В. С. ВЫХОВАНЕЦ

Организация ЭВМ и систем

Учебное пособие для вузов

Рекомендовано научно-методическим советом Приднестровского государственного университета в качестве учебного пособия для студентов высших технических учебных заведений

В учебном пособии, состоящем из 4 книг, излагается организация современных вычислительных средств. Рассматриваются теоретические основы обработки данных и принципы построения процессоров (книга 1), компьютеры универсального назначения (книга 2), системы высокопроизводительной обработки данных (книга 3), архитектура и организация современных систем обработки данных (книга 4). Представлен прикладной аспект автоматизации проектирования.
Для студентов, обучающихся по специальности "Вычислительные машины, комплексы, системы и сети".

Раздел 1. Теоретические основы дискретной обработки данных

Тема 1.1. Программная реализация автоматов

Данные (DATA) – есть информация (сведения), представленные (закодированные) в виде, пригодном для обработки вычислительными средствами и (или) человеком.

Обработка данных (processing) – это процесс преобразования входных данных в выходные, осуществляемый вычислительным средством и (или) человеком.

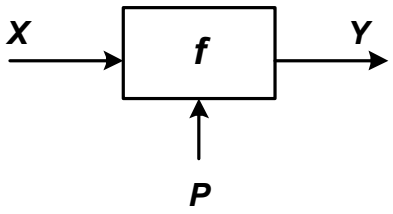


Рис. 1.1. Преобразование данных

P – дополнительное воздействие.

Существует 2 типа обработки: дискретная и аналоговая.

Аналоговая обработку (signal processing) – обработка данных представленных в виде аналоговых сигналов.

Дискретная обработка данных (data processing) – обработка данных представленных в виде конечной закодированной последовательности.

Рисунок 1.1 можно интерпретировать, как дискретную функцию R=f(D, P).

Общая постановка задачи состоит в необходимости выполнить декомпозицию произвольной дискретной функции над системой более простых функций.

Пусть A- входной алфавит, а B- выходной, тогда получим y=f(x),

где $y \in B=\{b_1, b_2, \ldots, b_m\}$

$x \in B=\{a_1, a_2, \ldots, a_n\}$

Предположим, что мы ограничимся следующими операциями:

$y=f_i(x)$ – унарная функции преобразуют текущий знак входного алфавита в выходной,

$y=g_j(x)$ – бинарные функции,

$y=h_k(x)$ – тернарные функции,

Введем дополнительную операцию, которая называется разделение переменных.

Функциональная декомпозиция

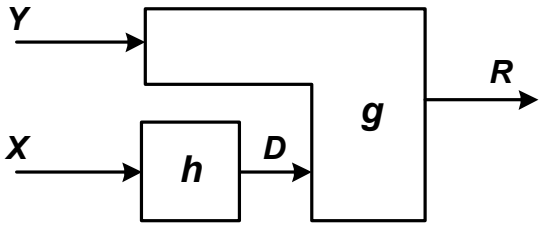
Представим преобразование входных данных в выходные в виде некоторой последовательности элементарных операций.

Пусть задана функция $f(X)$, где X – входные данные или множество переменных. Все известные подходы при обработке дискретных данных сводятся к следующей декомпозиции:

$f(X) = g(Y, h(Z))$

$X = Y \cup Z$ (исходные данные делим на 2 части, возможно пересекающиеся)

g, h – некие функции обработки данных.

Аппаратная реализация	Программная реализация
	$obj\ \ X\ X; obj\ \ Y\ Y;$ $obj\ \ Z\ Z; obj\ \ D\ d;$ $obj\ \ R\ r;$ $Y = obj\ Y(X); Z = obj\ Z(X);$ $obj\ \ D\ \ h(obj\ Z);$ $obj\ \ R\ \ g(obj\ Y, obj\ D);$ $r = g(Y, h(Z));$

Различают однократную и многократную декомпозиции.

$f(X) = g(Y_1,...,Y_r; h_1(Z_1),...,h_S(Z_S))$ - многократная.

Также существует итеративная декомпозиция (многократное повторение одной и той же обработки данных), рекурсивная (результат обработки данных является входными данными для такой же обработки).

Условно можно выделить 2 крайних подхода к дискретной обработке данных: 1-ый основан на нерегулярных формах, 2-ой – на регулярных формах декомпозиции.

Нерегулярные	Регулярные
Теоретической основой нерегулярных форм представления дискретной обработки данных служат известные парадигмы (принципы) анализа предметной области и связанные с ним методологии проектирования программных средств, а именно логическая, функциональная, структурная, объектно-ориентированная.	Основой регулярных форм декомпозиции является табличное представление обработки данных. Регулярные формы основаны на следующем виде декомпозиции: $f(X) = \sum_{(i)} g_i(Y) \cdot h_i(Z)$ Крайний случай – спектральное представление: $f(X) = \sum_{(i)} \theta_i(X) \cdot a_i$ a_i – это константа, или коэффициент разложения, θ – спектральная функция.

Автомат с операторной программой

Для любой заданной схемы декомпозиции функции f поставим в соответствие последовательность операций (элементарных функций) следующим образом.

1. Пронумеруем элементы схемы натуральными числами начиная с нуля, так чтобы на любом пути от входа к выходу номера элементов возрастали.
2. Пусть элемент схемы e_i функцию $q_i(e_{j1}, e_{j2}, \ldots, e_{jp})$.
3. Поставим элементу e_i в соответствии некоторую переменную a_i, равную результату вычисления функции $q_i(a_{j1}, a_{j2}, \ldots, a_{jp})$.
4. В итоге получили последовательность операций, в которых порядок нумерации соответствует нумерации функции в схеме декомпозиции, а их выполнение позволяет вычислять искомую функцию при заданных входных данных.

Программа – пронумерованная последовательность операций (команд) вида $К_1, К_2, \ldots, К_n$.
$К=\{ К_1, К_2, \ldots, К_p\}$

Система команд – совокупность команд вида: $a_i = q_i(a_{j1}, a_{j2}, \ldots, a_{ju})$ $u \in N$, выполняющих операцию q_i над u операндами и присваивающая результат переменной a_i.

Выполнение программы это последовательность шагов на каждом из которых выполняется одна команда программы.

Реализация автомата с операторной программой.

Постановка задачи:

Пусть имеется функционально полная система $\Omega=\{f_i, g_j, h_l\}$, $i=1,2,..,k_1$; $j=1,2,..,k_2$; $l=1,2,..,k_3$;

Построим автомат реализующий функции представленные в базисе Ω.

Архитектурные решения

1. Программы и данные хранятся в различных запоминающих устройствах.
2. Используется запоминающее устройство с произвольным доступом.
3. Кодируем систему команд

Данные кодируются в двоичной системе счисления. Разбиваем на n частей, n_X разрядов отводим под данные. Команда кодируется тоже; одна ячейка должна содержать код операции (КОП).

X, Y – номера, показывающие номера ячеек, где хранятся данные.

КОП	X	Y

Рис. 2.2. Кодирование команд и данных

$1 \div n$ – бинарные операции g_i: $\quad M(X)\, g_i\, M(Y) \to A$

Извлекается из памяти по адресу X первый операнд, по адресу Y – второй, выполняется операция g_i и результат помещается в аккумулятор A.

$n+1 \div 2n$ – унарные операции h_i: $\; h_j M(X) \to M(Y)$

Операнд по адресу X преобразуется унарной операцией h_j, результат записывается в ячейку памяти по адресу Y.

$2n+1$ – тернарные операции g_0: $\quad g_0(A, M(X), M(Y)) \to A$

В зависимости от значения аккумулятора значение из X или Y заносится в аккумулятор.

Y может быть константой (0-местная операция). При реализации констант или же непосредственных кодируемых в коде команды данных, можно отказаться от унарных операций, т.к. $M(X) g_i Y \to h_j$

2) Определяем объемы запоминающих устройств.

n_A – разрядность адреса запоминающего устройства (число разрядов для X и Y)

n_K – число разрядов на команду: $\quad n_K = 2n_A +]\log_2(2n+1)[$

$]$ $[$ – наименьшее целое, превосходящее число 2n+1

$]\log_2(2n+1)[$	n_A	n_A
КОП	X	Y

Рис.2.3. Кодирование команд и данных с указанием разрядности

Рассмотрим состояния автоматов

1. Выборка команд
2. Выборка операндов
3. Выполнение операции
4. Вычисление адреса следующей команды. Сохранить результат.

Тема 1.2. Проектирование дискретных устройств

Оценка сложности

Очевидно, что с точки зрения алгоритмов, функция вычислима, если она может быть выполнена за конечное время и требует конечного объема памяти.

Произвольную дискретную функцию будем задавать характеристическим вектором, состоящим из значений функции для всевозможных входных данных.

$f(X): F = [f(0), f(1), ..., f(m-1)], \quad x \in \overline{0, m-1}$

Представим эту функцию в виде таблицы истинности, предварительно разбив входное данное на n частей: $X_0, ..., X_{n-1}$ и будем рассматривать произвольное входное значение как совокупность n переменных.

$X = (X_{n-1}, ..., X_1, X_0)$

X_{n-1}	…	X_1	X_0	$f(X)$
0	…	0	0	$f(0)$
0	…	0	1	$f(1)$
…	…	…	…	…
0	…	0	$K_0 - 1$	$f(K_0 - 1)$
0	…	1	0	$f(K_0)$
0	…	1	1	$f(K_0 + 1)$
…	…	…	…	…
i_{n-1}	…	i_1	i_0	$f(i)$
…	…	…	…	…
$K_{n-1} - 1$	…	$K_1 - 1$	$K_0 - 1$	$f(m-1)$

Т.к. разделение входных данных на части произвольное, будем предполагать, что каждая переменная принимает свой диапазон значений $X_i \in N_{Ki} = \{0,1,..., K_i - 1\}$, что означает, что произвольная переменная X_i имеет значность K_i, где $K_i \in N$.

