МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

ОРГАНИЗАЦИЯ СУПЕРСКАЛЯРНЫХ ПРОЦЕССОРОВ. ОРГАНИЗАЦИЯ ПОДСИСТЕМЫ КЭШ ПАМЯТИ ПРОЦЕССОРОВ

Методические указания для самостоятельной работы

для студентов направления подготовки 09.03.01

Курск 2016

УДК 621.3
Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *А.Г. Спеваков*

Организация суперскалярных процессоров. Организация подсистемы кэш памяти процессоров: методические указания для самостоятельной работы по дисциплине «Теоретические основы организации многопроцессорных комплексов и систем» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 19 с.

Излагаются методические рекомендации к выполнения самостоятельной работы при освоении основ организации многопроцессорных комплексов и систем.

Предназначены для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60х84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Организация суперскалярных процессоров

Суперскалярным процессором (впервые так назван в 1987 году) называется центральный обрабатывающий процессор, который за один конвейерный такт конвейера команд одновременно выполняет более, чем одну скалярную команду, и в настоящие время – от 4-х до 8-ми RISC команд.

Исторически появление суперскалярных процессоров связано с переходом в микропроцессорах от CISC архитектур с полным набором команд к RISC архитектурам с сокращенным наборам команд. Главные отличия RISC архитектуры:

- максимально сокращено число команд, имеющих доступ к кэш и основной оперативной памяти, и введены специальные команды обращения в память: команды «Чтение» и «Запись»;
- все основные команды являются командами типа «регистр-регистр»;
- длины всех команд стандартные однословные, равные ширине шине данных, что позволило унифицировать конвейер команд;
- число форматов команд минимальное: не более 4-х;
-число способов адресации не более 4-х;
-число команд сравнительно невелико: не более 128-ми;
-75 % процентов команд при конвейерной их обработке выполняются за один конвейерный такт;
-25% процентов команд выполняется по подпрограммам, составленным в основных коротких RISC-командах, расходуя в среднем 4-5 RISC команд на эмуляцию CISC-команды;
- сравнительно большой файл регистров общего назначения: от 32 до 512;
- устройство управления обязательно реализуется на «жесткой» логике и оно гораздо (в 5 раз) проще и более быстродействующее, чем в CISC процессорах.

Хотя RISC-программы в среднем на 30% длиннее CISC- программ, они выполняются в 1,5 раза быстрее, чем CISC- программ.

Переносимость старых CISC-программ, т.е. преемственность программного обеспечения, достигается путем встраивания в RISC-процессоры аппаратного транслятора CISC-команд в RISC- команды. Таким образом, в системе команд RISC-процессора появляются команды, отсутствовавшие в CISC –процессорах: «Чтение» (иногда называются «Загрузка» в регистры); «Запись» (иногда называется «Сохранение» результатов в кэше данных); «Переход».

После перехода к RISC-архитектуре и широкого внедрения конвейера команд с шестью стадиями (ступенями):

-выборка команды;
-декодирование КОП;
-вычисление адреса операнда;
-выборка операнда;
-исполнение команды;
-запись результата;

было замечено, что самой длительной ступенью является пятая «исполнение команды» и она определяет длительность конвейерного такта синхронного конвейера:

$Tk=\max\{t1, t2, t3, t4, t5, t6,\}=t5$

При этом 4-я стадия для всех RISC-команд завершается гораздо быстрее, чем 5-я, т.е. 1-4-я ступени конвейера команд простаивают и могли бы за время t5 конвейерно подготовить к исполнению несколько команд. Тогда для первых 4-х стадий нужно выбрать свой более короткий конвейерный такт $Tk1=\max\{t1, 2, t3, t4\}$,который в среднем оказался в 4-5 раз меньше, чем t5ср, $Tk1\approx 1/5\ t5ср$. Следовательно, первые четыре ступени могут заготовить операнды не на одно АЛУ, а на 4-5 АЛУ. Если быстро записать результаты всех 5-ти АЛУ в память, а в RISC-архитектуре это возможно, так как

результаты записываются только в регистры или в быструю кэш-память данных, то на 5-й ступени конвейера команд можно организовать параллельную работу нескольких АЛУ с фиксированной запятой и дополнительных функциональных блоков типа «переход», «запись», «чтение». Далее стало очевидным, что параллельно с ним в 5-й ступени можно включить конвейерные АЛУ с плавающей точкой. Тогда на операциях с плавающей точкой конвейер команд удлинится на 4-ре ступени, так как конвейер с п.3. содержит 5 ступеней, его общая длина составит до 10 ступеней.

Структура суперскалярного конвейера команд

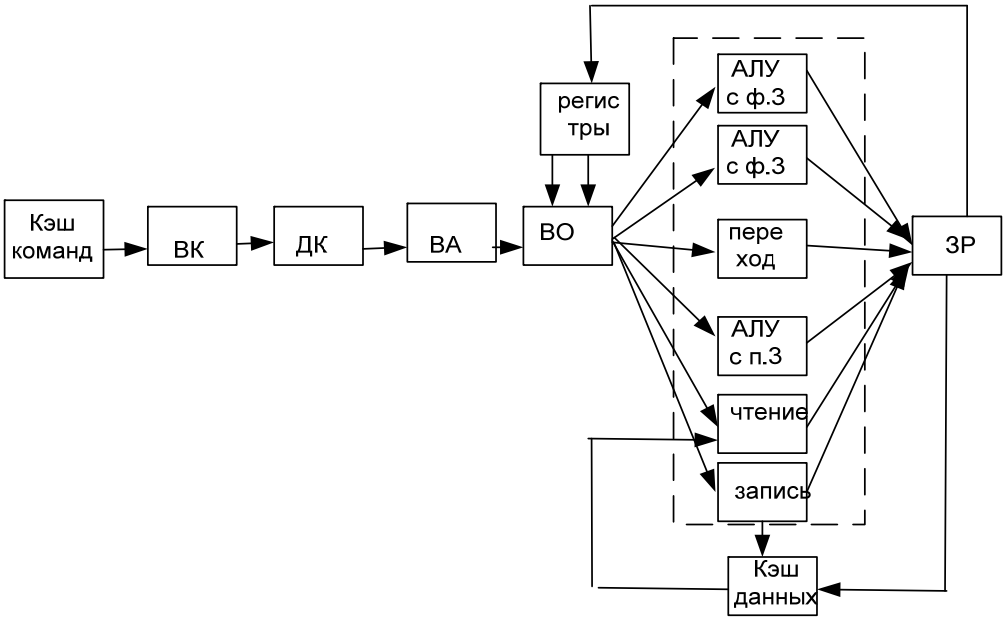

Рис. 1. Структурная организация суперскалярного процессора

Эта идея новой организации конвейерного процессора появилась в 1987 году и успешно была внедрена в отечественной супер-ЭВМ «Эльбрус». Но от идеи до серийного выпуска суперскалярных микропроцессоров прошло еще 8-10 лет. За это время было доказано и реализовано, что с помощью встроенных дополнительных средств можно автоматически выявлять скрытый в последовательных программах однопроцессорных фон Неймановских машин параллелизм на уровне команд и за счет этого повысить скорость их обработки в среднем в 2-8 раз. Эти аппаратные средства оказались весьма сложными и потребовали для своей реализации до 3-4 млн. транзисторов. Рассмотрим основы их функциональной организации.

