

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Нижегородский государственный университет им. Н.И. Лобачевского

Е.И. Шкелев

Аппаратные средства вычислительной техники

Учебное пособие

Рекомендовано ученым советом радиофизического факультета для студентов
ННГУ, обучающихся по направлениям подготовки 010800
«Радиофизика» и 090106 «Информационная безопасность
телекоммуникационных систем»

Нижний Новгород
2010

УДК 681.3
ББК 32.973.2
Ш-66

Ш-66 Шкелев Е.И. АППАРАТНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ: Учебное пособие. – Нижний Новгород: Нижегородский госуниверситет, 2010. – 222 с.

Рецензенты:

заведующий кафедрой теории цепей и телекоммуникаций ИРИТ
Нижегородского государственного технического университета
им. Р.Е. Алексеева д. ф.-м. н профессор В.И. Есипенко,

профессор кафедры бионики и статистической радиофизики
Нижегородского государственного университета им. Н.И. Лобачевского
д.т.н. В.Т. Ермолаев

В пособии изложены логика работы и схемотехнические решения цифровых функциональных узлов, включая базовые логические элементы, комбинационные схемы для выполнения логических и вычислительных операций и цифровые автоматы. К последним относятся элементы памяти (триггеры), запоминающие устройства и различной степени сложности машины состояний от регистров и двоичных счетчиков до арифметическо-логического устройства и микропрограммного автомата как основы для построения процессорных элементов (микропроцессоров, в частности). Рассмотрены архитектурные решения и логика работы центрального процессора (микропроцессора), особенности CISC- и RISC-архитектуры, построение и функциональный состав микропроцессорных систем, включая машину фон-Неймана, системы с гарвардской архитектурой и многопроцессорные системы. Приводятся примеры построения микроконтроллеров, цифровых процессоров сигналов и процессоров общего назначения.

Учебное пособие предназначено для студентов третьего-четвертого курсов естественнонаучных и технических высших учебных заведений, изучающих цифровые вычислительные устройства и устройства цифровой обработки сигналов, включая программируемую логику, микропроцессоры и микропроцессорные системы.

УДК 681.3
ББК 32.973.2

©Нижегородский государственный
университет им. Н.И. Лобачевского, 2010

Предисловие

Предмет, которому посвящено данное учебное пособие, можно рассматривать с разных точек зрения. Выделим две из них. Первая направлена на аппаратную реализацию цифровых вычислительных систем (ВС). Вторая связана с механизмом перемещения информации внутри системы и со способами взаимодействия ВС с внешними по отношению к ней устройствами.

Если вычислительное устройство представить как систему, основным элементом которой является процессор, то при ее реализации потребуется решить

- какие компоненты включить в состав процессорной системы,
- какими средствами поддерживать их работу,
- как обеспечить взаимодействие компонент системы друг с другом и функционирование вычислительной системы в целом,
- каковы способы передачи и способы селекции сигналов, которыми обмениваются компоненты системы,
- каковы конструктивное оформление и регламент работы используемых при этом линий (каналов) связи.

Принцип действия современных, как и ранее созданных электронных цифровых устройств во многом остается неизменным. Существенно меняется элементная база, основой которой стали микросхемы, разнообразные как по уровню интеграции, так и по функциональным возможностям и принадлежности.

В пособии рассмотрены составные части и логика работы цифровых функциональных узлов разной степени сложности, начиная от простейших логических элементов до сложных цифровых автоматов, каковым является, например, микропрограммный автомат. В отдельный раздел вынесены вопросы, связанные с иерархией памяти, поддержкой виртуального адресного пространства, функционированием кэш-памяти и построением многопроцессорных систем. Показывается, как выглядит обобщенная архитектура и алгоритм работы центрального процессора (ЦП) и какие функциональные узлы должны входить в его состав. Обсуждаются особенности CISC и RISC процессоров, вопросы конвейеризации и распараллеливания вычислительных операций. Рассматривается работа процессорной системы в целом, механизм взаимодействия процессора, памяти и устройств ввода-вывода, архитектурные решения микропроцессорных систем, включая архитектуру фон-Неймана, гарвардскую архитектуру и многопроцессорные системы. Приводятся примеры построения микроконтроллеров, цифровых процессоров сигналов и процессоров общего назначения.

1. Основные положения алгебры логики

Используемые в цифровой технике сигналы близки по форме к прямоугольным и имеют два фиксированных уровня:

низкий	0 – для положительной логики, 1 – для отрицательной логики
высокий	1 – для положительной логики, 0 – для отрицательной логики

Математическим аппаратом анализа и синтеза цифровых систем служит алгебра логики (булева алгебра – по имени создателя Дж. Буля).

Алгебра логики – это алгебра состояний, а не чисел. Любое логическое выражение, любая логическая функция

$$y = f(x_{n-1}, \dots, x_1, x_0)$$

принимает только два значения

$$y = 0, 1 \text{ (ложь – истина, да – нет),}$$

точно так же, как и аргументы

$$x_{n-1}, \dots, x_1, x_0 = 0, 1 \text{ (ложь – истина, да – нет).}$$

В основе алгебры логики лежат три основные операции:

1) логическое сложение (дизъюнкция, ИЛИ)

$$y = x_{n-1} + \dots + x_1 + x_0 \equiv x_{n-1} \vee \dots \vee x_1 \vee x_0 \text{ (рис. 1.1);}$$

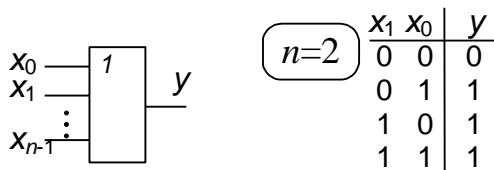


Рис. 1.1. Элемент ИЛИ

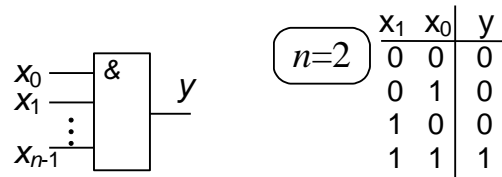


Рис. 1.2. Элемент И

2) логическое умножение (конъюнкция, И)

$$y = x_{n-1} \cdot \dots \cdot x_1 \cdot x_0 \equiv x_{n-1} \wedge \dots \wedge x_1 \wedge x_0 \text{ (рис. 1.2);}$$

3) инверсия (отрицание, НЕ), рис. 1.3.

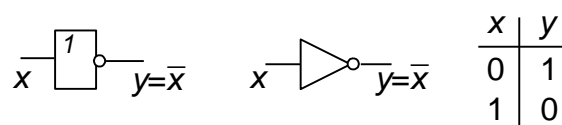


Рис. 1.3. Элемент НЕ

Операции И, ИЛИ и НЕ в совокупности образуют *логический базис*. С его помощью можно представить любое сколь угодно сложное логическое выражение. Есть две формы этого представления:

- совершенная дизъюнктивная (СДНФ) и
- совершенная конъюнктивная (СКНФ) нормальные формы.

При составлении СДНФ сначала задаются все значения $y = y_i$ и соответствующие им комбинации значений аргументов $(x_{n-1}, \dots, x_1, x_0)_i$:

$$(x_{n-1}, \dots, x_1, x_0)_i \rightarrow y_i; \quad x_i, y_i = 0 \text{ или } 1.$$

Сделать это можно в виде таблицы, которую называют *таблицей истинности*. Примером может послужить таблица 1.1 для некоторой произвольной функции $y = f(x_0, x_1, x_2, x_3)$ четырех переменных.

Затем по всем возможным комбинациям значений аргументов по приведенным ниже правилам составляются сначала конъюнкции K_i , затем их произведения $y_i \cdot K_i$ с требуемыми значениями $y = y_i$:

Таблица 1.1				
y	x ₃	x ₂	x ₁	x ₀
1	0	0	0	0
0	0	0	0	1
1	0	0	1	0
1	0	0	1	1
⋮	⋮	⋮	⋮	⋮
0	1	1	1	0
1	1	1	1	1

$y_0 \cdot K_0 \quad \leftarrow \overline{x_{n-1}} \cdot \overline{x_{n-2}} \cdot \dots \cdot \overline{x_1} \cdot \overline{x_0} = K_0;$
 $y_1 \cdot K_1 \quad \leftarrow \overline{x_{n-1}} \cdot \overline{x_{n-2}} \cdot \dots \cdot \overline{x_1} \cdot x_0 = K_1;$
 $y_2 \cdot K_2 \quad \leftarrow \overline{x_{n-1}} \cdot \overline{x_{n-2}} \cdot \dots \cdot x_1 \cdot \overline{x_0} = K_2;$
 \vdots
 $y_{2^{n-2}} \cdot K_{2^{n-2}} \leftarrow \overline{x_{n-1}} \cdot \overline{x_{n-2}} \cdot \dots \cdot x_1 \cdot \overline{x_0} = K_{2^{n-2}};$
 $y_{2^{n-1}} \cdot K_{2^{n-1}} \leftarrow \overline{x_{n-1}} \cdot \overline{x_{n-2}} \cdot \dots \cdot x_1 \cdot x_0 = K_{2^{n-1}}.$

В K_i табличным значениям $x_i = 0$ соответствуют \bar{x}_i ($\bar{x}_i \rightarrow x_i = 0$), а значениям $x_i = 1 \rightarrow x_i$. Конъюнкции K_i называют *минтермами*. Их общее число равно 2^n – числу возможных комбинаций двоичного кода $x_{n-1}, x_{n-2}, \dots, x_1, x_0$. СДНФ получается после логического сложения всех полученных произведений:

$$y = \sum_{i=0}^{2^n-1} y_i \cdot K_i \equiv \bigcup_{i=0}^{2^n-1} y_i \wedge K_i. \quad (1.1)$$

Аналогичным образом составляется СКНФ, но заменой конъюнкций K_i на дизъюнкции D_i , называемые *макстермами*. Макстермы D_i суммируются с соответствующими значениями $y = y_i$ и полученные после сложений результаты логически перемножаются:

$$y = \prod_{i=0}^{2^n-1} (y_i + D_i) \equiv \bigcap_{i=0}^{2^n-1} (y_i \vee D_i). \quad (1.2)$$

Здесь

$$D_0 = \bar{x}_{n-1} \vee \bar{x}_{n-2} \vee \dots \vee \bar{x}_1 \vee \bar{x}_0;$$

$$D_1 = \bar{x}_{n-1} \vee \bar{x}_{n-2} \vee \dots \vee \bar{x}_1 \vee x_0;$$

$$D_2 = \bar{x}_{n-1} \vee \bar{x}_{n-2} \vee \dots \vee x_1 \vee \bar{x}_0;$$

$$\vdots$$

$$D_{2^n-1} = x_{n-1} \vee x_{n-2} \vee x_1 \vee \dots \vee x_1 \vee x_0.$$

Схемная реализация логических функций всегда предполагает минимизацию соответствующих логических выражений. При этом уменьшается число логических операций и, как следствие, уменьшаются аппаратные затраты. Для минимизации логических выражений используются *аксиомы* и *законы* алгебры логики. Они достаточно очевидны и потому ограничимся их простым перечислением.

Аксиомы

1) $\overline{(\bar{x})} = x;$	6) $x \wedge 0 = 0;$
2) $x \vee 0 = x;$	7) $x \wedge 1 = x;$
3) $x \vee 1 = 1;$	8) $x \wedge x = x;$
4) $x \vee x = x;$	9) $x \wedge \bar{x} = 0.$
5) $x \vee \bar{x} = 1;$	

Законы

1) **Переместительный (закон коммутативности):**

$$x \vee y = y \vee x; \quad x \wedge y = y \wedge x;$$

2) **Сочетательный (закон ассоциативности):**

$$x \vee y \vee z = (y \vee x) \vee z = x \vee (y \vee z);$$

$$x \wedge y \wedge z = (y \wedge x) \wedge z = x \wedge (y \wedge z).$$

3) **Распределительный (закон дистрибутивности):**

$$x \wedge (y \vee z) = (y \wedge x) \vee (x \wedge z); \quad (\text{или } x \cdot (y+z) = (y \cdot x) + (x \cdot z)).$$

В качестве примера рассмотрим функцию двух переменных – функцию равнозначности (эквивалентности)

$$y = x_0 \sim x_1. \tag{1.3}$$

Таблица 1.2

x_1, x_0	y	Минтермы	Макстермы
0, 0	$y_0 = 1$	$K_0 = \bar{x}_1 \cdot \bar{x}_0$	$D_0 = \bar{x}_1 + \bar{x}_0$
0, 1	$y_1 = 0$	$K_1 = \bar{x}_1 \cdot x_0$	$D_1 = \bar{x}_1 + x_0$
1, 0	$y_2 = 0$	$K_2 = x_1 \cdot \bar{x}_0$	$D_2 = x_1 + \bar{x}_0$
1, 1	$y_3 = 1$	$K_3 = x_1 \cdot x_0$	$D_3 = x_1 + x_0$

Составим ее СДНФ и СКНФ, а затем минимизируем полученные выражения. Минимизированные выражения называются соответственно *минимальной дизъюнктивной (МНДФ)* и *минимальной конъюнктивной (МКНФ) формами*. В

общем случае вид МНДФ и МНКФ различен, но не исключается и их совпадение. Таблица истинности для функции (1.3) представлена в таблице 1.2. В таблице 1.2 помимо значений функции y приведены выражения для всех ее минтермов и макстермов.

При переходе от табличного представления к алгебраическому каждому набору значений переменных в соответствие ставится либо минтерм, либо макстерм.

СДНФ для функции (1.3) равна

$$\begin{aligned} y &= (y_0 \cdot K_0) + (y_1 \cdot K_1) + (y_2 \cdot K_2) + (y_3 \cdot K_3) = \\ &= 1 (\bar{x}_0 \bar{x}_1) + 0 (\bar{x}_0 x_1) + 0 (x_0 \bar{x}_1) + 1 (x_0 x_1), \end{aligned}$$

и ей соответствует МНДФ

$$y = x_0 x_1 + \bar{x}_0 \bar{x}_1. \quad (1.4)$$

Другая алгебраическая форма представления функции (СКНФ) получается при использовании макстермов:

$$\begin{aligned} y &= (y_0 + D_0)(y_1 + D_1)(y_2 + D_2)(y_3 + D_3) = \\ &= (1 + \bar{x}_1 + \bar{x}_0)(0 + \bar{x}_1 + x_0)(0 + x_1 + \bar{x}_0)(1 + x_1 + x_0) = (\bar{x}_1 + x_0)(x_1 + \bar{x}_0) = \\ &= x_1 \cdot \bar{x}_1 + \bar{x}_1 \cdot \bar{x}_0 + x_0 \cdot x_1 + x_0 \cdot \bar{x}_0 = x_0 \cdot x_1 + \bar{x}_0 \cdot \bar{x}_1. \end{aligned}$$

Получающееся из нее минимизированное выражение также равно (1.4), что лишний раз подчеркивает равноправность обеих форм представления логических функций.

Как уже говорилось, любую логическую функцию можно представить как совокупность простейших операций логического базиса И, ИЛИ и НЕ. Но наряду с уже перечисленными законами алгебры логики важную роль играет закон инверсии для логических операций сложения и умножения, который известен как теорема де Моргана:

$$\begin{aligned} \overline{x \vee y \vee z} &= \bar{x} \cdot \bar{y} \cdot \bar{z}; \\ \overline{x \cdot y \cdot z} &= \bar{x} \vee \bar{y} \vee \bar{z}. \end{aligned} \quad (1.5)$$

Из этой теоремы следует, что всегда операцию И можно заместить операцией ИЛИ и наоборот, если изменить тип логики и перейти от логики положительной к логике отрицательной. По этой причине наряду с логическим базисом существует понятие *минимального логического базиса*, в основе которого лежат либо операции И и НЕ, либо операции ИЛИ и НЕ.

2. Схемотехническая реализация логических операций и функций

Рассмотрим наиболее часто используемые на практике логические узлы. К ним относятся как простые логические элементы, выполняющие основные операции И, ИЛИ и НЕ, так и более сложные функциональные блоки. Первоначально остановимся на относительно сложных функциях дешифрации и мультиплексирования. Связано это с тем, что названные функции позволяют наиболее простыми средствами реализовать СДНФ.

2.1. Полный дешифратор

Функции дешифрования, как и шифровальные функции, достаточно разнообразны и определяются конкретными приложениями. Здесь рассмотрим так называемый *полный дешифратор* (рис. 2.1) – устройство, реализующее весь набор минтермов от n переменных. Выходные сигналы полного дешифратора $F_i = K_i$ ($i = 0, 1, \dots, 2^n - 1$):

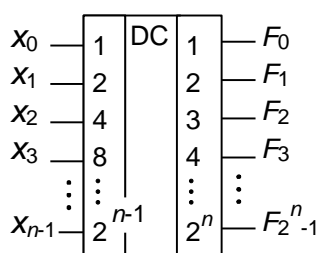


Рис. 2.1. Схемное обозначение дешифратора

$$F_0 = \overline{x_{n-1}} \cdot \dots \cdot \overline{x_1} \cdot \overline{x_0} = K_0 \text{ – на всех входах «0»};$$

$$F_1 = \overline{x_{n-1}} \cdot \dots \cdot \overline{x_1} \cdot x_0 = K_1 \text{ – только } x_0 = 1;$$

$$F_2 = \overline{x_{n-1}} \cdot \dots \cdot x_1 \cdot \overline{x_0} = K_2 \text{ – только } x_1 \text{ равен 1};$$

$$\vdots$$

$$F_{2^n-1} = x_{n-1} \cdot \dots \cdot x_1 \cdot x_0 = K_{2^n-1} \text{ – все } x_i \text{ равны 1.}$$

Полный дешифратор можно рассматривать как преобразователь двоичного кода

$$(x_{n-1}, \dots, x_1, x_0) \rightarrow x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \dots + x_1 \cdot 2^1 + x_0 \cdot 2^0$$

(n – разрядность двоичного кода) в унитарный, в котором значение только одного разряда отлично от нуля, а все остальные равны нулю.

Для полного дешифратора необходимы n логических элементов НЕ и 2^n n -входовых (n И) элементов И. В качестве примера на рис. 2.2 показана логическая схема полного дешифратора 2×4 .

Функциональные возможности дешифратора можно расширить, если во все элементы И добавить по одному – ($n+1$)-му – входу, соединив его с общим управляющим

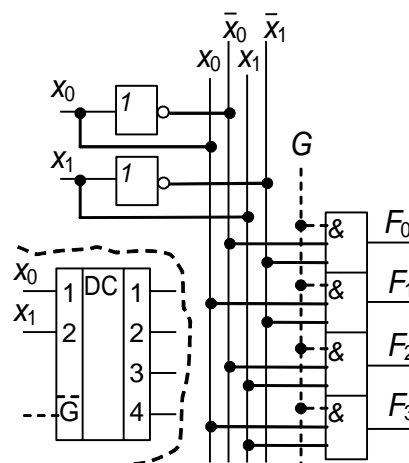


Рис. 2.2. Дешифратор 2×4

входом G . Тогда выходные функции дешифратора будут равны

$$F_i = G \cdot K_i. \quad (2.1)$$

В этом случае на выходах F_i будет результат дешифрации только при условии $G = 1$. Если $G = 0$, то всегда и на всех выходах будут логические нули. Такое устройство относится к классу *стробируемых дешифраторов*.

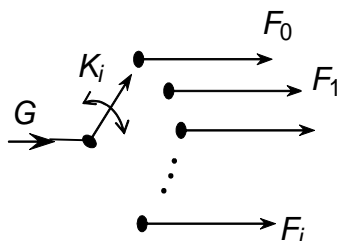


Рис. 2.3

Логическим выражением (2.1) можно описать работу еще одного устройства. Если вход G принять за информационный, а двоичный код $(x_{n-1}, \dots, x_1, x_0)$ считать кодом переключения выходов (рис. 2.3), то получаем устройство, функция которого состоит в передаче цифровых сигналов по одному из нескольких задаваемых кодом $(x_{n-1}, \dots, x_1, x_0)$ направлений. Это функция *демультиплексора*, являющегося разновидностью коммутаторов цифровых сигналов.

2.2. Мультиплексор

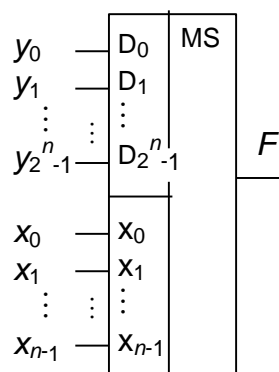


Рис. 2.4.

Мультиплексор

Если судить по названию, то мультиплексор также выполняет функцию коммутации, но противоположную той, которая свойственна демультиплексору. Однако сначала рассмотрим мультиплексор с другой позиции – с точки зрения логики его работы. Подаваемые на мультиплексор (рис. 2.4) сигналы разбиты на две группы. Одна из них образована варьируемыми комбинациями двоичного кода

$$(x_{n-1}, \dots, x_1, x_0)_i; \quad i = 0, 1, \dots, 2^n - 1.$$

Другая – фиксированной комбинацией значений $y_{2^n-1}, \dots, y_1, y_0$, устанавливаемой на входах D_j ($j = 0, 1, \dots, 2^n - 1$). По каждой из комбинаций $(x_{n-1}, \dots, x_1, x_0)_i$ образуются минтермы K_i , которые перемножаются с соответствующими по i значениями y_i . После логического сложения всех произведений образуется СДНФ (1.1) функции $y = f(x_{n-1}, \dots, x_1, x_0)$ n переменных. Вид этой функции определяется задаваемой заранее комбинацией $(y_{2^n-1}, \dots, y_1, y_0)$. В этом состоит одно из предназначений мультиплексора как функционального блока, реализующего СДНФ функции n переменных.

В соответствии с (1.1) логическая схема мультиплексора включает 2^n 2-входовых элементов И (элементов 2И), один 2^n -входовый элемент ИЛИ (элемент 2^n ИЛИ) и полный дешифратор $n \times (2^n)$. Для примера на рис. 2.5 приведена логическая схема мультиплексора, реализующего функцию $y = f(x_1, x_0)$ двух переменных, и показано его схемное обозначение.

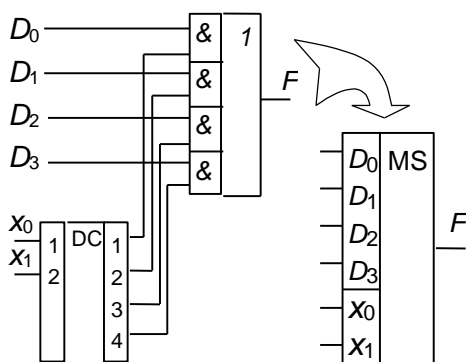


Рис. 2.5. Логическая схема мультиплексора 4×1

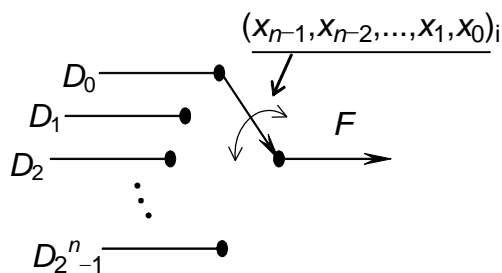


Рис. 2.6. Пояснение функции мультиплексора

Как уже упоминалось, другим предназначением мультиплексора, которое и определило его название, является коммутация цифровых сигналов (данных D_i), идущих от n источников, на одну информационную линию F (рис. 2.6). В этом случае сигналы на входах x_{n-1}, \dots, x_1, x_0 являются сигналами выбора одного из входов $D_{2^n-1}, \dots, D_1, D_0$, на которые поступают передаваемые данные.

2.3. Схемотехника базовых логических операций

Рассмотренные выше функциональные блоки (мультиплексор и демультиплексор) относительно сложны. Их работа представлена лишь логическими схемами без указания на физическую реализацию. На более низком уровне работа сложных логических узлов описывается посредством простейших операций (операций И, ИЛИ и НЕ), на которые распадаются выполняемые этими узлами функции. Схемотехнический уровень – это самый низкий уровень описания цифровых устройств, на основании которого выполняется их техническая, или, как еще говорят, аппаратная реализация.

Существуют разные технологические решения и варианты цифровых интегральных схем (ИС). Однако в общем случае логические элементы строятся по схеме, состоящей из двух частей или ступеней:

- 1) цепей, выполняющих логические операции;
- 2) цепей согласования логических ступеней с внешней (подключаемой к логическим элементам) нагрузкой.

Сначала остановимся на той части ИС, которая предназначена для выполнения операций дизъюнкции и конъюнкции.

2.3.1. Диодные дизъюнкторы

Схема диодного дизъюнктора состоит из развязывающих диодов D_i ($i = 0, 1, \dots, n-1$; n – число входов) и нагрузочного сопротивления R_H (рис. 2.7).

Резисторы R_0, R_1, \dots, R_{n-1} не являются функционально необходимыми и представляют внутренние сопротивления диодов. Обычно источники сигналов u_0, u_1, \dots, u_{n-1} имеют малые внутренние сопротивления. Если это не так, то их внутренние сопротивления можно включить в R_0, R_1, \dots, R_{n-1} .

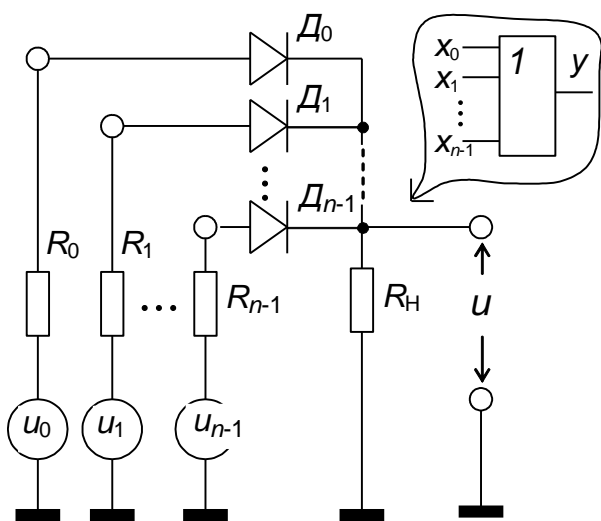


Рис. 2.7. Диодный дизъюнктор

Напряжения u_0, u_1, \dots, u_{n-1} соответствуют входным переменным x_0, x_1, \dots, x_{n-1} , а напряжение u – выходной переменной y логического элемента:

$$u_0 \rightarrow x_0, u_1 \rightarrow x_1, \dots, u_{n-1} \rightarrow x_{n-1};$$

$$u \rightarrow y.$$

В положительной логике

$$u_0, u_1, \dots, u_{n-1} = \begin{cases} U^+ > 0 \rightarrow x = 1; \\ U^- = 0 \rightarrow x = 0. \end{cases}$$

Высокому U^+ и низкому U^- уровням напряжений соответствуют логическая единица и логический нуль. При этом высокий уровень выходного напряжения u не должен зависеть от сопротивлений R_0, R_1, \dots, R_{n-1} . Поэтому их величины должны быть значительно меньше сопротивления нагрузки:

$$R_0, R_1, \dots, R_{n-1} \ll R_H.$$

Тогда появление на любом из входов (или на любой группе входов) высокого уровня U^+ приводит к возникновению тока через соответствующий диод (или группу диодов) и выходное напряжение будет примерно равно высокому уровню U^+ ($u \approx U^+$), соответствуя $y = 1$. Таким образом в *положительной логике* выполняется операция логического сложения (операция *n*ИЛИ)

$$y = x_{n-1} \vee \dots \vee x_1 \vee x_0.$$

Однако если перейти *от логики положительной к логике отрицательной*, то представленная на рис. 2.7 схема будет выполнять операцию конъюнкции. Действительно, только в том случае, когда на всех входах u_0, u_1, \dots, u_{n-1} присутствует низкий уровень напряжения (на всех входах – логические «1»), все диоды D_i закрыты и на выходе будет низкий уровень напряжения, то есть логическая «1». Во всех остальных случаях на выходе будет логический «0». Следовательно, в *отрицательной логике* реализуется функция И ($y = x_{n-1} \wedge \dots \wedge x_1 \wedge x_0$). Такая двойственность одного логического элемента в алгебре логики сформулирована как теорема де Моргана (см. соотношения (1.5)) и потому имеет характер общей закономерности, означающей, что любой базовый логический элемент (И или ИЛИ) в зависимости от типа логики можно

использовать как элемент И/ИЛИ (И – в положительной, ИЛИ – в отрицательной логике) или ИЛИ/И (ИЛИ – в положительной, И – в отрицательной логике).

2.3.2. Диодные конъюнкторы

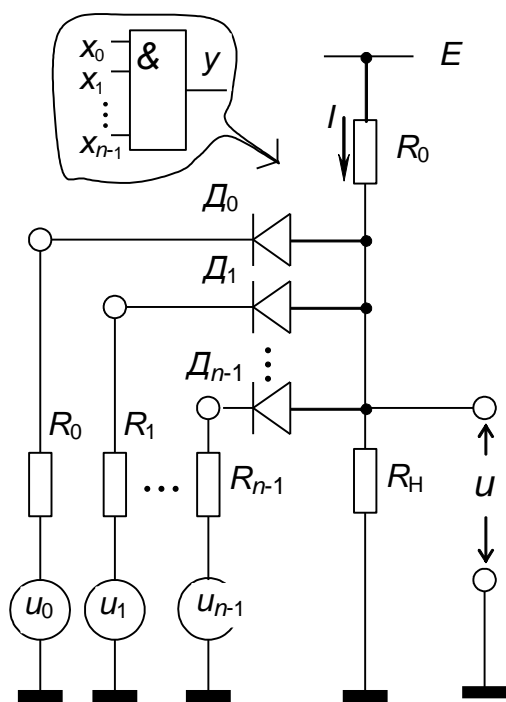


Рис. 2.8. Диодный конъюнктор

пропускают ток, если

$$u_0 = u_1 = \dots = u_{n-1} = 0 \quad (x_0 = x_1 = \dots = x_{n-1} = 0).$$

При этом напряжение на выходе $u \approx 0$ и $y = 0$. Включение высокого уровня U^+ по любому из входов $u_0 \dots u_{n-1}$ не изменит величины u и оставит $y = 0$ за исключением того случая, когда

$$u_0 = u_1 = \dots = u_{n-1} = U^+ \quad (x_0 = x_1 = \dots = x_{n-1} = 1). \quad (2.2)$$

При условии (2.2) $u = U^+$ и $y = 1$. Следовательно, в положительной логике реализуется операция *n*И: $y = x_{n-1} \wedge \dots \wedge x_1 \wedge x_0$. Отметим еще раз, что смена логики на отрицательную превращает конъюнктор в дизъюнктор. Справедливо это для всех разновидностей логических элементов.

2.3.3. Выполнение логических операций с помощью транзисторов

Схемотехническая реализация логических элементов предполагает наличие в том или ином виде электронных ключей, которые могут пребывать в

двух (если не учитывать переходные процессы) состояниях – закрытом и открытом. Подобное имеет место в рассмотренных выше диодных логических узлах. Транзисторные схемы разнообразны не только по своей структуре, но и по используемым в них типам транзисторов. Для построения цифровых интегральных схем в настоящее время по большей части используется кремний (Si), который служит материалом для создания ИС *со сверхвысокой степенью интеграции* (до и более нескольких десятков миллионов транзисторов на кристалл) и *сверхвысокой производительностью*. Для создания *сверхбыстродействующих* цифровых устройств применяются также полупроводники группы $A_{III}B_{V}$, в частности арсенид галлия (GaAs), однако уровень интеграции таких ИС не выше средней (до нескольких сотен транзисторов на кристалл).

Основные разновидности применяемых в цифровых ИС транзисторов представлены в таблице 2.1. Интегральные схемы, как правило, имеют планарную структуру, и ее элементы получают на поверхности полупроводника методами литографии (для кремниевых приборов) или молекулярно-лучевой эпитаксии (для GaAs-приборов). Отсюда происходит используемый в таблице термин «подложка».

Выделим сначала группу кремниевых приборов, в основе работы которых лежат контакты полупроводников с разным типом проводимости – электронной n и дырочной p . Это биполярные (p - n - p или n - p - n), а также полевые с управляющим p - n -переходом транзисторы.

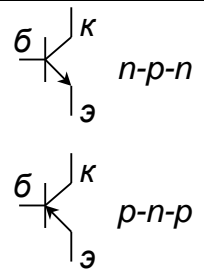
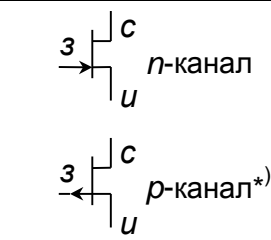
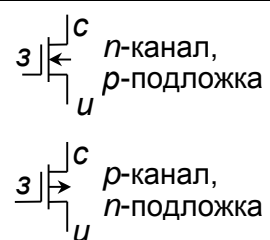
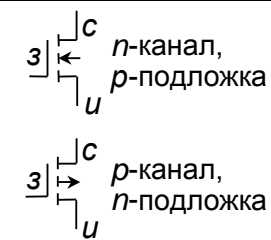
С точки зрения применений основное отличие биполярных транзисторов от полевых заключается в том, что биполярные транзисторы – это приборы, управляемые током, а полевые – приборы, управляемые напряжением¹ (что отражено в названии «полевые»). Это значит, что сопротивление входов логических элементов на биполярных транзисторах значительно ниже, чем у элементов на полевых транзисторах. Последнее накладывает свой отпечаток как на техническую реализацию логических операций, так и на способы согласования одних элементов с другими.

Полевые транзисторы с управляющим p - n -переходом подразделяются по типу проводимости канала: имеются транзисторы с n -каналом (p -затвором) и транзисторы с p -каналом (n -затвором).

У полевых транзисторов на арсениде галлия управляющим является барьер Шоттки – переход металл–полупроводник. Канал транзистора – это примыкающая к металлическому затвору область полупроводника, по обе стороны от которой расположены исток и сток. Такая структура транзистора нашла отражение в его названии МЭП(металл–полупроводник)-транзистор.

¹ Управляющим напряжением для биполярных транзисторов является разность потенциалов между базой и эмиттером, а для полевых – между затвором и истоком.

Разновидности и схемные обозначения транзисторов

Биполярные транзисторы	Полевые транзисторы		
	С управляющим <i>p-n</i> -переходом и транзисторы с барьером Шоттки (МЭП-транзисторы)	С изолированным затвором (МДП-транзисторы)	
		С встроенным каналом (нормально открытые транзисторы)	С индуцированным каналом (нормально закрытые транзисторы)
 <p> <i>n-p-n</i> <i>p-n-p</i> э - эмиттер к - коллектор б - база </p>	 <p> <i>n</i>-канал <i>p</i>-канал*) и - исток с - сток з - затвор </p>	 <p> <i>n</i>-канал, <i>p</i>-подложка <i>p</i>-канал, <i>n</i>-подложка и - исток с - сток з - затвор </p>	 <p> <i>n</i>-канал, <i>p</i>-подложка <i>p</i>-канал, <i>n</i>-подложка и - исток с - сток з - затвор </p>

*) МЭП-транзисторы имеют только *n*-каналы.

В цифровых интегральных схемах широко применяются транзисторы с изолированным, то есть отделенным от канала слоем диэлектрика, затвором. Образованная таким образом структура металл–диэлектрик–полупроводник (МДП) дала название этому типу транзисторов – МДП-транзисторы. Их отличительной особенностью является высокое сопротивление постоянному току со стороны управляющего электрода – затвора. Во всем многообразии применений в интегральной схемотехнике МДП-транзисторы до настоящего времени являются преимущественно кремниевыми приборами. В качестве диэлектрика в МДП-структурах традиционно применялась и применяется двуокись кремния SiO_2 . В таких случаях для обозначения МДП-транзисторов используется аббревиатура МОП (металл–окисел–полупроводник) вместо МДП.

Как и все полевые МДП-транзисторы различаются по типу проводимости канала (p - или n -типу). К этому добавляется еще различие по пороговому напряжению $U_{\text{ПОР}}$ на управляющем электроде – затворе, определяющему границу перехода транзистора из закрытого состояния в проводящее. По этому признаку различают транзисторы с *встроенным и индуцированным* каналами. У транзисторов с встроенным каналом проводимость сток–исток существует при нулевой разности потенциалов между затвором и истоком. Поэтому такой транзистор называют еще *нормально открытым*. В отличие от нормально открытого, транзистор с индуцированным каналом приобретает проводимость лишь тогда, когда управляющее напряжение на затворе превысит величину порога⁹ $U_{\text{ПОР}}$. Этим объясняется и название его как *нормально закрытого*.

Логические операции можно выполнять, соответствующим образом выстраивая сети ключевых транзисторов. Принцип построения таких сетей достаточно прост и не зависит от типа используемых приборов. В конкретных реализациях, естественно, должны быть учтены физические различия в функционировании электронных приборов и составленных из них цепей. Внешне эти различия проявляются прежде всего в действующих токах и напряжениях. Однако на функциональном уровне схемы, с помощью которых выполняются логические операции, можно представить как сочетание простейших последовательных и параллельных соединений ключевых транзисторов. Использование таких соединений для выполнения базовых логических операций И, ИЛИ, И-НЕ и ИЛИ-НЕ в обобщенном виде показано на рис. 2.9.

В положительной логике параллельное соединение транзисторов дает возможность выполнять операции ИЛИ с инверсией (n ИЛИ-НЕ, рис. 2.9а) или без инверсии (n ИЛИ, рис. 2.9б). В отрицательной логике те же элементы выполняют операции n И-НЕ и n И соответственно. Последовательное соединение транзисторов в положительной/отрицательной логике реализует операции n И-НЕ/ n ИЛИ-НЕ (рис. 2.9в) или n И/ n ИЛИ (рис. 2.9г).

⁹ Для транзисторов с разной проводимостью каналов $U_{\text{ПОР}}$ имеет разные знаки.

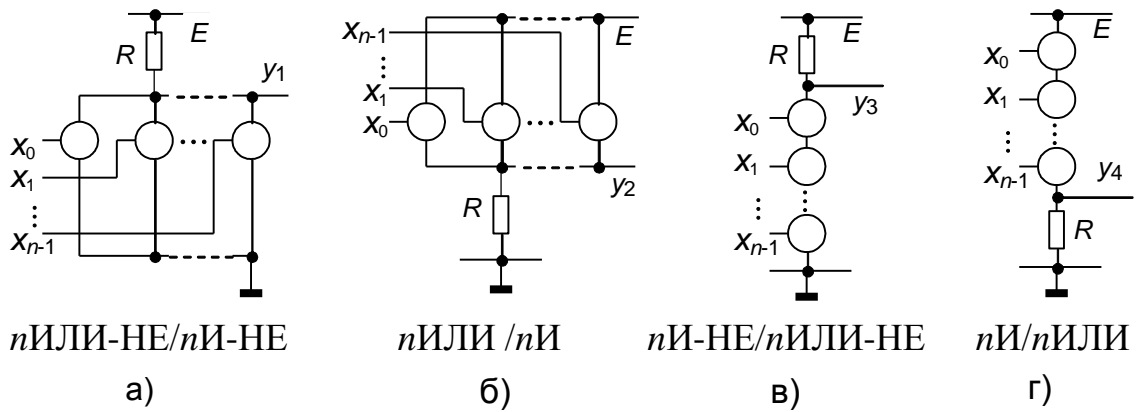


Рис. 2.9. Схема построения логических элементов на ключевых транзисторах

2.3.4. Диодно-транзисторная логика

Как уже упоминалось выше, в общем случае логические элементы в своем составе имеют цепи, отвечающие за выполнение логических операций, и выходные цепи, служащие для согласования выхода одного элемента с входами других. В диодно-транзисторной логике (ДТЛ) для выполнения логических операций используются диодные конъюнкторы или дизъюнкторы, а в качестве согласующих цепей – простые или сложные инверторы. Один из таких элементов (элемент *nИ-НЕ/nИЛИ-НЕ*) представлен на рис. 2.10. В нем логический узел выполнен как диодный конъюнктор, а согласующим (выходным) каскадом является простой инвертор, выполненный по схеме с общим эмиттером. В дальнейшем, чтобы избежать повторений, логические элементы будем классифицировать по операциям в положительной логике, не делая напоминаний о том, как меняется их функция при смене логики на отрицательную.

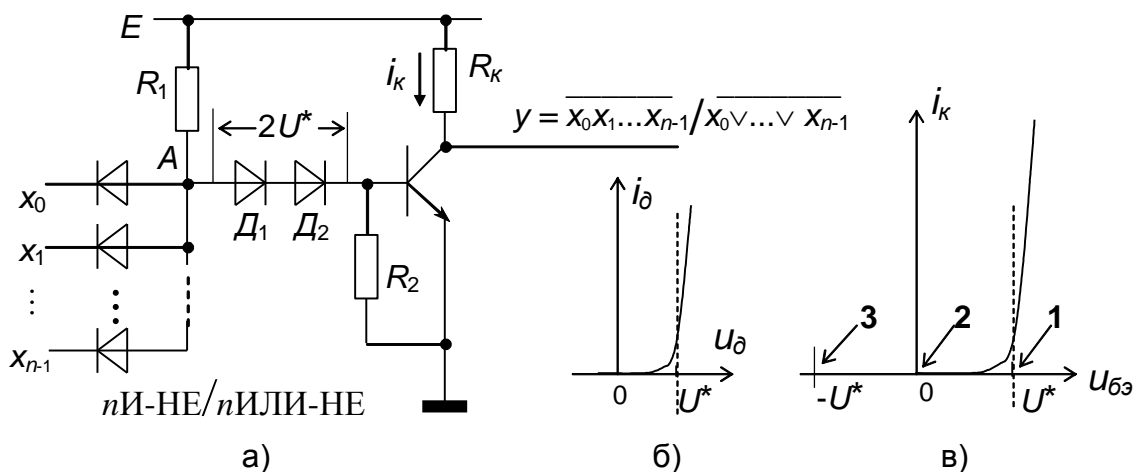


Рис. 2.10. Элемент диодно-транзисторной логики, выполненный по гибридно-интегральной технологии (а) и вольт-амперные характеристики кремниевого диода (б) и кремниевого биполярного транзистора (в)

Если на каком-то одном или нескольких входах логического элемента на рис. 2.10 $x_i = 0$, то один или несколько фазоразделительных диодов будут в проводящем состоянии и напряжение u_A в точке их соединения (рис. 2.10а, точка А) будет равно напряжению на открытом p - n -переходе U^* . Кремниевый p - n -переход отличен тем, что ток диода i_D в прямом направлении начинает заметно расти только тогда, когда напряжение на диоде u_D достигнет величины $u_D = U^*$ (рис. 2.10б). После этого с увеличением u_D ток i_D растет настолько быстро, что напряжение на открытом p - n переходе можно считать почти постоянным и примерно равным U^* ($u_D \approx U^*$). При этом на открытом переходе диапазон изменений u_D незначителен и составляет величину $\Delta u_D \approx (0,1 \dots 0,2)$ В, в то время как $U^* \approx (0,65 \dots 0,7)$ В. Поэтому напряжение на открытом переходе $u_D \approx U^*$ называют еще «напряжением пятки», или просто «пяткой». Подобными свойствами обладают также эмиттерный и коллекторный p - n -переходы кремниевого транзистора. Это значит, что транзистор в открытое состояние переходит лишь тогда, когда управляющее напряжение между базой и эмиттером $u_{бэ} > U^*$.

Помимо развязывающих диодов в схеме логического элемента имеются два дополнительных диода D_1 и D_2 . С их помощью повышается помехозащищенность логического элемента. В подтверждение этого рассмотрим три схемных решения.

1) *Развязывающие диоды соединены с базой транзистора непосредственно.* Помехи наиболее опасны, когда среди всех сигналов x_i есть $x_i = 0$. В этом случае напряжение u_A в общей для диодов точке А, а значит, и разность потенциалов между базой и эмиттером $u_{бэ}$, равны примерно U^* (рис. 2.10в, точка 1). Ток через транзистор минимален (близок к нулю) и плохо контролируем.

2) *Развязывающие диоды соединены с базой через один диод (например, только через D_1).* В этом случае по-прежнему при одном или нескольких $x_i = 0$ $u_A \approx U^*$, но $u_{бэ} \approx u_A - U^* \approx 0$ (рис. 2.10в, точка 2). Транзистор инвертора закрыт.

3) *Развязывающие диоды соединены с базой через два диода (D_1 и D_2).* Надежность закрытого состояния транзистора повышена за счет того, что при $x_i = 0$ $u_{бэ} \approx u_A - 2U^* \approx -U^*$ (рис. 2.10в, точка 3).

Исторически сложилось так, что термин «диодно-транзисторная логика» – ДТЛ используется применительно к одним из самых первых логических элементов, выполненных по гибридно-интегральной технологии. Однако здесь для общности рассмотрения термином ДТЛ будем пользоваться в отношении тех логических элементов, в которых логические операции выполняются с помощью диодов, а функции согласования возложены на транзисторы.

Другой пример построения диодно-транзисторной логики представлен на рис. 2.11. Логический узел этих кремниевых приборов выполнен на диодах Шоттки (диоды D_1 и D_2). Барьер Шоттки имеет более высокую скорость

переключения по сравнению с $p-n$ -переходом, что обеспечивает повышенное быстродействие логического элемента. В выходных каскадах также применяются диоды Шоттки. Подключаются они параллельно коллекторному переходу (транзистор T на рис. 2.11) в наиболее ответственных участках схемы и предохраняют транзистор от попадания в режим глубокого насыщения. Происходит это потому, что при переходе в открытое состояние барьер Шоттки начинает проводить ток раньше, чем коллекторный переход транзисторного ключа. Тем самым сокращается время переключения транзистора из открытого состояния в закрытое и повышается быстродействие логического элемента. Транзисторы с параллельным коллекторному переходу барьером Шоттки называют *транзисторами Шоттки*. В схемах для них используется специальное (такое, как для транзистора T на рис. 2.11) обозначение.

Логический элемент на рис. 2.11 по входам защищен от отрицательных выбросов напряжения соединенными с линией нулевого потенциала (с «землей») диодами Шоттки, что также способствует повышению быстродействия.

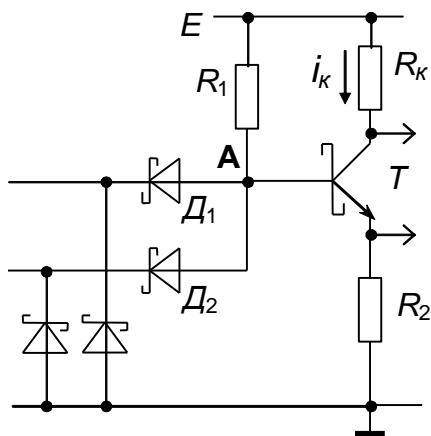


Рис. 2.11. Фрагмент элемента 2И-НЕ/2ИЛИ-НЕ с барьерами Шоттки

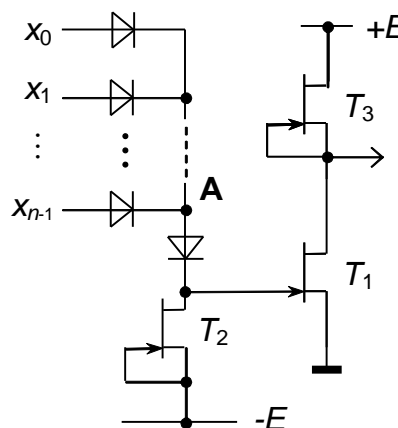


Рис. 2.12. Базовый элемент диодно-полевой логики

По диодно-транзисторной схеме строятся не только кремниевые, но и сверхбыстродействующие GaAs логические элементы. На рис. 2.12 показан базовый элемент *диодно-полевой логики* (ДПЛ), в котором фазоразделительные диоды выполнены как барьер Шоттки; ключевой T_1 и нагрузочные T_2 и T_3 транзисторы – МЭП-транзисторы. Транзисторы T_2 и T_3 , у которых затворы соединены с истоком, выполняют функцию резисторов, но в отличие от резисторов являются не постоянной, а динамической (зависящей от приложенного напряжения) нагрузкой для протекающего через них тока. Замена резисторов на динамическую нагрузку позволяет упростить процесс изготовления ИС элементов логики и понизить вольтность источников питания. Более подробно эти вопросы обсудим при рассмотрении логических элементов на транзисторах с изолированным затвором.

2.3.5. Транзисторно-транзисторная логика

Для логических операций удобно использовать многоэмиттерные и многоколлекторные транзисторы. Рассмотрим сначала группу элементов, в которой логические узлы выполнены как многоэмиттерные транзисторы – это группа *транзисторно-транзисторной логики* (ТТЛ) (рис. 2.13). Принцип выполнения в ней логических операций исходит из диодной логики. Однако если диодные *p-n*-переходы работают автономно, то в многоэмиттерном транзисторе они имеют общую базу, каковой является база транзистора.

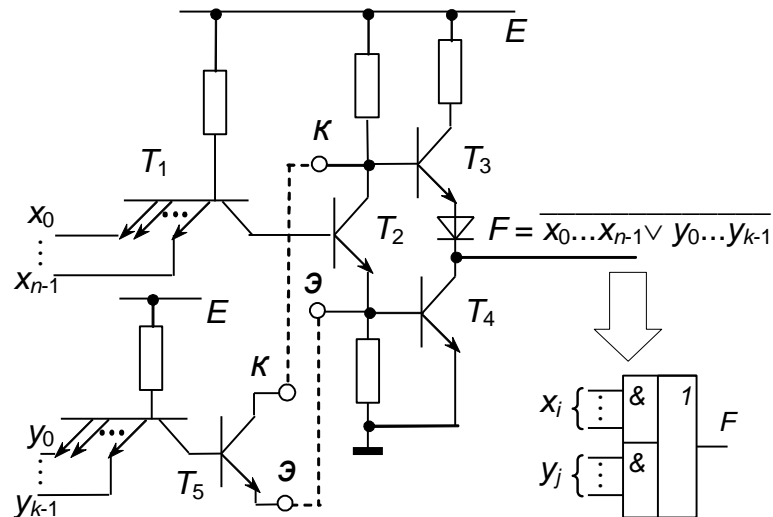


Рис. 2.13. Базовый элемент ТТЛ с расширением по ИЛИ

Если на каком-либо эмиттере транзистора T_1 присутствует низкий уровень напряжения, то соответствующий эмиттерный переход открыт и коллекторный ток транзистора T_1 течет в сторону от базы транзистора T_2 , переводя последний в закрытое состояние. При этом транзистор T_3 открыт, а транзистор T_4 закрыт и на выходе присутствует высокий уровень напряжения. Если на всех входах x_i равны единице, то T_1 оказывается в *инверсном* включении, то есть в состоянии с закрытыми эмиттерными и открытым коллекторным переходами. В этом случае ток от T_1 течет в сторону базы транзистора T_2 и открывает его. При этом на коллекторе T_2 возникает отрицательное напряжение, а на эмиттере – положительное, что приводит к закрыванию T_3 и открыванию T_4 . Как следствие, на выходе логического элемента будет низкий уровень напряжения. Таким образом выполняется логическая операция *nИ-НЕ*.

Диод, стоящий в последней ступени выходного каскада, смещает вниз (к нулю) низкий уровень выходного сигнала, обеспечивая его соответствие уровням входных сигналов в транзисторно-транзисторной логике, что необходимо для соединений одних элементов с другими.

Отметим еще одну особенность ТТЛ. Если в промежуточной секции выходного каскада вместо одного транзистора T_2 применять несколько параллельно соединенных, то дополнительно реализуется операция

логического сложения (операция ИЛИ) так, как это показано на рис. 2.9. Число слагаемых определяется числом параллельных транзисторов. Таким образом получают логические элементы *И с расширением по ИЛИ*. На рис. 2.13 показано расширение с двумя параллельными транзисторами T_2 и T_5 . При каждом из них имеется свой логический узел, и это в совокупности дает операцию И-ИЛИ-НЕ.

Столь удачная реализация операций И/ИЛИ дала начало обширному семейству схем ТТЛ. Появились они в 1968 г., а через 15 лет составляли более 50% общего объема всех цифровых ИС, производимых в мире.

2.3.6. Интегральная инжекционная логика

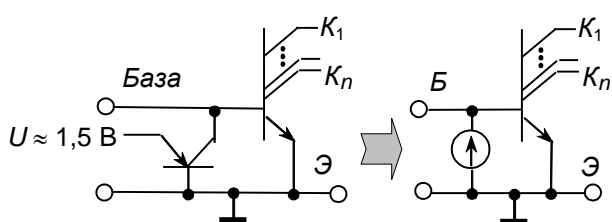


Рис. 2.14. Элементарная ячейка инжекционной логики и ее эквивалентная схема

В интегральной инжекционной логике (И²Л) удачно сочетаются *p-n-p*- и *n-p-n*-транзисторы. Элементарная ячейка инжекционной логики (рис. 2.14) состоит из инжектора (*p-n-p*-транзистор) и многоколлекторного ключа (*n-p-n*-транзистор). Напряжение питания U подается на эмиттер инжектора и составляет около 1,5 В. По

этой причине токи переключения ключевого транзистора относительно невелики, что сказывается на быстродействии элементов И²Л. Такая логика, будучи привлекательной с точки зрения технологического решения, употребляема лишь в низкоскоростных цифровых устройствах.

Логические операции в инжекционной логике превращаются в операции управления током, то есть подключения и отключения различных цепей протекания тока от источников на землю. Реализация логических операций обеспечивается посредством соединения коллектора одного транзистора с базой другого, а также за счет объединения коллекторов. Это иллюстрирует рис. 2.15, на котором приведена схема элемента 3ИЛИ. В зависимости от точки съема выходного сигнала (y или z) выполняется либо операция ИЛИ, либо операция ИЛИ-НЕ. Заметим, что инжекторы в логической схеме, так же, как и ключи, могут быть выполнены в виде многоколлекторных транзисторов, но *p-n-p*-типа.

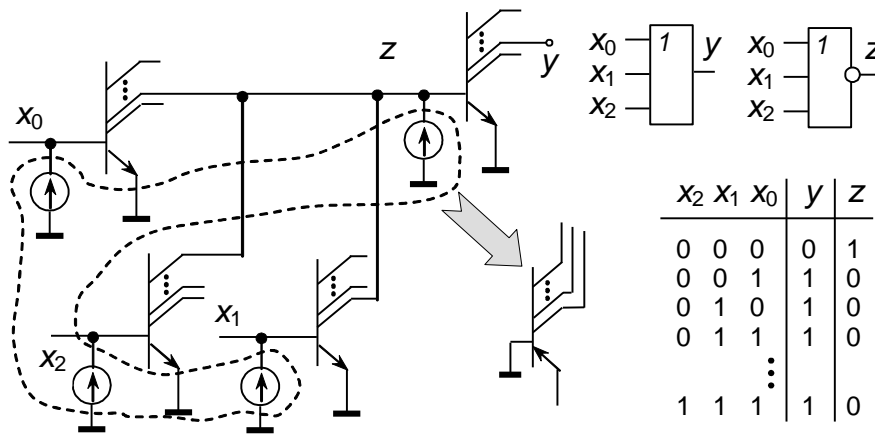


Рис. 2.15. Схема выполнения логических операций в интегральной инжекционной логике

2.3.7. Эмиттерно-связанная логика

Среди логик на биполярных транзисторах эмиттерно-связанная логика (ЭСЛ) является самой быстродействующей. В основе ее работы лежит переключатель тока (рис. 2.16), транзисторы T_1 и T_2 которого работают в активном режиме, непопадая в режим насыщения. Тем самым увеличивается скорость переключения и быстродействие логических элементов.

На базе T_2 присутствует постоянное опорное напряжение $U_{оп}$. Управляющее напряжение подается на базу T_1 . При $u_{вх} = U_{оп}$ оба транзистора открыты и через каждый из них протекает ток $I_0/2$. Потенциалы эмиттеров $u_{э1}$ и $u_{э2}$ одинаковы и равны $u_{э} = U_{оп} - U^*$, что обеспечивает равенство коллекторных ($i_{к1}$ и $i_{к2}$) и эмиттерных токов ($i_{э1}$ и $i_{э2}$):

$$i_{к1} = \alpha i_{э1}; i_{к2} = \alpha i_{э2}; i_{э1} = i_{э2} = I_0/2$$

(рис. 2.16в); $\alpha \approx 1$ – коэффициент передачи тока эмиттера в коллектор.

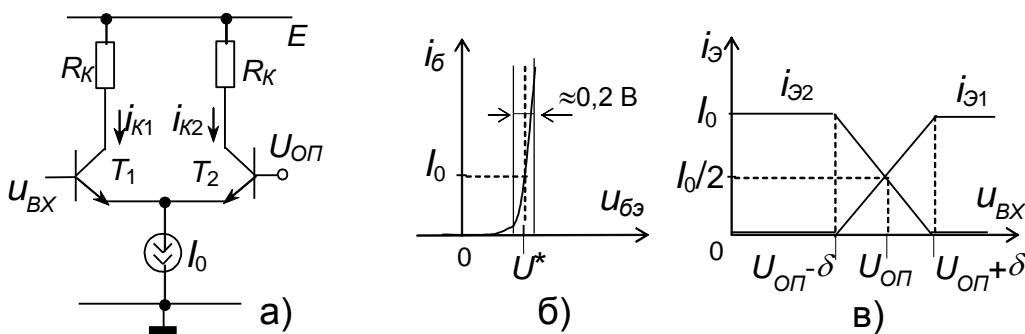


Рис. 2.16. Переключатель тока на дифференциальном каскаде

Уменьшение $u_{вх}$ на величину $\delta \geq 0,1$ В (рис. 2.16) не изменит потенциала эмиттеров $u_{э}$, поскольку его величина $u_{э} = U_{оп} - U^*$ и поддерживается транзистором T_2 . Это вызывает уменьшение разности потенциалов между базой и эмиттером транзистора T_1 на величину δ :

$$u_{бэ1} = U^* - \delta.$$

В результате ток транзистора T_1 в десятки раз уменьшится и будет близок к нулю, в то время как ток транзистора T_2 достигнет максимального значения I_0 :

$$i_{\varepsilon 1} \approx 0, \quad i_{\varepsilon 2} \approx I_0.$$

При повышенном управляющем напряжении $u_{BX} = U_{OП} + \delta$ открывается транзистор T_1 и в этом случае он будет фиксировать потенциал эмиттеров, что переведет T_2 в закрытое состояние:

$$u_{\varepsilon} = U_{OП} - U^* + \delta, \quad u_{БЭ2} = U^* - \delta, \quad i_{\varepsilon 2} \approx 0 \quad \text{и} \quad i_{\varepsilon 1} \approx I_0.$$

Таким образом, перепад потенциала $\Delta u_{Б1} = \pm \delta$ около средней величины $U_{OП}$ обеспечивает переключение тока I_0 от одного транзистора к другому.

Логические операции в базовом элементе ЭСЛ (рис. 2.17) выполняются за счет параллельного соединения транзисторов в управляемом плече

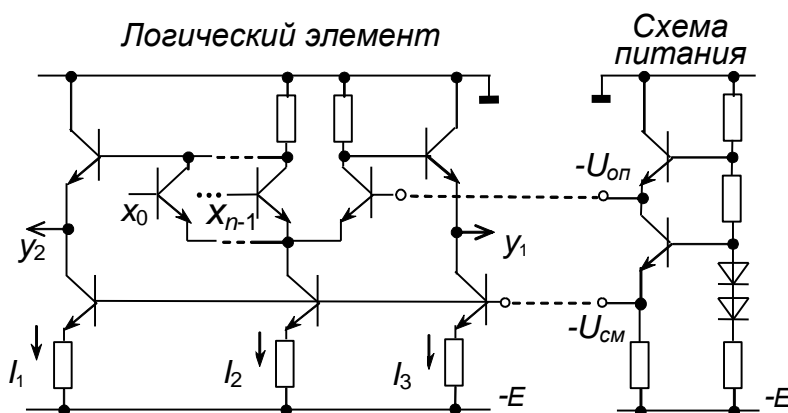


Рис. 2.17. Базовый элемент ЭСЛ

переключателя тока, что дает операцию ИЛИ. Входные сигналы x_0, x_1, \dots, x_{n-1} подаются на базы управляемых транзисторов. Два выходных каскада, выполненные как эмиттерные повторители, дают противофазные сигналы y_1 и y_2 . При этом

$$y_1 = x_{n-1} \vee \dots \vee x_1 \vee x_0 \text{ (нИЛИ)}, \quad y_2 = \overline{x_{n-1} \vee \dots \vee x_1 \vee x_0} \text{ (нИЛИ-НЕ)}.$$

Транзисторы нижнего ряда являются источниками токов I_1, I_2 и I_3 для переключателя и выходных каскадов. На рисунке показана также схема, вырабатывающая опорное напряжение $U_{OП}$ и напряжение смещения $U_{СМ}$ для всех интегрированных в одну ИС логических элементов.

2.3.8. МДП-ключи и логика на МДП-структурах

Основными активными компонентами больших (БИС) и сверхбольших (СБИС) интегральных схем являются полевые транзисторы с изолированным затвором (МДП(МОП)-транзисторы). В основе МДП-логики лежит МДП-ключ, обобщенная схема которого без учета свойств канала приведена на рис. 2.18а.

При работе с ключом должны быть учтены характерные особенности ключевого транзистора, в частности, его *статические проходные* (зависимость $i_c = i_c(u_{зи})$) тока стока i_c от напряжения затвор–исток $u_{зи}$) и *выходные* (семейство кривых $i_c = i_c(u_{си})$) характеристики¹⁰.

Для режима ключа важно значение порогового напряжения $U_{пор}$, поскольку им определяются высокий U^+ и низкий U^- уровни управляющего напряжения $u_{зи}$, переводящих транзистор в открытое/закрытое состояние. У транзисторов с встроенным каналом $U_{пор} < 0$, а у транзисторов с каналом индуцированным $U_{пор} > 0$ (рис. 2.18б). Это отражено и на семействе выходных (рис. 2.18в) характеристик, для которых $u_{зи}$ является параметром:

$$U_{пор} < u_{зи1} < u_{зи2} < u_{зи3} < u_{зи4} < u_{зи5} \dots$$

На выходных характеристиках МДП-транзистора (рис. 2.18в) различимы две области: область 1 левее пунктирной кривой, в которой ток стока заметно растет с ростом напряжения сток–исток, и область 2, где ток i_c определяется в основном управляющим напряжением $u_{зи}$ и слабо зависит от $u_{си}$. В области 1

$$i_c \approx b \cdot \left[(u_{зи} - U_{пор}) \cdot u_{си} - \frac{1}{2} u_{си}^2 \right], \quad (2.3)$$

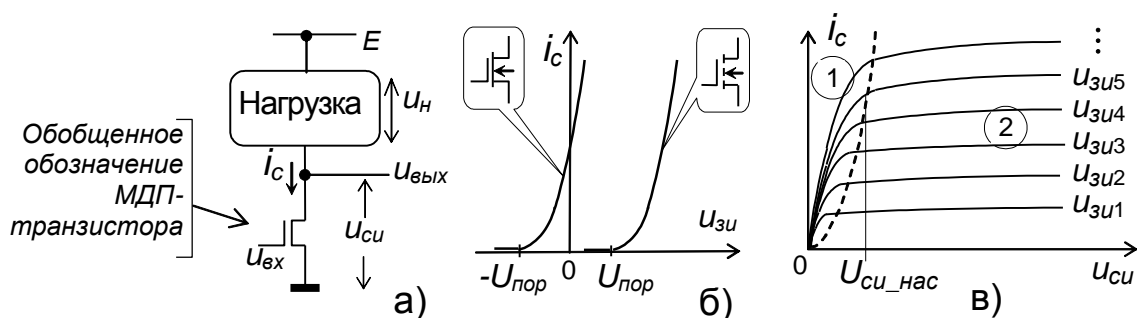


Рис. 2.18. Обобщенная схема МДП-ключа (а) и статические проходные (б) и выходные (в) вольт-амперные характеристики МДП-транзистора

$b = W\mu C_0/L$ – удельная крутизна; W и L – ширина и длина затвора транзистора, μ – подвижность носителей тока (электронов или дырок) в канале, $C_0 = \epsilon_{a_диэл.}/d_{диэл.}$ – удельная емкость затвора, $\epsilon_{a_диэл.}$ – абсолютная диэлектрическая проницаемость, $d_{диэл.}$ – толщина слоя диэлектрика между затвором и каналом и $\epsilon_{a_диэл.} = \epsilon_{диэл.}\epsilon_0$, где $\epsilon_{диэл.}$ – относительная диэлектрическая проницаемость, а ϵ_0 – диэлектрическая проницаемость вакуума.

Уравнение (2.3) справедливо при $u_{си}$, не превышающих *напряжения насыщения* $U_{си_нас}$. Величина $U_{си_нас}$ зависит от управляющего напряжения $u_{зи}$ и равна

¹⁰ Показаны характеристики для транзистора с n -каналом.

$$U_{СИ_НАС} = u_{ЗИ} - U_{ПОР}. \quad (2.4)$$

На рис. 2.18в зависимость $U_{СИ_НАС} = U_{СИ_НАС}(u_{СИ})$ отмечена пунктирной кривой, разделяющей области 1 и 2. В области 1 $u_{СИ} < U_{СИ_НАС} = u_{ЗИ} - U_{ПОР}$. В области 2 $u_{СИ} > U_{СИ_НАС}$ и ток стока определяется его значением при $u_{СИ} = U_{СИ_НАС}$. При этом аналитически проходные и выходные характеристики транзистора можно описать с помощью выражений:

$$i_C \approx i_C(U_{СИ_НАС}) = i_C(u_{ЗИ} - U_{ПОР});$$

$$i_C \approx 0,5b(u_{ЗИ} - U_{ПОР})^2.$$

МДП-ключи различают не только по свойствам ключевого транзистора, но и по типу нагрузки. Простейшим является ключ с резистивной нагрузкой $R_H = R$ (рис. 2.19а). Однако техническое исполнение резистора R требует иных по сравнению с транзисторами технологических операций. Кроме того, на резисторах рассеивается значительная доля энергии источника питания, что неприемлемо для интегральных схем с точки зрения их температурного режима и расходуемой мощности источника питания. Поэтому в ИС используется нелинейная нагрузка, роль которой выполняют транзисторы. Сопротивление транзистора зависит от величины тока в канале, и такая нагрузка является *динамической*. Есть две основные разновидности нелинейной нагрузки – *пассивная* и *активная*. Потенциал затвора в пассивной нагрузке либо фиксирован (рис. 2.19б), либо равен потенциалу истока (рис. 2.19в).

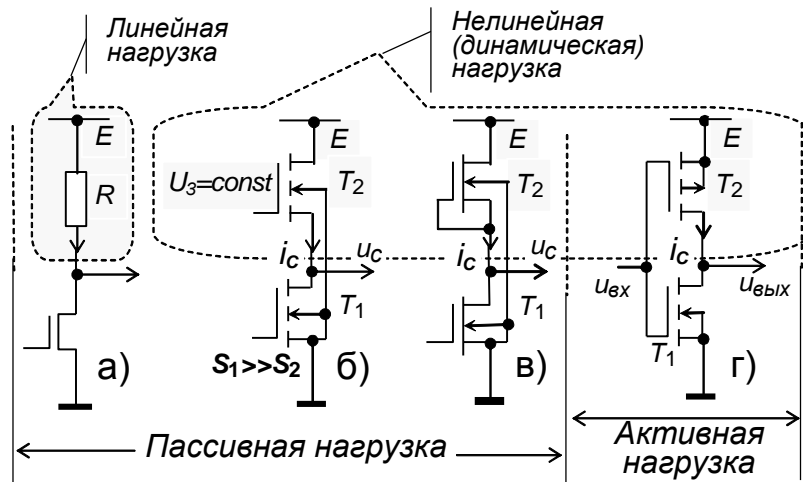


Рис. 2.19. МДП(МОП)-ключи с резистивной нагрузкой (а), на нормально закрытых (б), нормально открытых (в), комплементарных (г) транзисторах

В первом случае (рис. 2.19б) и ключевой T_1 , и нагрузочный T_2 транзисторы являются транзисторами с индуцированным (нормально закрытым) каналом. Как правило, затвор T_2 подключается к источнику питания ($U_3 = E$). Поэтому при высоком уровне потенциала на затворе T_1 открытыми оказываются оба транзистора T_1 и T_2 . Для логических операций необходимо, чтобы низкий уровень напряжения $U^- = U_{ост}$ ($U_{ост}$ – остаточное напряжение на стоке открытого транзистора T_1) был достаточно малым. Это выполняется

тогда, когда внутреннее сопротивление R_{i1} транзистора T_1 меньше R_{i2} – внутреннего сопротивления транзистора T_2 ($R_{i1} < R_{i2}$) или, что то же самое, крутизна T_1 больше крутизны T_2 ($S_1 > S_2$). О такой схеме ключа говорят как о *схеме с отношением*: для транзисторов T_1 и T_2 отношение $S_1/S_2 \neq 1$.

Во втором случае (рис. 2.19в) применяются нормально открытые (с встроенным каналом) транзисторы и для обеспечения проводимости канала не требуется повышенного напряжения на затворе. Поэтому затвор нагрузочного транзистора T_2 соединен с истоком.

В процессе переключения состояние ключа можно определить по его динамическим характеристикам (ДХ), в частности по выходной ДХ

$$u_{си}(i_c) = E - u_H(i_c). \quad (2.5)$$

При линейной пассивной нагрузке $R_H = R$ (рис. 2.19а) ДХ имеет вид прямой линии (линия 1 на рис. 2.20), описываемой уравнением $u_{си}(i_c) = E - i_c R$. При динамической нагрузке зависимость (2.5) более. В открытом ключе, построенном на нормально закрытых транзисторах (рис. 19б), $u_{зи2} > U_{пор.2}$ и напряжение $u_c < U_3 - U_{пор.2}$. Поэтому ток ключа контролируется нагрузочным транзистором и равен

$$i_c = i_{c2} \approx 0,5b_2(u_{зи2} - U_{пор.2})^2 = 0,5b_2(U_3 - u_c - U_{пор.2})^2,$$

поскольку сопротивление R_{i2} канала транзистора T_2 значительно больше сопротивления R_{i1} канала транзистора T_1 . При $U_3 = E$ это соответствует кривой 2 на рис. 2.20. Динамическая характеристика транзистора T_1 в ключе с нормально открытыми транзисторами (рис. 19в) соответствует инвертированной и смещенной на величину напряжения источника питания E по оси напряжений статической характеристике транзистора T_2 при $u_{зи2} = 0$ (кривая 3 на рис. 2.20).

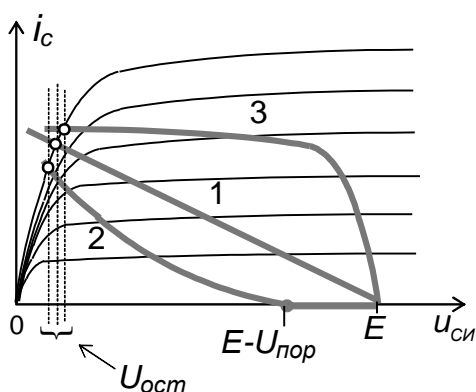


Рис. 2.20. Выходные динамические характеристики МДП-транзистора с резистивной (1) и динамической (2 и 3) нагрузкой (2 – нормально закрытый и 3 – нормально открытый транзисторы)

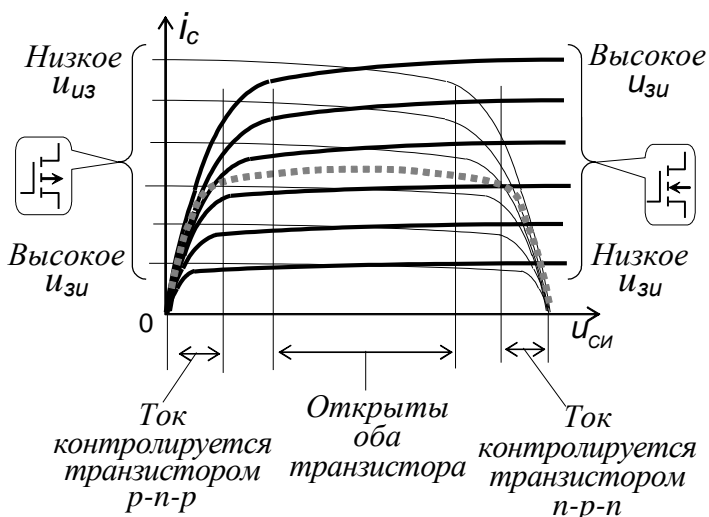


Рис. 2.21. Динамическая характеристика (пунктирная кривая) транзистора с активной нагрузкой на комплементарном транзисторе

Нелинейная активная нагрузка представляет собой комплементарный транзистор (T_2 на рис. 2.19г), управляемый тем же напряжением, что и транзистор T_1 (T_1 и T_2 – это транзисторы с индуцированным каналом). Стоки T_1 и T_2 соединены вместе, а исток T_2 подключен к источнику питания. При таком включении выходные статические характеристики имеют вид, представленный на рис. 2.21: тонкие сплошные линии – это семейство характеристик p -канального транзистора T_2 , а более толстые сплошные линии – семейство характеристик n -канального транзистора T_1 . Динамическая характеристика имеет вид пунктирной кривой.

В статическом режиме ток i_C через комплементарный ключ отсутствует и поэтому применение комплементарных ключей резко сокращает расход энергии источника питания и облегчает тепловой режим микросхем.

Ток комплементарного ключа отличен от нуля только в режиме переключения. Пусть, для примера, состояние ключа меняется под воздействием положительного перепада входного сигнала (от $u_{BX} = U^- = 0$ до $u_{BX} = U^+ = E$). Первоначально T_1 закрыт, а T_2 открыт, и на выходе присутствует высокий уровень напряжения $u_{ВЫХ} = U^+ = E$. При $u_{BX} > U_{ПОР_1}$ ($U_{ПОР_1}$ – пороговое напряжение транзистора T_1) возникает ток i_C , величину которого определяет транзистор T_1 (рис. 2.21). В соответствии с (2.4) напряжение насыщения T_1 , которое определяет ток на пологом участке выходных статических характеристик, $U_{СИ_НАС_1} = u_{ЗИ_1} - U_{ПОР_1}$. На основании (2.3)

$$i_C = i_C(u_{СИ_1}) \approx i_C(U_{СИ_НАС_1}) = 0,5b_1(u_{ЗИ_1} - U_{ПОР_1})^2 = 0,5b_1(u_{ВХ} - U_{ПОР_1})^2.$$

Здесь b_1 , $u_{ЗИ_1}$ и $u_{СИ_1}$ – удельная крутизна, управляющее напряжение затвор-исток и напряжение сток-исток транзистора T_1 .

Дальнейший рост $u_{ВХ}$ перемещает рабочие токи и напряжения обоих транзисторов на пологие участки выходных статических характеристик, где

$$i_C = 0,5b_1(u_{ВХ} - U_{ПОР_1})^2 = 0,5b_2(u_{ЗИ_2} - U_{ПОР_2})^2 = 0,5b_2(u_{ВХ} - E - U_{ПОР_2})^2. \quad (2.6)$$

Здесь b_2 и $u_{ЗИ_2}$ и $U_{ПОР_2}$ – удельная крутизна, управляющее и пороговое напряжения транзистора T_2 . (Напомним, что пороговое напряжение p -канального транзистора $U_{ПОР_2} < 0$.)

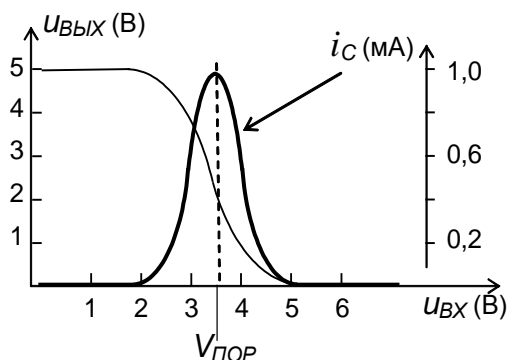


Рис. 2.22. Переключательная характеристика МДП-ключа

При дальнейшем увеличении $u_{ВХ}$ транзистор T_1 продолжает открываться и закрывается транзистор T_2 . Ток ключа вплоть до перехода в новое статическое состояние (T_1 открыт, T_2 закрыт, $u_{ВЫХ} = 0$) будет определяться транзистором T_2 :

$$i_C = 0,5b_2(u_{ВХ} - E - U_{ПОР_2})^2.$$

Переключательная характеристика комплементарного ключа приведена на

рис. 2.22. Порог переключения находится из (2.6) и равен

$$V_{ПОР} = \left[U_{ПОР_1} + \sqrt{\frac{b_2}{b_1}} (E + U_{ПОР_2}) \right] / \left(1 + \sqrt{\frac{b_2}{b_1}} \right).$$

Обычно $b_2/b_1 \approx 1$ и

$$V_{ПОР} \approx 0,5(E + U_{ПОР_1} + U_{ПОР_2}).$$

При $|U_{ПОР_1}| = |U_{ПОР_2}|$ порог переключения $V_{ПОР} \approx 0,5E$.

Описанный процесс смены состояния МДП-ключа не учитывает его инерционности, т. е. параметров транзисторов, которые влияют на скорость переключения, и прежде всего межэлектродных емкостей. Как уже отмечалось, в статическом режиме ток через комплементарный ключ отсутствует. Время пребывания ключа под током зависит от времени (скорости) переключения. Эти обстоятельства важны для СБИС, уровень интеграции которых достигает десятков миллионов вентилях на кристалл.

Логика на МДП-структурах классифицируется по типу используемых ключей – n -канальных (n -МОП, или n -МДП), p -канальных (p -МОП, или p -МДП) и комплементарных (КМОП, или КМДП). Различие транзисторов по типу канала (n - или p -канал, канал встроенный или индуцированный) не вносит принципиальных отличий в схему построения логических элементов – логические операции выполняются на сети ключевых транзисторов, например так, как показано на рис. 2.23.

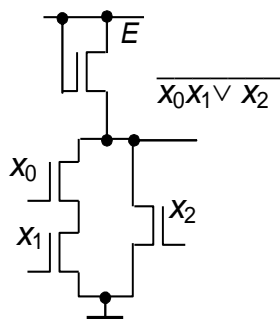


Рис. 2.23. Логический элемент на МОП-транзисторах

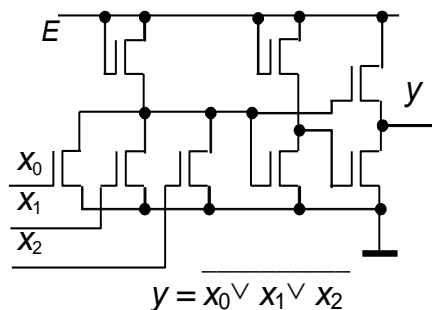


Рис. 2.24. Базовый логический элемент на МОП-транзисторах

Логический узел базового элемента n -МДП- и p -МДП-логики (рис. 2.24) строится на параллельных транзисторах. В каждом элементе имеется согласующий каскад, который состоит из инвертора и выходного ключа на двух управляемых одностипных транзисторах.

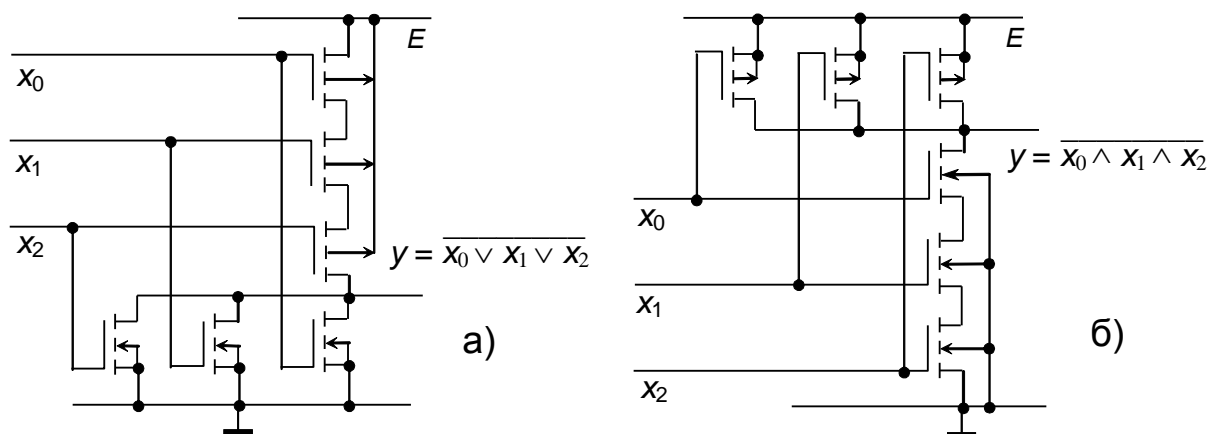


Рис. 2.25. Базовые элементы ЗИЛИ-НЕ (а) и ЗИ-НЕ (б) КМДП-логики

В КМДП-логике (КМДПЛ) также используются параллельно-последовательные соединения ключевых транзисторов. Специфика работы комплементарного ключа такова, что логические элементы на их основе не нуждаются в цепях согласования и разные логические узлы могут быть соединены друг с другом непосредственно. Базовый логический элемент (рис. 2.25) состоит из двух групп транзисторов: n -канальной и p -канальной. В элементе ИЛИ-НЕ (рис. 2.25а) n -канальные транзисторы включены параллельно и в положительной логике выполняют операцию ИЛИ. Ту же операцию выполняет и ряд из последовательных p -канальных транзисторов, т. к. по отношению к ним положительная для n -канальных транзисторов логика становится отрицательной. В элементе И-НЕ (рис. 2.25б), наоборот, n -канальные транзисторы соединены последовательно, а p -канальные параллельно.

2.3.9. Элементы с тремя состояниями

При построении многокомпонентных цифровых систем возникает необходимость в обеспечении бесконфликтного взаимодействия одних устройств в составе системы с другими особенно тогда, когда взаимодействие осуществляется с использованием одних и тех же линий связи. В этом случае источником информации может быть только одно устройство, а другие потенциально возможные источники должны быть в отключенном от линий связи состоянии. Механизмом отключения являются *элементы с тремя состояниями*, переводящие выходы неактивного устройства в высокоимпедансное (третье в дополнение к «0» и «1») состояние. Перевод входов/выходов одних схем в высокоимпедансное состояние дает возможность передавать информацию от других (к другим) по совместно используемым линиям.

