 и S1, определяющих режим работы регистра; в два следующих − вводить значения сигналов SR и SL, определяющих направление сдвига записанной информации в направлении от QА к QB, QC, а затем к QD после каждого положительного перепада импульса на тактовом входе  или, наоборот, от QD к QА. В старшие разряды нужно занести про­извольные (или по указанию преподавателя) значения 4-раз­ряд­ных чисел DCBA, которые передаются на соответствующие выходы.



S0

S1

SL

SR



Шаги

Рис. 7.33. Запись 9-разрядных кодовых комбинаций в 15 ячеек памяти генератора

В качестве примера на рис. 7.33 приведена запись 9-разрядных кодовых комбинаций в 15 ячеек памяти генератора XWG1, а на рис. 7.34 – реализация программы моделирования параллельного регистра в виде вре­менных диаграмм сигналов (выводимых в окне анализатора XLA1) на его входах и выходах при шаговом режиме работы генератора XWG1.

На первом шаге (первом такте работы генератора и регистра) при подаче сигнала  = 0 (см. первую стро­ку табл. 7.6 и рис. 7.34) на всех входах и выходах регистра установились нулевые значения. На втором шаге при  = 1, SR = 0, SL = 0 (разрешение записи числа в регистр), S0 = 1 и S1 = 1 происходит загрузка 4-раз­рядного двоичного числа DСВА = 0001 в регистр.

При задании направления сдвига данных влево (S0 = 1 и S1 = 0, такт или шаг 3) сигнал 0001 выводится на выходы: QD = 0, QC = 0, QB = 0 и QA = 1. С приходом очередного тактового импульса (шаги 4, 5 и 6) происходит перезапись (сдвиг) содержимого триггера каждого разряда в соседний разряд (от разряда А к разряду D) без изменения поря­дка следования единиц и нулей. По окончании шестого тактового импульса на выходе устанавливается число 1000 (см. рис. 7.34). Если выполнить еще один шаг при S0 = 1 и S1 = 0, то занесенная в регистр информация будет полностью из не­го выведена. Если при работе регистра в режиме сдвига влево (см. шаги 3, …, 6 на рис. 7.34) в ячейки памяти генератора внести SL = 1, то сигнал 1 будет формироваться на выходе QA и сдвигаться влево от QA к QD при каждом тактовом импульсе. В результате после шестого импульса на выходе установится сигнал 1111.

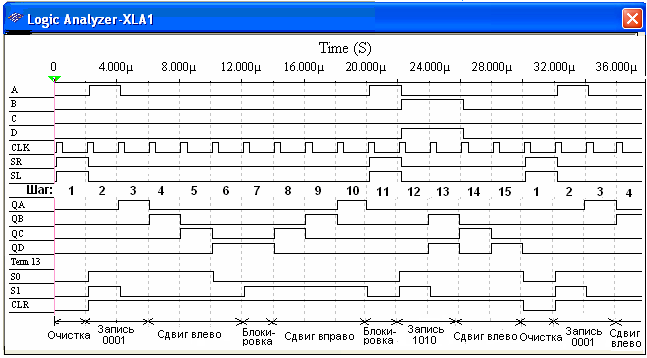


Рис. 7.34. Моделирование параллельного регистра в виде временных диаграмм сигналов

Режим блокировки реализуется при подаче на оба управляющих входа сигналов низкого уровня, т. е. S0 = S1 = 0 при  = 1 (см. шаг 7 и шаг 11 на рис. 7.34). В режиме блокировки данные в регистре не сдвигаются ни вправо, ни влево, а остаются на своих прежних позициях. При установке сигналов S0 = 0 и S1 = 1 с приходом 8, 9 и 10 тактовых импульсов происходит сдвиг сигнала 1000 вправо и его полный вывод из регистра. Если при работе регистра в режиме сдвига вправо (см. шаги 8, …, 10, на рис. 7.33) в ячейки памяти генератора внести SR = 1, то сигнал 1 будет формироваться на выходе QD и сдвигаться вправо от QD к QA при каждом тактовом импульсе. И, как следствие, после десятого импульса на выходе установится сигнал 1111.