Установим взаимно однозначное соответствие между значением переменной $X = i$ и значениями переменных $X_j = i_j$, $j = \overline{0, n-1}$ на основе представления числа i в позиционной системе счисления со смешанным (различным) основанием.

$i = (i_{n-1} ... i_1 i_0) K_{n-1} ... K_1 K_0$

Пример.

Задано число $(27)_{1010}$. Представим его в системе счисления с основанием $K_2 = 5$, $K_1 = 3$, $K_0 = 2$.

$(27)_{1010} = (411)_{532} = 1+2+6 \cdot 4 = 27$

$i = i_3(K_2 K_1 K_0) + i_2(K_1 K_0) + i_1(K_0) + i_0$

Произвольный характеристический вектор функции φ определяет $N_x(m)$ дискретных функций, равное числу представлений m в виде произведений натуральных чисел $m = K_{n-1}K_{n-2}...K_1K_0$, которые будут являться значностями переменных.

Необходимо получить выражение f в некоторой функционально полной системе операций. $\Omega=\{f_i, g_j, h_l\}$, где f_i, g_j, h_l соответственно унарные, бинарные и тернарные операции.

Систему операций Ω необходимо дополнить операциями разделения входных данных на части.

Декомпозицию дискретных функций общего вида принято считать задачей связанной с перебором большого числа решений и на практике на практике практически не реализуемую.

Будем рассматривать частный случай декомпозиции, спектральную декомпозицию вида:

$$f(X) = \sum_{i=0}^{m-1} \theta_i(X) \cdot a_i \ , \quad a_i \in N_K , \quad \theta_i(X) \in N_K = \{0,1,...,K-1\}$$

Алгебра образующих операций

Существует 5 типа алгебр, позволяющих представить произвольную дискретную функцию в виде спектрального разложения.

1. Алгебра логики $A_L =< N_K, +, \cdot >$

Потребуем, чтобы операции + и $\cdot$ были таковы, что $\exists\, \sigma$ и $\tau \neq \sigma$ такие, что

$$\begin{array}{l|l} \sigma + X = X & \sigma \cdot X = \sigma \\ X + \sigma = X & \tau \cdot X = X \end{array}$$

σ – 0 алгебры

τ – 1 алгебры

$\sigma = 0$

$\tau = 3$

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & * & * & * \\ 2 & * & * & * \\ 3 & * & * & * \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 0 \\ * & * & * & * \\ * & * & * & * \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Матрица сложения Матрица умножения

2. Мультипликативная алгебра

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & * & * & * \\ 2 & * & * & * \\ 3 & * & * & * \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Матрица сложения Матрица умножения

Операция умножения должна быть группой

3. Аддитивная алгебра

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 0 \\ * & * & * & * \\ * & * & * & * \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Матрица сложения Матрица умножения

Операция сложения должна быть абелевой группой

4. Конечное поле

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

Матрица сложения Матрица умножения

Действует дистрибутивный закон умножения относительно сложения.

5. Кольцо целых чисел
   R=<Z,+,*>

Асимптотические оценки

Теорема: (основная теорема оценки трудоемкости синтеза)

Существует метод синтеза формального представления дискретной функции при котором количество не нулевых коэффициентов разложения M(m) и число операций L(m) необходимых для ее вычисления, удовлетворяют следующим асимптотическим оценкам. Устремим длину характеристического вектора $m \to \infty$.

$L(m)$ – число операций, необходимых для вычисления функций.

$M(m)$ – объем памяти, необходимый для хранения коэффициентов.

Где m – число переменных, на которые разбивается аргумент функции.

$$M(m) \sim \frac{4}{(K_f^2 + K)^2} \frac{K^n}{n^2}$$

$$L(m) \sim \frac{8}{(K_f^2 + K)^2} \frac{K^n}{n}$$

$n \sim \log_k(m)$ $K \sim \sqrt[n]{m}$

Замечание:

1. При выводе теоремы использовалась произвольная функционально полная система операции, состоящая из всех известных унарных и бинарных операций.
2. Почти все функции не имеют эффективной программной или аппаратной реализации.
3. Необходимо использовать архитектурное проектирование вычислительных средств с использованием архитектурных принципов заключающихся в декомпозиции обработки данных на части, а так же представление этой декомпозиции в виде блок-схемы. Декомпозиция завершается когда полученные блоки представляют элементы которые могут быть реализованы на практике.

Тема 1.3. Архитектурные принципы Неймана.

Возможности автоматического проектирования аппаратных и программных средств ограничены.

1) При проектировании дискретных устройств полная автоматизация этого процесса достаточно трудоемка и практически нереализуема.

2) При проектировании вычислительных устройств происходит разделение на программную и аппаратную части. Проектирование аппаратной части ВС осуществляется в алгебре логики. Программные средства опираются на аппаратно реализуемые операции.

3) Как для программ, так и для устройств имеются следующие оценки:

а) Объем памяти V, который необходим для реализации программы или устройства. Он асимптотически стремится к величине

$$V \sim C_\Omega \frac{\prod_{j=0}^{n-1} K_j}{\left(\sum_{j=0}^{n-1} K_j\right)^2}$$

при реализации произвольной функции $f(x)$ с длиной характеристического вектора $m = \prod_{i=0}^{n-1} K_i$.

б) Число операций стремится к величине

$L \sim C_\Omega \frac{\prod_{j=0}^{n-1} K_j}{\sum_{j=0}^{n-1} K_j}$, где $C\Omega = const$, зависящая от выбора базиса, т.е. для произвольной функции объем памяти, необходимый для реализации этой функции, от нее не зависит.

При достаточно большом объеме входных данных, когда $m \to \infty$, почти все функции реализуются со сложностью, близкой к максимальной. Объем программного устройства, умноженный на время вычисления, необходимое для реализации функции асимптотически стремится к величине

$V \cdot L \sim C_\Omega^2 \frac{m}{\left(\sum_{j=0}^{n-1} K_j\right)^3}$, $m \to \infty$ и не зависит от вида реализаций функций.

Таким образом, при проектировании дискретных устройств мы вынуждены пользоваться некими принципами декомпозиции сложного устройства на части в рамках использования методологии проектирования. Т.е., проектирование дискретных устройств состоит из трех этапов:

1. Системное (архитектурное) проектирование.
2. Логическое (синтез).
3. Техническое.

Архитектурные принципы Неймана.

1. Реализация вычислительного средства в виде программируемого автомата.
2. Принцип хранимой программы и хранимых данных. Программа и данные хранятся в устройствах с линейной организационной памяти (с произвольным доступом, в отличие от МТ).
3. Низкий уровень системы команд.
4. Последовательное выполнение команд.
5. Наличие средств ввода-вывода данных.
6. Иерархическая организация памяти.

Раздел 2. Процессоры

Тема 2.1. Архитектура процессора

При проектировании дискретных устройств мы вынуждены пользоваться некими принципами декомпозиции сложного устройства на части в рамках использования методологии проектирования. Проектирование дискретных устройств состоит из трех этапов:

4. Системное (архитектурное).
5. Логическое (синтез).
6. Техническое.

Системное проектирование. На системном этапе, исходя из общего декомпозиционного подхода, осуществляется декомпозиция вычислительного средства на части (на сеть взаимодействующих автоматов более простой структуры). Данный этап не является формализуемым, а потому описание этих автоматов – неформализованное. Результат проектирования - общая блок схема вычислительного средства, структура устройства. Методы проверки результатов системного проектирования – архитектурное проектирование и математическое моделирование.

Логическое проектирование. При логическом проектировании осуществляется формализация и синтез структурных частей, полученных на предыдущем этапе. Исходные данными являются описания функционирующих блоков частей и их взаимодействий. Результатом является логические или функциональные схемы.

Логический этап разделяется на 2 подэтапа:

1. Формализация.
2. Структурный синтез.

Техническое проектирование. На основе логических и функциональных схем строятся принципиальные и монтажные схемы, и готовится технологическая документация производства и эксплуатации программного средства. То есть, имея функциональные декомпозиции структурных частей, строят схему из логических элементов с учетом отличия абстрактных элементов от реальных.

Процессор – это устройство или функциональная часть цифровой вычислительной системы, предназначенное для интерпретации программы (ГОСТ 15871-91).

Процессор – функциональная единица, которая распознает и выполняет команды (ISO 2382110-79).

Представим результат архитектурной декомпозиции по Нейману в виде структурной схемы:

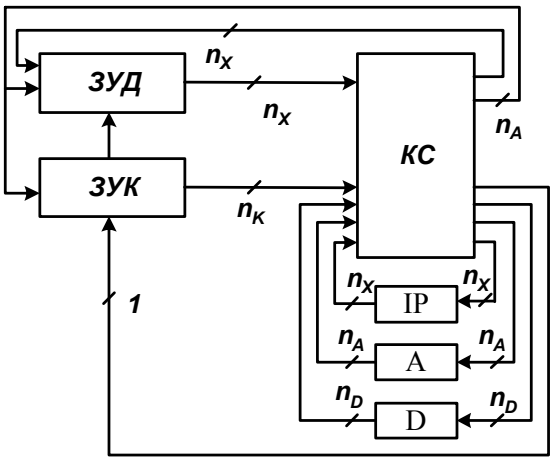


Рис.2.1. Структурная схема процессора

ЗУД – запоминающее устройство данных
ЗУК – запоминающее устройство команд
УК – указатель команд (счетчик команд)

D – внутренняя память конечного автомата, занимающаяся обработкой данных

n_p – число разрядов, которые используются для кодирования состояний автомата.