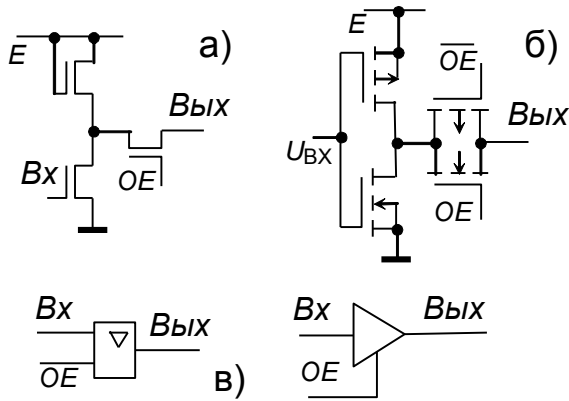


Рис. 2.26. Элементы с тремя состояниями на МДП-транзисторах (а, б) и их условные обозначения (в)

Элемент с тремя состояниями выхода кроме логического состояния 0 и 1 имеет состояние «отключено», в котором ток его выходной цепи пренебрежимо мал (состояние с высоким выходным импедансом). В это (третье) состояние элемент переводится специальным управляющим сигналом (на рис. 2.26 сигналом OE – *Output Enable*). Наиболее просто вентили с тремя состояниями выполняются на МДП-транзисторах. В n -МДП или p -МДП-схемах роль элемента с тремя состояниями может выполнять один передающий транзистор (рис. 2.26а), а в

КМДП-схемах – пара комплементарных транзисторов с противофазными управляющими сигналами OE и \overline{OE} на затворах (рис. 2.26б). Вместо электронного ключа в элементах с тремя состояниями (рис. 2.26а, б) может быть логический узел, построенный так, как это было показано в разделе 2.3.8.

2.3.10. Особенности логических операций на передающих транзисторах. Динамические элементы

Сеть передающих транзисторов может быть использована по-разному, в том числе и для выполнения логических операций. Примером может послужить несложная сеть из двух параллельных ветвей, состоящих из однопольных последовательно соединенных каналов, для операции «исключающее ИЛИ» (рис. 26а). Подобную функцию можно выполнить и на передающих комплементарных транзисторах (рис. 2.26б).

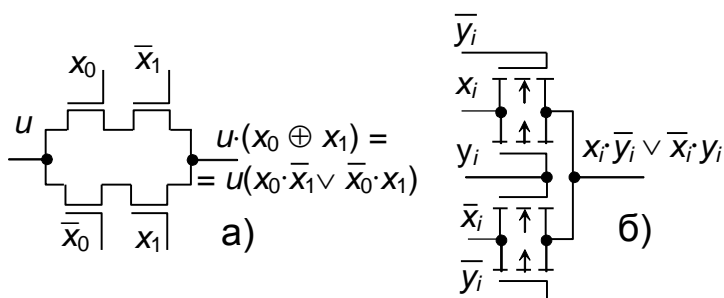


Рис. 2.26. Пример логических операций с помощью передающих МДП-транзисторов

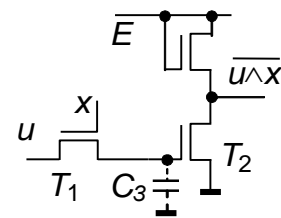


Рис. 2.27. Простейший динамический элемент на передающем и ключевом МДП-транзисторах

Другой способ реализации логических функций состоит в сочетании ключевых и передающих транзисторов. Однако в этом случае сеть приобретает новое качество, т. к. логические функции по-разному выполняются передающими и ключевыми транзисторами. В первом случае значение функции

зависит от возможности протекания тока, тогда как во втором – от уровня напряжения. Кроме того, потенциал затвора ключевого транзистора зависит от накопленного заряда и может сохраняться в отсутствие тока.

Пусть, например, ток к затвору подводится через передающий транзистор T_1 (рис. 2.27), но затем цепь для протекания тока разрывается. Это ведет к изоляции затвора ключевого транзистора T_2 , и заряд на нем будет существовать некоторое время, зависящее от скорости разряда емкости затвора C_3 через сопротивление утечки R_V . Время разряда $\tau_p = C_3 R_V$ составляет сотни миллисекунд, что несопоставимо с временем переключения τ_{II} (доли или несколько наносекунд). При достаточно частых (с частотой $f_{II} \gg 1/\tau_p$) переключениях емкость затвора может выполнять функцию задержки или *динамического элемента памяти*. В простейшем случае динамический элемент состоит из передающего транзистора и инвертора (рис. 2.27).

Двухфазные динамические элементы

Рассмотрим подробнее работу элемента, представленного на рис. 2.27. Для этого переставим местами инвертор и передающий транзистор (рис. 2.28). Элементы памяти представлены емкостями затвор–исток C_{31} и C_{32} передающего T_3 и следующего за ним (не показанного на рисунке) ключевого транзистора, являющегося нагрузкой T_3 . На затворы нагрузочного транзистора T_2 и передающего транзистора T_3 подается последовательность синхроимпульсов C_1 . Работа элемента иллюстрируется временными диаграммами на рис. 2.28в.

При возрастании потенциала U_{BX} на входе D выше порога переключения $V_{II} = U_{ПОР}$, где $U_{ПОР}$ – пороговое напряжение МДП-транзистора, открывается транзистор T_1 и емкость C_{31} разряжается до низкого потенциала $U_A \approx 0$. При поступлении синхроимпульса амплитудой $U_C > U_{ПОР}$ открываются транзисторы T_2 и T_3 и на выходе Q устанавливается низкий потенциал $U^0 = U_{ОСТ_1} + U_{ОСТ_3}$, где $U_{ОСТ}$ – остаточное напряжение между стоком и истоком T_1 и T_3 , работающих в крутой области (область 1 на рис. 2.18в) стоковых характеристик, которые описываются выражением (2.3). При этом нагрузочный транзистор T_2 работает в пологой области (область 2 на рис. 2.18в) стоковых характеристик и для него ток стока, а значит, и ток инвертора определяется значением $i_C \approx i_C(u_{СИ_2})$ при $u_{СИ_2} = U_{СИ_НАС}$. В силу этого

$$i_C \approx 0,5b_2 \cdot (U_C - U_{ПОР_2})^2. \quad (2.6)$$

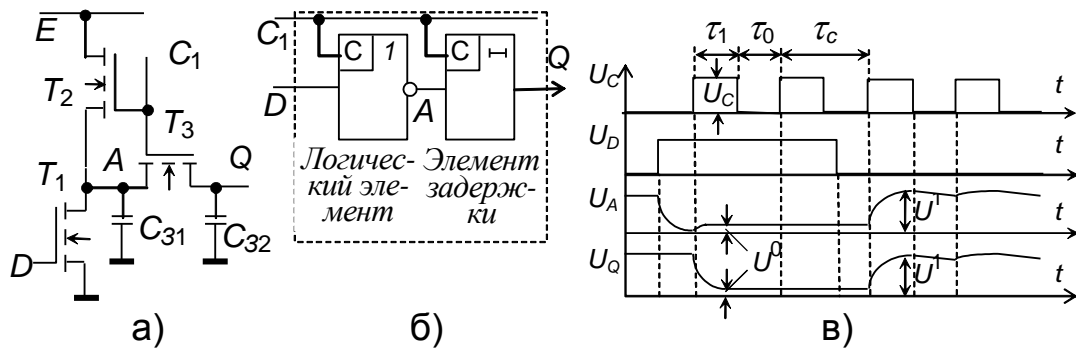


Рис. 2.28. Схема двухфазного динамического элемента «с отношением» (а), его структура (б) и временные диаграммы (в)

Так как через транзистор T_3 ток не протекает, то $U_{OCT_3} \approx 0$. Поэтому с учетом (2.3), (2.6) и того, что для открытого транзистора T_1 $i_{СИ_1} \ll U_{ПОР_1}$ и $i_{ЗИ_1} = U^1$, а для открытого транзистора T_2 $i_{ЗИ_2} \approx U_C$, получим

$$U^0 = U_{OCT_1} = i_C r_{СИ_1} = 0,5b_2(U_C - U_{ПОР_2})^2/b_1(U^1 - U_{ПОР_1}). \quad (2.7)$$

Здесь b_1, b_2 – удельная крутизна транзисторов T_1 и T_2 , $U_{ПОР_2} = U_{ПОР_1} = U_{ПОР}$, U^1 – высокий, соответствующий логической «1», уровень напряжения. По окончании синхроимпульса, когда устанавливается потенциал $U_C = 0$, транзисторы T_2 и T_3 запираются. Заряд на емкости C_{32} и соответственно низкий уровень потенциала U^0 на выходе Q будут сохраняться в течение времени хранения t_{XP}^0 , длительность которого определяется процессом заряда емкости C_{32} токами утечки: $t_3 = R_V C_{32}$, где R_V – сопротивление утечки. При типовых значениях $C_{32} = 0,1 \dots 1$ пФ, $R_V = 10^8 \dots 10^9$ Ом получим $t_3 = 10 \dots 1000$ мкс. Чтобы не произошло потери информации, т. е. ложного переключения элемента, необходимо подать новый синхроимпульс C_1 раньше, чем потенциал $U_{ВЫХ}$ вследствие заряда C_{32} возрастет до уровня $U_{ПОР}$, т. е. спустя время

$$\tau_0 < t_{XP}^0 \approx t_3 \cdot (U_{ПОР}/E). \quad (2.8)$$

Очередной синхроимпульс, открывая T_2 и T_3 , обеспечивает разряд C_{32} до потенциала $U_{ВЫХ} = U^0$ и восстанавливает информацию.

При уменьшении потенциала на входе D до низкого уровня $U_{ВХ} = U^0 < U_{ПОР}$ транзистор T_1 запирается. При поступлении синхроимпульса C_1 через открывающиеся транзисторы T_2 и T_3 емкости C_{31} и C_{32} заряжаются до высокого потенциала $U_{ВЫХ} = U^1 = U_C - U_{ПОР}$. Если синхроимпульс имеет достаточно большую амплитуду $U_C > E + U_{ПОР}$, то выходное напряжение достигает величины $U^1 = E$. По окончании синхроимпульса T_2 и T_3 запираются и потенциал на выходе поддерживается в течение времени t_{XP}^1 , пока емкость C_{32} сохраняет достаточный заряд. Поэтому необходимо обеспечить выполнение соотношения

$$\tau_0 < t_{XP}^1 \approx t_3 \cdot (U^1 - U_{ПОР})/U^1. \quad (2.9)$$

Следующий импульс C_1 открывает транзисторы T_2 и T_3 и подзаряжает емкости до уровня U^1 . При выполнении (2.8) и (2.9) динамический элемент ведет себя как синхронизируемый бистабильный элемент – триггер. Имеются два режима его работы: при $C_1 = 1$ – прием (запись) информации (с инверсией), при $C_1 = 0$ – ее хранение.

Для обеспечения достаточной помехоустойчивости динамических элементов следует уменьшить b_2/b_1 – отношение удельной крутизны нагрузочного и управляющего транзисторов, чтобы выполнялось условие $U^0 < U_{ПОР} - U_{П}$, где $U_{П}$ – требуемое значение помехоустойчивости. Используя (2.7), получаем следующее ограничение на величину b_2/b_1 :

$$b_2/b_1 \leq 2(U^1 - U_{ПОР})(U_{ПОР} - U_{П})/(U_C - U_{ПОР})^2.$$

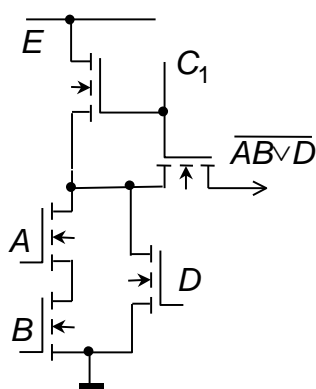


Рис. 2.29. Двухфазный динамический элемент И-ИЛИ-НЕ

При типовых значениях U^1 , U_C и $U_{ПОР}$ получаем следующее требование: $b_2/b_1 \leq 0,1$. На практике это достигается использованием МДП-транзисторов с различными геометрическими размерами. Так как удельная крутизна пропорциональна отношению ширины канала к его длине (W/L), то транзистор T_1 изготавливается с коротким и широким каналом, а транзистор T_2 – с более длинным и узким.

Используя параллельное и последовательное включение управляющих транзисторов T_1 , подобно тому, как это показано на рис. 2.23 и 2.24, получаем динамические элементы, выполняющие заданную логическую операцию, например, И-ИЛИ-НЕ (см. рис. 2.29).

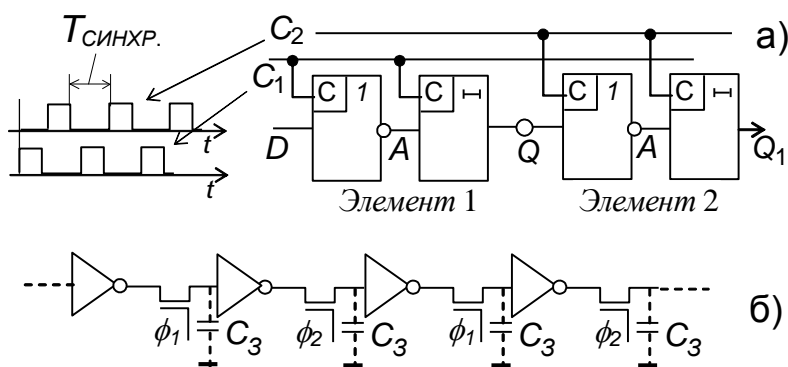


Рис. 2.30. Структура динамического триггера (а) и последовательная цепь динамических элементов (б)

При соединении динамических элементов (рис. 2.30) необходимо в последовательно включенных элементах (инвертор плюс передающий канал) разделить во времени фазы приема и хранения информации. Для этого используются две последовательности синхроимпульсов C_1 и C_2 . Каждый такт работы устройства, соответствующий одному периоду синхроимпульсов $T_{СИНХР.}$, состоит из двух фаз (ϕ_1 и ϕ_2). В одной фазе происходит переключение

элементов 1 (импульсами C_1) при сохранении состояния элементов 2. Во второй фазе импульсами C_2 переключаются элементы 2, а элементы 1 находятся в режиме хранения.

В рассмотренном динамическом элементе предъявляются противоречивые требования к параметрам транзисторов T_1 и T_2 . Во многих случаях предпочтительнее использовать динамические элементы «без отношения», параметры которых не зависят от отношения b_2/b_1 .

Двухфазные динамические элементы «без отношения»

Вариант динамического элемента «без отношения» показан на рис. 2.31. Рассмотрим работу двух последовательно соединенных элементов этого типа, используя временные диаграммы на рис. 2.31б. Пусть в исходном положении на вход D подан низкий потенциал $U_D \approx 0$. Тогда на емкостях C_{31} , C_{32} , C_{34} низкий потенциал $U_A = U_{Q1} = U_{Q2} = 0$, а на C_{33} – высокий потенциал $U_B = U^1$. Если на вход D поступит высокий потенциал $U_D = U^1 > U_{ПОР}$, то при подаче синхроимпульса C_1 (фаза ϕ_1) транзистор T_1 откроется, емкость C_{31} зарядится до $U_A \approx U^1$. Транзисторы T_2 и T_3 открываются, и емкость C_{32} заряжается до потенциала $U_{Q1} = U^1 = U_C - U_{ПОР}$.

После окончания синхроимпульса C_1 емкость C_{32} через открытый транзистор T_2 разряжается до $U_{Q1} \approx 0$. При поступлении синхроимпульса C_2 (фаза ϕ_2) открываются транзисторы T_4 и T_6 . Емкость C_{33} через открытые T_2 и T_4 разряжается до $U_B \approx 0$, и транзистор T_5 запирается. В дальнейшем при неизменном $U_D = U^1$ потенциал на емкостях C_{31} и C_{34} сохраняется высоким, а потенциал C_{33} – низким. Емкость C_{32} при поступлении каждого синхроимпульса C_1 заряжается до $U_{Q1} = U^1$, а после его окончания разряжается до $U_{Q1} = 0$. Низкий потенциал $U_B = 0$ сохраняется на емкости C_{33} , и транзистор T_5 остается закрытым. При поступлении импульса C_2 емкость C_{34} заряжается до $U_{Q2} = U^1$. При поступлении импульса C_1 заряд емкости C_{34} распределяется между ней и емкостью C_{35} в элементе-нагрузке. На выходе Q_2 устанавливается уровень U_{min}^1 , который должен быть достаточным для отпираания следующего МДП-транзистора:

$$U_{min}^1 = (U_C - U_{ПОР}) C_{34} / (C_{34} + C_{35}) \geq U_{ПОР}. \quad (2.10)$$

Выполнение соотношения (2.10) достигается соответствующим выбором амплитуды синхроимпульсов U_C . Для снижения требуемой величины U_C топология элемента проектируется так, чтобы обеспечить достаточно большое отношение $C_{34}/C_{35} = C_{32}/C_{33} \geq U_{ПОР} / (U_C - 2U_{ПОР})$.

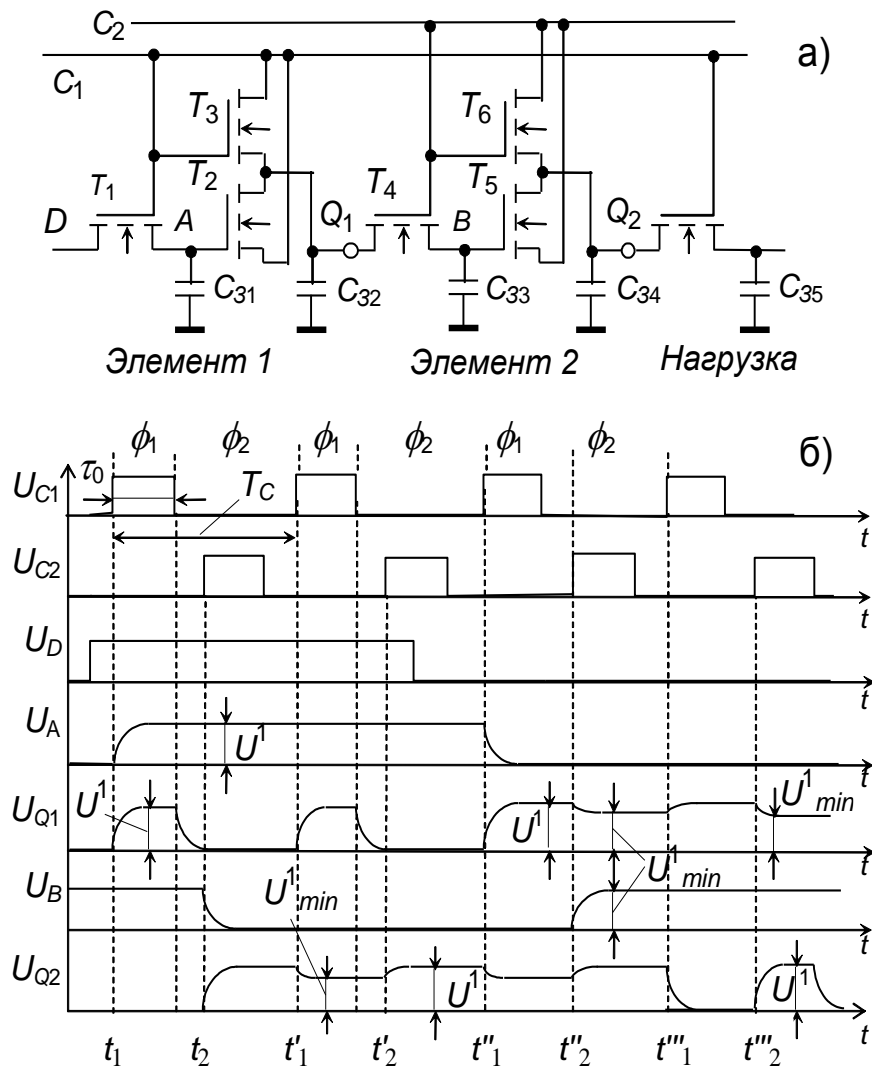


Рис. 2.31. Схема динамического триггера на двухфазных элементах «без отношения» (а) и его временные диаграммы (б)

После установления на входе потенциала $U_D \approx 0$ и поступления синхроимпульса C_1 (фаза ϕ_1) емкость C_{31} разряжается через транзистор T_1 до $U_A \approx 0$, и транзистор T_2 запирается. Емкость C_{32} через открытый транзистор T_3 заряжается до потенциала $U_{Q1} = U^1$, который сохраняется после окончания синхроимпульса C_1 . При поступлении синхроимпульса C_2 (фаза ϕ_2) открывается транзистор T_4 и заряд емкости C_{32} распределяется между емкостями C_{32} и C_{33} . В результате на емкостях C_{32} и C_{33} устанавливается потенциал $U_{Q1} \approx U_B \approx U^{1}_{min}$. На выходе второго элемента Q_2 формируется последовательность импульсов амплитудой U^1 , каждый из которых образуется при поступлении импульса C_2 . Аналогично работают элементы и в том случае, когда на вход вместо постоянного потенциала $U_D = 0$ поступает последовательность импульсов, синхронизированных с синхроимпульсами C_2 .

Таким образом, значение $U_Q = U^0$ на выходе динамических элементов поддерживается только во время паузы между синхроимпульсами соответствующей последовательности (C_1 или C_2), а в течение действия

синхроимпульса $U_Q = U^1$. Однако вследствие двухфазной синхронизации последующий элемент воспринимает только истинное значение, соответствующее сигналу на входе предыдущего элемента во время паузы между синхронизирующими импульсами. Возникновение на выходе «ложных» импульсов амплитудой U^1 не совпадает во времени с режимом записи информации в последующий элемент и поэтому не влияет на работу устройства.

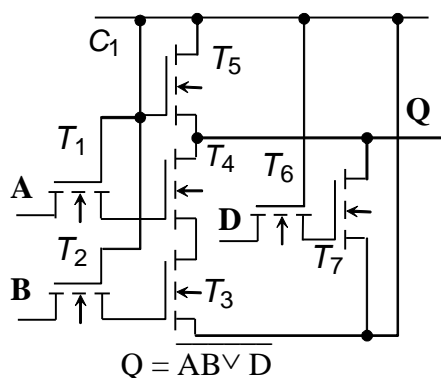


Рис. 2.32. Схема двухфазного динамического элемента И-ИЛИ-НЕ «без отношения»

Логические функции на элементах «без отношения» реализуются так же, как и на элементах с отношением, комбинацией последовательного и параллельного включений передающих транзисторов с подключением их к затворам ключевых транзисторов (рис. 2.32).

Рассмотренные примеры динамических элементов не исчерпывают вариантов их построения и приведены в основном для иллюстрации тех функциональных возможностей, которые приобретаются при совместном использовании передающих и ключевых транзисторов. К таким возможностям относится использование емкости затвора МДП-транзистора в качестве элемента памяти, что имеет большое значение при создании СБИС.

3. Функциональные узлы комбинационного типа

Функциональные узлы выполняют типовые для цифровых устройств (ЦУ) *микрооперации*. Микрооперации соответствуют низшему иерархическому уровню внутреннего языка цифрового устройства, они обозначены в этом языке и не содержат других операций, кроме обозначенных в нем. Как и все цифровые устройства вообще, *функциональные узлы делятся на комбинационные и последовательные*. В дальнейшем комбинационные узлы будем обозначать через КЦ (комбинационные цепи), а последовательные – через АП (автоматы с памятью). *Различия между КЦ и АП имеют фундаментальный характер*.

Выходные величины КЦ зависят только от текущего значения входных величин (аргументов). Предыстория не имеет значения. После завершения переходных процессов в КЦ на их выходах устанавливаются величины, на которые характер переходных процессов влияния не оказывает. С этой точки зрения переходные процессы в КЦ не опасны. Но в ЦУ в целом КЦ функционируют совместно с АП, что кардинально меняет ситуацию. *Во время переходных процессов на выходах КЦ появляются временные сигналы, не предусмотренные описанием работы КЦ и называемые рисками*. Со временем они исчезают и выход КЦ приобретает значение, предусмотренное логической формулой, описывающей работу цепи. Однако риски могут быть восприняты элементами памяти АП, необходимое изменение состояния которых может радикально изменить работу ЦУ. Примером служат рассмотренные выше динамические элементы, для работы которых важна синхронизация.

Некоторые из устройств комбинационного типа (полный дешифратор, демультимплексор и мультимплексор) были рассмотрены ранее (см. начало раздела 2). Причиной послужило то, что с помощью этих КЦ реализуется СДНФ логических выражений. Как продолжение рассмотрим комбинационные цепи, предназначением которых является выполнение логических функций, свойственных устройствам, входящим в состав цифровых вычислительных систем.

3.1. Арифметические устройства

3.1.1. Двоичные сумматоры

Операция суммирования – это одна из наиболее часто встречающихся операций в цифровой технике. Пусть слагаемыми являются два n -разрядных двоичных числа

$$\begin{aligned} X &= (x_{n-1} \dots x_1 x_0) = x_{n-1} 2^{n-1} + x_{n-2} 2^{n-2} + x_1 2^1 + x_0 2^0 \quad \text{и} \\ Y &= (y_{n-1} \dots y_1 y_0) = y_{n-1} 2^{n-1} + y_{n-2} 2^{n-2} + y_1 2^1 + y_0 2^0. \end{aligned}$$

6	5	4	3	2	1	– номер разряда
0	1	1	0	0	1	– $X = 25$
0	1	1	1	0	1	– $Y = 29$
1	1	0	0	1	0	– перенос (c)
1	1	0	1	1	0	– $S = 54$ (сумма)

Рис. 3.1

Определим алгоритм, по которому будет осуществляться операция суммирования. Сделаем это на примере, когда $X = 25$, $Y = 29$, а $n = 6$ (рис. 3.1). Суммируются все разряды, начиная с младшего. Правило суммирования заключается в следующем:

1) перенос в $(i+1)$ -й (следующий)

разряд из i -го (предыдущего) $c_{i+1} = 1$ (истина), если две или три величины из a_i , b_i и c_i истинны (равны 1), т.е.

$$c_{i+1} = (x_i y_i c_i \vee x_i y_i \bar{c}_i) \vee (x_i \bar{y}_i c_i \vee \bar{x}_i y_i c_i) = x_i y_i \vee (x_i \oplus y_i) \cdot c_i. \quad (3.1)$$

Знак \oplus обозначает операцию

$$x_i \oplus y_i = x_i \bar{y}_i \vee \bar{x}_i y_i,$$

называемую *сумма по модулю 2* или *исключающее ИЛИ*. Из (3.1) получаются более простые выражения для формирования переноса c_{i+1} :

$$\begin{aligned} c_{i+1} &= x_i y_i \vee x_i c_i \vee y_i c_i = \quad (\text{в базисе И-ИЛИ}) \\ &= \overline{x_i y_i \cdot x_i c_i \cdot y_i c_i} \quad (\text{в базисе И-НЕ}); \end{aligned} \quad (3.2)$$

2) значение i -го разряда суммы $s_i = 1$ (истинно), если истинно (равно 1) нечетное число величин x_i , y_i и c_i :

$$s_i = (x_i \oplus y_i) \oplus c_i = x_i \oplus y_i \oplus c_i, \quad (3.3)$$

или

$$\begin{aligned} s_i &= x_i y_i c_i \vee \bar{x}_i \bar{y}_i c_i \vee x_i \bar{y}_i \bar{c}_i \vee \bar{x}_i y_i \bar{c}_i = \quad (\text{в базисе И-ИЛИ}) \\ &= \overline{x_i y_i c_i \cdot \bar{x}_i \bar{y}_i c_i \cdot x_i \bar{y}_i \bar{c}_i \cdot \bar{x}_i y_i \bar{c}_i} \quad (\text{в базисе И-НЕ}); \end{aligned} \quad (3.4)$$

3) значение переноса в младший ($i = 0$) разряд $c_0 = 0$.

Одноразрядный сумматор имеет три входа (два слагаемых и перенос из предыдущего разряда) и два выхода (сумма и перенос в следующий разряд).

Непосредственное воспроизведение выражений (3.2) и (3.4) для сумматора в базисе И-НЕ приводит к применению трех элементов 2И-НЕ, пяти элементов 3И-НЕ и одного элемента 4И-НЕ (рис. 3.2).

Логiku работы сумматора можно построить на основе таких сочетаний значений x_i , y_i и c_i , при которых сумма и перенос будут равны ложны:

$$\begin{aligned} \bar{s}_i &= \bar{x}_i c_{i+1} \vee \bar{y}_i c_{i+1} \vee \bar{c}_i c_{i+1} \vee \bar{x}_i \bar{y}_i \bar{c}_i, \\ \bar{c}_{i+1} &= \bar{x}_i \bar{y}_i \vee \bar{x}_i \bar{c}_i \vee \bar{y}_i \bar{c}_i, \end{aligned} \quad (3.5)$$

Соответствующая (3.5) логическая схема сумматора в базисе И-ИЛИ-НЕ представлена на рис. 3.3.

Выбор логики работы сумматора зависит от того, какая элементная база используется для реализации логических операций. В качестве примеров схемно-ориентированной логики рассмотрим формирование суммы и образование переноса в транзисторно-транзисторной логике и логике на комплементарных транзисторах.

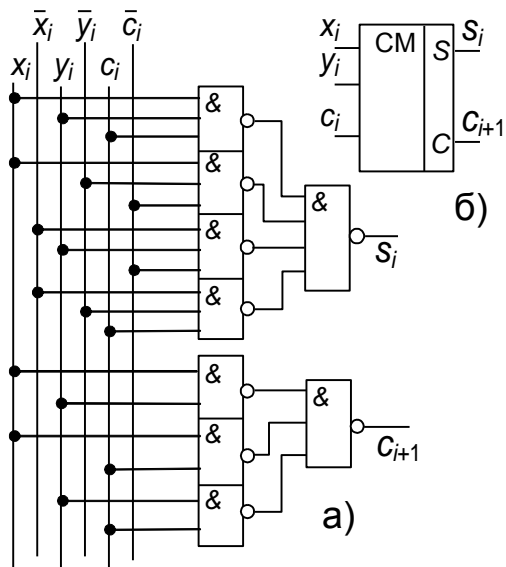


Рис. 3.2. Логическая схема в базисе И-НЕ (а) и условное обозначение (б) одноразрядного сумматора

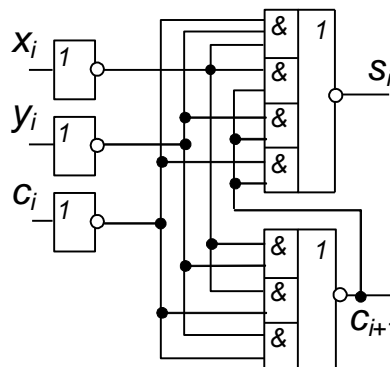


Рис. 3.3. Одноразрядный сумматор в базисе И-ИЛИ-НЕ

Для первого случая удобно воспользоваться выражениями

$$s_i = \overline{(x_i y_i c_i)} \cdot c_{i+1} \vee (\overline{x_i} \overline{y_i} \overline{c_i}) \quad (3.6)$$

и

$$\overline{c_{i+1}} = \overline{x_i} \overline{y_i} \vee \overline{x_i} \overline{c_i} \vee \overline{y_i} \overline{c_i}, \quad (3.7)$$

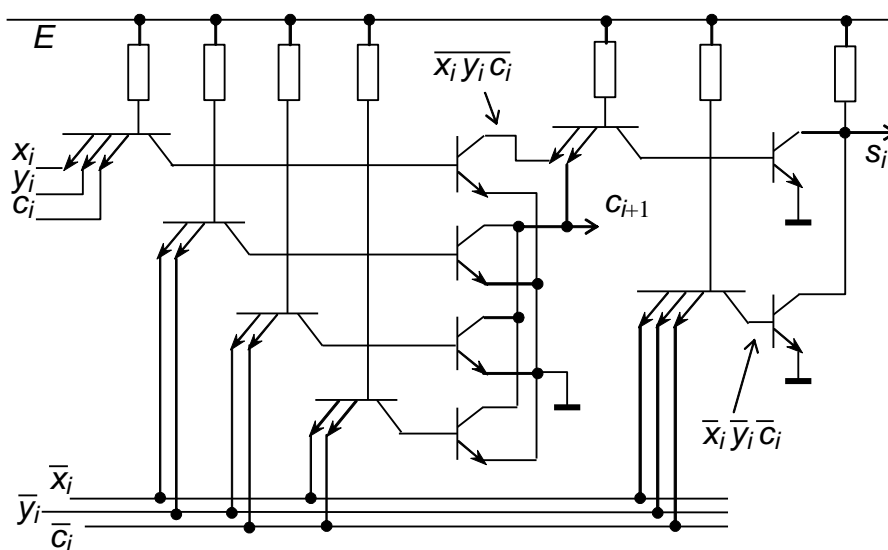


Рис. 3.4. Схемная реализация одноразрядного сумматора на основе ТТЛ

которые можно получить из (3.5), применив теорему де Моргана. Логические выражения (3.6) и (3.7) реализуются с помощью многоэмиттерных и параллельно включенных биполярных транзисторов (рис. 3.4)

Выражение (3.3) для суммы можно преобразовать к виду

$$\bar{s}_i = \overline{(x_i \oplus y_i) \cdot c_i} \vee (x_i \oplus y_i) \cdot c_i, \quad (3.8)$$

выразив операцию *исключающее ИЛИ* (символ \oplus) через функцию *равнозначности* (символ \sim) в соответствии с выражением

$$x \oplus y = x\bar{y} \vee \bar{x}y = \overline{\overline{x\bar{y}} \cdot \overline{\bar{x}y}} = \overline{(\bar{x} \vee y) \cdot (x \vee \bar{y})} = \overline{x \cdot y \vee \bar{x} \cdot \bar{y}} = x \sim y.$$

Такая логика суммирования подходит для реализации на комплементарных транзисторах. При этом перенос формируется в соответствии (3.1), т. е.

$$c_{i+1} = x_i y_i \vee (x_i \oplus y_i) \cdot c_i. \quad (3.9)$$

Для выполнения функций (3.8) и (3.9) удобно применить мультиплексоры 2×1 (рис. 3.5) – два мультиплексора на комплементарных (*p*- и *n*-канальных) передающих транзисторах для формирования суммы и мультиплексор на ключевых транзисторах для формирования переноса.

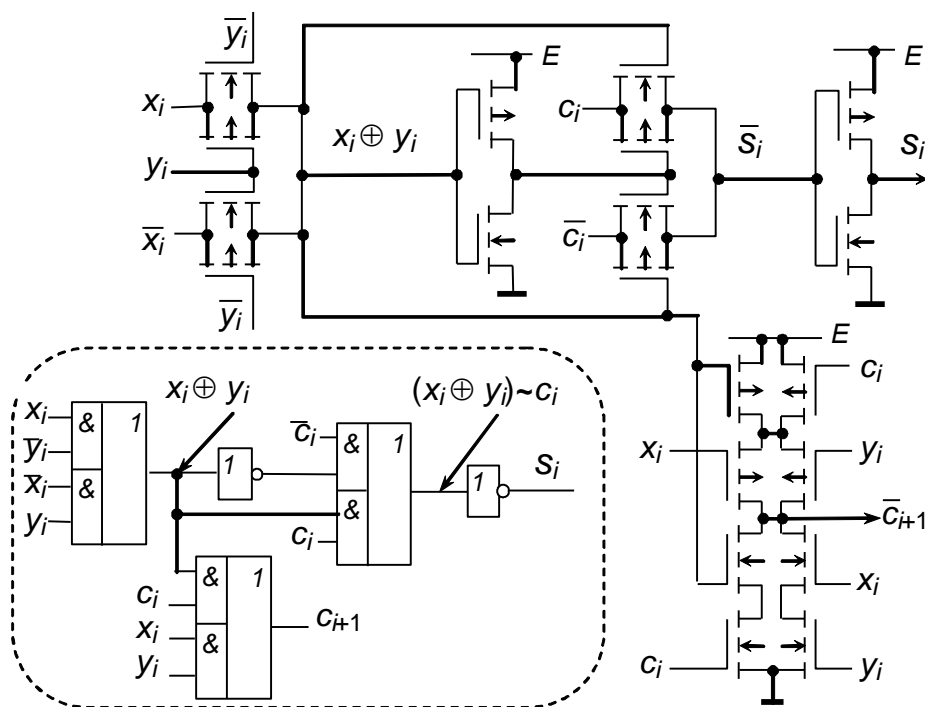


Рис. 3.5. Логическая и электрическая схемы одноразрядного сумматора на КМДП-транзисторах

Многоразрядные сумматоры делятся на последовательные и параллельные. Сумматор последовательных операндов содержит один одноразрядный сумматор и поочередно обрабатывает разряд за разрядом, начиная с младшего. При этом процедура суммирования требуется

запоминания переноса и занимает достаточно много времени. Поэтому такие сумматоры применимы лишь в относительно медленных устройствах.

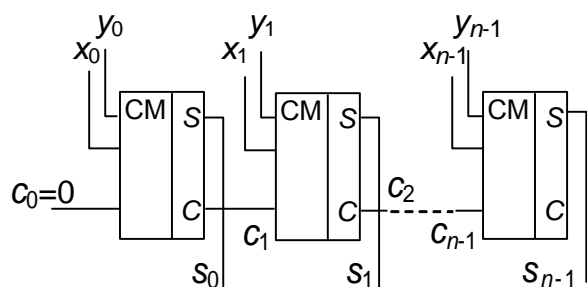


Рис. 3.6. Параллельный сумматор с последовательным переносом

Сумматор для параллельных операндов строится как цепочка одноразрядных сумматоров, соединенных последовательно по цепям переноса (рис. 3.6). Время суммирования определяется временем распространения сигналов переноса и составляет $t_{CM} = nt_3$, где t_3 – время задержки одноразрядного сумматора, а n – разрядность сумматора. Чем выше разрядность сумматора, тем больше

время суммирования.

Для увеличения быстродействия параллельного сумматора разработаны специальные схемы ускоренного (параллельного) переноса. Во всех разрядах сигналы переноса вырабатываются параллельно во времени. Для этого используются все переменные, необходимые для выработки переноса, т. е. те, от которых зависит его наличие или отсутствие. Таким образом, перенос для каждого разряда сумматора становится внешним, а схема самого сумматора упрощается, поскольку в ней не нужно формировать перенос и достаточно одного выхода суммы (рис. 3.7). Обозначение CR происходит от слова *carry* (перенос).

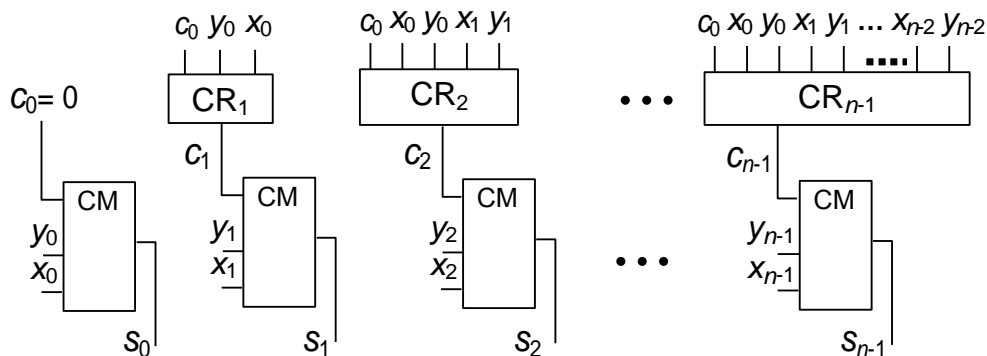


Рис. 3.7. Структура сумматора с параллельным переносом

Диапазон разрядностей, в которых проявляются достоинства сумматоров с параллельным переносом, невелик. До $n = 3...4$ преимущества имеют более простые схемы сумматоров с последовательным переносом, после $n = 8$ появляются перегруженные элементы и элементы с большим числом входов, что замедляет работу сумматоров, требует введения развязывающих элементов с их задержками и т. п. Рационально строить сумматоры с групповой структурой, в которой комбинируются варианты с последовательным и

параллельным переносами между группами. В самих группах перенос может быть любым.

3.1.2. Матричные умножители

Микросхемы множительных устройств появились в 1980-х годах, когда достигнутый уровень интеграции позволил разместить на кристалле достаточно большое количество логических элементов. Структура матричных умножителей тесно связана со структурой математических выражений, описывающих операцию умножения.

Пусть имеются два целых двоичных числа без знаков $A_m = a_{m-1} \dots a_0$ и $B_n = b_{n-1} \dots b_0$. Их перемножение выполняется по известной схеме «умножения столбиком». Если числа четырехразрядные, т. е. $n = m = 4$, то

$$\begin{array}{rcccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
 & & & & \times & & & \\
 & & & & b_3 & b_2 & b_1 & b_0 \\
 \hline
 & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 \\
 & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\
 & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\
 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\
 \hline
 p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
 \end{array}$$

Произведение выражается числом $P_{m+n} = p_{m+n-1} p_{m+n-2} \dots p_0$. Члены вида $a_i b_j$, где $i = 0 \dots (m - 1)$ и $j = 0 \dots (n - 1)$ вырабатываются параллельно во времени конъюнкторами. Их сложение в столбцах, которое можно выполнить разными способами, составляет основную операцию для умножителя и почти целиком определяет время перемножения.

Матричные перемножители могут быть просто множительными (МБ) или множительно-суммирующими (МСБ) блоками. Последние обеспечивают удобство наращивания размерности умножителя. МСБ реализуют операцию $P = A_m \times B_n + C_m + D_n$, т. е. добавляют к произведению два слагаемых: одно разрядности m , совпадающей с разрядностью множимого, другое разрядности n , совпадающей с разрядностью множителя.

Множительно-суммирующие блоки

Множительно-суммирующий блок для 4-разрядных операндов без набора конъюнкторов, вырабатывающих члены вида $a_i b_j$, показан на рис. 3.8а, а

принятое для одноразрядного сумматора обозначение – на рис. 3.8б. Для построения МСБ чисел равной разрядности требуется n^2 конъюнкторов и n^2 одноразрядных сумматоров.

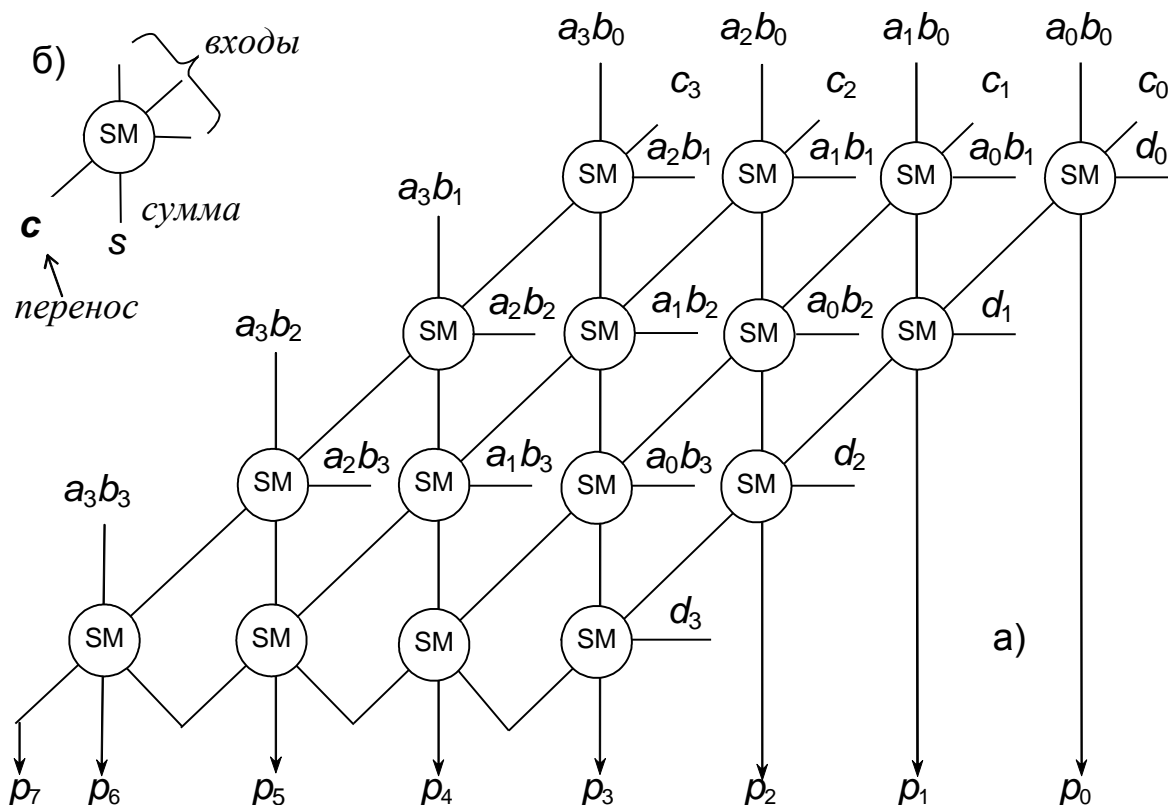


Рис. 3.8. Схема множительно-суммирующего блока для четырехразрядных сомножителей (а) и обозначение одноразрядного сумматора для данной схемы (б)

Максимальная длительность умножения – сумма задержек сигналов в конъюнкторах t_K для выработки членов $a_i b_j$ и задержки в наиболее длинной цепочке передачи сигналов в матрице одноразрядных сумматоров, состоящей из $2n-1$ ($m+n-1$ в общем случае) элементов. Таким образом, $t_{МСБ} = t_K + (2n-1)t_{СМ}$.

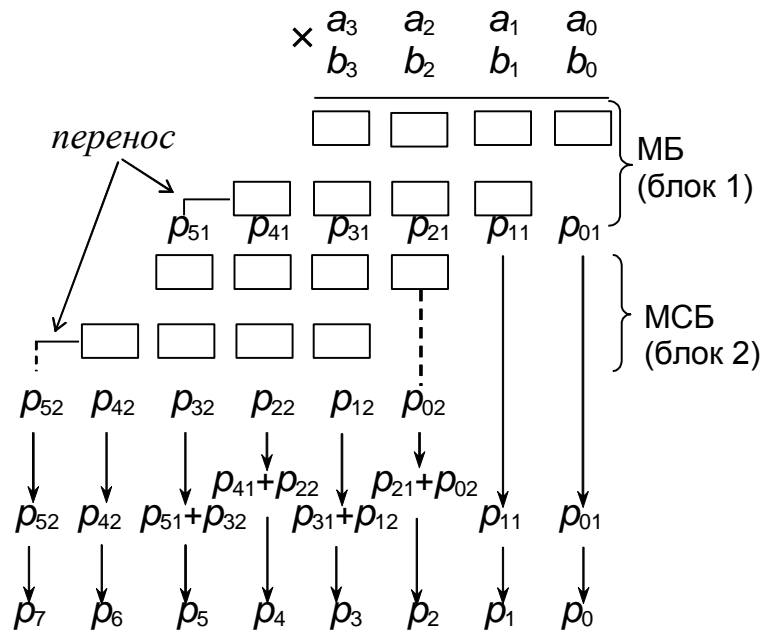


Рис. 3.9. Принцип наращивания разрядности множительных устройств

Схема множительного блока отличается от схемы МСБ тем, что в ней отсутствуют сумматоры правой диагонали, т. к. при $C_m = 0$ и $D_m = 0$ они не требуются.

Построение умножителей большей размерности из умножителей меньшей размерности на основе МБ требует введения дополнительных схем, называемых «деревьями Уоллеса», которые имеются в некоторых зарубежных сериях. При использовании МСБ дополнительные схемы не требуются. Принцип наращивания размерности умножителя иллюстрирует рис. 3.9 на примере матричного умножителя из МСБ «4×2». На поле частичных произведений выделены зоны, воспроизведение которых возможно на блоках размерности 4×2. Перемножение в пределах зон дает частичные произведения $p_1 = p_{51}p_{41}p_{31}p_{21}p_{11}p_{01}$ и $p_2 = p_{52}p_{42}p_{32}p_{22}p_{12}p_{02}$. Для получения конечного значения произведения эти частичные произведения нужно сложить с учетом их взаимного расположения (сдвига одного относительно другого).

3.2. Программируемые логические матрицы (ПЛМ)

В общем случае ПЛМ представляют собой логическую схему для реализации системы функций $Y = (y_{m-1}, \dots, y_1, y_0)$ от переменных $X = (x_{n-1}, \dots, x_1, x_0)$:

$$X = (x_{n-1}, \dots, x_1, x_0) \Rightarrow Y = (y_{m-1}, \dots, y_1, y_0).$$

Правило преобразования задается таблицей истинности. Разряды выходного кода y_{m-1}, \dots, y_1, y_0 рассматриваются как система булевых функций от двоичных переменных x_{n-1}, \dots, x_1, x_0 . Обычно эту систему представляют в

минимальной дизъюнктивной нормальной форме (МНДФ), например, как показано в (3.10) для системы из четырех функций $Y=(y_3, y_2, y_1, y_0)$ с набором из m ($m \leq n = 5$) аргументов, образующих вектор $X = (x_4, x_3, x_2, x_1, x_0)$:

$$\begin{array}{l}
 y_0 = F_0 \vee F_1 \vee F_5, \\
 y_1 = F_1 \vee F_2 \vee F_3 \vee F_6 \\
 y_2 = F_0 \vee F_3 \vee F_5, \\
 y_3 = F_4 \vee F_5 \vee F_6,
 \end{array}
 \left| \right. , \quad \text{где} \quad \left| \begin{array}{ll}
 F_0 = \bar{x}_0 x_1 x_3 x_4, & F_4 = x_0 \bar{x}_1 x_2 \bar{x}_3 x_4, \\
 F_1 = x_0 x_2 \bar{x}_3, & F_5 = \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4, \\
 F_2 = x_0 \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4, & F_6 = \bar{x}_0 x_1 x_2 \bar{x}_3 \bar{x}_4, \\
 F_3 = \bar{x}_0 \bar{x}_1 x_2, &
 \end{array} \right. \quad (3.10)$$

Программируемые логические матрицы имеют два поля: поле, на котором выполняются операции конъюнкции, и поле дизъюнкторов. Для их построения используются те же электронные приборы, что и в базовых логических элементах, – например, диоды для операций конъюнкции (рис. 3.10а) и транзисторы для дизъюнкций (рис. 3.10б). Отличительным свойством здесь является то, что диоды и транзисторы соединены с нагрузкой через переключки, которые можно разрушать (например, расплавить) электрическим воздействием на них в процессе программирования.

Для программируемых дизъюнкторов и конъюнкторов используются специальные обозначения (рис. 3.10в, г). Точки показывают на сохраненные при программировании соединения, а их отсутствие – на разрушенные.

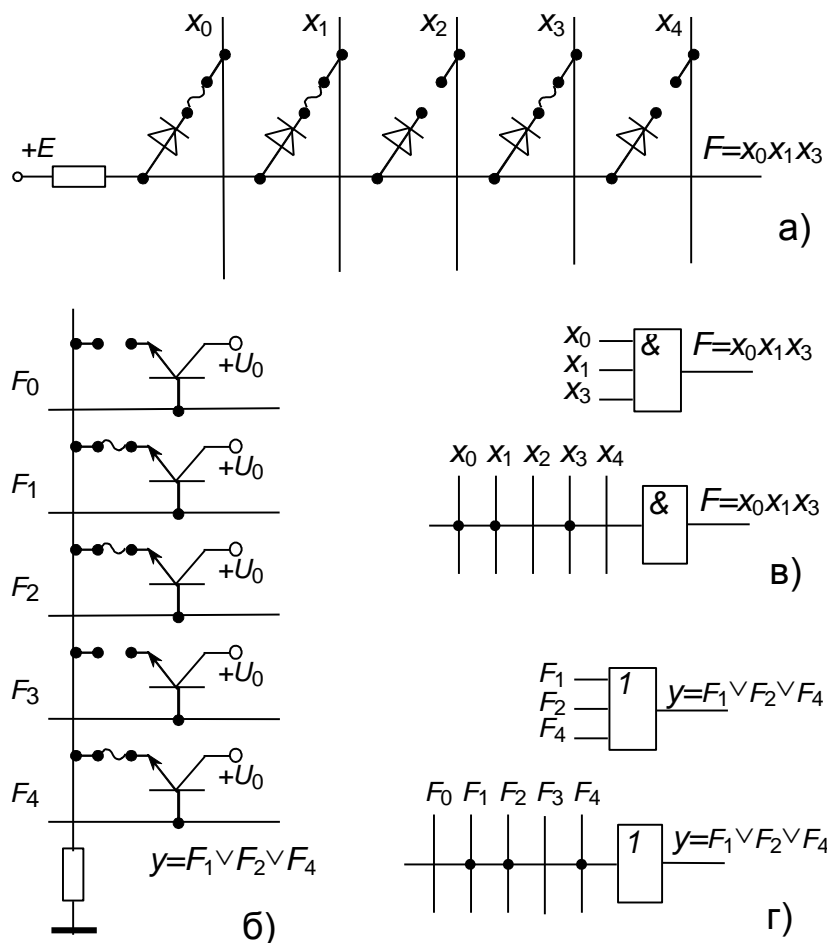


Рис. 3.10. Конъюнктор (а) и дизъюнктор (б) с программируемыми (плавкими) переключками и их схемные обозначения (в) и (г)

Типовая структура ПЛИС (см. пример на рис. 3.11) содержит два коммутационных поля (матрицу И и матрицу ИЛИ), где осуществляются необходимые соединения при реализации заданного набора функций. Число строк в матрице И совпадает с числом строк матрицы ИЛИ и равно числу различных компликант в ДНФ заданных функций. Число столбцов матрицы И соответствует размерности входного вектора X . Кроме того, ПЛИС имеет набор инверторов по числу входных переменных. Число столбцов в матрице ИЛИ определяет число логических функций, одновременно реализуемых на выходах ПЛИС.

ПЛИС на рис. 3.11 запрограммирована так, чтобы выполнялась система из четырех функций $y_3...y_0$, представленных выражениями (3.10).

Программирование ПЛИС выполняется различными способами. Вместо рассмотренных выше разрушаемых металлических соединений («пережигаемых» переключек) между определенными входами и выходами элементов в структуре ПЛИС можно применять транзисторы с изменяемым (проводящим или непроводящим) состоянием (рис. 3.12). При этом используются специальные МДП-структуры, в которых проводящий канал

индуцируется под действием заряда, накапливаемого на границе раздела двух диэлектриков под затвором или на изолированном («плавающем») затворе при подаче программирующего импульса напряжения (см. раздел 6.4).

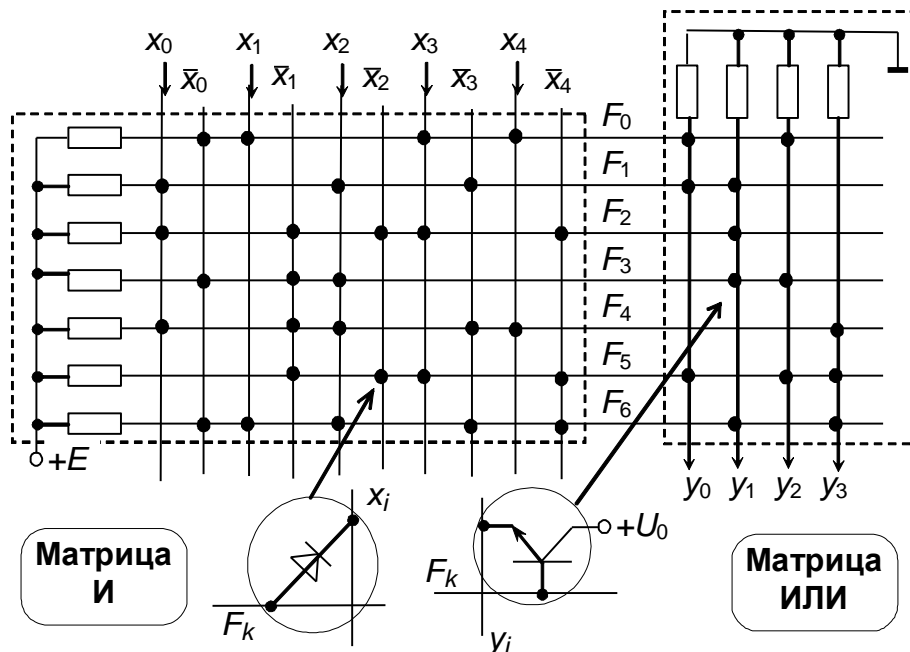


Рис. 3.11. Логическая матрица, запрограммированная для выполнения системы функций в соответствии с выражениями (3.6)

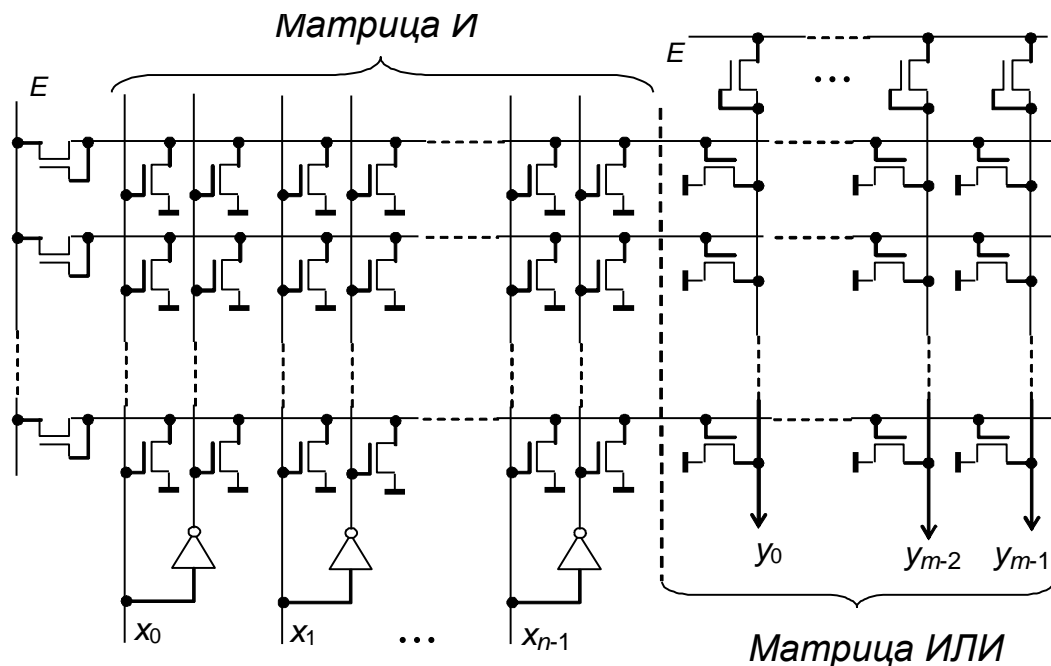


Рис. 3.12. ПЛМ на МДП-структурах

Накопленный заряд сохраняется в течение нескольких лет, обеспечивая необходимое соединение элементов ПЛМ. Перед программированием все МДП-структуры переводятся в непроводящее состояние путем облучения

ультрафиолетом или подачей специального электрического сигнала. Затем формируются сигналы (например, с помощью программатора), переводящие определенные МДП-структуры в проводящее состояние путем образования в них индуцированных каналов. При этом возможно многократное программирование (репрограммирование) ПЛМ для реализации различных наборов функций.

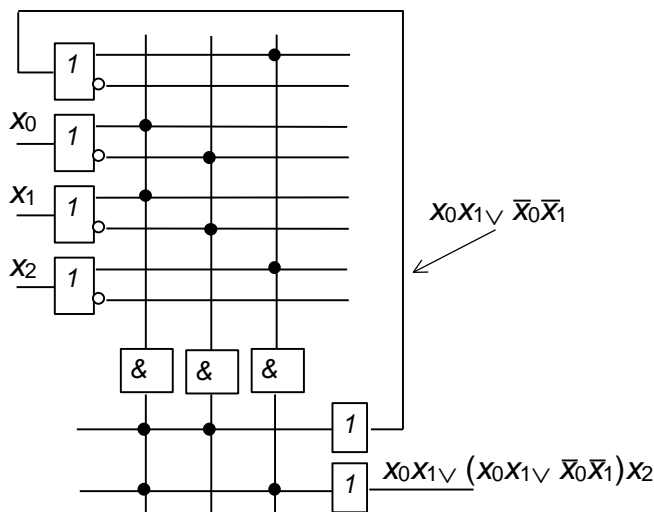


Рис. 3.13. Воспроизведение скобочных форм с помощью ПЛМ

На рис. 3.12 показаны все транзисторы ПЛМ. После программирования только некоторая их часть перейдет в проводящее состояние и будет задействована для выполнения логических операций.

С помощью ПЛМ можно воспроизводить не только нормальные формы логических функций, но и скобочные формы. В этом случае сначала получают выражения в скобках, а затем их используют как аргументы для получения окончательного результата. В схеме появляются обратные связи – промежуточные

результаты с выхода вновь подаются на входы, логическая глубина схемы увеличивается, задержка выработки результата растет. Пусть, например, требуется получить функцию

$$F = x_0x_1 \vee (x_0x_1 \vee \bar{x}_0\bar{x}_1)x_2.$$

Для этого ПЛМ следует включить по схеме, показанной на рис. 3.13.

3.3. Программируемая матричная логика (ПМЛ)

Одно из важных применений программируемой логики – замена ИС малого и среднего уровня интеграции при реализации так называемой произвольной логики. В этих применениях мощность ПЛМ зачастую используется неполно. Это проявляется, в частности, при воспроизведении типичных для практики систем логических функций, не имеющих больших пересечений друг с другом по одинаковым термам. В таких случаях возможность использования выходов любых конъюнкторов любыми дизъюнкторами (как предусмотрено в ПЛМ) становится излишним усложнением. Отказ от этой возможности означает отказ от программирования матрицы ИЛИ и приводит к структуре ПМЛ.

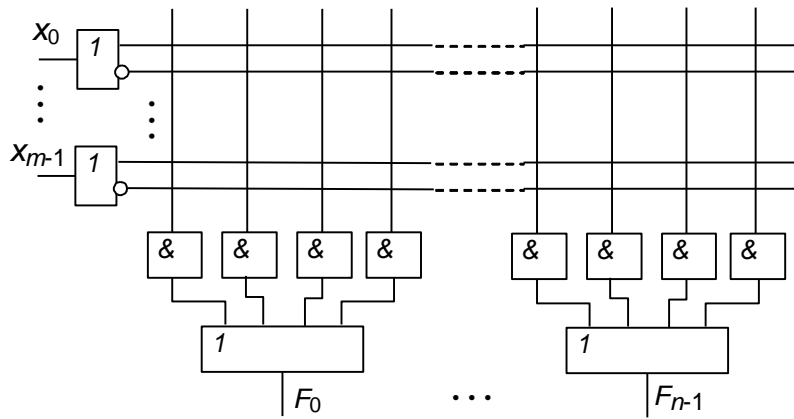


Рис. 3.14. Базовая структура ПМЛ

В ПМЛ (рис. 3.14) выходы элементов И (выходы матрицы И) жестко распределены между элементами ИЛИ (входами матрицы ИЛИ). В показанной на рисунке ПМЛ m входов, n выходов и $4n$ элементов И, поскольку каждому элементу ИЛИ придается по четыре конъюнктора.

Большой функциональностью обладает матричная логика, в которой имеются не только однонаправленные входы и однонаправленные выходы, как это представлено на рис. 3.11-3.13, но и двунаправленные выходы «вход/выход». Схему, в которой некоторые выходы можно использовать для работы в качестве входов или выходов, можно построить, используя элементы с тремя состояниями. В такой схеме один из конъюнкторов в матрице И предназначается для управления элементом с тремя состояниями выхода (рис. 3.15). Выход этого элемента одновременно связан с матрицей И как вход.

Возможны четыре режима вывода Вх/Вых в зависимости от того, как запрограммированы входы конъюнктора К:

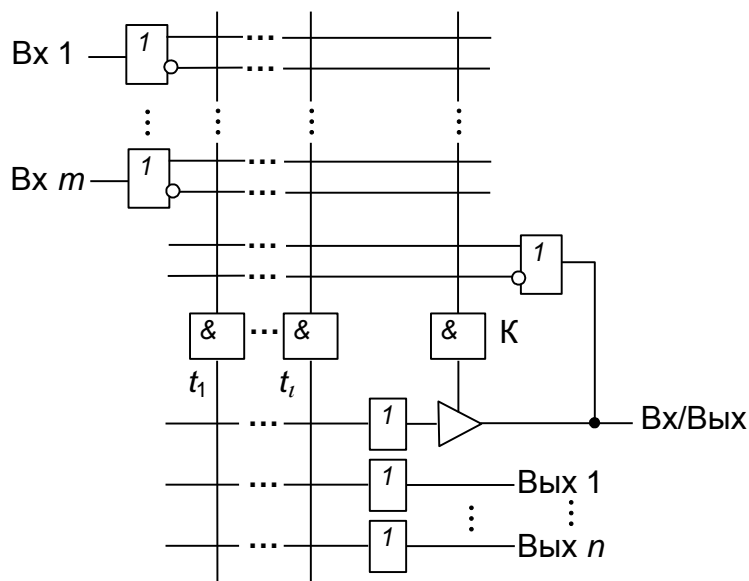


Рис. 3.15. Схема ПМЛ с двунаправленным буфером

1. На всех входах конъюнктора К соединения не тронуты. В этом режиме на выходе конъюнктора К будет нуль, буфер имеет третье состояние выхода и вывод функционирует как ввод.
2. На всех входах конъюнктора К соединения разрушены, буфер активен, вывод работает как выход (его сигналы не используются в матрице И).
3. Выход с обратной связью. Этот режим отличается от предыдущего только тем, что сигналы вывода используются в матрице И.
4. Управляемый выход. Здесь входы конъюнктора К программируются. При заданной комбинации входных сигналов конъюнктор приобретает единичный выход, вывод срабатывает как выход.

В схеме с некоторым числом двунаправленных буферов можно изменять соотношение числа входов-выходов.

3.4. Базовые матричные кристаллы (БМК)

Первые образцы базовых матричных кристаллов появились в 1975 г. как средство реализации нестандартных схем для высокопроизводительной ЭВМ. Термин БМК характерен для литературы на русском языке и поэтому используется здесь. В английской терминологии принят термин GA (*Gate Array*), чему соответствует русский термин «вентильная матрица».

Основа БМК первого поколения – совокупность регулярно расположенных на кристалле базовых ячеек, между которыми имеются свободные зоны для создания соединений (каналы). Базовые ячейки занимают внутреннюю область БМК, в которой они расположены по строкам и столбцам, и содержат группы некоммутированных элементов (транзисторов, резисторов и др.). В периферийной области кристалла размещены ячейки ввода-вывода, набор схемных компонентов которых ориентирован на реализацию связей БМК с внешними цепями.

Таким образом, БМК является заготовкой, которая преобразуется в требуемую схему выполнением необходимых соединений. Потребитель может реализовать на основе БМК некоторое множество устройств определенного класса, задав для кристалла тот или иной вариант рисунка межсоединений компонентов.

В матричные кристаллы могут быть интегрированы не только цифровые базовые ячейки, но аналоговые. В связи с этим БМК подразделяются на *цифровые, аналоговые и цифроаналоговые*. Аналоговые и цифроаналоговые БМК, появившиеся позднее цифровых, имеют состав базовых ячеек, позволяющий получать такие схемы, как операционные усилители, аналоговые ключи и компараторы и т. д.

Рост уровня интеграции ведет к возможностям реализации на одном кристалле все более сложных устройств и систем. Это вызвало к жизни *блочные структуры* БМК, архитектура которых упрощает построение комбинированных устройств, содержащих как блоки логической обработки

данных, так и память или другие специализированные блоки. При этом в БМК реализуются несколько блоков-подматриц, каждый из которых имеет как бы структуру БМК меньшей размерности. Между блоками располагаются трассировочные каналы.

4. Функциональные узлы последовательного типа (автоматы с памятью)

4.1. Триггерные устройства.

Классификация. Основные сведения

Возьмем цепочку последовательно соединенных ключей (рис. 4.1а). Каждый ключ находится по соседству с ключами, находящимися в противоположных состояниях. Значит, в произвольной паре смежных ключей выходное напряжение предыдущего ключа такое же, как входное последующего.

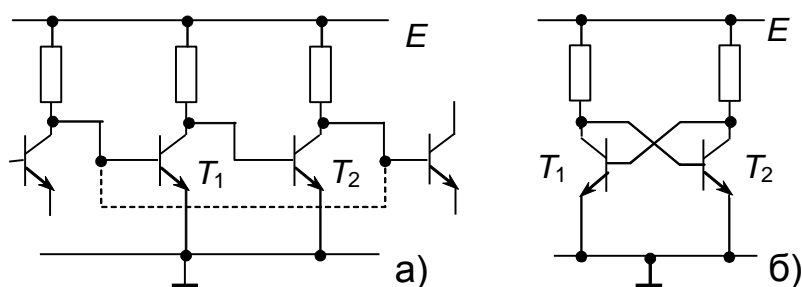


Рис. 4.1. Цепочка биполярных ключей (а) и изолированная пара ключей (б)

Если изолировать такую пару от соседних ключей и соединить ее выход со входом, то ее состояние не изменится. Это *устойчивое состояние*, и этих устойчивых состояний два:

- 1) транзистор T_1 – закрыт, T_2 – открыт и
- 2) транзистор T_1 – открыт, T_2 – закрыт.

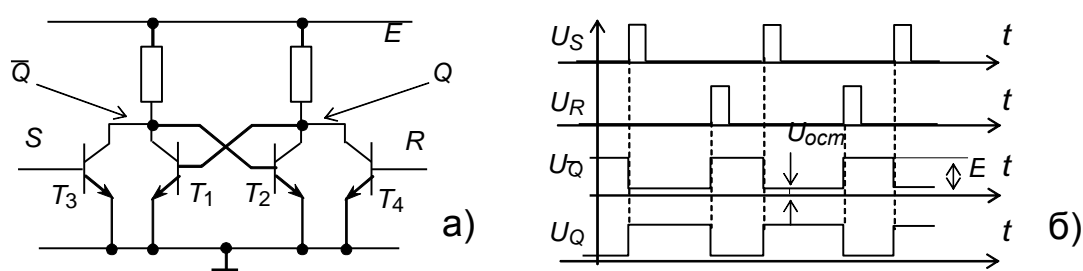


Рис. 4.2. Бистабильная ячейка (а) и осциллограммы напряжений (б), поясняющие ее работу

Изолированную пару ключей можно представить в виде ячейки, имеющей два равноценных устойчивых состояния (рис. 4.1б). Такие ячейки являются основой для построения триггеров и элементов памяти. Но при этом возникает необходимость занесения в ячейку информации, т. е. перевода ее в одно из устойчивых состояний в соответствии с заносимой двоичной информацией. Для этой цели можно использовать дополнительные

управляющие транзисторы T_3 и T_4 (рис. 4.2а). Таким образом возникают пары параллельно соединенных транзисторов T_1, T_3 и T_2, T_4 с общей для каждой из пар нагрузкой. В положительной логике параллельно соединенные транзисторы с общей коллекторной нагрузкой выполняют операцию ИЛИ-НЕ (см. рис. 2.9). Исходя из этого получаем логическую схему бистабильной (имеющей два устойчивых состояния) ячейки (рис. 4.3а) с управляющими R (*Reset*) и S (*Set*) входами для установки одного и «сброса» другого состояния. В составе бистабильной ячейки (БЯ) имеются два дизъюнктора (два элемента 2ИЛИ-НЕ), в силу чего такую бистабильную ячейку называют дизъюнктивной.

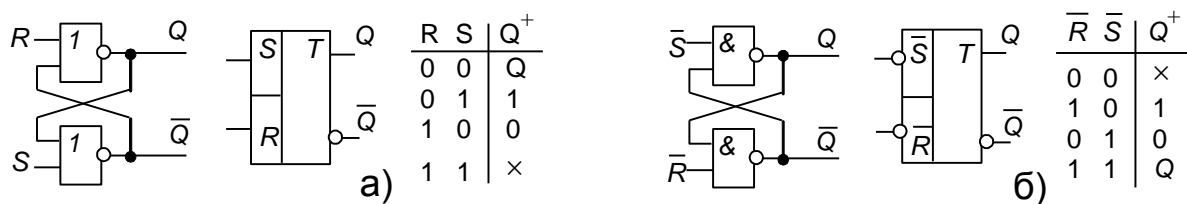


Рис. 4.3. Дизъюнктивная (а) и конъюнктивная (б) БЯ со схемными обозначениями и пояснениями логики работы

Логика работы дизъюнктивной БЯ представлена таблицей истинности на рис. 4.3а и пояснена осциллограммами на рис. 4.2б. Состояние БЯ Q^+ , которое она примет после управляющего воздействия, зависит от сигналов R и S и от текущего состояния Q . При $R = 0$ и $S = 0$ ячейка находится в режиме хранения ($Q^+ = Q$). После $R = 1$ и $S = 0$ происходит сброс БЯ (установка $Q^+ = 0$), а после $R = 0$ и $S = 1$ – ее установка (установка $Q^+ = 1$). Комбинация $R = 1$ и $S = 1$ *запрещена*, т. к. состояние БЯ после такого воздействия неконтролируемо. Следовательно,

$$Q^+ = S \vee Q\bar{R}. \quad (4.1)$$

При замене положительной логики на отрицательную элементы ИЛИ превращаются в элементы И, и дизъюнктивная БЯ преобразуется в конъюнктивную (рис. 4.3б). Работу конъюнктивной БЯ можно описать в рамках положительной логики, однако активными в этом случае становятся низкие уровни напряжений. Это учтено в схемном обозначении на рис. 4.3б, где активные уровни управляющих сигналов отмечены знаками инверсии. Заметим, что для конъюнктивной БЯ *запрещенной* является комбинация $\bar{R} = 0$ и $\bar{S} = 0$.

В отношении дизъюнктивной и конъюнктивной БЯ применимо обобщающее название – *RS-триггер*, точнее, асинхронный *RS-триггер*.

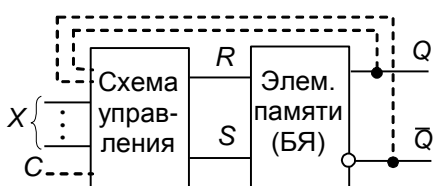


Рис. 4.4. Обобщенная структура триггера

С общих позиций триггеры – это элементарные автоматы, содержащие собственно элемент памяти и схему управления (рис. 4.4). В отличие от упоминавшихся ранее динамических элементов, здесь элементы памяти являются *статическими* и представляют собой бистабильные ячейки, или *RS-триггеры*.

Кроме внешних сигналов (вектор X), на схему управления триггера могут подаваться сигнал синхронизации и сигналы обратной связи. В *асинхронных триггерах* нет входов синхронизации и использование обратной связи невозможно. Состояние *синхронизируемого триггера* изменяется в строго определенные интервалы (*триггеры, синхронизируемые уровнем сигнала C*) или моменты (*триггеры, синхронизируемые перепадом сигнала C*) времени. Обратная связь возможна только в триггерах, *синхронизируемых перепадом*, и по сути она является обратной связью с задержкой – схема управления учитывает предшествующее состояние триггера, но во время перепада (смены уровня) синхронизирующего сигнала выходы триггера стабильны.

По логике функционирования различают триггеры типов RS , D , T , JK и другие. Кроме того, используются комбинированные триггеры и триггеры со сложной входной логикой (с группами входов, связанных между собой логическими зависимостями).

Триггеры, синхронизируемые уровнем

Асинхронный RS -триггер имеет два входа – установки в единицу (S) и установки в ноль (R). В *синхронизируемый RS -триггер* добавлена схема управления в виде двух элементов 2И-НЕ (рис. 4.5а), благодаря чему действие управляющих сигналов возможно только при наличии сигнала синхронизации C . Это поясняют осциллограммы сигналов на входах и выходе триггера (рис. 4.5в).

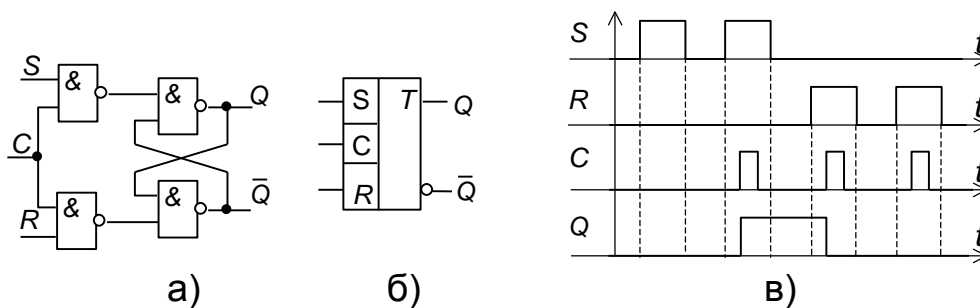


Рис. 4.5. Логическая схема (а), условное обозначение (б) синхронизируемого уровнем RS -триггера и поясняющие его работу осциллограммы сигналов (в)

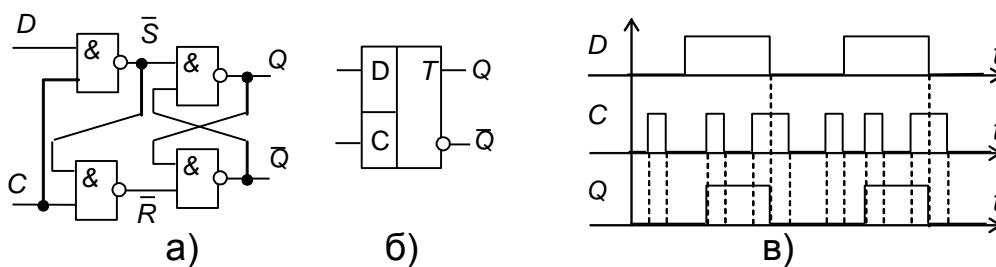


Рис. 4.6. Логическая схема (а), условное обозначение (б) синхронизируемого уровнем D -триггера и поясняющие его работу осциллограммы сигналов (в)

Логика работы синхронизируемого RS -триггера та же, что и у асинхронного (см. выражение (4.1)), но *при действии активного уровня синхросигнала C* . Сигнал C как входной показан и в условном обозначении триггера (рис. 4.5б).

Триггер типа D (от слова *Delay* – задержка) имеет один информационный вход (рис. 4.6а, б). Его состояние повторяет входной сигнал, но с задержкой, определяемой сигналом синхронизации (рис. 4.6в): $Q^+ = D$.

Триггеры, синхронизируемые перепадом

Синхронизируемые перепадом триггеры также имеют свои разновидности (RS , D , JK и др.). Чтобы пояснить отличия в работе триггеров, синхронизируемых уровнем и перепадом, рассмотрим работу синхронизируемого перепадом RS -триггера (рис. 4.7). В нем схема управления наряду с логическими элементами содержит *управляющую бистабильную ячейку*, что дает структуру, состоящую из двух синхронизируемых уровнем RS -триггеров (структуру типа «ведущий–ведомый»). При наличии управляющей БЯ состояние элемента памяти триггера соответствует логическому выражению (4.1), но меняется только в моменты отрицательных перепадов (перехода с высокого уровня на низкий, или *среза*) синхронизирующих (тактовых) импульсов (рис. 4.7б).

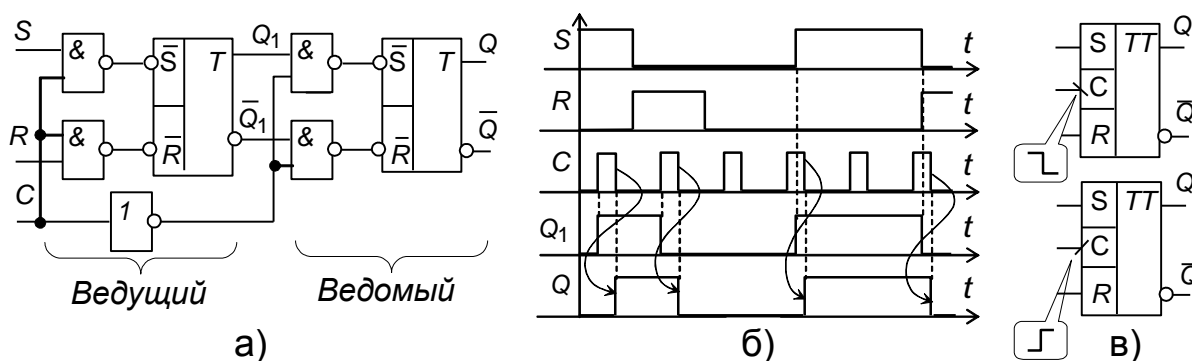


Рис. 4.7. Логическая схема (а), осциллограммы сигналов (б) и условные обозначения (б) RS -триггеров, синхронизируемых перепадом (срезом и фронтом)

Условное обозначение синхронизируемого срезом RS -триггера представлено на рис. 4.7в (верхний рисунок). Сдвоенное обозначение « TT » показывает на структуру типа «ведущий–ведомый», а косая черта в поле входа « C » – на вид синхронизирующего перепада. При синхронизации положительным перепадом (фронтом) в обозначении триггера (в поле входа « C ») меняется наклон косой черты (нижний рис. 4.7в). Такой способ указания

вида синхронизации применяется и в условных обозначениях других типов триггеров, например, *D*- и *T*-триггеров. Сами синхровходы при этом называют *динамическими – прямыми или инверсными* в зависимости от синхронизации фронтом или срезом.

Триггер типа *T* имеет только один вход – вход сигнала синхронизации – и меняет свое состояние при каждом воздействии этого сигнала, то есть работает в режиме счетчика.

