Глава 4

СТРУКТУРЫ ПРОЦЕССОРОВ НА ОСНОВЕ СКАЛЯРНОГО ПАРАЛЛЕЛИЗМА И ДРУГИЕ ТИПЫ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОРОВ

§ 4.1. Скалярный параллелизм

Термин *скалярный параллелизм* соответствует параллелизму операций внутри тела цикла или отдельного выражения. Вместо термина “скалярный” часто используется термин “локальный параллелизм”, чтобы отличить его от “глобального параллелизма”, соответствующего параллелизму итераций цикла или независимых ветвей программы.

Скалярный параллелизм определяется в границах *базового блока* (ББ), под которым понимают отрезок программы, не содержащий условных или безусловных переходов. ББ обычно содержит одно или несколько выражений ЯВУ.

Из закона Амдала:

$$r = \frac{1}{a + \frac{1-a}{n}}$$

следует, что значение ускорения r весьма чувствительно к удельной величине скалярной части вычислений a, поэтому в быстродействующих ЭВМ прикладывают значительные усилия для распараллеливания скалярных участков.

Результаты анализа программ показывают, что в одном такте можно выполнить 1,5...2 операции или команды. Комплексную команду, содержащую больше одной параллельной команды, называют длинной командой (ДК). Для увеличения скалярного параллелизма используется ряд методов, рассмотренных в § 5.5, например, развертка циклов. Если при этом число параллельных операций превышает 2, то такой параллелизм называется *суперскалярным*, а команда — *сверхдлинной* (СДК).

Величина скалярного параллелизма зависит от уровня представления программы. Параллелизм на уровне ЯВУ будем назы

вать *математическим*, на уровне системы команд — *аппаратным*. Существует несколько уровней систем команд, и каждому уровню соответствует своя величина скалярного параллелизма. Рассмотрим основные уровни систем команд.

1. Исторически в больших ЭВМ и ПЭВМ наибольшее распространение получила *система команд типа CISC* (Complex Instruction Set Computer), содержавшая большое количество команд. Основой этой системы команд являлся формат

КОП R1, D2 (B2) (4.1)

где КОП представлял арифметико-логическую операцию в АЛУ, а второй адрес содержал обращение в память с базой или индексом *B*2 и смещением *D*2. Такая команда требовала многотактного выполнения.

2. Сокращенная *система команд типа RISC* (Redused Instruction Set Computer) более перспективна для параллельных процессоров, поскольку число команд в ней меньше и вследствие их простоты большинство команд можно реализовать аппаратно и выполнить за один такт.

Сокращение числа команд в *RISC*-системе достигается за счет разбиения формата (4.1) на два более простых формата:

Чт, Зп, Ri, D2 [B2] (4.2)
КОП Ri, Rj

Первый формат предназначен только для работы с памятью, второй — только для работы с АЛУ.

Если предположить, что имеется k кодов операций для работы с памятью и l кодов — для работы с АЛУ, то система команд типа *CISC* должна содержать $k*l$ команд формата (4.1), а система *RISC* — только $k+l$ команд. Следовательно, отказ от формата (4.1) значительно сокращает список системы команд.

3. Последующее упрощение системы команд достигается при переходе к *типу MISC* (Minimum Instruction Set Computer), в котором команда формата (4.2) разбивается на две команды:

Rk = B2 + D2 (4.3)
Чт, Зп, Ri, [Rk] (4.4)

где команда (4.3) выполняет индексную операцию, а команда (4.4) — упрощенное обращение в память.

При понижении уровня системы команд увеличивается число служебных операций (команд), которые можно выполнять параллельно, то есть растет скалярный параллелизм.

Скалярный параллелизм длительное время использовался для ускорения процессоров в ЭВМ типа ОКОД, затем в конвейерных ЭВМ типа CRAY. Скалярный параллелизм имеет особое значение для микропроцессорной техники по следующим причинам:

1. Скалярный параллелизм есть свойство последовательных программ и может быть выявлен автоматически сравнительно несложными средствами. Таким образом, ЭВМ со скалярным параллелизмом не требует изменения традиционной системы последовательного программирования. Это обеспечивает массовое применение таких ЭВМ с точки зрения программирования.

2. Успехи микроэлектроники позволяют разместить на одном кристалле процессор ЭВМ с несколькими АЛУ, что предполагает массовость производства с точки зрения аппаратуры.

Таким образом, суперскалярные процессоры, характеризующиеся последовательным программированием и параллельным функционированием, являются массовым видом вычислительной техники. По классификации Флинна они относятся к классу ОКОД.

§ 4.2. Структура и функционирование конвейера

Параллелизм ассемблерного уровня реализуется двумя способами:

1. В виде конвейера (конвейера команд, арифметического конвейера или конвейера смешанного типа). Конвейер команд использует технический параллелизм, возникающий вследствие необходимости выполнять команду с помощью блоков различного функционального назначения (управление, память, АЛУ и др.).

Одновременная работа этих блоков позволяет параллельно обрабатывать сразу несколько смежных команд.

Арифметический конвейер использует разбиение арифме-

тической операции на логически законченные микрооперации. Например, команда сложения с плавающей запятой может быть разделена на следующие микрооперации: вычитание порядков, выравнивание мантисс, сложение, нормализация. Арифметический конвейер также позволяет параллельно обрабатывать несколько команд.

2. В виде многопроцессорной системы, где совокупность процессоров обрабатывает в каждом такте один ярус ЯПФ некоторого выражения. Обычно в каждом процессоре используется и конвейерная обработка. Поэтому в суперскалярном процессоре одновременно обрабатывается m * n команд, где m — число этапов в конвейере, а n — число конвейеров; в каждом такте вырабатывается *n* результатов. Если в идеальном случае для конвейера принять:

$$\Delta t = t / m ,$$

где Δt — время одного такта, а t — время выполнения одной команды, то быстродействие конвейера будет:

$$V = m / \Delta t = \frac{m \cdot n}{t} . \qquad (4.5)$$

Таким образом, увеличение быстродействия суперскалярного процессора достигается как за счет увеличения числа конвейеров, так и за счет удлинения конвейера. При этом по возможности сначала нужно увеличивать число ступеней конвейера, так как для этого требуется меньше оборудования, чем для увеличения числа процессоров при достижении того же прироста быстродействия.

Следовательно, конвейер является основой суперскалярного процессора. В настоящем параграфе будут рассмотрены структура, функционирование, недостатки и способы получения максимального быстродействия одиночных конвейеров [19, 20].

Конвейеры можно разделить на две большие группы:

1. Векторные конвейеры (см. § 2.2), которые выполняют одну операцию над группами данных, называемых векторами. Такие конвейеры, как правило, являются арифметическими, то есть их ступени выполняют части арифметико-логических операций.

2. Скалярные конвейеры, в которых на разных ступенях обработки одновременно находятся команды с разными кодами операций. Скалярные конвейеры могут содержать только конвейер команд, но в процессорах с плавающей запятой часто скалярный

конвейер включает и арифметические ступени. Для выполнения векторных операций в скалярном конвейере необходимо на вход последнего в каждом такте подавать один и тот же код операции.

С точки зрения скалярного параллелизма интерес представляют преимущественно скалярные конвейеры. В качестве примера такого конвейера рассмотрим наиболее распространенный конвейер микропроцессора типа *CISC* (за прототип взят конвейер МП "*Пентиум*" из семейства МП 80 × 86).