Рассмотрим возможные состояния процессора:

1. Остановка
2. Модификация УК и выборка команды
3. Выборка операндов
4. Выполнение операций
5. Сохранение результата

Элементы архитектуры Процессора.

1. Методы кодирования и типы обрабатываемых данных.
2. Адресная структура памяти.
3. Форматы и типы команд
4. Способы адресации данных.
5. Описание регистровых структур и управляющих регистров.
6. Механизмы и принципы взаимодействия с внешним окружением.

Структурная организация процессора

Глушков В.М. предложил декомпозировать КА (процессор) на операционный и управляющий блоки.

Операционный блок – автомат, предназначенный для выполнения операций преобразования данных.

Управляющий блок – предназначен для управления операционным блоком и внешним окружением процессора в соответствии с программой.

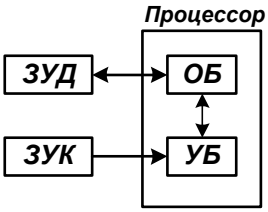


Рис.2.2. Структурная организация процессора

ОБ – автомат, предназначенный для выполнения операций преобразования данных.

УБ – автомат, предназначенный для управления операционным блоком и внешним окружением процессора в соответствии с последовательностью команд, которые он инициирует.

Функциональная организация процессора

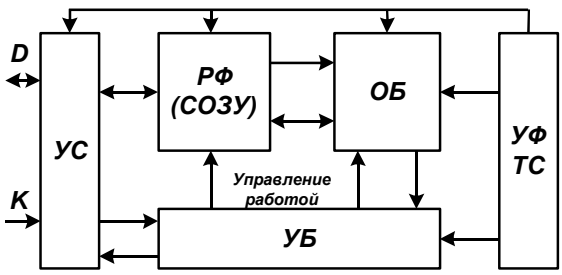


Рис. 2.3. Функциональная организация процессора

РФ – регистровый файл,
УС – устройство связи с внешним окружением,
D – данные,
K – команды,

УФТС – устройство формирования тактовых сигналов.

Тема 2.2. Операционный блок

Регистровый файл – есть совокупность ячеек промежуточной памяти необходимых для реализации команд процессора в виде последовательности операций низкого уровня. Он предназначен для хранения и доступа к наиболее часто используемым операндам.

Регистровый файл может работать в двух режимах: чтения или записи, но не одновременно. Регистровый файл состоит из множества регистров (сколько регистров столько ячеек).

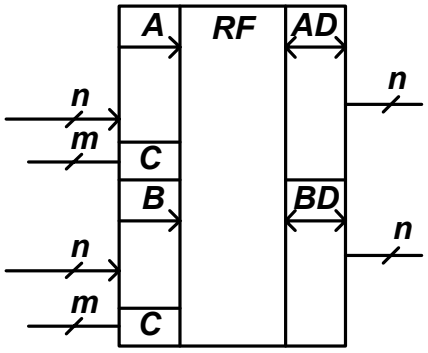


Рис. 2.4. Регистровый файл

A, B – адресные линии,
AD, AB – операнды A и D.

Схемотехническое устройство регистрового файла.

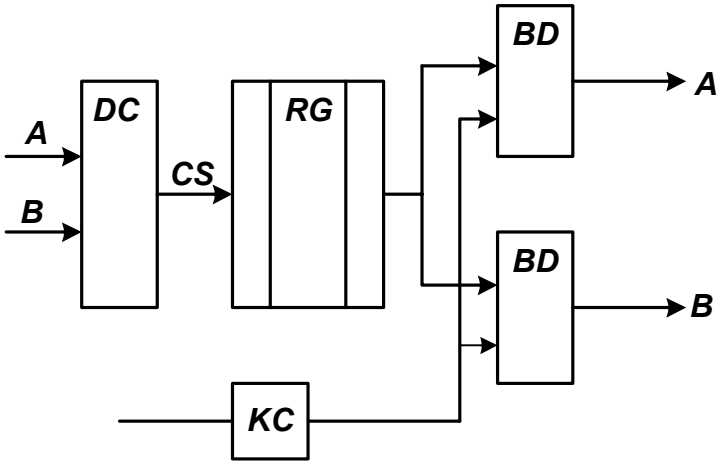


Рис. 2.5. Схемотехническое устройство регистрового файла

Операционный блок (ОБ) – это автомат, предназначенный для выполнения арифметических и логических преобразований операндов.

Операционный блок должен реализовывать функционально полную систему операций. Рассматривая примеры бинарных и операторных программ, мы сделали вывод, что команды, которые позволяют нам реализовывать функционально полную систему операций, бывают двух типов:

1. Команды условного перехода (Когда мы рассматривали булевую алгебру, в качестве вычисляемого условия была проверка на значение ячейки памяти). В общем случае, когда мы перейдем к к-значной логике, эта проверка на какое-то значение из к возможных значений.

2. Команды операторных программ. В общем случае – это команды: $a_i = \varphi_i(a_{j1}, a_{j2}, .., a_{jp})$ - это n-местная команда извлекает значения $a_{j1}, a_{j2}, .., a_{jp}$ и записывает в a_j. С помощью таких команд можно представить произвольную обработку данных, причем

эти команды должны образовать функционально-полную систему операций и эти команды в памяти располагаются последовательно.

Доказано, что если использовать команды условных переходов и операторные команды, то объем программы сокращается (по отношению к использованию только операторных команд).

Чем больше операций выполняет операционный блок, тем более эффективно может быть реализована обработка данных. Эффективность программы понимается в следующем смысле: это объем памяти необходимый для ее хранения, представления и среднее время вычислений.

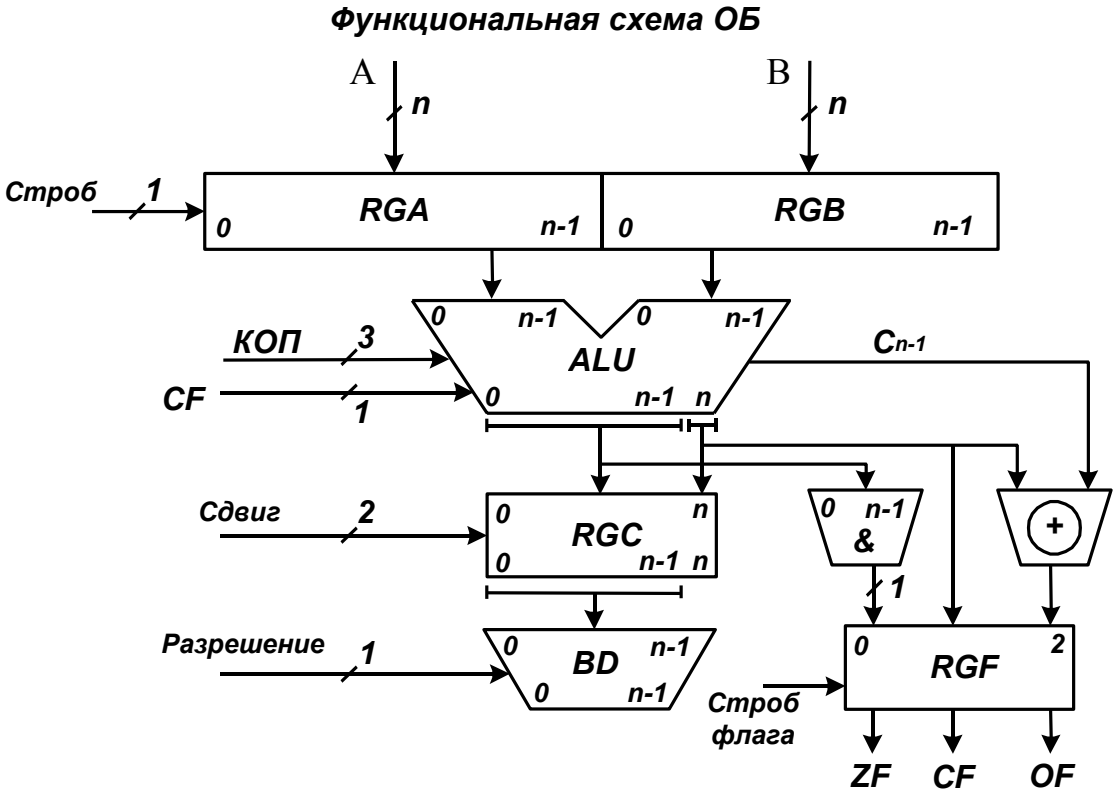


Рис. 2.6. Функциональная схема операционного блока

1. На входы A и B подаются адреса.
2. Регистровый файл выставляет на выходе данные.
3. ALU выполняет с этими данными операцию в соответствии кодом операции.
4. Результат записывается в регистр сдвига, который выполняет или не выполняет сдвиги. На этом же шаге вычисляются регистры.
5. Результат записывается в регистровый файл по адресу первого операнда.

Существует операция «нет операции» которая используется тогда, когда от ALU ничего не требуется. При выполнении унарной операции используется вход A.

Тема 2.3. Управляющий блок

Микрооперацией называется элементарная операция выполняемая за 1 тактовый интервал и приводимая в действие одним управляющим сигналом (примеры:код операции, сдвиг, выход, строб).

Микрокоманда – это совокупность микроопераций выполняемых за 1 тактовый интервал.