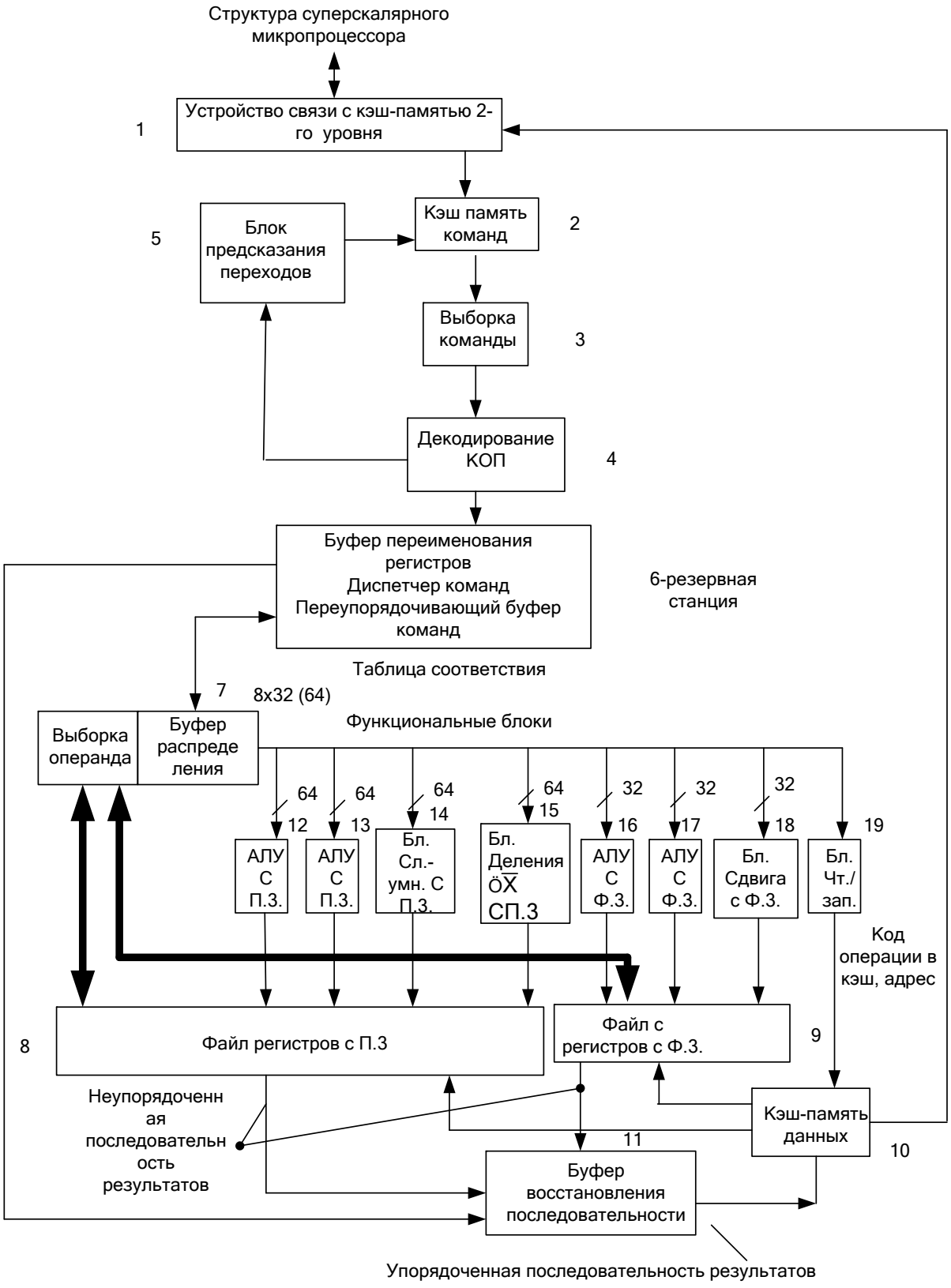

Рис. 2. Детализация суперскалярной организации

Блок предсказания переходов введен для повышения пропускной способности конвейера команд на основании истории переходов, а также для снижения частоты промахов при обращении в кэш- память команд.

Это позволяет выполнять команду по предположению (агрессивно на недостоверных «спекулятивных» операндах) с опережением по отношению к моменту

времени окончательного определения направления условного перехода, от которого данная команда зависит.

Блоки 6, 7, 11 предназначены для выявления скрытого параллелизма команд в наследуемых последовательных программах однопроцессорной ЭВМ фон Неймана и организации параллельной обработки распараллеленных RISC-команд.

Основные виды скрытого параллелизма, выявляемого аппаратными средствами суперскалярного процессора:

-независимость между командами на линейных участках программ;

-возможность изменения порядка последовательной обработки независимых команд.

В старых последовательных программах фактически независимые между собой команды вынужденно из-за наличия только одного АЛУ искусственно объединяются в последовательные линейные участки (ветви). При этом с целью экономии регистровой памяти один и тот же регистр и одна и таже ячейка ОП последовательно путем замещения содержимого используется вначале для хранения значения одной переменной, а после его использования замещается значением другой переменной. Из-за этого фактически независимые команды условно считаются информационно связанными. Например, исходный последовательный участок

1.A=B+C,
2.B=A+E,
3.A=X+Y,
4.F=A*Z.

организован как последовательный линейный путем занесения на третьем шаге в одну и ту же ячейку A результата обработки новых переменных X и Y, не связанных с предыдущим отрезком участка. Если ввести дополнительную переменную A1 и дополнительный регистр для хранения ее значения, то эти два последовательных отрезка линейного участка можно выполнить параллельно:

1. A1=B+C,	1. A=X+Y,
2. B=A1+E	2. F=A*Z.

Такой способ преобразования последовательных участков в несколько параллельных отрезков называется метод переименования регистров (или переменных).

Переименование регистров может не потребоваться, если на линейном участке исходной последовательной программы независимые отрезки искусственно объединены программистом в одну линейную ветвь. Такие отрезки выявляются в динамике исполнения программы на основе принципа потоковой обработки по степени готовности их операндов. При диспетчеризации команд создается буфер команд (окно исполнения) и постоянно выполняется проверка готовности их операндов. Любая из команд буфера направляется на обработку как только будет готовы ее операнды. В этом случае может измениться очередность обработки команд по сравнению с порядком их исполнения, предусмотренным программистом наследуемой последовательной программы. В то же время такой принцип направления команд на обработку позволяет естественно перейти от последовательной их обработки к параллельной. Этот метод выявления скрытого параллелизма называется переупорядочиванием команд. Конечно, после исполнения команд нужно обязательно восстановить упорядоченную последовательность результатов перед их записью в кэш-память данных. Восстановление можно выполнить, если сохранить во входном буфере команд в блоке 6 исходную последовательность команд обрабатываемой программы.