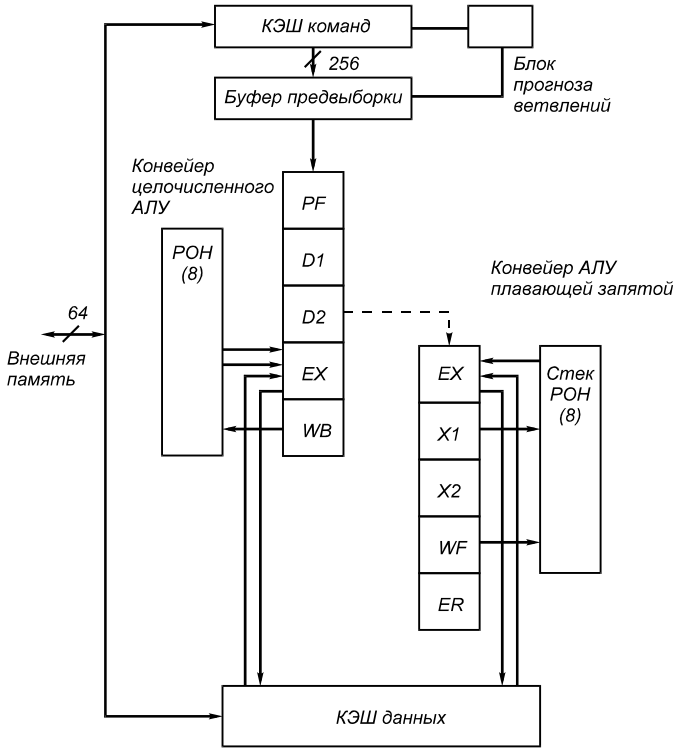

Рис. 4.1. Целочисленный и плавающий конвейер CISC-процессора

Целочисленный конвейер содержит следующие *ступени* (рис. 4.1):

- ступень предвыборки PF (Prefetch), которая осуществляет предварительную упреждающую выборку группы команд в соответствующий буфер;

- ступень декодирования полей команды D1 (Decoder 1);
- ступень декодирования D2 (Decoder 2), на которой производится вычисление абсолютного адреса операнда, если операнд расположен в памяти;
- на ступени исполнения EX (Execution) производится выборка операндов из РОН или памяти и выполнение операции в АЛУ;
- наконец, на ступени записи результата *WR* (Write Back) производится передача полученного результата в блок РОН.

Плавающий конвейер имеет общие с целочисленным конвейером ступени *PF*, *D1*, *D*2, но кроме того, имеет еще 5 дополнительных ступеней (переход показан пунктирной стрелкой):

- на ступени EX производится чтение из памяти или регистров или запись в память;
- на ступени X1 выполняется часть плавающей операции (фаза 1) или запись в плавающий РОН. Возможен обход (bypass) на ступени EX;
- на ступени X2 выполняется продолжение обработки (фаза 2);
- на ступени WF производится округление и обратная запись результата в блок РОН;
- на ступени ER (Error Reporting) выводится сообщение о наличии ошибок.

Наибольшая производительность конвейера достигается, когда каждый такт на вход поступает новая команда, а на выходе получается очередной результат.

Однако получению максимального быстродействия в *CISC*-конвейере препятствуют следующие обстоятельства:

1. Простои конвейера из-за наличия команд, которые требуют многотактного исполнения в АЛУ или на других ступенях конвейера.

2. Простои конвейера из-за зависимости по данным между соседними командами.

3. Простои конвейера из-за очистки и повторной загрузки конвейера в случае ошибки при предварительном выборе направления условного перехода.

4. Ограничения пропускной способности аппаратных средств: РОН, памяти различных видов, шин связи.

Рассмотрим влияние этих факторов на уменьшение быстродействия конвейера и некоторые способы нейтрализации этого влияния.

Многотактные команды. В таблице 4.1 представлено время нахождения некоторых типичных команд в целочисленном конвейере на этапе *EX*, поскольку на остальных этапах на выполнение команды затрачивается только один такт. (Здесь и далее рассматриваются команды семейства 80×86).

Таблица 4.1.

Длительность некоторых команд в целочисленном конвейере

N пп	Формат команды	Число тактов
1	LD R1, D[Ri]	1
2	ST D[Ri], R1	1
3	ADD R1, R2	1
4	ADD R1, [Ri]	2
5	ADD [Ri], R1	3
6	IMUL R1, R2	11

Команда 4 задерживается на этапе *EX* два такта, поскольку один такт тратится на обращение к памяти за операндом, а другой — на выполнение арифметической операции. Команда 5 занимает три такта из-за двукратного обращения к памяти. Команда 6 умножения занимает много тактов из-за большого числа выполняемых ею микроопераций.

Если конвейер выполняет только однотактные команды, то он продвигается каждый такт и достигается максимальная (пиковая) производительность, при этом темп выдачи результатов — один результат за такт. Если в конвейер попадают многотактные команды, то конвейер приостанавливается на время их выполнения, а производительность падает.

В таблице 4.2 приведены времена выполнения некоторых важных команд для конвейера плавающей запятой; в скобках ука

зано время полного выполнения команды в АЛУ, а перед скобкой — темп получения результатов в потоке команд данного типа.

Таблица 4.2. Длительность некоторых команд в плавающем конвейере

N пп	Формат матрицы	Ступени					Итого
		EX	X1	X2	WF	ER	
1	FLD[M]	1	1	-	-	-	1
2	FST[M]	2	-	-	-	-	2(2)
3	FADD[M]	1	1	1	-	-	1(3)
4	FMUL[M]	1	1	1	-	-	1(3)
5	FDIV[M]						39
6	FSQRT						70

В плавающем конвейере *CISC*-процессора файлы РОН реализованы обычно в виде стека и операции выполнения на верхушке стека, поэтому в командах не указываются номера РОН, а только адрес ячейки памяти.

Из таблицы 4.2 следует, что большинство команд находятся в АЛУ (ступени *EX*, *X*1, *X*2) несколько тактов (время в скобках), однако на каждом этапе задерживаются только 1 такт.

Следовательно, темп выполнения в конвейере — 1 такт на команду (команды *FLD*, *FADP*, *FMUL*). Некоторые команды могут на одном этапе находиться несколько тактов, приостанавливая на это время конвейер.

Поскольку целочисленное (ступень *EX*) и плавающее АЛУ работают независимо, то прием команд на вход общей части конвейера приостанавливается только в следующих случаях:

1. Оба конвейера выполняют многотактные операции.

2. Один из конвейеров выполняет многотактную операцию, но на ступенях *D*1 или *D*2 находятся команды для этого же конвейера.

Зависимость по данным. Рассмотрим пример выполнения в процессоре с плавающей запятой следующего отрезка программы:

```
1  FLD [M1]
2  FMUL [M2]
```

3 FADD [M3] (4.6)
4 FST [M4]

в котором каждая команда размещает свой результат на верхушке стека, а последующая команда использует его как один из операндов. Это означает, что между смежными командами существует строгая зависимость по данным.

На рис. 4.2 представлена временная диаграмма выполнения этого отрезка программы.