Триггер типа *JK* (рис. 4.8) универсален, имеет входы установки (*J*) и сброса (*K*), подобные входам *RS*-триггера. В отличие от последнего он допускает одновременную подачу логических единиц на оба управляющих входа ($J = K = 1$). В этом режиме работает как счетный триггер относительно тактового входа. Логика работы триггера, как это следует из таблицы истинности на рис. 4.8в, описывается выражением $Q^+ = \bar{K}Q \vee J\bar{Q}$.

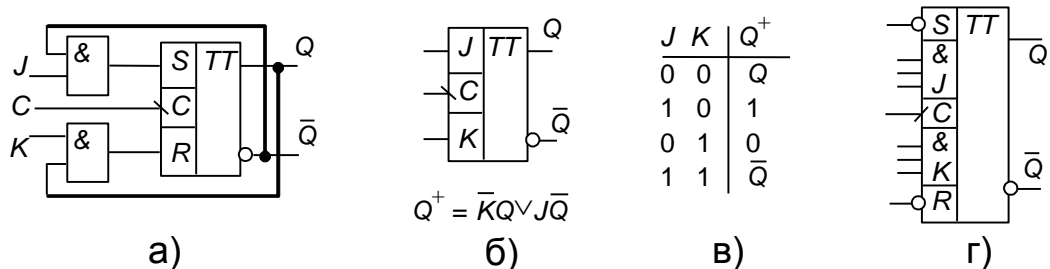


Рис. 4.8. Логическая схема (а), условное обозначение (б) и таблица истинности (в) *JK*-триггера. *JK*-триггер со сложной входной логикой (г)

Примером *триггера со сложной входной логикой* служит *JK*-триггер с группами входов, объединенных операциями конъюнкции $J = J_1 J_2 J_3$ и $K = K_1 K_2 K_3$ (рис. 4.8г). Кроме того, синхронизируемые перепадом триггеры могут иметь асинхронные входы *R* и *S* (как, например, на рис. 4.8г), которые обычно используются для его начальной установки.

4.2. Регистры и регистровые файлы

Регистры – самые распространенные узлы цифровых устройств. Они оперируют с множеством связанных переменных, составляющих двоичные слова. Над словами выполняется ряд операций: прием, выдача, хранение, сдвиг в разрядной сетке, поразрядные логические операции. Регистры состоят из разрядных схем, в которых имеются триггеры и, чаще всего, также и логические элементы.

По составу линий передачи выходных переменных регистры делятся на однофазные и парафазные, по системе синхронизации – на одноктактные, двухтактные и многотактные. Однако главным классификационным признаком является способ приема и выдачи данных. По этому признаку различают

параллельные (статические) регистры, последовательные (сдвигающие) и параллельно-последовательные.

В параллельных регистрах прием и выдача слов производятся по всем разрядам одновременно. В них хранятся слова, которые могут быть подвергнуты поразрядным логическим преобразованиям.

В последовательных регистрах слова принимаются и выдаются разряд за разрядом. Их называют сдвигающими, т. к. тактирующие сигналы при вводе и выводе слов перемещают их по разрядной сетке. Сдвигающий регистр может быть нереверсивным (с однонаправленным сдвигом) или реверсивным (с возможностью сдвига в обоих направлениях).

Последовательно-параллельные регистры имеют входы-выходы одновременно последовательного и параллельного типа. Имеются варианты с последовательным входом и параллельным выходом, параллельным входом и последовательным выходом, а также варианты с возможностью любого сочетания способов приема и выдачи слов.

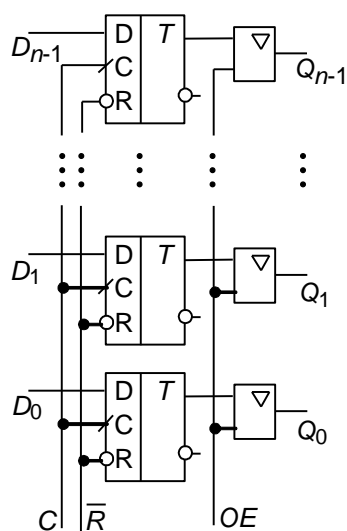


Рис. 4.9. Схема статического регистра

В параллельных регистрах схемы разрядов не обмениваются данными между собой. Общими для разрядов обычно являются цепи тактирования, сброса/установки, разрешения выхода или приема, т. е. цепи управления. Пример параллельного регистра на триггерах типа *D* с прямыми динамическими входами и входами сброса представлен на рис. 4.9. Выходы триггеров регистра имеют *три состояния*, одно из которых является *высокоимпедансным*. Перевод выходов регистра в третье (высокоимпедансное) состояние, осуществляемый под воздействием сигнала *OE (Output Enable)*, позволяет запретить использование регистра как источника информации, что необходимо тогда, когда разные цепи используют одни и те же линии для передачи или приема информации.

Для современной схемотехники характерно построение регистров на *D*-триггерах, преимущественно с динамическим управлением. Многие имеют выходы с третьим состоянием, некоторые регистры относятся к числу буферных, т. е. рассчитаны на работу с большими емкостными и/или низкоомными активными нагрузками. Это обеспечивает их работу непосредственно на магистраль без дополнительных схем интерфейса.

Регистровые файлы – это блоки регистровой памяти. Информационные входы регистров соединены параллельно. Выбор регистра для записи/чтения обеспечивается соответствующими схемами дешифрации адреса и сопровождающими сигналами синхронизации (сигналы чтение/запись).

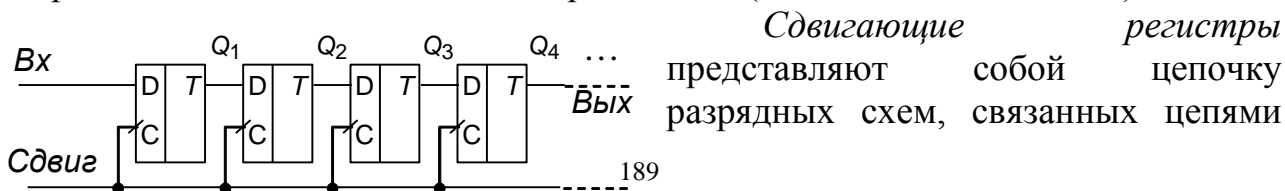


Рис. 4.10. Регистр сдвига вправо

Сдвигающие регистры представляют собой цепочку разрядных схем, связанных цепями

переноса. В одноктактных регистрах со сдвигом на один разряд вправо (рис. 4.10) слово сдвигается ($Q_i \rightarrow Q_{i+1}$) при поступлении каждого синхросигнала. Вход и выход последовательные. Сделав пересоединение входов и выходов, можно получить сдвиговый регистр влево, а применив коммутационную логику – реверсивный сдвиговый регистр.

Многотактные регистры управляются несколькими последовательностями синхроимпульсов. Из их числа наиболее известны двухтактные с основным и дополнительным регистрами, построенными на простых одноступенчатых триггерах, управляемых уровнем. По такту C_1 содержимое основного регистра переписывается в дополнительный, а по такту C_2 возвращается в основной, но уже в соседние разряды, что соответствует сдвигу слова. По затратам и быстродействию этот вариант близок к одноктактному регистру с двухступенчатыми триггерами.

Здесь уместно упомянуть *регистры сдвига на динамических элементах* (см. рис. 2.30б), в которых элементами памяти являются емкости затворов ключевых МДП-транзисторов. При этом в сдвиговом регистре используется значительно меньшее количество транзисторов, что важно при создании СБИС.

4.3. Двоичные счетчики

Понятие «счетчик» является очень широким. К счетчикам относят автоматы, которые под действием входных импульсов переходят из одного состояния в другое, фиксируя тем самым число поступивших на их вход импульсов в том или ином виде.

Специфичной для счетчиков операцией является изменение их содержимого на единицу (может быть, условную). Прибавление такой единицы соответствует операции инкрементации, вычитание – декрементации. Обычно счетчиками выполняются также и другие операции – сброс, установка, начальная загрузка и др.

Счетчик характеризуется *модулем счета M (емкостью)*. Модуль определяет число возможных состояний счетчика. После поступления на счетчик M входных сигналов начинается новый цикл, повторяющий предыдущий.

4.3.1. Асинхронные (последовательные) счетчики

Как любой автомат, счетчик можно строить на триггерах любого типа, однако удобнее всего использовать для этого триггеры T (счетные) и JK , имеющие при $J = K = 1$ счетный режим.

Состояние счетчика читается по выходам разрядных схем как слово $Q_{N-1}, Q_{N-2}, \dots, Q_0$, входные сигналы поступают на младший разряд счетчика. Двоичным счетчиком называется счетчик, имеющий модуль $M = 2^N$, где N – целое число, и

естественную последовательность кодов состояний, выражаемую двоичным числом.

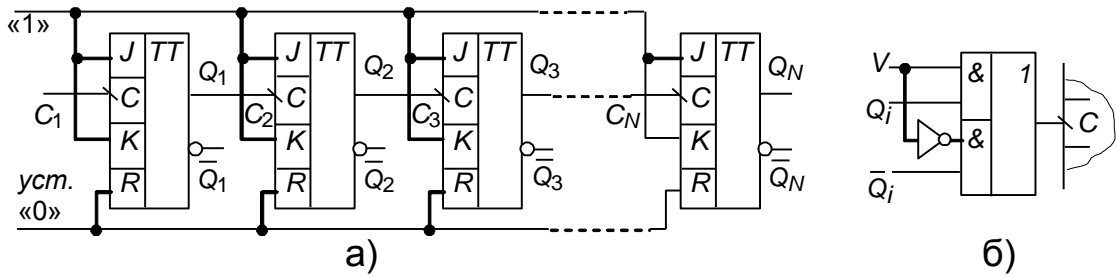


Рис. 4.11. Последовательный (асинхронный) счетчик на JK-триггерах

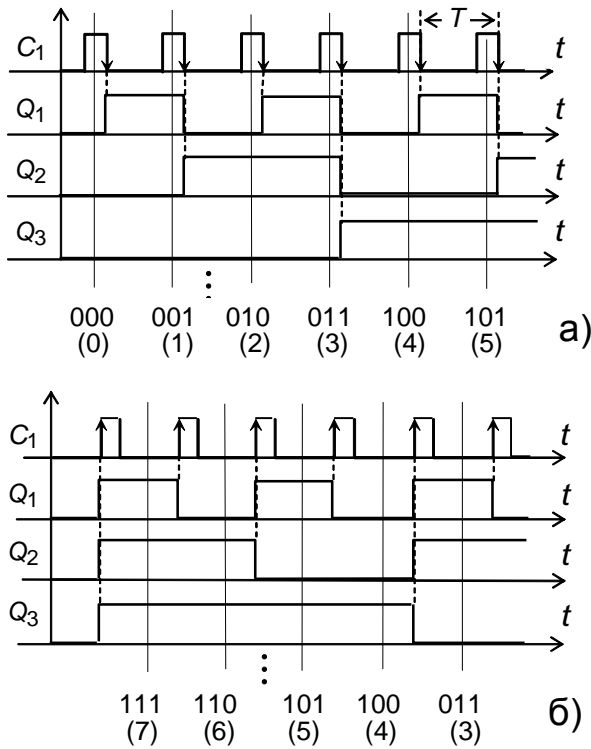


Рис. 4.12. Осциллограммы сигналов, поясняющие работу суммирующего (а) и вычитающего (б) двоичных счетчиков

Последовательный двоичный счетчик строится так, что каждая следующая ступень в нем находится под воздействием только одной предшествующей ступени. На рис. 4.11а представлен счетчик на JK-триггерах, синхронизируемых отрицательным перепадом (срезом). Функция возбуждения i -го триггера ($i = 1, 2, \dots, N$ – номер разряда, N – их общее число)

$$C_i = \bar{Q}_1 \bar{Q}_2 \dots \bar{Q}_{i-1} \quad (i \geq 2) \quad (4.2)$$

такова, что состояние Q_i каждого следующего триггера меняется на противоположное при возникновении отрицательного перепада сигнала на выходе предыдущего, т. е. при смене состояния $Q_{i-1} = 1$ предыдущего триггера на $Q_{i-1} = 0$ (рис. 4.12а). В результате численное значение

двоичного кода $Q_N Q_{N-1} \dots Q_1$ на прямых выходах триггеров возрастает на 1 при поступлении очередного импульса синхронизации на вход первого триггера. Поэтому такой счетчик является *суммирующим*.

При замене триггеров, синхронизируемых срезом, на триггеры, синхронизируемые фронтом (положительным перепадом), функция возбуждения (4.2) заменяется на функцию

$$C_i = Q_1 Q_2 \dots Q_{i-1} \quad (i \geq 2) \quad (4.3)$$

и счетчик становится *вычитающим* – значение двоичного кода $Q_N Q_{N-1} \dots Q_1$ уменьшается на 1 под действием фронта каждого импульса C_1 на входе первой ступени (рис. 4.12б).

Если в счетчик включить логику, с помощью которой можно изменять функцию возбуждения, то возможно изменение направление счета в сторону увеличения или в сторону уменьшения. Так получаются *реверсивные счетчики*. На рис. 4.11б показана логика изменения направления счета с помощью мультиплексора, который видоизменяет функцию возбуждения C_i под действием управляющего сигнала V :

$$C_i = VQ_1 Q_2 \dots Q_i + \bar{V}\bar{Q}_1 \bar{Q}_2 \dots \bar{Q}_i. \quad (4.4)$$

Мультиплексор ставится между соседними ступенями счетчика, и на его входы подаются прямой Q_i и инверсный \bar{Q}_i сигналы предшествующей i -й ступени. Выходной сигнал мультиплексора C_i подается на триггер следующей $(i+1)$ -й ступени. При этом выходами счетчика независимо от направления счета являются прямые выходы триггеров.

4.3.2. Параллельные (синхронные) счетчики

В последовательном счетчике интервал между соседними импульсами счета (или период) $T > t_p$ – времени распространения сигнала по цепи от первого триггера до последнего. Это время можно существенно сократить, введя одновременный перенос из одного разряда счетчика в каждый последующий и распределяя синхросигнал на входы всех триггеров. Таким образом строится схема параллельного (синхронного) счетчика (рис. 4.13). По J и K входам триггера каждой ступени делается операция логического умножения сигнала разрешения счета E и сигналов с выходов предшествующих ступеней. При этом на J и K входы первой ступени подается только сигнал E . Быстродействие параллельного счетчика определяется временем установления одного триггера.

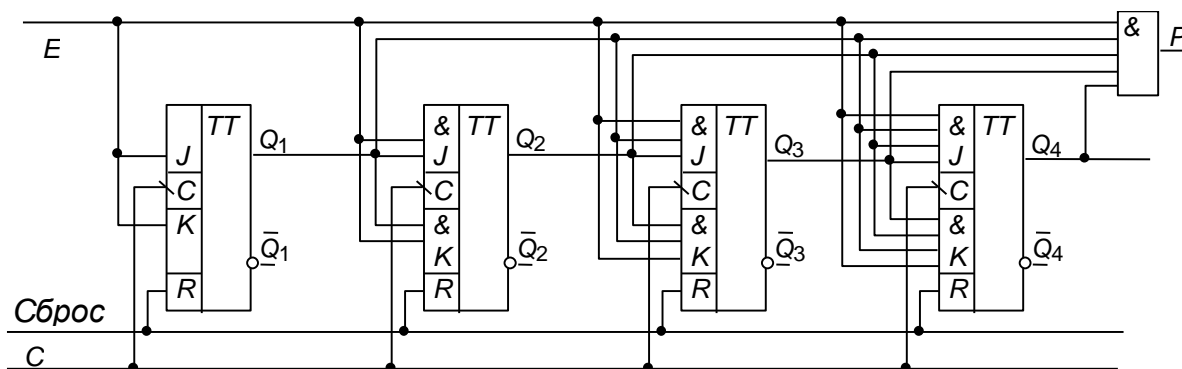


Рис. 4.13. Параллельный (синхронный) суммирующий счетчик на JK -триггерах

Таким образом, каждый из триггеров меняет свое состояние на противоположное при наличии сигнала разрешения счета E и сигналов переноса p_i :

- 1-й триггер – при наличии сигнала E ,
- 2-й триггер – при наличии переноса $p_1 = EQ_1$,
- 3-й триггер – при наличии переноса $p_2 = EQ_1Q_2$,
- 4-й триггер – при наличии переноса $p_3 = EQ_1Q_2Q_3$

и т. д. вплоть до $p_{N-1} = EQ_1Q_2 \dots Q_{N-1}$, если число разрядов равно N .

Счетчик с помощью элемента $(N+1)И$ формирует выходной сигнал переноса

$$P = EQ_1Q_2 \dots Q_N, \quad (4.5)$$

который следует с цикличностью счета и используется обычно для наращивания разрядности. Для счетчика на рис. 4.13 $N = 4$ и общий перенос P формируется элементом 5И.

Начальное состояние счетчика устанавливается по сигналу «Сброс», подаваемому на асинхронные входы R всех триггеров. После формирования общего переноса P на следующем такте синхронизации с номером $M = 2^N$ счетчик возвращается в исходное состояние и цикл счета повторяется. Для $N = 4$ модуль счета $M = 16$.

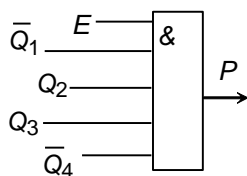


Рис. 4.14. Логика формирования переноса в двоично-десятичном счетчике

Сказанное выше относится к двоичному счетчику, который при заданном числе разрядов обладает наибольшим значением M . Длительность цикла счета можно уменьшить и получить $M < 2^N$, если ввести обратные связи между триггерами. На этом основано построение счетчиков с управляемым числом состояний. Так, например, при $N = 4$, изменив выходную логику формирования переноса P (4.5) на $P = E\bar{Q}_1Q_2Q_3\bar{Q}_4$ (см. рис. 4.14) и введя принудительный сброс всех триггеров по сигналу $R = (E\bar{Q}_1Q_2Q_3\bar{Q}_4) \cdot C = P \cdot C$, можно получить двоично-десятичный счетчик (счетчик по модулю $M = 10$). При этом счет с нуля будет возобновляться каждый раз после перехода счетчика в состояние $Q_4Q_3Q_2Q_1 = 1001$, соответствующее десятичному числу 9 (счет от 0 до 9).

Из суммирующего счетчика легко сделать вычитающий, если подобно тому, как это делалось в отношении последовательных счетчиков, для возбуждения триггеров отдельных разрядов использовать инверсные выходы $\bar{Q}_1, \bar{Q}_2, \dots, \bar{Q}_{N-1}$, а результат счета по-прежнему определять по прямым выходам. В этом случае функция возбуждения

- 1-го триггера равна E ,
- 2-го – $p_1 = E\bar{Q}_1$,

- 3-го – $p_2 = E\bar{Q}_1\bar{Q}_2$ и
- 4-го – $p_3 = E\bar{Q}_1\bar{Q}_2\bar{Q}_3$.

Условием формирования сигнала переноса в этом случае будет

$$P = E\bar{Q}_1\bar{Q}_2\bar{Q}_3\bar{Q}_4. \quad (4.6)$$

Наконец, если предусмотреть возможность возбуждения триггеров и прямыми, и инверсными выходами предыдущих ступеней, то можно сделать реверсивный счетчик. Но при этом требуется дополнительный (помимо E) сигнал V для управления направлением счета и формирования переноса, как это было в случае с последовательным счетчиком (см. выражение (4.4)). С учетом (4.5), (4.6) и сигнала V для функций возбуждения получаем выражения

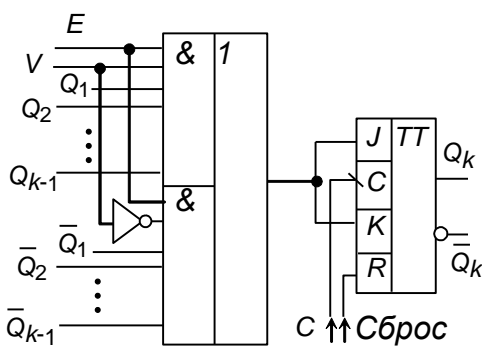


Рис. 4.15. Логика управления направлением счета в параллельном реверсивном счетчике на JK-триггерах

$$p_1 = E(V \cdot Q_1 + \bar{V} \cdot \bar{Q}_1),$$

$$p_2 = E(V \cdot Q_1Q_2 + \bar{V} \cdot \bar{Q}_1\bar{Q}_2),$$

$$p_3 = E(V \cdot Q_1Q_2Q_3 + \bar{V} \cdot \bar{Q}_1\bar{Q}_2\bar{Q}_3)$$

и

$$P = E(V \cdot Q_1Q_2Q_3Q_4 + \bar{V} \cdot \bar{Q}_1\bar{Q}_2\bar{Q}_3\bar{Q}_4).$$

Схемы формирования сигналов возбуждения триггеров, как и для последовательного счетчика, можно построить на мультиплексорах, каналы которых коммутируются посредством сигнала V (рис. 4.15). При этом разрядность каналов мультиплексоров увеличивается по

мере роста номера разряда управляемого триггера.

В режиме суммирования $V = 1$ и

$$P = EP, \text{ где } P = V \cdot Q_1Q_2Q_3Q_4$$

– сигнал переполнения или переноса за пределы разрядной сетки счетчика в сторону старших разрядов.

В режиме вычитания $V = 0$ и

$$P = EW, \text{ где } W = \bar{V} \cdot \bar{Q}_1\bar{Q}_2\bar{Q}_3\bar{Q}_4.$$

Сигнал W устанавливается, когда обнулены все разряды счетчика. Это соответствует «заем» из младшего двоичного разряда, относящегося к числу тех, которые находятся за пределами разрядной сетки данного счетчика.

Сигналы P и W являются выходными сигналами счетчика. Используя их можно наращивать разрядность счета путем каскадирования ступеней малой разрядности. Для суммирующего счетчика схема каскадирования представлена на рис. 4.16.

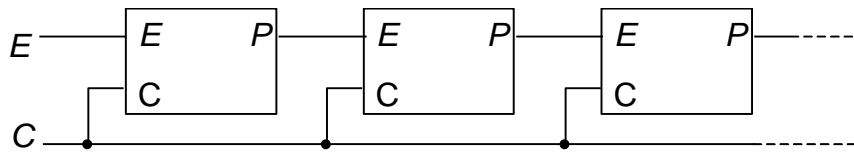


Рис. 4.16. Схема наращивания разрядности счетчиков

4.4. Регистровое арифметическо-логическое устройство

В состав цифровых систем входят последовательные функциональные блоки, выполняющие различные процедуры обработки информации и управления этим процессом. Арифметическо-логические устройства (АЛУ) служат для выполнения арифметических и логических преобразований над словами, называемыми в этом случае *операндами*.

Выполняемые АЛУ операции можно разделить на следующие группы:

- операции двоичной арифметики для чисел с фиксированной запятой;
- операции двоичной (или шестнадцатиричной) арифметики для чисел с плавающей точкой;
- операции индексной арифметики (при образовании и модификации адресов команд);
- операции специальной арифметики;
- операции над логическими кодами (логические операции);
- операции над алфавитно-цифровыми полями.

К арифметическим операциям относятся сложение, вычитание, вычитание модулей, умножение и деление. Группу логических операций составляют операции дизъюнкции и конъюнкции над многоразрядными двоичными словами, сравнение кодов. Специальные арифметические операции включают нормализацию, арифметический сдвиг (сдвигаются только цифровые разряды, знаковый разряд остается на месте), логический сдвиг (знаковый разряд сдвигается вместе с цифровыми разрядами). Обширна группа операций редактирования алфавитно-цифровой информации. Современные процессоры поддерживают высокоточные операции с плавающей точкой, графические операции.

Можно провести следующую классификацию АЛУ.

По способу действия над операндами АЛУ делятся на последовательные и параллельные. В последовательных АЛУ операции представляются в последовательном коде, а операции производятся последовательно во времени над их отдельными разрядами. В параллельных АЛУ операнды представляются параллельным кодом и операции совершаются параллельно во времени над всеми разрядами операндов.

По способу представления чисел различают АЛУ для чисел с фиксированной точкой (запятой) и для чисел с плавающей точкой.

По своим функциям АЛУ является операционным блоком, выполняющим *микрооперации*, обеспечивающие прием из других устройств (например,

памяти) операндов, их преобразование и выдачу результатов преобразования в другие устройства. Арифметическо-логическое устройство есть составная часть центрального процессора (ЦП) и находится под управлением сигналов, которые вырабатываются в ЦП после дешифрации поступившей в него команды (см. разделы 5.3 и 7.3).

Нами будут рассмотрены не все из перечисленных выше операций АЛУ. Остановимся лишь на наиболее широко используемых операциях алгебраического сложения целых чисел. Алгоритмы выполнения АЛУ арифметических операций зависят от того, в каком виде хранятся в памяти отрицательные числа – в *прямом* или *дополнительном*. В последнем случае сокращается время выполнения операции за счет исключения операции преобразования получаемого в АЛУ дополнительного кода отрицательного результата в прямой код, хотя при этом несколько усложняется операция умножения.

Для представления отрицательных целых чисел используется специальный знаковый бит, который равен 1, если число отрицательное, и равен 0 в противном случае. Если при этом код цифр числа остается неизменным, то получающийся в результате код отрицательного числа называют *прямым кодом*. В компьютерах почти всегда применяется дополнительный код, в котором отрицательные целые числа представляются в виде:

$$-b \rightarrow 2^n - b.$$

Здесь b – абсолютное значение числа, а n – число бит, включая бит знака. Образование дополнительного кода поясним на примере с $n = 8$ и $b = 46_{10} = 00101110_2$:

$2^8 = 1\ 0000\ 0000$	
$2^8 - 1 = 1111\ 1111$	
$b = 0010\ 1110$	
$(2^8 - 1) - b = 1101\ 0001$	<i>Разность $(2^8 - 1) - b$</i>
+1	<i>образует инверсный код.</i>
$2^8 - b = 1101\ 0010$	<i>После добавления 1 получаем</i>
	<i>дополнительный код</i>
	<i>(старший бит – знаковый).</i>

При выполнении операции сложения целых чисел положительные слагаемые представляются в прямом коде, а отрицательные – в дополнительном. Производится сложение двоичных кодов, включая разряды знаков. Результат имеет правильное представление, если сохранять только младшие n бит, а перенос в старший $(n+1)$ -й бит не учитывать.

АЛУ наряду с результатом операции формирует *признаки* или *флаги* – признак нулевого результата (ZF – *Zero Flag*), признак отрицательного результата (SF – *Sign Flag*), признак переполнения (OF – *Overflow Flag*) и признак переноса (CF – *Carry Flag*). Эти признаки устанавливаются в зависимости от типа операции и используются в операциях условных

сумма отрицательна и представлена в дополнительном коде – примеры 2 и 4 на рис. 4.17. Таким образом, признак (флаг) переполнения

$$OF = \overline{PnSm[n-1]} \cdot PnSm[n-2] \vee PnSm[n-1] \cdot \overline{PnSm[n-2]}.$$

Признак знака SF всегда повторяет значение старшего (знакового) разряда суммы $Sm[n-1]$. Признак нуля ZF устанавливается ($ZF = 1$) при равенстве всех бит числа нулю.

На рис. 4.18 представлена упрощенная структурная схема *регистрового АЛУ* для операций сложения и вычитания n -разрядных двоичных чисел с фиксированной точкой. Предполагается, что отрицательные числа хранятся в памяти в *дополнительном коде*.

В состав АЛУ входят комбинационная схема параллельного сумматора Sm , регистр сумматора $PzSm$, входные регистры PzB и PzA , входной регистр АЛУ $Pz1$. Операнды в АЛУ поступают из памяти по входной шине *ШИВх*: положительные числа в прямом коде, а отрицательные в дополнительном. Операнды размещаются в PzB (первое слагаемое или уменьшаемое) и $Pz1$ (второе слагаемое или вычитаемое); $Pz1$ связан с PzA цепями прямой и инверсной передачи кода. Прямая передача используется при операции алгебраического сложения, а инверсная – при операции вычитания.

При алгебраическом сложении поступившие в АЛУ коды операндов находятся на входных регистрах PzB и PzA сумматора. Код суммы формируется на выходах схемы Sm и фиксируется в регистре $PzSm$.

Операция алгебраического вычитания может быть сведена к изменению знака вычитаемого и операции алгебраического сложения. Для смены знака используются инверсные выходы регистра $Pz1$ и последующее подсуммирование 1 в младший разряд сумматора.

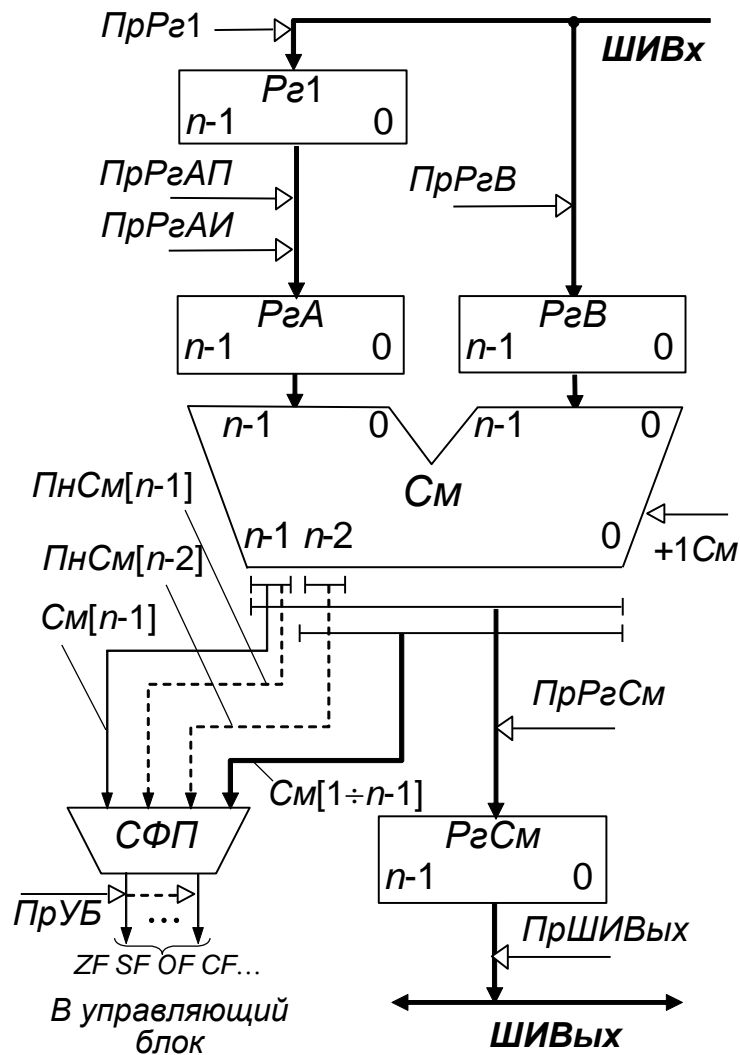


Рис. 4.18. Структурная схема АЛУ для сложения и вычитания двоичных чисел с фиксированной точкой

Свою работу АЛУ выполняет под управлением сигналов, называемых *микрооперациями*. На рис. 4.18 эти сигналы показаны именованными треугольными стрелками. Слово из R_{21} в R_{2A} может быть передано в прямом (управляющий сигнал PrR_{2AP}) или в инверсном (управляющий сигнал PrR_{2AI}) кодах. Результат из АЛУ выдается сначала в $R_{2См}$ (управляющий сигнал $PrR_{2См}$), а затем по сигналу $PrШИВых$ на выходную шину $ШИВых$ для сохранения в памяти. Признаки (флаги операций) ZF , SF , CF и OF формирует схема формирования признаков ($СФП$) на основании анализа состояния бит кода результата и бит переноса $ПнСм[n-1]$ и $ПнСм[n-2]$. Флаги операций передаются в управляющий блок и сохраняются в предназначенном для этого регистре по сигналу $PrУБ$.

Ниже приведено описание последовательности действий (*микропрограмма*) АЛУ при операциях сложения и вычитания. Каждая *микрооперация* в программе – это *управляющий сигнал*. В тексте микропрограммы управляющие сигналы и обозначения вызываемых ими

действий разделены двоеточиями. Комментарий выделен угловыми скобками, а отдельные операции разделены символами ';'.
'

ПрР2В: Р2В := ШИВх; <прием 1 операнда>
ПрР21: Р21 := ШИВх; <прием 2 операнда>
Прием: если сложение то ПрР2АП: Р2А := Р21 иначе
ПрР2АИ: Р2А := Р21;
Сумма: если сложение то ПрР2См: Р2См := Р2А + Р2В иначе
ПрР2См: 1См: Р2См := Р2А + Р2В + 1;
ПрУБ: если OF = 1 то прерывание иначе
ПрШИВых: ШИВых := Р2См; <выдача результата>
конец;

Микрооперация *ПрУБ* состоит в выдаче управляющему блоку кода признаков, которые могут быть использованы, например, для формирования запроса прерывания исполняемой вычислительным устройством программы в случае переполнения разрядной сетки.

5. Машины состояний. Микропрограммные автоматы

Регистровое АЛУ относится к числу цифровых автоматов, которые, как и все рассмотренные ранее последовательные функциональные узлы, находятся под воздействием управляющих сигналов. В случае с регистровым АЛУ эти сигналы принадлежат к числу тех, которые распространяются по цифровому устройству, в составе которого АЛУ работает. Такое устройство может выполнять конкретные (всегда одни и те же) вычислительные операции по обработке данных или быть *процессором (микропроцессором)*, выполняющим возложенные на него функции путем исполнения команд, реализующих ту или иную программу действий. Независимо от предназначения цифровое устройство, которым можно управлять с помощью внешних команд имеет по крайней мере две компоненты – *операционный блок и устройство управления*. Устройство управления воспринимает внешние команды и, превращая их в функциональные сигналы (микрокоманды) для операционного блока, управляет выполнением указанных в команде операций. Для этого требуется некоторое число тактов, в каждом из которых выполняется одна или несколько (при возможности параллельного исполнения) микроопераций.

Интервал времени, отводимый на одну микрооперацию, называют *рабочим тактом* или просто *тактом* цифрового устройства. Для приведенной в разделе 4.4 микропрограммы сложения АЛУ распределение управляющих сигналов по тактам выглядит следующим образом:

Такты	Управляющий сигнал (микрокоманда)
1	<i>ПрР2В</i>
2	<i>ПрР21</i>
3	<i>ПрР2АП</i> или <i>ПрР2АИ</i>
4	<i>ПрР2См</i> или <i>ПрР2См, +1См</i>
5	<i>ПрШИВых</i> и <i>ПрУБ</i>

Рассмотренные в предшествующих разделах цифровые устройства были разделены на два класса – на устройства, не имеющие памяти, и устройства ею обладающие. Наличие памяти в цифровой системе дает ей качественно новое свойство – возможность перехода в различные *контролируемые состояния*, что характеризует такую систему как *цифровой автомат*. При решении сложных алгоритмических задач возникает необходимость в цифровых автоматах, способных исполнять многочисленные и разнообразные инструкции. В этом случае число состояний цифрового автомата может достигать больших величин – вплоть до нескольких тысяч, что требует соответствующих математических моделей, пригодных для проектирования и технической реализации. В связи с этим рассмотрим цифровые автоматы с некоторых общих позиций, определив их как *машины состояний*, то есть машины, обладающие множеством

устойчивых состояний с детерминированной или управляемой внешними воздействиями системой переходов из одного состояния в другое.

Большинство систем, представляющих практический интерес, поддаются как наблюдению, так и управлению. Для наблюдения за системой используются ее выходы (выходные переменные $I = (i_n, \dots, i_2, i_1)$), а управление системой производится через ее входы (входные переменные $Q = (q_m, \dots, q_2, q_1)$). Один из возможных способов выявления внутренней структуры машины состоит в поиске отношения между входными и выходными переменными, или передаточной функции $H = I/Q$.

В цифровых электронных устройствах используется огромное число комбинаций все расширяющегося набора различных элементов. Это и простейшие логические элементы, и сложные комбинационные схемы (цепи), и последовательные функциональные узлы. Значительное количество вариантов реализации систем обуславливает развитие большого числа формальных моделей систем и средств их описания. Здесь мы продолжим рассмотрение последовательных автоматов в части их применения в цифровых процессорах. Главной особенностью таких систем является их способность воспринимать многочисленные и сложные по характеру команды, реакция на которые может выглядеть как длинный ряд переходов из одного состояния в другое.

Разделим структуры цифровых систем на классы в порядке возрастающей сложности.

5.1. Классификация машин состояния

Машины класса 0 (комбинационные схемы)

Если использовать понятие передаточной функции H , то машина простейшей структуры будет иметь передаточную функцию, выражаемую матрицей-константой H : $I = HQ$. В том случае, когда требуется подчеркнуть зависимость I от Q , вместо матричного представления используется, как это делалось нами ранее, представление в виде функции $I = f(Q)$. Здесь все наблюдаемые переменные являются функциями входных переменных, а состояние системы не является наблюдаемым. *Состояние системы* – это минимальная информация, которая необходима для предсказания ее поведения. К машинам класса 0 относятся все комбинационные схемы (КС).

Машины класса 1

Введем в рассмотрение независимую переменную t (время), которая может быть либо стохастической, либо детерминированной. Если наступление события в момент t вообще непредсказуемо, то соответствующую систему называют «*системой, управляемой событиями*», или «*событийной системой*». В этом случае t является временем наступления события. Если события

наступают регулярно (тактируются), то время t носит название «*времени перехода состояния*», или «*времени состояния*». Переход из начального состояния в конечное происходит в момент времени перехода состояния и отображается стрелкой \leftarrow , направленной из начального состояния в конечное. Таким образом, машину класса 1 можно представить уравнением

$$I \leftarrow f(Q),$$

которое показывает, что выходные переменные являются функциями входных переменных (сигналов), задержанных на время одного состояния. Такая машина является машиной с элементом задержки (памяти), который способен запоминать входные сигналы Q , соответствующие предыдущему времени состояния и затем выполнять функцию $f(Q)$ или, выполнив функцию $f(Q)$, сохранять ее значения в памяти для использования на следующем такте синхронизации. Простым примером машины с задержкой является регистр или синхронизируемый фиксатор.

Введение памяти и сигнала записи в память позволяет запоминать информацию и использовать ее впоследствии. Поэтому состояние машины может полностью или частично определяться предысторией процесса. Введение сигнала записи в память разделяет время состояния на два периода: переходный период и период стабильности. Сигналы на входе памяти должны оставаться неизменными при записи в память и могут изменяться тогда, когда сигнал записи в память не действует. Те сигналы, которые изменяются в течение периода записи, или стабильности, являются по отношению к системе *асинхронными*. *Синхронные системы* имеют равные времена состояний или синхронизируются внешним по отношению к системе источником сигнала. *Асинхронные машины* имеют время состояния, которое зависит только от внутренних задержек, связанных со структурой машины.

Машины класса 2

С помощью обратной связи информация о состоянии $X(t) = (x_k, \dots, x_2, x_1)$ машины может подаваться на вход машины состояний. Можно представить себе машину, у которой следующее состояние $X(t+dt)$ (dt – интервал между соседними временами состояния или период тактирования) и выходные переменные являются функциями только текущего состояния $X(t)$ (рис. 5.1). Примером машины класса 2 может служить двоично-десятичный счетчик (от двоичного отличается тем, что при счете по модулю 10 необходима обратная связь).

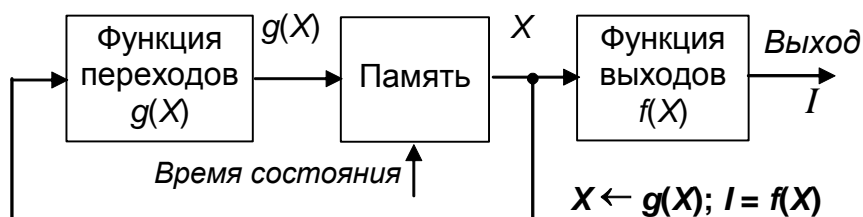


Рис. 5.1. Модель машины класса 2

Машины класса 3 (автомат Мура)

В машине класса 2 реализуется циклическая детерминированная последовательность состояний X при отсутствии внешних воздействий. *Машины класса 3* (рис. 5.2) являются обобщением машин класса 2. Они позволяют изменять последовательность состояний в ответ на входные сигналы – сигналы управления.

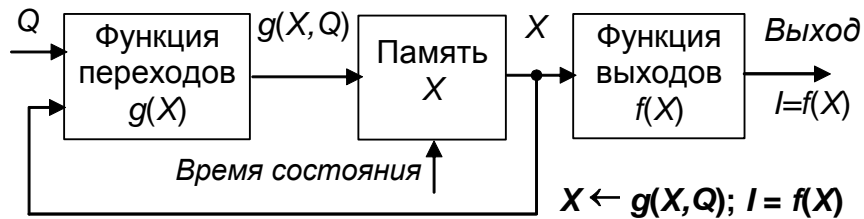


Рис. 5.2. Модель машины класса 3

Машины класса 4 (автомат Мили)

Обобщением машин класса 3 является машина, в которой входные сигналы являются функциями как сигналов управления $Q(t)$, так и текущего состояния $X(t)$ (рис. 5.3). В этом случае аргументами функции выходов являются не только состояние, но и вектор сигналов управления: $I = f(Q, X)$.

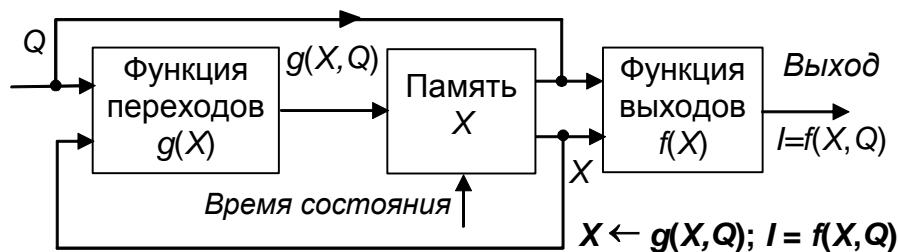


Рис. 5.3. Модель машины класса 4

5.2. Машины состояния и матричная логика

Функции отображения g и f реализуют, как правило, с помощью матричной логики (см. раздел 3.2). Эволюцию вектора состояния системы $X(t)$ можно описать матричным преобразованием

$$X(t+dt) = U(t+dt) X(t), \text{ или } X(t+1) = U(t+1) X(t). \quad (5.1)$$

Это преобразование определяет вектор состояния в момент времени $t+dt$, исходя из вектора состояния в момент времени t . С учетом дискретности времени состояния при периодическом тактировании ($dt = T = const$) проще

оперировать безразмерным временем со шкалой, равной периоду T , который можно принять за единицу (правое из выражений (5.1)).

Преобразование (5.1) можно представить как произведение матрицы U (матрицы вход–выход) на вектор-столбец X :

$$Y = U \cdot X, \quad \text{или} \quad y_i = \sum_{j=1}^k u_{ij} x_j.$$

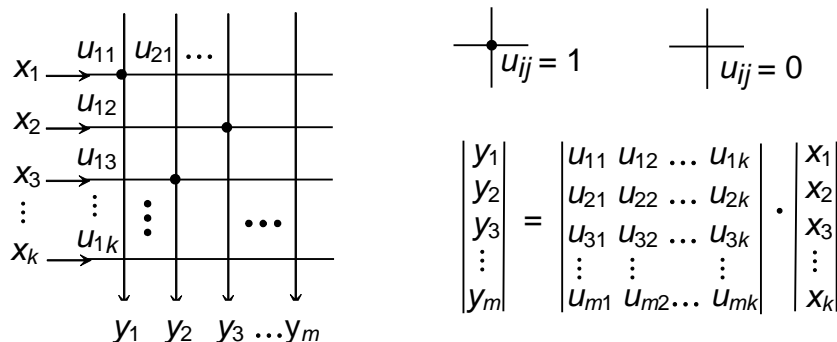


Рис. 5.4. Матричная логическая схема и эквивалентное преобразование с булевой матрицей

Матрица U является матрицей переключений, вид которой зависит от функции переходов g . При этом функцию переходов g , как и функцию выходов f , также можно представить в виде матрицы:

$$x_j(t+1) = \sum_{k=1}^m g_{jk} x_k(t); \tag{5.2}$$

$$i_j = \sum_{k=1}^n f_{jk} x_k.$$

Выражения (5.2) написаны в предположении, что следующее состояние машины $X(t+1)$ и ее выходные сигналы I зависят только от текущего состояния X , что соответствует машине класса 2.

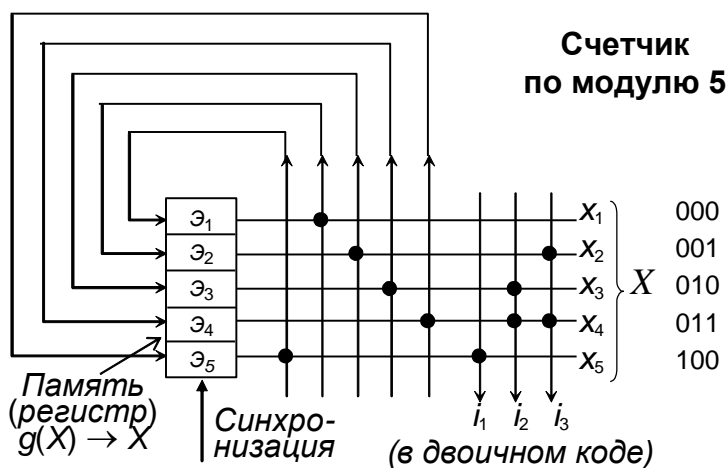


Рис. 5.5. Матричное представление преобразования текущего состояния $X(t)$ в следующее $X(t+1)$ и отображение состояния $X(t)$ на выходные переменные $I(t)$

В качестве примера рассмотрим счетчик по модулю $N = 5$ (рис. 5.5). Состояния счетчика будем кодировать унитарным кодом $X = (x_5, x_4, x_3, x_2, x_1)$, обозначающим одно состояние из N возможных. Если предположить, что вначале счетчик находился в состоянии $x_1 = 00001$, то на следующих тактах синхронизации он будет переходить в состояния $x_2 = 00010$, $x_3 = 00100$, $x_4 = 01000$ и $x_5 = 10000$, после чего вновь вернется в состояние $x_1 = 00001$, циклически продолжая эту цепочку переходов. В качестве выходной функции $I = f(X)$ взята функция, выполняющая преобразование унитарного кода в двоичный $I = (i_3, i_2, i_1)$.

Применение унитарного кода для кодирования N состояний машины требует N разрядов и, как следствие, N отдельных линий для передачи соответствующих сигналов по каждому из разрядов, что заведомо избыточно, поскольку кодируя состояния, например, двоичным кодом, можно уменьшить это число до L , удовлетворяющего условию $2^L \geq N$. Переход к более экономному способу кодирования позволяет сократить разрядность матрицы вход-выход. С этим связана одна из проблем, возникающих при реализации и программировании подобных систем – проблема выбора между двумя альтернативами *горизонтальным и вертикальным программированием*.

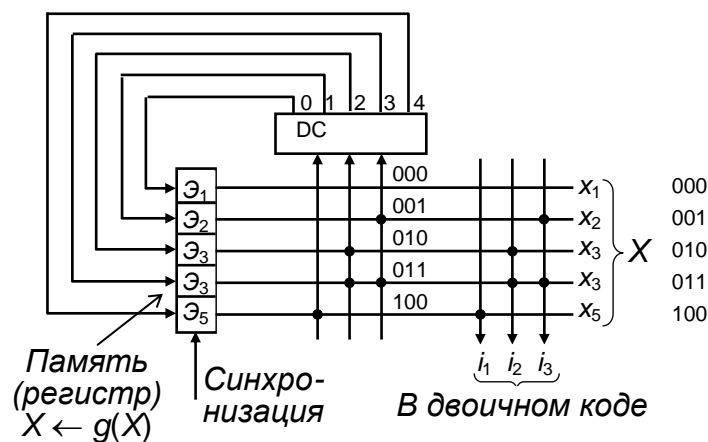


Рис. 5.6. Счетчик по модулю 5 с двоичным кодированием вектора состояний X

Если состояния кодируются в сжатой форме, то для передачи «сжатой» информации о состоянии требуется меньшее число линий, но перед использованием кодированная информация должна быть декодирована, т.е. развернута в «вертикальном» (по расположению элементов памяти \mathcal{E}_1 - \mathcal{E}_N) направлении. В счетчике по модулю 5 это делается так, как показано на рис. 5.6. Но здесь есть и элемент горизонтального программирования, поскольку для сжатого кода требуются строки меньшей размерности в матрице вход-выход.

Термин «горизонтальное программирование» обычно применяется для указания на отсутствие кодирования множества состояний 1 из N и использования отдельной линии для передачи каждого из состояний. Однако если все сигналы в линиях состояния являются взаимоисключающими, или ортогональными, в том смысле, что никогда не появляются одновременно, то при горизонтальном программировании для уменьшения числа линий может быть использовано кодирование состояний.

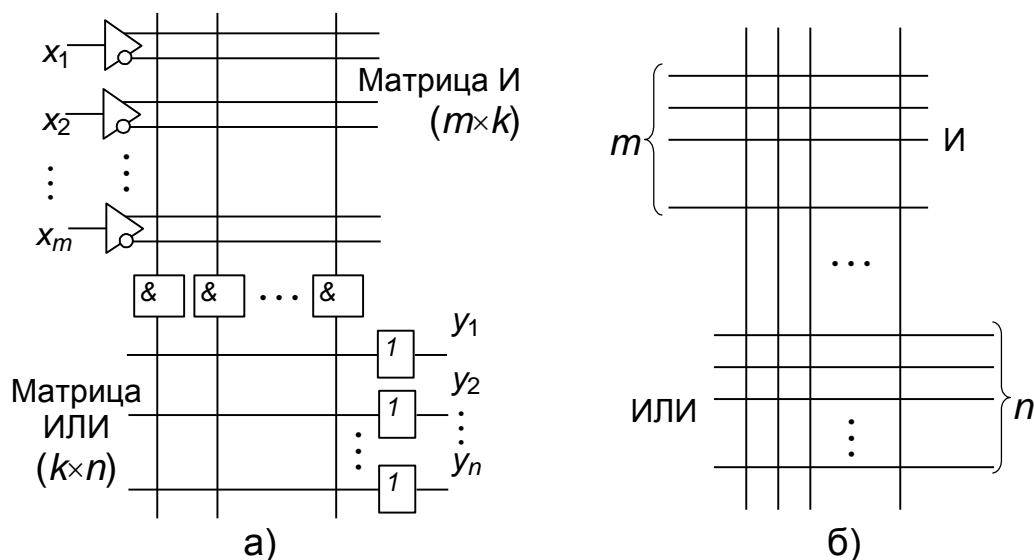


Рис. 5.7. Логическая схема двухуровневой матрицы И-ИЛИ (а) и ее упрощенное обозначение (б)

Универсальным средством реализации матрицы вход–выход являются программируемые логические матрицы (см. раздел 3.2). Не вдаваясь в детали технического исполнения, будем пользоваться двухуровневой логической схемой ПЛМ (рис. 5.7а) и ее упрощенным обозначением, которое представлено на рис. 5.7б. Входные переменные $X = (x_m, \dots, x_2, x_1)$ и их инверсные значения подаются на матрицу И. Для этого в схеме имеются элементы с двумя (прямым и инверсным) выходами. Выходные переменные $Y = (y_n, \dots, y_2, y_1)$ образуются на выходах матрицы ИЛИ.

Для построения цифровых автоматов удобно использовать микросхемы ПЛМ с памятью, которые помимо комбинационной части содержат на кристалле триггеры (регистры) обычно типа D (рис. 5.8). Структура такой ПЛМ совпадает с канонической схемой автомата Мура.

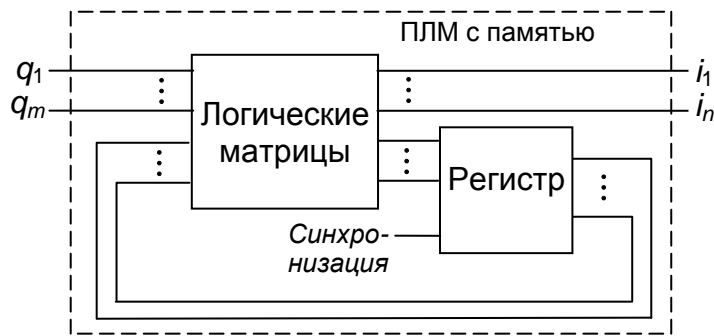


Рис. 5.8. Структура ПЛМ с памятью

Промышленные образцы логики на программируемых матрицах появились в виде микросхем ПМЛ, в которых выходы матрицы И жестко распределены между элементами ИЛИ (входами матрицы ИЛИ). Структурная схема одной из первых отечественных ПМЛ с памятью (микросхема КР1556ХП4) представлена на рис. 5.9. Элементами памяти в ней являются триггеры типа D. Матрица представляет логику первого уровня, на котором образуются термы входных переменных. Второй уровень – матрица ИЛИ, состоящая из 8 дизъюнкторов (четырех 7-входовых и четырех 8-входовых). Выходные буферы выполнены по схеме с тремя состояниями. Четыре D-триггера имеют управление от положительного перепада внешнего синхросигнала С. Сигнал ОЕ управляет буферами, подключенными к выходам триггеров.

В микросхемах более позднего времени распределение конъюнкторов между элементами ИЛИ стало варьироваться, что позволило на одном кристалле выполнять разные по сложности функции.

Продолжением линии ПМЛ стали БИС/СБИС CPLD (Complex Programmable Logic Devices). Напомним, что наряду с ПЛМ и ПМЛ имеется линия базовых матричных кристаллов (БМК), продолжением которой стали кристаллы FPGA (Field Programmable Gate Array). Стремление объединить достоинства обеих линий привело к созданию БИС/СБИС смешанной (комбинированной) архитектуры, для которых еще не выработано общепринятое название (фирма Altera, например, пользуется названием FLEX (Flexible Logic Element Matrix) – гибкие). Рост уровня интеграции дол возможность размещать на кристалле схемы, сложность которых соответствует целым системам. Эти системы именуется SOC (System on Chip).

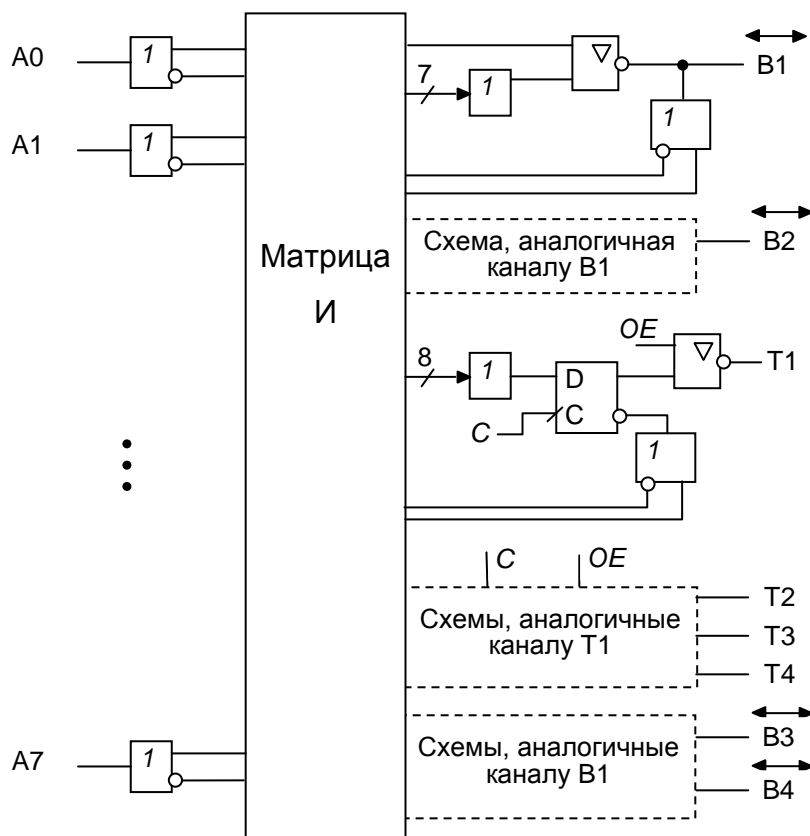


Рис. 5.9. Структура ПМЛ КР155ХЛ8

Логическая схема машины класса 3 (рис. 5.10) получается из схемы машины класса 2 добавлением поля внешних сигналов $Q = (q_m, \dots, q_2, q_1)$ в матрицу функции переходов. В этом случае функция переходов

$$g(X, Q) = g(X) + HQ,^{11} \quad (5.3)$$

а логика перехода из текущего состояния в следующее определяется по выражению

$$x_j(t+1) = \sum_{l=1}^k g_{jl} x_l(t) + \sum_{l=1}^m h_{jl} q_l. \quad (5.4)$$

¹¹ Под операцией «+» в (5.3) и (5.4) следует понимать операцию логического сложения.

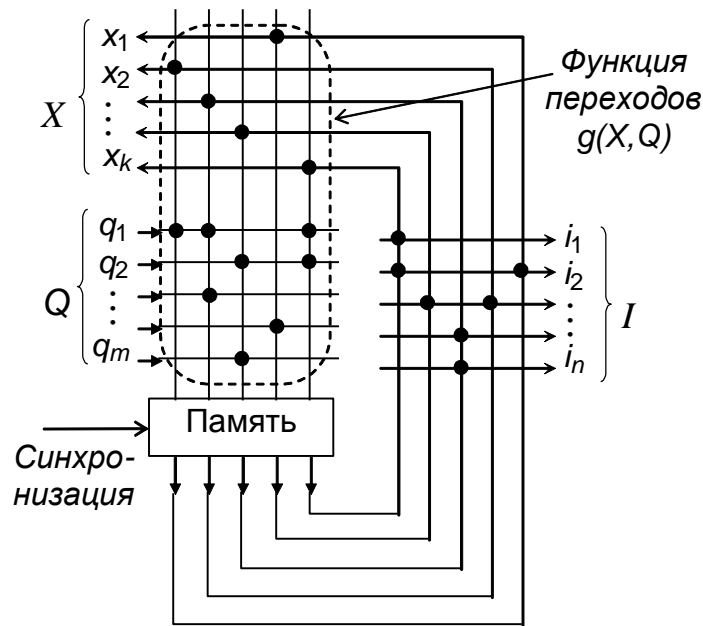


Рис. 5.10. Представление машины класса 3 матричной логической схемой

В выражении (5.3) присутствует квалификатор входа H , с помощью которого можно отключать воздействие сигналов управления Q . В присутствии сигналов Q (матрица H не является нулевой) следующее состояние машины зависит от текущего состояния $X(t)$ и от вектора Q . Если в матрице H все $h_{ji} = 0$, то переход в следующее состояние зависит только от текущего состояния.

Аналогичным образом путем добавления поля Q в матрицу выходной функции $I = f(X, Q)$ строится логическая схема машины класса 4.

5.3. Микропрограммирование и устройство управления выполнением программы

Машины состояний классов 3 и 4 представляют собой универсальные машины, предназначенные для управления процессом обработки данных путем последовательного выполнения отдельных операций, но во многих случаях более простым и удобным оказывается использование специализированных устройств – *устройств управления выполнением программы (УУВП)*. Программирование таких устройств называют *микропрограммированием*, и на нем основано создание промышленных образцов устройств управления выполнением программы.

5.3.1. Архитектура и реализация

Наиболее приемлемое определение *архитектуры ЭВМ* подразумевает *программную модель* машины, т. е. ее представление с точки зрения

программиста. Эта модель включает такие элементы, как регистры, команды передачи данных между регистрами и операций с данными. Архитектура в принципе не зависит от реализации системы; одна и та же архитектура может быть реализована на разной элементной базе. Еще в начале 50-х годов XX столетия было высказано предположение, что управляющие блоками обработки данных сигналы целесообразно хранить в специальной *управляющей памяти*, содержимое которой определяется на стадии проектирования устройства управления и остается неизменным в процессе его функционирования. Процесс разработки алгоритма работы УУВП и создания кода для управляющей называют *микропрограммированием*, а получающееся в результате устройство управления – *микропрограммным автоматом*. Микропрограммный автомат является неотъемлемой составной частью *процессоров* (микропроцессоров, в частности) со сложным набором команд (о классификации процессоров по характеру исполняемых ими команд см. раздел 7). Одна из проблем, возникающих при реализации подобных систем, связана, как уже говорилось выше, с двумя подходами к микропрограммированию – с *горизонтальным и вертикальным программированием*.

При рассмотрении сложных автоматов, способных выполнять различные и многочисленные инструкции нужно учитывать то, что для каждого $X(t)$ существенными являются значения не всех, а лишь некоторых компонент в векторе внешнего воздействия $Q(t)$, задающих переходы из состояния $X(t)$ в состояние $X(t+1)$. Число таких переходов в микропрограммных автоматах обычно невелико, и поэтому для каждого $X(t)$ можно выделить только ему свойственную группу переходов в следующее состояние $X(t+1)$. Таким образом, все множество состояний можно разбить на группы, исключив при этом ту их часть, которая многократно повторяется. В этом случае *каждой группе состояний* можно выделить *отдельную область* в поле кода состояний и о такой системе говорят, что в ней использовано *вертикальное микропрограммирование*. Обусловлено это тем, что микропрограмма при выполнении определяемых вектором $Q(t)$ действий реализуется как последовательность переходов от одной группы состояний к другой и последовательность этих переходов, а также их число зависят от того, что заложено в управляющее воздействие $Q(t)$.

микропрограммного автомата. Возможны различные реализации подобной структуры, отличающиеся в основном интерпретацией входа Q.

Без учета входа Q система сводится к машине класса 2, т. е. к машине, выполняющей связанный список микрокоманд (рис. 5.12а). При этом микрокоманды могут исполняться совершенно в ином порядке, чем тот, который предписан их расположением в памяти.

Внешние сигналы Q, если они действуют, влияют на выбор адреса следующей команды и таким образом активизируют те участки микропрограммного кода в ППЗУ, которые необходимы для реализации действия Q.

Совокупность внешних управляющих сигналов Q отвечает за тот набор команд, посредством которого программируется работа процессора с данным УУВП. Предназначенная для центрального процессора программа хранится во внешней по отношению к ЦП памяти (в оперативной или постоянной памяти вычислительной системы) и в виде команд (инструкций) поступает на вход Q микропрограммного автомата. В дальнейшем затребованная в соответствии с кодом Q операция реализуется как одна микрооперация или последовательность нескольких микроопераций. Вырабатываемые УУВП функциональные сигналы – это команды самого низкого уровня. В виде электрических сигналов они распределяются на все входящие в ЦП и требующие управления устройства и, в частности, в операционный(е) блок(и).

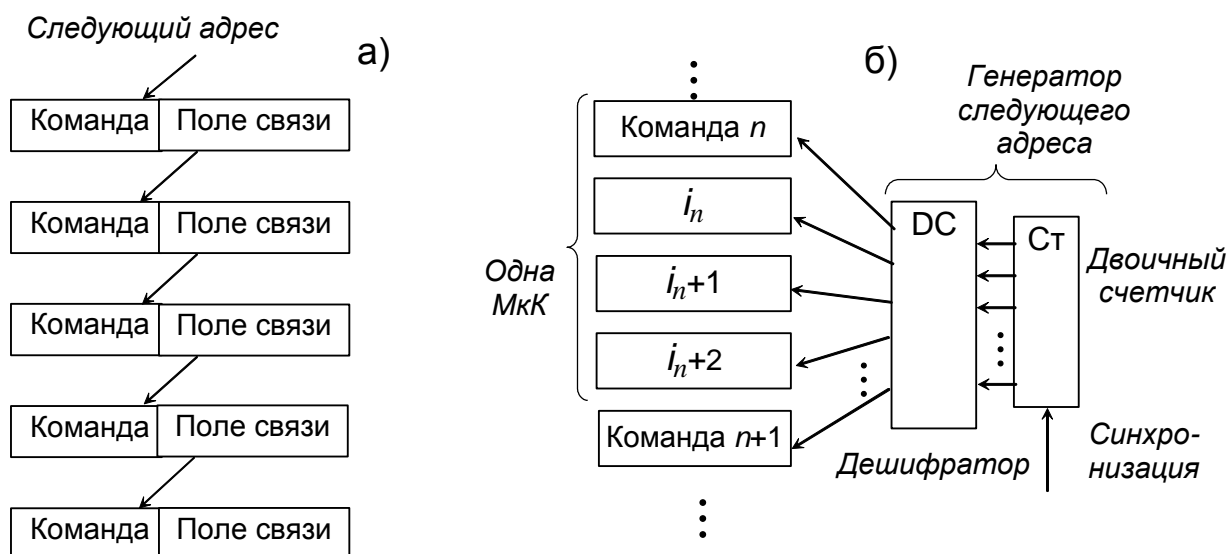


Рис. 5.12. Структура системы управления выбором следующего состояния, выполненная с использованием указателя (а) и счетчика микрокоманд (б)

Специфика горизонтального и вертикального микропрограммирования накладывает различия на способ формирования управляющих функциональных сигналов. При горизонтальном программировании каждому разряду операционной части микрокоманды $I = (i_n, \dots, i_2, i_1)$ ставится в соответствие определенная микрооперация. При таком способе операционная часть МкК

содержит n разрядов, где n – общее число микроопераций. Достоинствами горизонтального микропрограммирования являются возможность одновременного выполнения в одном такте синхронизации любого набора из n микроопераций и простота формирования функциональных сигналов. Существенным недостатком является то, что требуется большая длина микрокоманды, поскольку функциональные сигналы могут быть достаточно многочисленны.

При вертикальном микропрограммировании микрооперация определяется не состоянием одного из разрядов микрокоманды, а двоичным кодом, содержащимся в операционной части МкК. Достоинством вертикального микропрограммирования является небольшая длина МкК. Однако в этом случае на большое количество микроопераций требуются сложные дешифраторы, а главное – в каждой микрокоманде указывается лишь одна микрооперация, что приводит к увеличению длины микропрограмм по сравнению с их длиной при горизонтальном программировании.

Наибольшее распространение имеет *смешанное микропрограммирование* как сочетание горизонтального и вертикального микропрограммирования.

В практических применениях значительная часть всего множества состояний микропрограммного автомата выглядит как последовательно зависящая цепь. Если говорить на языке микропрограммирования, то это (по аналогии с высокоуровневым программированием) можно связать с последовательным выполнением значительной части микропрограммного кода, в том числе и при ветвлениях, циклических повторениях (циклах), вызовах микропроцедур и т. д. В таких случаях ссылку на следующую микрокоманду можно формировать с помощью двоичного счетчика (рис. 5.12б) и тем самым значительно сократить объем управляющей памяти. Перед входом в каждую последовательную часть микропрограммы в счетчике устанавливается соответствующее значение адреса, которое затем линейно наращивается.

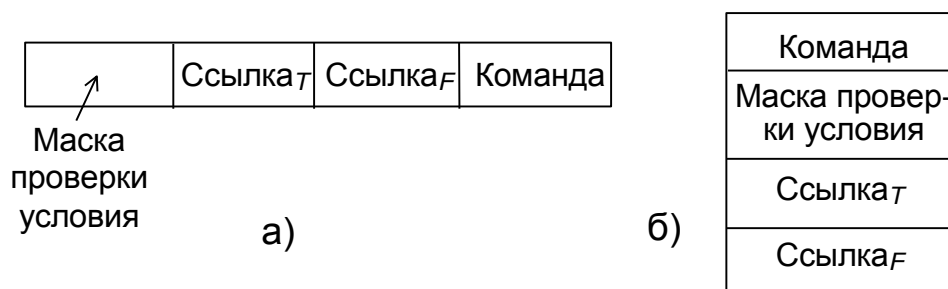


Рис. 5.13. Формат микрокоманды: (а) – горизонтальное микропрограммирование; (б) – вертикальное микропрограммирование

Важным вопросом является *формат микрокоманды*, а также ее расположение в памяти. Появление различных систем адресации обусловлено стремлением к компромиссу между обеспечением требований минимизации времени обработки и минимизации памяти. На рис. 5.13 представленные две структуры, демонстрирующие различие в представлении микрокоманд при

горизонтальном и вертикальном микропрограммировании. В первом случае различные биты микрокоманды расположены в одном слове управляющей памяти и используются одновременно для выполнения указанных в микрокоманде действий. Во втором случае необходимы меньшие по размеру ячейки памяти, но те же самые действия реализуются последовательно, что требует большего числа машинных тактов для выполнения заданной операции.

В представленном на рис. 5.13 формате микрокоманд имеются четыре поля. Поле «Команда» предназначено для указания микроопераций (функциональных сигналов). Два поля – «Ссылка_T» и «Ссылка_F» содержат два указателя на следующую микрокоманду, один из которых используется, когда обозначенное в поле «Маска проверки условия» условие выполнено (*true*), другой – при невыполнении (*false*) этого условия. Это один из способов, предусматривающий возможность ветвлений при выполнении микропрограммы.

5.3.2. Структуры адресации памяти микропрограмм

Представленная на рис. 5.10 машина класса 3 обладает обобщенной структурой формирования следующего адреса микрокоманды, которая позволяет совместно использовать *квалификатор входа Q* и функцию $g(X)$, определяющую следующее состояние (рис. 5.14). Последнюю можно модифицировать так, чтобы на дешифратор адреса поступали данные либо от первого, либо от второго из указанных источников (но не от обоих вместе). Такую систему выбора следующего адреса называют *ортогональной*. Реализуется она с помощью мультиплексора, для управления которым используются специальные линии (или линия), по которым (или которой) передаются сигналы из управляющей памяти (рис. 5.15).

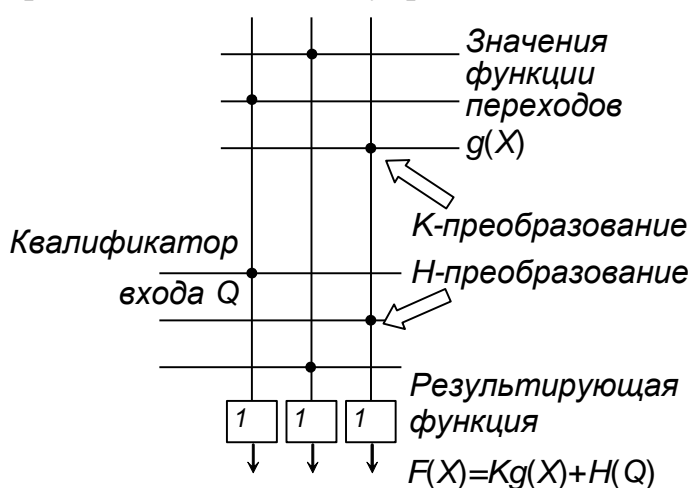


Рис. 5.14. Выбор следующего состояния по обобщенной схеме

В этом случае сигналы Q от внешнего источника могут инициировать выполнение определенного управляющего слова (микрокоманды), которое в нужное время откроет «внутренний» канал мультиплексора, после чего последовательностью выполнения микрокоманд будет управлять находящаяся в самих МКК информация. По завершении выполнения этой последовательности вновь откроется «внешний» канал

мультиплексора, делая микропрограммный автомат восприимчивым к внешним сигналам Q .

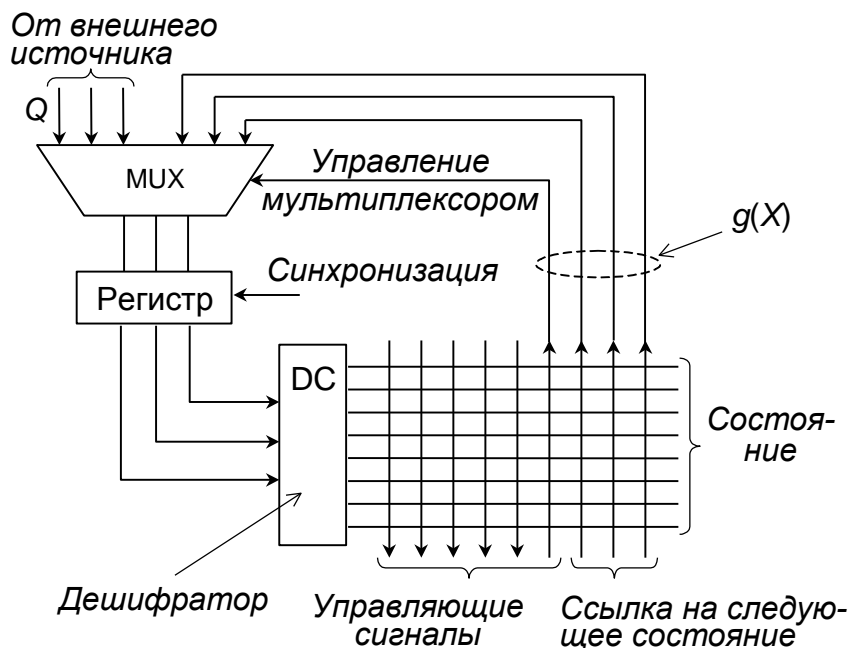


Рис. 5.15. Машина класса 3 с ортогональной структурой системы выбора следующего состояния

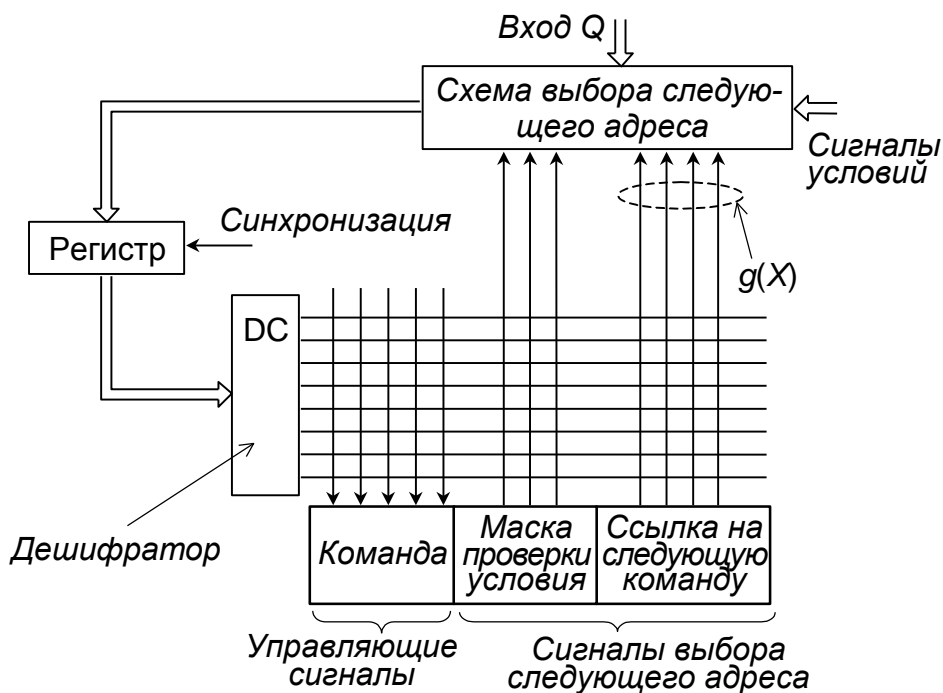


Рис. 5.16. Универсальная машина с обобщенной логической схемой выбора следующей микрокоманды

Мультиплексором в тракте выбора адреса следующей МкК управляет специальное поле микрокоманды с числом бит, зависящим от числа

переключаемых каналов мультиплексора. Это поле, называемое *полем проверки условий* (условий переключения), в явном виде показано на рис. 5.16. Поле проверки условий определяет выбор либо входов Q , либо входов сигналов условий от внешних источников, которые, как и содержащееся в микрокоманде поле ссылки, используются при формировании адреса следующей МкК.

Формировать последовательность адресов можно с помощью счетчика или другого подобного устройства (рис. 5.17), поскольку при последовательной выборке микрокоманд следующий адрес равен текущему адресу плюс 1. Однако для данной цели лучше подходит устройство приращения на 1, т. к. реализовать его можно на базе комбинационных схем, благодаря чему схема формирования следующего адреса из текущего становится более простой. Регистр задержки, располагаемый на выходе устройства приращения, обычно называют счетчиком микропрограммы, или счетчиком микрокоманд (СМК).

Выходные сигналы устройства приращения должны передаваться по каналу обратной связи и использоваться в качестве потенциального источника адреса. Для этой цели используется канал *СМК* мультиплексора (рис. 5.17).

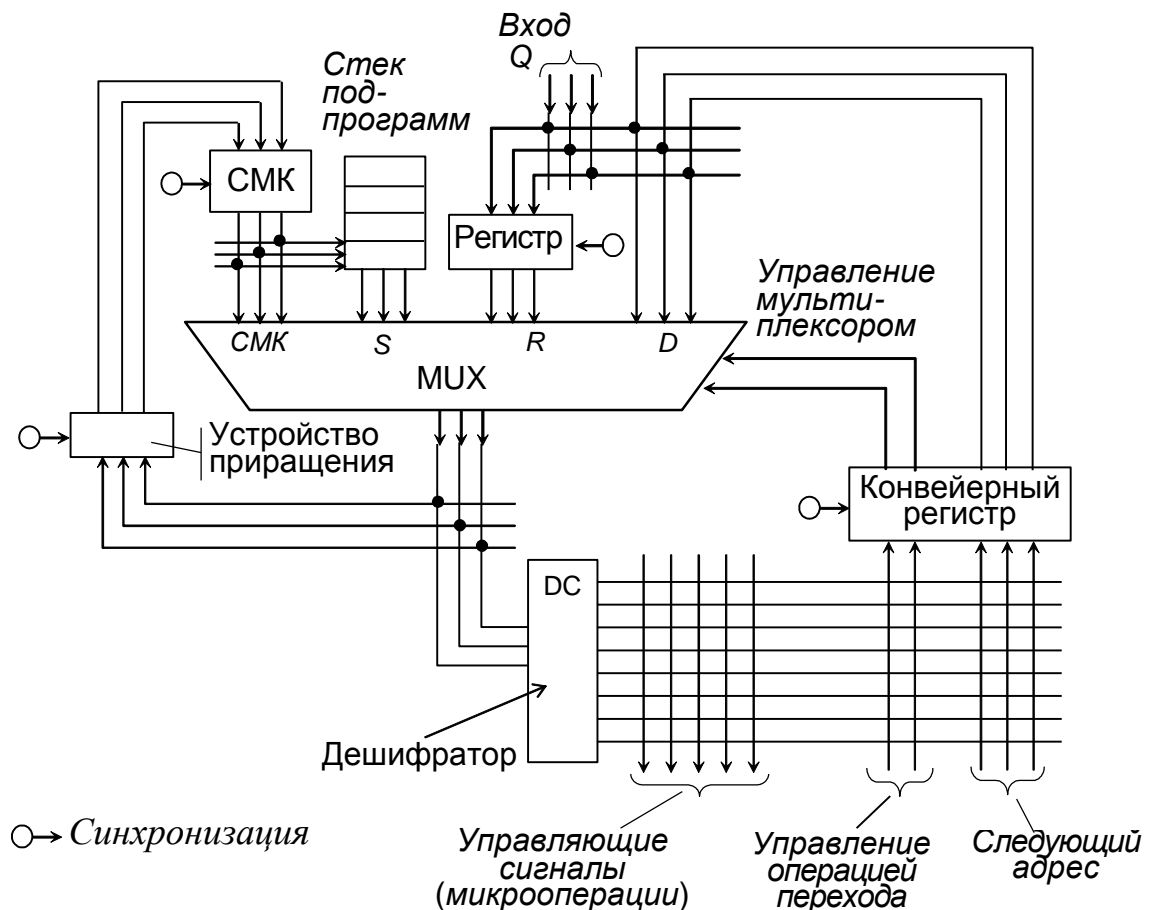


Рис. 5.17. Архитектура системы управления выполнением последовательности микрокоманд, включающей микропрограммный счетчик и стек адресов возврата

Использование микропрограммного счетчика позволяет значительно уменьшить объем памяти микропрограммы вследствие сокращения формата

микрокоманд, а также за счет возможности определения адреса следующей МкК по умолчанию (посредством автоматического наращивания СМК).

Инструкции от внешнего источника действуют через канал R мультиплексора. Сигналы на линиях входа Q мультиплексора и линиях, через которые в мультиплексор передается содержимое поля следующего адреса микрокоманды, должны быть ортогональными, а сами линии должны управляться по схеме с тремя состояниями так, чтобы всегда один из этих источников адреса находился в состоянии высокого выходного импеданса. В этом состоит один из элементов ортогональной системы выбора адреса следующей микрокоманды.

В схемах на рис. 5.10, 5.11, 5.15 и 5.16 присутствует один регистр, в котором содержится адрес следующей микрокоманды. В схеме на рис. 5.17 функцию этого регистра выполняют *конвейерный регистр* и регистр канала R мультиплексора. Если ссылка на следующую МкК находится в самой микрокоманде, то эта ссылка передается через вход D мультиплексора. Внешняя команда Q воспринимается через регистр, подключенный к входу R . В совокупности два входа мультиплексора (R и D) и два регистра (включая конвейерный регистр) реализуют ортогональную систему выбора адреса следующей микрокоманды, так как это показано на рис. 5.15. Дополнительная связь подключенных в входу D мультиплексора линий следующего адреса с регистром на входе R позволяет сохранять в последнем состоянии линий следующего адреса, что бывает необходимо, например, при выполнении циклически повторяющихся микрокоманд (циклов). При входе в цикл в регистре R сохраняется адрес первой микрокоманды тела цикла, что позволяет вернуться к ее исполнению, если мультиплексор будет переключен на вход R .

При последовательной выборке микрокоманд счетчику микропрограмм обеспечивается присвоение значения адреса МкК по умолчанию путем автоматического наращивания его содержимого и переключения входа мультиплексора на канал $СМК$. С помощью счетчика реализуются также требуемые при исполнении микропрограммы переходы и ветвления.

Вызов подпрограмм может быть реализован, если предусмотреть специальные средства для запоминания предполагаемого по умолчанию адреса следующей МкК при переходе на подпрограмму. Этим обеспечивается возврат к исполнению программы, при исполнении которой произошел вызов подпрограммы. Наиболее предпочтительной структурой для хранения адресов возврата из подпрограмм является показанный на рис. 5.17 набор ячеек памяти, обслуживаемый по принципу LIFO (*Last-In-First-Out*): *последний вошел – первый вышел*. Эту структуру называют *стеком* – в данном случае *стеком адресов возврата из подпрограмм*. При возврате из подпрограммы вход мультиплексора переключается на канал S .