При установке S0 = S1 = 0 с приходом 11-го импульса происходит блокировка выходов, на следующем шаге выполняется параллельная запись числа DCBA = 1010 в регистр, далее сдвиг данных влево и т. д.

**Задание 3.** Запустить среду разработки Multisim и собрать на рабочем поле среды Multisim схему для испытания *последовательного регистра сдвига* (рис. 7.35) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопировать схему в отчет.

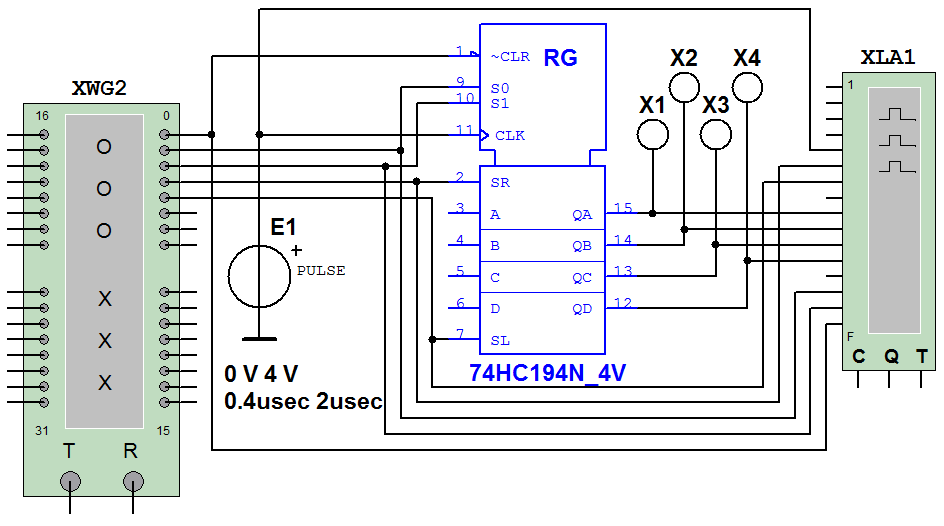


Рис. 7.35. Схема для испытания последовательного регистра сдвига

Чтобы микросхема 74HC194N\_4V работала в качестве *последовательного регистра сдвига влево*, нужно подать на управляющий вход S0 высокий уровень напряжения, а на вход S1 – низкий уровень, т. е. установить S0 = 1 и S1 = 0, и подавать в последовательной форме на вход SR данные, например 1, 0, 1 и 0, которые записываются в разряд А и передаются на выход QA (рис. 7.36). Регистр последовательно сдвигает влево эти сигналы от QA к QD, на выходе QD они теряются (см. шаги 3, …, 9 на рис. 7.36).

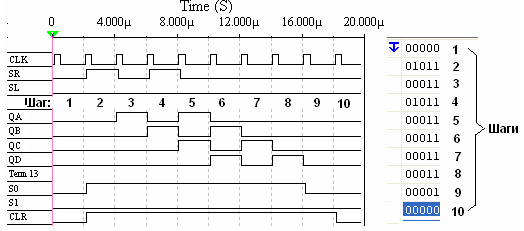


Рис. 7.36. Моделирование последовательного регистра сдвига влево в виде временных диаграмм сигналов

При установке S1 = 0 и S1 = 1 и подаче на вход SL данных в последовательной форме, например 1, 0, 0 и 1, которые записываются в разряд D (и передаются на выход QD), микросхема работает в режиме *последовательного регистра сдвига вправо* (без кольцевого перемещения данных): сигналы 1, 0, 0 и 1 сдвигаются по направлению к разряду А, на выходе QA они теряются (см. шаги 3, …, 9, рис. 7.37).