Очевидно, что перед переупорядочиванием команд нужно в начале выполнить распараллеливание путем переименования регистров. Оно выполняется путем выявления типа зависимости по данным:

ЧПЗ - чтение после записи,

ЧПЧ - чтение после чтения,

ЗПЗ - запись после записи,

ЗПЧ - запись после чтения.

ЗПЗ и ЗПЧ - это лишние зависимости, которые появляются в исходных последовательных программах из-за ограниченного числа регистров общего назначения, недостаточной емкости ОП и наличия программных циклов. Тогда если фактически, как в RISC-процессорах, или виртуально увеличить число РОН, запись можно выполнять в любой другой регистр, а не в тот же как в ЗПЗ и ЗПЧ, и тем самым удалить эти лишние зависимости по данным.

После выполнения в конвейере команд этапа декодирования команды по коду операции должен быть выбран требуемый функциональный обрабатывающий блок, созданы указатели на операнды и на место помещения результата и затем выполнена выборка операндов.

В связи с возможностью аннулирования результатов команд при неправильном предсказании перехода, из-за переименования регистров и переупорядочивания команд в суперскалярном процессоре чаще всего применяется неупорядоченная выдача команд на исполнение и неупорядоченное их завершение. При выдаче команд необходимо распределить готовые команды по требуемым функциональным блокам с учетом их занятости. Эта функция называется диспетчеризацией. С целью упрощения диспетчеризации на входах функциональных блоков устанавливаются буферы распределения, в которых хранятся очереди готовых команд (декодированный КОП и готовые значения операндов), например в изображенной структуре требуется 6 параллельных буферов небольшой емкости (не более 5-ти команд в каждом). На однотипные блоки: на 2АЛУ с п.з. и на 2 АЛУ с ф.з. достаточно по одному буферу. Готовые команды из буферов распределения считываются периодически в каждом конвейерном такте конвейеров с плавающей запятой. При этом в блоках с ф.з. они исполняются за одним такой конвейерный такт, а в блоках с П.З - за 5 конвейерных тактов. Готовые команды передаются в требуемые буферы распределения блоком 6 диспетчера команд с учетом значения кода их КОП.

Функции переименования регистров, переупорядочивания и диспетчеризации команд выполняются обычно одним сложным блоком, который называется резервирующей станцией. Основой этого блока чаще всего является кольцевой регистровый FIFO-буфер с перемещаемыми указателями «начало-конец» очереди и память таблицы соответствия, в которой ведется учет местоположений операндов, а именно каждой исходный операнд и какой результат команды, находящейся в FIFO-буфере, хранится в данный момент времени в каком физическом регистре или в каком элементе FIFO-буфера. Эти места хранения операндов устанавливаются после декодирования каждой команды и заносятся в таблицу соответствия. При этом используется механизм динамического отображения логических ресурсов старых программ на физические ресурсы новых микропроцессоров. Каждая команда создает новое имя физического регистра для исходного логического регистра и помещает в него свой операнд или результат. Если последующие команды используют это же значение, то они при декодировании получают в процессоре это же новое имя физического регистра. Эта процедура называется переименованием регистров.

В процессорах с FIFO-буферами число используемых физических регистров РОН всегда выбирается равным числу логических регистров программы. Требуемые дополнительные регистры для переименования используются не из числа РОН, а из FIFO – буфера. Операнды могут находиться либо в физических регистрах, либо временно в FIFO- буфере. В нем всегда хранятся результаты выполненных команд, пока они не достигнут «начала» очереди в буфере. Этот же буфер используется и для переупорядочивания, и для диспетчеризации команд. Его емкость ограничена и рассчитана на 5 -10 команд. Если буфер заполнен, то диспетчеризация приостанавливается до освобождения хотя бы одного места.

Процедура переименования регистров инициализируется после декодирования каждой очередной команды и завершается определением места хранения ее результата. Для этого ассоциативно просматриваются логические адреса (левая часть таблицы соответствия) и сравниваются с логическим адресом результата, указанным в коде команды. Если такая строка в таблице соответствия уже имеется, это значит, что по адресу, указанному в правой ее части уже выполнена запись, а возможно и было чтение. Следовательно, по этому месту записывать новое значение результата нельзя, иначе сохранится лишняя зависимость между командами типа ЗПЗ и ЗПЧ. Тогда начинает заполняться новая строка в таблице соответствия, и этот же логический адрес записывается в левую часть свободной строки таблицы, а правая ее часть пока остается пустой (неопределенной). Затем декодированный КОП этой команды записывается в кольцевой FIFO-буфер. Формат ее записи: минимальной длины <декодированной КОП, место для хранения значения результата этой же команды>. По-видимому, длину записи команды придется увеличить для того, чтобы реализовать все функции резервирующей станции. Эта новая команда заносится в конец FIFO - очереди, поймав при циклическом сдвиге содержимого буфера отметку «конец» очереди и перенеся эту отметку на новую команду. Одновременно по счетчику числа сдвигов от подвижного «начала» FIFO-очереди определяется адрес в FIFO – буфере для записи результата этой команды и копируется в правую часть новой строки таблицы соответствия. Отметки «начало» и «конец» FIFO-очереди в буфере обнаруживаются при прохождение мимо неподвижного окна по ходу циклических сдвигов содержимого FIFO-буфера. Аналогично по счетчику, сбрасываемому отметкой «начало», легко найти в дальнейшем это же место для последующей записи или чтения значения результата какой-либо команды, адресуемого содержимым правой части строки таблицы соответствия. Итак, имеется два возможных места хранения результата команды либо в регистре РОН, либо в FIFO-буфере. Место его хранения в РОН назначается и заносится в таблицу соответствия, если при ассоциативном ее просмотре его логическим адресом такой заполненной строки не обнаружится т. е строка соответствия выносится впервые. В таблицу заносится содержимое счетчика занятых регистров РОН с инкрементом +1 в строку, поименованную в левой части логическим адресом результата.

Исходные данные, читаемые из кэш-памяти данных, записываются в регистры РОН, а их местоположение описывается аналогично в таблице соответствия < логический адрес-адрес физического регистра >.