6. Запоминающие устройства

Запоминающие устройства (ЗУ) служат для хранения информации и обмена ею с другими цифровыми устройствами (ЦУ). Системам памяти свойственна многоступенчатая иерархическая структура, и в зависимости от роли того или иного типа ЗУ его реализация может быть существенно различной. В иерархии памяти можно выделить следующие уровни:

- *регистровые ЗУ*, находящиеся в составе процессора или других устройств, благодаря которым уменьшается число обращений к другим уровням памяти, реализованным вне процессора и требующим большего времени для операций обмена информацией;
- *кэш-память*, служащая для хранения копий информации, используемой в текущих операциях обмена. Высокое быстродействие кэш-памяти повышает производительность ЭВМ;
- *основная память* (оперативная, постоянная, перепрограммируемая), работающая в режиме непосредственного обмена с процессором и по возможности согласованная с ним по быстродействию. Исполняемый в текущий момент фрагмент программы обязательно находится в основной памяти;
- *специализированные* виды памяти, характерные для некоторых специфических архитектур (многопортовые, ассоциативные, видеопамять и др.);
- *внешняя память*, хранящая большие объемы информации. Эта память обычно реализуется на основе устройств с подвижными носителями информации (магнитные и оптические диски, магнитные ленты и др.).

Основная память является *адресной*. Организована она в виде массива запоминающих ячеек. Наряду с адресной памятью в современных ЦУ широкое применение находит память *ассоциативная* и *стековая*.

В *ассоциативной памяти* поиск нужной информации производится не по адресу, а по ее содержанию (по ассоциативному признаку). При этом поиск по ассоциативному признаку происходит параллельно во времени для всех ячеек запоминающего массива. Во многих случаях ассоциативный поиск позволяет существенно ускорить обработку данных. Поэтому ассоциативная память применяется в кэш-ЗУ.

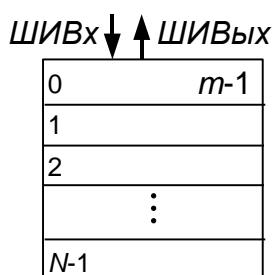


Рис. 6.1. Стековая память

Стековая память, так же как и ассоциативная, является безадресной. В стековой памяти ячейки образуют одномерный массив, в котором соседние ячейки связаны друг с другом разрядными цепями передачи слов (рис. 6.1). Запись нового слова производится в верхнюю ячейку (ячейку 0), при этом все ранее записанные слова (включая слово в ячейке 0) сдвигаются вниз в соседние ячейки с большим на 1 номерами. Считывание возможно только из

верхней (нулевой) ячейки памяти, при этом все остальные слова в памяти сдвигаются вверх в соседние ячейки с меньшими номерами. В этой памяти порядок считывания слов соответствует правилу *LIFO*: *последним поступил – первым обслуживается*. Иногда стековая память снабжается счетчиком стека *СчСт*, показывающим количество занесенных в память слов. Значение счетчика $СчСт = 0$ соответствует пустому стеку, а $СчСт = N-1$ – заполненному стеку.

Часто стековую память организуют используя адресную память. При этом на вершину стека – на адрес верхней ячейки стековой памяти – показывает специальный регистр, называемый *указателем стека* (*Stack Pointer – SP*). В этом случае при занесении информации в стек (или извлечении ее из стека) физического перемещения содержимого ячеек стековой памяти не происходит, а изменяется лишь содержимое *SP*. Например, *SP* может показывать на свободную ячейку в вершине стека. Тогда после записи в нее содержимое *SP* уменьшается на 1, чтобы вновь показывать на свободную ячейку. При извлечении из стека, напротив, содержимое указателя *SP* сначала увеличивается на 1, а потом происходит считывание из ячейки по вновь образовавшемуся адресу. В рассмотренном случае вершина стека перемещается в область меньших адресов. Однако используя указатель стека можно организовать стек, наращиваемый в область старших адресов памяти. При этом *SP* может показывать как на свободную ячейку стека (случай, рассмотренный выше), так и на последнюю занятую.

Говоря о ячейках памяти, необходимо учитывать их размерность, т. е. число двоичных разрядов (бит) n , которые они занимают. Для разных систем разрядность ячеек памяти различна. Наиболее широкое применение находит 8-разрядная организация основной памяти, в которой отдельная ячейка соответствует одному *байту*, т. е. имеет $n = 8$. При этом используемые в памяти микросхемы могут строиться как однобитовые или многобитовые структуры. При использовании малоразрядных микросхем необходимое число разрядов в ячейках основной памяти получают путем параллельного соединения адресных входов отдельных микросхем и объединения их информационных линий данных в соответствии с числом разрядов n .

В последующих разделах описываются различные типы адресных ЗУ, поскольку они являются основой для построения других типов запоминающих устройств. Работа адресного запоминающего устройства характеризуется рядом временных параметров, поскольку ЗУ управляется сигналами со строго определенным расположением во времени. Один из возможных наборов сигналов ЗУ (рис. 6.2) составляют

- $A = a_{n-1} \dots a_1 a_0$ – адрес, разрядность которого n зависит от числа ячеек ЗУ, точнее от максимально возможного числа хранимых в ЗУ слов;
- \overline{CS} (*Chip Select – активен низкий уровень*) – запрещает работу данной микросхемы;
- R/\overline{W} (*Read/Write*), задающий операцию чтения или записи;

- *DI* и *DO* (*Data Input* и *Data Output*) – сигналы на входных и выходных шинах данных, разрядность которых m определяется организацией ЗУ (разрядностью его ячеек). В некоторых ЗУ эти линии объединены.

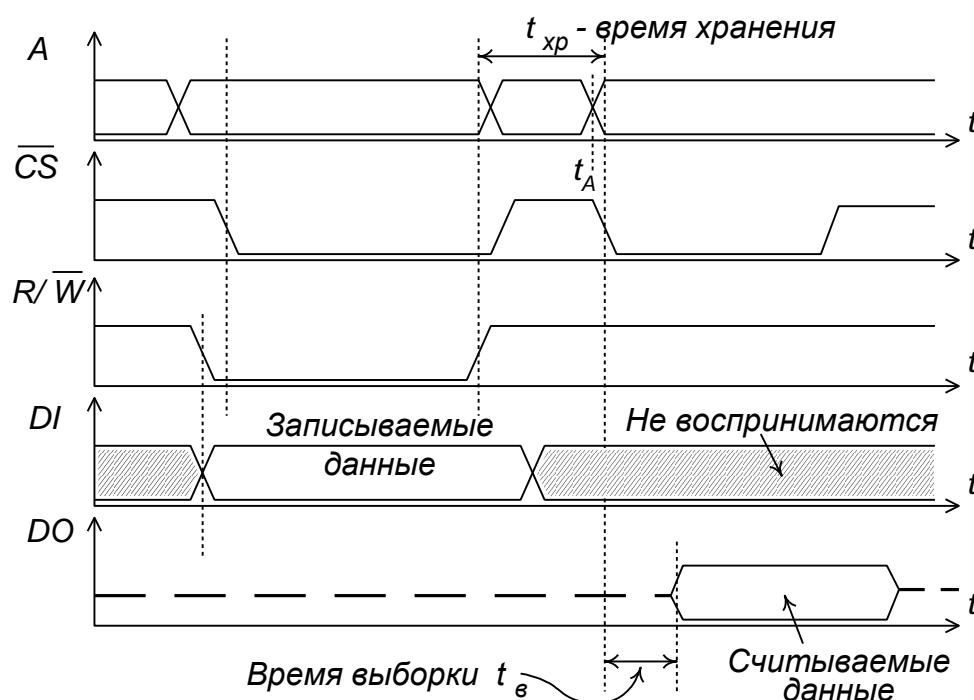


Рис. 6.2. Цикл записи и цикл чтения ЗУ

При обращении к ЗУ прежде всего подается адрес, чтобы последующие операции не коснулись какой-либо другой ячейки, кроме выбранной. Затем сигналом \overline{CS} разрешается работа выбранной микросхемы. При записи в ЗУ записываемые данные должны быть установлены до появления активного (низкого) уровня stroba чтения/записи R/\overline{W} . В цикле чтения после подачи сигнала \overline{CS} ЗУ готовит данные для чтения. Это требует некоторого времени, поэтому данные считываются с задержкой на время выборки t_s , длительность которого определяется тем, насколько быстро память отреагирует на обращение к ней. Время выборки фактически определяет *время доступа* к ЗУ – время до появления информационного сигнала на выходе с момента t_A установки адреса.

Между циклами чтения и записи запоминающее устройство находится в режиме хранения и время t_{xp} пребывания в этом режиме зависит от того, как делаются обращения к ЗУ.

6.1. Основные структуры адресных запоминающих устройств

Адресные ЗУ делятся на статические и динамические оперативные запоминающие устройства (ОЗУ, или *Random Access Memory* – RAM) и память

типа постоянных ЗУ (ПЗУ, или *Read-Only Memory* – ROM). Многочисленные варианты этих ЗУ имеют много общего с точки зрения структурных схем. Поэтому рационально их представление в виде обобщенных структур с последующим описанием запоминающих элементов для различных ЗУ.

Общность структур особенно проявляется для статических ОЗУ и памяти типа ROM. Структуры динамических ОЗУ имеют свою специфику и рассмотрены в Разделе 6.3. Для статических ОЗУ и памяти типа ROM наиболее характерны структуры 2D, 3D и 2DM.

Структура 2D

В структуре 2D (рис. 6.3) запоминающие элементы ЗЭ организованы в прямоугольную матрицу размерностью $M = k \times m$, где M – информационная емкость памяти в битах; k – число хранимых слов; m – их разрядность.

Дешифратор адресного кода DC при наличии разрешающего сигнала CS (*Chip Select* – сигнала выбора микросхемы) активизирует одну из выходных линий, разрешая одновременный доступ ко всем элементам выбранной строки, хранящей слово, адрес которого соответствует номеру строки. Элементы одного столбца соединены вертикальной линией – внутренней линией данных (разрядной линией, линией записи/считывания). Элементы столбца хранят одноименные биты всех слов. Направление обмена определяется усилителями чтения/записи под воздействием сигнала R/W (*Read* – чтение, *Write* – запись).

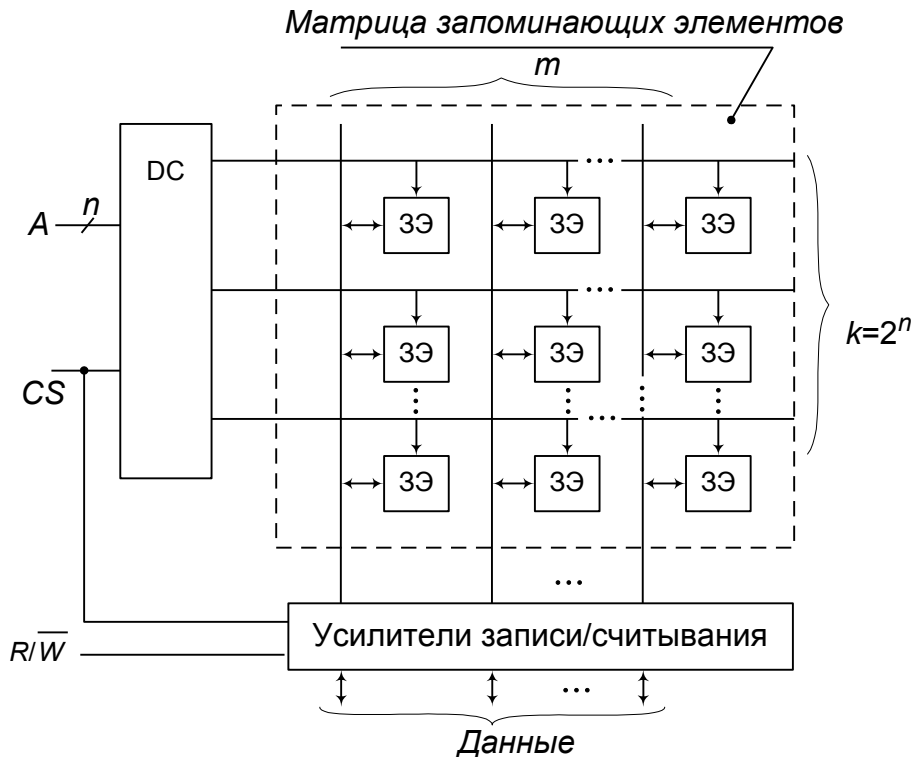


Рис. 6.3. Структура ЗУ типа 2D

Дешифратор адресного кода DC при наличии разрешающего сигнала CS (*Chip Select* – сигнала выбора микросхемы) активизирует одну из выходных линий, разрешая одновременный доступ ко всем элементам выбранной строки, хранящей слово, адрес которого соответствует номеру строки. Элементы одного столбца соединены вертикальной линией – внутренней линией данных (разрядной линией, линией записи/считывания). Элементы столбца хранят одноименные биты всех слов. Направление обмена определяется усилителями чтения/записи под воздействием сигнала R/\overline{W} (*Read* – чтение, *Write* – запись).

Структура типа 2D применяется лишь в ЗУ малой информационной емкости, т. к. при росте емкости проявляется несколько ее недостатков, наиболее очевидный из которых – чрезмерное усложнение дешифратора адреса (число выходов дешифратора равно числу хранимых слов).

Структура 3D

Структура 3D позволяет упростить дешифраторы адреса за счет двухкоординатной выборки запоминающих элементов. Принцип двухкоординатной выборки поясняется (рис. 6.4а) на примере ЗУ типа ROM, реализующего только операции чтения данных. Здесь код адреса разрядностью n делится на две половины, каждая из которых декодируется отдельно. Выбирается запоминающий элемент, находящийся на пересечении активных линий выходов обоих дешифраторов. Таких пересечений будет как раз

$$2^{n/2} \times 2^{n/2} = 2^n.$$

Суммарное число выходов обоих дешифраторов составляет

$$2^{n/2} + 2^{n/2} = 2^{n/2+1},$$

что гораздо меньше, чем 2^n при реальных значениях n . Уже для ЗУ небольшой емкости видна эта существенная разница: для структуры 2D при хранении 1К слов потребовался бы дешифратор с 1024 выходами, тогда как для структуры типа 3D нужны два дешифратора с 32 выходами каждый. Недостатком структуры 3D в первую очередь является усложнение элементов памяти, имеющих двухкоординатную выборку.

Структура типа 3D, показанная на рис. 6.4а для ЗУ с одноразрядной организацией, может применяться и в ЗУ с многоразрядной организацией (рис. 6.4б), приобретая при этом «трехмерный» характер. В этом случае несколько матриц управляются от двух дешифраторов, относительно которых они включены параллельно. Каждая матрица выдает один бит адресованного слова, а число матриц равно разрядности хранимых слов.

Достоинства обеих рассмотренных структур сочетаются в структурах типа 2DM (2D модифицированная).

Структура 2DM

ЗУ типа ROM (рис. 6.5а) структуры 2DM для матрицы запоминающих элементов с адресацией от дешифратора DC_x имеет как бы характер структуры 2D: возбужденный выход дешифратора выбирает целую строку. Однако в отличие от структуры 2D, длина строки не равна разрядности хранимых слов, а многократно ее превышает. При этом число строк матрицы уменьшается и, соответственно, уменьшается число выходов дешифратора.

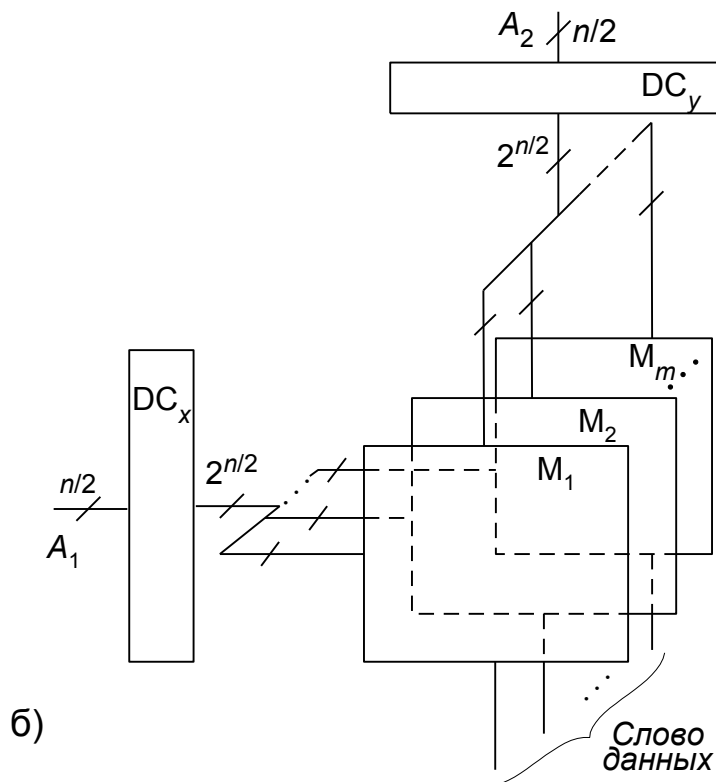
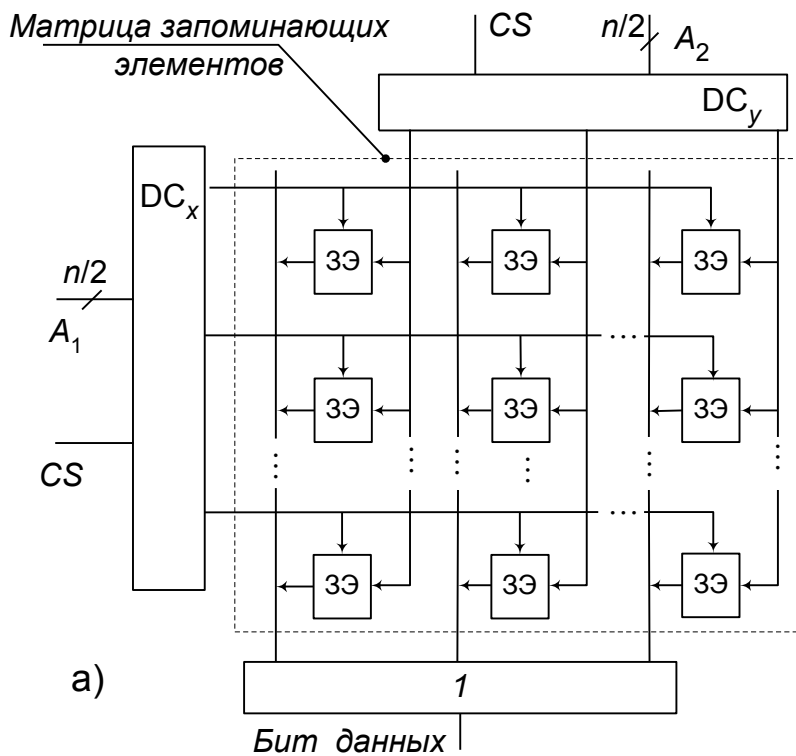


Рис. 6.4. Структура ЗУ типа 3D с одноразрядной (а) и многоразрядной (б) организацией

Для выбора одной из строк служат не все разряды адресного кода, а их часть $a_{n-1} \dots a_k$. Остальные разряды адреса (от a_{k-1} до a_0) используются, чтобы выбрать необходимое слово из того множества слов, которое содержится в строке. Это выполняется с помощью мультиплексов, на адресные входы которых подаются коды $a_{k-1} \dots a_0$. Длина строки равна $m2^k$, где m – разрядность хранимых слов. Из каждого «отрезка» строки длиной 2^k мультиплексор выбирает один бит. На выводах мультиплексов формируется выходное слово. По разрешению сигнала CS , поступающего на входы OE управляемых буферов с тремя состояниями, выходное слово передается на внешнюю шину.

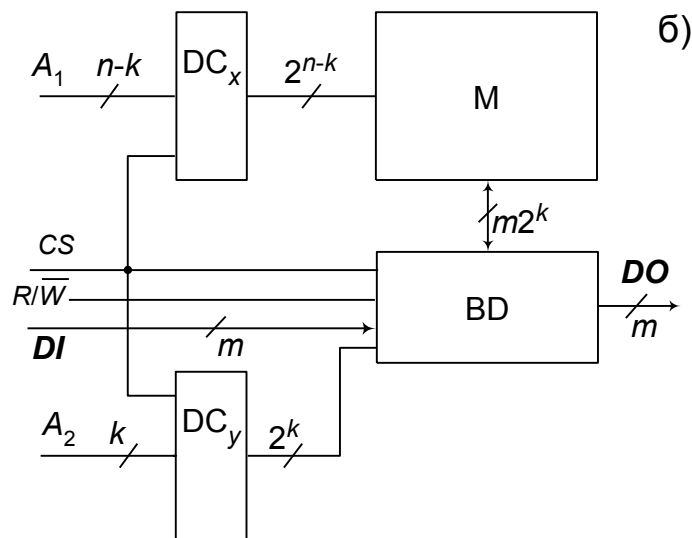
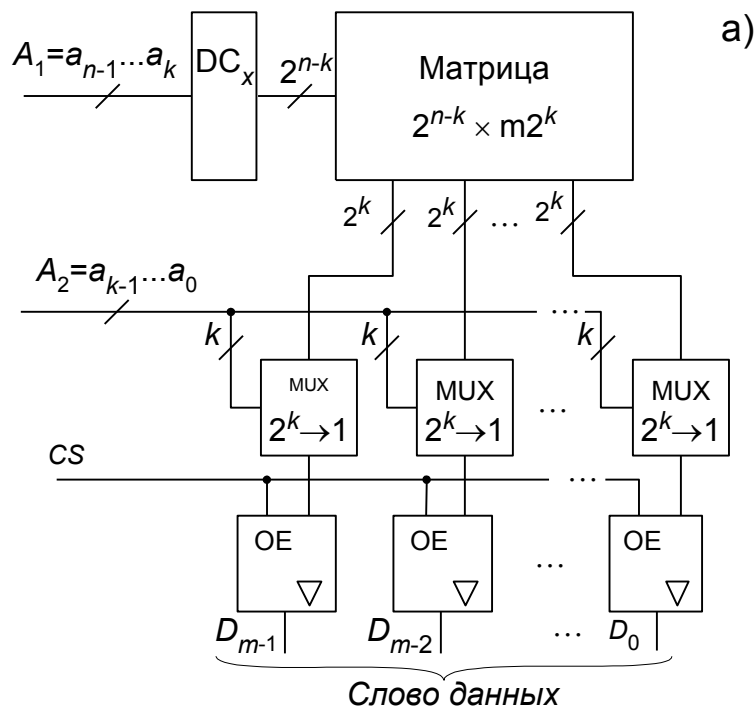


Рис. 6.5. Структура ЗУ типа 2DM
для ROM (а) и для RAM (б)

На рис. 6.5б в более общем виде структура 2DM показана для ЗУ типа RAM с операциями чтения и записи. Из матрицы M по-прежнему считывается «длинная» строка. Данные в нужный отрезок этой строки записываются (или считываются из нее) управляемыми буферами данных BD , воспринимающими выходные сигналы второго дешифратора DC_y и выполняющими не только функции мультиплексирования, но и функции изменения направления передачи данных под воздействием сигнала R/\overline{W} .

6.2. Статические оперативные запоминающие устройства

Статические запоминающие устройства относятся к числу наиболее быстродействующих. В частности, они широко используются в кэш-памяти, которая при сравнительно малой емкости должна иметь максимальное быстродействие. Статические ОЗУ (SRAM), как правило, имеют структуру 2DM, часть их при небольшой информационной емкости строится по структуре 2D.

Запоминающими элементами (ЗЭ) статических ОЗУ служат триггеры. Триггеры можно реализовать по любой схмотехнологии (ТТЛ(Ш), И²Л, ЭСЛ, n-МОП, КМОП GaAs и др.). Различие в параметрах этих ЗУ отражает специфику той или иной схмотехники. В последнее время наиболее интенсивно развиваются ЗУ, выполненные по КМОП-технологии. По мере уменьшения топологических норм технологического процесса КМОП-ЗУ приобретают высокое быстродействие при сохранении своих традиционных преимуществ.

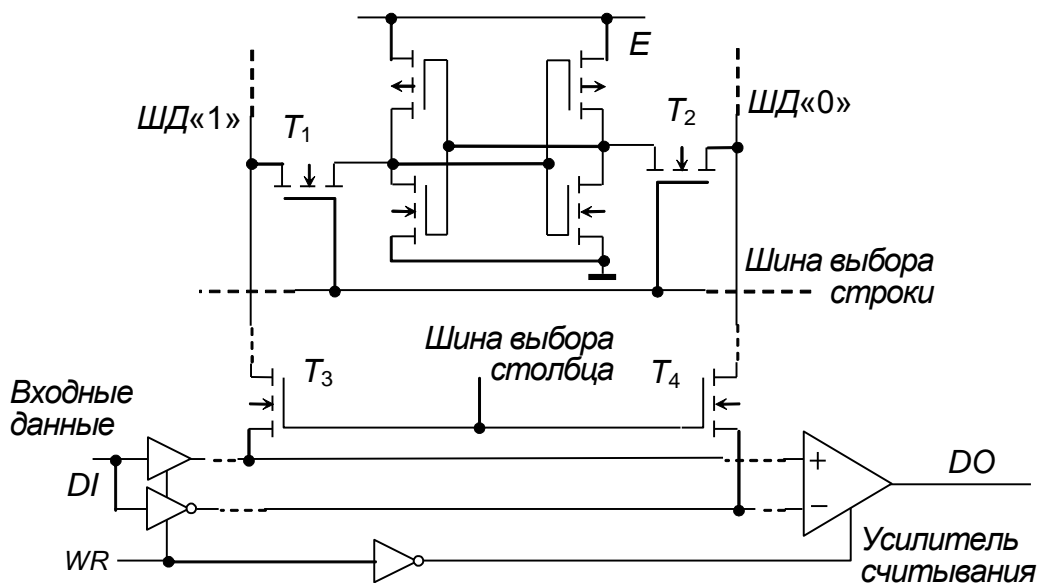


Рис. 6.6. Схемное решение статического ОЗУ

Запоминающий элемент двумерного ЗУ на КМОП-транзисторах (рис. 6.6) представляет собой *RS*-триггер с транзисторами выборки T_1 и T_2 . При обращении к ЗЭ активизируются сигналы *выбора столбца* и *выбора строки*, которые открывают каналы транзисторов T_1 , T_2 и T_3 , T_4 . В режиме записи сигнал $WR = 1$ переводит входные буферы в состояние прямой (для одного) и инверсной (для другого) передачи информационного сигнала *DI* на шины данных ШД«0» и ШД«1» выбранного столбца. Передаваемый по линиям ШД«0» и ШД«1» сигнал воспринимается той ячейкой запоминающего массива, у которой открытыми окажутся транзисторы T_1 и T_2 . Поступивший через эти транзисторы бифазный сигнал установит триггер ячейки в соответствующее *DI* состояние.

Чтение данных из ОЗУ также сопровождается активизацией линий выбора столбца и строки, но при сигнале $WR = 0$. Последний переводит входные буферы в высокоимпедансное состояние и открывает выходной буфер – дифференциальный усилитель считывания – с тремя состояниями. Считываемые данные передаются на линию *DO*. Применение буферов с тремя состояниями по входу и выходу позволяет передавать входные и выходные данные по одной шине с разделением во времени.

6.3. Динамические оперативные запоминающие устройства

В динамических ЗУ (DRAM) данные хранятся в виде зарядов емкостей МОП-структур и основой ЗЭ является просто конденсатор небольшой емкости. Такой ЗЭ значительно проще триггерного, что позволяет разместить на кристалле намного больше ЗЭ (в 4...5 раз) и обеспечивает динамическим ЗУ максимальную емкость. Вместе с тем конденсатор неизбежно теряет со временем свой заряд и хранение данных требует их периодического (через несколько миллисекунд) восстановления.

Запоминающие элементы

Известны конденсаторные ЗЭ разной сложности. В последнее время практически всегда применяют однотранзисторные ЗЭ – лидеры компактности. Электрическая схема и конструкция однотранзисторного ЗЭ показаны на рис. 6.7. Транзистор выборки отключает запоминающий конденсатор от линии записи-считывания *ЛЗС* или подключает его к ней. Сток транзистора не имеет внешнего вывода и образует одну из обкладок конденсатора. Другой обкладкой служит подложка. Между обкладками расположен тонкий слой диэлектрика – оксида кремния SiO_2 .

В режиме хранения транзистор выборки заперт. При выборке данного ЗЭ на затвор по линии выборки (*ЛВ*) подается напряжение, отпирающее транзистор. Запоминающая емкость через проводящий канал подключается к

линии записи-считывания, оказывая тем самым влияние на потенциал ЛЗС. При записи потенциал линии записи-считывания передается на конденсатор, определяя его состояние.

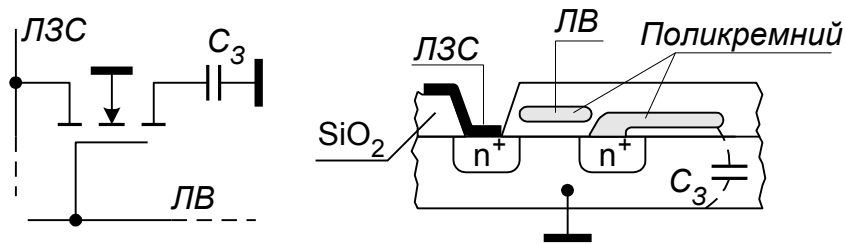


Рис. 6.7. Схема и конструкция запоминающего элемента динамического ЗУ

Схемное построение динамического запоминающего устройства

Фрагмент динамического ЗУ с несколькими запоминающими ячейками и усилителем считывания показан на рис. 6.8. Запоминающее устройство организовано так, что каждый столбец запоминающего массива обслуживается одним регенеративным (с положительной обратной связью) усилителем считывания, выполненным на бистабильной ячейке (RS -триггере). Состояние триггера может меняться как под действием внешнего бифазного сигнала, идущего от линий данных D и \bar{D} через транзисторы T_3 и T_4 , так и под влиянием электрических зарядов на емкостях C_M ячеек памяти. В последнем случае состояние RS -триггера будет определяться зарядом той емкости, которая выбрана по сигналу выбора строки (Y_i или Y_i').

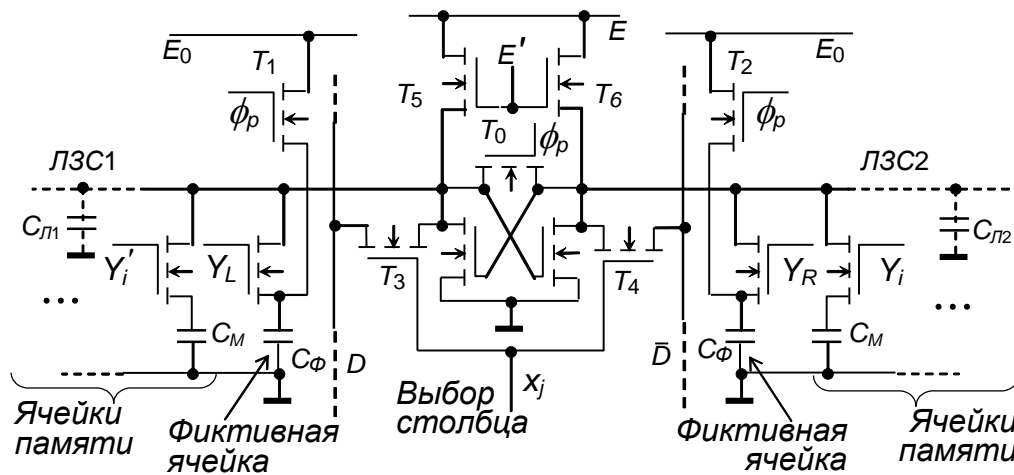


Рис. 6.8. Фрагмент схемы динамического ЗУ

К линиям записи-считывания (в симметричной конструкции на рис. 6.8 их две — $ЛЗС1$ и $ЛЗС2$) подключено столько ЗЭ, сколько строк имеется в запоминающей матрице. Особое значение имеют емкости $ЛЗС$ $C_{Л1}$ и $C_{Л2}$, которые по величине значительно превосходят емкости ЗЭ C_M . Поэтому перед считыванием производится предзаряд $ЛЗС1$ и $ЛЗС2$. Имеются варианты ЗУ с

предзарядом до уровня напряжения питания E и до уровня его половины $E/2$. На рис. 6.8 представлен последний вариант в силу его большей схемной простоты.

Пусть хранение «1» соответствует заряженной емкости, а хранение «0» – разряженной. При считывании нуля к ЛЗС подключается емкость C_M с нулевым зарядом. Часть заряда емкости линии записи-считывания C_L перетекает на емкость C_M , и напряжение на них уравнивается. Потенциал ЛЗС снижается на величину ΔU , которая и является сигналом, поступающим на усилитель считывания. При считывании единицы, напротив, напряжение на C_M составляло вначале величину E и превышало напряжение на ЛЗС. При подключении C_M к ЛЗС часть заряда стекает с запоминающей емкости в C_L и напряжение на ЛЗС увеличится на ΔU . Если до считывания C_L имела заряд

$$Q = EC_L/2,$$

то после считывания нуля он будет равен той же величине Q , но при уменьшенном на ΔU потенциале:

$$Q = (C_L + C_M)(E/2 - \Delta U).$$

Отсюда следует, что

$$\Delta U = EC_M/[2(C_M + C_L)] \approx \frac{C_M E}{2} / C_L.$$

В силу неравенства $C_M \ll C_L$ сигнал ΔU оказывается слабым. Кроме того, считывание является разрушающим – подключение запоминающей емкости к ЛЗС изменяет ее заряд. По этой причине в ЗУ для считывания данных применяются усилители-регенераторы (УР).

Симметричная структура ЗУ на рис. 6.8 оправдана тем, что «разрезание» ЛЗС на две половины вдвое уменьшает емкость линий, к которым подключены ЗЭ и, следовательно, вдвое увеличивает ΔU . В режиме считывания к одному плечу УР подключена одна половина ЛЗС (1 или 2) с адресованным ЗЭ, а к другому – вторая половина ЛЗС (2 или 1) с фиктивной емкостью C_Φ , с которой считывается промежуточный потенциал $U_{ПР}$. Перед считыванием от устройства управления ЗУ поступает сигнал ϕ_p (высокий потенциал), открывающий транзистор T_0 . При этом БЯ оказывается в промежуточном состоянии, когда транзисторы в обоих плечах открыты, и на них устанавливается промежуточный потенциал

$$U_{ПР} = [U_{ПОР_У} + B(E' - U_{ПОР_Н})]/(1 + B),$$

где $B = \sqrt{b_H / b_U}$; b_H , b_U и $U_{ПОР_Н}$, $U_{ПОР_У}$ – относительная крутизна и напряжение отпирания нагрузочных транзисторов T_5 и T_6 и управляющих транзисторов T_3 и T_4 . Напряжение E' и параметры транзисторов выбираются такими, чтобы обеспечить $U_{ПР} \approx 0,5E$. Одновременно такой же потенциал при наличии сигнала ϕ_p поступает на емкость C_Φ .

При считывании одновременно с поступлением сигналов на шины выборки строки (Y_i или Y_i') и столбца (X_j) запирается транзистор T_0 . БЯ оказывается в неустойчивом состоянии и стремится перейти в какое-либо устойчивое состояние, когда один из управляющих транзисторов (T_3 или T_4) открыт, а другой закрыт. Направление переключения БЯ определяет заряд на выбранной емкости C_M . Таким образом, благодаря использованию УР в процессе считывания формируются необходимые уровни сигналов, которые поступают на выход ЗУ, а также вновь заносятся в ЗЭ. Аналогично выполняется регенерация информации с той разницей, что сигналы из регенерируемых ячеек на выход ЗУ не передаются, а происходит только восстановление потенциалов на емкостях всех запоминающих элементов (поочередно по строкам).

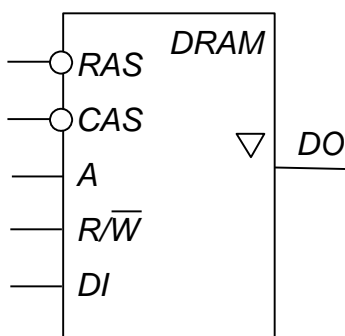


Рис. 6.9

В динамических микросхемах памяти адресация обычно производится в мультиплексном режиме. Адрес делится на два полуадреса, один из которых представляет адрес строки, а другой — адрес столбца матрицы ЗЭ. Полуадреса подаются на одни и те же выходы А корпуса ИС (рис. 6.9) поочередно с использованием стробирующих адресных сигналов *RAS* (*Row Address Strobe*) и *CAS* (*Column Address Strobe*). Мультиплексирование адресов позволяет уменьшить число выводов корпуса ИС, а также дает возможность различного использования

поступающих на ЗУ адресных сигналов. Например, в режиме регенерации адрес столбца совсем не нужен. При выполнении регенерации на адресных входах устанавливается адрес строки и стробирующий сигнал $RAS = 0$. На входах *CAS* и *R/W* сохраняются значения, равные 1. В результате производится выборка запоминающих элементов целой строки, усиление поступивших от ЗЭ на ЛЗС сигналов и их перезапись (восстановление). При этом сигнал выбора столбца пассивен и выход *DO* микросхемы остается в отключенном состоянии. Путем последовательной установки адресов строк, задаваемых с помощью счетчика, и подачи $RAS = 0$ осуществляется регенерация информации во всех строках накопителя.

Современные процессоры характеризуются высоким быстродействием. Это требует увеличения скорости работы ОЗУ. Особенно остро эта задача стоит перед разработчиками динамических ОЗУ, составляющих основную память компьютера. Методы повышения быстродействия ОЗУ основаны на предположении о кучности адресов при обращении к ОЗУ. Это отвечает тенденции, проявляющейся при выполнении самых разных программ и состоящей в том, что адреса последующих обращений к ОЗУ вероятнее всего расположены рядом с текущим адресом. Наибольший эффект достигается в синхронной динамической памяти (*Synchronous DRAM* — *SDRAM*), использующей конвейеризацию тракта продвижения информации. При этом синхросигналы *SDRAM* увязаны с тактовой частотой системы.

6.4. Постоянные и репрограммируемые запоминающие устройства

Микросхемы постоянных запоминающих устройств (ПЗУ, или *Read-Only Memory* – ROM) делятся на два класса: с однократной и многократной записью информации. Остановимся подробнее на ПЗУ с многократной записью информации, которые допускают программирование в оперативном режиме. К ним относятся репрограммируемые ПЗУ с ультрафиолетовым стиранием (РПЗУ-УФ, EPROM, *Electrically Programmable Read-Only Memory*) и с электрическим стиранием (РПЗУ-ЭС, EEPROM (или E²PROM), *Electrically Erasable Programmable Read-Only Memory*).

В репрограммируемых ЗУ типов EPROM и E²PROM возможны стирание старой информации и замена ее новой в результате специального процесса, для проведения которого ЗУ выводится из рабочего режима. Рабочий режим (чтение данных) – процесс, выполняемый с относительно высокой скоростью. Замена же содержимого памяти требует выполнения гораздо более длительных операций.

Запоминающими элементами современных РПЗУ являются транзисторы типов МНОП и ЛИЗМОП (добавление ЛИЗ к обозначению МОП происходит от слов «лавинная инжекция заряда»).

МНОП-транзистор отличается от обычного МОП-транзистора двуслойным подзатворным диэлектриком. На поверхности кристалла расположен тонкий слой двуокиси кремния SiO₂, далее более толстый слой нитрида кремния Si₃N₄ и затем уже затвор (рис. 6.10а). На границе диэлектрических слоев возникают центры захвата заряда. Благодаря туннельному эффекту носители заряда могут проходить через тонкую пленку окисла толщиной не более 5 нм и скапливаться на границе раздела слоев. Этот заряд и является носителем информации, хранимой МНОП-транзистором. Заряд записывают созданием под затвором напряженности электрического поля, достаточной для возникновения туннельного перехода носителей заряда через тонкий слой SiO₂. На границе раздела диэлектрических слоев можно создавать заряд любого знака в зависимости от направленности электрического поля в подзатворной области. Наличие заряда влияет на пороговое напряжение транзистора.

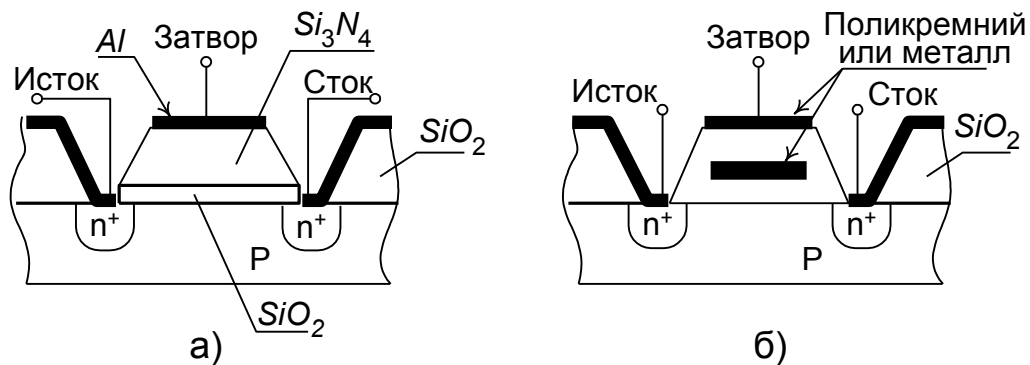


Рис. 6.10. Структуры транзисторов типа МНОП (а) и ЛИЗМОП с двойным затвором (б)

Для МНОП-транзистора с *n*-каналом отрицательный заряд на границе раздела слоев повышает пороговое напряжение (экранирует воздействие положительного напряжения на затворе, отпирающего транзистор). При этом пороговое напряжение возрастает настолько, что рабочие напряжения на затворе транзистора не в состоянии создать в нем проводящий канал. Транзистор, в котором заряд отсутствует или имеет другой знак, легко открывается рабочим значением напряжения. Так осуществляется хранение бита в МНОП: одно из состояний трактуется как отображение логической единицы, другое – нуля.

При программировании ЗУ используются относительно высокие напряжения, около 20 В, из-за чего вследствие туннельного эффекта носители тока попадают на границу раздела диэлектриков. После снятия высоких напряжений туннельное прохождение носителей заряда через диэлектрик прекращается и заданное транзистору пороговое напряжение остается неизменным.

После $10^4 \dots 10^6$ перезаписей МНОП-транзистор перестает устойчиво хранить заряд. РПЗУ на МНОП-транзисторах энергонезависимы и могут хранить информацию месяцами, годами и десятками лет.

Перед новой записью старая информация стирается записью нулей во все запоминающие элементы. Тип ЗУ – РПЗУ-ЭС.

Транзисторы типа ЛИЗМОП имеют так называемый плавающий затвор, который может быть единственным или вторым, дополнительным к обычному (управляющему) затвору. Транзисторы с одним плавающим затвором используются в ЗУ типа РПЗУ-УФ, а транзисторы с двойным затвором пригодны для применения как в РПЗУ-УФ, так и в РПЗУ-ЭС. Рассмотрим более современный тип – ЛИЗМОП-транзистор с двойным затвором (рис. 6.10б).

Принцип работы ЛИЗМОП с двойным затвором близок к принципу работы МНОП-транзистора – здесь также между управляющим затвором и областью канала помещается область, в которую при программировании можно вводить заряд, влияющий на величину порогового напряжения транзистора. Только область введения заряда представляет собою не границу раздела слоев диэлектрика, а окруженную со всех сторон диэлектриком проводящую область

(обычно из поликристаллического кремния), в которую, как в ловушку, можно ввести заряд, способный сохраняться в ней в течение очень длительного времени. Эта область и называется плавающим затвором.

При подаче на управляющий затвор, исток и сток импульса положительного напряжения относительно большой амплитуды 20...25 В в обратно смещенных *p-n*-переходах возникает лавинный пробой, область которого насыщается электронами. Часть электронов, имеющих энергию, достаточную для преодоления потенциального барьера диэлектрической области, проникает в плавающий затвор. Снятие высокого программирующего напряжения восстанавливает обычное состояние областей транзистора и запирает электроны в плавающем затворе, где они могут находиться длительное время (в высококачественных приборах – многие годы).

Заряженный электронами плавающий затвор увеличивает пороговое напряжение транзистора настолько, что в диапазоне рабочих напряжений проводящий канал в транзисторе не создается. При отсутствии заряда в плавающем затворе транзистор работает в обычном ключевом режиме.

Стирание информации может производиться двумя способами – ультрафиолетовым облучением или электрическими сигналами. В первом случае корпус ИС имеет специальное прозрачное окошко для облучения кристалла. Двуокись кремния и поликремний прозрачны для ультрафиолетовых лучей. Эти лучи вызывают в областях транзистора фототоки и тепловые токи, что делает области прибора проводящими и позволяет заряду покинуть плавающий затвор. Операция стирания информации этим способом занимает десятки минут, информация стирается сразу во всем кристалле. В схемах с УФ-стиранием число циклов перепрограммирования существенно ограничено, т. к. под действием ультрафиолетовых лучей свойства материалов постепенно изменяются.

Электрическое стирание информации осуществляется подачей на управляющие затворы низкого (нулевого) напряжения, а на стоки – высокого напряжения программирования. Электрическое стирание имеет преимущества: можно стирать информацию не со всего кристалла, а выборочно (индивидуально для каждого адреса). Длительность процесса «стирание–запись» значительно меньше, сильно ослабляются ограничения на число циклов перепрограммирования (допускается $10^4 \dots 10^6$ таких циклов). Кроме того, перепрограммировать ЗУ можно, не извлекая микросхему из устройства, в котором она работает. В то же время схемы с электрическим стиранием занимают больше места на кристалле, в связи с чем уровень их интеграции меньше, а стоимость выше. В последнее время эти недостатки быстро преодолеваются и ЭС-стирание вытесняет УФ-стирание.

Предшественниками двухзатворных ЛИЗМОП-транзисторов были однозатворные, имевшие только плавающий затвор. Эти транзисторы обычно с *p*-каналом, поэтому введение электронов в плавающий затвор приводило к созданию в транзисторе проводящего канала, а удаление заряда – к исчезновению такого канала. При использовании таких транзисторов

запоминающие элементы состоят из двух последовательно включенных транзисторов: ключевого МОП-транзистора обычного типа для выборки адресованного элемента и ЛИЗМОП-транзистора, состояние которого определяет хранимый бит. Стирание информации производится ультрафиолетовыми лучами.

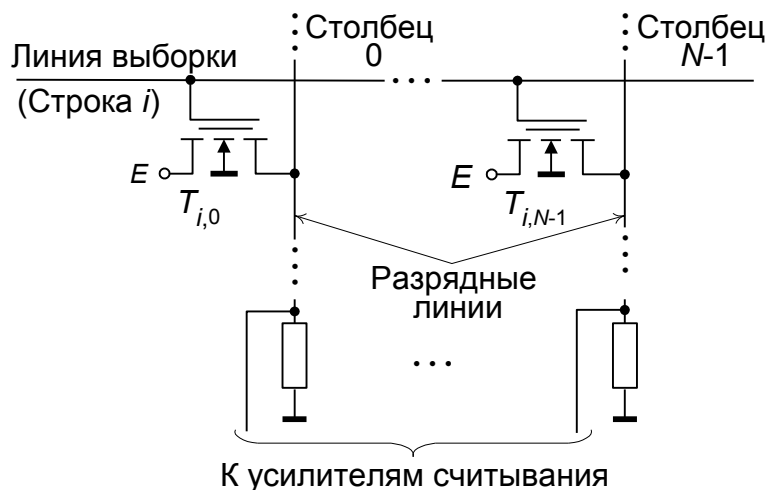


Рис. 6.11. Схема подключения ЛИЗМОП-транзисторов с двойным затвором

Подключение двухзатворных ЛИЗМОП-транзисторов к линиям выборки строк и линиям чтения в матрицах ЗУ показано на рис. 6.11. Запись логического нуля осуществляется путем инжекции в плавающий затвор «горячих» электронов и накопления на нем нужного заряда в режиме программирования. Стирание информации (удаление заряда из плавающего затвора) приводит к записи во все ЗЭ логических единиц.

6.5. Энергонезависимые оперативные запоминающие устройства

Ключевой особенностью энергонезависимой оперативной памяти является то, что в ней информация сохраняется при отключении источника питания. В этом смысле она ведет себя как постоянное запоминающее устройство. Принципиальным отличием от ПЗУ (ROM) является то, что доступ к ячейкам памяти в режиме записи осуществляется в том же порядке, как и к оперативному запоминающему устройству (RAM), в то время как любое ПЗУ требует специфического процесса программирования. К энергонезависимым ЗУ относятся ферроэлектрическая память F-RAM, магниторезистивной памяти (*Magnetoresistive RAM*, M-RAM) и фазопеременной памяти (*Phase-Change RAM*, P-RAM), над разработкой и внедрением которых сейчас работает множество компаний.

Фазопеременная память P-RAM использует тот же принцип хранения, что и перезаписываемые компакт-диски – халькогенидные сплавы приобретают

аморфное или кристаллическое состояние при определенных температурах нагрева. Только для считывания с компакт-дисков используется различие в отражательной способности аморфных и кристаллических участков поверхности, а в микросхемах памяти – различное электрическое сопротивление халькогенидных ячеек в разном фазовом состоянии. Ключевое преимущество P-RAM – высокая устойчивость к ионизирующим излучениям.

Второе преимущество P-RAM – размер ячейки, представляющей собой диод, к одному из электродов которого, как простой резистор, подключен халькогенидный столб. При такой архитектуре ячейки возможно построение на одном кристалле массивов памяти большого размера.

Над коммерциализацией P-RAM совместно работают фирмы Intel и ST-Microelectronics. Основные проблемы, которые стоят перед разработчиками P-RAM на пути к массовому производству:

- диалектическое противоречие – с одной стороны, разрушение информации при нагреве, а, с другой, снижение энергозатрат при программировании, чем обусловлена необходимость в таких сплавах, которые меняли бы фазовое состояние при как можно более низкой температуре для снижения длительности и величины тока записи;
- недостаточная разница сопротивлений сплава в аморфном и кристаллическом состоянии требует применения чувствительных схем сравнения и распознавания логического «0» и «1». Это снижает устойчивость работы микросхем в условиях электромагнитных помех;
- отвод тепла при интенсивных операциях записи;
- ресурс по количеству циклов перезаписи хотя и велик, но не достаточен для действительно произвольного обращения — $10^8 \dots 10^{10}$ циклов.

Применение P-RAM, вероятно, будет лежать в тех областях, где интенсивность операций записи не будет столь велика и будет существенная потребность в стойкости к внешним воздействиям.

Что касается M-RAM, то эта память является энергоемкой и она может найти применение в стационарных высокопроизводительных устройствах, способных обеспечить потребность M-RAM в энергии. Имеются прототипы коммерческой M-RAM на тактовые частоты до 250 МГц.

Существующая технология производства F-RAM позволяет получать запоминающие массивы объемом до 64 Мбит. Ведутся разработки, направленные на уменьшение топологического размера ячейки. Быстродействие F-RAM достаточно высокое – на изменение логического состояния ячейки требуется менее 1 нс и существует потенциал для дальнейшего увеличения быстродействия F-RAM за счет применения быстрых схем КМОП-обрамления запоминающего массива.

Есть две структуры ферроэлектрической ячейки – планарная и ярусная (рис. 6.12). Последняя, появившаяся в 2007 году с проектной нормой 130 нм за счет размещения ферроэлектрического конденсатора непосредственно над

стоком МОП-ключа, а не в одной плоскости с ним, позволила уменьшить размер ячейки памяти до $0,71 \text{ мкм}^2$.

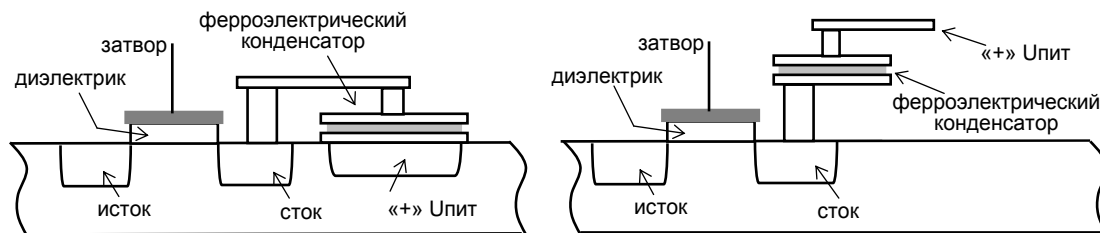


Рис. 6.12. Планарная и ярусная структуры ячеек F-RAM

Для F-RAM свойственно низкое энергопотребление, одинаковое в режимах записи и чтения. На самых высоких тактовых частотах, доступных для выпускаемых сейчас микросхем F-RAM, ток потребления в моменты переключения уровней действующих напряжений не превышает 25 мА. В периоды установившихся логических уровней ток потребления в несколько раз меньше. По сравнению с EEPROM или Flash, для записи полного массива микросхемы F-RAM требуется почти на 2 порядка меньше энергии источника питания.

Микросхемы F-RAM устойчивы к внешними воздействиями, таким как электрические и магнитные поля, электромагнитные помехи, электростатические разряды. Например, чтобы изменить поляризацию ячейки F-RAM внешним электрическим полем, необходимо приложить непосредственно к корпусу микросхемы поле напряженностью $\sim 130 \text{ кВ}$.

6.6. Кэш-память

Кэш-память запоминает копии информации, передаваемой между устройствами (прежде всего между процессором и основной памятью). Она имеет небольшую емкость в сравнении с основной памятью и более высокое быстродействие (реализуется на триггерных элементах памяти).

При чтении данных сначала выполняется обращение к кэш-памяти. Если в кэше имеется копия адресованной ячейки основной памяти, то кэш вырабатывает сигнал Hit (попадание) и выдает данные на общую шину данных. В противном случае сигнал Hit не вырабатывается и выполняется чтение из основной памяти и одновременное помещение считанных данных в кэш.

В основу работы кэш-памяти положен принцип локальности программы или, как еще говорят, гнездовой характер обращений, имея в виду, что адреса последовательных обращений к основной памяти (ОП) образуют компактную группу. При обращении к ОП в кэш-память копируются не отдельные данные, а блоки цифровой информации, включающие те данные, которые с большой степенью вероятности будут использованы в ЦП на последующих шагах работы

Объем кэш-памяти много меньше емкости основной памяти и любая единица информации, помещаемая в кэш, должна сопровождаться дополнительными данными (тегом), определяющими, копией содержания какой ячейки основной памяти является эта единица информации.

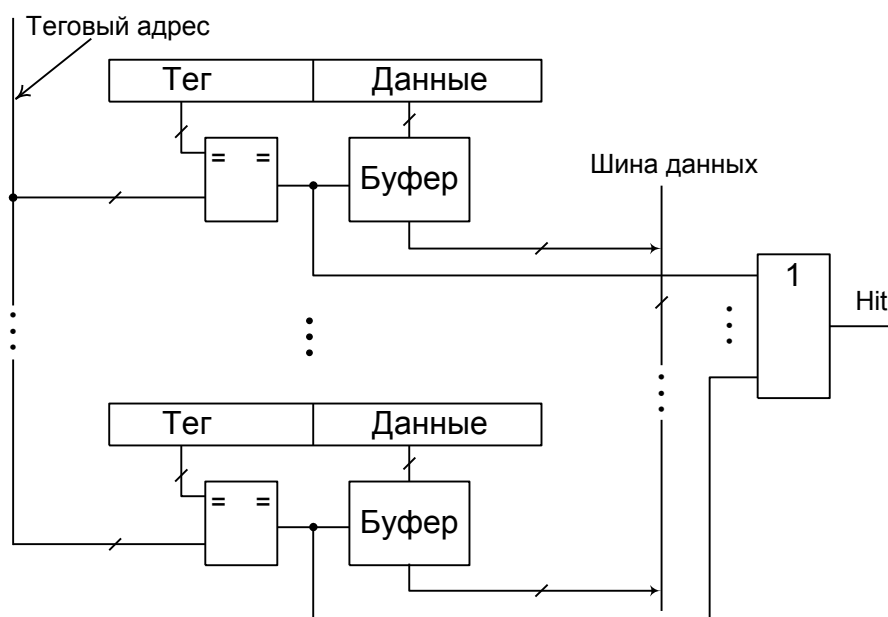


Рис. 6. 13. Структура полностью ассоциативной кэш-памяти.

В полностью ассоциативной кэш-памяти (FACM – *Fully Associated Cache Memory*), структура которой показана на рис. 6.13, каждая ячейка хранит данные, а в поле «тег» всех ячеек – полный физический адрес информации, копия которой записана. При любых обменах физический адрес запрашиваемой информации сравнивается с полями «тег» всех ячеек и при совпадении их в любой ячейке устанавливается сигнал Hit.

При чтении и значении Hit = 1 данные выдаются на шину данных, если же совпадения нет (Hit = 0), то при чтении из основной памяти данные вместе с адресом помещаются в свободную ячейку или наиболее давно не используемую ячейку кэш-памяти.

При записи данные вместе с адресом сначала, как правило, размещаются в кэш-памяти (в обнаруженную при Hit = 1 или в свободную при Hit = 0). Копирование данных в основную память выполняется под управлением специального контроллера, когда нет обращений к памяти.

Память типа FACM является сложным устройством и используется только при малых емкостях, главным образом в специальных приложениях. В то же время этот вид кэш-памяти обеспечивает наибольшую функциональную гибкость и бесконфликтность адресов, т. к. любую единицу информации можно загрузить в любую ячейку кэш-памяти. Сложность FACM заставляет искать иные структуры кэш-памяти, более экономичные по аппаратным затратам на их реализацию. К числу таких структур относятся *кэш-память с прямым*

отображением и кэш-память с ассоциацией по нескольким направлениям (наборно-ассоциативная).

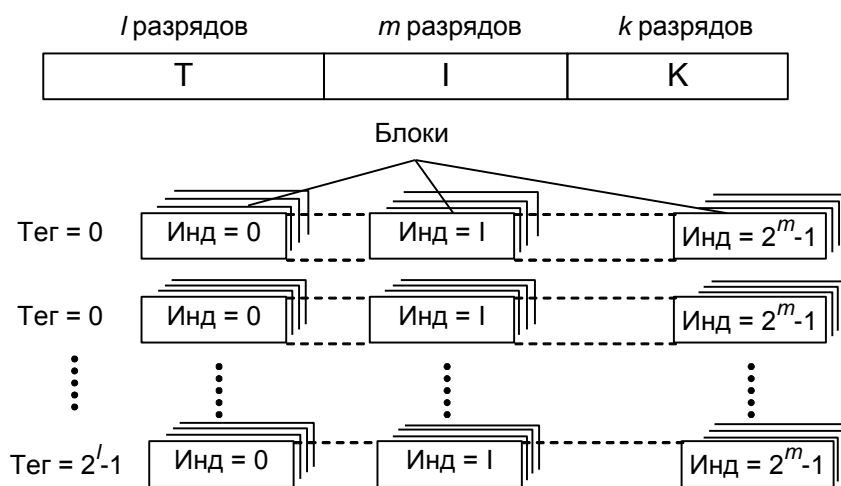


Рис. 6.14. Представление памяти при использовании механизма кэширования

При использовании таких структур память условно представляется в виде двумерного массива (рис. 6.14), состоящего из строк и столбцов. На пересечении строк и столбцов находятся блоки, состоящие из нескольких слов с числом кратным 2 (2, 4, 8, 16, 32), разным для разных микропроцессорных (МП) систем. Адрес блока в строке называется *индексом* блока, а адрес строки, которой он принадлежит – *тегом*. Разряды в поле K определяют смещение входящего в блок слова по отношению к базовому адресу блока – адресу первого слова. Размер слов в блоке может быть разным (равным байту или слову большей разрядности), и это зависит от того, как организована основная память системы.

Во всех механизмах кэш-память представляет собой две сверхоперативные памяти (СОП): память отображения данных и память тегов. Для кэш-памяти с прямым отображением память отображения данных выглядит как строка двумерного массива с размером ячейки, равным размеру блока (рис. 6.15). Память тегов имеет тот же объем, но с размером ячейки, определяемым размером тега.

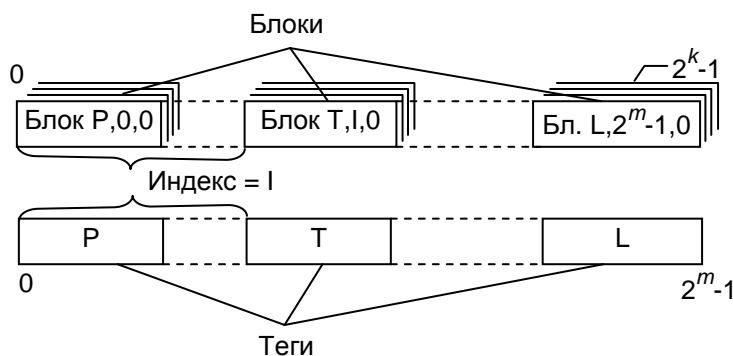


Рис. 6.15. Организация кэш-память с прямым отображением

На рис. 6.15 показаны три размещенных в памяти отображения данных блока с базовыми адресами $P00$, $T10$ и $L(2^m-1)0$, каждый из которых образован полем тега (P, T и L), полем индекса (0, I и 2^m-1) и смещением $K = 0$ ($K = 0$, т. к. первое слово в блоке имеет нулевое смещение). Теги P, T и L размещены в соответствующих ячейках памяти тегов с индексами 0, I и 2^m-1 .

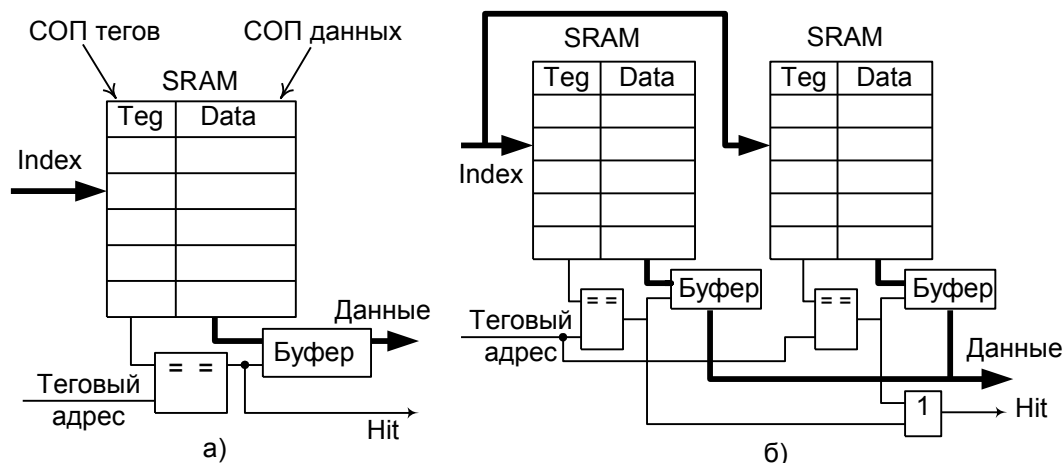


Рис. 16. Кэш-память с прямым отображением (а) и наборно-ассоциативная кэш-память (б)

Структурная схема кэш-памяти с прямым отображением представлена на рис. 6.16а. При обращении к памяти значение тега в строке с индексом *Index* сравнивается с теговым адресом. При наличии совпадения вырабатывается сигнал *Hit* и данные берутся из блока данных или помещаются в блок данных по адресу, определяемому смещением *K*. Напомним, что запись в кэш-память всегда сопровождается последующим обновлением основной памяти и делается это всегда независимо от того, какой механизм кэширования используется.

Промежуточным по сложности и эффективности вариантом между структурами FАСМ и кэш-памяти с прямым отображением является *кэш-память с ассоциацией по нескольким направлениям (наборно-ассоциативная)*. В этом варианте несколько строк кэша объединяются в наборы, а средние разряды адреса памяти (*Index*) определяют уже не одну строку, а набор строк (рис. 16.б и 17).

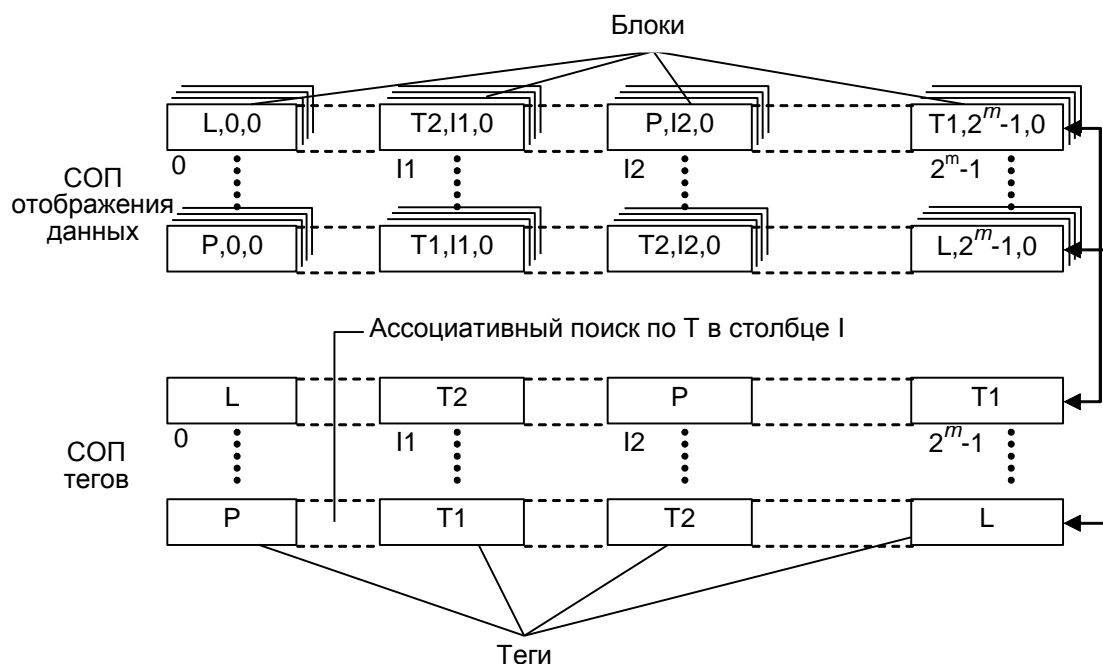


Рис. 6.17. Организация наборно-ассоциативной кэш-памяти

В отличие от предыдущей в наборно-ассоциативной кэш-памяти и СОП данных, и СОП тегов содержат по несколько строк. Число строк может быть разным для разных систем, но обычно кратно 2. Известны кэш-памяти на 2, 4, 8 строк. Число строк отображения данных всегда равно числу строк тегов, и каждой строке данных однозначно соответствует определенная строка тегов.

На рис. 6.17 приведен пример организации кэш-памяти с наборно-ассоциативным отображением данных. В примере скопированы блоки с адресами $L,0,0$; $T2,I1,0$; ...; $P,I2,0$; ...; $T1,2^m-1,0$; $P,0,0$; ...; $T1,I1,0$; ...; $L,2^m-1,0$.

Обращение к ОП по адресу T,I,K происходит следующим образом. Проверяется, нет ли блока Ti в кэш-памяти. По признаку T адреса обращения осуществляется ассоциативный поиск в СОП тегов в столбце с индексом I из адреса обращения. Рис. 16б поясняет эту процедуру для кэш-памяти, состоящей из двухслойного набора статического ОЗУ, используемого в качестве СОП данных и СОП тегов.

Кэш-память может быть использована как для хранения программного кода, так и для хранения данных. Это может быть объединенный или отдельный для программы и данных кэш. В последнем случае имеется различие обусловленное тем, что кэш-память программы используется только для чтения, в то время как кэш-память данных доступна и для чтения и для записи.

6.7. Механизм виртуальной памяти

В тех случаях, когда объем имеющейся системе памяти недостаточен и не соответствует перекрываемому процессором адресному пространству, а

генерируемые адреса выходят за пределы адресов физической памяти, формируемые процессором адреса нельзя непосредственно использовать при обращениях к памяти. Возникает необходимость отображения генерируемых адресов на физическую память системы. Это входит в одну из задач, которые решаются с помощью *механизма виртуальной адресации*.

В системах с виртуальным режимом адресации память делится на отдельные сегменты. Процессор, обращаясь к памяти, формирует линейный (виртуальный) адрес, исходя из логического адреса, определяемого по содержимому специального регистра – селектора сегмента и относительному адресу. Линейный адрес находится как сумма базового адреса (адреса начала) сегмента и относительного адреса (смещения в сегменте). Схема формирования линейного адреса представлена на рис. 6.18.

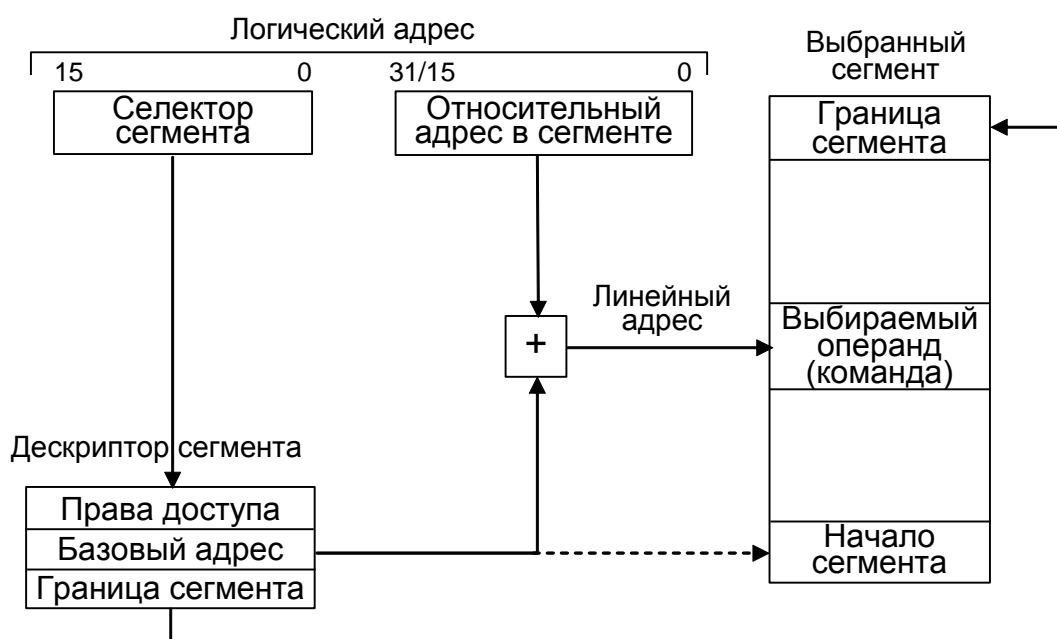


Рис. 6.18. Формирование линейного адреса

В памяти может быть выделено несколько сегментов, например, сегмент кода, сегмент данных, сегмент стека и т.д. Кроме того в многозадачной среде каждой задаче выделяется свой сегмент памяти. В таком случае желательно иметь несколько регистров для хранения селекторов соответствующих сегментов, что имеет место на практике в процессорах, поддерживающих виртуальную адресацию.

При генерации адреса преследуется не только цель обмена информацией с памятью, но решается задача защиты памяти от случайного (непредусмотренной решаемой задачей) обращения к сегментам и страницам памяти. Система защиты предусматривает различные виды контроля:

- контроль доступа к сегментам и страницам с помощью системы привилегий;
- контроль использования сегментов и страниц путем ограничений на разные виды обращения к ним: запрещение записи в сегменты данных

(страницы), разрешение только для чтения, запрещение чтения сегментов (страниц) программ, разрешенных только для выполнения, запрещение обращения к незагруженным (отсутствующим) сегментам и страницам и ряд других;

- ограничение набора выполняемых команд в зависимости от уровня привилегий выполняемой программы (выделение привилегированных команд).

Страничная организация памяти

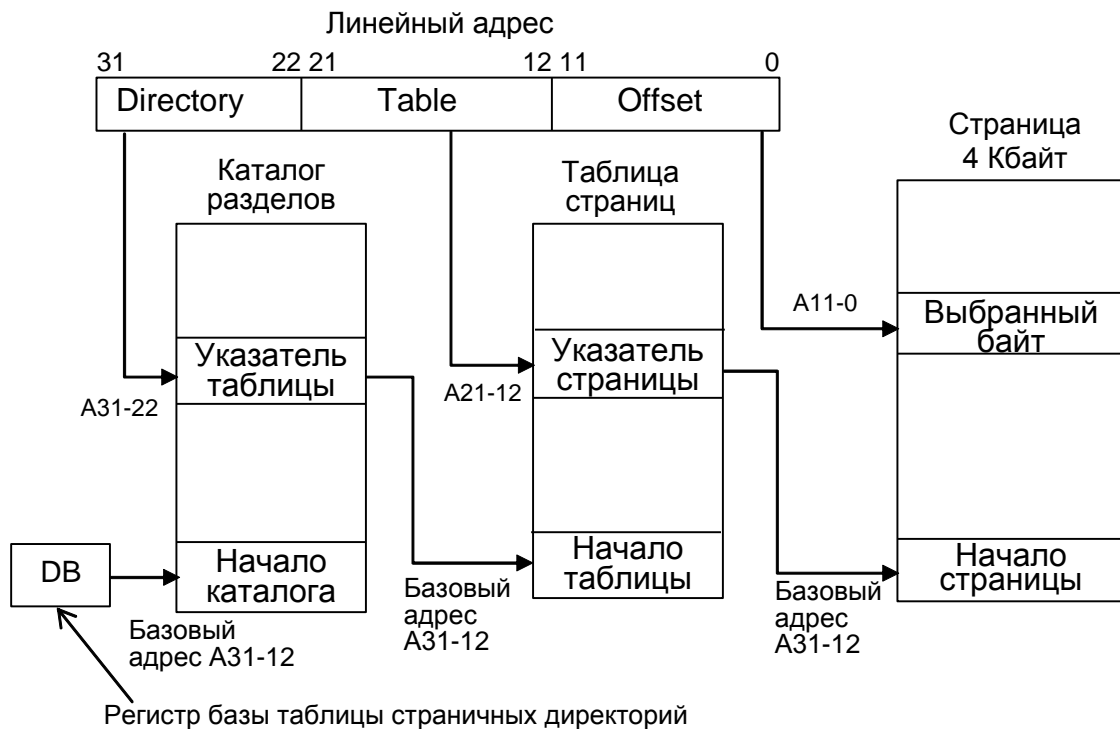


Рис. 6.19 Формирование 32-разрядного физического адреса при страничной организации памяти для страниц стандартного размера

В дополнение к сегментации, виртуальная память разбивается на страницы, представляющие собой ту минимальную единицу, которая используется в операциях замещения содержимого оперативной памяти тем, что размещено во внешней памяти. Если при обращении к памяти оказывается, что адресованной страницы в основной памяти нет, то происходит загрузка этой страницы из внешней памяти в свободное место или вместо страницы ОП, обращение к которой было наиболее давним. Размеры страниц могут быть различными для разных процессоров или для одного и того же процессора при разных режимах его работы с памятью: от одного или нескольких килобайт (стандартным является размер в 4 Кбайта) до нескольких мегабайт (например, 4 Мбайта).

Линейный адрес разбивается на три поля (рис. 6.19): поле каталога разделов – директории (*Directory*), поле таблицы страниц (*Table*) и поле адреса в странице (*Offset*). Базовый адрес каталога разделов задается значением

специального регистра DB (*Directory Base*) процессора. Регистр программно доступен. Его содержимое определяется действиями на стадии инициализации системы и не меняется при ее работе. Показанные на рис. 6.19 размеры полей директории, таблицы страниц и смещения в странице относятся к частному случаю – к процессорам фирмы Intel. В общем случае и для разных процессоров границы полей могут иметь разное расположение, а поля *Directory*, *Table* и – разную размерность.

По содержащимся в каталоге разделов указателям находится базовый адрес (начало) нужной таблицы страниц. Затем из выбранной таблицы страниц берется базовый адрес (начало) страницы, в которой находится адресуемая ячейка памяти и по смещению *Offset* находится ее адрес.

Таким образом, трансляция адреса выполняется как последовательность действий, предполагающих многократное обращение к памяти, где хранятся каталог разделов и таблицы страниц. Существенное сокращение времени преобразования адресов достигается путем использования дополнительной (в дополнение в кэш-памяти программы и кэш-памяти данных) специализированной внутренней кэш-памяти, которая называется буфером ассоциативной трансляции (*Translation Look-Aside Buffer* – TLB). Эти буферы хранят физические адреса нескольких ранее выбранных страниц. При повторном обращении к этим страницам процедура трансляции не выполняется, а из TLB выбирается ранее полученный адрес страницы.

Нередко процессоры, поддерживающие механизм виртуальной адресации, содержат два буфера TLB – один для адресов команд, другой для адресов данных.

Кэш память адресной трансляции

Современные МП поддерживают различные по размеру страницы и для каждого размера страниц используются отдельные кэш-буферы (ассоциативные буферы страничной трансляции – TLB).

В качестве примера на рис. 6.20 представлена структура TLB в МП Intel для 4-Кбайтных страниц. Такой буфер имеет 64 записи с ассоциативным поиском по четырем входам четырехслойного набора SRAM. В СОП данных хранятся физические адреса страниц. В памяти тегов помимо теговых адресов имеются дополнительные биты, используемые при обращениях к страницам и их замещении на вновь затребованную в случае ее отсутствия в TLB. Эти биты указывают

- на принадлежность страницы операционной системе (супервизору) или пользовательской программе – бит U/S пользователь/супервизор,
- на возможность кэширования страницы – бит CD кэш-копирование разрешено,
- на способ обновления содержимого ячеек в основной памяти после записи данных в кэш-память – бит WT сквозная запись,

- на возможность модификации принадлежащих странице ячеек памяти – бит R/W разрешение записи,
- на корректность хранящейся в странице информации – бит V корректность
- и, наконец, на то, что содержащиеся на странице данные являются «мусором» и могут быть замещены адресом новой страницы – бит D «грязь».

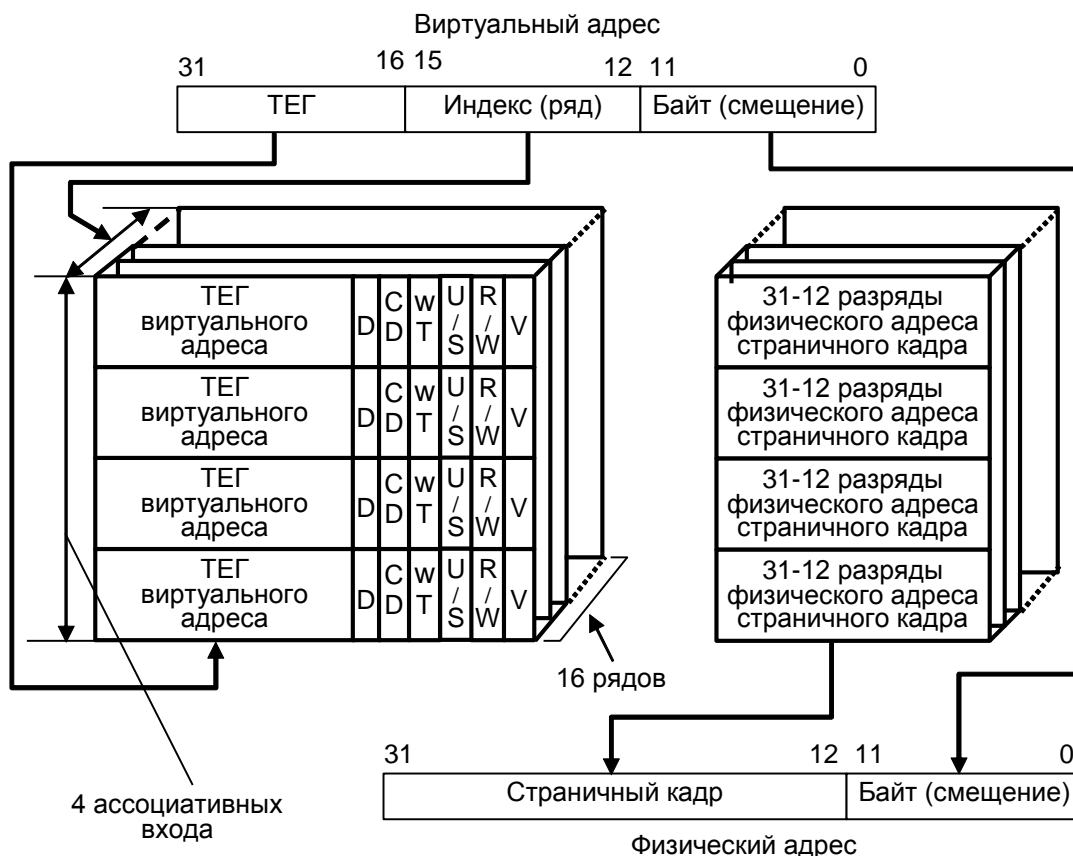


Рис. 6.20. Организация буфера TLB для 4-Кбайтных страниц (D – «грязь»; CD – кэш-копирование разрешено; WT – сквозная запись; U/S – режим пользователь/супервизор; R/W – разрешение записи; V – корректность)

7. Микропроцессоры: архитектура и структурное построение

Микропроцессор (МП) – это программно управляемое устройство для обработки цифровой информации и управления процессом обработки. Префикс «микро» в слове «микропроцессор» отражает тот факт, что все основные компоненты процессора интегрированы в одну микросхему. Исторически определение «микропроцессор» возникло тогда, когда наряду с МП использовались процессоры на ИС с малой степенью интеграции и секционированные процессоры, выполненные с помощью печатного монтажа на одной или нескольких платах. Сегодня определение «микропроцессор» утратило свою первоначальную смысловую нагрузку, поскольку процессоры по большей части реализуются на основе СБИС-технологии. Поэтому не будем акцентировать внимание на различии терминов «процессор» и «микропроцессор», тем более, что по отношению к ним есть обобщающий термин – *центральный процессор* (ЦП).