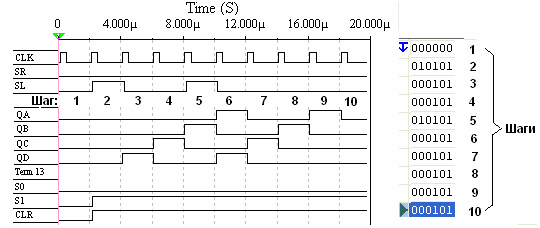


Рис. 7.37. Моделирование последовательного регистра сдвига вправо в виде временных диаграмм сигналов

**Задание** **4.** Составить план исследования последовательного регистра 74НС194\_4V, заполнив ячейки памяти генератора XWG1 произвольными (или по заданию преподавателя) 4-разрядными кодовыми комбинациями, вводимыми последовательно сперва в регистр А, а затем в регистр D.

Запустить программу моделирования последовательного регистра, скопировать в отчет временные диаграммы сигналов на входах и выходах регистра при сдвиге данных влево (см. рис. 7.36) и вправо (см. рис. 7.37).

**Содержание отчета**

1. Название и цель работы.

2. Перечень приборов, использованных в экспериментах, с их крат­кими характеристиками.

3. Изображения электрических схем для испытания параллельного и последовательного регистров.

4. Копии временных диаграмм, отображающих работу исследуемых регистров.

5. Выводы по работе.

**7.5. Запоминающие узлы. Счетчики**

**Практические задания**

**Задание 1.** Запустить среду разработки Multisim, собрать на рабочем поле среды Multisim схемудля испытания *синхронного двоичного счетчика* (рис. 7.38) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему на страницу отчета.

В библиотеке программной среды Мultisim имеются 4-, 8- и 12-разрядные счетчики различных типов. Среди них: счетчики асинхронные (**SN7493**, **SN74393**), синхронные (**74NC161**, **SN74163**), реверсивные (**SN74191**), двоично-десятичные (**SN7493**, **SN74160**) и др.

В схему (рис. 7.38) включен синхронный двоичный 4-разрядный счетчик **74НС161**, к входу  которого подключен источник тактовых импульсов **Е1**, а к выходам **QA**, **QB**, **QC** и **QD** − шестнадцатеричный 7-сегментный индикатор **DCD\_HEX** и дешифратор **DC** 4×10. Выход дешифратора соединен с входами логического анализатора **XLA1**.