Каждая очередная декодированная команда по результату обращения в таблицу соответствия логическими адресами своих операндов может оказаться в двух возможных состояниях: готова к использованию, не готова. Проверка готовности выполняется очень просто. Если после ассоциативного опроса левой части таблицы логическим адресом операнда такая строка в таблице найдена, то этот операнд готов, а по содержимому ее правой части ясно, где он хранится: в регистре или в FIFO-буфере. Если оба операнда по таблице соответствия готовы, команда готова к исполнению. Рассмотрим вначале принцип обработки такой готовой команды. Аналогично предыдущему, опрашивая таблицу соответствия логическим адресом результата, заполняется новая ее строка и в случае необходимости выполняется переименование регистра. Если можно запланировать помещение ее результата без переименования в регистр РОН, то готовая команда сразу же направляется на исполнение, а именно по ее декодированному КОП находится буфер распределения и туда переносятся значения ее операндов либо из РОН, либо из FIFO-буфера. Адреса этих операндов, как известно, имеются в правой части таблицы соответствия и находятся в результате обращения в таблицу по логическим адресам операндов команды. После исполнения команды – вновь обращение в таблицу логическим адресом ее результата и запись результата по адресу в правой ее части, либо в регистр РОН, либо в FIFO-буфер. Следовательно, логический адрес результата также

нужно хранить в буфере распределения и извлекать его оттуда после окончания исполнения очередной команды.

Если декодированная команда оказалась неготовой, т.е. в таблице соответствия нет указателей на место хранения одного или обоих ее операндов, она должна ожидать пока не будут готовы операнды. Тогда ее код заносится в FIFO-буфер в FIFO-очередь. Она будет циркулировать в кольцевом FIFO - буфере до тех пор, пока в таблице соответствия не появятся ссылки на места хранения ее операндов. Следовательно, в FIFO-буфер нужно записывать и логические адреса ее операндов (один или два). Если заносится два, то после проверки готовности и наступления готовности одного из операндов его логический адрес FIFO- буфере нужно отмечать тегом готовности. Готовность операндов проверяется и уточняется в момент прохождения при циклическом сдвиге мимо неподвижного окна. Если логические адреса операндов еще не отмечены тегами готовности, тогда сдвиг FIFO-буфера приостанавливается, неготовый логический адрес считывается из FIFO-буфера, по нему выполняется обращение в таблицу соответствия, и, если оно было успешным, данный логический адрес в FIFO-получает тег готовности, а FIFO-буфер продолжает циклические сдвиги. Если же оба логических адреса тегированы как готовые, то по декодированному КОП быстро находится требуемый буфер распределения, куда путем переадресации через таблицу соответствия переписываются из РОН или FIFO-буфера значения операндов, и команда наконец уходит на исполнение. Результат ее будет записан либо в РОН, либо в FIFO-буфер с учетом переадресации по его логическому адресу через таблицу соответствия, а эта строка переадресации, как известно, создается сразу же после декодирования команды. Следовательно, логический адрес результата тоже нужно хранить в FIFO-буфере по всем неготовым командам и переписывать его в буфер распределения вместе со значениями готовых операндов.

Любая готовая и уже исполненная команда стирается в кольцевом FIFO-буфере только после фиксации достоверных ее результатов, заключающейся в записи ее результатов в выходной буфер восстановления последовательности. По принципу FIFO-очереди она всегда в «начале» очереди. Как только при циклическом сдвиге она появится в неподвижном окне, содержимое ее ячейки стирается, а метка «начало очереди» переносится на следующую ячейку буфера. Переполнение FIFO-буфера обнаруживается, когда в неподвижном окне за меткой «конец очереди» сразу же появится метка «начало очереди» в соседней ячейке кольцевого буфера. Тогда диспетчеризация команд приостанавливается путем приостановки конвейера команд.

Несколько иная ситуация с удалением из FIFO-буфера команды, внесенной в него по процедуре переименования регистров. Она сохраняется в регистре до тех пор, пока значение ее результата не будет использовано, т.е. не только будет выполнено его копирование в один из буферов распределения, но и команда, его затребовавшая, не будет исполнена, а результат последней не будет записан. Только после этого команда-источник переименованного операнда стирается в FIFO-буфере, а значение ее результата переписываются в тот регистр РОН, который по таблице соответствия соответствует месту хранения логического результата. Это необходимо для того, чтобы, во-первых, исключить использование в качестве операнда устаревшего значения логического регистра и, во-вторых, не допустить, чтобы очередная команда из-за нарушения последовательности выполнения команд занесла свой результат в регистр еще до того, как это сделала предшествующую команда, т.е. не нарушить последовательность использования с замещением записей одного и того же логического регистра, предусмотренную исходной последовательной программой. В FIFO-буфере команды сохраняются до фиксации их результатов в выходном буфере восстановления последовательности. Значения условных переходов, вычисленные по команде перехода, не фиксируются, а используются для управления очисткой FIFO-буфера от неправильно предсказанных команд и их ложных результатов, также еще не зафиксированных.

Поэтому, когда команда условного перехода достигает начала очереди в FIFO-буфере и выясняется, что предсказание было неправильным, содержимое буфера гасится, и начинается выборка команд из другой ветви. Так как в FIFO-буфере циркулируют команды разных типов: внесенные по переименованию, неготовые команды, а некоторое время и готовые команды, а также ложные неправильно предсказанные команды, в общем случае нужно команду хранить в его ячейке в широком формате < декодированный КОП; тег, логический адрес операнда; логический адрес результата; место для записи значения результата >. Утешает только то, что число хранимых команд (окно исполнения) невелико: 5-10 команд.

Таким образом, с целью обеспечения максимальной загрузки параллельных функциональных блоков в суперскалярном процессоре применяется неупорядоченная выдача и неупорядоченное завершение команд. Поэтому на выходе перед записью результата в кэш – данных их нужно упорядочить в соответствии с логической их адресацией, предусмотренной в исходной программе. Это сделать сравнительно просто по таблице соответствия с помощью дополнительного буфера восстановления последовательности. При перезаписи из регистра РОН в этот буфер нужно ассоциативно адресом регистра обратится теперь в правую часть таблицы соответствия, а из левой части найденной строки прочитать истинный логический адрес значения этого регистра РОН и передать его в выходной буфер вместе со значением, а далее использовать как адрес в памяти данных при записи в КЭШ.

Организация подсистемы кэш памяти процессоров

Для повышение производительности целесообразно использовать параллелизм в операциях, связанных с обращениями к памяти.

Однопортовое ОЗУ

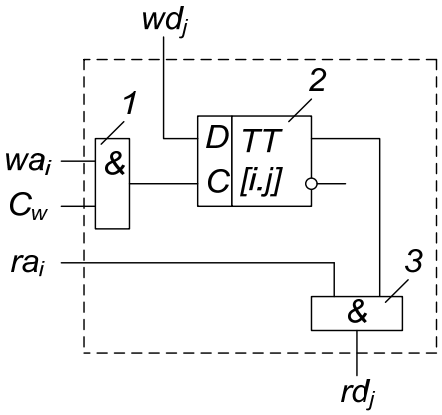

Рис. 3. Ячейка однопортового ОЗУ

Синхросигнал C_w проходит на вход триггера только в том случае, если выбрана строка wa_i (открыт элемент И 1).

Данные на выход rd_j проходят только в том случае, если выбрана строка ra_i (открыт элемент И 3).