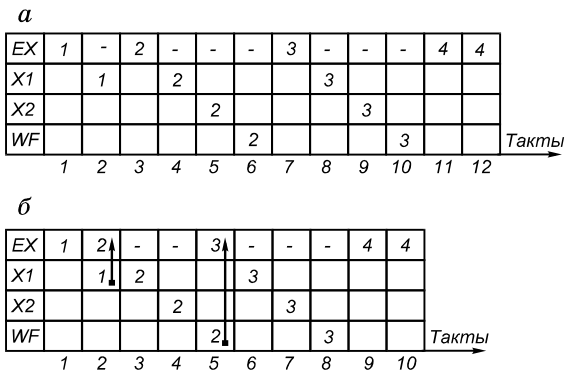

Рис. 4.2. Диаграмма работы конвейера

Из рис. 4.2, а следует, что ступень *EX* простаивает 7 тактов из 12:

1. В такте 1 ступень *EX* не работает, так как результат команды 1 является операндом команды 2, поэтому в такте 2 он записывается в стек и только в такте 3 может быть считан командой 2.

2. Аналогично команда 3 ожидает такты 4, 5, 6, пока результат команды 2 на ступени *WF* не окажется в стеке, и только в такте 7 может считать этот результат из стека.

3. Зависимость между командами 3 и 4 также дает такты простоя 8, 9 и 10 ступени *EX*.

Частично ситуация может быть исправлена благодаря механизму обходов, который использован в плавающем конвейере для уменьшения простоев. Благодаря обходу 1 результат операции *FLD*, который записывается в стек на ступени *X*1, может одновре

менно быть послан прямо на ступень *EX* для использования следующей инструкцией.

Обход 2 позволяет послать результат арифметической операции со ступени *WF* на приемную ступень *EX* одновременно с записью этого результата в стек.

На рис. 4.2, б представлена диаграмма той же программы, но в такте 2 результат команды 1 одновременно с записью в стек передается на ступень *EX,* благодаря чему может быть начата команда 2. То же происходит и в такте 5. Команда 4 не может быть начата в такте 8, так как команда *FST* начинает работать только после записи результата в РОН предыдущей командой.

Таким образом, обходы оказали небольшую помощь (исключены 2 такта из 12).

Принципиальное решение вопроса об устранении простоя конвейеров из-за зависимости по данным состоит в том, чтобы размещать между зависимыми командами другие команды программы, не зависящие по данным друг от друга, от предшествующих и последующих команд. В этом случае программа (4.6) выглядела бы с независимыми командами (НК) следующим образом (временная диаграмма на рис. 4.3):

```
1  FLD [M1]
2  FMUL [M2]
3  НК
4  НК
5  FADD [M3]                                   (4.7)
6  НК
7  НК
8  НК
9  FST [M4]
```

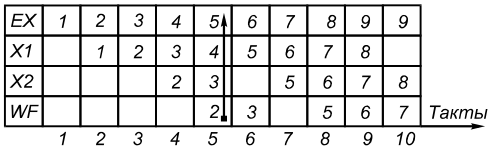

Рис. 4.3. Диаграмма выполнения программы с включением независимых команд

Программа (4.6) согласно рис. 4.2, б выполняется за 10 тактов, следовательно, архитектурная скорость будет равна:

$$z_1 = 4/10 = 0{,}4.$$

Программа (4.7) также выполнится за 10 тактов, но при этом согласно рис. 4.3

$$z_2 = 9/10 = 0{,}9.$$

Программа (4.7) реализует скалярную форму параллелизма в конвейерных процессорах. К сожалению, размещение независимых команд в программах с плавающей запятой в *CISC*-процессорах вызывает трудности из-за стековой организации файла РОН.

Условные переходы. Рассмотрим отрезок программы:

```
        ...
        1    ADD R1, R2
        2    ST [M], R1
        3    JZ LONG
        4    ADD R1, [M1]                    (4.8)
        5    MUL R3, [M2]
        ...
        ...
LONG    11   ADD R1, [M3]
        12   SUB R4, [M4]
        ...
```

Известно, что в конвейере адрес перехода определяется только на ступени *EX*. Но, чтобы не допустить простоев на этапах конвейера, в (4.8) необходимо непосредственно за командой перехода 3 начинать выборку одной из ветвей, начиная с команды 4 либо с команды 11.

Предположим, что принято решение выбирать ветвь с командой 4, и это решение оказалось неверным, тогда потребуется очистка конвейера и его повторное заполнение. На рис. 4.4, а показано, что в такте 5 определяется правильный адрес перехода на ступени *EX* и с такта 6 начинается перезагрузка конвейера. При этом потерянными оказались три такта.

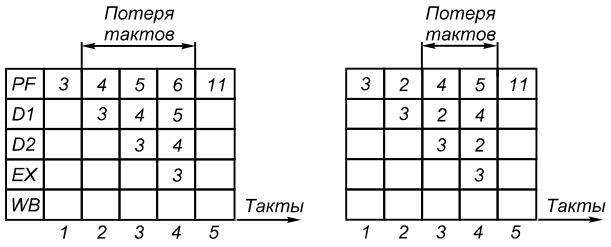

Рис. 4.4. Перезагрузка конвейера при ошибке выбора направления перехода

Потеря тактов будет больше, если команды 11 не окажется в буфере предварительной выборки.

Если учесть, что многие программы содержат от 10 до 20% команд условных и до 10% безусловных переходов, то правильная обработка ветвлений в конвейере оказывается весьма важной.

Методы управления переходами можно разделить на две группы: статические, выполняемые на этапе компиляции, и динамические, выполняемые аппаратурой конвейера в процессе выполнения программы.

К статическим методам относятся:

1. Исключение ветвлений в программе, если это возможно.

Для этого применяют: вычисление ветвлений на этапе компиляции; вынос ветвлений за пределы цикла; развертку итераций.

В приведенном ниже примере развертка позволяет в два раза сократить число ветвлений:

```
      DO1 I = 1, 100                DO 1 I = 1, 100, 2
1     A (I) = A (I) ∗ 1.1           A (I) = A (I) ∗ 1.1
                                1   A (I+1) = A (I+1) ∗ 1.1
```

2. В программе переход назад обычно соответствует циклу и выполняется в 90% случаев, а переход вперед обозначает ветвление и выполняется в 50% случаев. На этапе компиляции определить это несложно, поэтому в машинной программе можно предполагаемое направление перехода пометить специальным битом перехода в команде.

3. Еще один метод называется отложенным ветвлением. В примере (4.8) команда 2 не влияет на условный переход, поэтому ее можно переставить после команды 3 и всегда выполнять после команды перехода независимо от направления перехода.

Программа будет выглядеть следующим образом:

```
        ...
        1     ADD R1, R2
        3     JZ LONG
        2     ST [M], R1
        4     ADD R1, [M1]
        5     MUL R3, [M2]
        ...
        ...
LONG    11    ADD R1, [M3]
        12    SUB R4, [M4]
        ...
```

На рис. 4.4, б показано, что в этом случае потеря тактов уменьшается. Таких отложенных команд может быть и больше одной, но для реализации отложенного ветвления требуется наличие в программе скалярного параллелизма.

В динамике управление переходами осуществляется следующими способами:

1. Самый простой способ состоит в том, чтобы остановить прием команд на вход конвейера до тех пор, пока команда перехода не достигнет ступени *EX* и не будет вычислено реальное направление перехода. Однако этот метод, упрощая управление, не решает проблемы простоев.

2. Наиболее сложный способ заключается в том, чтобы при появлении на входе команды перехода выбирать и условно выполнять обе ветви возможных переходов. Условное выполнение обозначает, что до момента определения адреса перехода запрещается запись в регистры или память.