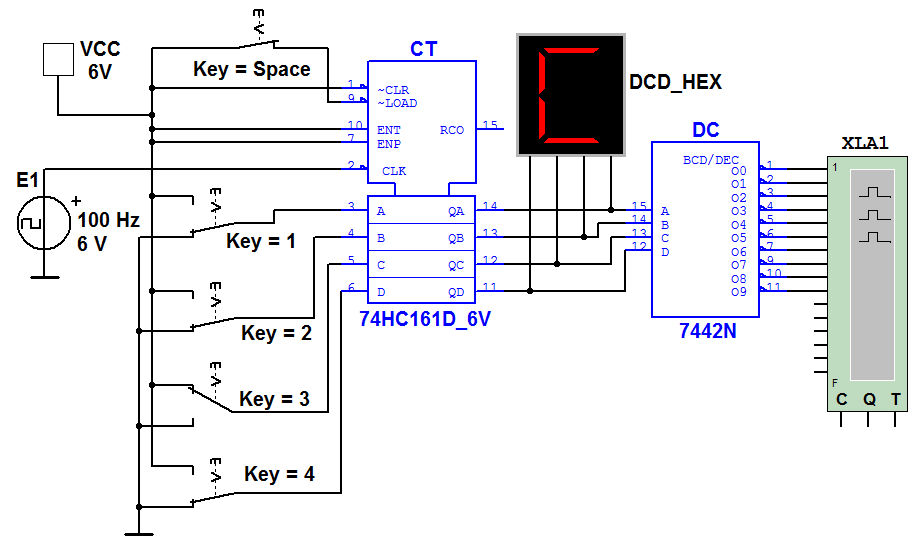


Рис. 7.38. Схема для испытания синхронного двоичного счетчика

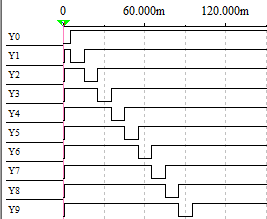
К входам **А**, **В**, **С** и **D** счетчика **СТ** подключен источник постоянного напряжения **VCC**, переключатели **1**, …, **4** для формирования входных двоичных кодов и ключ **Space** для изменения режима работы счетчика. В синхронном счетчике заданные с помощью ключей уровни сигналов подаются на входы всех триггеров, как и тактовые импульсы, которые подаются на счетные входы  всех разрядов счетчика.

При *замкнутом* ключе **Space** число поданных от генератора **Е1** на вход счетчика импульсов высвечивается на индикаторе **DCD\_HEX** в десятичном коде, от 0 до 15, после чего счетчик обнуляется и вновь начинается счет. При этом на одном из выходов дешифратора **DC** формируется сигнал низкого уровня (логический 0), номер которого соответствует коду входного числа: от 0000 до 1001 (910).

При *разомкнутом* ключе **Space** сформированное с помощью переключателей на входе счетчика 4-разрядное двоичное число высвечивается на индикаторе в десятичном коде, а на экране анализатора на одном из выходов, соответствующем входному коду счетчика, формируется логический 0.

**Задание 2.** Замкнутьключ **Space**, запустить программу моделирования суммирующего счетчика и наблюдать за показаниями индикатора. Убедиться, что на экране анализатора **XLA1** логические нули перестают формироваться после прихода 11-го тактового импульса и появляются вновь только с приходом 17-го импульса (рис. 7.39).

Разомкнутьключ **Space**. Установить в диалоговом окне анализатора **XLA1** напряжение **V** = 5 B, частоту таймера *fa* = 2 кГц, число импульсов, приходящихся на одно деление, **Clocks/div** = 60. (При таком режиме лучи медленно перемещаются на экране анализатора). С помощью активных клавиш 1, 2, 3 и 4 клавиатуры сформировать произвольные (или по указанию преподавателя) двоичные входные числа (коды), например 1001, 0011, 0000, 1110 и подавать их на входы **D**, **С**, **В** и **А** счетчика. Убедиться, что при подаче числа 11102 (1410) ни на одном выходе дешифратора 4×10 не сформировался низкий уровень сигнала.



Каналы

Рис. 7.39. Результаты моделирования синхронного суммирующего счетчика

Скопировать на страницу отчета результаты моделирования синхронного суммирующего счетчика (см. рис. 7.39).

**Задание 3.** Собрать на рабочем поле среды Multisim схему для испытания *реверсивного двоичного счетчика* (рис. 7.40) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопировать схему на страницу отчета.