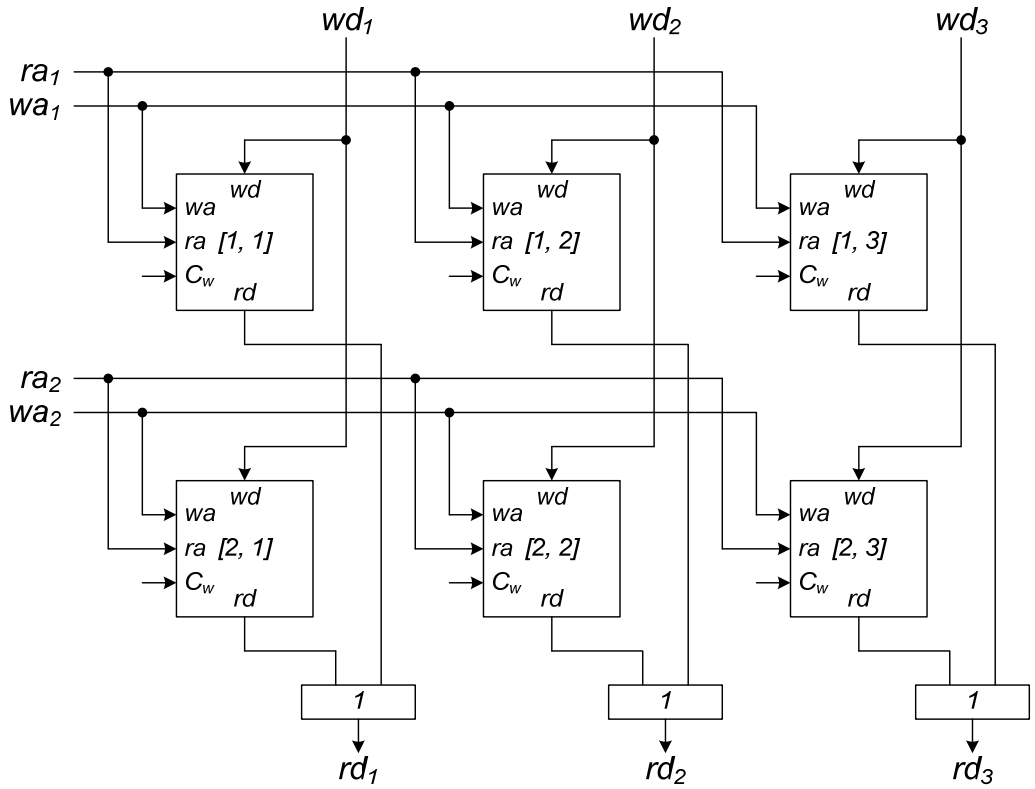

Рис. 4. ОЗУ 2 слова × 3 бита

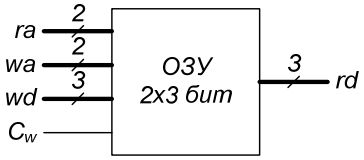

Рис. 5. ОЗУ 2×3 схема

Для **чтения** на вход *ra* (Read Address) подается вектор «0…010…0» с единицей в разряде, который необходимо прочитать (вектор может быть получен с выхода дешифратора). На выходе *rd* (Read Data) появляется результат (выбранная единицей в *ra* строка, остальные замаскированы).

Время записи:

$t_w = \underbrace{t_0}_{И1} + \underbrace{2t_0}_{ТТ} = 3t_0 = const$ – не зависит от структуры ОЗУ!

Для **записи** на вход *wd* подается слово, которое необходимо записать; на вход *wa* – вектор «0…010…0». Запись происходит по синхросигналу C_w.

Время чтения:

$t_w = \underbrace{t_0}_{И3} + \underbrace{\lceil \log_2 n \rceil t_0}_{ИЛИ} = \left(1 + \lceil \log_2 n \rceil\right) t_0$ – растет с ростом емкости ОЗУ (числа слов *n*)

С целью снижения разрядностей внутренних шин возможно разбиение ОЗУ на банки или организация в виде матрицы (RAS/CAS).

Логарифм появляется при реализации элемента ИЛИ с использованием пирамидальной схемы. Возможно использование двухвходового элемента ИЛИ в составе ячейки и

реализация работы памяти в конвейерном режиме – $t_w = (n-1)t_0 + t_{конв}$, где $t_{конв}$ – дополнительные временные затраты на записи в буферные регистры. Похожий принцип применяется в синхронных конвейерах и систолических процессорах.

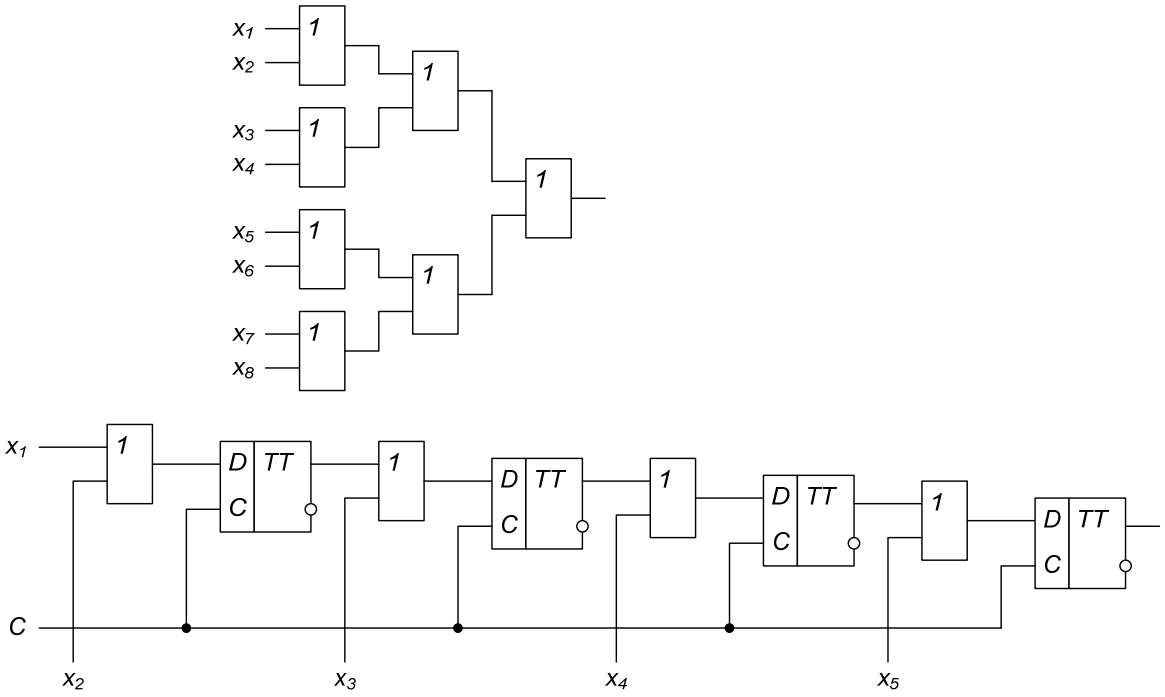

Рис. 6. Пирамидальная и конвейерная организация многовходовых элементов

Многопортовое ОЗУ

Цель создания – возможность организации параллельного чтения строк ОЗУ.