Поясним это определение. Если внимательно посмотреть на наш ОБ, то окажется, что все микрооперации выполняются в разное время, а т.к. наш ОБ изначально является последовательной схемой, то при проектировании такие устройства делают синхронными, а значит можно представить себе работу нашего ОБ, как устройство, которое работает в несколько тактов (3 такта) и в результате выполнения такта подается та или иная команда.

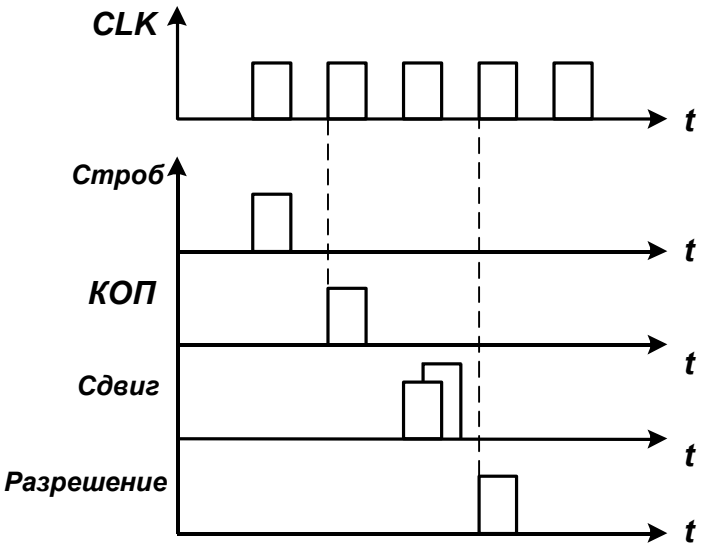


Рис. 2.7. Временная диаграмма работы ОБ

В начале мы должны подать некую команду на регистровый файл, как правило, это команда чтения содержимого двух каких-то регистров регистрового файла. Из кода операции, как правило, извлекается регистр первого операнда, регистр второго операнда и выдается команда чтения. Поэтому в 1 такте выполняется микрооперация чтения/записи (R/W) регистрового файла. Понятно, что в этот такт выдать сигнал на ALU - не имеем возможности, т.к. данные не извлечены из регистрового файла. В последующем такте данные появились на входе ALU и только после того как эти данные появились, мы можем выдать операцию на ALU преобразования этих данных. После того, как результат появился на входе регистра сдвига, мы должны будем выполнить в следующем такте сдвиговую операцию. Мы можем выполнить операцию сдвига только в свое время. Понятно, что в это время выдается операция строба, для того чтобы зафиксировать полученные в результате выполнения этой операции какие-то флаги. Только после этого в следующем такте можно сформировать сигнал «Разрешение» на шинном формирователе и вместе с этим подать сигнал чтения/запись для регистрового файла (в данном случае запись).

Микропрограмма – это последовательность микрокоманд каждая, из которых выполняется за 1 тактовый интервал. Каждая команда имеет свою микропрограмму.

Управляющий автомат с «жесткой» логикой

Любой оперативный блок содержит фиксированное конечное количество микроопераций, которое он может выполнить. То есть если у нас есть N микроопераций, то возможно существования 2^N микрокоманд. Если длина микропрограммы m, то у нас есть 2^{mN} микропрограмм. Ставится задача обеспечить возможность выдачи любой последовательности микрокоманд, каждая из которых состоит из произвольного количества микроопераций.

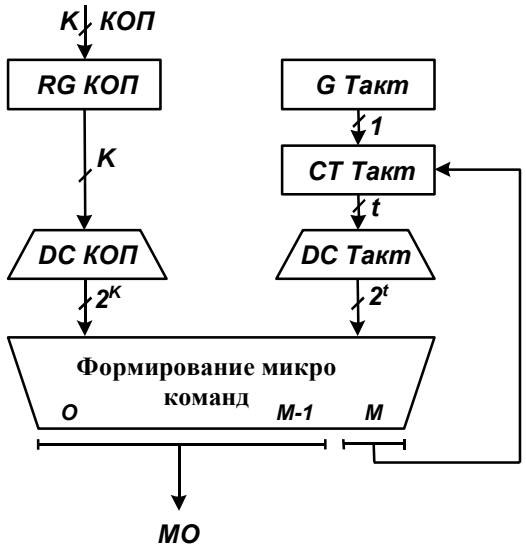


Рис. 2.8. Управляющий автомат с «жесткой» логикой работы

«Жесткая» логика означает, что управляющий блок изначально изготавливается при проектировании процессора и в процессе подстроить его работу нельзя.

Первое что нам потребуется – это регистр, где хранится код текущей исполняемой команды (RG КОП). Код операции является микрооперацией. Код команды – способ кодирования команды в ЗУК. В этот регистр данные могут записываться из ЗУК. Код исполняемой команды нужен для определения, какую команду мы должны выполнить. Понятно, что команд несколько, поэтому далее мы ставим дешифратор, который определяет, какая по счету команда исполняется на нашем устройстве. Поэтому если у нас возможно N разрядов определяющих команду, то выходов у DC будет 2^N. Далее команда поступает на некоторую комбинационную схему (схема формирования команд), которая определяет какую команду необходимо исполнить в данный момент. Мы должны выработать последовательность микрокоманд, каждая микрокоманда требует для своего исполнения 1 такт. Поэтому необходим также дешифратор такта, на вход которого подается сигнал от счетчика тактов (СТ). В общем случае разрядов счетчика – L, а значит команда может иметь длину $0\text{-}2^L$ тактов. На вход счетчика подается сигнал от тактового генератора.

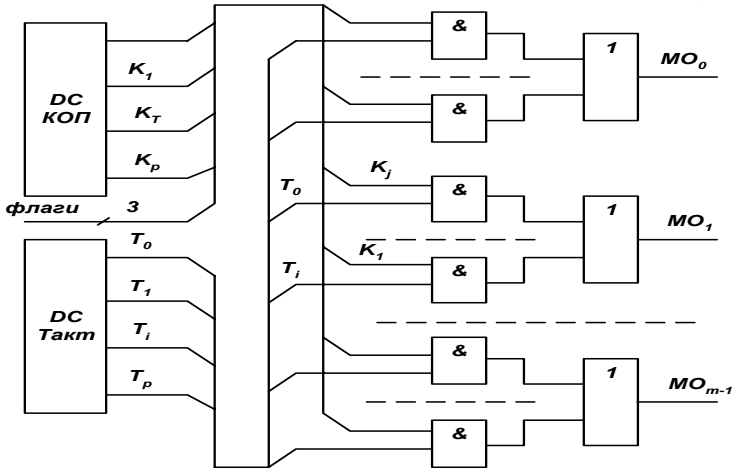


Рис. 2.9. Схема формирования микрокоманд

Для микрооперации 1 можно сказать, что она выдается в такте T_0 при исполнении команды K_j и т.д., в такте T_i при исполнении команды K_1.

Самым сложным в современных процессорах является логика работы. Логика заключается в том, что процессор берет на себя функции поддержки операционной системы. Чем больше проходит времени, тем сложнее те операции или команды, которые выполняет процессор, но микрооперации, через которые эти команды реализуются, остаются фактически неизменными. Таким образом, мы должны усложнять устройство управления.

При проектировании процессора все ноу-хау на 90% включаются в проектировании устройства управления. Стоимость устройства управления составляет 90% стоимости процессора. А в связи с этим хотелось бы сделать такой процессор, который позволял бы изменять логику своей работы. То есть, к примеру, хотелось бы добавить какую либо команду или изменить выполнение некоторой команды, сохранив некое ядро.

В 1956 году английским ученым было предложено использовать запоминающее устройство для хранения логики работы управляющего блока. А уже запоминающее устройство можно программировать.

Управляющий автомат с хранимой в памяти логикой.

Управляющий автомат с хранимой в памяти логикой – это последовательностное устройство, вырабатывающее распределенные во времени управляющие функциональные сигналы, задаваемые содержимым микропрограммной памяти.

Во всех случаях, перед тем как выполнить некую команду, надо считать ее из запоминающего устройства. А значит, что надо иметь некоторый регистр, где эта команда будет храниться во время исполнения.

Регистр адреса микрокоманд (РАМК) является синхронным, и мы будем тактировать его некоторым сигналом $d\overline{H}$. Необходимо так же учитывать состояние операционного блока, то есть его флаги. По адресу микрокоманды из памяти микропрограмм извлекается микрокоманда.