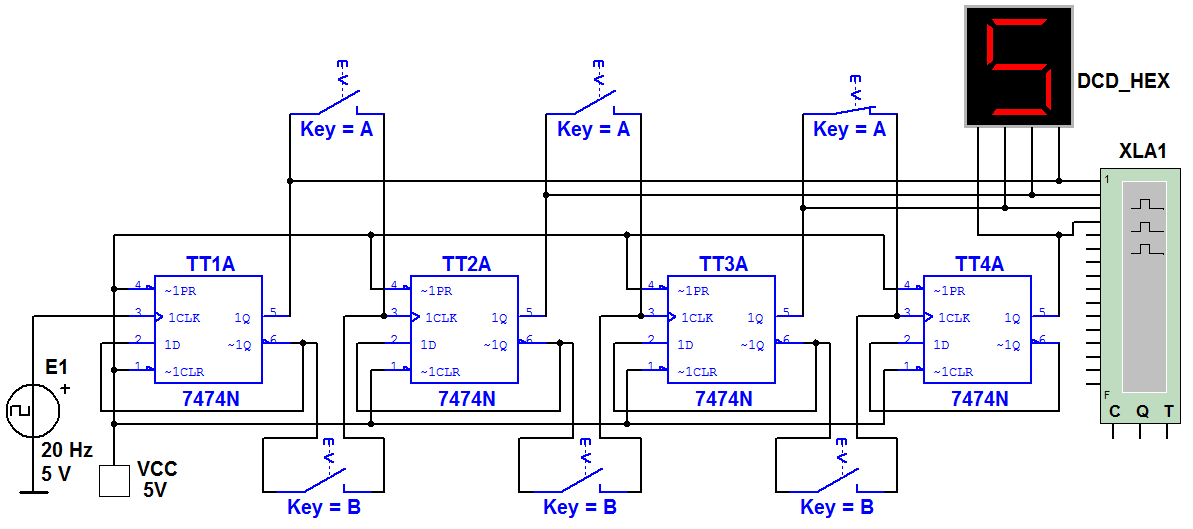


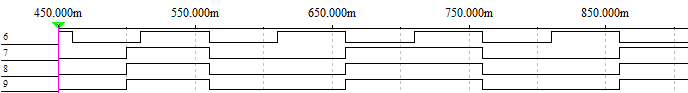
Рис. 7.40. Схема для испытания реверсивного двоичного счетчика

В схеме реверсивного двоичного счетчика (рис. 7.40) с помощью групп ключей **А** и **В** осуществляетсязамыкание или размыкание выходов высокого или низкого уровня предыдущего триггера с входами или  триггера следующего разряда, причем при замкнутых ключах **А** и разомкнутых **В** (режим суммирования) с каждым тактовым импульсом увеличивается результат счета, а при замкнутых ключах **В** и разомкнутых **А** (режим вычитания) − результат счета уменьшается.

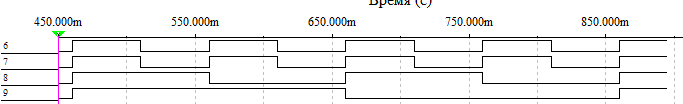
Установить в диалоговом окне анализатора **XLA1** напряжение **V** = 5 B, частоту таймера *fa* = 2 кГц, число импульсов, приходящихся на одно деление, **Clocks/div** = 60. Разомкнуть ключи **В** и замкнутьключи **А**. Запустить программу моделирования счетчика.

При высвечивании числа 15 на 7-сегментном индикаторе щелкнуть мышьюна кнопке **Stop** (остановки моделирования) и скопировать окно анализатора с результатами моделирования на страницу отчета (см. верхнюю часть рис. 7.41).

Разомкнутьключи **А** и замкнутьключи **В**. Щелкнуть мышьюна кнопке **Stop** (продолжить моделирование), остановить моделирование при высвечивании числа 0 на индикаторе и скопировать окно анализатора с результатами моделирования в отчет (см. нижнюю часть рис. 7.41).



Число 1 2 3 4 5 6 7 8 9



Число 9 8 7 6 5 4 3 2 1

Рис. 7.41. Результаты моделирования реверсивного двоичного счетчика

**Задание 4.** Собрать на рабочем поле среды Multisim схему для испытания *десятичного счетчика* (рис. 7.42) и установить в диалоговых окнах компонентов их параметры или режимы работы. Скопироватьсхему на страницу отчета.

Результаты моделирования 4-разрядного двоичного счетчика (см. рис. 7.39 и 7.40) показали, что с его помощью можно сосчитать до 15.