Как уже отмечалось, *архитектура* МП – это *программная модель* процессора, т. е. его представление с точки зрения программиста.

Структуру микропроцессора образуют входящие в него функциональные блоки, с помощью которых реализуется предназначенная для процессора программа, размещенная в виде двоичного машинного кода в памяти микропроцессорной системы.

7.1. Функционально-структурные особенности микропроцессоров

В процессе многолетнего развития произошла дифференциация микропроцессоров по функционально-структурным особенностям и областям применения. В настоящее время имеются следующие основные классы МП:

- универсальные *микропроцессоры с CISC-архитектурой*;
- универсальные *микропроцессоры с RISC-архитектурой*;
- специализированные микропроцессоры (сигнальные и др.);
- микроконтроллеры.

Универсальные микропроцессоры с CISC-архитектурой (*Complicated Instruction Set Computer, компьютер со сложным набором команд*) применяются главным образом в персональных компьютерах и серверах. Используются они также и в устройствах управления, встраиваются в различные приборы и системы.

Наряду с CISC-процессорами все активнее в микропроцессорную технику внедряются RISC-процессоры. Общепринятая расшифровка термина RISC – *Reduced Instruction Set Computer* – компьютер с укороченным набором команд.

Исходно аббревиатура RISC расшифровывалась несколько иначе, а именно как *Rational Instruction Set Computer* – компьютер с рациональной системой команд. RISC-архитектурные решения используются при создании микроконтроллеров, цифровых процессоров сигналов (*Digital Signal Processor – DSP*). Универсальные микропроцессоры с RISC-архитектурой применяются в основном в рабочих станциях и мощных серверах.

В классе специализированных микропроцессоров наиболее широко представлены процессоры цифровой обработки сигналов. Практически все сигнальные процессоры имеют схожие базовые модули: вычислительное ядро, служащее для выполнения математических операций; память для хранения данных и программ; устройства преобразования аналоговых сигналов в цифровые и наоборот, таймеры и другие устройства.

Микроконтроллеры являются наиболее массовыми представителями микропроцессорной техники. Интегрируя на одном кристалле высокопроизводительный процессор, память и набор периферийных устройств, можно с минимальными затратами реализовать на микроконтроллерах разнообразные системы управления объектами и процессами. Использование микроконтроллеров в системах управления и обработки информации обеспечивает исключительно высокие показатели эффективности при столь низкой стоимости, что микроконтроллерам практически нет альтернативной элементной базы для построения качественных и дешевых систем. Во многих применениях система может состоять только из одного микроконтроллера.

В области обработки сигналов и особенно при параллельной обработке больших потоков данных все шире применяются программируемые логические интегральные схемы (ПЛИС) с высокой степенью интеграции.

Все шире применяются микропроцессоры, специализированные для обработки графической информации, для передачи и обработки информации в системах коммуникации, коммуникационные контроллеры.

7.2. Формат команд центрального процессора

Архитектура микропроцессора тесно связана с тем, как построены команды (или инструкции), с помощью которых управляют работой МП. Существуют два принципиально различных вида архитектур: *стековая и регистровая*. Стековые команды служат для обработки данных, которые размещаются в ячейках, выбираемых по умолчанию, т. е. на вершине стека, который часто реализуется в регистрах, известных только на аппаратном уровне. Отметим, что стек вычисления выражения концептуально и функционально отличается от стека для хранения записей. Стековые команды (иногда называемые также безадресными командами) позволяют получать очень компактный машинный код для вычисления выражения, поскольку они не содержат битов для указания адресов операндов и адресов, где будут храниться промежуточные результаты.

В регистровых архитектурах регистры, используемые для хранения операндов, явно показываются в командах.

В общем случае команда состоит из нескольких битовых полей – поля кода операции (КОП), полей операндов и поля адреса следующей команды (рис. 7.1). Операнд может быть указан либо непосредственно, либо с помощью адреса, по которому он расположен в основной памяти МП системы.

Обобщенный формат команды с указанием адресов памяти представлен на рис. 7.1а. Двоичное кодирование адресов операндов, результата и следующей инструкции требует большого числа разрядов, из-за чего вполне естественно возникла необходимость в более рациональном построении микропроцессорных инструкций.

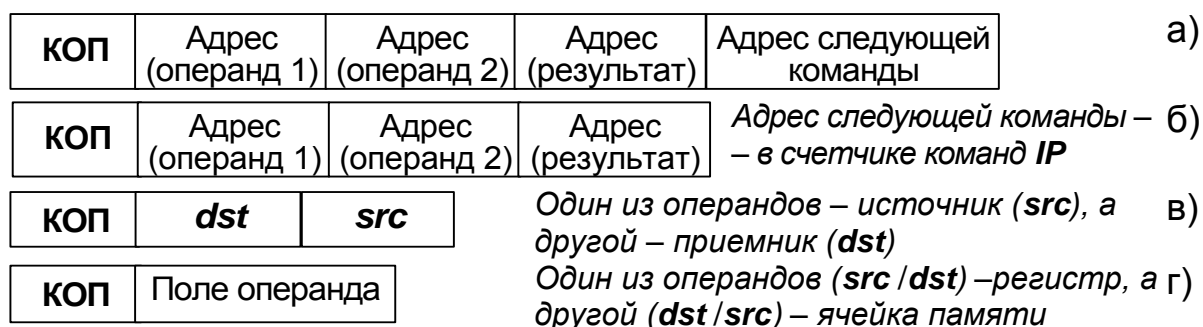


Рис. 7.1. Обобщенный формат команд и его сокращенные варианты

Размер команды можно сократить, если в качестве указателя адреса следующей команды использовать выделенный для этой цели регистр ЦП. Такой регистр называют указателем следующей инструкции (*Instruction Pointer – IP*), или *счетчиком команд (программным счетчиком)*. В результате команда становится *трехадресной* (рис. 7.1б), но и в этом случае может иметь неприемлемую длину, если адресуемые операнды расположены в памяти. Поэтому трехадресный формат обычно используется в тех МП, которые выполняют *операции* только с теми данными, которые расположены в регистрах процессора, т. к. регистры можно кодировать относительно небольшим количеством бит. Этому требованию отвечают, например, регистрово-ориентированные *RISC*-архитектуры.

Следующим шагом к уменьшению размера команд является переход на двухадресный формат за счет разделения операндов на *операнды-источники* (**src** – от слова *source*) и *операнды-приемники* (**dst** – от слова *destination*) (рис. 7.1в). Операнд-приемник, будучи первоначально источником, после выполнения операции принимает значение результата, в то время как значение операнда-источника остается без изменения.

Формат с одним полем операнда (рис. 7.1г) получается, если один из операндов (**dst** или **src**) – регистр. Такой формат приемлем для процессоров с небольшим числом регистров, при котором адресные коды регистров можно разместить в поле кода операции.

Наконец, формат с одним полем (полем КОП) применим в тех случаях, когда операнды определяются по умолчанию.

В представленных на рис. 7.1 форматах не указан тот, который позволял бы процессору работать с непосредственными, т.е. закодированными в команде, операндами. Эта проблема легко разрешается, если набор команд процессора дополнить командами, в которых поле адреса операнда источника (в случае трехадресной команды одно из полей адреса операнда) заменить непосредственным операндом. При этом для работы с непосредственным операндом не требуется дополнительных обращений к памяти, поскольку процессор получает значение этого операнда непосредственно из инструкции. Если выбор инструкций из памяти аппаратно поддержан счетчиком команд *IP*, то *IP* отвечает и за выборку непосредственного операнда. В отличие от этого получение значений операндов, обращение к которым происходит по адресам, указанным в команде или в регистрах, связано с дополнительными действиями на стадии исполнения инструкции.

Разнообразие форм представления команд свойственно CISC-процессорам – процессорам со сложным набором команд. Сложность команд можно понимать в том смысле, что при исполнении команды процессору приходится выполнять действия, связанные не только с выполнением операций с операндами, но и действия по загрузке/сохранению операндов из/в памяти, что требует кроме основного машинного цикла – цикла выборки команды – дополнительных машинных циклов обращения к памяти.

Исторически использование сложного набора команд было обусловлено тем, что применение сложных инструкций позволяло создавать компактный программный код, который можно было разместить в относительно небольших объемах памяти. Это было важно в то время, когда микросхемы памяти позволяли хранить относительно небольшие объемы информации. Позднее во времена, когда уровень интеграции памяти значительно возрос, возникло стремление создавать процессоры, команды которых по кодируемому в них функциям приближались к операторам языка высокого уровня. Это позволяло значительно упростить разработку компиляторов языков высокого уровня, но сильно усложняло работу процессоров и прежде всего работу входящего в его состав устройства управления выполнением программы (УУВП). Да и само УУВП при усложнении набора команд стало чрезвычайно сложным цифровым автоматом – микропрограммным автоматом, требующим для его реализации больших объемов памяти и высокоинтеллектуальных средств разработки. В связи с этим можно упомянуть семейство процессоров x86 фирмы *Intel*.

Помимо сказанного команды CISC-процессоров имеют разную длину в зависимости от того, с какими операндами они работают. Как следствие, в процессорах, использующих счетчик команд для хранения адреса следующей инструкции, усложняется процесс вычисления загружаемого в *IP* адреса.

Достоинством RISC-архитектуры является фиксированный формат команд, т.е. формат, при котором все команды имеют фиксированную длину и фиксированное место для расположения полей операндов. Это позволяет

- 1) при последовательном выполнении программного кода легко определять адрес следующей инструкции – задается простым инкрементированием счетчика команд,
- 2) значительно упростить процесс дешифрации и исполнения поступающей в процессор инструкции и, как следствие, упростить структуру и логику работы УУВП,
- 3) увеличить число регистров и, как следствие, уменьшить число обращений к памяти за данными.

Последнее дает основания рассматривать RISC-процессоры как процессоры с регистрово ориентированной архитектурой.

Принцип «укорочения» набора команд для RISC-процессоров состоит в разбиении сложных команд на ортогональные подмножества. При этом операции (арифметические, логические и др.) могут выполняться только над содержимым регистров. Регистры могут быть и операндами источниками, и операндами приемниками. Поскольку число регистров в процессоре относительно невелико, то для их кодирования достаточно небольшого количества бит, что делает возможным использование подобного представленному на рис. 7.1а трехадресного формата с фиксированным расположением полей регистров источников и приемников. Последнее упрощает процесс дешифрации регистров и уменьшает время, затрачиваемое на выполнение инструкции.

В отдельное (ортогональное) подмножество выделяются операции с регистрами. Другим подмножеством являются команды загрузки/хранения. При таком наборе команд операции с данными в памяти возможны при условии их предварительной загрузки в регистры. Результат всегда размещается в регистре и для его сохранения в памяти требуется дополнительная команда.

Еще одно ортогональное подмножество образуют команды управления. К ним относятся команды условных и безусловных переходов, команды вызовов процедур и возвратов из них, команды, позволяющие изменять отдельные биты (флаги) в регистре состояния процессора.

Упомянутые форматы команд во многом характерны для большинства микропроцессоров с последовательным выполнением поступающих из памяти инструкций. Отличия могут быть в расположении полей операндов, в их размерности, в месте расположения информации, касающейся интерпретации полей операндов. Однако есть иные форматы, которые ориентированы, например, на параллельные вычисления. Можно указать в связи с этим на многоядерные процессоры, для которых требуются инструкции, направленные на эффективное использование имеющихся в составе МП ядер. Одно из возможных решений этой задачи состоит в использовании инструкций с большой длиной командного слова (*Very Long Instruction Word – VLIW*). Подобного рода инструкции используются в высокопроизводительных векторных вычислителях.

Специфическими чертами обладают команды микроконтроллеров PIC, что обусловлено малой разрядностью регистров и, отчасти, простотой их технической реализации.

7.3. Типовая архитектура и последовательность выполнения команд центральным процессором

Центральный процессор выполняет следующие действия:

- загрузка/сохранение регистров и результатов операций в памяти;
- присваивание, логические и арифметические операции;
- безусловные и условные переходы, циклы;
- безусловные и условные вызовы процедур;
- ввод/вывод.

В ЦП должны быть средства для организации и создания массивов и других структур данных.

Типовая архитектура ЦП, обладающая такими возможностями, представлена на рис. 7.2. В центральном процессоре имеется устройство управления для дешифрирования и исполнения команд, набор рабочих регистров, АЛУ и секция управления вводом/выводом. Рабочие регистры в общем случае можно разделить на регистры, используемые для хранения данных и производства вычислительных и логических операций (*регистры данных*), и регистры, используемые для адресации команд и данных (*регистры адреса*). Некоторая часть отмеченных на рис. 7.2 устройств центрального процессора (обозначены пунктирными линиями) являются программно недоступными. К ним относятся устройство управления с принадлежащим ему регистром команд (*Instruction Register – IR*), АЛУ и секция управления вводом/выводом.

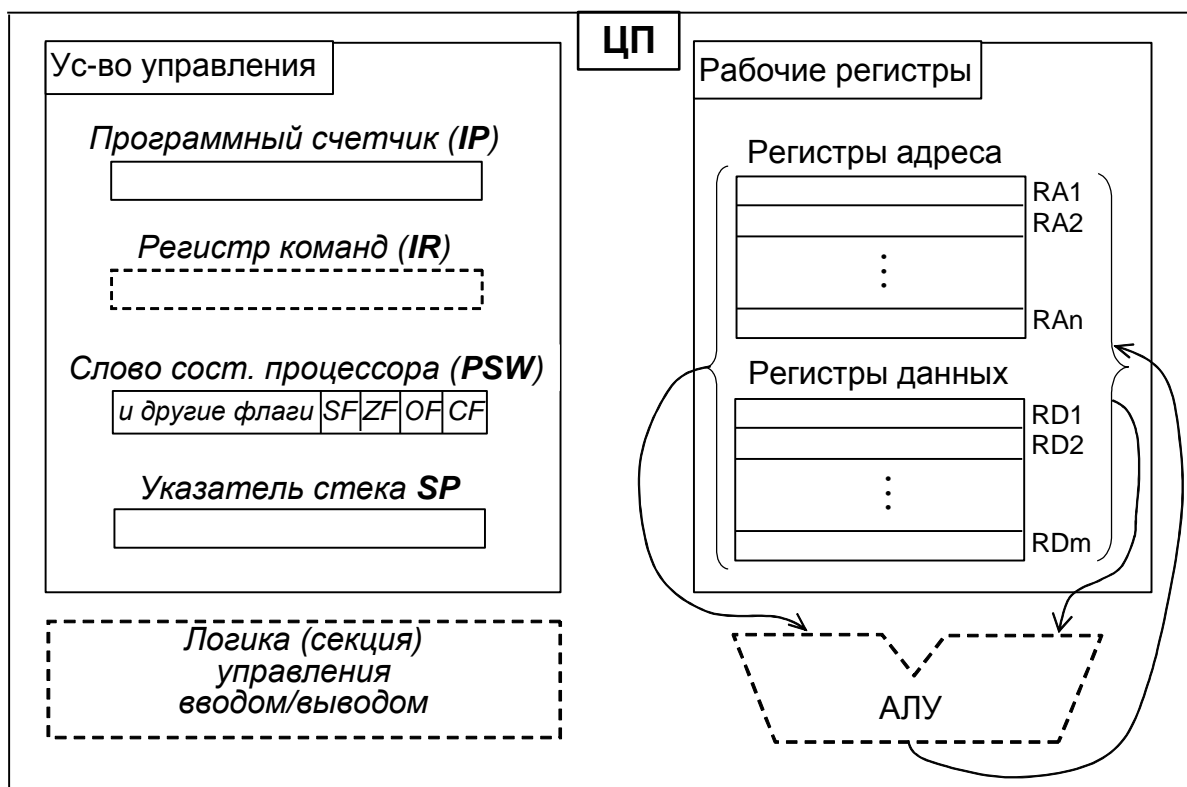


Рис. 7.2. Типовая архитектура центрального процессора

Программа работы процессора в виде машинного кода и кода данных хранится в памяти системы. Процессор исполняет программу последовательно до тех пор, пока не встретится команда перехода или вызова процедуры. Регистр команд *IR* содержит текущую команду на время ее дешифрации и выполнения, а программный счетчик *IP* предназначен для хранения адреса следующей команды. Когда текущая команда завершена, адрес из *IP* посылается в память. Адресованная команда передается в ЦП, и процессор вводит ее в регистр команд. Затем эта команда дешифрируется, определяется ее длина в байтах и содержимое *IP* инкрементируется на эту длину. Таким образом в *IP* образуется адрес следующей команды. Когда выполнение текущей команды заканчивается, содержимое *IP* вновь посылается в память и цикл повторяется.

Команды перехода позволяют изменить естественный порядок следования команд путем замещения содержимого *IP* (т. е. адреса следующей по порядку команды) адресом, определяемым самой командой перехода. *Команды условных переходов* замещают или не замещают содержимое *IP* в зависимости от состояния операционного блока (например, АЛУ) и процессора после выполнения предыдущих команд. Состояние процессора обозначается в виде флагов (признаков) в специальном регистре, называемом *словом состояния процессора* (*Processor State Word – PSW*). В этом регистре, в частности, находятся биты, показывающие флаги АЛУ (*ZF, SF, OF, CF* и др.). Если, например, после команды «вычитания» находится команда «перехода по нулю», переход осуществляется, если в *PSW* установлен флаг нулевого

результата *ZF*. Если же *PSW* фиксирует ненулевой результат, переход не производится. Когда переход реализован, начинается новая последовательность команд с адреса, к которому осуществлен переход.

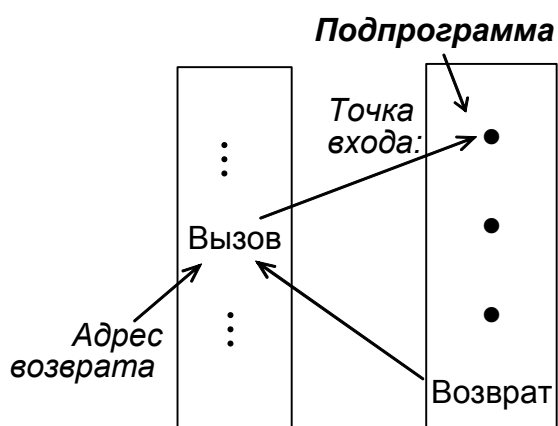


Рис. 7.3. Вызов подпрограммы

Действия, связанные с *вызовом подпрограмм*, требуют специальной разновидности перехода (рис. 7.3). Как и в других переходах, при вызове подпрограммы заменяется содержимое *IP* на адрес перехода, но при этом запоминается текущее содержание *IP*, образующее адрес возврата. Команда возврата из подпрограммы должна восстановить в *IP* адрес возврата, чтобы после завершения подпрограммы продолжилось последовательное выполнение

основной программы. При вызове подпрограммы кроме запоминания адреса возврата обычно требуется временно сохранить и другую информацию, например содержимое рабочих регистров. Это объясняется тем, что подпрограмма может разрушить первоначальное содержимое этих регистров, которое потребуется при возврате в основную программу. Обычно такая информация запоминается в специальной области памяти – в *стеке*. Адрес ячейки стека, к которой производилось последнее обращение (или, как в некоторых компьютерах, к которой будет происходить следующее обращение), находится в регистре *указателя стека SP (Stack Pointer)*.

При сохранении данных в стеке значение *SP* изменяется в сторону свободных (еще не использованных) ячеек стековой памяти, при извлечении данных из стека – в сторону последней загруженной в стек ячейки. Направление изменения значения *SP* зависит от того, как организован стек – как стек с записью в нижнюю часть стековой памяти, т.е. в область старших (наращиваемых) адресов, или как стек с записью в вершину стека, т.е. в область младших адресов.

Циклы реализуются либо с помощью команд условных переходов, либо с помощью специально предназначенных для этого команд ЦП, которые объединяют счет и (или) проверку *PSW* с условным переходом. В большинстве циклов, например в циклах типа *DO* языка Фортран, осуществляется инкремент или декремент счетчика и цикл повторяется до тех пор, пока счетчик не достигнет предела. При каждом изменении счетчика результат сравнивается с пределом, соответственно устанавливается *PSW* и в зависимости от его содержимого переход производится или не производится.

Важным для любой МП системы является взаимодействие с периферийными устройствами. большей частью устройства выполняют свои функции не затрагивая работу процессора. При этом процессор, следуя исполняемой программе, может обращаться к устройству для получения или

передачи информации, но чаще само устройство будучи готовым к обмену данными сообщает об этом событии процессору. Связанные с устройствами ввода-вывода события опознаются по запросам прерывания, и всякому событию соответствует своя программа обработки, которая находится в оперативной памяти системы. Любой запрос прерывания сопровождается двоичным кодом (вектором прерывания), по которому процессор опознает устройство и вызывает соответствующую процедуру обработки, загружая в программный счетчик ее стартовый адрес и приостанавливая таким образом выполнение основной программы. Запрос прерывания можно рассматривать как аппаратный способ вызова программных процедур, в чем состоит его существенное отличие от программного вызова. Процедура обработки прерывания всегда завершается командой возврата из прерывания.

Упрощенная блок-схема алгоритма выполнения команд микропроцессором приведена на рис. 7.4. Первые три блока представляют основной машинный цикл – цикл выборки команды из памяти по адресу, находящемуся в программном счетчике. Все дальнейшие после основного цикла действия процессора обусловлены содержащимся в команде кодом операции.

В представленной на рис. 7.4 блок-схеме отсутствуют ветви, связанные с вызовом и возвратом из процедур, с циклами и с обработкой запросов прерывания, т. к. действия процессора по их реализации имеют много общего с действиями при выполнении команд переходов. Имеющиеся особенности подлежат отдельному рассмотрению, что будет сделано в разделах, посвященных построению микропроцессорных систем.

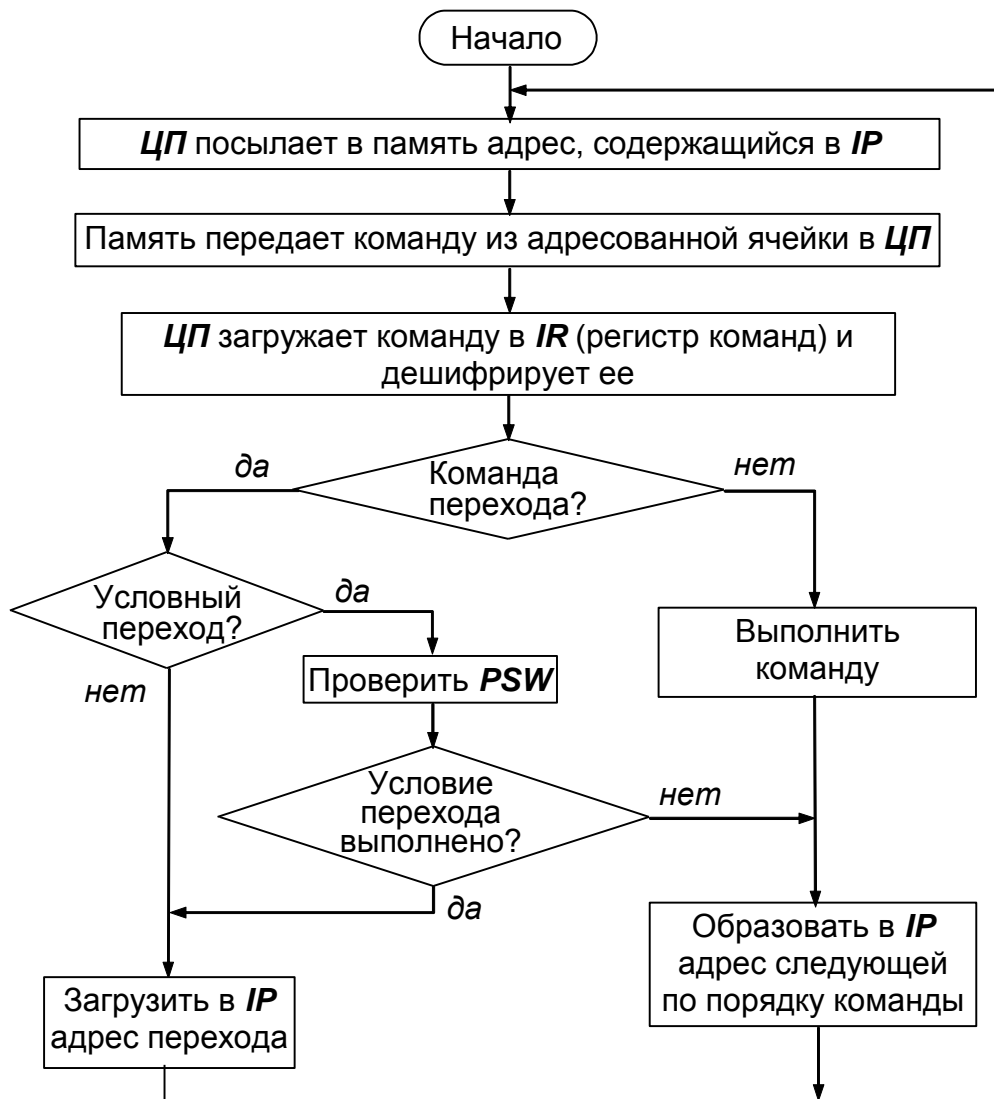


Рис. 7.4. Последовательность выполнения команд центральным процессором

Рабочие регистры предназначены для временного хранения информации, которая помогает при адресации и в вычислительном процессе. Их можно разделить на две группы: регистры адреса и регистры данных, хотя некоторые регистры могут выполнять совмещенные функции.

Регистры данных $RD1...RDn$ предназначены для временного хранения операндов и результатов операций с ними. Обращение к регистру осуществляется намного быстрее, чем обращение к памяти. Поэтому при необходимости выполнения нескольких операций над набором данных лучше ввести данные в регистры процессора, произвести требуемые вычисления и после этого поместить результаты в память, чем обрабатывать данные непосредственно из памяти. Обычно чем больше регистров данных находится в составе ЦП, тем выше его быстродействие.

Адресная группа применяется для гибкой адресации данных. Обработываемое командой данное может быть частью команды, частью

команды может быть его адрес, оно может находиться в регистре, в регистре может находиться его адрес, или адрес данного может быть суммой части команды и содержимого одного или нескольких регистров. Иногда указанный в команде регистр содержит исполнительный адрес *ИспАдр*, но часто адрес определяется более сложно. Например, при обращении к элементам массива адрес элемента состоит из двух частей – *базового адреса* (т. е. адреса первого элемента массива) и *смещения* по отношению к базовому адресу. При обработке всего массива удобно простое инкрементирование или декрементирование смещения. Поэтому адрес элемента массива часто определяется по сумме содержимого двух регистров, один из которых содержит базовый адрес и потому называется *регистром базы*, а другой содержит смещение и называется *индексным*. Работа с двумерным массивом более сложная требует, например, сложения адреса базы, смещения столбца и смещения в столбце. При этом обычно суммируются часть команды из поля операнда, значения базового и индексного регистров. Базовые регистры применяются также для перемещения в памяти блоков данных и фрагментов программ.

Таблица 7.1

Способ определения исполнительного адреса	Пояснение
$ИспАдр = АдрОп$	Адрес операнда находится в команде – режим прямой адресации.
$ИспАдр = (RAi)^*$	Адрес операнда содержится в регистре RAi (i – номер любого из регистров $RA1 \dots RAm$) – режим косвенно-регистрационной адресации.
$ИспАдр = АдрОп + (RAi)$	Адрес операнда вычисляется как сумма значений поля операнда в команде и одного из регистров адреса с номером i – режим косвенно-регистрационный со смещением $АдрОп$, указанным в команде.
$ИспАдр = (RAi) + (RAj)$	Адрес операнда равен сумме значений адресных регистров RAi и RAj с номерами i и j – базовый-индексный режим (в одном регистре находится адрес базы, в другом – смещение).
$ИспАдр = АдрОп + (RAi) + (RAj)$	Адрес операнда равен сумме значений поля операнда в команде и адресных регистров RAi и RAj с номерами i и j – режим относительный (относительно $АдрОп$) базовый-индексный.

* Круглыми скобками обозначено содержимое регистров RAi .

Способы использования адресных регистров определяются *режимами адресации*. Наиболее частот употребляемые режимы адресации представлены в таблице 7.1. Для обозначений адресных регистров использованы представленные на рис. 7.3 имена $RA1 \dots RAn$.

Среди названных режимов адресации не указаны те, которые при выполнении команды не требуют обращений к основной памяти. К таким режимам относятся режимы регистровой и непосредственной адресации. В первом случае операнд находится в регистре, код (адрес) которого указан в команде. Во втором – непосредственно в команде, отчего происходит название этого режима как режима с непосредственной адресацией.

Таблица 7.2

Мнемоническое обозначение команды	Действия, вызываемые командой
MOV Rd, Addr	Загрузить один из регистров данных Rd значением, взятым из ячейки памяти с адресом Addr.
MOV Addr, Rd	Сохранить регистр данных Rd в ячейке памяти по адресу Addr.
ADD Rd, Rd	Сложить содержимое двух регистров данных и полученный результат сохранить в регистре-приемнике.
ADD Rd, Addr	Сложить содержимое ячейки памяти по адресу Addr с содержимым регистра данных Rd и результат сохранить в регистре Rd.
ADD Addr, Rd	Сложить содержимое ячейки памяти по адресу Addr с содержимым регистра Rd и результат сохранить в ячейке памяти с адресом Addr
ADD Rd, const	Прибавить константу к содержимому регистра Rd и результат сохранить в регистре Rd.
SUB Rd, [Ra]	Вычесть из регистра данных Rd значение ячейки памяти, адрес которой указан в регистре адреса Ra.
MOV Addr[Ra], Rd	Сохранить содержимое регистра данных Rd в ячейке памяти с адресом образованным суммой значения поля операнда команды ADD и содержимого адресного регистра
ADD Rd, Addr[Ra][Ra]	Сложить содержимое регистра данных с содержимым ячейки памяти по адресу, образованному суммой значения поля операнда в команде, и значений двух регистров адреса Ra.

Перечисленные режимы адресации используются не во всех процессорах, а если используются, то по-разному отображаются в командах. Это обусловлено как структурой процессоров, так и форматом команд. Для иллюстрации способов отображения режимов адресации в командах, воспользуемся некоторой обобщенной (приемлемыми только для рассматриваемого здесь случая и безотносительно к какому-либо процессору) структурой команд и их мнемоническими обозначениями. Воспользуемся мнемониками, которые нередко можно встретить в программах, написанных на языке Ассемблер для некоторых процессоров. К рассмотрению примем двухоперандные команды CISC процессора (таблица 7.2) с форматом, представленным на рис. 7.1в.

Один из операндов *src* является операндом-источником, другой *dst* – операндом-приемником, используемым вначале как источник, а по завершении операции как приемник. Кроме того принято, что одним из операндов всегда является регистр данных *Rd* – любой регистр из набора *RD1...RDn*. Второй операнд определяется по указанному в команде режиму адресации. Для обозначения режимов адресации используется следующая символика: прямая адресация обозначается адресом *Addr* в команде, косвенно-регистровая – регистром адреса *Ra* (любой из регистров *RA1...RAm*) в квадратных скобках, косвенно-регистровая со смещением – регистром адреса *Ra* в квадратных скобках со стоящим перед ним адресом *Addr* в команде, относительная базовая-индексная – адресом *Addr* в команде и двумя регистрами адреса *Ra* в квадратных скобках.

Микропроцессоры воспринимают только программы написанные двоичным машинным кодом в соответствии с установленным для них форматом. Человеку трудно пользоваться машинным языком и поэтому приходится каждому двоичному коду команды МП присваивать мнемоники (сокращенные названия команд), используя которые программист может создавать программный продукт с большей уверенностью в его безошибочной работе. Программы, написанные с использованием мнемоник, называют мнемокодами или программами на языке Ассемблер. Машинный язык разрабатывается одновременно с созданием процессора и не может быть изменен, как не может быть изменена схема микропроцессора. Ассемблер также создается при разработке МП, но в принципе одни и те же машинные команды можно обозначить разными мнемониками и получить таким образом различные Ассемблеры. Перевод программ, написанных на языке Ассемблер, в машинные коды осуществляет специальный транслятор, носящий также название Ассемблера. Одна из компонент этого транслятора – это таблица перевода мнемоник в исполняемые машинные коды.

Микропроцессорные команды делятся на несколько групп:

1) Команды пересылки данных. Осуществляют пересылку данных между регистрами или между памятью и регистрами (пример в таблице 7.3).

Таблица 7.3

Мнемоника	Вызываемое действие	Режим адресации.
MOV R1, R2	Пересылка из регистра в регистр	Регистровый
MOV R, M	Пересылка из регистра в память	Исполнительный адрес МП определяется посредством прямой, косвенно-регистровой, косвенно-регистровой со смещением и другой свойственной МП адресации данных
MOV M, R	Пересылка из памяти в регистр	
MVI R, <i>const</i>	Загрузка регистра константой	Непосредственный

2) Арифметические команды. Предназначены для сложения, вычитания, увеличения или уменьшения содержимого регистров или ячеек памяти, а также для более сложных операций, таких как умножение и деление (пример некоторых команд в таблице 7.4).



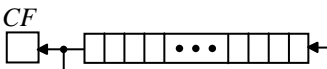
Таблица 7.4

Мнемоника	Вызываемое действие	Режим адресации.
ADD R1, R2	Сложить регистра с регистром	Регистровый
SUB R1, R2	Вычесть из регистра регистр	Регистровый
ADD R, M	Сложить регистр с содержимым ячейки памяти	Исполнительный адрес МП определяется посредством прямой, косвенно-регистровой, косвенно-регистровой со смещением и другой свойственной МП адресации данных
SUB R, M	Вычесть из регистра содержимое ячейки памяти	
ADI R, <i>const</i>	Сложить с регистром константу	Непосредственный
SBI R, <i>const</i>	Вычесть из регистра константу	Непосредственный
INR R	Инкремент регистра	Регистровый
DEC R	Декремент регистра	Регистровый

3) Логические и оперативные команды. Выполняют логические операции «И», «ИЛИ», «исключающее ИЛИ», «НЕ», сравнение, сдвиг данных в регистрах и ячейках памяти (пример в таблице 7.5).

Таблица 7.5

Мнемоника	Вызываемое действие	Режим адресации.
AND R1, R2	Логическое И регистра с регистром	Регистровый
OR R1, R2	Логическое ИЛИ регистра с регистром	Регистровый
XOR R1, R2	Исключающее ИЛИ регистра с регистром	Регистровый
AND R, M	Логическое И регистра с содержимым ячейки памяти	Исполнительный адрес МП определяется посредством прямой, косвенно-регистровой, косвенно-регистровой со смещением и другой свойственной МП адресации данных
OR, M	Логическое ИЛИ регистра с содержимым ячейки памяти	
XOR, M	Исключающее ИЛИ регистра с содержимым ячейки памяти	
ANI R, <i>const</i>	Логическое И регистра с константой	Непосредственный
ORI R, <i>const</i>	Логическое ИЛИ регистра с	Непосредственный

	константой	
XRI R, <i>const</i>	Исключающее ИЛИ регистра с константой	Непосредственный
SHR R	 Сдвиг содержимого регистра вправо	Регистровый
SHL R	 Сдвиг содержимого регистра влево	Регистровый
RLC R	 Циклический сдвиг регистра влево	Регистровый
RRC R	 Циклический сдвиг регистра вправо	Регистровый

4) Команды переходов (условных и безусловных), вызовов подпрограмм и возвращение из подпрограмм (пример в таблице 7.6).

Таблица 7.6

Мнемоника	Вызываемое действие	Режим адресации.
JUMP <i>addr</i>	Переход безусловный по указанному в команде адресу <i>addr</i>	Непосредственный
Jxx <i>addr</i> (JZ, JNZ, JC, JNC, JGT, JLT, JEQ и др.)	Переход условный по нулевому или ненулевому результату ($xx = Z$ или NZ), по переносу или отсутствию переноса ($xx = C$ или NC), по больше нуля, меньше нуля или равно нулю ($xx = GT$, LT или EQ)	Непосредственный
CALL <i>addr</i>	Вызов подпрограммы по указанному в команде адресу <i>addr</i>	Непосредственный
RTS	Возврат из подпрограммы по адресу, хранящемуся в ячейке стека, на которую указывает <i>SP</i>	Операнд по умолчанию
RTI	Возврат из процедуры обработки прерывания по адресу, хранящемуся в ячейке стека, на которую указывает <i>SP</i> , и восстановление слова состояния процессора	Операнд по умолчанию
PUSH R	Протолкнуть содержимое регистра в стек	Регистровый косвенный
POP R	Восстановить содержимое регистра из	Регистровый косвенный

	стека	
DI	Запретить прерывания	
EI	Разрешить прерывания	
HLT	Останов	
NOP	Нет операции	

5) Команды управления и работы со стеком (таблица 7.6) Предназначены для управления флагами в слове состояния процессора, для запрещения и разрешения прерываний.

Некоторые процессоры имеют команды ввода-вывода (*IN port* и *OUT port*) для работы с устройствами ввода-вывода. Одним из операндов является адрес порта ввода-вывода, второй – операнд по умолчанию, например регистр-аккумулятор в АЛУ. Такие команды есть в процессорах, имеющих два адресных пространства – пространство адресов памяти и пространство ввода-вывода. В процессорах с одним адресным пространством адреса портов ввода-вывода отображаются на память.

7.4. Управление выполнением программы

Извлечение команды из памяти и последующее ее исполнение по-разному могут быть реализованы в разных процессорах. Представленный на рис. 7.4 алгоритм работы процессора, несмотря на имеющиеся упрощения, в основном отражает действия процессора, которые начинаются с загрузки команды в предназначенный для этого регистр команд *IR* и заканчиваются формированием адреса следующей инструкции в программном счетчике (в счетчике команд) *IP*.

Рассмотрим эти действия более подробно, включая последовательное исполнение программного кода, переходы, вызовы, возвраты и циклы. Во многом они напоминают действия микропрограммного автомата с мультиплексором источника следующего адреса, к входам которого подключены микропрограммный счетчик и стек адресов возврата из подпрограмм (рис. 5.17). Для управления мультиплексором используются два бита, взятые из соответствующего поля микрокоманды. Подобный механизм применим и для формирования *адреса следующей команды* с той разницей, что микрокоманды расположены в памяти микропрограммы и используются для формирования функциональных сигналов, распространяемых по кристаллу микропроцессора, а команды микропроцессора хранятся в основной памяти МП системы и используются для управления работой МП процессора. Источник адреса следующей команды определяется тем, что указано в поле КОП (в поле кода операции) поступившей в процессор команды.

Большей частью процессор выполняет программу линейно, последовательно извлекая команды из памяти и исполняя их (рис. 7.5).

Изменения в линейном потоке команд обуславливаются *переходами* (JUMP), *вызовами процедур* (CALL) и *циклами* (DO UNTIL – делай, пока выполнено условие). Вызов процедуры предполагает возврат, который выполняется во команде RTS – *Retune from Subroutine*.

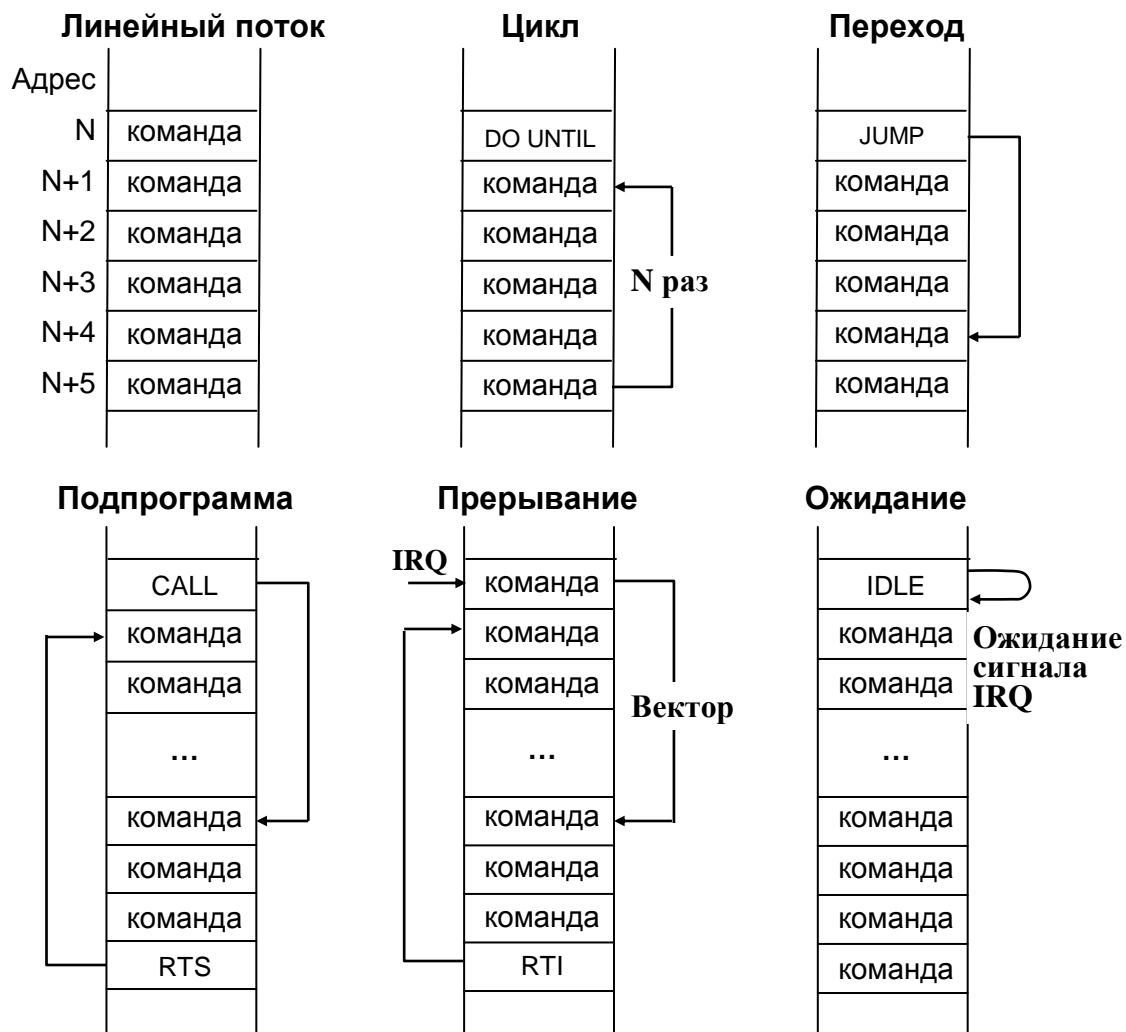


Рис. 7.5. Структуры, изменяющие линейный поток программы.

Линейное исполнение программы нарушается также вследствие *запросов прерываний* (*Interrupt Request – IRQ*), когда выполнение основной программы прерывается событием, которое происходит во время ее выполнения. Процедура обработки прерывания всегда завершается командой возврата из прерывания RTI – *Return from Interrupt*.

В настоящее время важное значение придается обеспечению возможности перевода микропроцессоров в режим пониженного потребления энергии. Особенно это касается микроконтроллеров и цифровых процессоров сигналов. Для этого используются встраиваемые в МП средства и, в частности, перевод микропроцессора в энергосберегающий режим работы с помощью специальной команды (команда IDLE на рис. 7.5). По этой команде процессор прекращает операции по обработке данных и переходит в режим *ожидания* с

пониженным энергопотреблением, оставаясь восприимчивым к запросам прерываний. Когда запрос прерывания поступает, процессор обрабатывает его, после чего продолжает стандартное выполнение программы.

Вызовы, прерывания и циклы могут поддерживаться как аппаратными, так и программными средствами. Однако алгоритмическая сторона этой поддержки в своей основе остается без изменений. Применение аппаратных средств сокращает накладные расходы (связанные, например, с обращениями к стеку) и временные затраты на вызовы, возвраты и организацию циклов

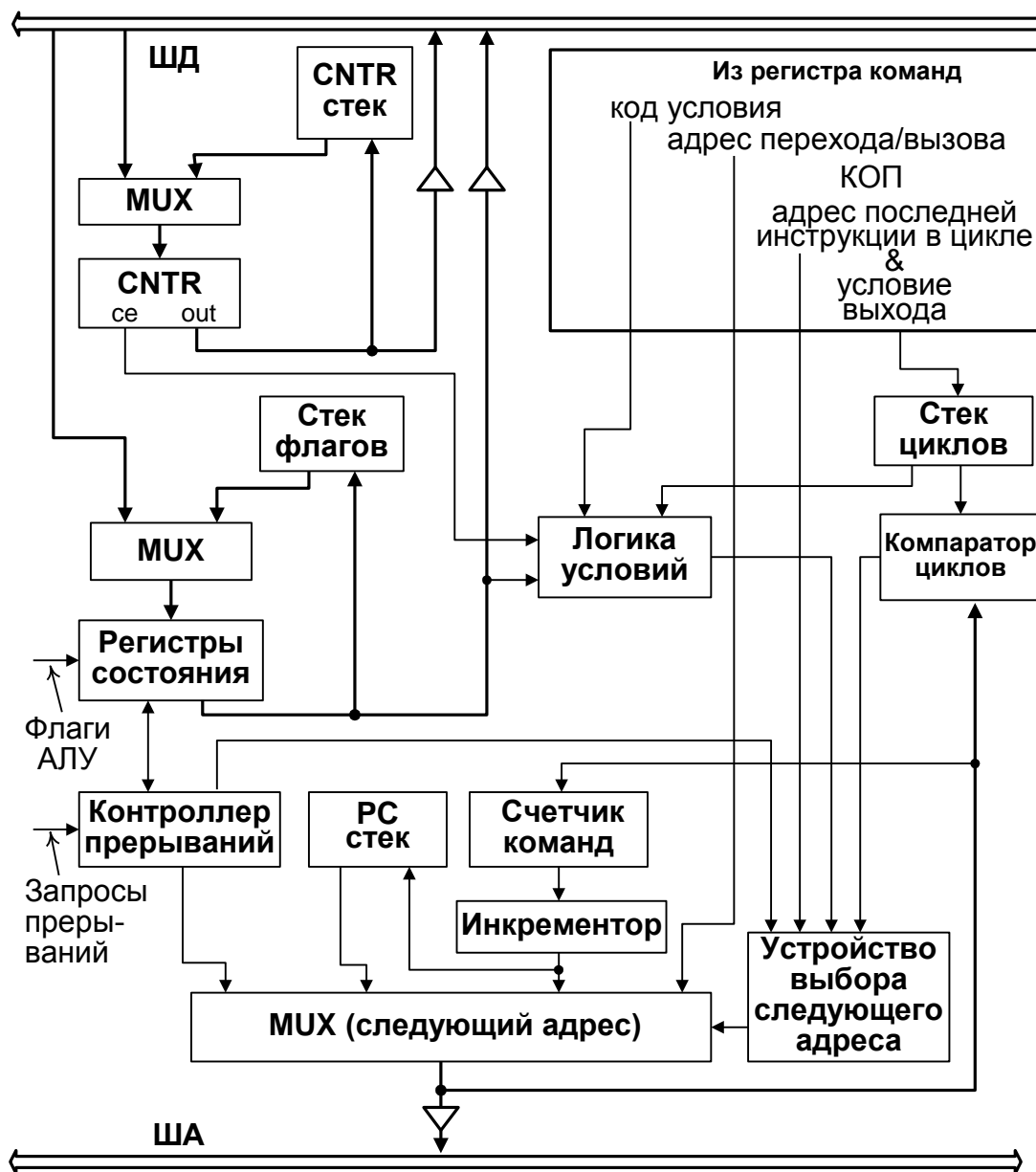


Рис. 7.6. Программный автомат МП семейства ADSP 21xx.

В качестве примера устройства управления выполнением программы можно взять программный автомат (*Program Sequencer*) цифрового процессора сигналов семейства ADSP 21xx фирмы Analog Devices (рис. 7.6). Его

особенностью являются аппаратные (интегрированные в кристалл МП) стеки для хранения адресов возврата из процедур и поддержки циклов. Рассмотрение этого примера обусловлено еще и тем, что в нем компактно на аппаратном уровне представлены все основные действия процессора, связанные с управлением выполнением программы.

Пока процессор выполняет текущую команду, программный автомат (ПА) осуществляет предварительную выборку следующей команды. Адрес этой команды определяется одним из четырех источников:

- инкрементор счетчика команд (*Program Counter – PC*),
- стек счетчика команд (*PC стек*),
- регистр команд *IR*,
- контроллер прерываний.

Каждый из этих источников связан с соответствующим портом мультиплексора следующего адреса. Коммутацией портов управляет устройство выбора следующего адреса, работа которого основывается на данных, поступающих

- 1) из регистра команд,
- 2) от логического устройства проверки выполнения условий переходов/вызовов,
- 3) от компаратора циклов, который проверяет адрес текущей команды на совпадение с адресом последней команды в теле цикла, что обеспечивает переход к его первой команде, если условие выхода из цикла не выполнено,
- 4) из контроллера прерываний.

Рассмотрим сначала работу устройства управления без участия контроллера прерываний, о функциях которого поговорим позднее после рассмотрения действий ПА, не связанных с реакцией на события. Такой режим работы может быть установлен в любом процессоре с помощью специального запрещающего восприятие запросов прерывания флага *IF – Interrupt Flag* в регистре (или в одном из регистров) состояния процессора.

При последовательном выполнении программы, когда не предпринимается переход или возврат из процедуры, или когда заканчивается цикл *DO UNTIL*, в качестве источника следующего адреса выбирается инкрементор счетчика команд *PC*. Выходное значение инкрементора выводится на шину адреса ША, по которой адрес передается в память, и одновременно загружается в *PC*, заменяя его предшествующее значение для последующей выборки команды из памяти.

Значение счетчика команд по-разному наращивается в микропроцессорах с *CISC* и *RISC* архитектурой. *CISC* процессоры управляются командами, длина которых зависит от того, где расположены операнды, и какой режим используется для их адресации. Это требует дополнительных операций, связанных с определением длины команды. *RISC* процессоры имеют фиксированный размер команд и значения программного счетчика

наращивается всегда на одно и то же число, что не требует операций по определению длины команды и, как следствие, повышает производительность МП. Микропроцессоры семейства ADSP 21xx, в формате команд которых отражены черты RISC архитектуры (все команды МП семейства ADSP 21xx упакованы в 24-разрядные слова), этими преимуществами обладают, и при линейном выполнении команды значения программного счетчика в них наращиваются на единицу.

При выполнении перехода по адресу, указанному в команде (команда типа JUMP), выход мультиплексора следующего адреса переключается на вход, связанный с регистром команд *IR*, и адрес перехода берется из команды, зафиксированной в *IR* во время предыдущего цикла выборки команды из памяти. Если переход условный, то предварительно *логика условий* проверяет выполнение условия перехода, сравнивая флаги АЛУ и указанное в коде операции условие. Переход реализуется, если условие выполняется, и откладывается в противном случае.

Точно также при выполнении команды вызова (команды типа CALL) из регистра команд берется адрес перехода на подпрограмму. Но при этом дополнительно используется стек *PC*, в который при выполнении команды вызова подпрограммы помещается значение инкрементора. При возвращении из подпрограммы по команде RTS выход мультиплексора следующего адреса переключается на вход, связанный со стеком *PC*, и сохраненный в стеке *PC* адрес используется как адрес возврата.

Стек *PC* используется при выполнении циклических операций по команде DO UNTIL. Адрес первой команды в теле цикла сохраняется в вершине стека *PC*. Одновременно адрес последней команды и условие завершения цикла размещаются в *стеке цикла*. При каждом очередном проходе тела цикла *компаратор циклов* сравнивает генерируемый программным автоматом адрес с адресом последней команды тела цикла (этот адрес содержится в команде DO UNTIL). При выполнении последней команды цикла процессор проверяет условие выхода из цикла и, если оно не выполнено, делает условный переход на начало цикла, выбирая в качестве источника адреса стек *PC*. Выполнение цикла контролирует логика сравнения флагов АЛУ с указанным в команде DO UNTIL условием. Для поддержки вложенных циклов используется *стек циклов*, на вершине которого всегда находятся параметры выполняемого в текущий момент времени цикла.

Циклы могут выполняться по заданному числу итераций. Для организации таких циклов используются *счетчик циклов (CNTR)* и *стек счетчика (CNTR стек)*. Счетчик цикла работает в режиме вычитания и перед исполнением цикла в него через шину данных ШД заносится число, задающее количество проходов тела цикла. Окончание цикла фиксирует *логика условий* по обнулению счетчика – по условию *CE* (счетчик пуст).

Стек счетчика циклов позволяет организовывать вложенные циклы. Каждый раз при выполнении условия *CE* (счетчик пуст) содержимое вершины

стека автоматически извлекается и загружается в счетчик цикла, что позволяет продолжить выполнение внешнего цикла (если таковой имеется). Для случаев, когда требуется преждевременный выход из цикла, предусматривается возможность непосредственного извлечения содержимого стека счетчика. Текущее значение счетчика автоматически помещается в стек при загрузке в него нового значения с шины данных ШД.

В стеке *PC* сохраняется также адрес инструкции, при исполнении которой возник запрос прерывания. При обработке прерываний кроме адреса возврата сохраняется состояние (флаги) процессора. Для этого используется стек флагов. При завершении процедуры обработки прерывания – состояние процессора восстанавливается¹³. Этим отличаются действия процессора при вызове процедуры обработки прерывания от действий при программном вызове.

Запросы прерываний от устройств ввода-вывода (УВВ), расположенных на кристалле процессора или за его пределами, принимает контроллер прерываний. Контроллер прерываний разбирается в приоритетах запросов и наиболее высокоприоритетный из них транслирует процессору. В схеме на рис. 7.6 сигнал о таком запросе поступает на устройство выбора следующего адреса, а адрес, по которому определяется путь к программе обработки запроса прерывания (*Interrupt Service Routine – ISR*), передается на вход мультиплексора следующего адреса. Получив сигнал от контроллера прерываний, устройство управления мультиплексором следующего адреса переключает выход мультиплексора на вход, связанный с контроллером прерываний, и на шину адреса ША передается сформированный контроллером адрес. Этот адрес относится к одному из элементов таблицы, расположенной в основной памяти и называемой *таблицей прерываний*.

Каждый из элементов таблицы прерываний содержит либо стартовый адрес *ISR* (или команду перехода на *ISR*), либо саму *ISR*, если размер ее программного кода не превышает размера элемента таблицы. Число элементов в таблице определяется числом устройств, которые процессор может обслуживать в режиме прерываний. Таблица прерываний обычно расположена, начиная с нулевого адреса памяти.

В качестве примера в таблице 7.7 представлена таблица прерываний МП ADSP 2171, каждый элемент которой содержит по четыре слова. Первому по списку элементу соответствует наиболее высокий приоритет, а последнему – наиболее низкий. Особенностью таблицы прерываний МП семейства ADSP 21xx является то, что длина слова в ней, как и длина слов в памяти, где расположена программа, равна 24 битам. При байтовой организации памяти

¹³ Стеки счетчика команд, циклов, счетчика циклов и состояния имеют ограниченное количество вложений. Однако глубина стеков подобрана таким образом, чтобы по возможности исключить обращение к памяти при выполнении наиболее часто выполняемых процедур. Обращение к памяти становится необходимым, если уровни вложений превышены.

адреса элементов в таблице прерываний определяются адресами соответствующих байтов.

Помимо аппаратных запросов прерывания в таблице представлены два вектора программных прерываний, использование которых вызывает действия, подобные действиям при вызове подпрограммы, но с сохранением состояния процессора в стеке.

Таблица 7.7

Источник прерывания	Адрес вектора прерываний
Запуск программы после сигнала «сброс» (или после выхода из режима пониженного энергопотребления)	0x0000 (высший приоритет)
Понижение потребляемой мощности	0x002C
Запрос прерывания от внешнего источника, поступивший на контакт микросхемы IRQ2	0x0004
Порт интерфейса с управляющей цифровой процессором машиной в режиме записи	0x0008
Порт интерфейса с управляющей цифровой процессором машиной в режиме чтения	0x000C
Интегрированный в МП кристалл последовательный порт SPORT0 в режиме «передача»	0x0010
Интегрированный в МП кристалл последовательный порт SPORT0 в режиме «прием»	0x0014
Программируемое прерывание	0x0018
Программируемое прерывание	0x001C
Интегрированный в МП кристалл последовательный порт SPORT1 в режиме «передача» или запрос прерывания от внешнего источника через контакт микросхемы IRQ1	0x0020
Интегрированный в МП кристалл последовательный порт SPORT1 в режиме «прием» или запрос прерывания от внешнего источника через контакт микросхемы IRQ0	0x0024
Интегрированный в кристалл МП таймер	0x0028 (низший приоритет)

Переходя к обработке запроса прерывания, процессор автоматически сбрасывает флаг прерываний *IF* в соответствующем регистре состояния процессора, запрещая тем самым обработку других запросов. Реакция на другие запросы возможна только после того, как процедура обработки текущего запроса установит флаг *IF* или тогда, когда выполнится команда RTI возврата из прерывания, по которой *IF* вместе с другими флагами процессора восстанавливается автоматически, благодаря чему вновь разрешается восприятие запросов прерываний. Именно этим обусловлены действия по сохранению регистров (или регистра, если он один) состояния процессора при вызовах ISR. Этот механизм обслуживания запросов прерываний является

универсальным и свойственен в основных своих чертах всем современным микропроцессорам.

7.5. Структура центрального процессора и взаимодействие с МП системой

Современные микропроцессоры отличаются разнообразием структурного и архитектурного построения, и всякое претендующее на полноту описание существующих МП и МП комплектов не может вполне отобразить это разнообразие. Сегодня на российском рынке представлены многочисленные микропроцессорные средства различных зарубежных компаний: микропроцессоры для персональных компьютеров и рабочих станций; микроконтроллеры (МК) и интегрированные процессоры (ИП) для встраиваемых систем. В первую очередь представляют интерес МК и ИП, ориентированные на встроенные применения для таких областей, как управление оборудованием, сетевые приложения, связь, портативная техника, автоматизация и т. п. Обширная номенклатура и богатые функциональные возможности МП, МК и ИП есть следствие удачного сочетания технологических решений и накопленных идей, методов и средств проектирования. Благодаря этому большие системы с развитой функциональностью и высокой вычислительной мощностью стало возможным интегрировать в сверхбольшие ИС. Преимущество в решениях (использование принципов открытых систем) и бережное отношение к имеющемуся опыту лишь увеличивает возможности разработчика по наращиванию вновь создаваемых систем.

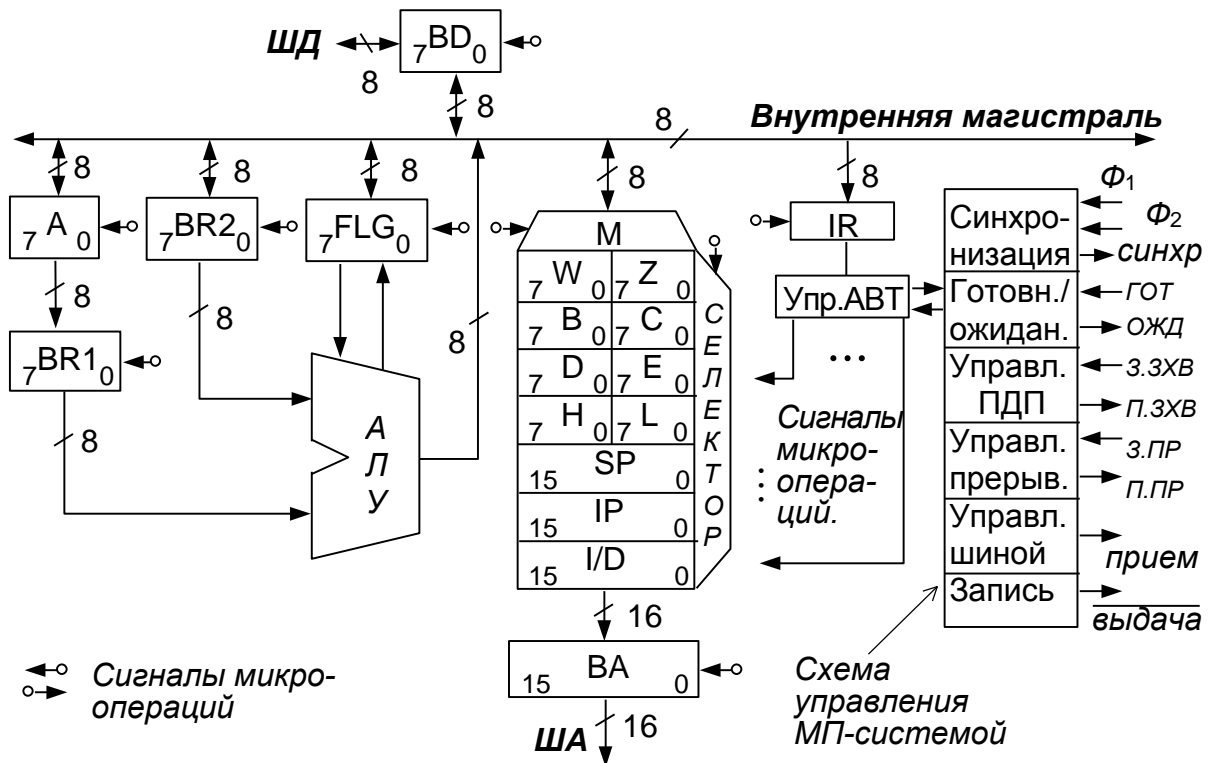


Рис. 7.7. Структура микропроцессора Intel-8080

Устройство ЦП и его взаимодействие с внешней по отношению к нему средой рассмотрим на примере 8-разрядного МП Intel-8080 (отечественный аналог – ЦП из МП комплекта К580), относящегося к классу процессоров со сложным набором команд (к CISC-процессорам). С появлением этого МП началось уверенное продвижение микропроцессоров как в уже освоенные цифровыми устройствами области, так и в области их новых применений. Процессор содержит все необходимые для решения сложных алгоритмических задач компоненты, включая микропрограммный автомат, регистровое АЛУ и рабочие регистры, выполняющие функции регистров данных и регистров адреса, а также счетчик команд, указатель стека и регистр состояния (регистр флагов).

Компоненты МП Intel-8080 (рис. 7.7) объединены *внутренней 8-разрядной информационной магистралью* (шиной). К числу основных компонентов процессора относятся:

- *Регистровое АЛУ* с регистром-аккумулятором *A*, буферными регистрами *BR1* и *BR2* и регистром флагов (признаков) *FLG*, а также с комбинационной схемой АЛУ, реализующей логику его работы.

- *Блок 8-разрядных рабочих регистров B, C, D, E, H и L* (регистры *W* и *Z* программно недоступны и выполняют только внутренние служебные функции), которые могут быть объединены в 16-разрядные пары *B-C*, *D-E* и *H-L*. Здесь, как и для любого другого МП, *имена регистров уникальны*. Имена регистрам «присваиваются» на стадии разработки процессора с тем расчетом, чтобы использовать их при написании исполняемых микропроцессором программ. Такие программы создаются с применением архитектурно зависимого (низкоуровневого) программирования, т. е. программирования в машинных (двоичных) кодах или на языке *Ассемблера*. Информация в рабочие регистры (или из них) передается через мультиплексор *M*, а для адресации (выбора) регистров используется дешифратор (селектор).

- *Регистр слова состояния процессора* (регистр флагов *FLG*), *счетчик команд IP* и *указатель стека SP*, а также *инкрементатор/декрементатор I/D*, используемый при образовании адреса следующей инструкции по счетчику команд *IP* и при работе с указателем стека *SP*;

- *Управляющий автомат с регистром команд IR* на входе.

- *Двухнаправленный буфер данных BD* с тремя состояниями, который с одной стороны соединен с внутренней магистралью процессора, а с другой – с образующими шину данных *ШД* контактами на корпусе микросхемы процессора.

- *Буфер адреса BA* с тремя состояниями, входы которого обращены к регистровому блоку, а выходы соединены с адресными выходами (контактами) ИС процессора, формирующими шину адреса *ША*.

- *Схема управления микропроцессорной системой*, которая формирует, передает во внешнюю среду и принимает из внешней среды сигналы,

необходимые для синхронизации работы процессора и его связи с постоянной, оперативной и дисковой памятью, а также с устройствами ввода/вывода.

Пояснения требуют сигналы, с помощью которых процессор взаимодействует с внешними по отношению к нему компонентами МП системы. К ним помимо сигналов, идущих по шинам адреса и данных (*ША* и *ШД*), относятся:

1. *Сигналы* внешней синхронизации Φ_1 и Φ_2 . В данном случае имеются две фазы синхронизации. Современные процессоры, как правило, синхронизируются одним сигналом. Более того, внутренняя тактовая частота МП может значительно превышать частоту внешнего синхросигнала. В последнем случае внешний синхронизирующий сигнал является опорным для внутреннего тактового генератора МП.

2. *Сигнал «синхр»* – используется при передаче во внешнюю среду информации о состоянии процессора, точнее о том, что ЦП будет делать при очередном обращении к устройствам в составе МП системы: 1) обращаться к памяти в основном машинном цикле – цикле выборки команды из памяти и размещении ее в *IR*, 2) обращаться к памяти в режиме чтения или записи данных, 3) обращаться к внешним устройствам в режиме ввода или вывода и т. д. Обозначенная информация в МП Intel-8080 передается по шине данных, а сигнал «*синхр*» является сигналом сопровождения, по которому данные с *ШД* фиксируются в предназначенном для этой цели регистре, входящем в состав интерфейса (электронного устройства сопряжения) микропроцессора с МП системой. Здесь важно подчеркнуть, что процессоры имеющие средства оповещения о своем состоянии, делают это, используя разные механизмы, но все они служат тому, чтобы обеспечить интерфейс ЦП с другими компонентами МП системы.

3. *Сигналы ГОТ и ОЖД* – предназначены для работы с памятью. В том случае когда процессор, обратившись к памяти, не обнаруживает от нее сигнала готовности *ГОТ*, он переходит в режим ожидания данных, посылая памяти сигнал *ОЖД*.

4. *Две группы сигналов 3.ЗХВ, П.ЗХВ и 3.ПР, П.ПР* – служат для реализации двух режимов обмена данными с внешними устройствами (ВУ) в МП-системе – режима прямого доступа к памяти (*ПДП*) и режима прерывания (*ПР*). Эти сигналы обычно формируются соответствующими контроллерами – *контроллером прямого доступа к памяти (КПДП)* и *контроллером прерываний (КПр)*. Контроллеры принимают запросы от ВУ и затем ретранслируют их ЦП с учетом приоритетов обслуживания.

Существенным для *режима ПДП* является то, что *КПДП* может реализовать свою функцию по пересылке данных от внешнего устройства (во внешнее устройство) лишь тогда, когда ЦП предоставит ему эту возможность – отдаст системную магистраль (системную шину), которой ЦП управляет и по которой перемещаются информационные потоки МП системы (см. раздел 9.1), в распоряжение *КПДП*. Запрос на предоставление магистрали под свое управление *КПДП* посылает процессору в виде сигнала *3.ЗХВ* – *запрос захвата*.

В ответ на этот сигнал ЦП переводит свои буферы адреса и данных *BA* и *BD* в третье состояние, отсоединяясь тем самым от системной магистрали, и с помощью сигнала *П.ЗХВ* – *подтверждение захвата* – передает управление магистралью контроллеру *ПДП*. По завершении цикла *ПДП* контроллер снимает *З.ЗХВ*, а процессор в ответ на это снимает *П.ЗХВ*.

Режим прерываний требуется для тех ВУ, которые обмениваются данными непосредственно с МП, но при этом сами инициируют процедуру обслуживания с помощью сигналов запроса, обращенных к контроллеру прерываний *КПр*. Учитывая приоритеты всех поступивших запросов, *КПр* ретранслирует наиболее высокоприоритетный из них процессору в виде сигнала *З.ПР* – *запрос прерывания*. Цель запроса *З.ПР* состоит в том, чтобы процессор приостановил выполняемую программу и перешел на выполнение процедуры обработки поступившего от ВУ запроса. Что это за устройство, *КПр* сообщает процессору в виде кода, передаваемого по шине данных в ответ на сигнал *П.ПР*.

Перечисленные сигналы *не являются отличительными* для рассматриваемого процессора. Они в том или ином виде генерируются и воспринимаются любыми МП. В отличие от них сигналы «*прием*» и «*выдача*», смысл которых заключен в самом их названии, специфичны для МП Intel-8080.

Чтобы подчеркнуть различие между структурой и архитектурой МП приведем программную модель МП Intel-8080 (рис. 7.8). Она содержит все выше

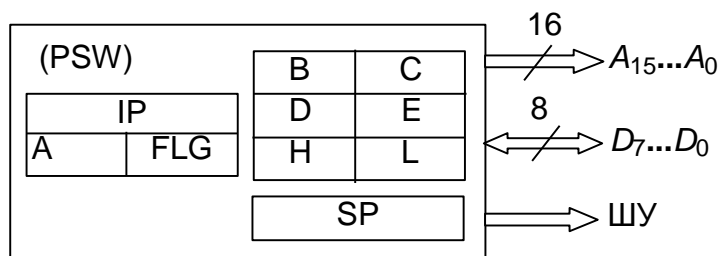


Рис. 7.8. Программная модель МП Intel-8080

перечисленные регистры и слово состояния *PSW*. В отличие от общепринятого для слова состояния ЦП формата, *PSW* МП Intel-8080 помимо флагов, размещенных в регистре *FLG*, включает счетчик команд *IP* и аккумулятор *A*. Такой состав *PSW* – особенность данного процессора. Обычно принято в

слово состояния ЦП включать флаги АЛУ, а также другие флаги, число и назначение которых зависят от конкретной разновидности ЦП. Сложные процессоры для отображения состояния могут иметь несколько регистров.

8. Регистрово-ориентированные архитектуры

Существенной их особенностью CISC процессоров являются относительно сложный микропрограммный управляющий автомат и небольшое число программно доступных регистров. Инструкции (команды) для таких процессоров отличаются достаточно большим разнообразием и по размеру могут изменяться в значительных пределах – от одного до нескольких байт. Требования высокой производительности заставляют обращаться к архитектурам, обладающим значительной регистровой памятью и командами, при выполнении которых можно эффективно использовать распараллеливание и конвейеризацию обработки данных. Такими свойствами обладают RISC процессоры.

8.1. Типы операндов и иерархия памяти

Обрабатываемые процессорами операнды можно разделить на три основные категории:

1) *Константы* – не меняются во время выполнения программ и имеют обычно небольшие значения.

2) *Скаляры* – как правило, обращение к ним производится явно, по их собственному имени; их обычно немного и они описываются в процедурах как локальные.

3) *Элементы массивов и структур* – обращение к ним производится путем косвенной адресации, при помощи индексов и указателей; их может быть довольно много.

Запоминающие устройства, которые интересуют нас в данном случае, – это 1) регистровый блок, 2) кэш-память и 3) основная память.

Процесс доступа к операндам состоит из двух частей, причем обе они должны учитываться при оптимизации скорости доступа. Прежде всего необходимо вычислить адрес, т. е. то место, где хранится искомый операнд. Только после этого может быть произведен сам доступ к операнду.

Регистровые блоки традиционно имеют меньший, чем кэш-память, размер и, следовательно, меньшую задержку доступа. Кэш-память имеет более высокую стоимость в расчете на слово, чем регистровые блоки, поскольку требует дополнительных схем для организации хранения тегов, параллельного чтения многих групп слов и для реализации логики сравнения и замещения. Если по времени доступа кэш-памяти в принципе могут почти не уступать регистровым блокам, то «накладные расходы» на адресацию – и соответственно задержка – для кэш-памяти значительно выше. Обращение к кэш-памяти осуществляется при помощи адресов памяти полной длины, и благодаря этому кэш-память прозрачна для программиста на машинном уровне. А регистровые блоки адресуются короткими номерами регистров, так что они должны быть

видимы на машинном уровне. Короткие номера регистров обычно указываются в команде, поэтому их декодирование производится очень просто и быстро.

Результаты измерения программ показывают, что из каждых ста операндов языка высокого уровня (ЯВУ), к которым осуществляется обращение во время выполнения, примерно 25 – константы, 50 – скаляры и 25 – данные структурного типа. Если исключить из рассмотрения константы, поскольку их можно помещать в команды, остается соотношение ссылок 50 скалярных на 25 структурных. Однако для каждой структурной ссылки ЯВУ необходим доступ минимум к одной скалярной переменной – индексу массива или указателю «структуры» («записи»). Таким образом, на уровне машинного языка получается минимум $50+25=75$ скалярных ссылок на 25 структурных.

Из-за большого различия между регистровыми блоками и кэш-памятью по общей скорости доступа и большой частоты использования скаляров в программах распределение скалярных переменных по регистрам имеет важное значение для увеличения быстродействия. Чтобы добиться достаточно высокой эффективности распределения переменных по регистрам, необходимо решить ряд проблем. Одна из таких проблем касается запоминания и восстановления локальных скалярных переменных с соответствующими обменов между регистрами и памятью при вызовах и возвратах из процедур. Здесь цель видится в том, чтобы свести к минимуму «накладные расходы» на вызовы процедур.

8.2. Многочисленные и перекрывающиеся окна регистров

Распределение аргументов процедур и скалярных переменных по регистрам приводит к существенному выигрышу во времени выполнения благодаря высокой скорости доступа к регистрам. С другой стороны, необходимость запоминания содержимого регистров в памяти при вызовах процедур и восстановления при соответствующих возвратах снижают производительность. Как минимум содержимое тех регистров, которые используются родительской процедурой и к тому же должны использоваться дочерней процедурой для хранения собственных переменных, необходимо направить в память до того, как дочерняя процедура произведет их перезапись. Вызовы процедур в современных структурированных программах осуществляются весьма часто, так что расходы на подобное запоминание/восстановление никак нельзя игнорировать.

Хотя индивидуальные вызовы/возвраты производятся весьма часто, большие колебания динамической глубины вложенности процедур для типичных программ нехарактерны. Это свойство иногда называют *локальностью глубины вложенности процедур*.

Свойство локальности вложения можно с успехом использовать, если в процессоре предусмотреть ряд наборов, или банков регистров. При вызовах

процедур производится выделение нового банка регистров, благодаря чему исключаются «накладные расходы» на запоминание/восстановление содержимого регистров. По завершении процедуры соответствующий банк регистров может быть предоставлен новой процедуре. Кроме того, желательно также ускорить передачу параметров и значений возврата между родительскими и дочерними процедурами, используя для этой цели регистры. Для этого необходимо, чтобы к определенным регистрам имели доступ и родительская, и дочерняя процедуры, другими словами, чтобы банки регистров обеих процедур перекрывались. Удобным способом реализации подобной возможности является механизм с многими перекрывающимися «окнами», накладываемыми на непрерывный блок регистров.

В процессоре с многочисленными регистровыми окнами имеется *указатель текущего окна*, который выбирает одно из окон и делает его регистры доступными на период выбора. При вызовах и возвратах из процедур значение указателя текущего окна модифицируется таким образом, чтобы указывать на следующее или предыдущее окно соответственно. Когда глубина вложения процедур превышает количество окон w , происходит (положительное) переполнение и содержимое некоторых старых окон приходится направлять в память, чтобы освободить место для новых процедур. Желательно организовать управление окнами как циклическими буферами, в которых хранились бы w записей об активации самых последних процедур. Пока глубина вложения колеблется в пределах w , ни одно окно не приходится записывать в основную память или восстанавливать из памяти.

Подобная организация регистрового блока используется в микропроцессорах SPARC и SuperSPARC фирмы Sun Microsystems. Регистровый блок процессоров этого семейства является 9-ти оконным.

8.3. Наборы команд, ориентированные на регистровую архитектуру

Измерения параметров программ свидетельствуют о простоте большинства операций, встречающихся в вычислениях общего характера, и о высокой доле числа обращений к скалярным операндам. Соответствующая архитектура должна быть ориентирована на эффективное распределение скалярных переменных по регистрам. Чтобы обеспечить распределение наиболее часто используемых операндов по регистрам с учетом простоты наиболее часто встречающихся операций, была предложена регистрово-ориентированная RISC-архитектура компьютера с сокращенным набором команд. Подобная архитектура имеет простые команды, распадающиеся на ортогональные подмножества: 1) команды типа «регистр–регистр», 2) команды для пересылки данных между регистрами и памятью, 3) простые команды передачи управления. Все эти команды можно реализовать при помощи

эффективного конвейерного тракта обработки данных и несложной схемы синхронизации и управления.

Операции типа «регистр-регистр»

К простейшим операциям относятся те, которые выполняются над данными, находящимися внутри МП. Свойственные регистровой архитектуре команды типа «регистр-регистр» в наиболее общем виде можно записать следующим образом:

$$R_d \leftarrow R_{s1} \text{ op } R_{s2} \quad \text{и} \quad R_d \leftarrow R_s \text{ op константа.}$$

Здесь R_d – регистр-приемник; R_s , R_{s1} и R_{s2} – регистры-источники, *op* – производимая операция (арифметическая, логическая и др.). Такие команды являются трехадресными (рис. 7.1б), но отличаются относительно коротким форматом, поскольку операндами являются регистры. Второй формат играет важную роль, т. к. позволяет задавать константы непосредственно в командах. Эти форматы как особый случай $d = s1$ охватывают варианты с двумя операндами $R_A \leftarrow R_A \text{ op } B$, где B может обозначать регистр R_B или константу. Они охватывают также регистровые команды пересылки и сравнения. Если $R_{s2} = 0$ и оператор *op* «+», образуется команда «переслать». Если оператор «-» и R_d вначале проверяется, а затем отбрасывается, то образуется команда «сравнить». Эти особые случаи, которые в программах встречаются довольно часто, можно закодировать как самостоятельные и компактные команды. Однако подобное кодирование разрушает простой и унифицированный однословный формат команды, дающий возможность ее быстрого и экономичного декодирования. Особенности RISC-архитектуры рассмотрим на примере одного из первых RISC-процессоров, МП RISC II Калифорнийского университета (г. Беркли, США), аттестованного в 1983 г.

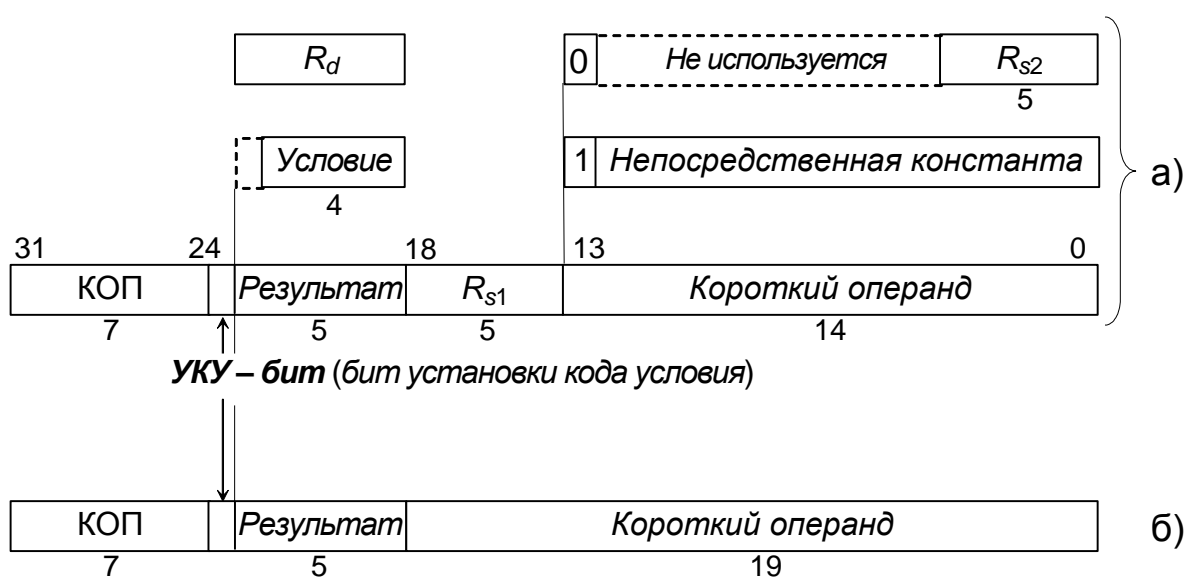


Рис. 8.1. Форматы команд МП RISC II: (а) – типа «регистр-регистр» и (б) – передачи управления

На рис. 8.1 показаны форматы команд этого процессора. Заметим, что команды с подобными форматами использовались и используются в промышленных образцах МП, например, в МП i860 фирмы Intel, Alpha фирмы DEC, SPARC фирмы Sun Microsystems). По структуре к ним близки команды МП Nios II фирмы Altera.

МП RISC II имеет *многооконный регистровый блок*. Программисту доступны 32 регистра текущего окна. Поэтому поля номеров регистров имеют длину 5 бит. В наиболее общем виде команды типа «регистр–регистр» соответствуют формату рис. 8.1а. Поле кода операции (КОП) содержит 7 бит. Поле «результат» указывает R_d , за ним следует поле операнда R_{s1} . Младшие 14 бит команды могут указывать R_{s2} или содержать 13-бит константу (большинство констант имеют малую длину). Один из 32 видимых регистров, R_d , всегда содержит (аппаратно реализованную) константу «0»; запись в этот регистр эквивалентна аннулированию записываемого значения. Бит 24 всех команд – это признак «установить коды условия» (УКУ-бит): если этот бит имеет значение 1, то производится установка признаков нуля, отрицательного результата, арифметического переполнения и переноса соответственно результату выполнения данной операции. Эти признаки используются при синтезе команд проверки и сравнения.

В архитектуре RISC университета Беркли реализованы целочисленные операции *ор* сложения и вычитания, булевы операции над битами и сдвиги на произвольное количество разрядов. Нет команд пошагового умножения и пошагового деления и операций с плавающей точкой. Таким набором команд, как правило, располагают современные RISC-процессоры, а для операций с плавающей точкой используются преимущественно специализированные вычислительные блоки или сопроцессоры.