3. Наконец, третьим способом динамической обработки ветвлений является предсказание. Самым простым способом предсказания является выбор для выполнения той ветви, которая следует непосредственно за командой перехода (короткий переход). Но в 50% случаев этот выбор будет ошибочным.

Более эффективной является система предсказаний на основе истории процесса вычислений.

На рис. 4.1 присутствует устройство прогнозирования ветвлений. Оно содержит таблицу достаточно большого размера, например, на 256 строк. В каждой строке таблицы записан для выполнения части программы:

1. Адрес команды перехода.

2. Адрес дальнего перехода.

3. Бит “истории”, который указывает, по какому направлению произошел переход при последнем использовании команды перехода.

Буфер упреждающей выборки обычно содержит от нескольких до нескольких десятков предварительно выбранных команд, чтобы сгладить задержки, связанные с обращениями в память. Одновременно с подачей команд на вход конвейера устройство управления производит в буфере предварительной выборки просмотр вперед выбранных команд. Если при просмотре обнаружена команда перехода, то по таблице “истории” определяется направление перехода.

Устройство управления функционирует на основе предположения, что при повторном выполнении одной и той же команды перехода переход будет осуществлен по одному и тому же адресу. В соответствии с этим в буфер упреждающей выборки выбирается ветвь, предписанная битом “истории”, если этой ветви в буфере упреждающей выборки еще нет.

Буфер упреждающей выборки содержит две зоны: для текущей и альтернативной ветви и переключение с зоны на зону не вызывает простоев.

Описанный механизм ветвления позволяет выбирать правильные пути ветвления с вероятностью более 80%.

КЭШ. Фактором, также ограничивающим быстродействие конвейера, является недостаточное быстродействие ресурсов, в частности, памяти. Рассмотрим некоторые требования к памяти.

Чем меньше такт конвейера, тем более быстрой должна быть память, обслуживающая этот конвейер. Если учесть, что цикл внешней памяти обычно больше, чем длительность такта конвейера, то становится очевидным, что между конвейером и внешней памятью должны быть расположены вспомогательные средства

памяти различного быстродействия и назначения и усовершенствованная система шин.

Регистры общего назначения являются наиболее быстрым видом памяти, их число колеблется от 8 до 32. РОН используется для хранения информации, как правило, локальной для тела цикла или базового блока. Чтобы избежать конфликтов при обращении к РОН, их делают многопортовыми (многовходовыми). Число портов обычно равняется трем: два для выдачи операндов и один — для приема результата.

Следующий шаг для уменьшения задержки при обращении к памяти состоит в разделении памяти на два независимых блока: память для программ и память данных. Такое разделение носит название гарвардской архитектуры. Для нее характерно использование буфера упреждающей выборки (рис. 4.1), в который за один такт принимается сразу несколько команд из памяти с большой разрядностью ячейки. Этот же буфер позволяет выбирать заранее альтернативные ветви в случае команд условных переходов. Все это сглаживает скорость поставки команд в конвейер.

Наиболее эффективным средством ускорения обращения к памяти является локальная (для АЛУ) память небольшого размера, называемая КЭШ.

Известно, что 90% обращений в память производится в ограниченную область адресов. Эта область называется рабочим множеством, которое медленно перемещается в памяти по мере выполнения программы. Для рабочего множества можно сделать промежуточную память небольшого размера, а значит, в несколько раз более быструю, чем основная память.

Для однокристальных микропроцессоров особенно важно то, что память такого размера можно разместить внутри кристалла, благодаря чему исчезают потери времени на вывод данных из кристалла, вызываемые малым числом выводов, задержками сигнала на контактах и из-за больших расстояний на плате.

Введем понятие строки и отображения [19]. Строка есть базовая единица информации, которая перемещается между основной и КЭШ-памятью. Следовательно, основная память состоит из большого числа строк с последовательными адресами, а в КЭШ-памяти понятие “последовательные адреса” отсутствует. В КЭШ-памяти каждое слово сопровождается адресным тэгом, указываю

щим, какую строку основной памяти представляет данная строка КЭШ-памяти.

Отображением называется способ размещения и выборки строк основной памяти из КЭШ-памяти.

Можно выделить четыре основных типа отображения:

1. Полностью ассоциативная КЭШ-память. Любая строка основной памяти может находиться в любой строке КЭШ-памяти. Каждая строка КЭШ-памяти содержит свой компаратор (устройство сравнения). Если процессору нужна строка с определенным адресом, он должен искать ее путем сравнения адреса с тэгом всех строк КЭШ-памяти. Если такое сравнение делать последовательно, то КЭШ-память теряет смысл из-за низкого быстродействия; если сравнение делать одновременно со всеми тэгами, то такая память из-за ее сложности не может иметь большой объем.

КЭШ-память ассоциативного типа обычно имеет небольшой объем и используется для вспомогательных целей.

2. Противоположной структурой является память с прямым отображением. Одна из возможных реализаций использует разрядное отображение. Например, если в адресе строки памяти содержится N разрядов, то младшие n из них выбирают, в какую строку КЭШ-памяти она может копироваться. Следовательно, все строки с одинаковыми младшими адресами попадают в одну и ту же строку КЭШ-памяти. Оставшиеся N-n разрядов используются как адресный тэг для сравнения с адресом, выставленным процессором, чтобы убедиться, имеется ли данная строка в КЭШ или отсутствует.

Основное достоинство состоит в том, что накопитель такой памяти имеет структуру обычной прямоадресуемой памяти и нужен всего один компаратор, следовательно, такая КЭШ-память может иметь большую емкость.

Основной недостаток состоит в том, что в КЭШ-памяти может находиться ограниченное число комбинаций строк. Например, в КЭШ-памяти не могут одновременно находиться строки с одинаковыми младшими адресами. Это заметно увеличивает число промахов.

3. Промежуточным между полностью ассоциативным и прямым отображением является секторное отображение, вариант которого описан в § 2.1. В этом случае в КЭШ-памяти располагаются

страницы основной памяти. При обращении в КЭШ-память базовые адреса страниц в КЭШ-памяти устанавливаются с помощью служебной полностью ассоциативной памяти небольшого размера, а поиск слов в странице КЭШ-памяти производится на основе прямого адресного доступа. Секторная память удобна для больших ЭВМ, где программа может целиком располагаться в одной или нескольких страницах КЭШ-памяти. Для суперскалярных процессоров, реализуемых на одном кристалле, такой подход трудно реализовать, поскольку внутрикристальная КЭШ-память реально имеет объем менее одной страницы.

4. Другим промежуточным отображением, которое в основном и используется в однокристальных суперскалярных микропроцессорах является КЭШ-память с *множественно-ассоциативным доступом*. В ней, как и в памяти с прямым отображением, используется выбор строки КЭШ-памяти по младшим разрядам адреса основной памяти, но в этой строке КЭШ-памяти содержится несколько слов основной памяти, выбор между которыми производится на основе ассоциативного поиска.

На рис. 4.5 представлен КЭШ с двукратной ассоциацией. Это означает, что по одному смещению выбирается строка, содержащая информацию по двум разным страницам памяти. Это увеличивает вероятность удачного обращения. Обычно кратность ассоциации колеблется от единицы до четырех. Обычная длина поля тэгов до 24 или 32 разрядов, длина строки данных (одной) — от 4 до 32 байтов; число строк в КЭШе — до 256.