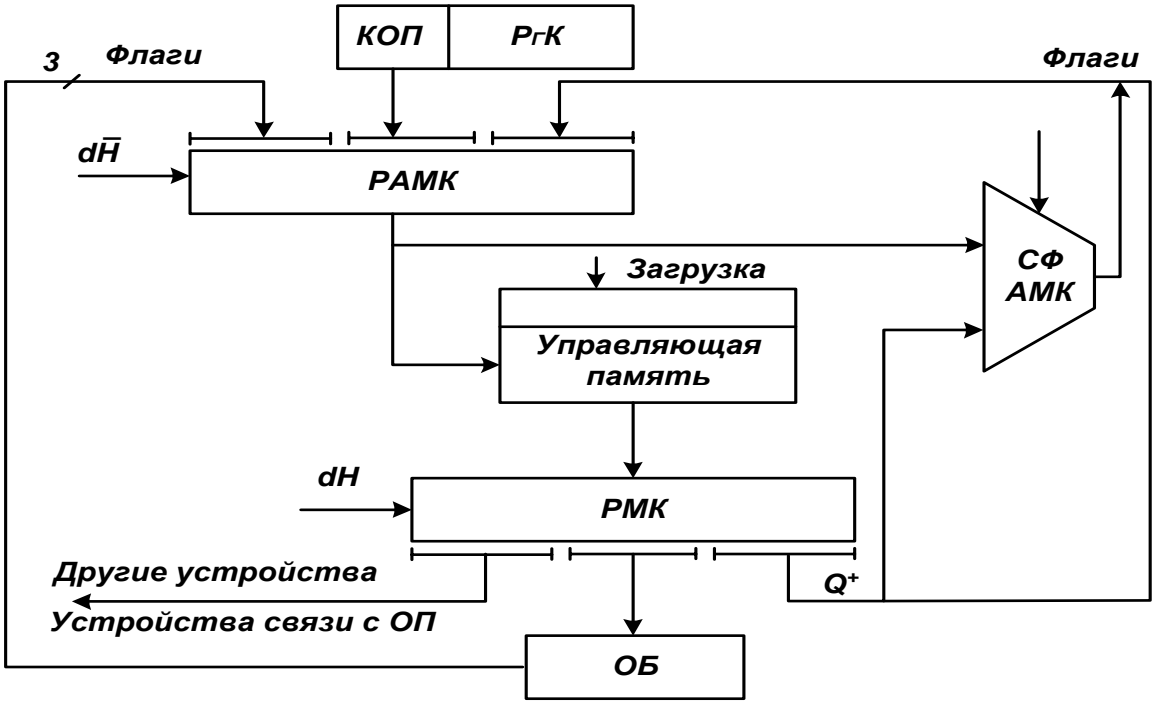


Рис. 2.10. Структура УБ, позволяющего нам изменять микропрограммы, используемые при функционировании процессора

Нарисуем временную диаграмму работы управляющего блока.

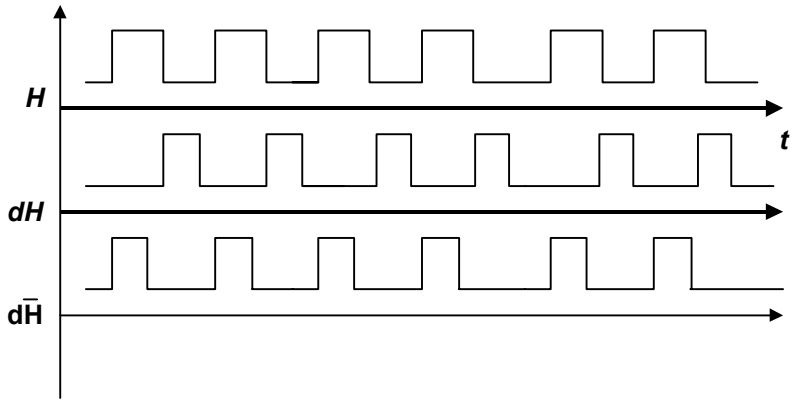


Рис. 2.11. Временная диаграмма УБ

1) КОП совместно с флагом из ОБ и с состоянием управляющего автомата Q^+ задает адрес ячейки управляющей памяти, где хранится микрокоманда и следующее состояние управляющего блока.

2) С приходом $d\overline{H}$ адрес в управляющей памяти фиксируется в РАМК, который определяет новое содержимое регистра микрокоманды (РМК).

3) С приходом сигнала dH в РМК переписывается микрокоманда и новое состояние управляющего автомата.

4) Микрокоманда состоит из микроопераций, которые подаются в управляемые устройства (ОБ, устройство связи с внешней памятью) и до прихода следующего сигнала dH выполняется извлеченная микрокоманда.

5) По окончании выполнения микрокоманды формируется состояние управляемых блоков, которые поступают на вход РАМК.

6) С приходом следующего сигнала $d\overline{H}$ фиксируется новый адрес микрокоманды и процесс повторяется.

Замечание 1. Управляющая память может допускать загрузку содержимого до начала работы процессора или в процессе его работы. Это позволяет изменять логику работы управляющего автомата (например систему команд).

Замечание 2. Число состояний управляющего автомата Q^+ невелико и формирование следующей микрокоманды можно выполнять с помощью комбинационной схемы – СФАМК (схема формирования адреса микрокоманды). Это позволяет реализовать последовательное выполнение микрокоманд с изменением этой последовательности в случае использования микрокоманды перехода.

Микропрограммирование

Существует 3 способа формирования адреса следующей микрокоманды:

1) Способ принудительного формирования (часть адреса следующей микрокоманды содержится в самой микрокоманде).
2) Естественная адресация, при которой микрокоманды исполняются последовательно, а при необходимости используются специальные микрокоманды условного (безусловного) перехода.
3) Смешанный способ.

Способы формирования управляющих функциональных сигналов или микрокоманд:

1. Горизонтальное микропрограммирование.
2. Вертикальное микропрограммирование.
3. Смешанное микропрограммирование.

При горизонтальном микропрограммировании каждому разряду микрокоманды ставится в соответствие микрооперация. Недостаток – большой объем управляющей памяти: одновременно выдается до 10% микроопераций от общего их числа.

При вертикальном микропрограммировании микрооперация разбивается на группы, совместные и несовместные во времени. Под каждую группу несовместных во времени микроопераций выделяется поле в микрокоманде, где кодируется одна из микроопераций по

номеру. В этом случае используется дешифратор микрооперации в этой группе и экономится микропрограммная память.

При смешанном микропрограммировании, которое наиболее распространено, используется как горизонтальный, так и вертикальный способ формирования микрокоманды. Совместные операции во времени кодируются горизонтально, а не совместные - вертикально. Частным случаем смешанного микропрограммирования является горизонтальное размещение групп вертикально кодируемых микроопераций. В этом случае можно эффективно использовать память микропрограмм и тогда условная операция не будет содержать постоянной адресной части..

Замечание. Процессор состоит из множества слабосвязанных блоков (устройств), между которыми устанавливается асинхронное взаимодействие. В этом случае процессор содержит множество слабовзаимодействующих асинхронно управляющих автоматов.

Тема 2.4. Примеры архитектур процессора.

RISC - процессор

RISC (Redused Instruction Set Computer) – процессор с сокращенной системой команд.

Особенности архитектуры RISC:

1) Управляющий автомат с «жесткой» логикой работы.
2) Простая система команд. То есть команды, реализуемые RISC процессором, не сильно отличаются от операций, выполняемых операционным блоком. В связи с тем, что мы вынуждены формулировать программу в командах, которые очень простые, объем кода программы гораздо больше, того который мы получили если бы у нас была сложная система команд.
3) Требуется увеличенный объем ЗУ, а значит и высокое быстродействие, то есть сколько времени выполняется обработка операций операционным блоком. Желательно, чтобы столько же времени происходила выборка команды.
4) Большая загрузка системного интерфейса.
5) В RISC архитектуре время выполнения команды 1,2 такта.
6) В связи с тем, что доступ к ЗУ очень медленный (2 такта), то требуется большой объем внутренней памяти (регистрового файла) процессора для того чтобы хранить промежуточные данные.

Система команд.

Рассмотрим двухоперандные команды:

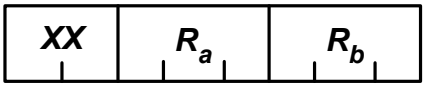


Рис. 2.12. Двухоперандная команда

Закодируем следующим образом:

00 – команда сложения (ADD), которая работает следующим образом Rb←Ra+Rb

01 – LOAD (чтение из памяти), которая работает следующим образом: в регистр Rb заносится содержимое ячейки памяти по адресу Ra: Rb←M[Ra]

10 – STORE (сохранение, запись), алгоритм: M[Ra]←Rb

11 – признак однооперандной команды.

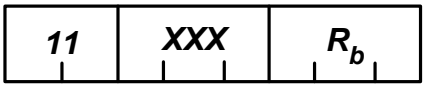


Рис. 2.13. Однооперандная команда

Понятно что для одноопернадной команды у нас требуется наличие какого-то одного регистра в качестве хранителя этого операнда регистр B, а 3 освободившихся разряда будем использовать для кодирования кода операции одноопернадной команды, где

000 – CLR(команда очистки), алгоритм: Rb←0

001 – NOT(инверсия разряда),алгоритм выполнения: Rb←~Rb

010 – INC(команда увеличения на 1) алгоритм: Rb←Rb+1

011 – LOADI (загрузка непосредственно) за командой непосредственно следуют 4 байта загружаемых данных. Алгоритм: Rb←M[PC+1], PC ← PC+n/8

100 – SHL (сдвиг влево) алгоритм: Rb←Rb<<1

101 – SHR алгоритм: Rb←Rb>>1

110 – JC(переход, если установлен флаг переноса) алгоритм: PC←CF?Rb:PC+1

111 – префикс нульместной операции

Нульоперандная команда:

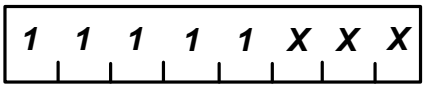


Рис. 2.14. Нульоперандная команда

У нас есть операции только условного перехода. Для того чтобы сделать безусловный переход мы должны установить операции сброса установки флага. Если мы сбросим флаг, а потом выполним JC, то понятно, что у нас будет переход по содержимому регистра B, а если мы сбросим, то эта команда будет называться «нет операции», состоящая из 2 байт.