Следовательно, для создания счетчика натуральных десятичных чисел в двоичном коде для одной декады нужно в схему двоичного 4-разрядного счетчика ввести обратные связи с логическими элементами, посредством которых сигнал с какого-то старшего разряда поступает в младшие и т. п., обеспечивая в целом выработку счетчиком двоичного эквивалента счетной декады.

В функциональной схеме десятичного счетчика (рис. 7.42), собранной на триггерах *JK*-типа, на каждый одиннадцатый тактовый импульс результат счета сбрасывается в нуль и далее результат счета увеличивается. Возврат счетчика при поступлении одиннадцатого тактового импульса в начальное состояние обеспечивается дополнительной комбинационной схемой с встроенными логическими элементами И (**AND**), ИЛИ (**OR**) и НЕ (**NOT**).

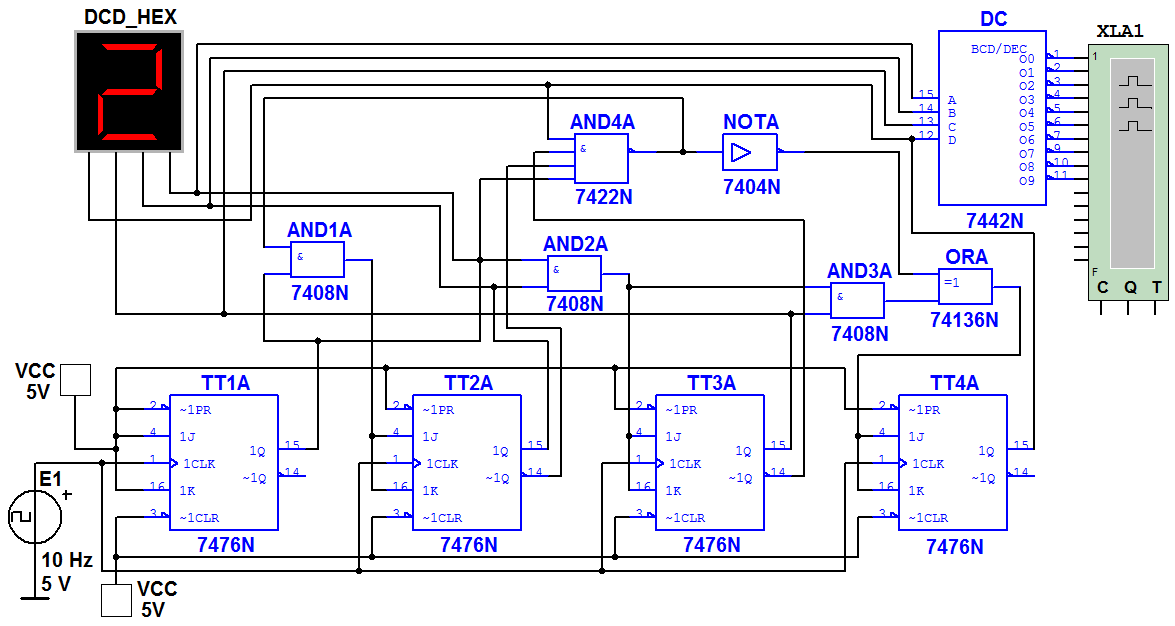


Рис. 7.42. Схема для испытания десятичного счетчика

Счетчик работает так же, как синхронный двоичный счетчик до поступления седьмого импульса, а далее, благодаря обратной связи, нарушается изменение естественной последовательности двоичных чисел на входах и выходах триггеров при подсчете тактовых импульсов.

Запустить программу модели­рования десятичного счетчика и скопировать окно анализатора с результатами моделирования на страницу отчета.

**Содержание отчета**

1. Название и цель работы.

2. Перечень приборов, использованных в экспериментах, с их крат­кими характеристиками.

3. Изображения электрических схем для испытания суммирующего, реверсивного и десятичного счетчиков.

4. Копии временных диаграмм, отображающие работу исследуемых счетчиков.

5. Выводы по работе.