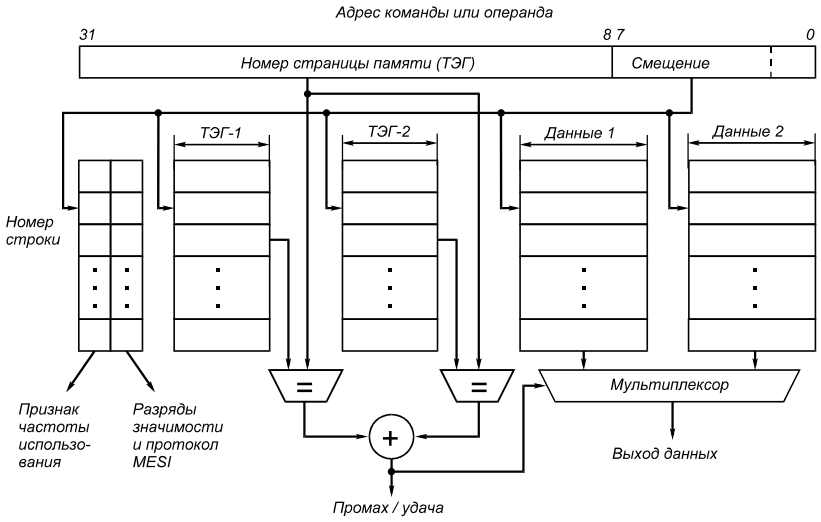

Рис. 4.5. Структура множественного ассоциативного КЭШа

Важной величиной является число портов в КЭШе. Как правило, число адресных портов (входы тэга) составляет 2...3, а порт данных — единственный. Объем внутрикристального КЭШа в МП колеблется в пределах от 4 до 32 килобайт, а внешнего — составляет несколько сотен килобайт.

На рис. 4.6 представлена зависимость числа попаданий от размера КЭШ-памяти с множественно-ассоциативным доступом. Из рисунка следует, что, если строка КЭШ-памяти содержит более четырех строк основной памяти, то это уже увеличивает степень попаданий незначительно.

Переход к RISC. Ранее было показано, что вследствие наличия многотактных команд, зависимости по данным, влияния условных переходов и аппаратных ограничений нельзя достичь предельной для конвейера архитектурной скорости 1 команды за такт. В рамках архитектуры *CISC*-процессора можно осла бить влияние указанных выше причин. В частности, для умень-

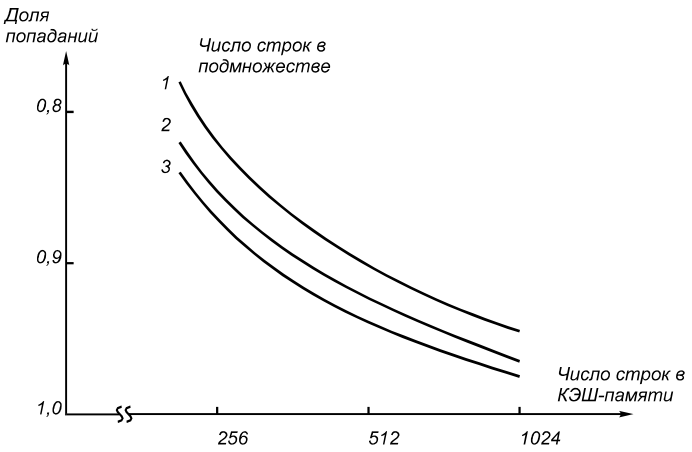

Рис. 4.6. Влияние характеристик КЭШ-памяти
на вероятность попадания

шения влияния зависимости по данным выше были описаны аппаратные обходы и введение независимых команд; для ослабления влияния условных переходов предложены различные методы предсказания; для повышения пропускной способности памяти предложена иерархия запоминающих устройств и шинные системы с высокой пропускной способностью.

Однако проблема многотактных команд может быть решена только при переходе от *CISC* к *RISC* и *MISC* системам команд.

Этот переход позволяет:

1. Сократить систему команд за счет устранения сложных команд.

2. Увеличить количество РОН; использовать для операций АЛУ трехадресные команды, ввести файлы РОН с произвольным доступом вместо стека для плавающей запятой.

3. Использовать команды фиксированного формата.

Рассмотрим эти средства несколько подробнее.

Переход к системе команд типа *RISC* приводит к разрешению системы команд на две группы: одна — для работы с АЛУ, другая — для работы с памятью. Такое разделение приводит к *Load/Store архитектуре*, представленной на рис. 4.7.

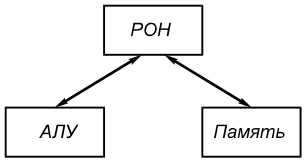

Рис. 4.7. Load-Store архитектура RISC-процессора

В такой структуре АЛУ и память работают параллельно. Чтобы обеспечить максимальное быстродействие системы, необходимо обеспечить упреждающую работу памяти.

Число РОН в *RISC*-процессоре достигает величины 32, 64 и даже более, в то время как в *CISC* оно не превосходит 8 или 16. Увеличение числа РОН уменьшает число обращений к памяти, сокращает длину программы, позволяет устранить некоторые зависимости по данным.

Трехместный (неразрушающий) формат команды выбран по следующим причинам: уменьшается число обращений к памяти (при достаточном числе РОН), устраняются излишние зависимости по данным. Например, в программе:

R3 = R1 + R2
R1 = R4 + R5

существует зависимость обратного типа, а в программе

R3 = R1 + R2
R6 = R4 + R5

использующей большее число регистров, она отсутствует, поэтому команды могут выполняться параллельно.

В *RISC*-процессорах РОН плавающей запятой организованы в виде файла с произвольным доступом, а не в виде стека, как в *CISC*. Это устраняет зависимости между смежными командами, вызванные единственным ресурсом — верхушкой стека, и уменьшает число обращений к памяти.

Препятствием к однотактной реализации команд *CISC* является последовательная структура дешифрации полей микрокоманды и переменная длина команды. Например, в системе команд МП *i*80×86 характер дешифрации полей команды последовательный:

по префиксу или первому байту определяется длина команды, после выборки оставшихся байтов производится дешифрация их полей. В отличие от этого в *RISC* команды имеют фиксированный формат, все поля независимы, поэтому дешифруются одновременно.

Все эти мероприятия позволяют выполнять большинство команд за 1 такт, при этом микропрограммирование перестает быть необходимым.

Таким образом, конвейер на базе системы команд типа *RISC* позволяет получить предельную для одиночного конвейера архитектурную скорость — 1 результат за такт.

Переход к новой системе команд типа *RISC* требует перепрограммирования громадного объема прикладного программного обеспечения, накопленного для широко распространенных систем команд типа *CISC*, используемых в МП *i*80×86, МС 680×80, поэтому естественным является стремление получить более высокую архитектурную скорость для *CISC*-МП на базе приближения свойства *CISC*-процессоров к свойствам *RISC*-процессоров. Для этой цели предпринимаются следующие действия программного и аппаратного характера:

1. Новые компиляторы *CISC*-процессоров строятся таким образом, чтобы для генерации машинных кодов из большого списка команд *CISC*-процессора использовать только небольшой набор простых команд, называемый *RISC*-ядро.