В набор команд МП RISC II включены сдвиги на произвольное количество разрядов. Выполняются они при помощи матричного сдвигателя. В реальных программах такие сдвиги встречаются довольно редко. Если строго придерживаться концепции RISC, то этот функциональный блок следовало бы исключить из тракта обработки данных. Желательно иметь сдвиги на один бит – они встречаются в программах гораздо чаще, и их можно экономично реализовать в арифметическо-логическом устройстве.

RISC-процессор университета Беркли имеет 32-бит архитектуру с побайтовой адресацией *виртуальной памяти* объемом 2^{32} байт. Все команды типа «регистр–регистр» обрабатывают свои операнды как 32-бит целые в дополнительном двоичном коде. Символы и полусловные целые при загрузке в регистры преобразуются в полные слова.

Команды обращения к памяти и передачи управления

RISC-архитектуре не свойственны команды, которые непосредственно работают с операндами, находящимися в основной памяти. Они предусматривают только простые команды обмена данными: загрузить и

запомнить. Сложные манипуляции с данными разбиваются на ортогональные компоненты, относящиеся к обмену и обработке. Это упрощает формат команд, временную диаграмму, конвейеризацию и обработку прерываний.

Команды обращения к памяти для МП RISC II тесно связаны с командами передачи управления (переход, вызов и возврат). Команды записи в память имеют вид $M[ИспАдр] \leftarrow R_d$, команды загрузки регистра – $R_d \leftarrow M[ИспАдр]$; а команды передачи управления – $IP \leftarrow ИспАдр$, где *ИспАдр* – это исполнительный адрес, $M[ИспАдр]$ – соответствующая ячейка памяти, а *IP* – программный счетчик. Возможные способы определения исполнительного адреса являются важной характеристикой архитектуры – они называются режимами адресации.

1) Для *глобальных скаляров* $ИспАдр = константа$; эта константа может иметь малую величину, если компилятор распределяет все глобальные скаляры вблизи адреса 0.

2) Для элемента $a[i]$ линейного глобального массива $ИспАдр = a_база + размер \cdot i$, где *a_база* – базовый адрес $a[]$ (константа, которая может быть большой), а *размер* – размер элемента массива (константа, равная, как правило, 2, 4 или 8).

3) Для элементов массива или других данных, доступ к которым осуществляется по *указателю*, $ИспАдр = указатель$. Этот способ адресации обычно более быстрый, чем предыдущий. Он часто используется для последовательного просмотра массивов – особенно символьных буферов.

4) Для поля (члена) некоей структуры (записи) $ИспАдр = p + смещение$, где *p* – указатель данной структуры, а *смещение* – смещение, указанное в команде (константа, обычно короткая). Этот же режим адресации применим для локальных скаляров, для хранения которых используется исполнительный стек.

5) Для большинства передач управления $ИспАдр = IP + расстояние\ передачи$; расстояние передачи – обычно короткая указанная в команде константа. В программах встречаются и другие более сложные режимы адресации (например, при работе с многомерными массивами или при многоуровневой косвенной адресации).

RISC-архитектура университета Беркли предусматривает следующие режимы адресации (сравни с режимами, представленными в таблице 7.1):

$$ИспАдр = R_{s1} + R_{s2},$$

$$ИспАдр = R_{s1} + 13\text{-бит константа},$$

$$ИспАдр = IP + 19\text{-бит константа}.$$

Эти режимы адресации в совокупности с использованием регистра R_0 ($\equiv 0$) охватывают все основные способы формирования адресов, перечисленные выше (в предположении, что значение *a_база* хранится в регистре), за исключением так называемой шаговой адресации (со сдвигом на размер элемента массива). Приведенное выше вычисление исполнительного адреса очень похоже на команду сложения, что в значительной мере способствует унификации порядка выполнения команд, а благодаря этому – достижению

простоты тракта обработки данных, конвейеризации, временной диаграммы и управления.

Команды, использующие первые два режима адресации, соответствуют формату, показанному на рис. 8.1а, а команды с режимом относительной адресации с значением программного счетчика как базы имеют формат с длинным кодом непосредственного адреса, показанный на рис. 8.1б. Команды условного перехода содержат условие перехода в поле «результат» (рис. 8.1). Команды вызова запоминают значения программного счетчика в R_d , меняют значение указателя текущего регистрового окна после проверки на положительное переполнение, а затем осуществляют безусловную передачу управления по целевому адресу. Команды возврата меняют значение указателя текущего окна после проверки на отрицательное переполнение, а затем передают управление по значению программного счетчика, которое соответствующая команда вызова запомнила в регистровом блоке.

8.4. Конвейеризация и регистровая память

Конвейеризация – весьма распространенный и эффективный способ повышения быстродействия машины. Главная проблема конвейерной архитектуры – необходимость учитывать взаимозависимости данных, обрабатываемых различными командами на различных уровнях, ступенях конвейера. Традиционный подход к решению этой проблемы заключается в том, что подобные взаимозависимости обнаруживаются аппаратно, после чего выполнение последующих команд приостанавливается в ожидании момента, когда появятся необходимые результаты предыдущих операций. Такой подход, главная цель которого состоит в том, чтобы сделать реализацию конвейера невидимой с архитектурного уровня, не всегда является оптимальным, поскольку может вносить необязательные задержки.

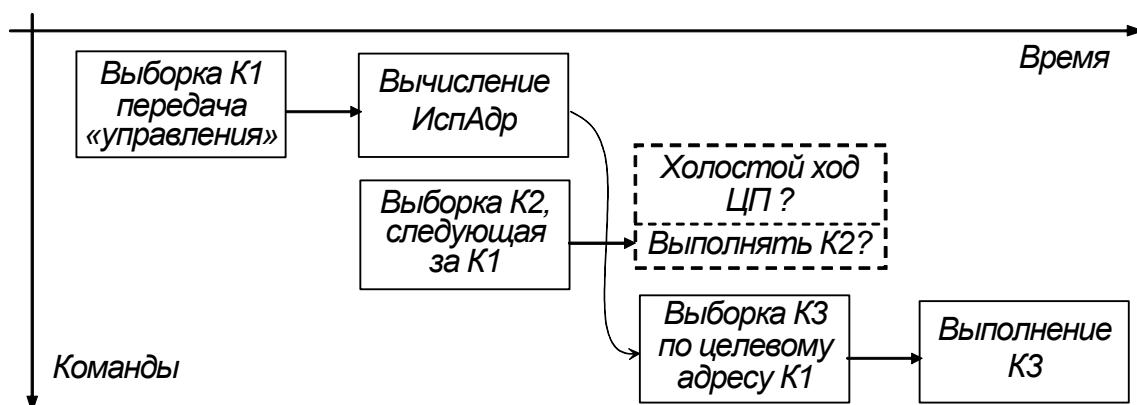


Рис. 8.2. Совмещение выборки и выполнения команд и задержанный переход управления

В качестве примера можно привести «задержанный переход» («ветвление и выполнение»). На рис. 8.2 показан простой двухуровневый конвейер, образуемый в результате совмещения выполнения текущей команды с выборкой следующей команды. Когда выполняется команда условной передачи управления $K1$ при удовлетворении указанного условия, ее целевую команду $K3$ невозможно выполнить в течение ближайшего следующего такта конвейера. В традиционных архитектурах МП находится в режиме холостого хода во время выборки $K3$. В процессоре RISC II во время выполнения команды $K1$ производится выборка команды, следующей непосредственно за $K1$, а затем эта команда выполняется при одновременной выборке команды по целевому адресу $K1$. Компилятор, оптимизатор или «реорганизатор» конвейера стремятся вставить после команды управления $K1$ какую-либо полезную команду $K2$, не зависящую от $K1$. Благодаря этому после передач управления приблизительно две трети квантов времени, которые в противном случае просто терялись бы, используются для выполнения полезной работы.

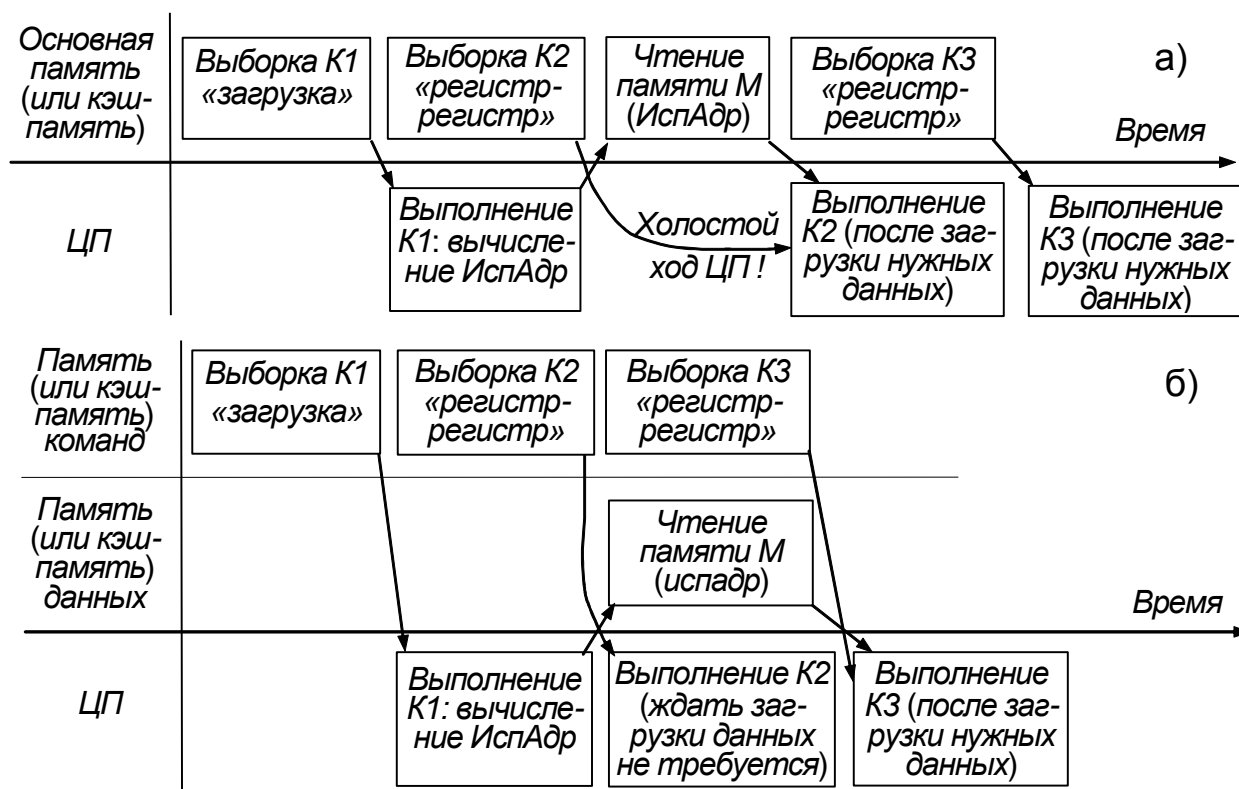


Рис. 8.3. Выполнение команд обращения к памяти для случаев одной общей памяти (а) и раздельной памяти команд и данных (б)

Этот способ задержанного ветвления служит одним из примеров того, каким образом удастся обеспечить занятость МП в течение максимально возможного числа тактов при использовании набора команд, содержащего лишь простые команды с описанием примитивных операций. Разработчики компьютеров со сложными наборами команд CISC стремятся добиться такого

же эффекта, комбинируя несколько примитивных операций в одной сложной команде, а затем микропрограммируя параллельное выполнение этих примитивов.

Аналогичная ситуация возникает в случае, когда нужно обеспечить занятость МП во время выборки данных из памяти. Рис. 8.3 иллюстрирует эту ситуацию на примере некоторых команд компьютера типа RISC. Все подобные команды описывают простые, примитивные операции МП, для выполнения которых требуется один такт МП, совмещаемый с одним обращением к основной памяти (или кэш-памяти). На рис. 8.3а показан случай, когда имеется одна шина обмена МП с общей памятью для команд и для данных, и поэтому МП не может производить выборку новой команды во время обращения к памяти за данными и ему приходится простаивать. Во многих архитектурах эту проблему пытаются решить путем комбинирования нескольких примитивных операций в одной команде, с тем чтобы некоторые из этих операций можно было выполнять во время обращения к памяти за данными (примером могут служить команды загрузки с инкрементом значения указателя). Возможен другой путь: предусмотреть в МП два отдельных порта для доступа к командам и данным, как показано на рис. 8.3б. Благодаря этому ни один такт МП не теряется (пока есть работа, которую МП может сделать и которая не зависит от загружаемых данных), набор команд остается простым и достигается гибкость с точки зрения совмещения обращений к основной памяти данных и любых других примитивных операций. В машине RISC университета Беркли команды загрузки и запоминания употребляются гораздо реже, чем обычно, благодаря многооконному регистровому механизму.

Распределение часто используемых скаляров по регистрам упрощает выполнение наиболее употребительных операций, делая эти операции примитивами типа «регистр–регистр». Малая частота употребления сложных операций делает нецелесообразной их поддержку при помощи специальной аппаратуры, поскольку это приводит к неоправданному увеличению объема схем и длительности такта машины. Конвейеризация является более эффективным решением, обеспечивающим занятость центрального процессора. Простой формат команды, разрядность которых совпадает с числом линий шины выборки команды, позволяет добиться высокой скорости и эффективности выборки и декодирования команд. Таким образом, в целом применение RISC-архитектуры обеспечивает резкое уменьшение объема схем управления.

8.5. Микроархитектура процессора RISC II

Микроархитектура процессора, т. е. организация его аппаратуры в конкретной реализации, в существенной степени определяет быстродействие компьютера в целом. В настоящем разделе описан трехуровневый конвейер и тракт обработки данных, а также секции управления МП RISC II.

8.5.1. Трехуровневый конвейер микропроцессора RISC II

На рис. 8.4 показана схема конвейера операций, реализованная в МП RISC II. Представлены последовательности команд типа «регистр–регистр» и передач управления. Каждая команда типа «регистр–регистр» выполняется как ряд следующих действий:

- 1) выборка команды из основной памяти или из кэш-памяти команд;
- 2) дешифрация двух входящих в формат команды полей номеров регистров-операндов (*чт. рег*), одновременное декодирование остальных полей команды и направление непосредственной константы в тракт обработки данных;
- 3) выполнение операции АЛУ или сдвига согласно указанному в команде коду операции (КОП);
- 4) запись результата операции в регистр-потребитель, регистр результата (*зп. рег*).

Команды передачи управления выполняются аналогичным образом с тем отличием, что на последнем этапе значение результата используется как исполнительный адрес для выборки команды. В качестве регистра операнда эти команды часто используют программный счетчик.

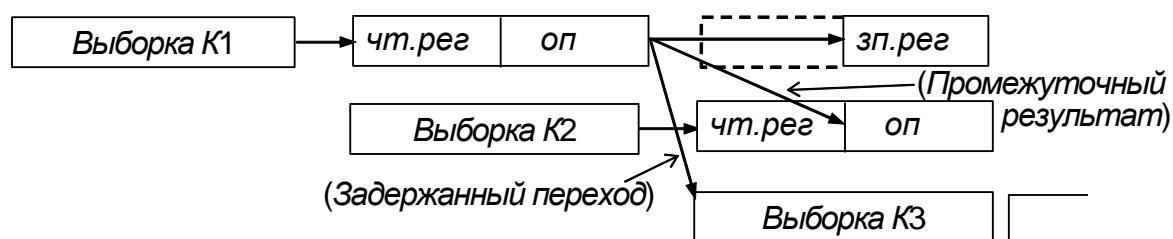


Рис. 8.4. Конвейер операций процессора RISC II

На первом уровне конвейера производится выборка команды. На втором уровне читаются регистры операндов и выполняются операции. Результат записывается в регистр результата на третьем уровне параллельно с выполнением операции, указанной следующей командой. При таком подходе возникает взаимозависимость по данным. Это случается в тех случаях, когда регистр операнда в команде *K2* совпадает с регистром результата команды *K1*. Эту ситуацию необходимо обнаруживать при помощи специальной аппаратуры, а результат команды *K1* должен предоставляться команде *K2* путем *внутренней передачи с уровня на уровень конвейера* или *организации подходящей конвейерной цепочки*, например, без выполнения обычной пары операций записи в регистр и чтения из регистра.

Операции чтения и записи для регистрового блока выполняются одновременно, а время выборки команды должно быть таким же коротким, как время выполнения операции АЛУ. Что касается тракта обработки данных

RISC II, то его регистровый блок имеет два порта чтения и один порт записи, связанный с двумя отдельными шинами чтения/записи, которые рассчитаны на чтение/запись с предварительным зарядом. Конвейер с большим числом уровней предъявляет более серьезные требования к регистровому блоку и к механизму выборки команд.

8.5.2. Тракт обработки данных микропроцессора RISC II

Конвейер операций требует соответствующего тракта обработки данных. Существуют некоторые основные временные зависимости, которые определяют организацию конвейера и ограничивают его предельное быстродействие. Они зависят от выбранного набора команд и принятого способа конвейеризации, но сравнительно мало зависят от конкретных деталей тракта обработки данных.

На рис. 8.5 показана схема тракта обработки данных МП RISC II. В *многооконном* регистровом блоке процессора программисту доступны 32 регистра текущего окна. Чтение регистров блока осуществляется по двум портам – через шины **A** и **B**. Один операнд поступает на вход АЛУ по шине **D**. Это может быть значение либо шины **A**, либо программного счетчика *IP* (в случае адресации относительно *IP*). Вторым операндом АЛУ поступает либо по цепи *шинаB*→*блокSRC*→*шинаR*, либо по цепи *константа*→*шинаL*→*шинаR*. Результат АЛУ может быть выдан либо на выходную шину *OUT* и использован в качестве исполнительного адреса, либо на шину **D** и принят в *DST*. Из блока *DST* этот результат записывается в регистровый блок во время последнего шага конвейера. На шины *OUT* и **D** может также выдаваться содержимое регистров программного счетчика (имеется три варианта значений *IP* – по одному на уровень конвейера). Значение *IP* выдается на шину *OUT* при нормальных выборках команд, а на шину **D** (вместо значения АЛУ) при выполнении команды вызова процедуры, когда значение *IP* необходимо запомнить в регистре. При выполнении операций сдвига значение шины **A** принимается в блок *SRC*, а затем поступает либо на шину **R** сдвигателя, либо на его шину **L**. Результат образуется на второй из этих шин сдвига и принимается в блок *DST*. По командам загрузки, использующим цепь *данные/конст.*→*шинаL*→*шинаR*→*блокDST*, производится выравнивание данных, поступающих из памяти, с использованием сдвигателя.

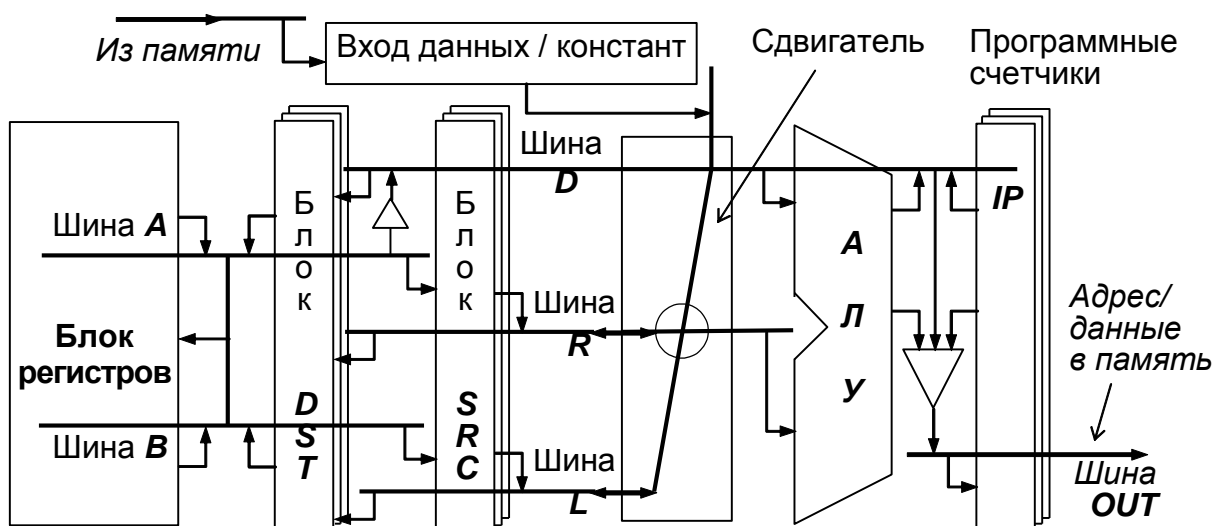


Рис.8.5. Тракт обработки данных микропроцессора RISC II

По сути представленный на рис. 8.5 тракт обработки данных является регистровым АЛУ, приспособленным для работы в условиях конвейера операций. Одиночные буферные регистры обычного регистрового АЛУ (см. рис. 4.18) в конвейерном тракте обработки заменены блоками *SRC* и *DST* с числом регистров, равным числу уровней конвейера. Числом уровней конвейера определяется также и число версий программного счетчика *IP*.

Особенностью данного АЛУ является также то, что оно выполняет и операции с данными, и операции с адресами, реализуя все режимы адресации данных и программы. В настоящее время, как это будет видно из дальнейшего рассмотрения, эти функции стремятся разделить, поручив адресную арифметику и операции с данными разным устройствам в составе МП. Более того, отдельными устройствами обеспечивают адресацию данных и адресацию программного кода.

Добавленный в тракт обработки данных матричный сдвигатель на произвольное число бит служит как для передачи непосредственных констант в АЛУ, так и для сдвига или выравнивания данных. Однако, как показала практика, он вносит дополнительные задержки на интенсивно используемом пути между АЛУ и регистровым блоком, что приводит к увеличению общей длительности такта машины. Поэтому сдвигатель на произвольное число бит не должен входить в критический путь тракта обработки данных – его можно разместить в каком-то другом месте, с возможностью доступа только по командам, допускающим меньшую скорость выполнения. Кроме того, сдвиги на один или два бита, составляющие подавляющее большинство всех сдвигов, можно производить в АЛУ.

Приведенные рассуждения позволяют понять, в чем заключается особенность архитектуры RISC. Те или иные средства, включаемые в схему МП с целью ускорения выполнения определенных операций, могут вызывать замедление других операций. В частности, слишком большая или слишком сложная схема микропрограммного ядра процессора будет снижать его собственное быстродействие. Таким образом, в схему следует вводить только

такие средства, которые способствуют ускорению выполнения часто используемых операций.

8.5.3. Секция управления микропроцессора RISC II

Одно преимущество сокращенного набора команд – кардинальное уменьшение объемов кремниевых ресурсов, требуемых для реализации функций управления. В микропроцессоре RISC II дешифратор кода операции занимает 0,5% площади кристалла, содержит лишь 0,7% общего числа транзисторов, а трудоемкость его проектирования с учетом разработки топологии составила менее 2% общих трудозатрат.

Дешифратор кода операции выполняет функции, для реализации которых в микропрограммируемых МП используется память микрокода (см. раздел 5.3), занимающая значительную часть площади кристалла. В настоящем разделе описана секция управления микропроцессора RISC II и объяснено, каким образом достигнуты ее малые размеры.

На рис. 8.6 показаны основные компоненты секции управления МП RISC II, которые занимают всего 10% общей площади кристалла и содержат 6% общего числа транзисторов¹⁴. Фиксированный формат команд микропроцессора RISC II с небольшим количеством ортогональных полей дает возможность разбить биты команды на три группы – код операции, номера регистров и непосредственная константа – и при их поступлении на кристалл МП принимать отдельно на регистры-защелки, расположенные вблизи тех мест, где эти поля должны использоваться. Номера регистров и непосредственная константа проходят по *конвейеру* регистров-защелок («станций») и используются в соответствующем месте в соответствующее время. Декодирование семибитового кода операции вместе с однобитовым признаком состояния дает 30 бит информации расширенного кода операции. Коды операций декодируются по одному за такт. *Имеется только один бит состояния*, и служит он для того, чтобы различать нормальный внутрипроцессорный цикл и цикл обращения к памяти.

¹⁴ Имеется ввиду технология, по которой был изготовлен МП RISC II.

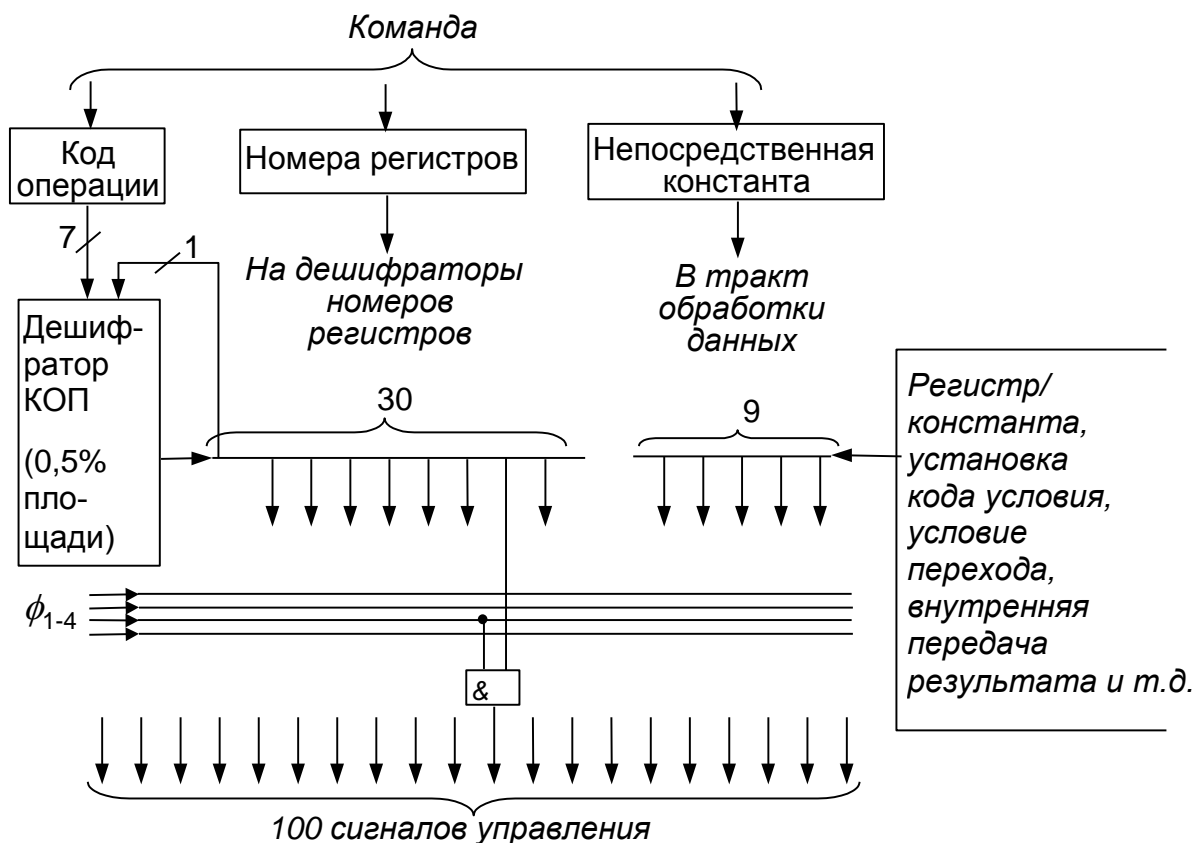


Рис. 8.6. Секция управления микропроцессора RISC II

Кроме 30 бит расширенного кода операции в управлении ЦП участвуют еще 9 бит. Они содержат информацию, ортогональную по отношению к коду операции, в частности, определяют *выбор регистра или константы* как операнда-2, *признак* установки кодов условий (УКУ), *результат анализа условия* перехода или служат для обнаружения взаимозависимости по данным с целью организации внутренней передачи промежуточных результатов с уровня на уровень конвейера.

Сто сигналов управления, необходимых для процессора в целом, вырабатываются путем пропускания через логические вентили И одного или более из упомянутых выше 30+9 бит с одним или более из четырех фазовых синхросигналов ϕ_{1-4} .

Чтобы понять, откуда берутся четыре фазы синхронизации, рассмотрим, в какой последовательности передаются данные по двум наиболее часто используемым направлениям *шинаB* → блок SRC → *шинаR* и *константа* → *шинаL* → *шинаR*.

	<i>шинаB</i> → блок SRC → <i>шинаR</i>	<i>константа</i> → <i>шинаL</i> → <i>шинаR</i>
Такт 1:	<i>регистр</i> → <i>шинаB</i>	<i>данные/конст.</i> → <i>рег ВходДанных/констант</i>

Такт 2:	<i>шинаB</i> → <i>блокSRC</i>	<i>регВходДанных/констант</i> → <i>шинаL</i> → <i>шинаR</i>
Такт 3:	<i>результат</i> → <i>блокDST</i>	1. <i>результат</i> → <i>блокDST</i> *) 2. <i>результат</i> → <i>шинаOUT</i> и <i>IP</i>
Такт 4:	<i>блокDST</i> → <i>регистр</i>	1. <i>блокDST</i> → <i>регистр</i> 2. <i>выборка данных/команды из памяти</i>

*) Здесь пп. 1 и 2 обозначают варианты операций на 3-ем и 4-ом тактах.

По первому пути (*шинаB*→*блокSRC*→*шинаR*) информация перемещается при выполнении операций АЛУ. На первом такте данные из адресованного регистра поступают на шину *B*, на втором – в блок *SRC*, на третьем результат операции принимается в блок *DST* и на четвертом сохраняется в регистре приемнике. Если полученный результат используется при исполнении следующей команды, он сохраняется в блоке *DST* для передачи на следующий уровень конвейера (рис. 8.4).

Второй путь (*константа*→*шинаL*→*шинаR*) применим

- 1) в операциях АЛУ, когда одним из операндов является константа, записанная в регистр *регВходДанных/констант* при выборке инструкции (рис. 8.5). Результат операции поступает в блок *DST* и при отсутствии необходимости его передачи на следующий уровень конвейера передается в регистр приемник (в регистровый блок) на последней стадии,
- 2) при загрузке регистра из памяти. В этом случае данные из регистра входных данных/констант загружаются в адресованный регистр регистрового блока,
- 3) при вычислении исполнительного адреса данных или адреса инструкции. При адресации данных результат АЛУ выдается на выходную шину *OUT*. При адресации инструкции – сохраняется в счетчике команд как адрес *задержанного перехода* (рис. 8.5), для его реализации (посредством передачи значения *IP* на шину *OUT*) на соответствующем уровне конвейера.

Для управления выполнением команды достаточно нескольких бит информации. Это существенно меньше, чем число бит микрокода, требуемых для выполнения команд в типичных микропрограммируемых машинах. Такое малое число достигнуто благодаря стандартному базовому порядку выполнения всех команд RISC машины: чтение того или иного операнда, выполнение указанной операции и сохранение результата в соответствующем месте. Все это происходит по одной и той же фиксированной временной диаграмме, что дает возможность аппаратно завести необходимую временную информацию в логические вентили, с помощью которых вырабатываются сигналы управления.

По своей структуре и принципу действия управляющий автомат МП RISC II близок к автомату с жесткой логикой (рис. 8.7), который имеет в своем составе дешифратор команд, дешифратор тактов и логику формирования управляющих (функциональных) сигналов. Тактовые импульсы от

синхронизирующего генератора поступают на счетчик. Состояния счетчика представляют собой номера тактов, изменяющихся от 1 до n .

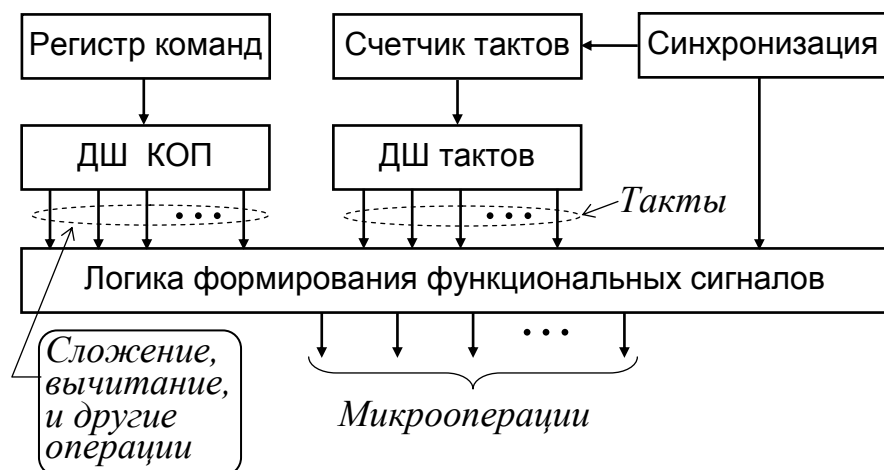


Рис. 8.7. Автомат с жесткой логикой.

Дешифратор тактов формирует на i -м выходе единичный сигнал при i -м состоянии счетчика тактов, т.е. во время i -го такта. На выходах дешифратора тактов получается необходимое для работы автомата число импульсных последовательностей.

Дешифратор кода операции вырабатывает единичный сигнал на j -м выходе, если выполняется j -я команда.

Логические схемы образования управляющих функциональных сигналов для каждой команды возбуждают формирователи функциональных сигналов для выполнения требуемых в данном такте микроопераций.

Принцип построения логических схем образования управляющих сигналов поясняется на рис. 8.8, где показан фрагмент схемы, обеспечивающей выработку управляющего сигнала U_k в i -м и n -м тактах выполнения j -й команды.

В общем случае значения управляющих сигналов зависят еще и от оповещающих сигналов, отражающих ход вычислительного процесса. Для реализации этих зависимостей элементы, представленные на рис. 8.8, берутся многоходовыми и на них заводятся требуемые сигналы логических условий.

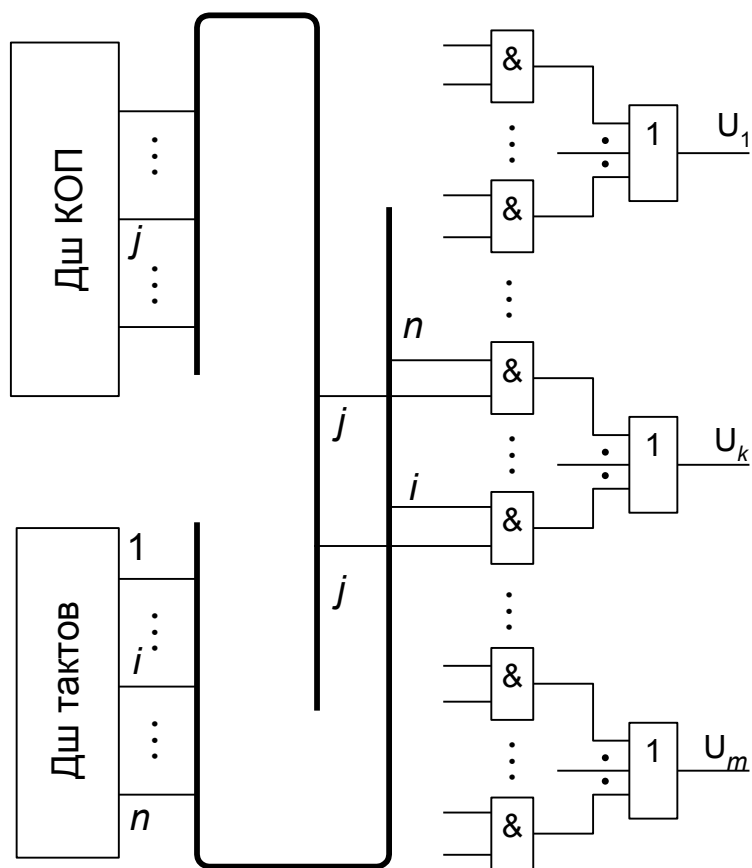


Рис. 8.8. Формирование функциональных сигналов в автомате с жесткой логикой

Серьезным недостатком рассмотренных схем является одинаковое число тактов для всех команд. Это требует выравнивания числа тактов исполнения команд по наиболее «длинной» команде, что ведет к непроизводительным затратам времени. Чтобы устранить этот недостаток, схемы строят с использованием нескольких счетчиков тактов.

9. Микропроцессорные системы с архитектурой фон-Неймана

Приступая к обсуждению микропроцессорных систем, определимся с предметом обсуждения, несмотря на всю его кажущуюся очевидность. Разнообразие МП систем столь велико, что их исследование невозможно без определенных и заранее сформулированных ограничений, выраженных либо в форме модельных представлений, либо в виде разбиения систем на классы. Важными для микропроцессорных систем являются не только способность работать по хранимой в памяти программе, но и способность реагировать на внешние воздействия. Последнее позволяет корректировать работу системы, согласуя ее с поступающей извне (например, от пользователя или от физических датчиков) информацией. Для реализации такой возможности МП система уже в своей минимальной конфигурации помимо центрального звена – процессора должна иметь память и устройства ввода-вывода (УВВ).

Как уже отмечалось (см. раздел 8.4), процессор в МП системе работает более эффективно, если программный код и коды данных размещены в памяти так, что есть возможность независимого к ним обращения. По отношению к памяти различают фоннеймановскую и гарвардскую архитектуры процессоров.

Архитектура фон Неймана, предложенная этим ученым в 1940-х годах для реализации первых моделей цифровых ЭВМ, сыграла большую роль в развитии цифровых вычислительных систем. Она наиболее проста, т. к. программа и данные физически располагаются в одной и той же памяти. В результате за один цикл обращения процессор может получить доступ либо к программе, либо к данным.

9.1. Магистрально-модульная структура микропроцессорной системы

Традиционно микропроцессорные системы, соответствующие архитектуре фон Неймана, имеют магистрально-модульную структуру (рис. 9.1), в которой отдельные устройства (модули), входящие в состав системы, обмениваются информацией по общей системной шине – системной магистрали.

Основным модулем системы является процессор (или микропроцессор). Оперативное запоминающее устройство (ОЗУ) служит для хранения выполняемой программы (или ее фрагментов) и данных, подлежащих обработке. Постоянное запоминающее устройство (ПЗУ) служит для хранения констант и стандартных (неизменных) программ. В ПЗУ обычно записываются программы начальной инициализации (загрузки) системы, тестовые и диагностические программы и другое служебное программное обеспечение. В МП системах, управляющих определенными объектами с использованием

фиксированных и редко изменяемых программ, используются также репрограммируемые ПЗУ.

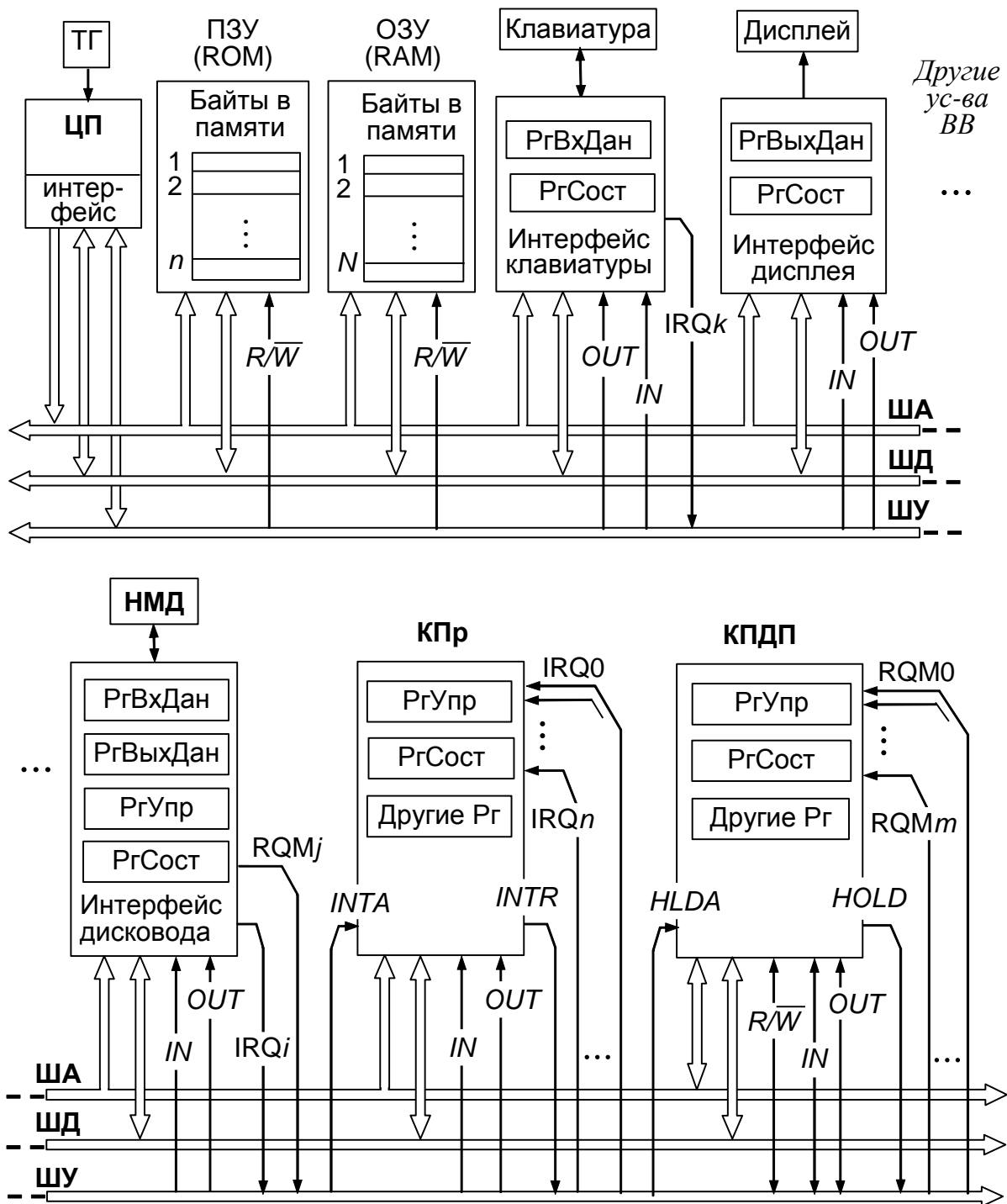


Рис. 9.1. Структура МП системы с трехшинным магистральным интерфейсом

Остальные устройства являются внешними и подключаются к системе с помощью интерфейсных устройств (ИУ), реализующих протоколы параллельного или последовательного обмена данными. Такими внешними устройствами могут быть клавиатура, монитор (дисплей), внешние запоминающие устройства с накопителями на жестких и гибких магнитных дисках (НМД), оптических дисках, магнитных лентах и других носителях

информации, датчики и преобразователи информации (аналого-цифровые и цифро-аналоговые), разнообразные исполнительные устройства (индикаторы, принтеры, электродвигатели, реле и другие). Для реализации различных режимов работы к системе могут подключаться дополнительные устройства – контроллеры прерываний (КПр), прямого доступа к памяти (КПДП) и другие, выполняющие специальные функции управления.

Системная шина содержит несколько десятков (в сложных системах более 100) проводников, которые в соответствии с их функциональным предназначением подразделяются на отдельные шины – адреса *ША*, данных *ШД* и управления *ШУ*. Шина *ША* служит для передачи адреса, который формируется процессором и позволяет выбрать необходимую ячейку памяти или требуемое ИУ при обращении к устройствам ввода-вывода. Шина *ШД* служит для выборки команд, поступающих из ОЗУ или ПЗУ в процессор, и для пересылки обрабатываемых данных (операндов) между процессором и ОЗУ или ИУ. По шине *ШУ* передаются управляющие сигналы, определяющие режимы работы памяти (запись или считывание – R/\bar{W}), интерфейсных устройств (ввод или вывод – *IN*, *OUT*), запросы внешних устройств на обслуживание и другие сигналы.

Интерфейсы внешних устройств (клавиатуры, дисплея, дисков и др.) разнообразны по своему предназначению, но со стороны системной магистрали видятся как совокупность регистров, или *портов ввода-вывода* (ВВ). В составе интерфейса УВВ в общем случае имеются регистр(ы) данных (входных – *РгВхДан*, выходных – *РгВыхДан* или тех и других), регистр(ы) состояния (*РгСост*) и регистр(ы) управления (*РгУпр*).

Регистры данных содержат получаемые от УВВ данные, и в них помещается выводимая в УВВ информация. При этом интерфейс внешнего устройства предоставляет процессору информацию о своем состоянии с целью обеспечить положительный результат обмена данными. Такая информация заключена в *регистре (регистрах) состояния*. В простейшем случае *РгСост* показывает готовность ИУ к операции ввода-вывода. Для этого бывает достаточно двух бит – один для операций ввода, другой для операций вывода.

В регистре (или регистрах) управления размещается информация, необходимая для контроля и управления внешним устройством. Эта информация столь же разнообразна, насколько разнообразны сами внешние устройства. Например, дисководы требуют указания направления передачи данных, приведения в действие вращающегося диска механизма, позиционирования головки считывания/записи, определения считываемого или записываемого сектора. В принтерах нужно контролировать перемещение бумаги и состояние печатающего механизма и т. д.

Общее число регистров (портов) в ИУ определяется спецификой УВВ. Обращение к портам ВВ, как и к памяти, происходит по указанному в командах ввода-вывода адресу и в сопровождении соответствующего сигнала синхронизации (на рис. 9.2 *IN* – для ввода и *OUT* – для вывода). Однако в отличие от памяти время доступа к портам ВВ различных устройств изменяется

в широких пределах, что необходимо учитывать при реализации циклов ввода-вывода. В этом заключена одна из причин выделения портам ВВ отдельного адресного пространства – *пространства ввода-вывода*. Схемотехнически решается это тем, что в шину ШУ системной магистрали наряду с линиями сигналов чтения/записи R/\bar{W} включаются линии сигналов IN/OUT (рис. 9.1), используемые для синхронизации обращений к портам ввода-вывода.

В том случае, когда архитектура системы не предполагает наличия отдельного адресного пространства для портов ВВ, им выделяется фиксированная область из общего пространства адресов и обращение к портам происходит под управлением тех же сигналов R/\bar{W} , что и для памяти. В таком случае говорят о портах ВВ, как о портах, *отображенных на память*.

Обмен данными с устройствами ввода-вывода имеет три разновидности: 1) программный обмен с проверкой готовности по значению регистра состояния, 2) обмен по запросу прерывания IRQ (*Interrupt Request*) и 3) обмен в режиме прямого доступа к памяти по запросу RQM (*Request Memory*). Запросы на обслуживание от УВВ принимаются соответствующими контроллерами прерываний (КПр) и прямого доступа к памяти (КПДП) (см также разделы 7.4 и 7.5).

Контроллер прерываний принимает запросы от УВВ по соответствующим линиям, входящим в состав ШУ (линии $IRQ_0...IRQ_n$). На рис. 9.1 одна из этих линий (линия IRQ_k) связана с клавиатурой. Запрос с наивысшим на текущий момент времени приоритетом преобразуется в сигнал $INTR$, направляемый процессору. Если прерывания разрешены, на что показывает флаг IF разрешения прерываний в регистре (или в одном из регистров) состояния процессора, то в ответ на $INTR$ процессор направляет контроллеру подтверждение $INTA$ и получает от КПр идентификатор устройства, запросившего обслуживание. Информация об этом устройстве передается по ШД и используется процессором для перехода на процедуру (программу) обслуживания запроса. Функции, которые КПр будет выполнять в процессе своей работы (например, приоритетное обслуживание), программируются с помощью управляющих слов путем вывода их в соответствующие регистры управления контроллера. Состояние контроллера и характер выполняемых им функций контролируются также путем обращения к регистрам (портам) КПр.

Контроллер прямого доступа к памяти (ПДП) принимает запросы от обслуживаемых им внешних устройств, чтобы разместить поступающие от УВВ данные в ОЗУ или данные из ОЗУ передать в УВВ без участия процессора. Однако свою функцию КПДП может реализовать только

1. после занесения в соответствующие порты КПДП информации о том, в какую область (или из какой области) памяти и в каком количестве данные должны быть переданы из УВВ (или в УВВ);
2. после получения от процессора разрешения выполнять функцию ПДП;
3. после получения запроса от УВВ, которое предварительно также было запрограммировано на работу в режиме прямого доступа к памяти путем

установки соответствующих бит в регистре управления, один из которых определяет направление передачи данных, а другой разрешает работу в режиме ПДП.

При обработке запросов ПДП, как и при обработке запросов прерывания, действует система приоритетов. Запросы ПДП поступают по линиям $RQM_0...RQM_m$ и далее с учетом приоритета перенаправляются процессору в виде сигнала запроса магистрали *HOLD*. На рис. 9.1 устройством, для обмена данными с которым используется режим ПДП, является накопитель на магнитных дисках НМД с назначенной ему запросной линией RQM_j .

Получив запрос ПДП, процессор подтверждает его сигналом разрешения ПДП *HLDA*, давая тем самым КПДП возможность выполнить работу по обслуживанию запроса. Процессор отключается от шин адреса, данных и от необходимых для работы КПДП линий *ШУ*. После этого сигналы на адресных линиях, сигналы сопровождения R/\overline{W} для памяти и сигналы *IN* или *OUT* для ввода или вывода из регистра или в регистр данных интерфейса обслуживаемого устройства начинает генерировать контроллер ПДП. По этой причине *ША* и линии R/\overline{W} , *IN* и *OUT* со стороны КПДП являются двунаправленными. Когда КПДП закончит работу, управление системной магистралью вновь переходит к процессору. При этом контроллер ПДП запросом прерывания (запрос IRQ_i на рис. 9.1) инициирует запуск процедуры завершения цикла прямого доступа к памяти с целью проверки корректности выполненных операций.

9.2. Системы с магистральным интерфейсом

Магистральный интерфейс находит применение при построении широкого класса микропроцессорных систем, имеющих отношение не только к вычислительным устройствам. На его основе могут быть построены измерительно-вычислительные и управляющие системы, начиная от простейших встраиваемых в радиоэлектронные устройства и в иное техническое оборудование систем, до сложных автоматизированных комплексов, применяемых для сбора и обработки информации, в научных исследованиях, при управлении объектами и технологическими процессами и т. п. Технические решения магистрального интерфейса также разнообразны и зависят от конкретного применения. В одном случае определяющим фактором являются аппаратные затраты. В другом – быстродействие, пропускная способность и способность обеспечить работу в реальном масштабе времени.

Для встраиваемых в техническое оборудование (например, в измерительные приборы, в приемопередающую аппаратуру) систем, предназначением которых является управление интегрированными в состав этого оборудования компонентами, применима простая двухпроводная линия. Одним из стандартных интерфейсов подобного типа является разработанный

фирмой Philips интерфейс I²C (*Inter-Integrated Circuit*) для внутрисистемного обмена, т.е. для обмена между образующими систему интегральными схемами. Адресная информация и данные по двухпроводной линии I²C передаются последовательным кодом в пакетном режиме. Каждый передаваемый информационный пакет содержит заголовок и присоединенную к заголовку битовую последовательность. По заголовку определяется принадлежность и характер передаваемой информации.

Магистральный интерфейс в структуре персонального компьютера

Скорость передачи информации повышается при использовании параллельного интерфейса, подобного тому, который представлен на рис 9.1. Параллельный магистральный интерфейс является основой для построения персональных компьютеров. В компьютерах с процессорами семейства x86/88 фирмы Intel в свое время как системная использовалась трехшинная магистраль ISA (*Industry Standard Architecture*) с 24-разрядной шиной адреса, 16-разрядной шиной данных и шиной управления. Затем шина ISA стала использоваться преимущественно как шина ввода-вывода, а для взаимодействия процессора с системой стала применяться более быстродействующая *локальная* (или *системная*, как показано на рис. 9.2) шина.

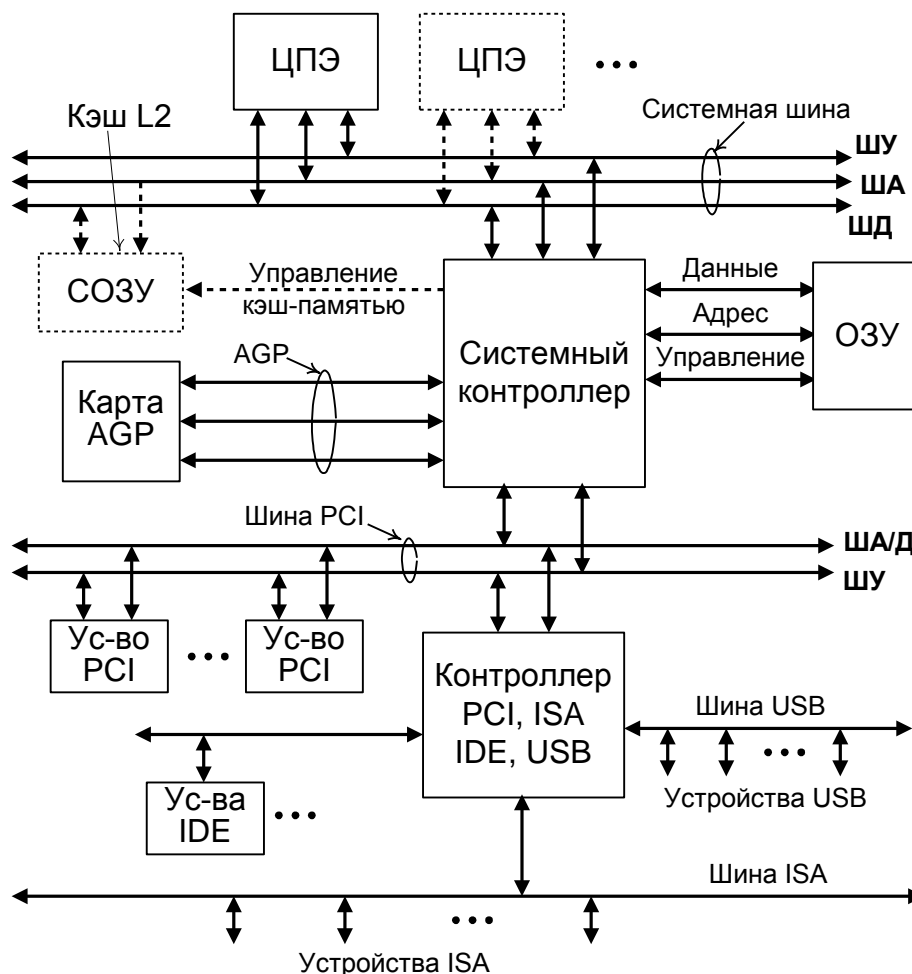


Рис. 9.2. Принцип функционирования микросистемы персонального компьютера

Непосредственно с центральным процессорным элементом ЦПЭ по локальной шине связана кэш-память второго L2 уровня (кэш-память первого L1 уровня расположена на кристалле процессора), выполненная на статическом запоминающем устройстве СОЗУ. Та же локальная шина служит для связи процессора с оперативной памятью (ОЗУ), но через системный контроллер, на который возлагается также функция управления кэш-памятью второго уровня и функция связи с видеопамью (как частью основной памяти) компьютера при участии ускоренного графического интерфейса AGP (*Accelerated Graphics Port*).

Для подключения к системе внешних устройств стала использоваться шина ввода-вывода PCI (*Peripheral Component Interconnect*) с большей, чем у шины ISA производительностью. Шина PCI имеет совмещенную шину адреса/данных. Существуют две ее модификации – 32- и 64-разрядная. Связь системной шины с шиной PCI осуществляет системный контроллер.

Выход на подсистему ввода-вывода, не поддерживающую стандарт PCI, реализует дополнительный контроллер, конфигурация которого зависит от того, каким набором периферийных устройств подсистема ввода-вывода обладает. На рис. 9.2 представлена конфигурация, которая дает возможность подключения к системе

- накопителей на магнитных дисках и других устройств внешней памяти с параллельным интерфейсом – через параллельную шину IDE (*Integrated Drive Electronics*),
- периферийных устройств с последовательным способом передачи информации – через шины USB (*Universal Serial Bus*) и
- устройств, выполненных в стандарте шины ISA.

Развитие в сторону *многопроцессорных* систем привело к тому, что кэш-память второго уровня вместе с процессором стала размещаться в отдельном процессорном блоке со щелевым разъемом для подключения к системной шине. Это позволило к одной системной шине подключать несколько процессоров с принадлежащей им кэш-памятью. На рис. 9.2 показаны два процессора из числа тех, которые могут присутствовать в многопроцессорной системе. Один из процессоров обозначен пунктирными линиями, т.к. его может не быть в системе с одним процессором. Пунктирными линиями обозначена и перемещенная в процессорный блок кэш-память. При таком расположении кэш-памяти отпадает необходимость в управлении ею со стороны системного контроллера, поскольку функция управления кэш-памятью передается в процессорный блок.

По мере роста уровня интеграции стало возможным размещать большие объемы кэш-памяти на кристалле микропроцессора, что повлияло на место расположения кэш-памяти и на техническое исполнение интерфейса ЦПЭ с системной магистралью, но не затронуло структуру и состав его

функциональных компонент. Кроме того, наряду с многопроцессорными системами все шире стали применяться системы с многоядерными процессорами, в которых идеи параллелизма реализуются за счет параллельного исполнения команд, поступающих в процессор из памяти. Поскольку для написания программ преимущественно используются языки высокого уровня, то большое значение в этом случае приобретает то, какой программный код для процессора будет сгенерирован соответствующим транслятором.

Многомодульные системы с магистральным интерфейсом

При построении систем обработки-управления и контрольно-измерительных систем приходится объединять в одном комплексе многочисленные модули различного предназначения. Типовые решения в этом направлении базируются на использовании магистрального интерфейса. В зависимости от степени разветвленности и функциональных возможностей системы модули могут размещаться как в автономно работающих ячеечного типа конструкциях – крейтах (шасси) с встроенными источниками питания и системой кондиционирования, так и быть распределенными по нескольким базовым блокам (*mainframes*).

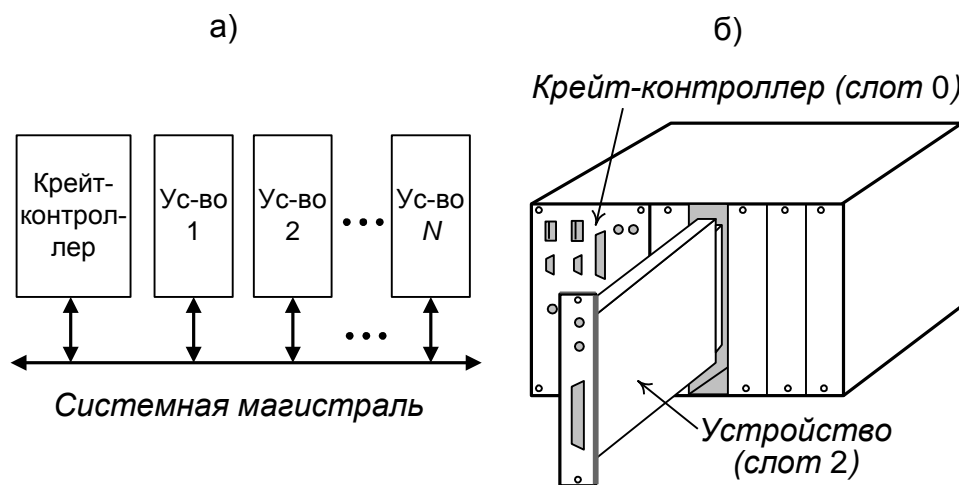


Рис. 9.3. Однокрейтовая система со слотами расширения для подключения периферийных устройств

На рис. 9.3 представлена система, конструктивно выполненная в одном шасси, где все устройства подключены к системной магистрали (рис. 9.3а) через соединители, расположенные на задней панели шасси. Крейт-контроллер выполняет функцию управляющей ЭВМ. Стандартные устройства ввода-вывода (клавиатура, монитор, печатающее устройство, устройства в стандарте USB, сетевой интерфейс и т.д.) подключаются через соединители, расположенные на передней панели крейт-контроллера. Со стороны передней панели расположены также регулировки и разъемы, принадлежащие

периферийным устройствам ввода-вывода и предназначенные для подключения связанных с ними объектов.

Для микропроцессорных систем, обладающих подобной структурой, широкое распространение получила шина VME (*Versa Module Eurucard bus*), появившаяся в 1981 г. Успех шины VME определяет то обстоятельство, что заложенные в ее архитектуру идеи не только не претерпели никаких изменений, но при этом несколько не устарели. Шина имеет разрядность адресов и данных до 32 бит. Конструктивно платы VME соответствуют стандарту Eurocard на платы одинарной (160×100 мм) и двойной (160×234 мм) высоты. Наличие двух типоразмеров плат позволяет создавать как малые, экономичные по габаритам и стоимости, так и высокопроизводительные системы. Для подключения плат к объединительной панели используются 96-контактные соединители, имеющие высокую надежность.

Стандарт VMEbus получил распространение также в области измерительных и управляющих систем реального времени. Его расширением для контрольно-измерительных задач повышенной метрологической обеспеченности является интерфейс VXI (*VMEbus eXtention for Instrumentation*), предложенный в 1987 г. ведущими в области измерительной техники фирмами. Интерфейс VXI регламентирует правила объединения модульных и одноплатных измерительных приборов и считается наиболее перспективным для контрольно-измерительной техники.

Интерфейс VXI разработан на базе двух широко используемых стандартов: VMEbus и приборного интерфейса IEEE-488 (HPIB – *Hewlett-Packard Interface Bus*, переименованный позднее в GPIB – *General Purpose Interface Bus*)¹⁵ и объединил в себе преимущества магистрально модульных и приборных измерительных систем. Такое объединение позволило создать высокопроизводительный и высокоточный интерфейс, обеспечивающий широкие возможности для построения контрольно-измерительных систем различного назначения.

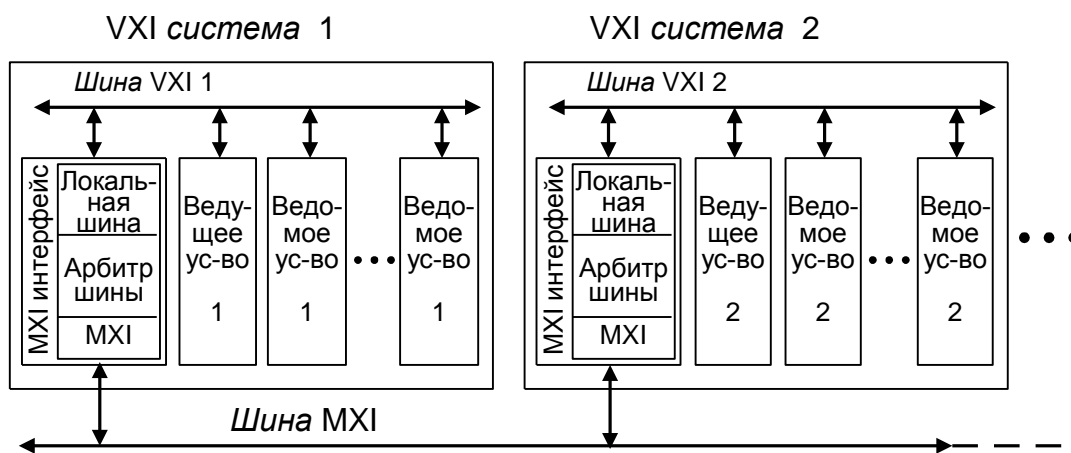


Рис. 9.4. Структура многомодульной системы на базе шины MXI

¹⁵ Отечественный аналог интерфейса GPIB называется каналом общего пользования (КОП).

Основу шины VXI составляет интерфейс межмодульного обмена информацией в рамках одного крейта (см. рис. 9.3), включающий шину VMEbus и шины расширения, среди которых имеются

- локальная шина, которая может использоваться разработчиками модулей для организации обмена данными между соседними модулями по автономным (недоступным для других модулей) линиям, и
- аналоговая шина для обмена аналоговыми сигналами.

Для построения однокрейтовых, подобных изображенной на рис. 9.3, систем фирмой National Instruments была предложена шина PXI (*PCibus eXtention for Instrumentation*), которая является расширением шины PCI для контрольно-измерительных систем. В основу расширения легли те же идеи, которые определили разработку шины VXI, но с отличиями, которые обусловлены, прежде всего, отличием шины PCI от шины VME, а также составом дополнительных включенных в PXI линий и конструктивным оформлением соединителей.

Существование VME- и VXI-систем сделало вполне естественным развитие контрольно-измерительных комплексов в сторону создания систем, объединяющих возможности не одного, а нескольких базовых блоков – блоков, построенных с использованием отдельных шасси. Работа в этом направлении привела к разработке интерфейса MXI (*Multisystem eXtention Interface*, фирма National Instruments), который позволил создавать системы, состоящие из нескольких базовых VXI блоков, объединенных кабельной магистралью. На рис. 9.4 представлено структурное построение такой системы.

Входящие в состав MXI системы базовые VXI блоки состоят из несколько VXI устройств, одно из которых в каждый текущий момент времени является ведущим, а остальные могут быть только ведомыми. Обычно ведущим является крейт-контроллер, если он не передает управление шиной какому-либо другому устройству, например, устройству, способному обмениваться данными в режиме прямого доступа к памяти.

Для связи с MXI магистралью в каждой VXI системе имеется MXI интерфейс, который выполняет функцию сопряжения линий шины VXI с линиями шины MXI, а также функцию арбитра, необходимую всегда, когда ресурсы системной магистрали (в данном случае ресурсы шины MXI) разделяются несколькими активными устройствами, каковыми являются VXI системы. Арбитр шины в каждый текущий момент времени предоставляет MXI магистраль в распоряжение только одному VXI блоку. Этим исключаются возможные конфликтные ситуации, когда более чем одно устройство пытается передавать данные по шине. Шинный арбитраж осуществляется на основе установленной системы приоритетов и порядка их изменения при работе.

На рис. 9.5 представлены два варианта использования MXI магистрали для построения многомодульных систем. Систему можно построить, включив в нее компьютер со всеми принадлежащими ему стандартными устройствами ввода-вывода и один или несколько VXI блоков (рис. 9.5а). Для этого

компьютер должен иметь интерфейс с шиной МХІ. Это может быть плата, подключаемая к шине ввода-вывода, например, к шине РСІ. Блок VХІ (*VXI mainframe*) подключается к МХІ шине через свой интерфейс, имеющийся в составе крейт-контроллера или выполненный как отдельное связанное с крейт-контроллером устройство. Система может быть расширена за счет подключения к ней других базовых VХІ блоков с помощью системы кабелей с соединительными разъемами.

МХІ система может быть получена также путем объединения нескольких VХІ блоков (рис. 9.5б). При этом один из базовых блоков должен иметь стандартные устройства ввода-вывода для организации интерфейса пользователя. Все эти устройства подключаются к крейт-контроллеру через имеющиеся на лицевой панели соединители.

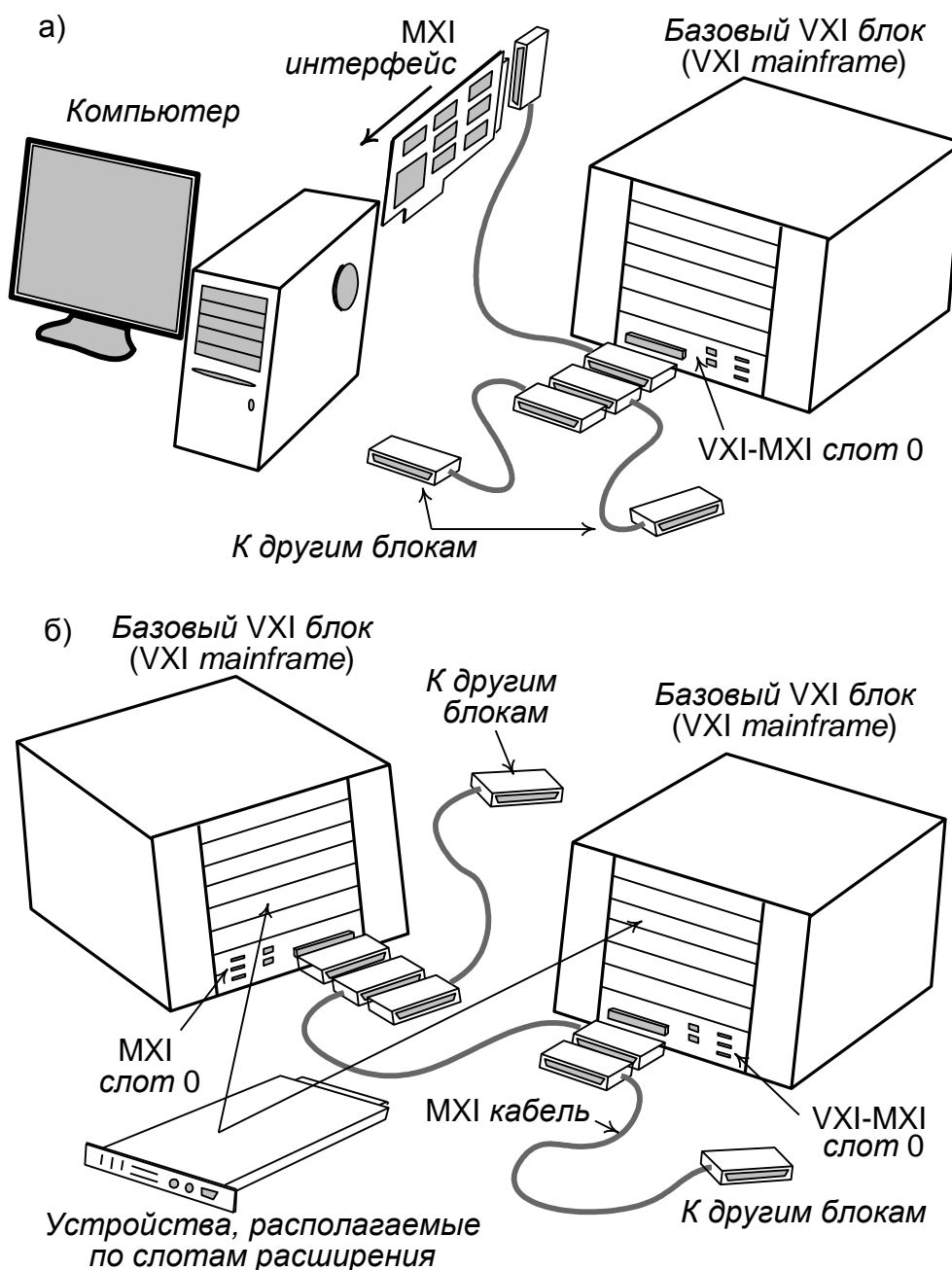


Рис. 9.5. Варианты использования МХІ магистрали

9.3. Магистральный интерфейс в микроконтроллерах

Микроконтроллеры (МК) широко используются в современной радиоэлектронике в качестве встроенного средства контроля, цифровой обработки и управления характеристиками различных объектов или процессов. Возрастающая степень интеграции цифровых микросхем определила появление промышленных МК, реализованных на одном кристалле, в который кроме центрального процессорного элемента (ЦПЭ) интегрирован набор компонентов, состав и предназначение которых определяется конкретным приложением. К числу таких компонентов могут относиться устройства постоянной и перепрограммируемой памяти, последовательные и параллельные порты разного назначения, средства поддержки цифровых каналов связи, аналого-цифровые (АЦП) и цифро-аналоговые (ЦАП) преобразователи и др.

Как пример системы, построенной на основе магистрального интерфейса, все компоненты которой интегрированы в один кристалл, рассмотрим микроконтроллер MSP430F1611 фирмы Texas Instruments.

Программное обеспечение для МК обычно разрабатывается на инструментальной ЭВМ с использованием интегрированной среды разработки (IDE), позволяющей генерировать двоичные коды команд целевого микроконтроллера. Результатом программирования в среде IDE является двоичный код программы, который через соединение с каким-либо портом ЭВМ (например, через USB порт) заносится в программируемую память (flash-память) микроконтроллера. После завершения процесса программирования МК способен функционировать в автономном режиме, следуя загруженной в его память программе.

Микроконтроллеры семейства MSP430 построены как машина фон-Неймана (см. рис. 9.6, на котором приведена структурная схема микроконтроллера MSP430F1611) с общей внутри кристалла системной магистралью, образованной 16-разрядной шиной адреса ША (MAB – *Memory Address Bus*), 16-разрядной шиной данных ШД (MDB – *Memory Data Bus*) и шиной управления ШУ (MCB – *Memory Control Bus*). Адреса портов ввода/вывода отображены на память и занимают выделенное для них 4-битовое адресное пространство. Часть шины данных также имеет пониженную 8-битовую разрядность, соединенную с основной ее частью с помощью моста 16/8.

В состав МК входят:

1) ЦПЭ с RISC архитектурой и с 16-ю 16-разрядными регистрами, включая счетчик команд (PC), указатель стека (SP), регистр состояния (SR) и генератор констант (CG). Генератор констант служит для получения наиболее часто используемых констант (-1, 0, 1, 2, 4, 8) и позволяет, в дополнение к 27 основным командам, эмулировать еще 24 дополнительные инструкции.

Последние не относятся к исполняемым ядром МК командам и введены лишь для облегчения программирования. Компилятор автоматически в процессе трансляции исходного ассемблерного кода заменяет эмулирующие инструкции на эквивалентные им реально исполняемые инструкции;

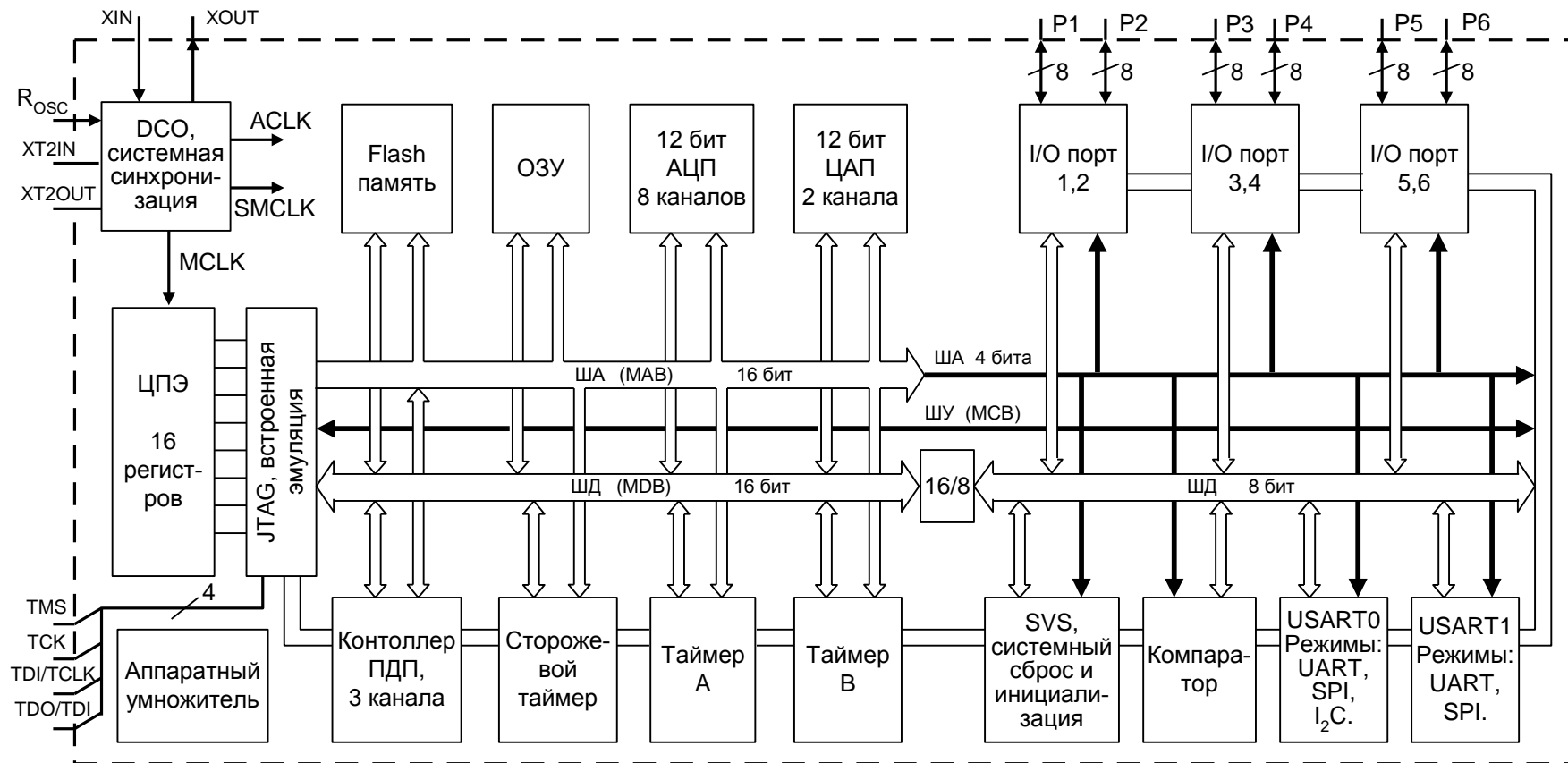


Рис. 9.6 Структура микроконтроллера MSP430F1611

2) *Встроенная flash-память* на 48 Кбайта используется для хранения программ. Эта память может быть записана и стерта только при помощи программатора, а в процессе работы программы к ней возможно обращение только для чтения информации. Начальный адрес flash-памяти зависит от объема адресного пространства, конечный адрес flash-памяти всегда 0FFFh. Flash-память может использоваться для хранения как программного кода, так и данных;

3) *Оперативная память (ОЗУ)*. Микроконтроллер содержит 10 Кбайт оперативной памяти.

Периферийные модули (порты ввода/вывода) отображены в адресном пространстве памяти. Адреса с 0100 до 01FFh зарезервированы для 16-разрядных периферийных модулей. Они доступны с помощью команд-слов. Адресное пространство с 010h по 0FFh зарезервировано для 8-разрядных периферийных модулей. Эти модули доступны с помощью однобайтных команд;

4) *12-разрядный аналогово-цифровой преобразователь*. Имеет 8 внешних входов для аналоговых сигналов, буфер на 16 слов для хранения результатов преобразования, организованный как очередь с автоматическим заполнением и сохранением ее содержимого в основной памяти. Скорость преобразования – до 200 выборок в секунду. Источник опорного напряжения встроен в схему АЦП. Кроме того, в него интегрированы датчик температуры и цепь обнаружения разряда источника питания, позволяющая контролировать состояние источника (батареи) при автономной работе;

5) *Двухканальный 12-разрядный цифро-аналоговый преобразователь (ЦАП)*;

6) *Аналоговый компаратор* – без гистерезиса со встроенным программируемым делителем для установки напряжения компарирования (опорного напряжения). Компаратор имеет несколько входов, переключаемых с помощью мультиплексора, и программно отключаемый RC-фильтр на выходе. С компаратором ассоциирован вектор прерываний, что позволяет сигнал компарирования обрабатывать в режиме прерывания;

7) *Контроллер прямого доступа к памяти (КПДП)*, имеющий три канала, каждый из которых может использоваться для перемещения данных из устройства (или в устройство) ввода/вывода в оперативную (или из оперативной) память(и) независимо от ЦПЭ;

8) *16-разрядный аппаратный умножитель*. Аппаратный умножитель не интегрирован в ЦПЭ и является расширением базовой архитектуры микроконтроллера, работая независимо от ЦПЭ. Умножитель имеет два 16-разрядных регистра для операндов и три регистра для сохранения результата умножения и приступает к работе сразу после загрузки регистров операндов. Результат автоматически сохраняется в регистрах результата. Поэтому для умножителя не требуется специальных инструкций и дополнительных циклов синхронизации. Операции выполняются при любых комбинациях 8- и 16-разрядных операндов. Поддерживаются четыре типа операций: знаковое и

беззнаковое умножение (MPYS и MPY), знаковое и беззнаковое умножение с накоплением (MACS и MAC);

9) *Супервизор напряжения источника питания с блоком системного сброса и инициализации* контролирует состояние источника питания и генерирует сигнал системного сброса при снижении напряжения питания ниже установленного порогового значения. Используется также при включении МК в работу, т.е. при подаче на него питания. После установки напряжения питания до рабочего уровня генерирует сигнал запроса прерывания, запускающего процедуру инициализации;

10) *2 универсальных 16-битных таймера (Timer A и Timer B) с режимами «захват/сравнение» и возможностью формирования сигнала широтно-импульсной модуляции (ШИМ)*. Выполнены в виде двоичных счетчиков, подсчитывающих импульсы, получаемые либо от устройства синхронизации, либо от выделенных для этого выводов микросхемы МК. Таймеры имеют 16-разрядные счетчики с разными режимами работы. Причем в таймере B разрядность счетчика может программно установиться на 8, 10, 12 или 16 бит. Для фиксации (захвата) данных счета и для генерации временных интервалов имеются блоки захвата/сравнения. Таймер A имеет три таких блока, а в таймере B их число может быть увеличено (в зависимости от модификации) до семи. Любой из таймеров может выполнять множественный одновременный счет, множественные операции захвата/сравнения, формирование выходных сигналов специальной формы, в частности, сигналы с широтно-импульсной (ШИМ) модуляцией, а также любые комбинации названных операций;

11) *Сторожевой таймер*. Главная функция сторожевого таймера – выполнение контролируемого системой рестарта при сбое в работе программы: если программа не производит обращение к сторожевому таймеру в течение заданного интервала времени, то производится системный сброс. В тех случаях, когда данная функция таймера не требуется, он может быть запрограммирован на работу в качестве интервального таймера.

12) *Два универсальных синхронно/асинхронных приемо-передатчика (USART0,1)*. USART0 работает в трех режимах: в режиме универсального асинхронного приемо-передатчика (UART режим), в режиме синхронного периферийного интерфейса (SPI) и в режиме внутрисистемного обмена между интегральными схемами (I²C). В USART1 последний режим работы не предусмотрен.