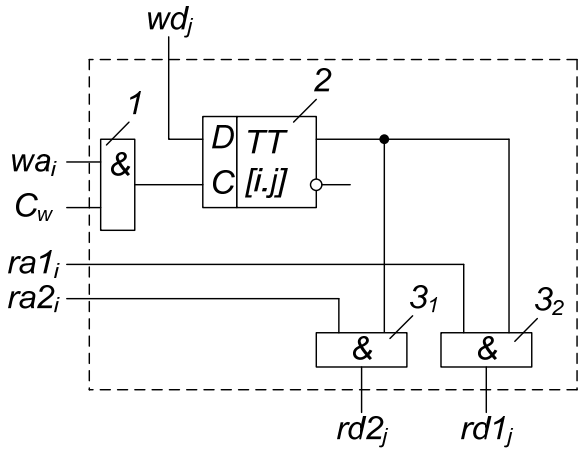

Рис. 7. Ячейка многопортового ОЗУ (показано 2 порта, может быть произвольное количество)

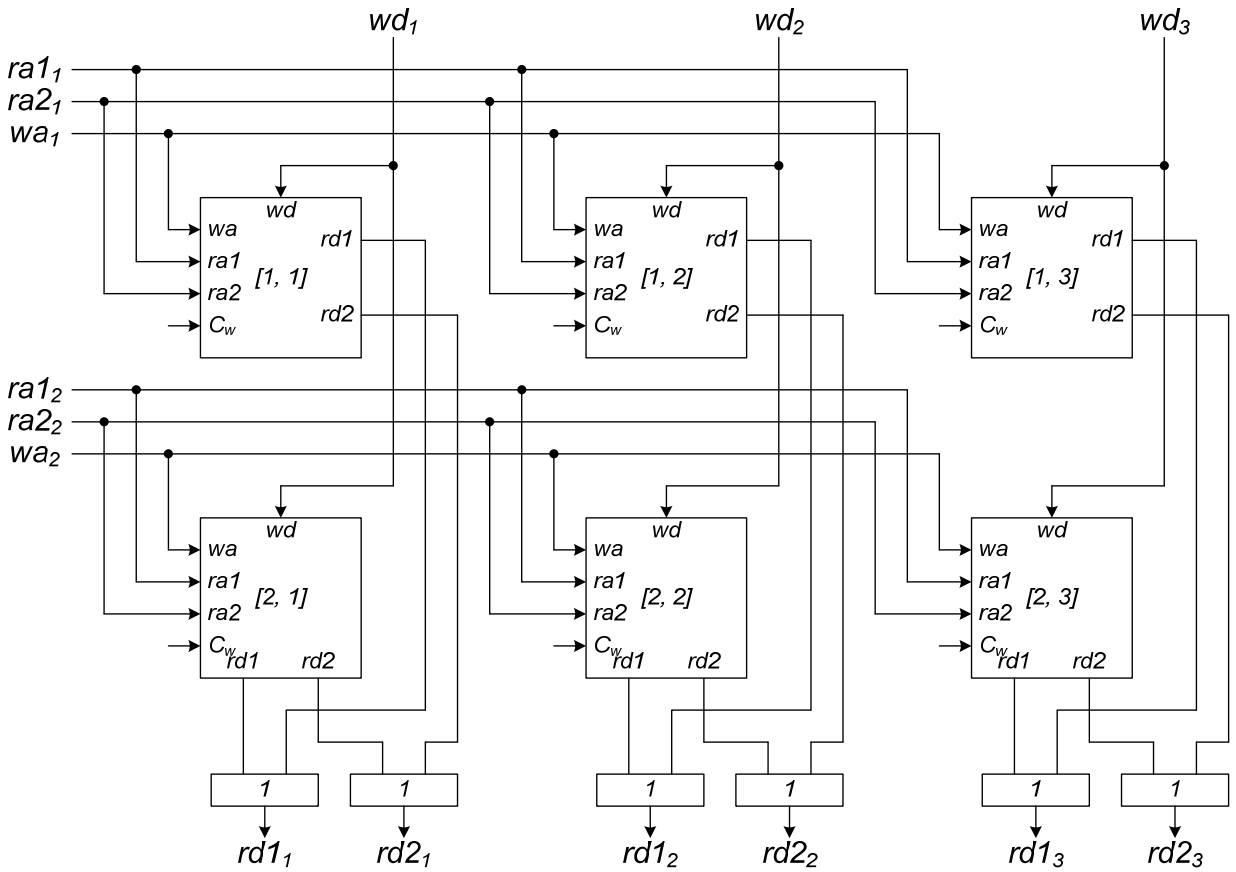

Рис. 8. Многопортовое ОЗУ 2 слова × 3 бита, 2 порта чтения

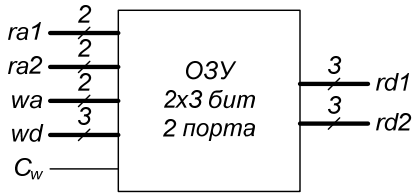

Рис. 9. ОЗУ 2×3, 2 порта

Каждый порт чтения по быстродействию совпадает в предыдущим рассмотренным ОЗУ.

Аналогично можно организовать многопортовое ОЗУ с параллельными портами записи путем дублирования входов *wa*, *wd* и C_w.

Совмещение записи и чтения в ОЗУ применялось в VRAM (Video RAM) – вывод картинки на монитор параллельно с ее обновлением.

ОЗУ с возможность ассоциативного поиска

Ассоциативный поиск производится по содержимому, в отличие от классического чтения по указанному адресу.
Content-addressable memory (CAM)

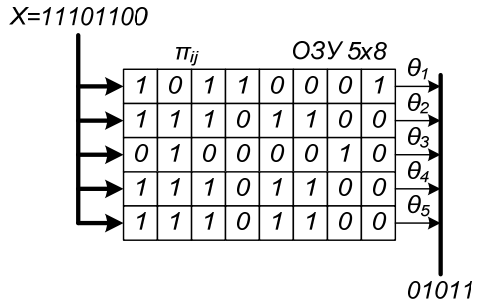

Рис. 10. Пример ассоциативного поиска в ОЗУ

Необходимо:

- сравнение информации на входе с информацией в ячейках строки, выработка двоичных признаков совпадения для ячеек $\pi_{ij} = d_{ij} \sim x_j = d_{ij}x_j \vee \overline{d}_{ij}\overline{x}_j$;
- формирования признаков совпадения для строк $\theta_i = \underset{j=\overline{1,m}}{\&} \pi_{ij}$.