2. Ограничение на количество РОН в *CISC*-процессорах (всего 8 в *i*80×86) преодолевается следующим образом. При проектировании нового кристалла для *CISC*-процессора предусматривается увеличенный файл РОН, например, 32, и эти РОН составляют виртуальный файл. В процессе выполнения команд каждому логическому РОН, описанному в команде, предоставляется свободный РОН из виртуального файла, благодаря чему, как было показано ранее, растет параллелизм исполнения.

3. Для приближения стека по способу функционирования к файлу РОН с произвольным доступом аппаратно реализуются быстрые команды адресного обмена между регистрами стека типа *FXCH*, благодаря которым верхушка стека в каждом такте предстает как новый РОН с требуемым номером.

§ 4.3. Структура суперскалярного процессора

В настоящем параграфе для иллюстрации особенностей построения суперскалярных систем в качестве базовой используется архитектура суперскалярного процессора Пентиум фирмы Intel с системой команд типа *CISC* [20]. Вызвано это следующими причинами:

- *суперскалярная организация процессора* Пентиум достаточно проста для изучения, однако в ней отражены все особенности построения суперскалярных систем;
- система команд процессора Пентиум практически совпадает с системой команд широко известного семейства микропроцессоров *i*80×86, поэтому примеры программ легко воспринимаются;
- наконец, в процессоре Пентиум на программном и аппаратном уровне принято ряд решений, приближающих его архитектуру к архитектуре *RISC*-процессоров, позволяющих, однако, сохранить программную совместимость с большим объемом ранее написанного прикладного матобеспечения.

Появление в структуре процессора более одного конвейера делает этот процессор суперскалярным.

Как правило, в суперскалярных процессорах в первую очередь увеличивают количество целочисленных конвейеров, так как статистика показывает, что в обычных пакетах прикладных программ для ПЭВМ около 80% команд — целочисленные, 15% — команды условных переходов, и только небольшой процент команд является командами с плавающей запятой.

Немедленно после введения второго или большего числа конвейеров возникают принципиально важные для суперскалярной структуры вопросы по организации и технической реализации вычислений:

1. Как организовать запуск параллельных команд во множественных конвейерах. Нетрудно видеть, что здесь решается и вопрос о распараллеливании последовательных программ.

2. Как уменьшить отрицательное влияние на работу конвейеров зависимости по данным для смежных команд и потери быст

родействия из-за появления в программе условных переходов. Планирование межконвейерного параллелизма должно производиться с учетом указанных эффектов.

3. Как обеспечить повышенную пропускную способность всех видов памяти в суперскалярном процессоре, особенно если число одновременно выполняемых операций составляет 20-30.

4. Каковы в перспективе предельные возможности, размеры суперскалярных процессоров.

Организация параллельного запуска команд предусматривает выполнение трех этапов: распараллеливание последовательной программы; планирование порядка выполнения команд для заданных ресурсов; представление спланированной программы в форме, пригодной для исполнения аппаратурой.

В зависимости от способа реализации этих этапов методы формирования и запуска параллельных команд можно разделить на статические, динамические и смешанные. Считается, что статические способы предпочтительнее, поскольку анализ программы в процессе компиляции обеспечивает более глубокое выделение и планирование параллелизма, кроме того, исключает служебные команды управления параллелизмом из вычислительного процесса [4]. Динамический же запуск позволяет более полно учитывать текущее состояние программы и ресурсов при исполнении программы.

Возможны следующие способы статического формирования параллельной команды:

1. Производится распараллеливание программы обычными методами, затем параллельные команды с помощью оптимизирующего планировщика упаковываются в виде СДК фиксированного формата. В таком СДК каждое поле формата закреплено за определенным конвейером и хранится в одной ячейке параллельной памяти. При чтении СДК из памяти выбираются все поля СДК независимо от их заполнения. Конвейеры запускаются синхронно в каждом такте. Это является возможным, поскольку в *RISC*-конвейерах основные операции выполняются за 1 такт. Основной недостаток схемы с фиксированным форматом СДК — неполное использование памяти при отсутствии команд в некоторых позициях СДК.

2. Более полное использование ячейки памяти фиксированного формата применяется в МП *i*860 (рис. 4.8).

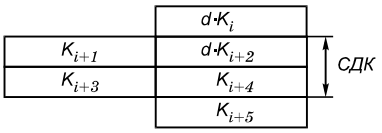

Рис. 4.8.Представление СДК в памяти МП *i*860

В МП *i*860 только два конвейера, поэтому СДК имеет длину 2. Если в некоторой команде, например, K_i указан признак *d* (*dual*-двойной), это означает, что следующие две команды по списку, а именно, K_{i+1} и K_{i+2} образуют параллельную пару. Физически эта пара может быть расположена либо в одной ячейке параллельной памяти, либо в смежных ячейках последовательной памяти.

3. Описанные выше способы годятся для вновь разрабатываемых систем команд, поскольку на этапе разработки можно заложить средства, обеспечивающие объединение команд в СДК фиксированного формата. Однако эти способы нельзя применить к системам команд *CISC*-процессоров, которые нельзя “подкорректировать”, поскольку для них уже существует большой объем написанного математического обеспечения.

В этом случае используются элементы смешанного формирования и запуска СДК. Рассмотрим это на примере суперскалярного МП Пентиум с двумя целочисленными конвейерами *U* и *V*, использующего систему команд *i*80×86 и имеющего те же ступени, что и на рис. 4.1.

Пусть имеется отрезок программы на ЯВУ и соответствующий ему ассемблерный вариант:

```
    DO 1 I = 1, 10          M:  mov edx, [eax + 40 + a]
    A (I) = A (I) + 1           mov ecx, [eax + 40 + b]
1   B (I) = B (I) + 1           inc edx
                                inc ecx
                                mov [eax + 40 + a], edx
                                mov [eax + 40 + b], ecx
                                jnz M
```

В этой программе команды расположены последовательно, никаких отметок о параллелизме команд нет, в МП Пентиум также отсутствует СДК фиксированного формата. Следовательно, программа для этого МП оформлена в памяти стандартным для всех МП 80×86 образом. Однако она существенно отличается от программ для МП 386/486, не являющихся суперскалярными. Отличие состоит в том, что на этапе компиляции скалярный параллелизм выявлен и команды переставлены так, что параллельные команды в программе являются смежными.

На ступени *D*1 целочисленных конвейеров аппаратно при последовательной выборке команд из памяти определяется текущий параллелизм. Проверка параллельности двух команд производится с учетом всех видов зависимостей по данным.

В результате потактного распараллеливания в конвейеры *U* и *V* будет загружена и исполнена следующая программа:

	Конвейер U	Конвейер V
M:	mov edx, [eax + 40 + a]	mov ecx, [eax + 40 + b]
	inc edx	inc ecx
	mov [eax + 40 + a], edx	mov [eax + 40 + b], ecx
	add eax, 4	jnz M

которая требует для своего исполнения всего 4 такта вместо 7 в исходном тексте.

Динамическое формирование и запуск СДВ используются в тех случаях, когда имеется большой объем прикладного матобеспечения в виде двоичных кодов и не представляется возможным произвести распараллеливание этих кодов с помощью программных распараллеливателей. Тогда задача распараллеливания возлагается на аппаратуру на этапе исполнения программ.

Рассмотрим следующие варианты динамического распараллеливания:

1. Покомандное формирование и запуск СДК в машинах типа CRAY, осуществляемое под управлением *табло* (scoreboard). Этот метод описан в § 2.1.

2. Пакетное формирование СДК по Флинну.