В режиме UART при обмене данными между двумя устройствами каждый передаваемый символ содержит стартовый бит, семь или восемь битов данных, бит контроля четности и один или два стоповых бит. Период следования битов определяется выбранным источником тактовых импульсов и настройкой регистров скорости передачи. Когда два устройства обмениваются информацией асинхронно, в качестве протокола используется формат «свободная линия». При таком формате блоки данных на линиях передачи или приема разделены временем простоя (временем, когда линия свободна). Простой линии приема обнаруживается, когда приняты 10 или более не

изменяющихся со временем логических единиц (меток) после первого стопового бита символа. При использовании двух стоповых битов, второй стоповый бит принимается за первый маркерный бит периода простоя. Когда связываются три или более устройств, к биту данных добавляется адресный бит, что позволяет USART поддерживать многопроцессорные коммуникационные форматы со «свободной линией».

В синхронном режиме (в режиме SPI) последовательные данные передаются и принимаются множеством устройств с использованием общего тактирования, обеспечиваемого ведущим. Ведущий определяется по значению дополнительного сигнала (сигнала STE – разрешение передачи ведомого), управляемого ведущим. Сигнал STE необходим для разрешения приема и передачи данных устройством на выводе которого этот сигнал установлен.

Режим I²C является режимом управления взаимодействием между входящими в состав микропроцессорной системы интегральными схемами и поддерживает любые ведущие или ведомые устройства, совместимые с интерфейсом I²C через последовательную двухпроводную шину I²C. Внешние компоненты, присоединенные к шине I²C последовательно передают и/или принимают последовательные данные в/из USART;

13) *Шесть цифровых портов (P1-P6)* – это многофункциональные цифровые порты ввода/вывода, которые могут быть использованы как 8-разрядные параллельные порты общего назначения, так и как средство для подключения к контактам микросхемы МК таймеров, входов АЦП, выходов ЦАП, входов и выходов компаратора и последовательных портов, и для выдачи и приема сигналов синхронизации. Для этого предусмотрены средства конфигурации портов посредством занесения в их регистры управления соответствующей информации.

Система тактирования разработана специально для использования в приложениях с питанием от батарей. Вспомогательная низкочастотная система тактирования (ACLK – *Auxiliary Clock*) работает с обычным 32 кГц часовым кристаллом. Модуль ACLK может использоваться в качестве фоновой системы реального времени с функцией самостоятельного «пробуждения». Интегрированный в микросхему генератор с высокой скоростью перестройки по частоте и с цифровым управлением (DCO – *Digitally Controlled Oscillator*) является источником основного тактирования (MCLK – *Master Clock*) для ЦПЭ и высокоскоростных (SMCLK – *Sub-System Master Clock*) периферийных устройств. Модуль DCO становится активным и стабильным менее чем через 6 мкс после запуска. Архитектура MSP430 позволяют эффективно использовать 16-разрядный RISC ЦПЭ только в нужные промежутки времени, что позволяет на низких тактовых частотах снизить энергопотребление, сводя его до ультранизкого, а на высоких при активизации основного высокоскоростного модуля тактирования – выполнять быструю обработку сигналов. Основным модуль тактирования может быть программно сконфигурирован 1) на работу с номинальной частотой без использования внешних компонент кроме одного

внешнего резистора и 2) на работу с одним или двумя внешними кристаллами (резонаторами) XT1 и XT2. Внешние компоненты подключаются к соответствующим контактам Rosc, XIN, XOUT, XT2IN и XT2OUT.

Для тестирования системы используется стандарт IEEE P1149 JTAG (*Joint Test Automation Group*) – последовательный интерфейс «Объединенной рабочей группы по автоматизации тестирования». Этот стандарт определяет метод поочередного сканирования состояний входа-выхода каждого компонента системы. Последовательный порт JTAG используется также внутрисхемным эмулятором для доступа к встроенной системе поддержки тестирования. Эмуляторы используют порт JTAG для текущего контроля и управления процессором, установленным на целевой печатной плате, при отладке системы и ее программного обеспечения. Делается это с использованием программного симулятора в сочетании с устройством отладки, подключаемым к одному из последовательных или параллельных портов инструментальной ЭВМ.

Сканирование состояний входов/выходов (сканирование границ) позволяет разработчику системы проверять схему соединений на печатной плате с привлечением минимального количества специального контрольного оборудования. Сканирование возможно благодаря наличию возможности управления и отслеживания каждого входа и выхода на каждом кристалле с помощью набора последовательно просматриваемых защелок. Каждый вход и выход подсоединяется к своей защелке, а защелки соединяются в длинный регистр сдвига, так что данные могут считываться или записываться в них через последовательный тестовый порт.

Сканирование границ предусматривает разнообразные функции, которые можно выполнять для каждого входного и выходного сигнала. Каждый вход имеет защелку, которая контролирует значение входного сигнала, а также через нее могут вводиться данные в кристалл. Аналогично, каждый выход имеет защелку, которая контролирует значение выходного сигнала, и может также выводить данные вместо выходного значения. Для двунаправленных выводов возможна комбинация функций ввода и вывода. Взаимодействие с JTAG портом осуществляется с помощью сигналов:

- TCK (входной) – сигнал тактовой синхронизации операций тестовой логики. Используется для синхронизации данных в защелках просмотра и управляющей последовательности тестового конечного автомата;
- TMS (входной) – выбор тестового режима. Первичный управляющий сигнал. Синхронный по отношению к TCK. Последовательность значений на входе TMS задает текущее состояние порта тестирования;
- TDI/TCLK (входной/выходной) – вход тестовых данных или вход синхронизации. Отсюда последовательные входные данные, подаются в защелки просмотра;

– TDO/TDI (выходной/входной) – выход тестовых данных. На этот контакт поступают последовательные выходные данные, извлекаемые из защелок просмотра. Может быть запрограммирован для приема входных данных.

JTAG интерфейс используется также для загрузки во flash-память микроконтроллера программ.

При написании программ для МК возможны различные подходы. Это может быть программирование на языке высокого уровня (таким языком в данном случае является язык Си) или низкоуровневое программирование на языке Ассемблер. Однако и в том, и в другом случае необходимо учитывать особенности целевого процессора, учитывать его архитектуру (регистровую модель) и набор команд, с помощью которых реализуется вычислительный процесс. Команды МК (команды, первоначально представленные мнемониками языка Ассемблер и затем преобразованные в двоичный загружаемый в память код) управляют действиями ЦПЭ и работающего вместе с ним умножителя-аккумулятора (МАС). С помощью команд реализуются условные и безусловные переходы, вызовы и возвраты из процедур, сохранение содержимого регистров в памяти и загрузка данных из памяти в регистры, обращение к портам (регистрам) ввода/вывода.

Центральный процессорный элемент

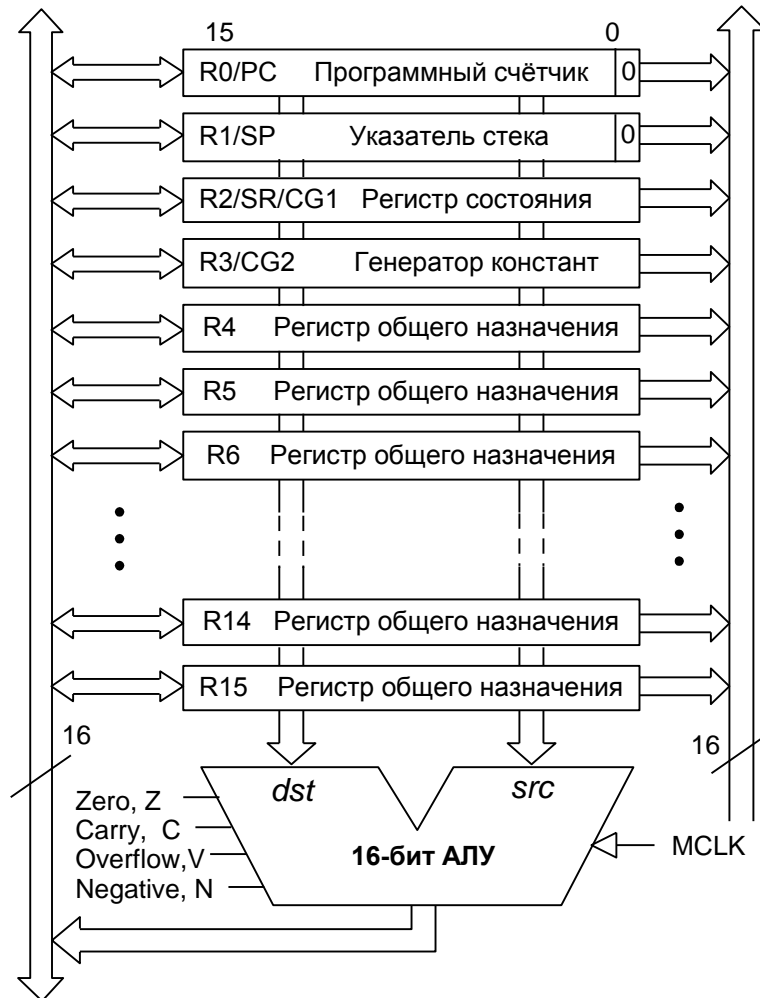


Рис. 9.7. Блок-схема ЦПЭ.

Центральный процессорный элемент (рис. 9.7) состоит из 16-битового арифметическо-логического устройства (АЛУ) и набора из 16-ти регистров – регистры R0-R15. Среди них 12 регистров R4-R15 – это регистры общего назначения, а четыре регистра R0-R3 выполняют соответственно функции счетчика команд PC, указателя вершины стека SP, регистра статуса SR и генератора констант CG.

Все операции ядра микроконтроллера выполняются с операндами, хранящимися в регистрах. Несмотря на некоторую специализацию части регистров (это относится к регистрам R0-R3), по отношению к АЛУ все регистры равноправны.

16-разрядный программный счетчик PC/R0 указывает на следующую команду, которая будет выполняться после исполнения текущей команды. PC инкрементируется соответственно тому, из какого четного числа байтов (два, четыре или шесть) состоит каждая команда. Команды располагаются в адресном пространстве 64 Кбайт и выравниваются по границам слов, поэтому значения PC всегда соответствуют четным адресам.

Указатель стека SP/R1 используется ЦПЭ для хранения адресов возврата из подпрограмм и процедур обработки прерываний. Стек построен по предекрементной постинкрементной схеме. Указатель стека SP может использоваться со всеми командами и во всех режимах адресации. Указатель стека SP инициализируется пользователем и выравнивается по четным адресам.

Регистр статуса (SR/R2) содержит признаки (флаги) АЛУ – признаки нуля Z (*zero*), переноса C (*carry*), переполнения V (*overflow*) и отрицательного результата N (*negative*), – а также бит общего управления прерываниями и биты управления таймерами. R2 используется как регистр источник или регистр приемник и при необходимости может быть использован для поддержки работы генератора констант.

Работу генератора констант CG поддерживают регистры R2 и R3, с помощью которых генерируются шесть констант (-1, 0, 1, 2, 4, 8), используемых при обращениях к памяти с использованием режимов косвенной регистровой и косвенной регистровой с атоинкрементацией или автодекрементацией адресации. Режим использования регистров R2 и R3 и значений генерируемых ими констант и выбираются в зависимости от режима адресации, определяемого соответствующими битами в кодах операций. Генератор констант дает следующие преимущества:

- не требуется дополнительное обращение к памяти для получения константы;
- позволяет сократить формат команд, связанных с обращением к памяти, поскольку не требуется дополнительного слова, если его значение совпадает с одним из значений шести автоматически генерируемых констант;
- ассемблер автоматически генерирует команды, рассчитанные на использование генератора констант, если одна из шести констант рассматривается как непосредственный операнд. При использовании регистров R2 и R3 в режиме генерации констант, их адресация не может быть явной – они действуют по умолчанию и только как регистры-источники;

Все эти регистры могут быть использованы в качестве регистров данных, указателей адресов или индексных значений и доступны при выполнении команд работы с байтами или словами.

Набор команд

Полный набор команд семейства MSP430 содержит 27 команд ядра и 24 эмулированные команды. Команды ядра – это команды, имеющие уникальный код операции, декодируемый ЦПЭ. Эмулированные команды представляют собой инструкции, облегчающие чтение и написание кода, но не имеющие собственного кода операции, поэтому ассемблер автоматически меняет их на эквивалентные команды ядра. Использование эмулирующих команд не

приводит к увеличению объема кода или снижению производительности. Существует три формата команд ядра:

- с двойным операндом – с операндом источником *src* и операндом приемником *dst* (таблица 9.2);
- с одиночным операндом – с одним операндом источником *src* или с одним операндом приемником *dst* (таблица 9.3);
- команды перехода, вызова, возврата (таблица 9.4).

Все команды с одним и двумя операндами могут быть командами для работы с байтами или со словами, на что указывают соответствующие расширения мнемоник «.B» или «.W». Если никакое расширение не используется, то такие команды работают со словами.

Режимы адресации

При работе с данными и реализации переходов используются несколько способов вычисления исполнительного адреса операндов *src* и *dst*, называемых режимами адресации. Если говорить об адресации данных, то это определение адресов операндов источников *src* и операндов приемников *dst*. Режимы адресации микроконтроллеров семейства MSP430 представлены в таблице 1.

Таблица 9.1

Режим адресации	Синтаксис	Описание
Регистровый	Rn	Содержимое регистра Rn является операндом.
Индексный	X(Rn)	Значение (Rn+X) указывает на операнд – ячейку памяти. Значение X сохранено в следующем за кодом команды слове.
Символьный	ADDR	Адрес ADDR определяется по смещению X относительно PC. Другими словами, на операнд указывает значение (PC+X). Значение X сохранено в следующем слове. Реализован как индексный режим X(PC)

Абсолютный	&ADDR	Слово, следующее за командой с адресом (PC + X=0), содержит адрес операнда. Реализован как индексный режим X(R2) с использованием генератора констант – константы 0, извлекаемой из регистра R2.
Косвенный регистровый	@Rn	Содержимое регистра Rn используется как указатель на операнд.
Косвенный с автоинкрементацией	@Rn+	Содержимое Rn используется как указатель на операнд. Содержимое Rn впоследствии увеличивается на 1 для байтовых команд и на 2 для команд-слов.
Прямой (непосредственный)	#N	Слово, следующее за командой, содержит непосредственную константу N. Реализован как косвенный автоинкрементный режим @PC+.

Команды с двойным операндом

Таблица 9.2

Команда	Действие команды	Биты статуса			
		V	N	Z	C
MOV(.B) <i>src, dst</i>	<i>src</i> → <i>dst</i>	-	-	-	-
ADD(.B) <i>src, dst</i>	<i>src</i> + <i>dst</i> → <i>dst</i>	*	*	*	*
ADDC(.B) <i>src, dst</i>	<i>src</i> + <i>dst</i> + C → <i>dst</i>	*	*	*	*
SUB(.B) <i>src, dst</i>	<i>src</i> + .not. <i>dst</i> + 1 → <i>dst</i>	*	*	*	*
SUBC(.B) <i>src, dst</i>	<i>src</i> + .not. <i>dst</i> + C → <i>dst</i>	*	*	*	*
CMP(.B) <i>src, dst</i>	<i>dst</i> – <i>src</i>	*	*	*	*
DADD(.B) <i>src, dst</i>	<i>src</i> + <i>dst</i> + C → <i>dst</i> (десятичное)	*	*	*	*
BIT(.B) <i>src, dst</i>	<i>src</i> .and. <i>dst</i>	0	*	*	*
BIC(.B) <i>src, dst</i>	.not. <i>src</i> .and. <i>dst</i> → <i>dst</i>	-	-	-	-
BIS(.B) <i>src, dst</i>	<i>src</i> .or. <i>dst</i> → <i>dst</i>	-	-	-	-
XOR(.B) <i>src, dst</i>	<i>src</i> .xor. <i>dst</i> → <i>dst</i>	*	*	*	*
AND(.B) <i>src, dst</i>	<i>src</i> .and. <i>dst</i> → <i>dst</i>	0	*	*	*

«*» – влияет на бит статуса;

«-» – не влияет на бит статуса;

«0» – бит статуса очищается;

В круглых скобках необязательное расширение мнемоник Ассемблера.

Команды с одним операндом

Таблица 9.3

Команда	Действие команды	Биты статуса			
		V	N	Z	C
RLA(.B) <i>dst</i>	Арифметический сдвиг вправо на 1 бит.	0	*	*	*
RLC(.B) <i>dst</i>	Циклический сдвиг вправо на 1 бит через триггер переноса C.	*	*	*	*
RRAB) <i>dst</i>	Арифметический сдвиг вправо на 1 бит.	0	*	*	*
RRC.B) <i>dst</i>	Циклический сдвиг вправо на 1 бит через триггер	*	*	*	*

	переноса C.				
PUSH(.B) <i>dst</i>	SP-2 → SP, <i>src</i> → @SP. Проталкивание в стек.	-	-	-	-
SWPB <i>dst</i>	Переставить байты местами.	-	-	-	-
CALL <i>dst</i>	SP-2 → SP, PC+2 → @SP, <i>dst</i> → PC. Вызов процедуры.	-	-	-	-
RET	@SP → PC SP+2 → SP. Возврат из процедуры	*	*	*	*
RETI	TOS → SR, SP+2 → SP; TOS → PC, SP+2 → SP. Возврат из прерывания (TOS – вершина стека).	*	*	*	*
SXT	Bit 7 → Bit 8.....Bit 15. Расширение знака байта на слово.	0	*	*	*

«*» – влияет на бит статуса;

«-» – не влияет на бит статуса;

«0» – бит статуса очищается.

Команды перехода

Таблица 9.4

Команда	Действие команды
JEQ/JZ <i>метка</i>	Переход к метке, если бит Z нуля установлен.
JNE/JNZ <i>метка</i>	Переход к метке, если бит Z нуля сброшен.
JC <i>метка</i>	Переход к метке, если бит C переноса установлен.
JNC <i>метка</i>	Переход к метке, если бит переноса C сброшен.
JN <i>метка</i>	Переход к метке, если бит N отрицательного результата установлен
JGE <i>метка</i>	Переход к метке, если (N.XOR.V)=0.
JL <i>метка</i>	Переход к метке, если (N.XOR.V)=0.
JMP <i>метка</i>	Безусловный переход.

«*» – влияет на бит статуса; «-» – не влияет на бит статуса;

«0» – бит статуса очищается.

Условные переходы обеспечивают ветвление программы относительно программного счетчика PC и не оказывают влияния на биты статуса. Возможный диапазон переходов с помощью команды перехода составляет от -511 до +512 слов относительно текущего значения PC. 10-разрядное смещение программного счетчика обрабатывается как 10-разрядное значение со знаком: удваивается и складывается с содержимым программного счетчика.

Умножитель-аккумулятор (MAC)

Умножитель поддерживает операции умножения без знака, умножения со знаком, умножения без знака с накоплением и умножение со знаком и накоплением. Тип операции выбирается адресом, в который записан первый операнд. Умножитель имеет два 16-разрядных регистра OP1 и OP2 и три регистра результата RESLO, RESHI и SUMEXT (рис. 9.8). В регистре RESLO

содержится младшее слово результата, в RESHI – старшее слово результата, а в регистре SUMEXT находится информация о результате. Для появления результата необходимо 3 такта MCLK. Результат может быть прочитан следующей командой после записи в OP2. Исключение составляет случай, когда используется косвенный режим адресации к регистру результата. В этом случае необходимо вставить команду NOP перед чтением результата.

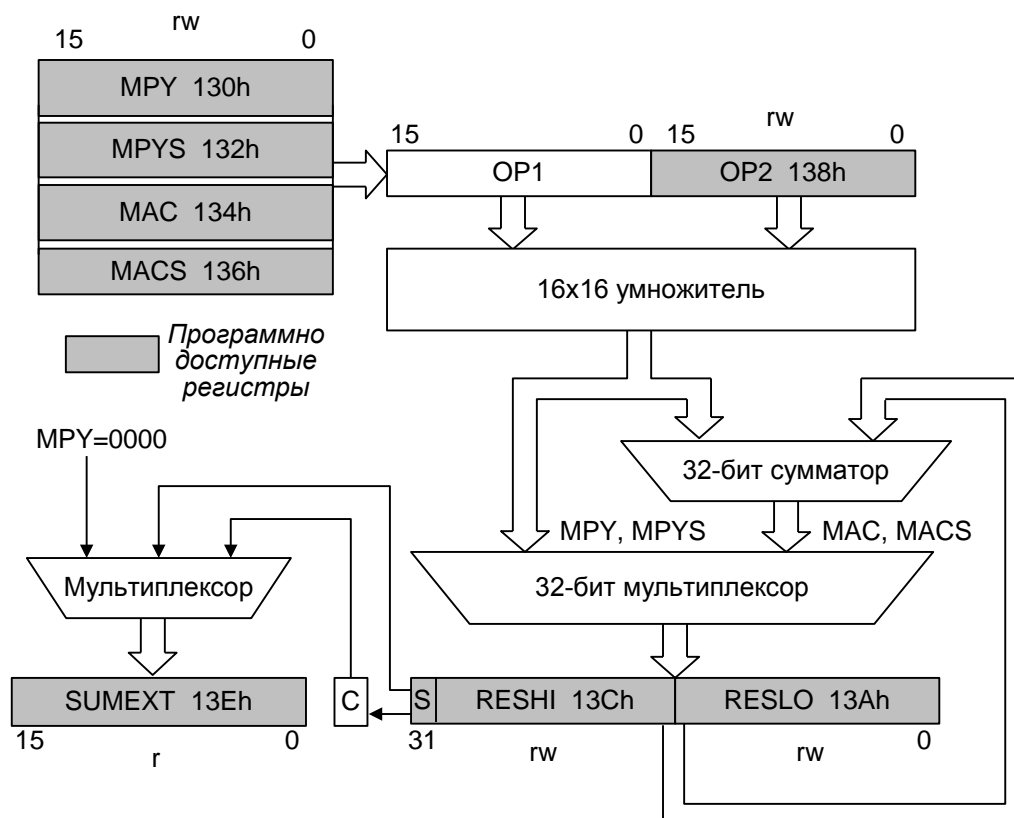


Рис. 9.8. Структурная схема умножителя-аккумулятора.

Регистр OP1 первого операнда имеет четыре адреса 130h (MPY), 132h (MPYS), 134h (MAC) и 136h (MACS), используемые при выборе режима умножения. Запись первого операнда по желаемому адресу позволит выбрать тип операции умножения, но не приведет к началу выполнения какой-либо операции. Запись операнда в регистр OP2 второго операнда инициирует операцию умножения. Запись в OP2 стартует выбранную операцию над значениями, сохраненными в OP1 и OP2. Результат записывается в три регистра результата RESLO, RESHI и SUMEXT. Повторение операций умножения может выполняться без перезагрузки OP1, если значение в OP1 используется для последовательных операций и если нет необходимости перезаписывать значение в OP1 для выполнения операций.

Результат размещается в двух регистрах – в младшем RESLO и старшем RESHI 16-разрядных регистрах результата. Содержимое старшего регистра результата RESHI зависит от операции умножения.

Заканчивая рассмотрение архитектуры и команд ЦПЭ МК MSP4301161 необходимо упомянуть систему прерываний этого микроконтроллера, поскольку все входящие в состав МК могут обслуживаться в режиме прерываний. Логика обработки запросов прерывания имеет следующую последовательность:

1. любая текущая команда выполняется до конца;
2. содержимое программного счетчика PC, указывающего на следующую команду, помещается в стек;
3. содержимое регистра статуса SR помещается в стек;
4. если поступило несколько прерываний во время выполнения последней команды, обрабатывается прерывание с наивысшим приоритетом, остальные ожидают обслуживания;
5. автоматически сбрасывается флаг одного источника прерывания. Флаги запроса остальных прерываний остаются установленными в ожидании обслуживания программным обеспечением;
6. регистр SR очищается, за исключением бита, отвечающего за работу системного тактового генератора 0, который остается неизменным. В результате прекращается работа в любом режиме пониженного энергопотребления;
7. содержимое связанного с устройством ввода-вывода прерывания загружается в PC и начинается выполнение процедуры обработки прерывания с загруженного адреса. Вектор прерывания ассоциируется с источником запроса, и в виде двоичного числа сохраняется в регистре вектора прерывания. Затем это число загружается в программный счетчик для входа в соответствующую процедуру обработки прерывания.

Векторы прерываний и стартовые адреса расположены в таблице прерываний с адресами от 0FFFFh до 0FFE0h. Вектор программируется пользователем с помощью указания 16-разрядного стартового адреса соответствующей процедуры обработки прерывания.

Любая процедура обработки прерывания заканчивается командой RETI (*возврат из подпрограммы обработки прерывания*). При возврате из прерывания выполняются следующие действия: из стека восстанавливаются

1. содержимое регистра SR со всеми его предыдущими установками и
2. содержимое программного счетчика PC, после чего продолжается выполнение программы с того места, где она была прервана.

9.4. Микроконтроллеры для коммуникационных приложений

Рассмотренный в разделе 9.3 микроконтроллер из семейства МК MSP430 фирмы Texas Instruments достаточно полно иллюстрирует ту особенность этих микропроцессорных кристаллов, которая связана с интегрированными в МК периферийными устройствами, набор которых определяется конкретными

приложениями. В зависимости от приложения строится и ядро (ЦПЭ) микроконтроллера, вычислительные возможности которого и объем используемой памяти могут приближаться к возможностям универсального процессора.

Возможности микроконтроллеров рассмотрим, обратившись к архитектуре ARM (*Advanced RISC Machine*), разработанной компанией ARM Limited. Архитектуру ARM отличает то, что на ее основе создаются как простые встроенные микросистемы с низким энергопотреблением, так и сложные многофункциональные аппаратно-программные комплексы, работающие под управлением таких операционных систем, как JavaOS, Linux, Microsoft WindowsCE и др.

В первом случае системы используют ресурсы микропроцессорного ядра, интегрированного в один кристалл вместе с разнообразными устройствами ввода-вывода (включая простейшие параллельные и последовательные цифровые порты и сложные, поддерживающие различные сетевые протоколы каналы связи). Сюда следует отнести, прежде всего, коммуникационные микроконтроллеры. Обычно их используют в тех приложениях, в которых конечный пользователь никогда не добавляет программное обеспечение к системе.

Структурная схема одного из микроконтроллеров для коммуникационных приложений (МК ARM7DMI), все компоненты которого интегрированы в одну микросхему, представлена на рис. 9.9.

Центральный процессорный элемент (ЦПЭ, CPU – *Central Processor Unit*) через встроенный интерфейс (*CPU Interface*) связан с кэш-памятью, где хранятся поступающие в ЦПЭ команды и данные, с буфером записи (*Write Buffer*) и с шинным маршрутизатором (*BUS Router*). Буфер записи используется при сохранении содержимого регистров в памяти: при выполнении инструкции записи в память ЦПЭ делает эту запись не непосредственно в память, а через буфер записи и, не ожидая перезаписи содержимого буфера в основную память, продолжает свою работу. Данные из буфера записи в основную память переносятся в то время, когда системная шина свободна и не занята каким-либо другим активным устройством, или когда механизм шинного арбитража предоставит системную шину в распоряжение процессора. Это освобождает ЦПЭ от необходимости ожидания магистрали для реализации обращения к памяти.

Шинный маршрутизатор управляет перемещением информации по системной магистрали между центральным процессорным элементом и расположенными в микросхеме МК устройствами. Среди устройств, включенных в состав микроконтроллера, имеются:

1. Контроллер прерываний (*Interrupt Controller*).
2. Двухканальный контроллер прямого доступа к памяти (*CDMA – General Direct Memory Access*).
3. Два универсальных асинхронных приемо-передатчика *UART – Universal Asynchronous Receiver/Transmitter* и интерфейс I^2C – *Inter-Integrated*

Circuit с линией передачи адреса/данных (SDA – *Serial Data/Address line*) и с линией синхронизации (SCL – *Serial Clock line*) (описание последних см. в разделах 9.2, 9.3).

4. Порт с 18-ю двунаправленными входами/выходами (контактами), которые могут быть сконфигурированы как самостоятельные цифровые порты ввода-вывода и как контакты, предназначенные для передачи входных и выходных сигналов входящих в состав МК микросхемы устройств. Среди них 4 сигнала запроса прерывания (*Ext INT req.*), два запроса ПДП (*Ext DMA REQ.*) и два сигнала подтверждения ПДП (*Ext DMA ACK*) от устройств, расположенных за пределами кристалла микроконтроллера, два выходных сигнала таймеров (*Timer out (0, 1)*).

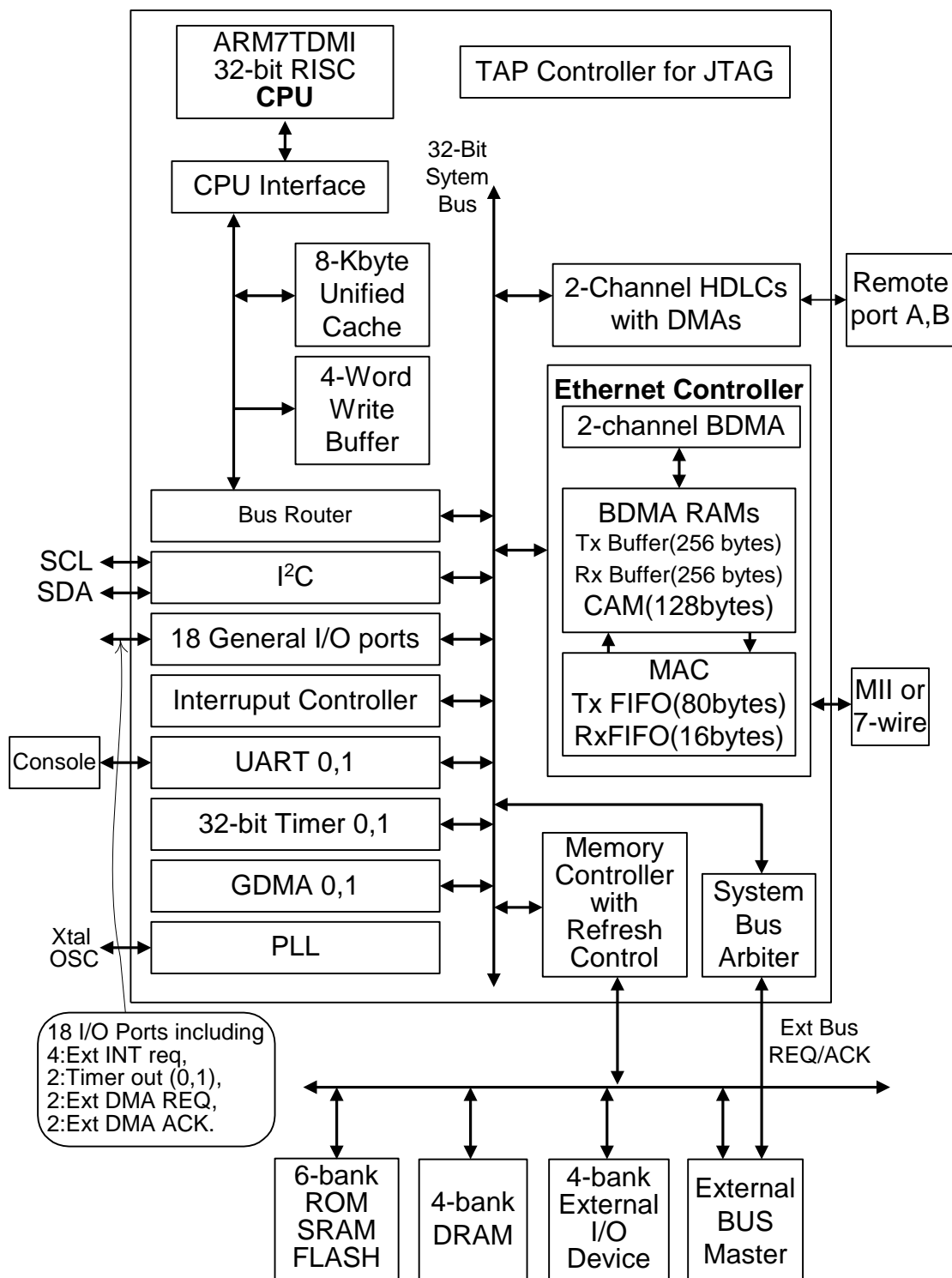


Рис. 9.9. Структура микроконтроллера для коммуникационных приложений

5. Два контроллера, поддерживающие высокоуровневый протокол канала пакетной передачи данных (HDLCs – *High Level Data Link Controllers*) для связи с удаленными портами (*Remote port A, B*) и имеющие полностью независимую приемную и передающую части. В состав каждого контроллера входят контроллеры прямого доступа к памяти DMA (*Direct Memory Access*) для обмена данными с основной памятью МК.

6. Ethernet контроллер поддерживает множественный доступ к коммуникационной сети через МП-интерфейс (*Media-Independent Interface*), а также через 7-проводной (*7-wire*) последовательный интерфейс. Имеет встроенный контроль сетевого доступа (MAC – *Media Access Control*) с двумя организованными как очередь FIFO (*First In First Out*) буфера памяти – один на прием, другой на передачу. Для обмена данными с основной памятью Ethernet контроллер использует системную шину. Операции чтения и записи выполняются под управлением 2-канального контроллера ПДП (*Buffered DMA*) с отдельными для принимаемых и передаваемых данных буферными оперативными запоминающими устройствами (RAM). При приеме Ethernet-пакетов адресная информация сравнивается с содержимым ассоциативной памяти САМ (*Content Addressable Memory*). Если обнаруживается совпадение, то управляющая приемом микропрограммный автомат продолжает прием пакета информации. В противном случае могут быть предприняты действия с целью посылки источнику информации контрольных пакетов, используемых для дальнейшего продолжения связи.

7. Два 32-разрядных таймера – *Timer* (0, 1).

Работа микроконтроллера тактируется внешним синхронизирующим сигналом 10-40 МГц, частота которого задается кварцевым резонатором Xtal. Тактовая частота внутри МК микросхемы увеличивается в пять раз, для чего используется цифровая система фазовой автоподстройки PLL (*Phase-Locked Loop*)

Микроконтроллер может работать с разными типами постоянной (ROM) и оперативной (динамической и статической) памяти (RAM) и взаимодействовать с находящимися за пределами микросхемы устройствами ввода-вывода, а также с другими процессорами или сопроцессорами в составе многопроцессорной системы. К числу сопроцессоров относится, в частности, тот, который расширяет возможности микроконтроллера при работе с памятью, увеличивая размер адресного пространства и реализуя механизм виртуальной адресации и со необходимыми средствами защиты.

Память системы делится на банки и для каждого банка устанавливается свое время доступа, что дает возможность отображения на память портов внешних устройств ввода-вывода (*External I/O Devices*). Управление внешней шиной осуществляет контроллер памяти (*Memory Controller with Refresh Control*), одной из функций которого является регенерация динамической памяти, если таковая в системе имеется.

Возможности ARM-систем могут быть расширены добавлением внешних сопроцессоров, в частности, сопроцессоров с плавающей точкой и управляющего сопроцессора. Последний предоставляет дополнительные средства для конфигурации и управления системой, включая средства защиты памяти, что важно для многозадачных приложений. При наличии других ведущих на внешней системной магистрали (*External BUS Master*), возникает необходимость шинного арбитража – разделения ресурсов шины между

отдельными потенциально ведущими. Эту задачу решает арбитр системной шины (*System Bus Arbiter*).

При создании сложных многофункциональные аппаратно-программные комплексы используются ARM микропроцессоры с кэшированным макроядром, способным выполнять операции с многобайтовыми данными и работать совместно с встроенными устройствами управления системой (с встроенным в кристалл управляющим сопроцессором). Такое макроядро ориентировано на использование в «открытых» системах, для которых необходимы полное управление виртуальной памятью и развитая защита собственной памяти.

Все это позволяет создавать масштабируемые системы с возможностью выбора необходимых для конкретных приложений микропроцессорных ресурсов. Этому служит то, что ARM-компоненты могут быть выполнены как на отдельных микросхемах (например, отдельно ЦПЭ с устройствами ввода/вывода, отдельно сопроцессор управления памятью и сопроцессор с плавающей точкой), так и быть интегрированными в одну микросхему. Кроме этого, ARM-процессоры, работающие обычно с 32-разрядными командами, могут управляться укороченными до 16-ти разрядов Thumb-инструкциями, что повышает компактность создаваемого для них программного кода.

ARM МП являются RISC-процессорами, поэтому

- операции в них выполняются только с данными, находящимися в регистрах,
- отдельно выполняются команды загрузки/сохранения регистров,
- режимы адресации используют только значения, взятые из регистров или из определенных в командах полей,
- все команды имеют фиксированный размер (32 бита при обычной работе и 16 бит при работе в Thumb-режиме),
- все команды являются условно исполняемыми.

В состав ARM микропроцессора помимо перечисленных устройств входят расширения отладки: контроллер TAP (*Test Access Port*) сканирования состояния входов и выходов интегрированных в МП ядро функциональных узлов (*boundary scan*) . Ядро процессора с подключенными к нему средствами отладки называют ARM-макроядром. Аппаратные расширения отладки облегчают разработку пользовательского прикладного программного обеспечения, операционных систем и самих аппаратных средств. Аппаратные расширения отладки позволяют останавливать ядро или при выборке заданной команды (в контрольной точке), или при обращении к данным (в точке просмотра), или асинхронно – по запросу в процессе отладки. В точках останова через JTAG последовательный интерфейс могут быть исследованы внутреннее состояние ядра, находящегося в состоянии отладки, и внешние признаки состояния системы. По завершении процесса отладки состояния ядра и системы могут быть восстановлены, а выполнение программы продолжено.

Режим отладки устанавливается запросом по одной из линий внешнего интерфейса отладки и программируется в последовательном режиме с использованием контроллера TAP – средства управления работой цепочек сканирования через последовательный интерфейс JTAG.

Некоторые ARM-микропроцессоры для расширения возможностей отладки имеют встроенный сопроцессор.

10. Микропроцессоры с гарвардской архитектурой и микросистемы на их основе

Вполне очевидным способом повышения производительности процессоров является распараллеливание операций, которое может быть выполнено разными способами и, в частности, путем параллельной работы устройств, отвечающих за обработку данных, за вычисление исполнительного адреса данных и адресов команд. Эффективным способом такого распараллеливания является использование независимых аппаратных средств для работы с данными и программным кодом, что свойственно гарвардской архитектуре, в основу которой положены отдельные память программ и память данных. Такая архитектура широко используется в цифровых процессорах сигналов (ЦПС). Черты гарвардской архитектуры отражены в цифровых процессорах сигналов. Их можно найти и в микроконтроллерах, и в универсальных процессорах. Одним из проявлений особенностей гарвардской архитектуры является использование отдельных кэш-памяти программ и кэш-памяти данных.

10.1. Цифровые процессоры сигналов семейства ADSP 21xx

Особенности функционирования процессоров с гарвардской архитектурой рассмотрим, воспользовавшись тем, как эти особенности отражены в цифровых процессорах сигналов фирмы Analog Devices. Обращение к процессорам этой фирмы обусловлено тем, что в них наиболее ясно выражены не только аппаратные решения, но и средства программирования процессоров с гарвардской архитектурой.

10.1.1. Структура ядра

Структура ядра цифровых процессоров сигналов, принадлежащих одному из первых семейств ЦПС фирмы Analog Devices – семейства ADSP-21xx, – представленная на рис. 10.1. Процессоры имеют интегрированную в МП кристалл память программ РМ (*Program Memory*) и память данных DM (*Data Memory*). При необходимости к этой памяти может быть добавлена внешняя, расположенная за пределами микропроцессора память.

Доступ к памяти системы осуществляется по четырем шинам:

- к памяти программ по шине адреса памяти программ РМА (*Program Memory Address*) и шине данных памяти программ РМД (*Program Memory Data*);

- к памяти данных по шине адреса памяти данных DMA (*Data Memory Address*) и шине данных памяти данных DMD (*Data Memory Data*).

В ядро ЦПС семейства ADSP-21xx входят: устройство выполнением последовательностью выполнения программы (*Program Sequencer*), два генератора адресов памяти данных DAG1,2 (*Data Address Generator 1, 2*) и операционный блок, составленный из арифметическо-логического устройства ALU (*Arithmetic-Logic Unit*), умножителя-аккумулятора MAC (*Multiplier-Accumulator*) и сдвигателя (*Shifter*). Структура ядра оптимизирована для выполнения наиболее часто встречающихся операций при цифровой обработке сигналов (вычисление свертки, функции корреляции, быстрого преобразования Фурье и др). Результаты операций ALU, MAC и сдвигателя размещаются в принадлежащих им выходных регистрах, после чего могут быть сохранены в памяти или в качестве операндов переданы из одного операционного блока в другой. Для этого служит дополнительная шина – шина результата (R BUS – *Result Bus*).

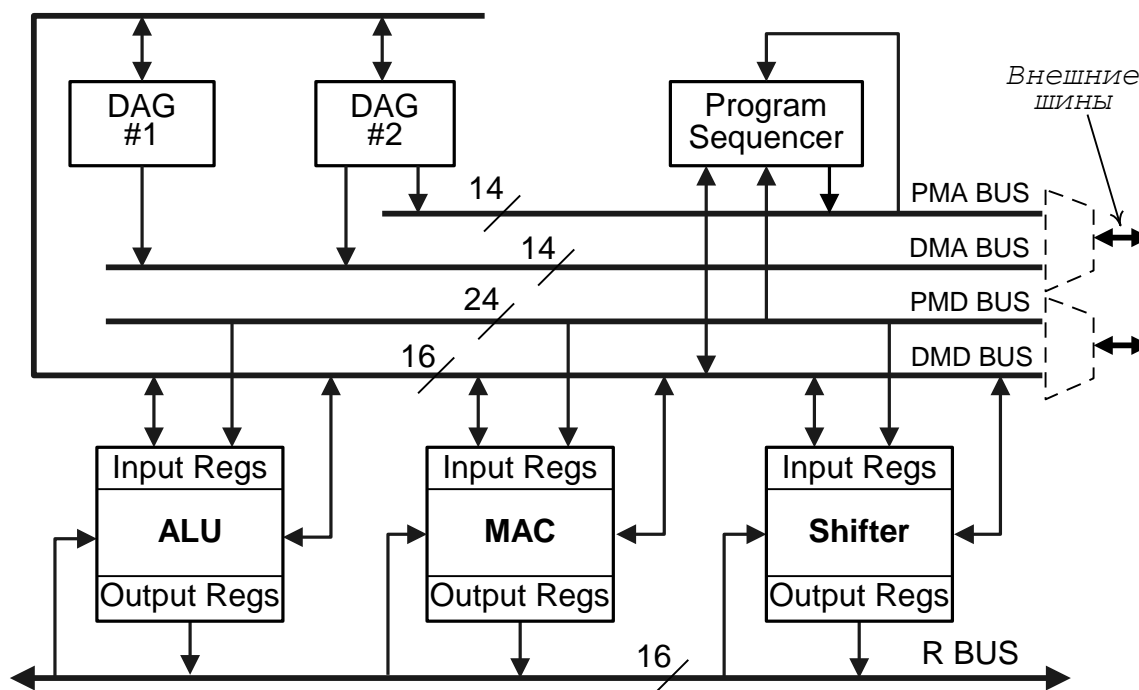


Рис. 10.1. Структура ядра цифрового процессора сигналов семейства ADSP-21xx

Все операции выполняются только с содержимым регистров – входных (Input Regs) или выходных (Output Regs). По этому признаку МП ADSP 21xx можно классифицировать как RISC-процессоры. Все команды процессора умещаются в 24-разрядные слова, чем определяется размер ячеек памяти программ и разрядность шины PMD. Единицей памяти данных ЦПС ADSP 21xx являются 16-разрядные слова, поэтому 16-разрядной является шина данных DMD.

Вместе с ядром и внутренней памятью программ и данных на кристалле микропроцессора размещены таймеры, устройства ввода-вывода и другие компоненты, состав которых зависит от конкретной модели МП ADSP 21xx.

Двойное адресное пространство требует удвоенного количества адресных линий и линий данных. В результате поиска оптимального решения между ценой и производительностью была создана *модифицированная гарвардская архитектура*. Ее особенностью является наличие только одной внешней шины адреса и одной внешней шины данных, формируемых путем мультиплексирования внутренних шин PMA и DMA, PMD и DMD.

Модифицированная гарвардская архитектура давно используется большинством производителей сигнальных процессоров. Однако в настоящее время деление внутренней памяти на память программ и память данных свойственно не только ЦПС, но и универсальным процессорам, и микроконтроллерам, хотя в их внешней памяти такое деление может отсутствовать.

На рис. 10.2 в качестве примера представлена МП система на базе ЦПС семейства ADSP-21xx с процессором, у которого есть таймер и два последовательных порта (SPORT1,2). Другие МП этого семейства могут иметь контроллер прямого доступа к памяти или хост-интерфейс для связи с управляющей (главной) ЭВМ, например, с персональным компьютером.

Архитектура системы – модифицированная гарвардская. Обращение к внешней памяти программ и памяти данных происходит с разделением времени в сопровождении селектирующих сигналов \overline{PMS} (*Program Memory Select*) и \overline{DMS} (*Data Memory Select*) и сигналов синхронизации (квитирующих сигналов) \overline{WR} и \overline{RD} .

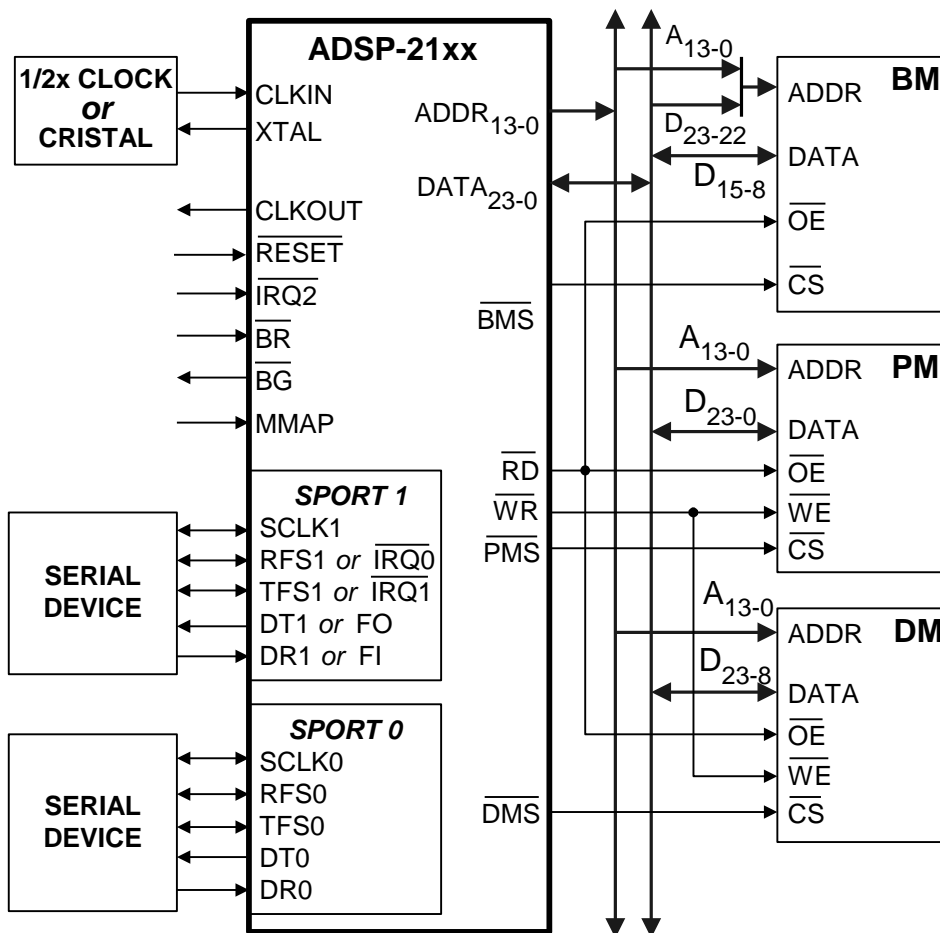


Рис. 10.2. ADSP-21xx система с внешней памятью

Память данных не является однородной и делится на блоки с устанавливаемым для каждого блока временем доступа. Часть адресного пространства DM выделена для регистров (портов) ввода-вывода.

Кроме оперативной памяти PM и DM в состав микросистемы входит внешнее репрограммируемое ПЗУ с байтовой организацией BM (*Boot Memory*). Обычно это 8-разрядное ППЗУ, в котором хранится код начальной загрузки, переносимый в память программ при старте ЦПС или при его программном сбросе. Нередко в качестве ППЗУ применяется флэш-память. Для доступа к BM используется селектирующий сигнал \overline{BMS} (*Boot Memory Select*). В случае, когда объем BM превышает свойственное процессору адресное пространство, недостающие для BM биты адреса передаются по внешней шине данных в мультиплексном режиме. В представленной на рис. 10.2 системе для этого используются линии D_{23-22} старших разрядов.

Интерфейс памяти с байтовой организацией можно построить так, что он будет поддерживать загрузку по ходу выполнения программы. При этом содержимое определенного числа ячеек внутренней памяти ЦПС может быть передано во внешнюю память или получено из нее без перерыва основной работы процессора.

В системе на рис. 10.2 использован процессор из конкретного семейства микросхем. Поэтому есть особенности его применения, требующие пояснений. Касаются они размещенных на кристалле ЦПС устройств ввода-вывода и принципиального значения не имеют. Кроме того, пояснений требуют сигналы, которыми процессор обменивается с внешней средой.

1) В составе ЦПС имеются два синхронных последовательных порта – SPORT 0 и SPORT 1. Принимаемые и передаваемые через эти порты данные (линии DR0,1 – *Data Receive* и DT0,1 – *Data Transmit* соответственно) формируются в кадры и независимо от направления передачи сопровождаются сигналами побитовой SCLK0,1 и кадровой RFS0,1 (*Receive Frame Synchronization*) и TFS0,1 (*Transmit Frame Synchronization*) синхронизации. Сигналы побитовой синхронизации представляют собой периодическую последовательность импульсов SKLC0 или SKLC1. Линии сигналов синхронизации – двунаправленные. Это делает возможной синхронизацию либо со стороны последовательного порта, либо со стороны связанного с ним устройства.

2) Последовательный порт SPORT 1 наделен дополнительными функциями: его можно сконфигурировать не как сериальный порт, а как порт для восприятия запросов прерываний $\overline{IRQ0}$ и $\overline{IRQ1}$ от внешних устройств, а также для передачи флагов готовности IF (*Input Flag*) и OF (*Output Flag*) по вводу и выводу.

3) Для внешних запросов прерываний имеется выделенная линия $\overline{IRQ2}$. Эта линия обычно используется для управления работой ADSP 21xx системы. Управляющим устройством может быть, например, компьютер, к которому ADSP 21xx система подключена как устройство ввода-вывода. При передаче управляющего воздействия код воздействия заносится в регистр управления, после чего генерируется запрос прерывания, передаваемый по линии $\overline{IRQ2}$. Процессор, получив сигнал запроса $\overline{IRQ2}$, запускает процедуру обработки, действия которой сводятся к чтению содержимого регистра управления, после чего происходит обработка управляющего воздействия.

4) Сигналы \overline{BR} (*Bus Request*) и \overline{BG} (*Bus Grant*) – это, соответственно, сигнал запроса и сигнал подтверждения прямого доступа к памяти. Сигнал \overline{BR} поступает со стороны внешнего по отношению к ADSP 21xx ведущего на системной магистрали, например, от управляющей ЭВМ тогда, когда необходимо прочитать данные из памяти или записать данные в память ADSP 21xx системы.

5) Для синхронизации процессора используется либо внешний сигнал CLKIN, либо сигнал внутреннего генератора с подключенным к нему через контакты CLKIN и XTAL кристаллом кварца.

6) Для аппаратного сброса процессора применяется сигнал \overline{RESET} . После сброса процессор считывает занесенный в ВМ программный код, занося его в свою оперативную память программ, и приступает к выполнению программы.

7) Высоким или низким уровнями напряжения на входе ММАР (*Memory Map*) устанавливается одна из двух возможных конфигураций памяти данных, определяющих то, как распределено адресное пространство между внутренней и внешней памятью DM.

10.1.2. Программный автомат

Для работы с памятью программ предназначено устройство управления последовательностью выполнения программы – *программный автомат* (*Program Sequencer*), генерирующий адреса РМ, при последовательном исполнении программного кода и при реализации переходов, вызовов, возвратов и циклов. Работа программного автомата (ПА) была рассмотрена в разделе 7.4. Здесь поясним лишь связь ПА с шинами микропроцессорного ядра и сделаем дополнения, касающиеся обработки запросов прерываний.

Представленная на рис. 7.4 шина адреса ША соответствуют шине РМА на рис. 10.1, а шина данных ШД – шине DMD. Обозначенная на рис. 10.1 дополнительная связь ПА с шиной РМД отражает то, что процессор в основном машинном цикле – цикле выборки команды – извлеченную из памяти программ команду размещает в регистре команд *IR*, а информация из *IR* (код операции, возможные ветвления и их условия) используется программным автоматом при генерации адреса следующей инструкции.

При обработке прерываний, как уже отмечалось в разделе 7.4, статусные регистры процессора сохраняются в стеке флагов, а при окончании обработки прерывания – восстанавливаются.

После восприятия запроса прерывания процессор обращается к таблице прерываний (как пример см. таблицу 7.7) по соответствующему запросу вектору прерываний. Вектора прерываний расположены в памяти программ с интервалом в четыре ячейки, начиная с нулевого адреса. Это позволяет размещать короткие (не более четырех слов) процедуры обслуживания прерываний в пределах одного элемента таблицы прерываний. В случаях, когда процедура обработки прерываний содержит более четырех команд, обслуживающая подпрограмма вызывается с помощью команды перехода. После обслуживания прерывания управление возвращается прерванной программе при помощи команды возврата из прерывания *RTI*, в ходе выполнения которой верхнее значение стека РС выставляется на шину РМА и далее заносится в счетчик команд. Кроме этого восстанавливается предыдущее состояния процессора, для чего содержимое вершины стека состояний переносится в соответствующие статусные регистры.

Есть два типа запросов прерывания – маскируемые и немаскируемые. К немаскируемым запросам прерывания относится, например, запрос от источника питания, возникающий при сбое в его работе. Немаскируемые прерывания за счет аппаратно реализованных стеков обрабатываются без

задержки (кроме задержки, связанной с синхронизацией). Маскируемые (не обрабатываемые в текущее время) прерывания обозначаются в регистре маски контроллера прерываний, являющемся программно доступным регистром.

Для обслуживания высокоприоритетных запросов предусмотрен механизм вложенных прерываний, который позволяет обслуживать запросы, если они поступают во время исполнения ранее вызванной процедуры обработки прерывания ISR, и если уровень приоритета вновь поступившего запроса превышает приоритет обрабатываемого. Но такое обслуживание возможно только в том случае, если восприятие вложенных запросов прерывания обеспечено со стороны процессора.

При получении запроса прерывания процессор фиксирует его на время, которое необходимо, чтобы закончилось выполнение текущей команды. Затем контроллер прерываний сравнивает поступивший запрос с регистром маски прерываний. В случае немаскируемого прерывания программный автомат помещает текущее значение счетчика команд в стек PC. Статусные регистры (в том числе и регистр маски) помещаются в стек состояний. В регистр маски загружается новое значение, которое определяет, разрешается или не разрешается дальнейшее использование вложенных прерываний.

Затем процессор выполняет команду NOP (нет операции), одновременно выбирая команду, расположенную по адресу вектора прерывания. По возвращении из подпрограммы обслуживания прерывания восстанавливаются значения счетчика команд и регистров состояний (включая регистр маски) путем извлечения их содержимого из стека PC и стека состояний и выполнение прерванной программы возобновляется.

Каждое отдельное прерывание (внешнее или внутреннее)

- маскируется или разрешается,
- конфигурируется на восприятие по перепаду или по уровню.

Имеется возможность принудительного прерывания, а также сброса задержанного чувствительного к перепаду прерывания. Для этого устанавливаются соответствующие биты в одном из регистров процессора, отвечающем за конфигурацию прерываний. Запросы внешних прерываний $\overline{IRQ0}$, $\overline{IRQ1}$ и $\overline{IRQ2}$ (см. рис. 10.2) могут быть сконфигурированы как воспринимаемые по перепаду или по уровню.

При обработке сконфигурированных по перепаду прерываний запрос на прерывание фиксируется каждый раз, когда уровень сигнала запроса $\overline{IRQ0-2}$ на соответствующем контакте микросхемы процессора изменяется от высокого к низкому. Запрос фиксируется до окончания обслуживания прерывания, а затем он автоматически сбрасывается. Задержанные прерывания по перепаду могут также сбрасываться программно путем установки соответствующего бита в одном из регистров управления прерываниями.

Прерывания по перепаду требуют, как правило, меньших аппаратных затрат, чем прерывания по уровню, что позволяет использовать в качестве запросов прерывания такие сигналы, как синхроимпульсы (например,

импульсы, определяющие частоту дискретизации аналого-цифровых преобразователей).

Прерывания по уровню не фиксируются процессором и должны оставаться подтвержденными до окончания обслуживания. При этом действие запроса прекращается устройством, от которого запрос поступил. Если этого не произошло, то «не сброшенное» прерывание будет обработано повторно. У прерываний по уровню есть особенность, заключающаяся в том, что за счет логического комбинирования одна и та же запросная линия может быть использована несколькими источниками прерывания.

Возможность использования вложенных прерываний задается программным путем. В режиме, не разрешающем использование вложенных прерываний, все запросы на прерывания автоматически маскируются и запрещаются при входе в процедуру обработки прерывания. В режиме, разрешающем использование вложенных прерываний действует система приоритетов.

Когда вложенные прерывания заблокированы, то при входе в процедуру обработки прерывания всех уровней автоматически маскируются. Если вложенность прерываний разрешена, то регистр маски устанавливается таким образом, что маскируются только прерывания с равным или более низким приоритетом. Конфигурация прерываний с более высокими приоритетами сохраняется.

Если сигнал прерывания по перепаду возникает в то время, когда это прерывание замаскировано, то запрос фиксируется, но не обслуживается. Так возникает отложенное прерывание. Затем это прерывание может быть обработано. Любое из отложенных прерываний можно сбросить программными средствами с использованием соответствующих команд.

В процессорах ADSP 2171, ADSP 2181 предусмотрены глобальные команды разрешения и блокирования прерываний ENA INTS и DIS INTS,

10.1.3. Устройства обработки данных

На рис. 10.3 и 10.4 представлены блок-схемы двух устройств, входящих в ядро ЦПС ADSP 21xx – арифметическо-логического устройства (ALU) и умножителя-аккумулятора (MAC).

Входными для ALU являются два регистра AX (AX0 и AX1) и два регистра AY (AY0 и AY1). Результат выполняемой в ALU операции сохраняется в регистре результата AR (*ALU Result*) или в регистре временного хранения AF (регистре обратной связи – *ALU Feedback*). ALU формирует также признаки (флаги) нулевого (AZ) и отрицательного результата (AN), переполнения (AV) и переноса (AC). Имеется операция, связанная с определением знака операнда, принимаемого через порт X. Результатом этой операции является признак знака X-операнда. Вход CI ALU предназначен для приема бита переноса, что бывает необходимо в операциях двойной точности.

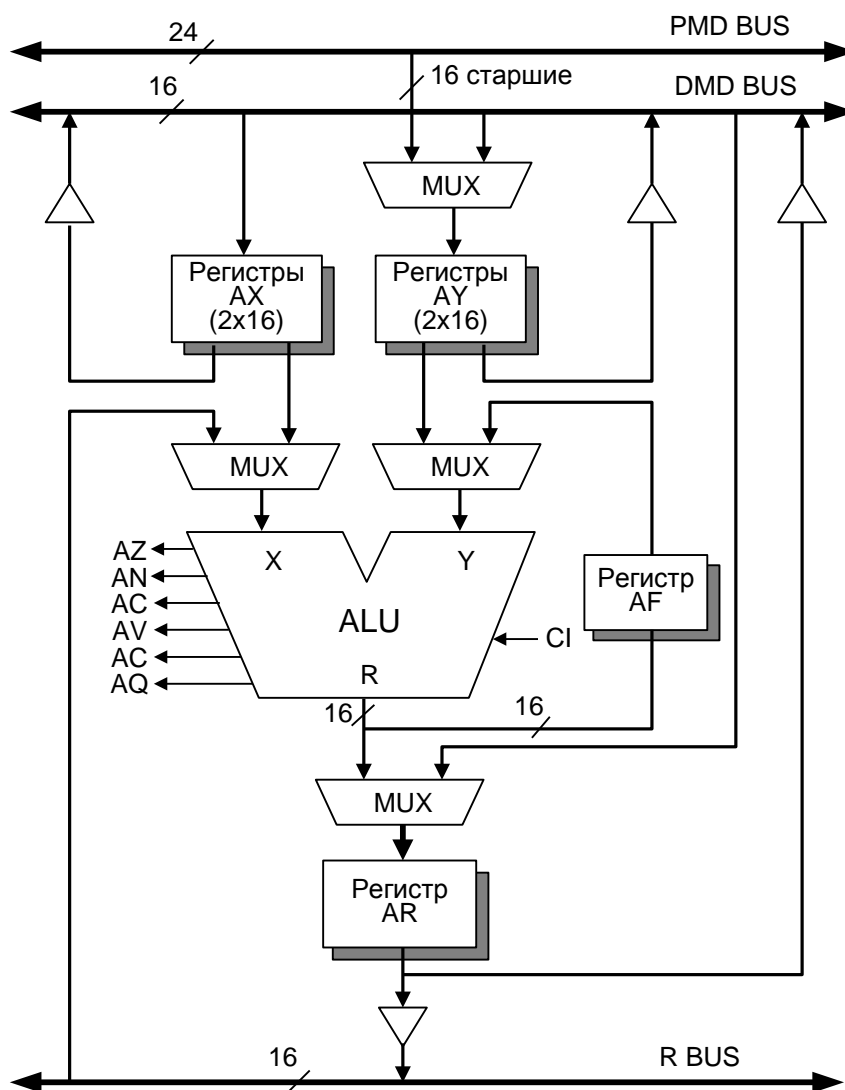


Рис. 10.3. Арифметическо-логическое устройство ЦПС ADSP 21xx.

В ALU выполняются также операции целочисленного деления. Деление происходит в пошаговом режиме со сдвигом делимого и каждый шаг вычисления сопровождается формированием бита частного, оформляемого как признак AQ.

Умножитель-аккумулятор также имеет два входных порта с регистрами MX (MX0 и MX1) и MY (MY0 и MY1). Регистр результата MR (MAC Result) составлен из двух 16-разрядных (MR0 и MR1) и одного 8-разрядного (MR2) регистров. Формируется один бит состояния – бит переполнения умножения MV. Имеется регистр временного хранения (регистр обратной связи) MF – MAC Feedback.

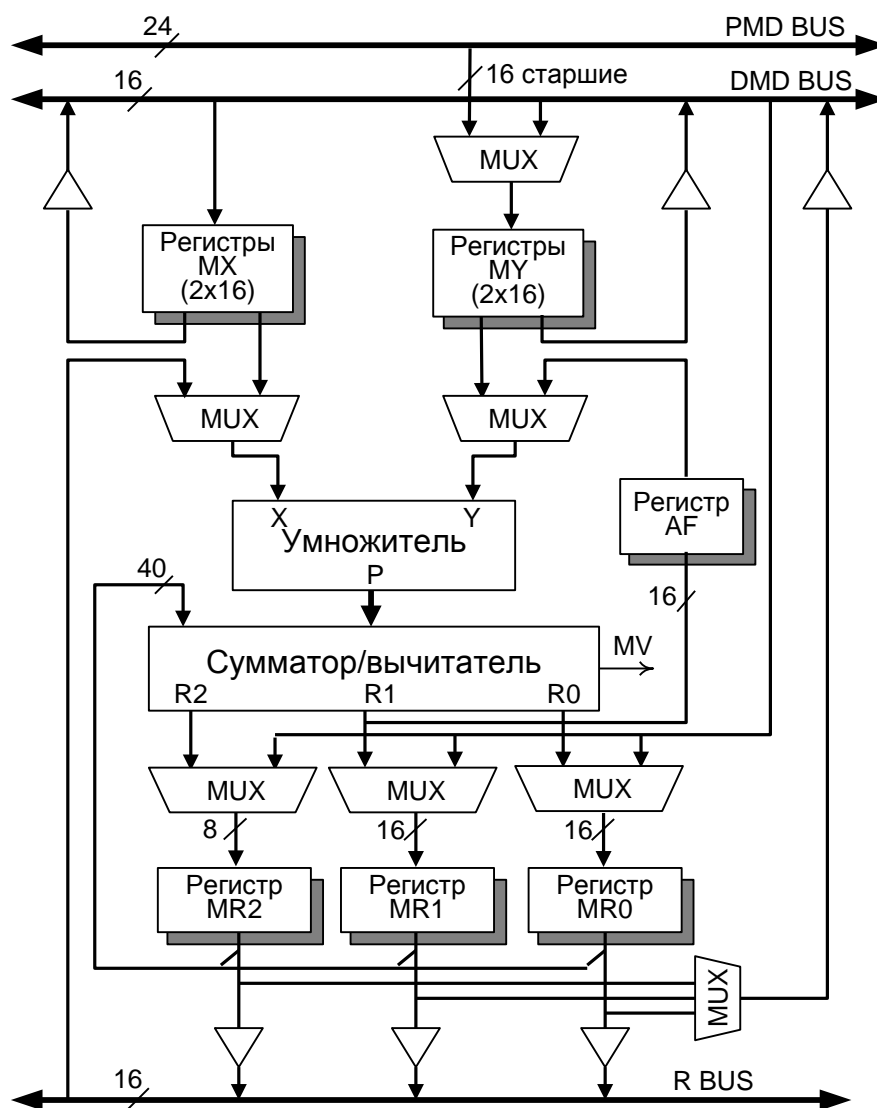


Рис. 10.4. Умножитель-аккумулятор ЦПС ADSP 21xx.

Устройство сдвига помимо стандартных сдвиговых операций выполняет операции нормализации и денормализации мантисс и операции с показателями чисел с плавающей точкой. Поскольку результат сдвиговых операций требует большего чем у операнда на входе числа разрядов, то регистр результата сдвигателя SR (Shifter Result) составлен из двух 16-разрядных регистров SR1 и SR2. Кроме SR имеются входной регистр SI (Shifter Input) и относящий к числу выходных регистр показателя SE (Shifter Exponent), необходимый при работе с числами с плавающей точкой.

Все регистры ALU, MAC и устройства сдвига доступны для чтения и записи через шину данных памяти данных DMD BUS. Через порты Y в ALU и MAC могут быть приняты значения шины данных памяти программ PMD BUS. Кроме этого, выходные регистры каждого из вычислительных устройств (регистры AR, MR и выходной регистр сдвигателя SR) могут быть операндами любого другого вычислительного устройства благодаря тому, что обмен информацией между ALU, MAC и устройством сдвига может осуществляться через шину результата R BUS.

Архитектурные решения компонент операционного блока МП ADSP 21xx мало отличаются от стандартных. Существенной особенностью является то, что в ЦПС семейства ADSP 21xx, как и во всех последующих, предусмотрен дополнительный (альтернативный, вторичный) набор регистров, которые в нужное время могут заменить основные регистры. К ним относятся входные и выходные регистры AX, AY, AR и AF арифметическо-логического устройства, регистры MX, MY, MR и MF умножителя-аккумулятора, а также входной SI и выходные SR и SE регистры устройства сдвига. На рис. 10.3 и 10.4 регистры, при которых есть альтернативные регистры, отмечены оттененными прямоугольниками.

Необходимость в подмене основных регистров альтернативными возникает при вызовах процедур, особенно при обработке запросов прерываний. Переключение с одного набора регистров на другой происходит под управлением специальных команд

ENA SEC_REG – разрешить вторичные регистры и
DIS SEC_REG – запретить вторичные регистры.

Вызываемое этими командами переключение с одного набора регистров на другой происходит без затрат времени на сохранение использовавшихся до вызова регистров. Задержка переключения равна длительности одного такта процессора. Действия, связанные с сохранением/восстановлением регистров при вызовах и возвратах называют *сохранением/восстановлением контекста*, или *контекстным переключением*. Время контекстного переключения является одной из важнейших характеристик систем реального времени.

Механизм поддержки реального времени, основанный на использовании альтернативных регистров, имеет ограничения, обусловленные тем, что допускает только один уровень вложения процедур с контекстным переключением. Поэтому его целесообразно применять лишь при многократно повторяющихся вызовах одной и той же (желательно короткой) процедуры, предпочтительно процедуры обработки прерывания. Примером может послужить обработка прерываний при вводе данных от аналого-цифрового преобразователя (АЦП), когда запросы прерывания идут с частотой дискретизации, а сама процедура обработки состоит из небольшого числа команд, предназначенных для ввода данных из регистра данных АЦП и сохранения их в памяти. Поскольку процедура работает с альтернативными регистрами, в нее дополнительно необходимо включать исполнение команд ENA SEC_REG при входе в процедуру и DIS SEC_REG перед выходом из нее.

Здесь уместно напомнить о механизме регистровых окон в процессорах с регистрово-ориентированной архитектурой, используемом при аппаратных и программных вызовах процедур: операции по сохранению регистров (контекста) процессора заменяются предоставлением вызываемой процедуре отдельного регистрового окна (см. раздел 8.2). Регистры МП ADSP 21xx вместе с альтернативными регистрами можно рассматривать «двухоконный» набор регистров. Отличие от действительно двухоконного набора состоит в том, что

регистры процессоров ADSP 21xx не оформлены как отдельный регистровый блок, а распределены по компонентам операционного блока – находятся в ALU, в MAC и в устройстве сдвига.

10.1.4. Адресация памяти данных

В каждом процессоре семейства ADSP 21xx имеются два генератора адресов данных DAG1 и DAG2, реализующих косвенно-регистровую адресацию данных с автоматической модификацией адреса. В обоих генераторах предусмотрена возможность организации циклических буферов. Генератор DAG1 работает только с памятью данных, а DAG2 может генерировать адреса памяти программ и памяти данных, чем обеспечивается *одновременный доступ к данным*, расположенным в DM и в PM.

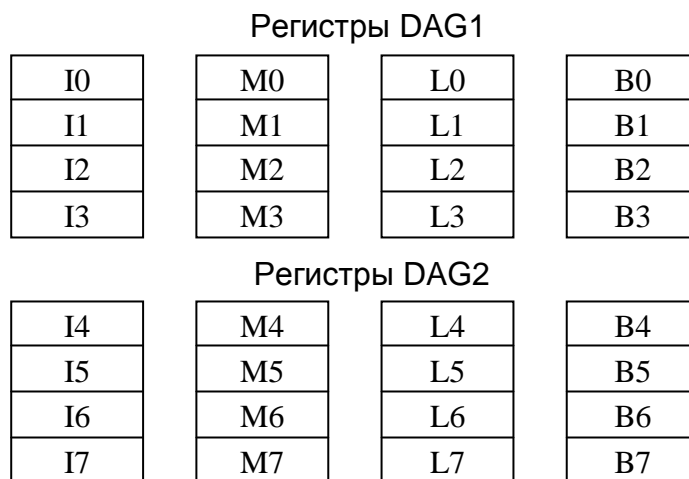


Рис. 10.5. Регистровая модель генераторов адресов данных

Регистровая модель генераторов адресов данных представлена на рис. 10.5. В DAG1 входят регистры базы B_n ($n = 0-3$), индексные регистры I_n регистры-модификаторы M_n и регистры L_n для работы с циклическими массивами. Точно такие же регистры B_n , I_n , M_n и L_n с номерами $n = 4-7$ имеются и в DAG2.

При работе с массивами базовый адрес массива указывается в регистре B_n . Индексные регистры I_n всегда работают в паре с соответствующим по номеру n регистром базы B_n , и в них содержится адрес текущего элемента массива, который при необходимости может автоматически наращиваться или уменьшаться для указания на следующий элемент. Предусмотрено автоматическое изменение содержимого индексных регистров с помощью регистров-модификаторов M_n . Регистры M_n не имеют строгой привязки к индексным регистрам. В них содержится лишь набор модификаторов, значения и порядок использования которых зависят от размеров элементов массива, а также от шага и направления движения по массиву. Но при этом регистры-

модификаторы одного генератора адресов данных не могут быть использованы в другом

Обычные и циклические массивы различаются по значениям регистров L_n . Ненулевое значение регистра L_n определяет длину организованных как очередь массивов. Для индексации элементов циклического массива используются регистры L_n с номером n , совпадающим с номером регистра базы V_n . Если содержимое какого-либо из регистров L_n равно нулю, то это означает, что соответствующий массив не является циклическим. Следствием этого является требование обязательной инициализации регистров L_n до вступления генератора адреса данных в работу

Вычислительные устройства и устройство сдвига процессоров ADSP 21xx работают только с операндами, расположенными в регистрах. Обращение к памяти поддерживается отдельным набором инструкций, которые служат для загрузки и сохранения регистров в памяти. Для примера можно привести следующие инструкции:

$AX0 = DM(I3, M0)$ – загрузка входного регистра ALU $AX0$ содержимым ячейки памяти данных DM , адрес которой указан в регистре $I3$, с последующей модификацией $I3$ по содержимому $M0$;

$MR1 = PM(I5, M5)$ – загрузка регистра $MR1$ умножителя-аккумулятора содержимым ячейки памяти программ PM , адрес которой указан в регистре $I5$, с последующей модификацией $I5$ по содержимому $M5$;

$PM(I4, M4) = AR$ – сохранение регистра AR ALU в ячейке памяти программ PM по адресу, указанному в регистре $I1$, с последующей модификацией $I1$ по содержимому $M3$;

$DM(I1, M3) = MR0$ – сохранение регистра результата $MR0$ MAC в ячейке памяти данных DM , адрес которой указан в регистре $I4$, с последующей модификацией $I4$ по содержимому $M4$;

Если модификация адреса не требуется, то модификатор в команде все равно указывается, но его значение берется равным нулю. В командах Ассемблера L- и V-регистры не показываются и их значения определяются на стадии компиляции и компоновки программы.

Наряду с адресацией по указателю к памяти можно обращаться по адресу, указанному в команде – в режиме прямой адресации, например, так как это показано в командах

$DM(адрес) = AR$ – сохранить регистр в DM по указанному адресу,

$MX0 = DM(адрес)$ – загрузить регистр $MX0$ из ячейки DM с указанным адресом

10.1.5. Особенности программирования

Есть две формы написания инструкций на языке Ассемблера: мнемоническая и алгебраическая. Традиционно наиболее широко

распространена мнемоническая форма. Для написания инструкций цифровых процессоров, производимых фирмой Analog Devices, используется алгебраическая форма, которая ближе к языкам высокого уровня.

Все команды ALU, MAC и устройства сдвига МП ADSP 21xx выполняются по условию. Однако использование в команде кода условия не является обязательным и применяется лишь в случае необходимости. В качестве примера можно привести команду ALU сложения с переносом:

$$\text{IF AC AR} = \text{AX0} + \text{AY0} + \text{C}.$$

В этой команде присутствует необязательное условие IF AC. Поэтому в данном случае проверяется на наличие бита переноса ALU (AC), и команда выполняется, если по результатам предыдущей команды был выставлен флаг переноса. В противном случае выполняется NOP и выполнение программы продолжается со следующей команды. Алгебраическое выражение $\text{AX0} + \text{AY0} + \text{C}$ означает, что в регистр результата ALU (AR) записывается сумма значений входных регистров плюс значение бита переноса.

В качестве другого примера возьмем команду для умножителя-аккумулятора

$$\text{IF NOT MV MR} = \text{MR} + \text{MX0} * \text{MY0}(\text{UU}).$$

По условию IF NOT MV в ней проверяется состояние бита переполнения умножителя. Если условие ложно, выполняется NOP. Выражение $\text{MR} = \text{MR} + \text{MX0} * \text{MY0}$ обозначает операцию умножения с накоплением: в регистр результата MR, образованный двумя 16-разрядными регистрами MR1 и MR0 и одним 8-разрядным регистром MR2, записывается сумма его текущего значения и произведения операндов X и Y из выбранных регистров ввода. Модификатор в скобках (UU) означает, что операнды являются беззнаковыми величинами. Наряду с UU применяются другие модификаторы:

- модификатор SS для операции со знаковыми операндами;
- модификаторы SU и US для случаев, когда один из операндов представляет число со знаком, а другой – число без знака.

Имеется еще один модификатор, означающий округление (подразумевается знакового) результата.

$$[\text{IF условие}] \begin{array}{l} \text{AR} \\ \text{AF} \end{array} = \text{хор} \begin{array}{l} + \text{уор} \\ + \text{C} \\ + \text{уор} + \text{C} \\ + \text{const} \\ + \text{const} + \text{C} \end{array};$$

$$[\text{IF условие}] \begin{array}{l} \text{AR} \\ \text{AF} \end{array} = \text{хор} \begin{array}{l} - \text{уор} \\ - \text{уор} + \text{C} - 1 \\ + \text{C} - 1 \\ - \text{const} \\ - \text{const} + \text{C} - 1 \end{array};$$

$$[\text{IF условие}] \begin{array}{l} \text{AR} \\ \text{AF} \end{array} = \text{хор} \begin{array}{l} \text{AND} \\ \text{OR} \\ \text{XOR} \end{array} \begin{array}{l} \text{уор} \\ \text{const} \end{array};$$

DIVS *уор, хор*; {Определить знак результата}
 DIVQ *хор*; {Определить разряд в частном}

Рис. 10.6. Команды ALU МП ADSP 21xx.

$$[\text{IF условие}] \begin{array}{l} \text{MR} \\ \text{MF} \end{array} = \text{хор} * \begin{array}{l} \text{уор} \\ \text{хор} \end{array} \left(\begin{array}{l} \text{SS} \\ \text{SU} \\ \text{US} \\ \text{UU} \\ \text{RND} \end{array} \right);$$

$$[\text{IF условие}] \begin{array}{l} \text{MR} \\ \text{MF} \end{array} = \text{MR} + \text{хор} * \begin{array}{l} \text{уор} \\ \text{хор} \end{array} \left(\begin{array}{l} \text{SS} \\ \text{SU} \\ \text{US} \\ \text{UU} \\ \text{RND} \end{array} \right);$$

$$[\text{IF условие}] \begin{array}{l} \text{MR} \\ \text{MF} \end{array} = \text{MR} - \text{хор} * \begin{array}{l} \text{уор} \\ \text{хор} \end{array} \left(\begin{array}{l} \text{SS} \\ \text{SU} \\ \text{US} \\ \text{UU} \\ \text{RND} \end{array} \right);$$

Рис. 10.7. Команды MAC МП ADSP 21xx.

На рис. 10.6 и 10.7 представлены команды ALU и MAC. Они не составляют полного перечня исполняемых этими операционными блоками команд и выбраны для того, чтобы проиллюстрировать правило, по которому строятся команды для МП ADSP 21xx.

В списке команд *условие* используется для обозначения всех возможных условий, которые могут быть проверены, а аббревиатуры *хор* и *уор* – для

обозначения регистров-операндов по X и Y вводам ALU и MAC. X-операндами каждого из операционных блоков могут быть свои входные X-регистры и регистры результата AR и MR вычислительных устройств и регистр результата устройства сдвига SR. Y-операндами являются свои входные Y-регистры и только свои регистры результата и регистры обратной связи AF и MF.

На рисунках

- квадратные скобки обозначают дополнительную (необязательную) часть команды;
- между параллельными вертикальными линиями помещается список операндов. Выбирается один из перечисленных в списке операндов;
- заглавными буквами обозначены команды и имена регистров;
- аббревиатура *const* обозначает непосредственные операнды;
- аббревиатура *dreg* (см. ниже) относится к любому из регистров данных (к регистрам ALU, MAC и устройства сдвига).

ALU выполняет также команды деления. Деление начинается с определения знака результата (команда *DIVS хор, уор*), а затем в пошаговом режиме путем исполнения команды *DIVQ хор* последовательно, начиная со старшего определяются биты результата. Число шагов равно числу значащих бит результата. Команда *DIVQ* имеет один операнд (делитель), поскольку второй операнд (делимое) до начала процедуры деления должен находиться в регистрах AY и AF – в двух регистрах потому, что делимое имеет вдвое большее число разрядов, чем делитель.

Здесь не представлены команды устройства сдвига, т. к. с точки зрения построения команд их рассмотрение не дает существенно нового.

Помимо простых команд, относящихся только к выполнению операций с данными, имеются многофункциональные команды, позволяющие эффективно использовать преимущества гарвардской архитектуры – одновременное выполнение за один цикл пересылок данных, операций записи/считывания в/из память(и) и вычислений. На рис. 10.8 приведен список многофункциональных команд. Команды внутри многофункциональной команды отделяются запятой, комбинированная многофункциональная команда оканчивается точкой с запятой.

Операции ALU и MAC (рис. 10.8a) могут выполняться одновременно с загрузкой регистров AX, AY или MX, MY из памяти данных DM и памяти программ PM.

Адреса ячеек DM определяются генератором адресов данных DAG1 с привлечением в качестве указателей индексных регистров I0-I3¹⁷ с регистрами-модификаторами M0-M3, а адреса PM – генератором DAG2 с индексными регистрами I4-I7 и модификаторами M4-M7. Каждый из регистров-модификаторов задает приращение (уменьшение) значения следующего адреса по отношению к текущему, содержащемуся в индексном регистре, с которым

¹⁷ Имена регистров I0-I3, M0-M3, I4-I7 и M4-M7 показываются в командах.