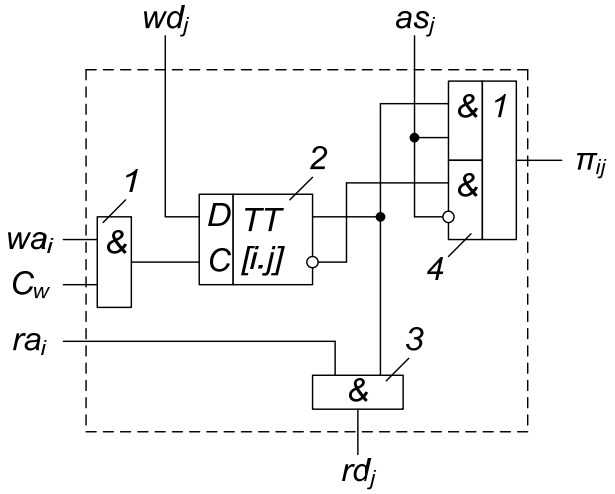

Рис. 11. Структура ячейки ОЗУ с возможностью ассоциативного поиска

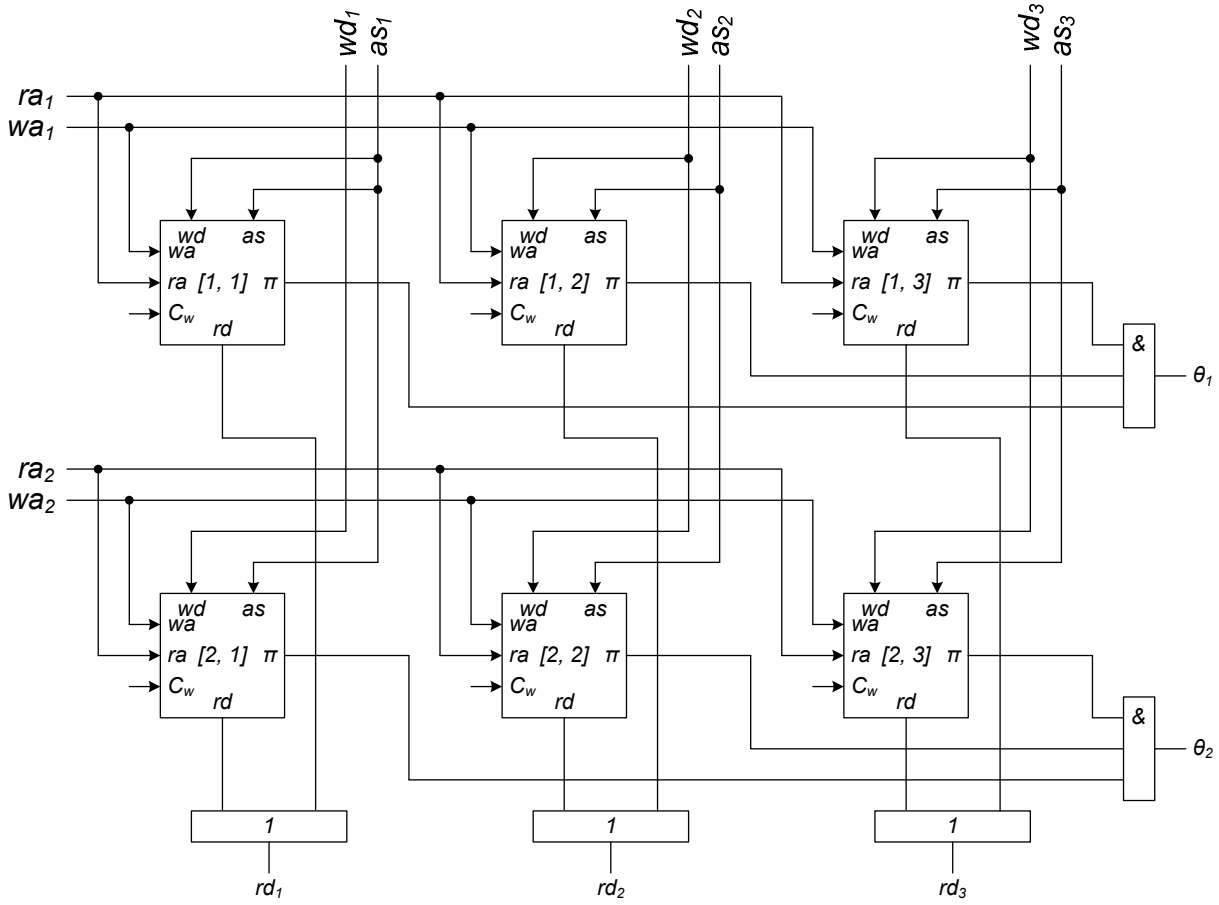

Рис. 12. ОЗУ с возможностью ассоциативного поиска

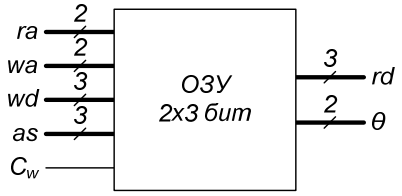

Рис. 13. Схема ОЗУ с возможностью ассоциативного поиска

При ассоциативном поиске на вход *as* подается искомая информация. Каждая ячейка вырабатывает значение двоичного признака π_{ij} совпадения. Признаки совпадения объединяются по строкам.

$t_{as} = \underbrace{2t_0}_{K4} + \underbrace{\lceil \log_2 m \rceil t_0}_{И} = \left(2 + \lceil \log_2 m \rceil\right) t_0$ – время ассоциативного поиска пропорционально размеру слов.

Обычное пословное чтение (еще необходимы сравнения!) – $n \cdot \left(\underbrace{1 + \lceil \log_2 n \rceil}_{\text{чтение}} + \underbrace{2}_{\text{сравнение}} \right) t_0 = n\left(3 + \lceil \log_2 n \rceil\right) t_0$

Выигрыш: $\eta = \dfrac{n\left(3 + \lceil \log_2 n \rceil\right) t_0}{\left(2 + \lceil \log_2 m \rceil\right) t_0} = \dfrac{n\left(3 + \lceil \log_2 n \rceil\right)}{2 + \lceil \log_2 m \rceil}$.

Для ОЗУ 1К×8 $\eta = 2250$ раз.

Возможна организация нескольких портов для ассоциативного поиска (по аналогии с многопортовым чтением).

Применение – проверка наличия адреса памяти в строке в кэш-памяти (поиск по старшим битам адреса), TLB-буфере.

Можно делать и другие виды поиска: по наибольшему совпадению, все элементы, большие/меньшие заданного и т.п.

ОЗУ с возможностью ассоциативной замены

Может быть использована для быстрой замены «старой» информации (*as*) на «новую» (*wd*).