Суть алгоритма пакетного формирования и запуска СДК состоит в том, что из блока в *n* инструкций, обычно составляющих ББ или часть ББ, выбирается и немедленно исполняется *m* инст

рукций. Во время их выполнения в исходный блок добавляется *m* новых инструкций и вновь повторяется поиск независимых инструкций. Таким образом, блок из *n* инструкций напоминает “скользящее окно”, перемещаемое по программе по мере ее исполнения, а *m* параллельных команд составляют СДК.

Нетрудно видеть, что анализ на независимость блока из *n* команд требует много времени. С этой целью перед анализом на независимость команды преобразуются в форму, представленную на рис. 4.9. Здесь поля *E* и *D* содержат биты, установленные на номерах входных и выходного регистров, необходимых для выполнения данной команды. Следовательно, сравнение двух команд на зависимость сводится к побитному сравнению полей *E* и *D*, что может быть сделано быстро, и выбор *m* независимых команд может быть выполнен за время выполнения предыдущей СДК.

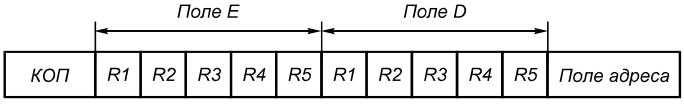

Рис.4.9. Модифицированный формат команды

Очевидно, что описанная схема, в отличие от ЭВМ CRAY, не требует предварительной перестановки команд ББ на этапе компиляции, что позволяет использовать традиционные компиляторы. Это является достоинством пакетной схемы. Однако большая длительность цикла формирования СДК аппаратурой снижает быстродействие процессора.

В качестве законченной структуры суперскалярного микропроцессора рассмотрим структуру МП Пентиум. Пентиум — это суперскалярный МП типа *CISC* семейства *i*80×86, следующий за МП 386 и 486. Он на 100% совместим на уровне двоичных кодов с МП 8086, 386*DX*, 486*DX*, 486*SX* и 486*DX*2. Структура Пентиума соответствует рис. 4.1, однако в нем содержится еще один целочисленный конвейер.

Таким образом, Пентиум содержит два целочисленных конвейера *U* и *V* и плавающий конвейер, входные ступени последнего (*PF*, *D*1, *D*2) являются общими с конвейером *U*. Назначение ступеней, особенности конвейеров описаны ранее в параграфах 4.2 и

4.3. При работе с плавающей запятой конвейеры *U* и *V* объединяются, создавая 64-разрядный обрабатывающий тракт.

Пентиум может запускать одну или две инструкции каждый такт. Чтобы запускать две инструкции одновременно, необходимо, чтобы они не содержали зависимостей по данным, были целочисленными и “простыми”. Простые команды выполняются за один такт. Исключением являются команды с обращением к памяти типа *mem*, *req* и *reg*, *mem*, которые требуют два и три такта соответственно, однако имеется специальная аппаратура, чтобы позволить этим командам выполняться, как простым.

Простыми являются следующие команды:

1. mov reg, reg/mem/imm
2. mov mem, reg/imm
3. alu reg, reg/mem/imm
4. alu mem, reg/imm
5. inc reg/mem
6. dec reg/mem (4.9)
7. push reg/mem
8. pop reg
9. lea reg, mem
10. jmp/call/jcc near
11. nop

Кроме того, команда межрегистрового обмена *FXCH* может образовывать пары с другими плавающими командами. Это сделано, как уже говорилось выше, чтобы ослабить ограничения на параллельную обработку, вызванные стековой организацией файла РОН плавающих регистров.

Быстродействие суперскалярного процессора в значительной степени определяется стилем программирования. Из этого следует, что существующие компиляторы для ЭВМ, построенных на базе МП *i*386, *i*486 не будут эффективными для Пентиума, поскольку они построены в расчете на экономию памяти и РОН, что приводит к появлению искусственных зависимостей по данным между смежными командами и уничтожает параллелизм команд.

Для суперскалярных процессоров необходимы новые компиляторы распараллеливающего типа.

Конвейер МП Пентиум выполняет команды строго в порядке их следования в программе. Такой способ функционирования существенно снижает эффективность конвейера, поскольку конвейер вынужден останавливаться, когда на этапе EX выполняется многотактная команда. Кроме того, для загрузки двух целочисленных конвейеров используется параллелиз только пары смежных команд, а он невелик, поэтому и вероятность совместной работы этих конвейеров невысока.

Повысить эффективность суперскалярной обработки можно, если допустить выполнение команд в порядке готовности их операндов. Такая архитектура использована в МП P6 фирмы Intel, упрощенная структурная схема которого представлена на рис. 4.10. В МП P6 используется та же система команд, что и во всех микропроцессорах i80*x*86.

Рис. 4.10. Структура МП с произвольным порядком выполнения команд

Функции блоков микропроцессора таковы:

1. Блок выборки и декодирования команд ВД обеспечивает чтение команд их КЭШ, их декодирование, преобразование и запись в буфер команд БК. Команды обрабатываются блоком ВД в порядке их следования в программе.

2. Буфер команд БК представляет собой ассоциативную память, в которой команды представлены в трехадресном формате.

Поскольку в системе команд i80x86 мало РОН (всего 8), то из-за этого между командами возникают ложные зависимости по данным. Для устранения этих зависимостей в МП P6 введено 40

дополнительных регистров, которые недоступны программисту. Они используются аппаратурой для временного хранения результатов. Обозначим эти регистры временного хранения через V. Тогда на рис. 4.10 V1, V2 иVP обозначают соответственно номера регистров для хранения первого, второго операндов и результата. Преобразование команд системы i80x86 в трехадресный формат и переименование регистров производится блоком ВД.

Каждая команда в БК сопровождается блоком событий БС, в котором отмечаются следующие состояния команды: готовность каждого операнда, готовность команды к исполнению, готовность результата и др.

3. Центральным блоком МП Р6 является блок планирования и выполнения команд ПВ. Именно он выполняет команды в порядке их готовности. ПВ содержит несколько АЛУ и устройств обращения к памяти. За один такт ПВ способен одновременно запустить на исполнение до пяти команд и передать в БК до пяти результатов.

Выборка готовых команд из БК производится путем ассоциативного опроса блока БС всех команд, а размещение нескольких команд по устройствам ПВ осуществляется в соответствии с определенным алгоритмом планирования. Одним из наиболее простых для аппаратной реализации является алгоритм FIFO (первым пришел — первым ушел).

При получении в ПВ каждого нового результата адрес его размещения используется как ассоциативный признак для полей V1, V2 буфера команд. Если одна или несколько команд откликнулись на этот признак, значит, эти команды завися по данным от выполненной команды. В блоках БС откликнувшихся команд устанавливаются биты и готовности операндов, а, если готовыми оказались оба операнда, то устанавливается и бит готовности для исполнения всей команды. Это позволяет сформировать очередной набор “готовых” команд для следующего такта работы ПВ.

Чтобы блок ПВ мог выполнять за один такт до 3...5 команд необходимо, чтобы в БК находилось до 20...30 команд. По статистике среди такого объема команд в среднем имеется 4...5 команд условных переходов. Следовательно в БК находится некоторая трасса выполнения команд. Выбор таких наиболее вероятных трасс является новой функцией МП с непоследовательным выпол

нением команд. Эта функция выполняется в блоке ВД на основе расширенного до 512 входов буфера истории переходов.