000 – STC(установка флага переноса) алгоритм: CF←1

001 – CLC(сброс флага переноса) алгоритм: CF←0

010 – STI(установка флага прерывания)

011 – CLI(сброс флага прерывания)

100 – CPO флаг переполнения

101 – CPZ флаг нуля

110 – CPS флаг знака

Функциональная схема

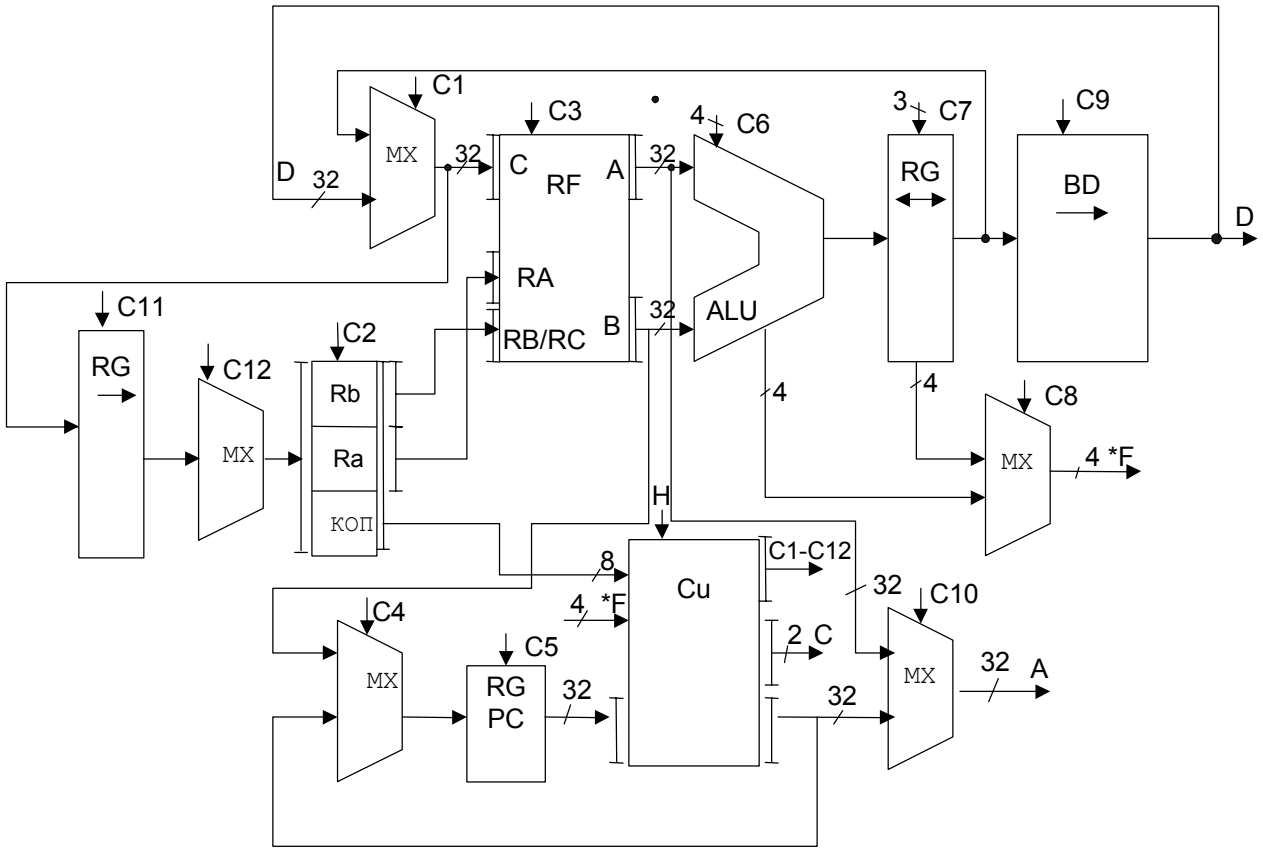


Рис. 2.15. Функциональная схема RISC процессора

Окна Питерсона

Обычно стек реализован путем взаимодействия процессора и основной памяти, то есть процессор в процессе выполнения команды вызова и возврата из процедуры обращается к основной памяти для записи или считывания сохраняемых или восстанавливаемых адресов. Но для RISC процессора это не подходит, потому что в этом случае увеличивается нагрузка на системный интерфейс.

Для повышения эффективности вызова и возврата из подпрограмм используется стек, который реализуется внутри процессора, ограниченного объема. В RISC – архитектурах используются регистровые окна Питерсона, когда регистровый файл разбивается на банки. В каждом банке содержится полный комплект регистров, каждая процедура имеет свой собственный банк.

Регистровые окна имеют кольцевую структуру и перекрываются.

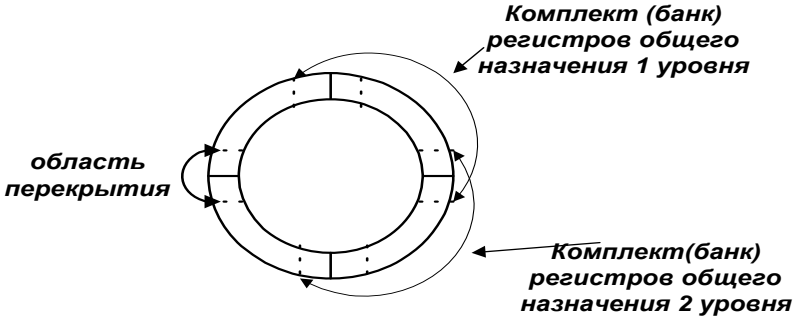


Рис. 2.16. Окна Питерсона

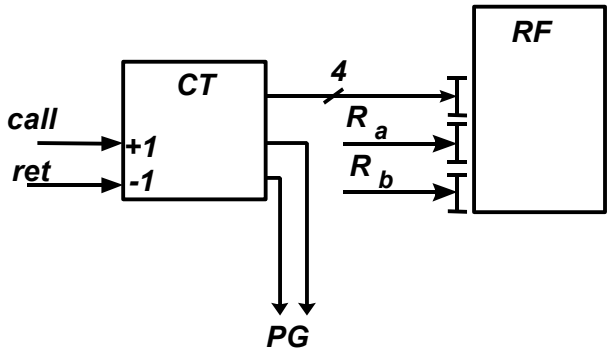


Рис. 2.17. Аппаратная реализация кольцевой памяти

Области перекрытия используются для передачи параметров от одной подпрограммы к другой и для возврата результата в вызывающую подпрограмму. Имеется специальный регистр, хранящий адрес возврата (в каждом банке).

Таким образом, обеспечивается связь подпрограмм с различной глубиной вложенности. То есть, если у нас есть K комплектов регистров, то процессор может реализовать вызов подпрограмм с глубиной вложенности не более чем K.

Возможна такая ситуация, когда происходит переполнение регистрового файла при глубине вложенности процедур ≥ K. Если не предпринять никаких мер, то регистры в одной из вызываемых процедур могут быть запорчены. Счетчик глубины вложенности CT фиксирует ситуацию, когда остался один свободный банк в регистровом файле и выдает сигнал P в управляющий блок. По этому сигналу запускается процедура с фиксированным жестким адресом, который имеет доступ к регистрам произвольного банка и выполняет сохранение этих регистров во внешней памяти. Таким образом обеспечивается глубина вложенности процедур более K.

Если регистровый файл исчерпан и произошел возврат из очередной процедуры, счетчик глубины вложенности выдает сигнал G, сигнализирующий об этом, по которому управляющий блок выполняет процедуру восстановления ранее сохраненного содержимого регистрового файла.

Это решение эффективно, т.к. колебания глубины вложенности процедур за достаточно длительный интервал времени Δt не превосходит объема регистрового файла K.

Заметим, что целесообразно не полное сохранение (восстановление) регистрового файла при превышении заданной глубины вложенности, а частичное, например, наполовину. Тогда, если следующей командой будет команда возврата из процедуры, то в регистровом файле сохраняются необходимые данные.

Т.к. необходимо где-то хранить адрес возврата из процедуры, определяющий точку, куда необходимо передать управление по команде return, используется специальный регистр, например R_0, доступный командам процессора. При вызове туда записывается адрес процедуры, после вызова – адрес возврата из процедуры.

Для работоспособности схемы, показанной на рис. 2.16 необходимо обеспечить плавающую границу начала регистрового файла (начального банка регистрового файла). Не трудно дополнить приведенную схему:

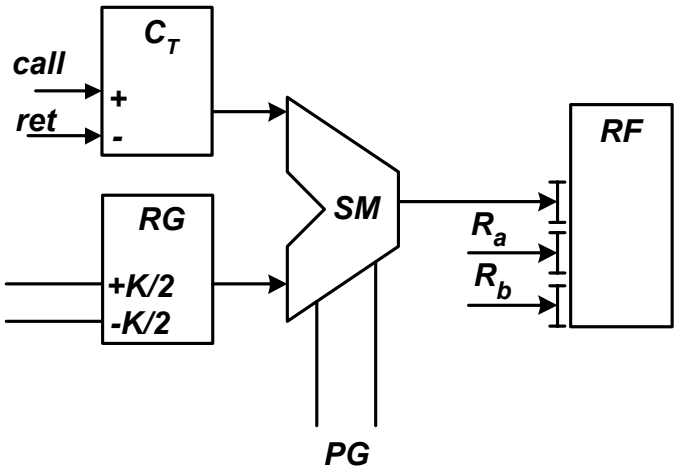


Рис. 2.18. Аппаратная реализация кольцевой памяти с плавающей границей регистрового файла

Проблема перестановки байт

Проблема перестановки байт появляется вследствие неравенства минимально адресуемой порции данных, обрабатываемых процессором, и порции данных, считываемой или записываемой через внешний интерфейс процессора. Вводится понятие слов (разрядность внешнего интерфейса), расположенных на границе слова в основной памяти, которое может быть считано за 1 цикл обмена, и слов не на границе слова, доступ к которым осуществляется за 2 цикла обмена с внешней памятью.