соответствующий модификатор работает. Модификатор в команде должен быть обязательно указан. Если адрес не модифицируется, то используется регистр-модификатор с нулевым содержимым.

$$\left| \begin{array}{l} \langle \text{ALU} \rangle \\ \langle \text{MAC} \rangle \end{array} \right|, \left| \begin{array}{l} \text{AX0} \\ \text{AX1} \\ \text{MX0} \\ \text{MX1} \end{array} \right| = \text{DM} \left(\left| \begin{array}{l} \text{I0} \\ \text{I1} \\ \text{I2} \\ \text{I3} \end{array} \right|, \left| \begin{array}{l} \text{M0} \\ \text{M1} \\ \text{M2} \\ \text{M3} \end{array} \right| \right), \left| \begin{array}{l} \text{AY0} \\ \text{AY1} \\ \text{MY0} \\ \text{MY1} \end{array} \right| = \text{PM} \left(\left| \begin{array}{l} \text{I4} \\ \text{I5} \\ \text{I6} \\ \text{I7} \end{array} \right|, \left| \begin{array}{l} \text{M4} \\ \text{M5} \\ \text{M6} \\ \text{M7} \end{array} \right| \right); \quad \text{а)}$$

$$\left| \begin{array}{l} \text{AX0} \\ \text{AX1} \\ \text{MX0} \\ \text{MX1} \end{array} \right| = \text{DM} \left(\left| \begin{array}{l} \text{I0} \\ \text{I1} \\ \text{I2} \\ \text{I3} \end{array} \right|, \left| \begin{array}{l} \text{M0} \\ \text{M1} \\ \text{M2} \\ \text{M3} \end{array} \right| \right), \left| \begin{array}{l} \text{AY0} \\ \text{AY1} \\ \text{MY0} \\ \text{MY1} \end{array} \right| = \text{PM} \left(\left| \begin{array}{l} \text{I4} \\ \text{I5} \\ \text{I6} \\ \text{I7} \end{array} \right|, \left| \begin{array}{l} \text{M4} \\ \text{M5} \\ \text{M6} \\ \text{M7} \end{array} \right| \right); \quad \text{б)}$$

$$\left| \begin{array}{l} \text{DM} \left(\left| \begin{array}{l} \text{I0} \\ \text{I1} \\ \text{I2} \\ \text{I3} \end{array} \right|, \left| \begin{array}{l} \text{M0} \\ \text{M1} \\ \text{M2} \\ \text{M3} \end{array} \right| \right) \\ \\ \\ \\ \text{PM} \left(\left| \begin{array}{l} \text{I4} \\ \text{I5} \\ \text{I6} \\ \text{I7} \end{array} \right|, \left| \begin{array}{l} \text{M4} \\ \text{M5} \\ \text{M6} \\ \text{M7} \end{array} \right| \right) \end{array} \right| = \text{dreg}, \left| \begin{array}{l} \langle \text{ALU} \rangle \\ \langle \text{MAC} \rangle \\ \langle \text{Shifter} \rangle \end{array} \right|; \quad \text{в)}$$

$$\left| \begin{array}{l} \langle \text{ALU} \rangle \\ \langle \text{MAC} \rangle \\ \langle \text{Shifter} \rangle \end{array} \right|, \text{dreg} = \text{dreg}; \quad \text{г)}$$

$$\left| \begin{array}{l} \langle \text{ALU} \rangle \\ \langle \text{MAC} \rangle \\ \langle \text{Shifter} \rangle \end{array} \right|, \text{dreg} = \left| \begin{array}{l} \text{DM} \left(\left| \begin{array}{l} \text{I0} \\ \text{I1} \\ \text{I2} \\ \text{I3} \end{array} \right|, \left| \begin{array}{l} \text{M0} \\ \text{M1} \\ \text{M2} \\ \text{M3} \end{array} \right| \right) \\ \\ \left| \begin{array}{l} \text{I4} \\ \text{I5} \\ \text{I6} \\ \text{I7} \end{array} \right|, \left| \begin{array}{l} \text{M4} \\ \text{M5} \\ \text{M6} \\ \text{M7} \end{array} \right| \\ \\ \text{PM} \left(\left| \begin{array}{l} \text{I4} \\ \text{I5} \\ \text{I6} \\ \text{I7} \end{array} \right|, \left| \begin{array}{l} \text{M4} \\ \text{M5} \\ \text{M6} \\ \text{M7} \end{array} \right| \right) \end{array} \right|; \quad \text{д)}$$

Рис. 10.8. Многофункциональные команды МП ADSP 21xx.

Частным случаем являются команды, связанные только с обращением к памяти (рис. 10.8б).

Операции устройства сдвига, как и операции ALU и MAC, могут выполняться одновременно с загрузкой/сохранением значений регистров данных либо в памяти программ PM, либо в памяти данных DM – рис. 10.8г, д.

Пересылки данных между регистрами данных также могут выполняться одновременно с операциями ALU, MAC и устройства сдвига (рис. 10.8д).

10.2. Средства повышения производительности ядра процессора и многопроцессорные системы

Цифровые процессоры сигналов развиваются в направлении увеличения производительности и расширения функциональных возможностей. Это отражается в устройствах обработки данных, в организации и объеме памяти, в работе с устройствами ввода-вывода. Важное значение имеет то, насколько эффективно используются вычислительные ресурсы процессора. В значительной степени повышение этой эффективности достигается за счет конвейера операций, при котором действия процессора распределяются на отдельные фазы (циклы), связанные

- с определением адреса следующей выбираемой из памяти команды,
- с циклом считывания команды из памяти,
- с декодированием команды и выработкой условий и сигналов, управляющих выполнением команды и, наконец,
- с выполнением команды.

При последовательном выполнении программы в то время, когда одна команда выбирается, команда выбранная несколькими (в представленном перечне тремя) циклами ранее, выполняется. Кроме этого, отдельные этапы конвейера могут быть разбиты на дополнительные подэтапы. Однако нужно заметить, что повышение числа уровней конвейера влечет за собой сложности в управлении, обусловленные, прежде всего, возникающими по ходу исполнения программы ветвлениями и переходами. Наличие последних заставляет прибегать к механизму опережающей выборки команд и к предсказанию ветвлений. В противном случае эффективность конвейера операций значительно снижается из-за необходимости его перезагрузки при поступлении команд переходов.

Производительность процессора во многом зависит от эффективности использования регистров процессора. Доступность регистров со стороны различных вычислительных устройств ядра процессора дает возможность параллельного исполнения ими вычислительных. Отсюда следует, что регистровый блок процессора должен быть многопортовым.

Конвейер операций и эффективное использование регистровой памяти – это одна из ключевых особенностей RISC процессорам. Поэтому RISC архитектурные решения рассматриваются как наиболее рациональные, и их

широко используют производители как простых микроконтроллеров, так и сложных однокристалльных вычислительных систем

10.2.1. Ядро ЦПС семейства ADSP 2106x

Тенденцию развития микропроцессорных систем проследим, продолжив рассмотрение цифровых процессоров сигналов фирмы Analog Devices.

На рис. 10.9 представлена структурная схема ядра ЦПС семейства ADSP 2106x с архитектурой SHARC. В состав ядра ADSP 2106x входят те же функциональные блоки, что и в процессорах семейства ADSP 21xx: программный автомат, арифметическо-логическое устройство ALU, умножитель-аккумулятор MAC, устройство сдвига Shifter и два генератора адресов данных DAG1,2. Но повышена разрядность регистров и шин PMA, DMA, PMD и DMD.

Шина адреса памяти программы PMA имеет 24-разряда и, как следствие, 24-разрядными являются регистры генератора адресов данных DAG2. Число разрядов в шине данных памяти программы PMD, как и в ADSP 21xx, обусловлено разрядностью команд процессора – все команды МП ADSP 2106x являются 48-разрядными. Это позволило расширить возможности многофункциональных команд – все они стали условными, за исключением команд, управляющих выполнением операций с регистрами при одновременном доступе к памяти программы и к памяти данных.

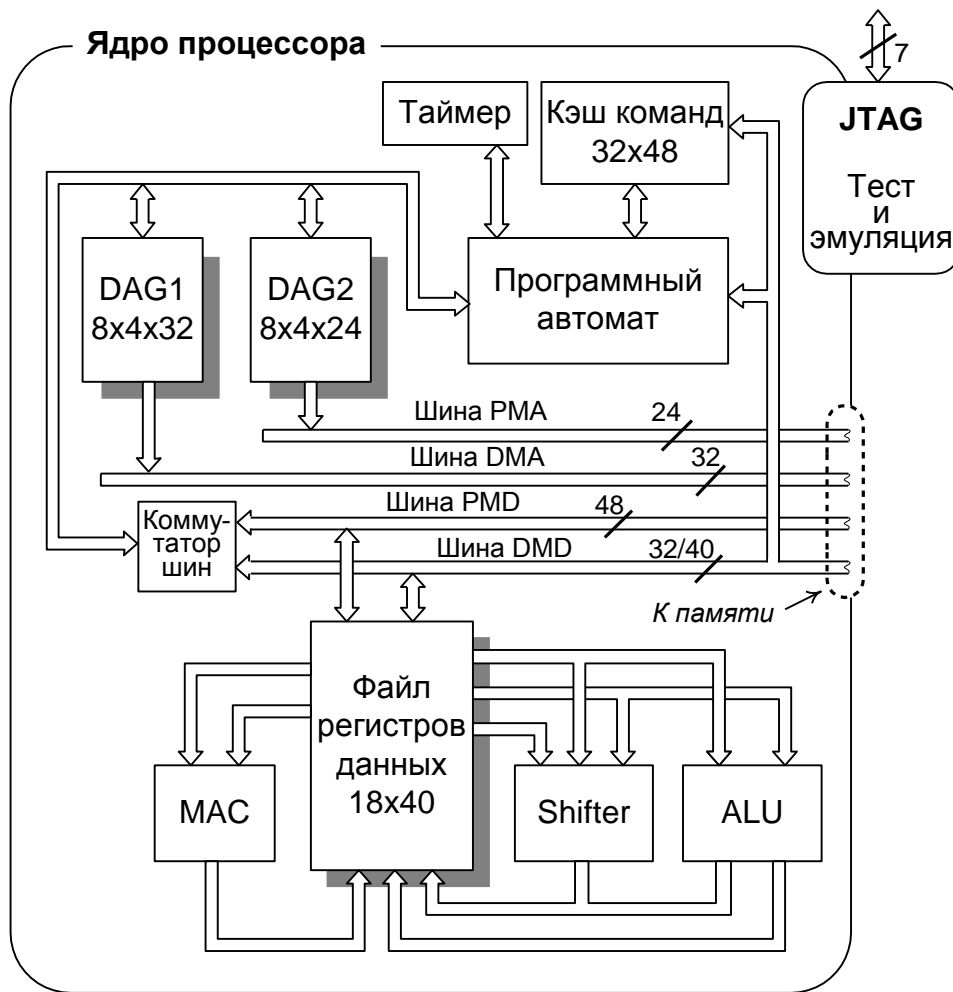


Рис. 10.9. Структура ядра ЦПС семейства ADSP 2106x

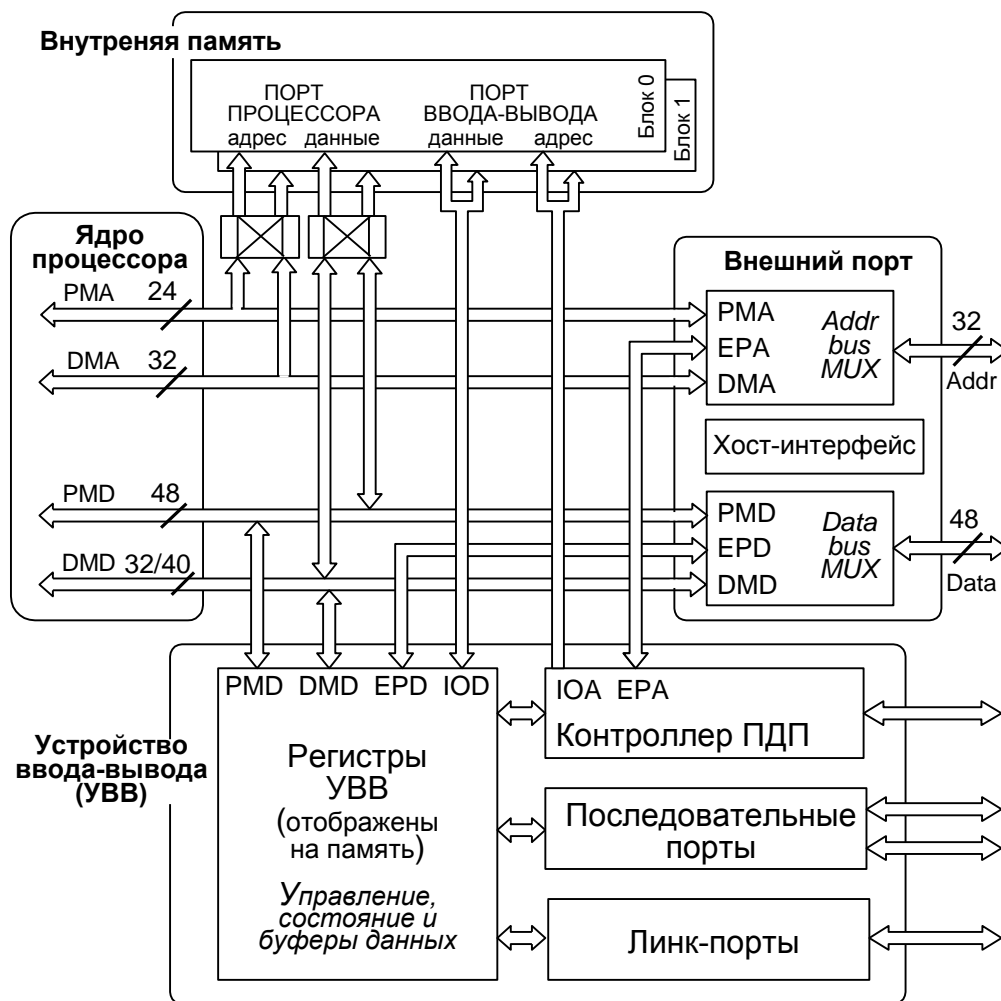


Рис. 10.10. Структура процессора ADSP 2106x

Расширено адресное пространство памяти данных, обращение к которой происходит по 32-разрядной шине адреса DMA и 40-разрядной шине данных DMD.

Операнды и промежуточные результаты работы MAC, ALU и устройства сдвига хранятся и запоминаются в 10-портовом файле регистров данных. В каждом цикле синхронизации регистровый файл может поддерживать следующие операции: (1) запись и считывание из регистрового файла двух операндов, (2) выдача двух операндов на входы ALU, (3) выдачу двух операндов на входы MAC и (4) сохранение результатов операций ALU и MAC.

Процессор ADSP 2106x обрабатывает команду за три цикла:

1. В цикле *выборки* команда считывается из кэша команд или из памяти программы.
2. В цикле *декодирования* производится декодирование команды определяются условия, необходимые для управления выполнением команды.
3. В цикле *выполнения* завершаются предписанные командой действия.

Эти циклы перекрываются и образуют конвейер. При последовательном выполнении программы, в то время, когда одна команда выбирается, команда,

выбранная в предыдущем цикле, декодируется, а команда, выбранная двумя циклами ранее, выполняется.

Для быстрого переключения контекста, например, при обработке прерываний, регистровый файл имеет два набора регистров (первичный и вторичный), каждый из которых включает 16 40-разрядных регистров.

Два набора регистров имеют также генераторы адресов данных. Регистры, активные после сброса процессора, относятся к первичным, а другие – к дополнительным (или вторичным). Какой из наборов регистров является активным в определенный момент времени, определяют соответствующие биты в регистре управления режимом работы процессора.

У процессоров ADSP 2106x *нет физического разделения памяти на память программ и память данных*. В этом состоит особенность гарвардской архитектуры ADSP 2106x. Внутренняя память процессора делится на блоки (рис. 10.10), в которых могут быть размещены и программа, и данные. Внешняя память разбита на банки. Место нахождения программы и данных определяется тем, как память сконфигурирована.

Обычно программный автомат выбирает команду из памяти в каждом машинном цикле. Иногда возникают ситуации, в которых одновременно (в одном цикле) иницируются обращения по шине PMD к данным и к программе. Если команда выбирается из памяти программ в то время, когда выполняется команда пересылки данных по шине данных PMD, может возникнуть конфликтная ситуация. Для устранения подобных конфликтных ситуаций процессор кэширует команды.

Когда возникают проблемы с выборкой какой-то команды, процессор записывает эту команду в кэш для того, чтобы в будущем исключить возникновение конфликта. Программный автомат проверяет кэш команд при каждом обращении по шине данных PMD. Если необходимая команда находится в кэше, то для ее выборки не требуется обращения к памяти программ и команда из кэша выбирается параллельно с обращением к данным памяти программы и без задержки. Если необходимой команды в кэше нет, то команда выбирается из памяти в цикле, следующем за обращением к данным памяти программы. Таким образом добавляется один непроизводительный цикл. Команда загружается в кэш, и если кэш разблокирован, то она будет доступна при повторных возникновениях подобного рода конфликтов.

Если при выполнении команды, находящейся по адресу n , делается обращение к данным по шине PMD и возникает конфликт при обращении блоку памяти, то этот конфликт связан с выборкой команды по адресу $n+3$ (предполагается последовательное выполнение программы), а не с исполняемой командой по адресу n . Это обусловлено опережающей выборкой и конвейерным выполнением команд. Как следствие, в кэше сохраняется команда с адресом $n+3$, а не та команда, для выполнения которой потребовалось обращение к данным памяти программы. При последовательном выполнении

программы это не сказывается на эффективности работы кэш-памяти. Однако при ветвлениях и переходах вероятность кэш-промахов увеличивается.

Режимом работы кэш-памяти можно управлять. За это отвечает соответствующий регистр процессора. В частности, использование кэша можно запретить или разрешить.

JTAG-порт поддерживает комплекс действий по тестированию соединений в системе, использующих методику последовательной проверки результатов ввода-вывода во всех компонентах системы в соответствии с предложенным IEEE стандартом.

10.2.2. Подсистема ввода-вывода ЦПС семейства ADSP 2106x

Процессоры ADSP 2106x содержат внутреннюю статическую оперативную память в виде двух двухпортовых блока, которые могут конфигурироваться для различных комбинаций хранения кода и данных. В одном цикле к каждому блоку памяти могут независимо обращаться ядро процессора (с использованием шин PMA, PMD и DMA, DMD) и контроллер DMA в устройстве ввода-вывода (с использованием шин IOA – *Input-Output Address* и IOD – *Input-Output Data*). Используя шины DMD и PMD, ядро процессора может обращаться к внешней или внутренней памяти в одном и том же машинном цикле.

Имеется система арбитража шин для обработки конфликтующих обращений. Приоритеты арбитража фиксированы и установлены в следующем порядке (по убыванию): обращения по шине DMD, обращения по шине PMD, обращения со стороны устройства ввода-вывода (УВВ) с использованием шины IOD. Кроме того, время обращения УВВ (которое в пакетном режиме может быть достаточно большим) не может выходить за устанавливаемые временные пределы, поэтому предоставление шины ядру процессора никогда не задерживаются более чем на 4 цикла.

В состав устройства ввода-вывода входят

- порт стандартного последовательного интерфейса с периферийными устройствами SPI – *Serial Peripheral Interface*,
- порт универсального асинхронного приемопередатчика UART – *Universal Asynchronous Receiver/Transmitter*,
- шесть 4-разрядных портов связи – линк-портов для организации многопроцессорных систем и
- контроллер прямого доступа к памяти – контроллер ПДП.

Обмен данными с периферийными устройствами (ПУ) происходит преимущественно в режиме прямого доступа к памяти под управлением контроллера ПДП, что позволяет более эффективно использовать ресурсы ядра при выполнении операций по обработке поступающих от периферийных устройств данных. Для конфигурации и управления периферийными

устройствами, для промежуточного хранения передаваемых и принимаемых от ПУ данных, а также для конфигурации и управления работой контроллера ПДП служат отображенные на память *регистры УВВ*. Обращение к регистрам УВВ происходит с использованием шин DMA и DMD. По этим же шинам при необходимости ядро процессора может обращаться к входящим в состав устройства ввода-вывода регистрам данных.

Интерфейс с внешней памятью и внешними периферийными устройствами обеспечивает *внешний порт* с 32-разрядной шиной адреса и 48-разрядной шиной данных, которые образованы путем мультиплексирования внутренних шин адреса (PMA, DMA и EPA – *External Port Address*) и данных (PMD, DMD и EPD – *External Port Data*). Кроме этого через внешний порт осуществляется связь с хост-машиной (с главной машиной).

10.2.3. Ресурсы и архитектура многопроцессорной системы на базе ADSP 2106x

Процессоры семейства ADSP 2106x имеют функциональные характеристики, которые позволяют создавать многопроцессорные системы. Используются две схемы связи между процессорами. По одной схеме с помощью портов связи (линк-портов) реализуются соединения «точка-точка». По другой процессоры совместно используют глобальную память системы и связь между процессорами осуществляется по общей шине, для чего имеются встроенный арбитраж шины и возможность обращения к внутренней памяти и регистрам устройств ввода-вывода других ADSP 2106x. Число объединённых общей шиной процессоров не должно превышать шести.

Процессоры ADSP 2106x обращаются к внешней памяти и портам ввода-вывода через внешний порт. Внешнее адресное пространство включает *пространство памяти многопроцессорной системы* (память на кристаллах других ADSP 2106x), а также *пространство внешней памяти* (область памяти, расположенной вне кристаллов).

Ядро процессора и устройство ввода-вывода имеют доступ к внешним шинам ($DATA_{47-0}$, $ADDR_{31-0}$) через внешний порт ADSP 2106x (рис. 10.10, 10.11). Внешний порт обеспечивает доступ к памяти, размещённой вне кристалла, и к периферийным устройствам. Через него можно обращаться к внутренней памяти других ADSP 2106x, соединённых в многопроцессорную систему. Схема соединения с общей шиной позволяет реализовывать одно объединённое адресное пространство, в котором могут храниться и код, и данные.

Внешняя память может быть 16-, 32- или 48-разрядная; контроллер прямого доступа в память автоматически упаковывает внешние данные в слова соответствующей разрядности: 48-разрядные команды или 32-разрядные данные.

В многопроцессорной системе любой из процессоров может стать ведущим. Ведущий процессор берёт на себя управление шиной, которая включает линии шины данных $DATA_{47-0}$ и шины адреса $ADDR_{31-0}$, а также линии управления. Ведущий ADSP 2106x способен блокировать шину для выполнения неделимой последовательности операций *чтение-модификация-запись* для семафоров.

В таблице 10.1 перечислены и определены выводы микросхемы ADSP 2106x, используемые для интерфейса с внешней памятью. Сигналы управления памятью дают возможность соединения как с быстрыми (например, со статическими), так и с медленными устройствами памяти, а также с разными по быстродействию периферийными устройствами (адресное пространство ввода/вывода в процессорах ADSP 2106x отображено на память). Пользователь программными средствами может определять нужную комбинацию состояний ожидания и аппаратных сигналов подтверждения связи. Сигналы на выводах \overline{SBTS} (перевод шины в третье состояние) и $PAGE$ (граница страницы) могут использоваться для интерфейса с динамической памятью (DRAM).

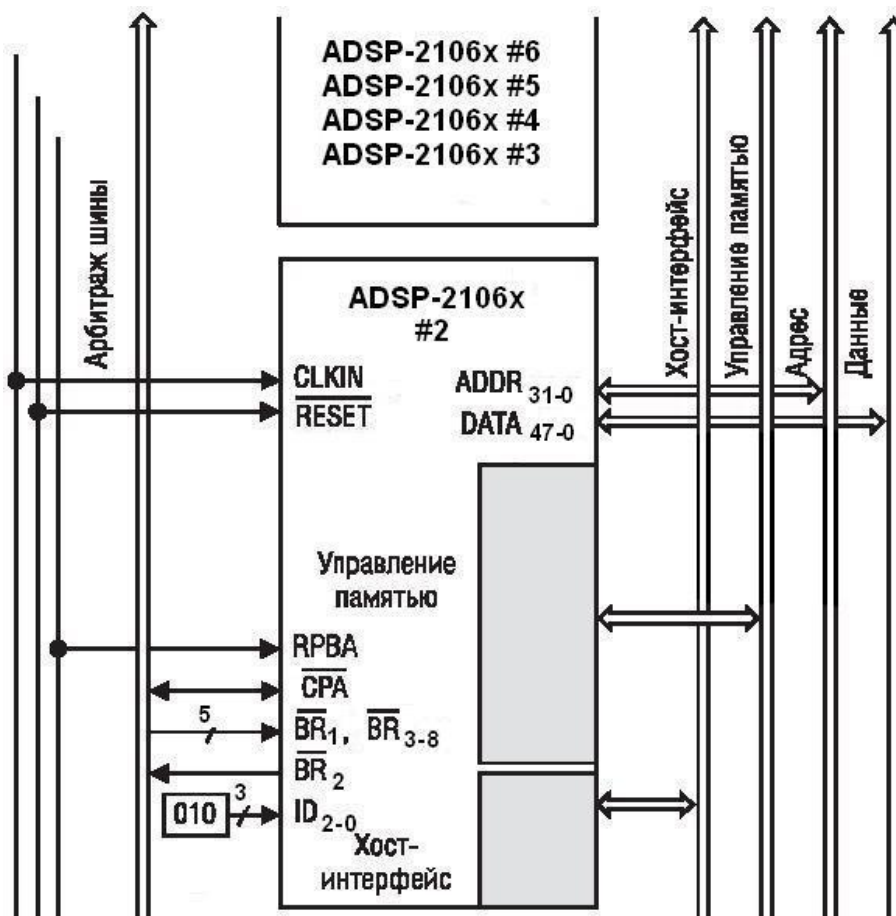


Рис. 10.11. Многопроцессорная система на базе ADSP 2106x.

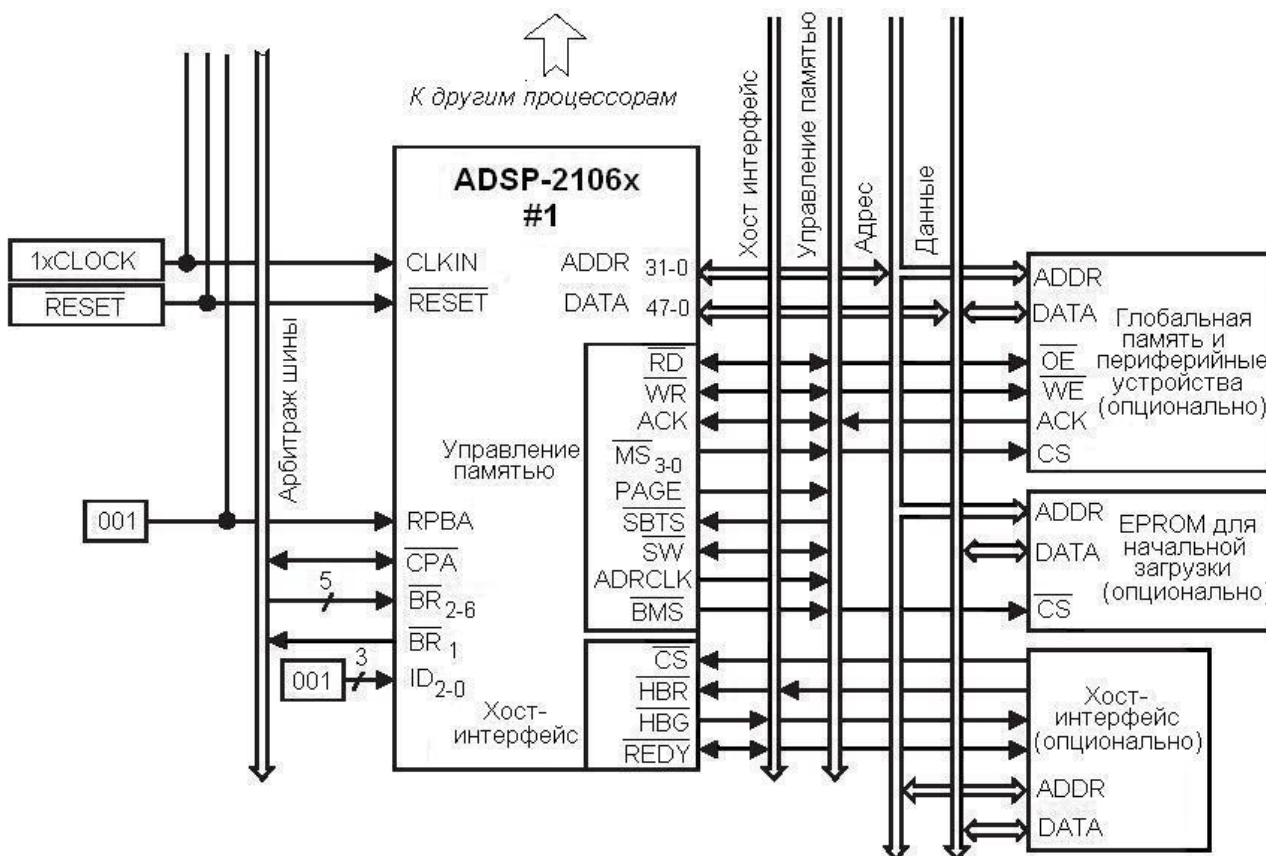


Рис. 10.11 (продолжение). Многопроцессорная система на базе ADSP 2106x

Таблица 10.1

Вывод	Тип	Функция
ADDR ₃₁₋₀	I/O/T	Внешняя шина адреса. По линиям внешней шины процессор выводит адреса внешней памяти и периферийных устройств. В многопроцессорной системе их активизирует ведущий процессор, обращаясь к внутренней памяти других ADSP 2106x или к регистрам УВВ. ADSP 2106x принимает адресные сигналы от хост-процессора или другого ведущего в многопроцессорной системе, когда производится считывание или запись в его внутреннюю память или в регистры УВВ.
DATA ₄₇₋₀	I/O/T	Внешняя шина данных. По линиям шины данных процессор получает и выводит данные и команды. 32-разрядные данные с фиксированной точкой и 32-разрядные данные с плавающей точкой передаются битами 47-16, 40-разрядные данные с плавающей точкой повышенной точности – битами 47-8, 16-разрядные короткие слова данных – битами 31-16.
\overline{MS}_{3-0}	O/T	Линии выбора памяти. Сигнал низкого уровня на этих линиях используется для выбора кристалла соответствующего банка внешней памяти. Размер банка памяти должен быть определен в регистре управления системой. В многопроцессорной системе сигналы на линиях выводятся ведущим на системной магистрали.

\overline{RD}	I/O/T	Строб чтения памяти. Этот сигнал сопровождает считывание из внешней памяти или из внутренней памяти других ADSP 2106x. Сигнал \overline{RD} выставляют также внешние устройства (включая другие ADSP 2106x) для считывания данных из внутренней памяти ADSP 2106x. В многопроцессорных системах сигнал \overline{RD} выводится ведущим и принимается всеми другими процессорами.
\overline{WR}	I/O/T	Строб записи в память. Активизируется, когда процессор выполняет запись во внешнюю или во внутреннюю память других ADSP 2106x. Внешние устройства (включая другие ADSP 2106x) должны выставлять строб \overline{WR} для записи данных во внутреннюю память ADSP 2106x. В многопроцессорных системах сигнал \overline{WR} выводится ведущим и принимается всеми другими ADSP 2106x.
PAGE	O/T	Граница страницы DRAM. Сигнал активизируется при пересечении границы страницы внешней динамической памяти (DRAM). Размер страницы DRAM определяется в регистре управления временем доступа к памяти WAIT. DRAM может быть размещена только в банке 0 внешней памяти; сигнал PAGE может быть использован только для обращений к банку 0. В многопроцессорных системах PAGE выводится ведущим.
\overline{SW}	I/O/T	Выбор синхронной записи (<i>Synchronous Write Select</i>). Этот сигнал используется для синхронного взаимодействия ADSP 2106x с устройствами памяти (включая внутреннюю память других ADSP 2106x). Процессор использует \overline{SW} для предварительного указания на то, что предполагается выполнение операции записи, которая может и не произойти, если строб \overline{WR} не будет установлен (например, при выполнении условной команды записи). В многопроцессорной системе сигналом \overline{SW} управляет ведущий процессор, а все остальные его принимают. \overline{SW} выставляется тогда, когда выводится адрес. Хост-процессор, использующий синхронную запись, должен выставлять этот сигнал при записи в ADSP 2106x.
ACK	I/O/S	Подтверждение доступа к памяти (<i>Memory Acknowledge</i>). ACK используется устройствами ввода/вывода, контроллерами памяти, другими периферийными устройствами, чтобы путём сброса этого сигнала продлить цикл обращения к внешней памяти. ADSP 2106x в режиме синхронной записи в его внутреннюю память может сбросить выходной сигнал подтверждения ACK, если с целью обеспечения синхронизма требуется продлить цикл обращения к его внутренней памяти.

Пояснения к таблице: I/O – вход/выход, S – синхронный, T – с тремя состояниями.

Карта памяти ADSP-2106x

Адресное пространство МП ADSP 2106x делится на три части: пространство внутренней памяти, пространство памяти многопроцессорной системы и пространство внешней памяти.

Внешняя память разделена на четыре равных банка (рис. 10.12). Для каждого из них устанавливается собственное время доступа, что позволяет отображать адреса портов медленных периферийных устройств в адресное пространство того банка, для которого установлены подходящие периферийному устройству способ синхронизации и время доступа.

Поля адресных кодов, генерируемых ADSP 2106x при обращении по шинам DM и PM, показаны на рис. 10.13. Адреса шины DM генерирует DAG1, а адреса шины PM – программный автомат (для команд) или DAG2 (для данных).

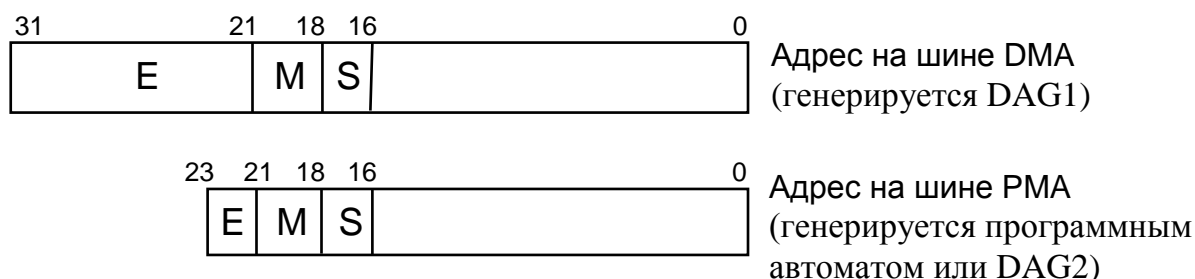


Рис. 10.13. Формат адресных кодов, передаваемых по шинам DMA и PMA.

Таблица 10.2

Поле	Значение	Пояснение
E	не ноль все нули	Адрес внешней памяти. Адрес внутренней памяти процессора или внутренней памяти другого ADSP-2106x (M и S активизированы).
M	000 не ноль 111	Адрес собственной внутренней памяти процессора. M = ID, адрес внутренней памяти другого процессора, где ID – идентификатор МП ADSP 2106x. Широковещательная запись во внутреннюю память всех ADSP 2106x.
S	00 01 1x	Адрес регистров устройства ввода-вывода (IOP). Адрес в пространстве адресов нормальных слов. Адрес в пространстве адресов коротких слов (x – старшие биты адреса короткого слова).

Устройство ввода-вывода проверяет адреса всех обращений к памяти и направляет их в соответствующее пространство памяти. Поля адреса E (внешний), M (многопроцессорный) и S (внутренний) декодируются устройством ввода/вывода. Если поле E нулевое, то поля M и S станут

активными и будут декодироваться. Более подробное объяснение назначения адресных полей E, M и S дано в таблице 10.2.

Устройству ввода-вывода принадлежат 256 отображенных в адресном пространстве памяти регистров (регистры УВВ на рис. 10.12). Регистры УВВ служат для управления и конфигурации системы ADSP 2106x, а также для управления выполнением операций ввода-вывода. Адресное пространство между адресами регистров УВВ и адресами нормальных слов с 0x0000 0100 по 0x0001 FFFF является не используемой памятью, и в него нельзя производить запись.

Когда ADSP 2106x генерирует адрес для одного из четырёх банков, то активизируются соответствующие линии выбора памяти \overline{MS}_{3-0} . Сигналы \overline{MS}_{3-0} могут быть использованы для выбора микросхем памяти или внешних устройств. Тем самым устраняется надобность во внешней декодирующей логике. \overline{MS}_0 в комбинации с сигналом PAGE обеспечивает выбор банка динамической памяти DRAM.

Размер банка памяти должен быть равен степени двойки и может изменяться от 8 килослов до 256 мегаслов. Размеры банков памяти задаются путем установки соответствующих бит в одном из регистров процессора, отвечающих за конфигурацию памяти.

Небанковая память

Область памяти выше банков 0-3 называется небанковым пространством внешней памяти. Для доступа в это адресное пространство линии выбора памяти не используются. Временные параметры доступа к небанковому пространству памяти при необходимости также можно изменять.

Начальная загрузка памяти

Когда ADSP 2106x сконфигурирован для начальной загрузки из EPROM, то начальная загрузка производится из отдельного внешнего пространства памяти в сопровождении сигнала \overline{BMS} (*Boot Memory Select*). В многопроцессорной системе сигналом \overline{BMS} управляет только ведущий ADSP 2106x.

Многопроцессорная система с общей системной магистралью

Структура многопроцессорной системы, в которой связь между процессорами осуществляется по общей параллельной шине, представлена на рис. 10.11. В состав такой системы могут входить до шести процессоров ADSP2106x и хост-процессор.

Управление шиной в многопроцессорной системе

В многопроцессорной системе внешняя шина может использоваться совместно несколькими процессорами без дополнительного устройства управления. Этому способствует заложенная в каждый процессор логика арбитража шины, которая позволяет включать в состав системы до шести ADSP 2106x и хост-процессор.

Управление шиной выполняется с использованием сигналов \overline{BR}_{1-6} , \overline{HBR} и \overline{HBG} . Сигналы \overline{BR}_{1-6} управляют переключением между несколькими ADSP 2106x, а \overline{HBR} и \overline{HBG} управляют передачей шины от ведущего процессора к хост-процессору и обратно.

В таблице 10.3 определены сигналы микросхемы ADSP 2106x, предназначенные для использования в многопроцессорных системах.

Для арбитража шины используется две схемы приоритетов: схема с фиксированными и с циклически изменяющимися приоритетами. Высокий уровень сигнала на входе RPBA (*Rotating Priority Bus Arbitration Select*) устанавливает схему с циклически изменяющимися приоритетами.

Когда процессор получает управление шиной, он может обращаться не только к внешней памяти, но и к внутренней памяти и регистрам УВВ всех других процессоров. Процессор может прямо передавать данные в другой процессор или инициализировать канал ПДП для передачи данных. Все процессоры отображены в общей карте памяти системы. Для идентификации адресного пространства любого процессора внутри объединенной карты памяти каждому процессору присваивается уникальный идентификатор ID.

Таблица 10.3

Вывод	Тип	Функция
\overline{BR}_{1-6}	I/O/S	Запрос шины в многопроцессорной системе (<i>Bus Requests</i>). Используются в многопроцессорных системах ADSP 2106x для арбитража шины. Каждый процессор управляет сигналом только на своей линии \overline{BR}_x , определяемой значением двоичного кода его идентификатора на входах ID ₂₋₀ , и контролирует все другие сигналы.
\overline{CPA}	I/O	Приоритетный доступ ядра (<i>Core Priority Access</i>). Сигнал позволяет ядру ведомого ADSP 2106x прервать фоновый ПДП и получить доступ к внешней шине. Выход \overline{CPA} является выходом с открытым стоком. Выводы всех сигналов \overline{CPA} в системе соединены вместе.
ID ₂₋₀	I	Многопроцессорный идентификатор . Определяет, какая из запросных линий \overline{BR}_{1-6} используется процессором с идентификационным номером ID. ID = 001 соответствует \overline{BR}_1 , ID = 010 соответствует \overline{BR}_2 и т. д. ID = 000 используется в

		системах с одним процессором. Эти линии определяют конфигурацию системы, и уровни напряжений на них устанавливаются аппаратно.
--	--	--

PRBA	I/S	Выбор вращающихся приоритетов для арбитража шины (<i>Rotating Priority Bus Arbitration Select</i>). Когда RPBA установлен, то это означает, что для арбитража шины многопроцессорной системы выбрана схема вращающихся приоритетов. Когда RPBA сброшен, выбрана схема фиксированных приоритетов. Этот сигнал определяет конфигурацию системы и должен устанавливаться одинаковым во всех ADSP 2106x.
------	-----	---

Внутренняя память МП ADSP 2106x организована таким образом, чтобы повысить эффективность операций ввода-вывода в многопроцессорных системах. Двухпортовая RAM, расположенная на кристалле, позволяет осуществлять высокоскоростную передачу данных между процессорами, а также параллельно выполнять двойной доступ к памяти со стороны ядра процессора.

Блокировка шины и семафоры

Для совместного использования ресурсов многопроцессорной системы, таких как память или ввод-вывод, могут использоваться семафоры. Семафор – это флаг, который может считываться и записываться любым из процессоров, совместно использующим данный ресурс, и его значение определяет возможность доступа к этому ресурсу. Семафоры полезны также для синхронизации задач, выполняемых различными процессорами в многопроцессорной системе.

Ключевым требованием при работе с семафорами в многопроцессорной (многозадачной) среде является считывание и изменение семафора в одной неделимой операции. Это можно сделать, применяя блокировку шины. Из-за того, что внешняя память, а также внутренняя память и регистры УВВ каждого процессора доступны любому другому, семафоры могут размещаться почти везде.

Операция *чтение-модификация-запись* семафора выполняется корректно, если следовать двум простым правилам:

- процессор не должен записывать семафор, если он не является ведущим. Это особенно важно, если семафор размещён в собственной внутренней памяти или в регистрах УВВ;
- при операции чтение-модификация-запись семафора процессор должен управлять шиной на протяжении всей операции.

Этих правил придерживаются и тогда, когда процессор использует блокировку шины, предотвращая одновременный доступ к ней (и к семафорам) со стороны других процессоров.

Блокировка шины может использоваться в комбинации с широковещательной записью для реализации *взаимных семафоров* в многопроцессорной системе. Взаимный семафор должен размещаться по одним и тем же адресам во внутренней памяти (или регистре УВВ) каждого

процессора. Для проверки семафора каждый процессор может считывать его из собственной внутренней памяти. Для изменения семафора процессор сначала запрашивает блокировку шины, а затем выполняет ширококвещательную запись по адресу семафора в каждый процессор, включая себя. Внешняя шина, в этом случае, используется только для модификации взаимных семафоров, но не для их считывания. Это значительно уменьшает трафик шины. Перед тем, как изменить значение семафора, необходимо проверить его состояние, чтобы убедиться, что над семафором не производились действия со стороны других процессоров.

Пример: взаимные семафоры при совместном использовании канала ПДП

Несколько процессоров могут совместно использовать один канал прямого доступа к памяти, используя *регистр управления каналом* в качестве взаимного семафора. Регистр управления каналом ПДП – это отображенный в памяти каждого процессора регистр устройства ввода-вывода. В нем содержится адрес *дескриптора канала* – начальный адрес области памяти, где хранится информация для инициализации канала ПДП: адрес предоставляемой для прямого доступа области памяти, количество и размер передаваемых слов, направление передачи, источник и приемник передаваемых данных. При обращении к регистру управления информация, хранимая в дескрипторе, заносится в регистры канала, после чего канал автоматически начинает свою работу. При этом не нужны никакие действия со стороны процессора. Эти действия должны быть предприняты заранее и направлены на инициализацию дескриптора канала ПДП.

Если содержимое регистра управления равно нулю, то регистр управления как семафор не используется никаким процессором. Если канал ПДП занят, то содержимое регистра управления не равно нулю.

Перед получением доступа к каналу ПДП, необходимо проверить принадлежащий ему семафор. Если канал свободен (значение семафора равно нулю), то процессор может запросить блокировку шины, а затем выполнить операцию чтение-модификация-запись для установки нового состояния семафора. Новое значение семафора в режиме ширококвещательной записи передается всем совместно использующим этот канал ПДП процессорам. Перед выполнением операции чтение-модификация-запись процессор должен повторно проверить семафор, чтобы убедиться, что канал ПДП все еще свободен. Получив в распоряжение семафор, процессор может выполнять запланированную передачу по ПДП. После завершения передачи семафор необходимо освободить, обнулив его значение и в режиме ширококвещательной записи сообщить об этом другим процессорам.

Межпроцессорные сообщения и векторные прерывания

Ведущий процессор может связываться с ведомыми путем записи сообщений в их регистры УВВ. Для передачи сообщений между процессорами могут использоваться предназначенные для этого универсальные регистры (восемь регистров сообщений) устройства ввода-вывода. Эти регистры могут быть полезны также при реализации семафоров и для совместного использования ресурсов разными процессорами.

К числу универсальных регистров относится также регистр многопроцессорного векторного прерывания VIRPT (*Vector Interrupt*).

Векторные прерывания используются для межпроцессорного взаимодействия в многопроцессорных системах. Внешний (другой или хост-) процессор вызывает векторное прерывание посредством записи стартового адреса процедуры обработки в регистр VIRPT.

Все названные универсальные регистры могут использоваться для передачи сообщения следующими способами:

- *Передача сообщения.* Ведущий процессор может связываться с ведомым путем записи и/или считывания любого из восьми регистров сообщений ведомого.
- *Векторные прерывания.* Ведущий процессор может запрашивать векторное прерывание у ведомого. Производится это путём записи адреса программы обработки прерывания в регистр VIRPT ведомого. Это вызывает прерывание с высшим приоритетом в ведомом процессоре, который переходит на определённую в векторе запроса программу обработки прерывания.

Таблица прерываний

Таблица векторов прерываний может размещаться как во внутренней, так и во внешней памяти. Это зависит от того, используется или не используется режим начальной загрузки, а также от установки соответствующего бита в имеющемся у процессора регистре конфигурации системы. Адреса в таблице представляют собой смещения относительно базового адреса. Для таблицы векторов прерываний во внутренней памяти базовый адрес – 0x0002 0000 (начало блока 0); для таблицы векторов прерываний во внешней памяти базовый адрес – 0x0040 0000. Каждому вектору выделяются четыре 48-разрядные ячейки памяти.

Кластерная многопроцессорная система

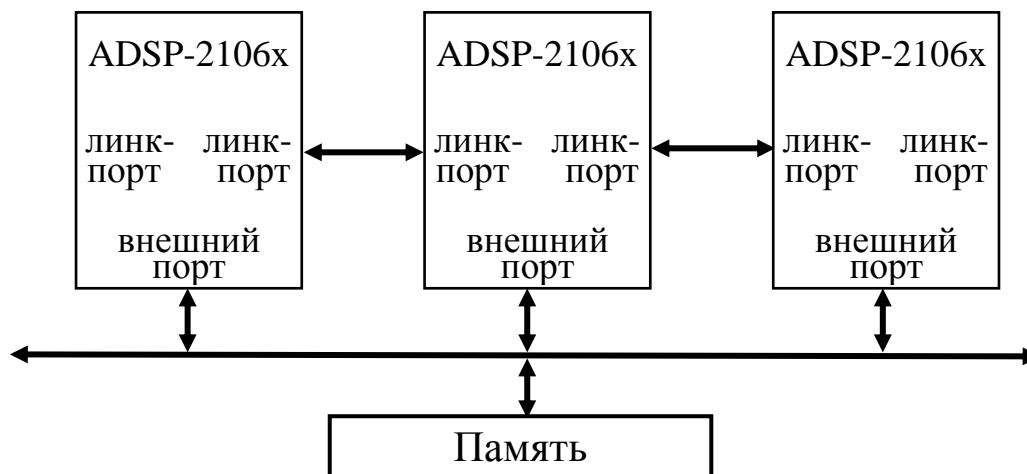


Рис. 10.14. Кластерная многопроцессорная система.

В общем случае многопроцессорная система может быть организована, исходя из совместного использования системной магистрали и линк-портов. Такую систему можно характеризовать как кластерную (имеющую не более шести процессоров ADSP 2106x и хост-процессор) многопроцессорную систему, которая обладает более развитой функциональностью, но требует значительно более трудоемкого программирования. Конфигурация кластерной многопроцессорной системы представлена на рис. 10.14.

10.2.4. Направление развития МП с архитектурой SHARC

Особенностью микропроцессоров с архитектурой SHARC является наличие в них средств межпроцессорного взаимодействия в виде линк-портов. Эта особенность сохраняется и в следующих версиях этих МП. Развитие идет в сторону увеличения вычислительных возможностей, в частности, путем создания многоядерных процессоров и увеличения разрядности внутренних шин. Примером может послужить процессор ADSP-21160 (рис. 10.15), структурное построение которого похоже на структуру ядер процессоров семейства ADSP-2106x, за исключением ширины шин и второго вычислительного блока с собственным умножителем, АЛУ, устройством сдвига и регистровым файлом. Такая структура свойственна процессорам с SIMD (*Single Instruction Multiple Data*) архитектурой.

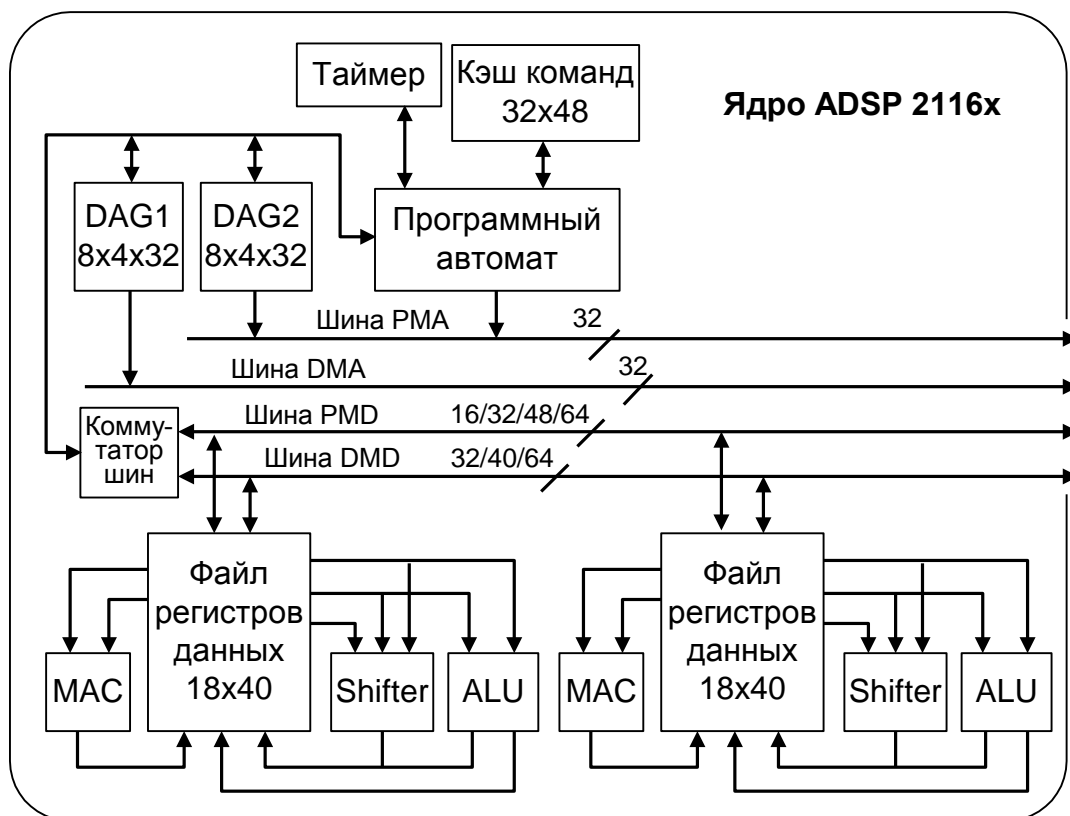


Рис. 10.15. Ядро микропроцессоров семейства ADSP 2116x

Процессоры ADSP 2116x имеют два процессорных элемента, которые могут одновременно выполнять команду каждый над своими данными, т.е. обрабатывать два потока данных параллельно при незначительном изменении программной модели.

Пути дальнейшего повышения производительности компания Analog Devices связывает со статическим выявлением параллелизма уровня команд. Примером может послужить процессор ADSP-TS001 – TigerSHARC. В нем возможности цифровой обработки сигналов основаны на сочетании особенностей RISC и VLIW (*Very Long Instruction Word*) архитектур.

Черты RISC отражены в фиксированной структуре команд и их конвейерном выполнении с предсказанием переходов, в наличии большого регистрового файла и, как следствие, в эффективной загрузке вычислительных блоков.

VLIW подход заключается в планировании загрузки функциональных блоков. Чтобы обеспечить поступление команд во все функциональные блоки, необходимо эффективно использовать доступную ширину слова команды. Иначе говоря, заложенные в многофункциональные команды управляющие воздействия должны подаваться на вычислительные блоки одновременно и связанный с этим параллелизм выполнения операций должен планироваться заранее, до непосредственного выполнения программы. Выявление параллелизма уровня команд и возможность независимого задания в программе порядка загрузки функциональных блоков происходит на этапе компиляции.

С точки зрения аппаратных решений VLIW подход требует увеличения разрядности шин и организации памяти с целью поддержки обращений к

памяти программы и операций одновременной загрузки и сохранения данных. Аппаратные решения процессоров ADSP-TS001 выглядят как развитие представленной на рис. 10.15 структуры ADSP 2116x в направлении расширения функциональных возможностей составляющих ее блоков.

10.2.5. Отражение гарвардской архитектуры в микроконтроллерах и процессорах общего назначения

В 1991 г. компании Motorola, IBM и Apple Computers объявили об организации консорциума для совместной разработки и внедрения RISC микропроцессоров новой архитектуры. Для решения этой задачи были сделаны крупные инвестиции (около 1 млрд долл.) и построен новый центр проектирования в г. Остин (Техас), открытый в мае 1992 г. Штат центра составили 300 ведущих специалистов из компаний IBM и Motorola, в результате работы которых уже в октябре 1992 г. были получены первые образцы 32-разрядных RISC микропроцессоров семейства PowerPC 60x, а с апреля 1993 г. начался их серийный выпуск. В октябре 1994 г. было положено начало новой ветви 64-разрядных процессоров.

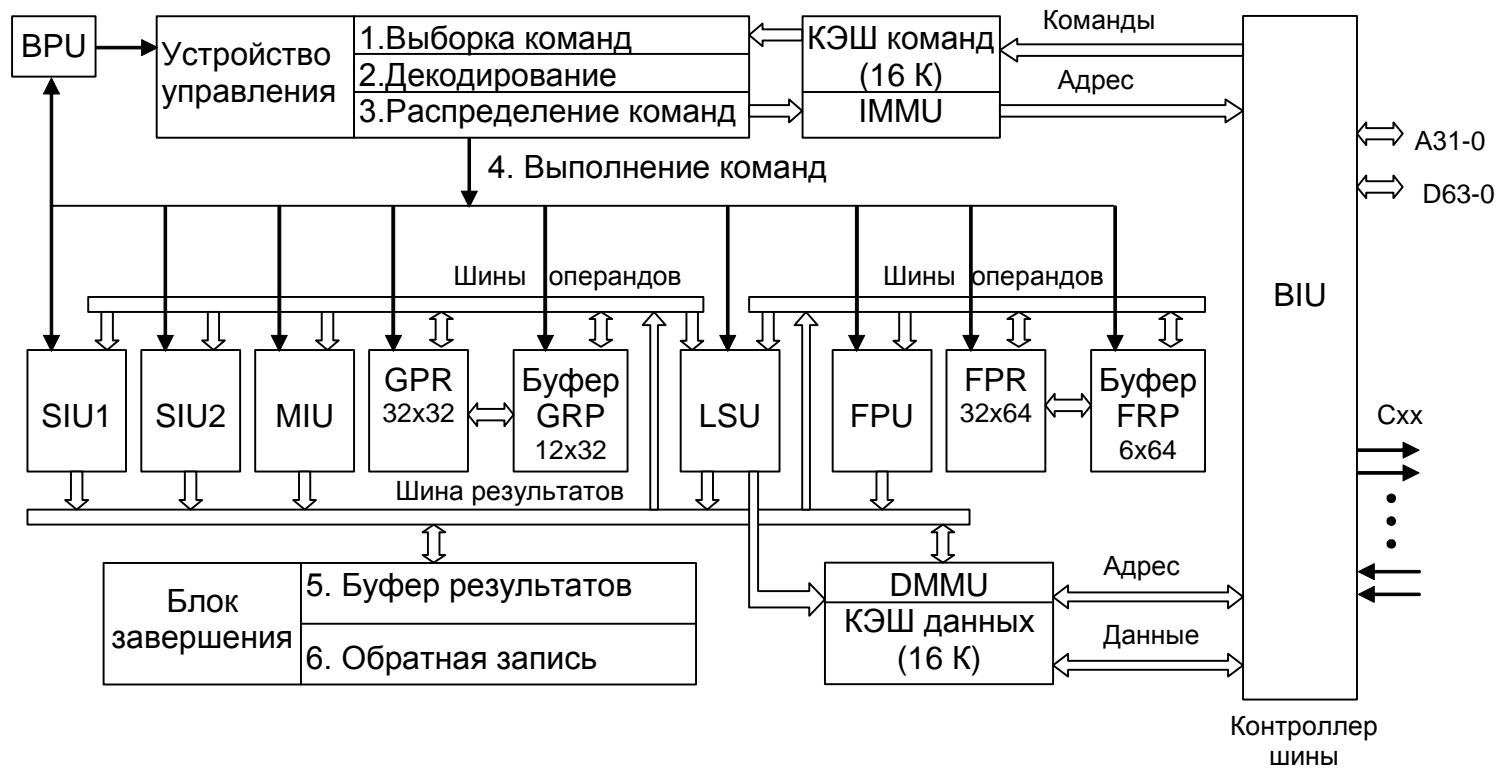


Рис. 10.16. Структура микропроцессора МСР604

Модели микропроцессоров семейства PowerPC 60x, выпускаемые фирмой Motorola, входят в семейство MPC60x. Все микропроцессоры семейства содержат от 4 до 6 параллельно работающих исполнительных устройств, обеспечивающих одновременное выполнение нескольких команд. Эти микропроцессоры послужили также основой для создания новых семейств микроконтроллеров MPC5xxx и коммуникационных контроллеров MCP860.

Рассмотрим структуру (рис. 10.16) и функционирование наиболее производительной 32-разрядной модели MPC604.

В микропроцессорах PowerPC, как и в семействе MCP6xx, реализован принцип выделения отдельных ресурсов для решения задач пользователя и супервизора (операционной системы). В соответствии с этим принципом архитектура PowerPC содержит регистры, входящие в модели пользователя или супервизора, а также ряд привилегированных команд, выполняемых только в режиме супервизора.

В регистровую модель пользователя, которая является общей для всего семейства PowerPC, входят:

- тридцать два 32-разрядных регистра общего назначения GRP31–0 для хранения целочисленных операндов;
- тридцать два 64-разрядных регистра FPR31–0 для хранения операндов с плавающей точкой;
- 32-разрядные регистры условий (признаков) CR и регистр FPSR состояния при обработке чисел с плавающей точкой;
- три 32-разрядных регистра, используемые при обработке исключительных ситуаций (событий внутри микропроцессора и прерываний).

Эта регистровая модель не содержит программного счетчика PC и указателя стека SP. Отсутствие PC связано с тем, что микропроцессоры PowerPC не выполняют команд записи или чтения программного счетчика (действия со счетчиком видны только на аппаратном уровне). Эти микропроцессоры не выполняют также автоматических операций со стеком. В случае необходимости стек реализуется программно, поэтому в регистровой модели отсутствует SP.

Микропроцессор MPC604 (рис. 10.16) содержит шесть параллельно работающих исполнительных устройств: блок обработки ветвлений BPU, два устройства для выполнения простых (одноцикловых) целочисленных операций SIU1 и SIU2, одно устройство для выполнения сложных (многоцикловых) целочисленных операций MIU, устройство обработки чисел с плавающей точкой FPU и блок выборки операндов из памяти LSU. При этом обеспечивается одновременное выполнение четырех команд. Все операции обработки данных выполняются с регистровой адресацией. Выборка данных из памяти производится только командами пересылки, которые выполняются блоком LSU и осуществляют загрузку данных в регистры GRP и FRP или запись содержимого этих регистров в память.

Выборка команд из памяти и обращение к данным осуществляются с использованием отдельных шин адреса и данных и отдельных устройств управления памятью команд IMMU (*Instruction Memory Management Unit*) и памятью данных DMMU (*Data Memory Management Unit*). Такое решение подобно тому, которое используется в процессорах с гарвардской архитектурой, но с той разницей, что непосредственное обращение к основной памяти по шинам адреса и данных отсутствует, поскольку внутренняя память процессора MPC604 представлена кэш-памятью команд и кэш-памятью данных.

При параллельной работе исполнительных устройств возможно их одновременное обращение к одним регистрам. Чтобы избежать ошибок, возникающих при этом, в случае записи в регистр нового содержимого до того, как другим устройством будет считано предыдущее, введены буферные регистры – 12 для GRP и 8 для FRP. Эти регистры служат для промежуточного хранения операндов, дублируя регистры GRP и FRP. После завершения этих операций производится перезапись полученных результатов в GRP и FRP (обратная запись).

Конвейер выполнения команд имеет шесть основных каскадов: выборка команды (1) и ее дешифрация (2), распределение команд по исполнительным устройствам (3), выполнение команд (4), запись результатов в буфер (5) и обратная запись результатов в регистры GRP или FRP. Устройство управления одновременно выбирает из кэша и декодирует четыре команды, которые распределяются в соответствующие исполнительные устройства. Большинство команд выполняется за один такт. Эти команды реализуются с помощью SIU1 и SIU2, которые осуществляют сложение/вычитание, сравнение, сдвиги, логические операции. Выполнение ряда команд требует нескольких тактов. Например, целочисленное умножение выполняется за 3-4 такта, деление за 20 тактов. Эти операции производятся MPU. Большинство операций с плавающей точкой (кроме деления) выполняется за три такта, поэтому во внутренней структуре FPU реализовано три исполнительных каскада. Блок завершения обеспечивает получение правильной последовательности результатов выполняемых операций. Он содержит буферную память типа очереди, в которую по мере поступления записываются результаты, получаемые от исполнительных устройств. Обратная запись результатов из буфера в регистры GRP и FRP производится в соответствии с порядком поступления команд, выбираемых устройством управления, т.е. восстанавливается необходимая последовательность результатов.

Блок обработки ветвлений BPU предсказывает направление ветвления программы при поступлении соответствующих команд. В микропроцессорах семейства MPC6xx реализуются два механизма предсказания переходов: статическое предсказание, которое дается программистом и содержится в тексте выполняемой программы, и динамическое предсказание, которое выполняется BPU по результатам предыдущих ветвлений. Блок BPU для предсказания использует кэш адресов ветвлений BTAC и таблицу истории

ветвлений ВНТ. Кэш ВТАС хранит 64 адреса ветвлений, выполнявшихся предыдущими командами. Если поступившая команда ветвления содержит адрес, совпадающий с одним из имеющихся в ВТАС, то процессор предсказывает ветвление по этому адресу и выбирает команды из соответствующей ветви в конвейер (ветвление выполняется). Если адрес ветвления не содержится в ВТАС, то выбирается следующая команда программы (ветвление отсутствует).

При поступлении команд условных ветвлений ВРУ анализирует таблицу ВНТ, которая представляет собой кэш-память, где содержатся 512 адресов предыдущих условных ветвлений. Для каждого адреса в таблице имеются два бита, определяющие вероятность ветвления по данному адресу: 00 – практически не выполняется, 01 – выполняется редко, 10 – выполняется часто, 11 – выполняется очень часто. Значения этих битов меняются после каждой операции ветвления: если производится ветвление по данному адресу, то вероятность увеличивается на единицу (максимальное значение 11), если данный адрес не использован, то вероятность уменьшается на единицу (минимальное значение 00). Новые адреса вводятся в ВНТ вместо адресов с нулевой вероятностью. Ветвления по имеющемуся в таблице адресу предсказываются, если его вероятность больше 01. Правильность предсказания проверяется после определения соответствующих признаков в регистре условий CR. Если условия не выполняются (неправильное предсказание), то процессор освобождает конвейер от команд из неправильно выбранной ветви и загружает новые команды из другой ветви. Таким образом, при неправильных предсказаниях требуется перезагрузка конвейера.

Так как одновременно могут выполняться несколько команд, изменяющих значения признаков в регистре CR, то в ВРУ имеются восемь буферных регистров, дублирующих содержимое CR. Эти регистры используются различными исполнительными устройствами, а после завершения соответствующих операций их содержимое переносится в основной регистр условий CR (обратная запись), обеспечивая правильный порядок выполнения условных команд.

Микропроцессор MPC604 осуществляет отдельную выборку команд и данных с помощью двух устройств управления памятью IMMU, DMMU, которые могут выполнять сегментную, страничную и блочную адресацию. Работа IMMU, DMMU обеспечивается с помощью 8 пар регистров IBAT, DBAT, 16 сегментных регистров SR0-SR15 и регистра SDR1, обращение к которым осуществляется только в режиме супервизора.

Процессор может генерировать адреса в реальном режиме, когда сформированный адрес поступает на адресные выходы в качестве физического адреса ячейки памяти или порта ввода-вывода. В этом случае IMMU, DMMU отключены и действия по трансляции адреса, подобные представленным в разделе 6.7, не производятся. Включение IMMU, DMMU производится путем установки соответствующих битов в одном из регистров, отвечающих за конфигурацию управления процессором. В этом случае сформированный адрес

команды или данных воспринимается как логический, который с помощью IMMU или DMMU транслируется в физический.

Рассмотрим реализуемые IMMU, DMMU варианты адресной трансляции.

1. *Блочная трансляция* обеспечивает обращение к блокам внешней памяти заданного объема – от 128 Кбайт до 256 Мбайт. Возможна организация четырех блоков для хранения команд и четырех – для хранения данных. Параметры каждого блока задаются дескриптором, который содержится в соответствующей паре регистров IBAT для команд и паре регистров DBAT.

Если устройство IMMU или DMMU включено, то старшие разряды сформированного логического адреса команды или данных, определяющие базовый адрес блока заданного размера, сравниваются со значениями индекса в дескрипторах блоков с целью идентификации блока. Младшие разряды, указывающие относительное положение байта (смещение), транслируются в младшие разряды физического адреса.

Помимо адресной информации в дескрипторах содержатся биты, обеспечивающие защиту памяти и определяющие работу кэша.

2. *Сегментная трансляция* обеспечивает обращение к сегментам памяти емкостью 256 Мбайт, которые разбиваются на страницы размером по 4 Кбайт, размещаемые в ОЗУ, или представляют массивы данных, расположенные во внешних устройствах. Для обращения к сегментам используются 16 сегментных регистров SR0-SR15, которые содержат дескрипторы сегментов. четыре старшие разряда логического адреса задают номер регистра SR_i, из которого выбирается дескриптор соответствующего сегмента. В зависимости от установленного типа дескриптора реализуется страничная трансляция памяти или адресация портов ввода-вывода.

При страничной трансляции устройство управления памятью IMMU или DMMU формирует 52-разрядный виртуальный адрес VA0-51¹⁰, составленный из виртуального адреса сегмента VSID, индекса страницы (разряды LA4-19 логического адреса) и относительного адреса байта в странице (разряды LA20-31 логического адреса). Старшие разряды VA0-39 виртуального адреса, представляющие виртуальный номер страницы VPN, поступают в блок страничной трансляции (БСТ), определяющий физический номер страницы RPN, который служит разрядами PA0-19 физического адреса. Младшие разряды физического адреса совпадают с адресом байта на странице PA20-31 = LA20-31. Для определения RPN используются дескрипторы страниц PD, которые хранятся в специальной таблице TPD, хранящейся в ОЗУ. Базовый адрес этой таблицы задается содержимым регистра SDR1.

Дескрипторы страниц, к которым выполнялось обращение, хранятся в специальной кэш-памяти TLB, входящей в состав IMMU и DMMU. Каждый TLB содержит 128 дескрипторов PD. Работа TLB организована по такому же принципу, как и работа кэш-памяти данных и кэш-памяти команд (см. разделы 6.6 и 6.7).

¹⁰ Здесь 0 относится к старшему разряду, а 51 – к младшему.

При организации многопроцессорных систем обычно делается так, чтобы кэш-память данных каждого процессора была доступной для других процессоров, но при этом необходимы аппаратные решения по синхронизации кэшей разных процессоров для того чтобы исключить возможность получения из кэш-памяти недействительных данных как со стороны своего, так и со стороны других процессоров. К своему собственному кэшу процессор обращается по виртуальному адресу. Обращение со стороны других процессоров происходит со стороны системной магистрали и только по физическому адресу. Поэтому в многопроцессорных системах кэш-память данных наряду с памятью виртуальных тегов должна содержать память физических тегов, а внутренняя связанная с DMMU шина адреса является двунаправленной.

Контроллер шины BIU обеспечивает интерфейс микропроцессора с внешними устройствами. Системная шина передает 32-разрядный адрес и 64-разрядные данные. Увеличенная разрядность шины данных позволяет быстрее производить загрузку строк кэш-памяти.

Рассмотренные применительно к архитектуре MSP6xx структурные и архитектурные решения в той или иной степени свойственны широкому классу как специализированных микропроцессоров, так и процессоров общего назначения. Их можно найти в процессорах семейства P6, в которых отражена общая линия микропроцессоров Intel 80/86, в 32-разрядных RISC процессорах семейств ARM9, ARM10 компании ARM Limited, используемых в качестве ядра для различных встроенных приложений, в процессорах серии SPARC фирмы Sun Microsystems. Перечень можно продолжать. Но и в том, что обозначено, можно найти характерные особенности развития:

- внутренняя гарвардская архитектура с разделением потоков команд и данных;
- одновременное выполнение нескольких команд в параллельно работающих исполнительных устройствах;
- конвейер операций с предсказанием ветвлений.

На базе RISC процессоров с архитектурой MSP6xx разработаны микроконтроллеры и коммуникационные контроллеры для использования в системах управления и связи. Их структурное построение мало чем отличается от рассмотренных в разделах 9.3 и 9.4 структур микроконтроллеров фирм Texas Instruments и ARM Limited. Различия имеются в составе периферийных устройств, а также, что более существенно, в архитектуре центральных процессорных элементов ЦПЭ, которые могут выглядеть как простые с архитектурой фон-Неймана процессорные элементы, так и как сложные функционально развитыми аппаратные комплексы.

В сфере встраиваемых приложений активно развивается производство высокопроизводительных 8-разрядных RISC-микроконтроллеров. Одним из признанных лидеров в этом направлении является фирма Atmel Corp. Широкое распространение нашли выпускаемые этой фирмой микроконтроллеры AVR.

Остановимся лишь на семействе AVR Classic, включающем в общей сложности 17 моделей микроконтроллеров (МК). Микроконтроллеры этого семейства построены по гарвардской архитектуре, в которой память программ выполнена как ПЗУ, составленное из FLASH- и EEPROM-памяти, а память данных – как статическое ОЗУ (рис. 10.17).

Микроконтроллер всегда работает под управлением программы, загруженной в ПЗУ. Для загрузки ПЗУ используется последовательный синхронный интерфейс SPI. Конфигурация микросистемы осуществляется под управлением программы путем записи соответствующей информации в регистры управления.

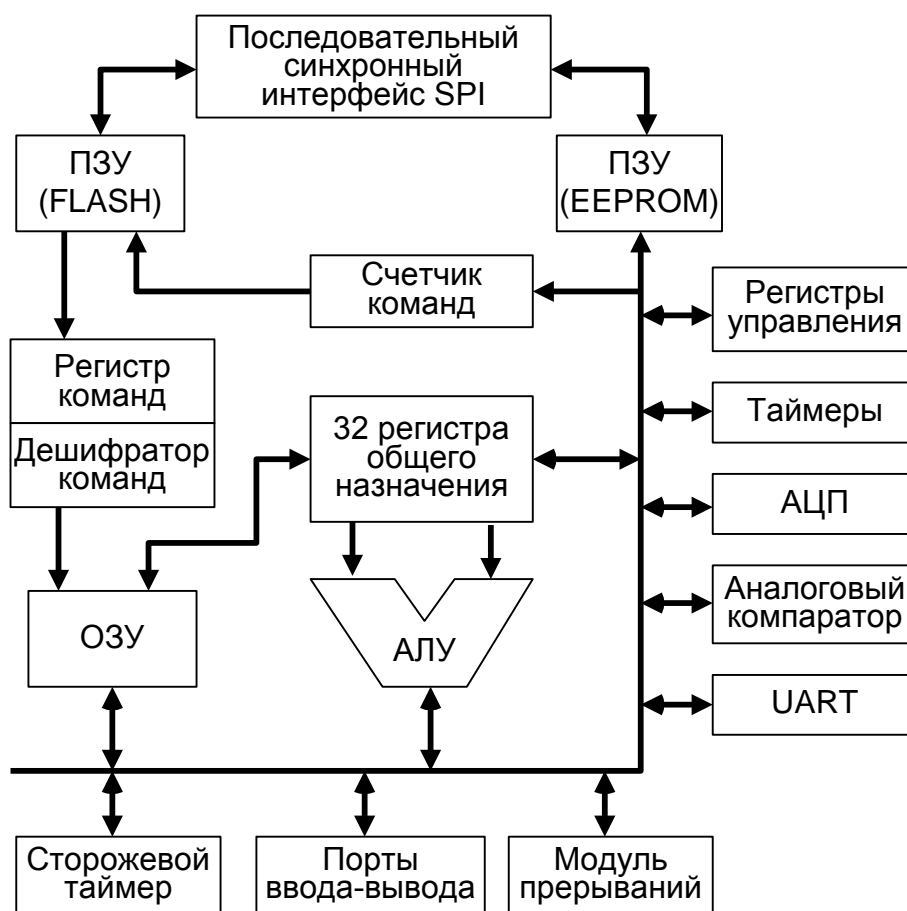


Рис. 10.17. Блок-схема микроконтроллера AVR

Набор периферийных устройств, интегрированных в микросхему МК зависит от требований по компоновке. В представленном на рис. 10.17 варианте число и функции периферийных устройств не выходят за пределы тех, что описаны в разделе 9.3, посвященном микроконтроллерам семейства MSP430 фирмы Texas Instruments.

Заключение

Целью данного учебного пособия было уяснение логики и принципов работы основных функциональных составляющих вычислительных и микропроцессорных систем – центрального процессора и памяти и на этой основе рассмотрение вопросов развития микропроцессорной техники, направленного, в частности, на повышение производительности и обработку сигналов в реальном масштабе времени, на встроенные приложения. За пределами рассмотрения осталось многое из того, что определяет тенденцию развития цифровых устройств. Это относится к программируемой логике с высокой степенью интеграции применительно к задачам обработки сигналов и реализации компонентов МП систем для радиосвязи, управления различными объектам и коммуникаций. Содержащиеся в пособии разделы, посвященные конвейеризации и распараллеливанию операций, следует рассматривать как первый шаг на пути к освоению мультипроцессоров с их разнообразной архитектурой и спецификой реализации, к системам, предназначенным для высокопроизводительных вычислений. Кроме того, полноценное изучение вычислительных систем и систем цифровой обработки сигналов невозможно без приобретения знаний в области их программирования, включая программируемую логику и системы на кристалле.

Учебное пособие разработано при поддержке программы «Научные и научно-педагогические кадры инновационной России на 2009-2013 г.г.» (Госконтракт № 02.740.11.0163 от 25.06.2009).

Литература

1. Алексенко А.Г., Шагурин А.А. Микросхемотехника: Учебное пособие для вузов – 2-е изд., перераб. и доп. – М.: Радио и связь, 1990. – 496 с.
2. Угрюмов Е.П. Цифровая схемотехника. – СПб.: БХВ-Петербург, 2002. – 528 с.
3. Каган Б.М. Электронные вычислительные машины и системы: Учеб. пособие для вузов. – М.: Энергоатомиздат, 1990. – 552 с.
4. Ю-Чжен Лю, Гибсон Г. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем: Пер. с англ. – М.: Радио и связь, 1987. – 512 с.
5. Клигман Э. Проектирование специализированных микропроцессорных систем: Пер. с англ. – М.: Мир, 1985. – 363 с.
6. Электроника СБИС. Проектирование микроструктур: Пер. с англ. / Под ред. Н. Айнспрука. – М.: Мир, 1989. – 256 с.
7. Микропроцессорные системы: Учебное пособие для вузов / Е.К. Александров, Р.И. Грушницкий, М.С. Куприянов, О.Е. Мартынов, Д.И. Панфилов, Т.В. Ремизевич, Ю.С. Татаринев, Е.П. Угрюмов, И.И. Шагурин; под общ. ред. Д.В. Пузанкова. – СПб.: Политехника, 2002. – 935 с.
8. Проектирование СБИС: Пер. с япон. / М. Ватанабэ, К. Асада, К. Кани, Т. Оцуки. – М.: Мир, 1988. – 304 с.
9. Куприянов М.С., Мартынов О.Е., Панфилов Д.И. Коммуникационные контроллеры фирмы Motorola. – СПб.: БХВ Петербург, 2001. – 500 с.
10. Корнеев В.В., Киселев А.В. Современные микропроцессоры. – 3-е изд., перераб. и доп. – СПб.: БХВ Петербург, 2003. – 448 с.

Содержание

Предисловие	3
1. Основные положения алгебры логики	4
2. Схемотехническая реализация логических операций и функций	8
2.1. Полный дешифратор	8
2.2. Мультиплексор	9
2.3. Схемотехника базовых логических операций	10
2.3.1. Диодные дизъюнкторы	10
2.3.2. Диодные конъюнкторы	12
2.3.3. Выполнение логических операций с помощью транзисторов	12
2.3.4. Диодно-транзисторная логика	16
2.3.5. Транзисторно-транзисторная логика	18
2.3.6. Интегральная инжекционная логика	20
2.3.7. Эмиттерно-связанная логика	21
2.3.8. МДП-ключи и логика на МДП-структурах	22
2.3.8. Элементы с тремя состояниями	28
2.3.10. Особенности логических операций на передающих транзисторах. Динамические элементы	29
3. Функциональные узлы комбинационного типа	35
3.1. Арифметические устройства	35
3.1.1. Двоичные сумматоры	35
3.1.2. Матричные умножители	40
3.2. Программируемые логические матрицы (ПЛМ)	42
3.3. Программируемая матричная логика (ПМЛ)	45
3.4. Базовые матричные кристаллы (БМК)	47
4. Функциональные узлы последовательного типа (автоматы с памятью)	49
4.1. Триггерные устройства. Классификация. Основные сведения	49
4.2. Регистры и регистровые файлы	53
4.3. Двоичные счетчики	55
4.3.1. Асинхронные (последовательные) счетчики	55
4.3.2. Параллельные (синхронные) счетчики	57
4.4. Регистровое арифметическо-логическое устройство	60
5. Машины состояний. Микропрограммные автоматы	65
5.1. Классификация машин состояния	66
5.2. Машины состояния и матричная логика	68
5.3. Микропрограммирование и устройство управления выполнением программы	74

5.3.1. Архитектура и реализация	74
5.3.2. Структуры адресации памяти микропрограмм	78
6. Запоминающие устройства	82
6.1. Основные структуры адресных запоминающих устройств	84
6.2. Статические оперативные запоминающие устройства	89
6.3. Динамические оперативные запоминающие устройства	90
6.4. Постоянные и репрограммируемые запоминающие устройства	93
6.5. Энергонезависимые оперативные запоминающие устройства	97
6.6. Кэш-память	99
6.7. Механизм виртуальной памяти	103
7. Микропроцессоры: архитектура и структурное построение	107
7.1. Функционально-структурные особенности микропроцессоров	107
7.2. Формат команд центрального процессора	108
7.3. Типовая архитектура и последовательность выполнения команд центральным процессором	112
7.4. Управление выполнением программы	121
7.4. Структура центрального процессора и взаимодействие с МП-системой	128
8. Регистрово-ориентированные архитектуры	132
8.1. Типы операндов и иерархия памяти	132
8.2. Многочисленные перекрывающиеся окна регистров	133
8.3. Наборы команд, ориентированные на регистровую архитектуру	134
8.4. Конвейеризация и регистровая память	138
8.5. Микроархитектура процессора RISC II	140
8.5.1. Трехуровневый конвейер микропроцессора RISC II	140
8.5.2. Тракт обработки данных микропроцессора RISC II	142
8.5.3. Секция управления микропроцессора RISC II	143
9. Микропроцессорные системы с архитектурой фон-Неймана	148
9.1. Магистрально-модульная структура микропроцессорных систем	148
9.2. Системы с магистральным интерфейсом	152
9.3. Магистральный интерфейс в микроконтроллерах	158
9.4. Микроконтроллеры для коммуникационных приложений	171
10. Микропроцессоры с гарвардской архитектурой и микросистемы на их основе	177
10.1. Цифровые процессоры сигналов семейства ADSP 21xx	177
10.1.1. Структура ядра	177
10.1.2. Программный автомат	181

10.1.3. Устройства обработки данных	184
10.1.4. Адресация памяти данных	187
10.1.5. Особенности программирования	189
10.2. Средства повышения производительности ядра процессора и многопроцессорные системы	193
10.2.1. Ядро ЦПС семейства ADSP 2106x	194
10.2.2. Подсистема ввода-вывода ЦПС семейства ADSP 2106x	197
10.2.3. Ресурсы и архитектура многопроцессорной системы на базе ADSP 2106x	198
10.2.4. Направление развития МП с архитектурой SHARC	209
10.2.3. Отражение гарвардской архитектуры в микроконтроллерах и процессорах общего назначения	210
Заключение	218
Литература	219

Евгений Иванович Шкелев

АППАРАТНЫЕ СРЕДСТВА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Учебное пособие

Государственное образовательное учреждение высшего
профессионального образования Нижегородский государственный
университет им. Н.И. Лобачевского

Подписано в печать _____. Формат 60x84 1/16.

Бумага офсетная. Печать офсетная. Гарнитура Таймс.

Усл. печ. л. _____. Уч.-изд. л. _____.

Заказ № _____. Тираж 300 экз.

Отпечатано в типографии Нижегородского госуниверситета
им. Н.И. Лобачевского

603600, г. Нижний Новгород, ул. Большая Покровская, 37

Лицензия ПД № 18-0099 от 14.05.01