Поскольку реально вычисленный в ПВ адрес перехода не всегда совпадает с предсказанным в блоке ВД, то вычисление в ПВ выполняется условно, т. е. результат записывается в регистр временного хранения. Только после того, как установлено, что переход выполнен правильно, блок удаления команд УК выводит из БК все выполненные команды, расположенные за командой условного перехода, преобразует их в формат системы i80x86 и производит запись результатов по адресам, указанным в исходной программе.

Блоки ВД, ПВ и УК совместно составляют конвейер из 12 этапов, однако все три части этого конвейера работают практически независимо, взаимодействуя только через БК. Следовательно, такой конвейер можно считать неблокируемым, так как остановка любой его части не прекращает работу других частей.

§ 4.4. Другие типы параллельных процессоров и ЭВМ

Ранее были рассмотрены четыре основных типа параллельных структур: конвейерные процессоры, процессорные матрицы, многопроцессорные ЭВМ с управлением от потока команд и многопроцессорные ЭВМ с управлением от потока данных. Однако число типов реально производимых и разрабатываемых параллельных ЭВМ значительно больше и не укладывается в жесткие рамки приведенной классификации по двум причинам: появляются новые, ранее неизвестные типы ЭВМ; некоторые разновидности машин внутри определенного типа приобретают собственное наименование.

В настоящем параграфе будут кратко рассмотрены следующие важные для теории и практики типы параллельных машин: матричные процессоры, ассоциативные ЭВМ, систолические массивы, многопроцессорные системы с программируемой архитектурой [5].

Матричные процессоры. С понятием *матричный процессор* связывают три различных характеристики: способ организации данных, способ организации устройств и класс решаемых задач.
В практике проектирования ЕС ЭВМ под матричным процессором понимают устройство, предназначенное в основном для выполне

ния операции $S = \sum_{l=1}^{i} X_i Y_i$. Такой матричный процессор функционирует на основе арифметического конвейера и отличается от структур, описанных в § 2.2, тем, что для уменьшения стоимости и упрощения программирования арифметический конвейер значительно укорочен и конструктивно выполнен в виде автономного блока. Последний подключается через стандартные каналы ввода-вывода к базовой ЭВМ, в качестве которой может использоваться любая серийная ЭВМ. Собственной ОП матричный процессор не имеет. Его программа и данные располагаются в ОП БМ.

Способ подключения матричного процессора к БМ и его структура изображены на рис. 4.11. Матричный процессор вместо одного из каналов подключается на внешнюю шину и содержит регистры для управления адресом оперативной памяти, буферные регистры для промежуточного хранения данных *БХ*, *БU, БУ* и двухступенчатое АЛУ.

Команда матричного процессора (рис. 4.12) состоит из четырех слов по 8 байт каждое. Описание массивов операндов *X*, *U*, *Y* определяет начальный адрес каждого массива, шаг по массиву (поле “Индекс”) и длину массива. Поле “Формат” указывает тип операндов (фиксированная, плавающая запятая и т. д.). Первое слово команды матричного процессора имеет структуру стандартной команды канала ЕС ЭВМ, однако назначение полей несколько иное. Так, поле “КОП” задает код арифметико-логической операции над массивами *X*, *U*; поле “Адрес описания массивов” задает начальный адрес описания массивов в ОП, а поле “Длина описания” — его длину.

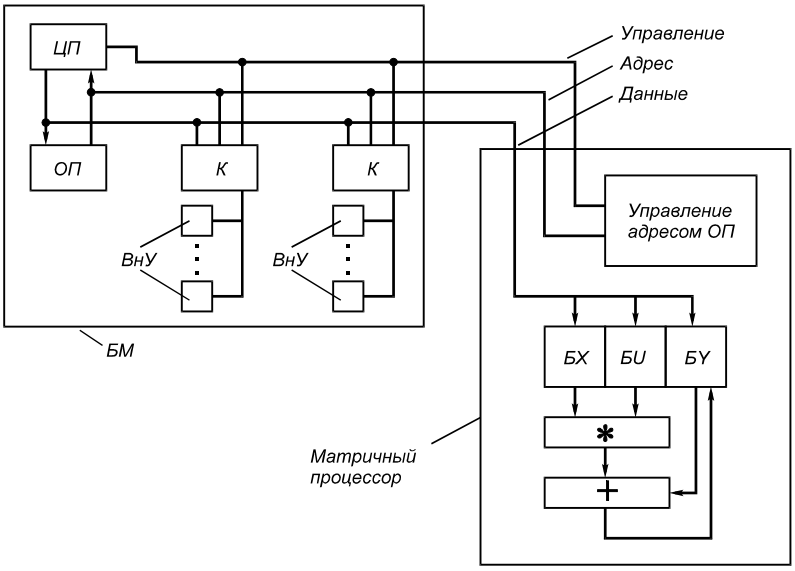

Рис. 4.11. Способ подключения и структура матричного процессора:
К — каналы ввода-вывода; * — умножитель; + — сумматор

Рис. 4.12. Структура команды матричного процессора

Программирование для матричного процессора состоит в том, что в программу для БМ вставляются в нужных местах команды матричного процессора, которые имеют вид

CALL APAM < код операции, описание массивов >

Это означает, что любая команда матричного процессора эквивалентна некоторой подпрограмме. Если в процессе выполнения

программы в БМ встречена команда *CALL APAM,* в матричный процессор передается информация об адресах массивов *X*, *U*, *Y*, после чего матричный процессор самостоятельно вырабатывает адреса ОП, выбирает из ОП в *БХ*, *БУ* операнды и размещает в ОП из *БУ* результаты. После выполнения команды матричного процессора центральный процессор продолжает программу базовой ЭВМ.

Система команд матричного процессора, которая позволяет решать сложные задачи в области геофизики, метеорологии, радиолокации, медицины и т. д., приведена в табл. 4.3. Знак [ ] означает необязательную часть операции, которая может быть опущена при использовании команд матричного процессора

Ассоциативные ЭВМ. Все ПМ потенциально обладают возможностью *ассоциативной обработки*, но в машинах широкого применения эти свойства не всегда явно выражены. Тогда для ускорения специальных видов ассоциативной обработки в ПМ вводят дополнительные средства и такие матрицы называются ассоциативными ЭВМ. Примером подобной ЭВМ является машина STARAN.

В машине STARAN, используется БМ, имеется ЦУУ, однако само процессорное поле устроено иначе (рис. 4.13).

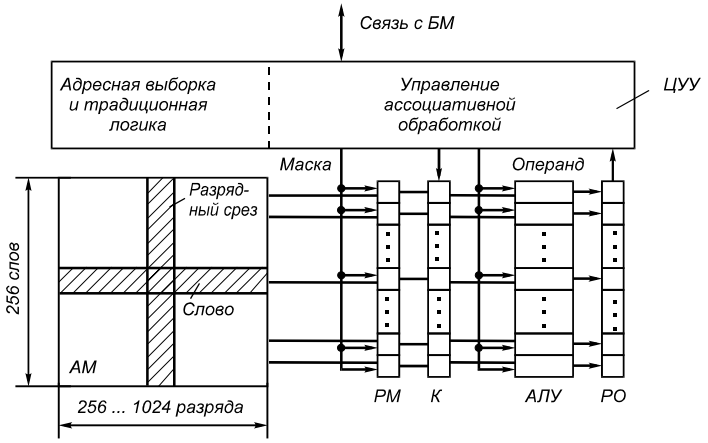

Рис. 4.13. Структура АМ