Для того, чтобы выполнить 4 последовательно расположенные команды мы 4 раза читаем из основной памяти одно и тоже слово. Это приводит к неэффективному использованию системного интерфейса. Поэтому для решения такой проблемы делается очередь команд, которая представляет собой сдвиговый регистр, который заполняется всякий раз, когда читается полное слово из основной памяти, а потом команды побайтно сдвигаются в регистр команд.

Каждая 4-я команда вызывает обращение к основной памяти.

Допустим у нас не байтовые команды, а 32 разрядное слово расположено по адресу, не кратному 4. В этом случае основная память может предоставить данные, которые расположены начиная с нулевого адреса по 3, причем этот нулевой адрес есть остаток от деления адреса на 4. И поэтому чтобы взять тот операнд, который находится по адресу не кратному 4 нам все равно необходимо сделать 2 обращения к памяти, потому что основная память не выполняет эту перестановку байт. Поэтому все процессоры должны содержать схему, которая обнаруживает, что требуемая порция данных не находится по адресу кратному 4 и если это так, то произвольное обращение к памяти реализовывать за 2 цикла. В 1 цикле читается то, что находится в предыдущем слове, а часть маршрутизируется в младшую часть нашего операнда.

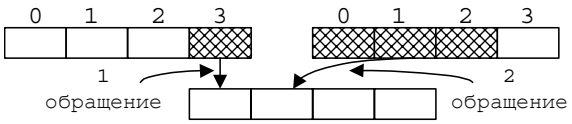


Рис. 2.19. Чтение слова не на границе слова.

Получается, что если требуемые данные у нас находятся по адресу, не кратному 4, то мы должны сначала прочитать первую часть этого данного, занести его в тот регистр, где это данное будет подготовлено, а потом за второе обращение прочитать вторую часть. При этом происходит перепутывание трассы доставки этих данных в этот регистр, которой занимается специальный мультиплексор. Сколько разрядов адреса столько и перепутывания. Такие сложные решения для RISC архитектуры не годятся, поэтому эту проблему решают следующим образом: младшие 2 разряда вообще не выводят из процессора; это означает, что все данные могут находиться только на границе слова.

Процессор архитектуры CISC

CISC –Complex Instruction Set Computer (процессор со сложной системой команд)

Особенности архитектуры:

1) Сложная система команд, определяющая не только возможности выполнения сложных операций, но и позволяющая использовать множество способов адресации данных (регистровый способ – данные находятся в регистре, прямой – адрес данного содержится в коде команды, косвенный – адрес данного в регистре, базовый – адрес вычисляется с использованием косвенного и прямого способа адресации одновременно, т.е. обращение к памяти происходит путем сложения содержимого регистра и части адреса, который кодируется в коде команды - смещение).

2) Микропрограммное устройство управления является автоматом с хранимой в памяти логикой работы.

3) Многотактовое выполнение команды. Тогда возникает интересная идея: коль у нас команда выполняется много тактов, то многие из устройств этого процессора при выполнении команды простаивают. Скорость обработки повышается путем использования суперскалярной обработки команд.

Суперскалярная обработка – это совмещение во времени выполнения двух несвязанных или слабо связанных команд.

Понятно, что можем придумать такое устройство управления, которое будет одновременно декодировать 2 команды и устанавливать факт их связи, которое позволит использовать разные операционные блоки для выполнения этой команды. Современный процессор Intel Pentium содержит 2 независимо работающих ALU и там фактически УБ разделен на несколько частей, которые синхронизируются УБ самого верхнего уровня, отвечающим за взаимосвязь всех частей процессора.

В общем случае, архитектура суперскалярной обработки является частным случаем архитектуры со сверхдлинным командным словом.

Структура микропроцессора

Укрупненная структурная схема МП ВМ86 содержит две относительно независимые части: операционное устройство, реализующее заданные командой операции, и устройство шинного интерфейса, осуществляющее выборку команд из памяти, а также обращение к памяти и внешним устройствам для считывания операндов и записи результатов. Оба устройства могут работать параллельно, что обеспечивает совмещение во времени процессов выборки и исполнения команд. Это повышает быстродействие МП, так как операционное устройство, как правило, выполняет команды, коды которых уже находятся в МП, и поэтому такты выборки команды не включаются в ее цикл.

Операционное устройство МП содержит группу общих регистров, арифметико-логическое устройство (АЛУ), регистр флагов F и блок управления.

Восемь 16-битовых регистров общего назначения участвуют во многих командах. В этих случаях регистры общего назначения кодируются трехбитовым кодом, который размещается в соответствующем поле (или полях) формата команды.

В соответствии с основным назначением рассматриваемых регистров выделяют регистры AX, BX, CX, DX, используемые, прежде всего для хранения данных, и регистры SP, BP, SI, DI, которые хранят главным образом адресную информацию. Особенностью регистров AX, BX, CX, DX является то, что они опускают раздельное использование их младших байтов AL, BL, CL, DL и старших байтов AH, BH, CH, DH. Тем самым обеспечивается возможность обработки, как слов, так и байтов и создаются необходимые условия для программной совместимости ВМ86 и ВМ80.

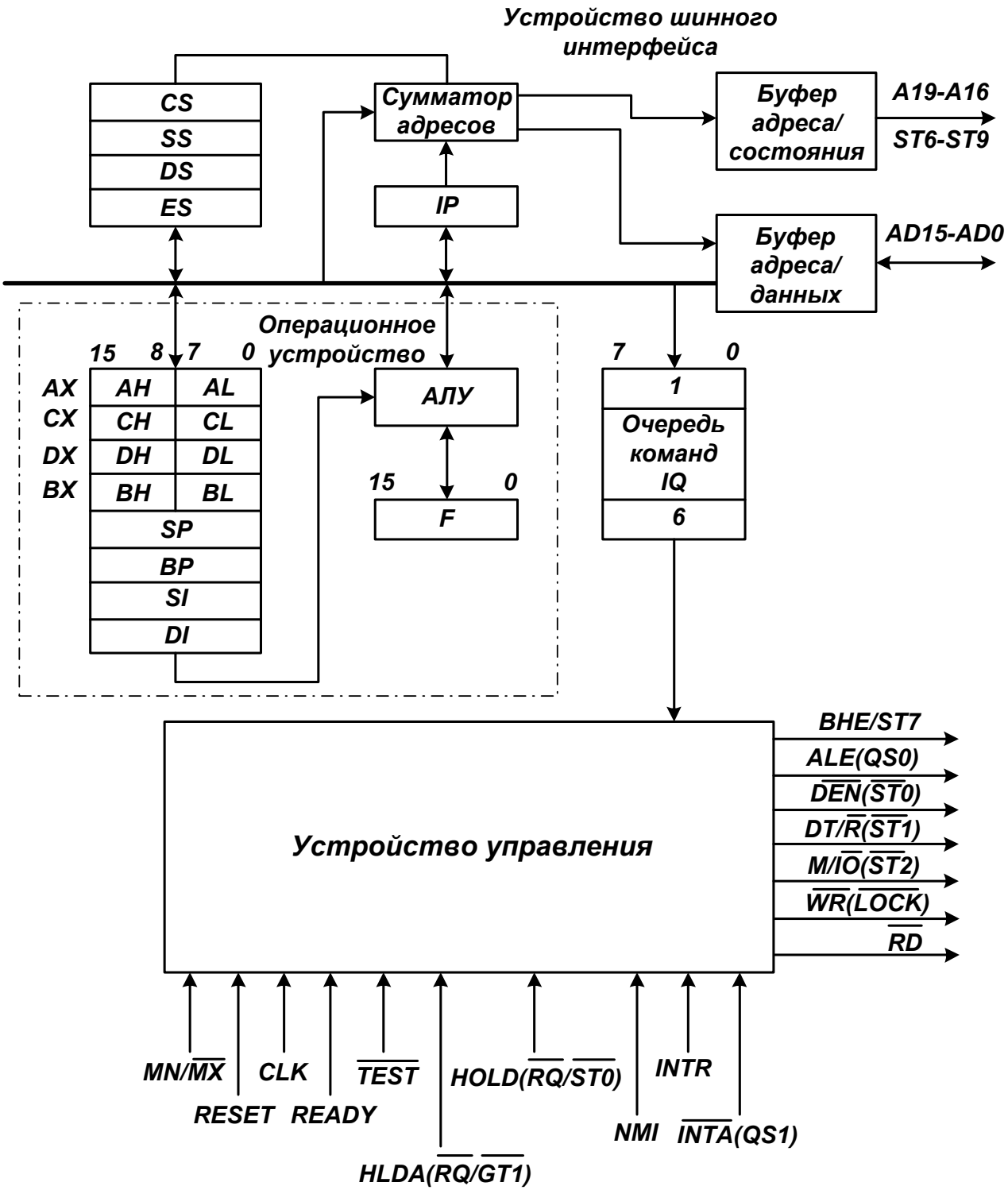


Рис. 2.20. Структурная схема МП ВМ86

Все остальные регистры являются неделимыми и оперируют 16-битовыми словами, даже в случае использования только старшего или младшего байтов. Указательные регистры SP и BP хранят смещение адреса в пределах текущего стекового сегмента памяти, а индексные регистры SI и DI хранят смещение адреса соответственно в текущем сегменте данных и в текущем дополнительном сегменте. Однако при использовании этих регистров для адресации операндов возможна смена сегментов памяти.