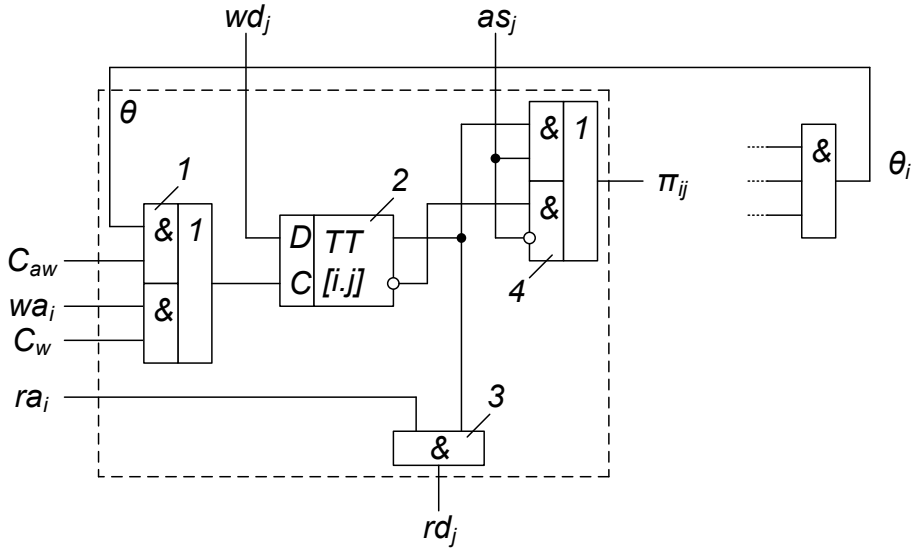

Рис. 14. Ячейка ОЗУ с ассоциативной записью

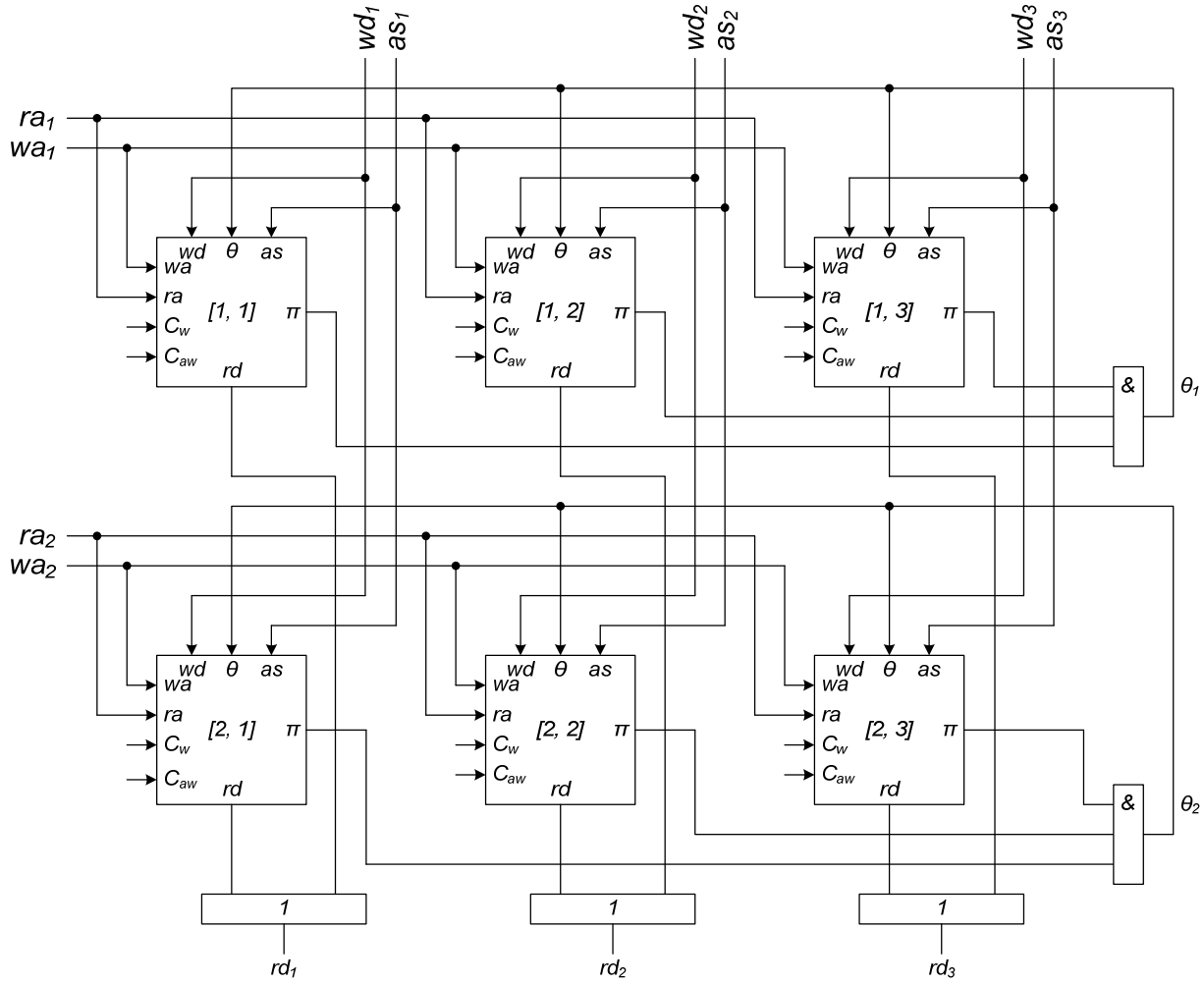

Рис. 15. ОЗУ с ассоциативной записью

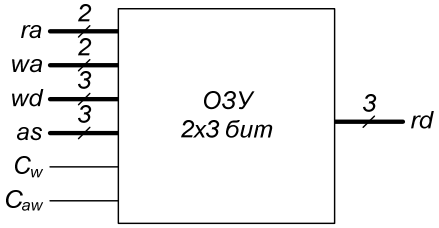

Рис. 16. Схема ОЗУ с возможностью ассоциативной записи

Оценка выигрыша в быстродействии:

$$t_{aw} = \left(\underbrace{2}_{K4} + \underbrace{\lceil \log_2 m \rceil}_{ИЛИ} + \underbrace{2}_{K1} + \underbrace{2}_{TT} \right) t_0 = \left(6 + \lceil \log_2 m \rceil\right) t_0 .$$

Реализация с использованием обычной памяти:

$$t = n\left(\underbrace{1 + \lceil \log_2 n \rceil}_{\text{чтение}} + \underbrace{2}_{\text{сравнение}} + \underbrace{1+2}_{\text{запись}} \right) t_0 = \left(6 + \lceil \log_2 n \rceil\right) n t_0 .$$

Выигрыш: $\eta = \dfrac{\left(6 + \lceil \log_2 n \rceil\right) n t_0}{\left(6 + \lceil \log_2 m \rceil\right) t_0} = \dfrac{\left(6 + \lceil \log_2 n \rceil\right) n}{6 + \lceil \log_2 m \rceil}$.

Для ОЗУ 1К×8 $\eta = 1820$ раз.

Ассоциативная память дороже обычной RAM, обладает большей площадью ячейки (из-за необходимости реализации электронной обвязки в виде коммутаторов и схем сравнения), большей выделяемой мощностью, меньшим объемом (при прочих равных). Находит ограниченные спецприменения.

Библиографический список

1. Орлов С.П., Ефимушкина Н.В. Организация компьютерных систем: Учебное пособие. - Самара: Самар.гос. техн. ун-т, 2011. - 203 с.