Арифметическо-логическое устройство (АЛУ) содержит 16-битовый комбинационный сумматор, с помощью которого выполняются арифметические операции, наборы комбинационных схем для выполнения логических операций, схемы для операций сдвигов и десятичной коррекции, а также регистры для временного хранения операндов и результатов.

К АЛУ примыкает регистр флагов F. Его младший байт FL полностью соответствует регистру флагов К580ВМ80, а старший байт FH содержит четыре флага, отсутствующие в К580ВМ80. Шесть арифметических флагов фиксируют определенные признаки результата выполнения операции (арифметической, логической, сдвига или загрузки регистра флагов). Значения этих флагов (кроме флага AF) используются для реализации условных переходов, изменяющих ход выполнения программы. Различные команды влияют на флаги по-разному.

Назначение арифметических флагов.

CF — флаг переноса, фиксирует значение переноса, возникающего при сложении (вычитании) байтов или слов, а также значение выдвигаемого бита при сдвиге операнда.

PF—флаг четности (или паритета), фиксирует наличие четного числа единиц в младшем байте результата операции, может быть использован, например, для контроля правильности передачи данных.

AF — флаг вспомогательного переноса, фиксирует перенос (заем) из младшей тетрады, т. е. из бита аз, в старшую при сложении (вычитании), используется только для двоично-десятичной арифметики, которая оперирует исключительно младшими байтами.

ZF — флаг нуля, сигнализирует о получении нулевого результата операции.

SF — флаг знака, дублирует значение старшего бита результата, который при использовании дополнительного кода соответствует знаку числа.

OF — флаг переполнения, сигнализирует о потере старшего бита результата сложения или вычитания в связи с переполнением разрядной сетки при работе со знаковыми числами. При сложении этот флаг устанавливается в единицу, если происходит перенос в старший бит и нет переноса из старшего бита или имеется перенос из старшего бита, но отсутствует перенос в него; в противном случае флаг OF устанавливается в нуль. При вычитании он устанавливается в единицу, когда возникает заем из старшего бита, но заем в старший бит отсутствует либо имеется заем в старший бит, но отсутствует заем из него. Имеется специальная команда прерывания при переполнении, которая в указанных случаях генерирует программное прерывание.

Для управления некоторыми действиями МП предназначены три дополнительных флага.

DF — флаг направления, управляемый командами CLD и STD; определяет порядок обработки цепочек в соответствующих командах: от меньших адресов (DF=0) или от больших (DF==1).

IF— флаг разрешения прерываний, управляемый с помощью команд CL1 и STI; при IF==1 макропроцессор воспринимает (распознает) и соответственно реагирует на запрос прерывания по входу INTR; при IF=0 прерывания по этому входу запрещаются (маскируются) и МП игнорирует поступающие запросы прерываний. Значение флага IF не влияет на восприятие внешних немаскируемых прерываний по входу NMI, а также внутренних (программных) прерываний, выполняемых по команде INT.

TF—флаг трассировки (прослеживания). При TF=1 МП переходит в покомандный (пошаговый) режим работы, применяемый при отладке программ, когда автоматически генерируется сигнал внутреннего прерывания типа 1 после выполнения каждой команды с целью перехода к соответствующей подпрограмме, которая обычно обеспечивает индикацию содержимого внутренних регистров МП. Команды установки или сброса флага TF отсутствуют, так что управление этим флагом осуществляется опосредованно, путем пересылки содержимого регистра флагов F через стек в общий регистр, установки требуемого значения восьмого бита и обратной пересылки

сформированного слова в регистр F.

FH								FL							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

Рис. 2.21. Формат регистра флагов F

Управляющее устройство (УУ) дешифрует команды, а также воспринимает и вырабатывает необходимые управляющие сигналы. В его состав входит блок микропрограммного управления, в котором реализовано программирование МП на микрокомандном уровне.

Устройство шинного интерфейса (или просто шинный интерфейс) содержит блок сегментных регистров, указатель команд, сумматор адресов, очередь команд и буферы, обеспечивающие связь с шиной. Шинный интерфейс выполняет операции обмена между МП и памятью или портами ввода — вывода по запросам операционного устройства. Когда операционное устройство занято выполнением команды, шинный интерфейс самостоятельно инициирует опережающую выборку кодов очередных команд из памяти.

Очередь команд представляет собой набор байтовых регистров и выполняет роль регистра команд, в котором хранятся коды, выбранные из программной памяти. Длина очереди составляет 6 байт, что соответствует максимально длинному формату команд. Наличие очереди команд, а также способность операционного устройства и шинного интерфейса работать параллельно позволяют совместить во времени фазы выборки команды и выполнения заданной операции: пока одна команда исполняется в операционном устройстве, шинный интерфейс осуществляет выборку следующей команды. Таким образом достигаются высокая плотность загрузки шины и повышение скорости выполнения программы. Пример, иллюстрирующий реализацию описанного конвейерного принципа, дан на рис. 2.22, где TI обозначает холостые такты работы шины, когда очередь команд заполнена, а операционное устройство занято выполнением текущей команды и не запрашивает выполнения цикла шины.

Шинный интерфейс инициирует выборку следующего командного слова автоматически, как только в очереди освободятся два байта. Как правило, в очереди находится минимум один байт потока команд, поэтому операционное устройство не ожидает выборки команды. Ясно, что опережающая выборка команд позволяет экономить время только при естественном порядке выполнения команд. Когда операционное устройство выполняет команду передачи управления (перехода) в программе, шинный интерфейс сбрасывает очередь, выбирает команду по новому адресу, передает ее в операционное устройство, а затем начинает заполнение очереди из следующих ячеек памяти. Эти действия предпринимаются в условных и безусловных переходах, вызовах подпрограмм, возвратах из подпрограмм и при обработке прерываний.

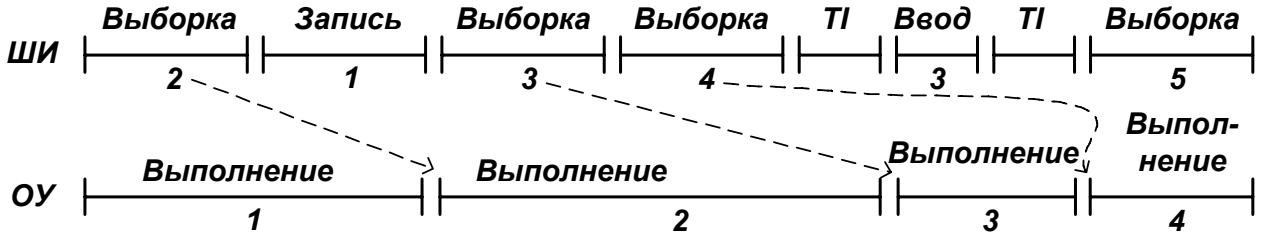


Рис. 2.23. Пример конвейерного выполнения команд

По мере необходимости операционное устройство считывает байт из очереди и выполняет предписанную командой операцию. При многобайтовых командах из очереди считываются и другие байты команды. В тех редких случаях, когда к моменту считывания очередь оказывается пустой, операционное устройство ожидает выборку очередного командного слова, которую инициирует шинный интерфейс. Если команда требует обращения к памяти или порту ввода — вывода, операционное устройство запрашивает шинный интерфейс на выполнение необходимого цикла шины для передачи данных. Когда шинный интерфейс не занят выборкой команды, он удовлетворяет запрос немедленно; в противном случае операционное устройство ожидает завершения текущего цикла тины. Со своей стороны, шинный интерфейс приостанавливает выборку команд во время обмена данными между операционным устройством и памятью или портами ввода — вывода.

Буфер шины адреса/данных (БАД) содержит 16 двунаправленных управляемых усилителей с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий AD15—ADO.

Буфер шины адреса/состояния (БАС) содержит четыре однонаправленных усилителя с тремя выходными состояниями и обеспечивает номинальную нагрузочную способность линий A19/S6—A16/S3.

Сегментные регистры хранят базовые (начальные) адреса сегментов памяти: кодового сегмента CS, в котором содержится программа; стекового сегмента SS; сегмента данных DS; дополнительного сегмента ES, в котором обычно содержатся данные. Наличие сегментных регистров обусловлено разделением памяти на сегменты и используемым способом формирования адресов памяти. Хотя МП имеет 20-битовую шину физического адреса памяти, он оперирует 16-битовыми логическими адресами, состоящими из базового адреса сегмента и внутрисегментного смещения. Внутрисегментное смещение может быть вычислено в соответствии с указанным в команде способом адресации, может находиться в формате команды или содержаться в общем регистре. Физический адрес формируется путем суммирования смещения и содержимого соответствующего сегментного регистра, которое дополняется четырьмя нулевыми младшими разрядами.

Сумматор адресов осуществляет вычисление 20-битовых физических адресов.

Указатель команд IP хранит смещение следующей команды в текущем кодовом сегменте, т. е. указывает на следующую по порядку команду. Он является аналогом стандартного программного счетчика с той лишь разницей, что его содержимое определяет адрес команды лишь в совокупности с содержимым регистра CS; если же CS заполнен нулями, аналогия становится полной. Модификация IP осуществляется шинным интерфейсом так, что при обычной работе IP содержит смещение того командного слова, которое шинный интерфейс будет выбирать из памяти. Оно не совпадает со смещением очередной команды (находящейся в этот момент на выходе очереди команд), которую будет выполнять операционное устройство. Поэтому при запоминании содержимого IP в стеке, например при вызове подпрограмм, оно автоматически корректируется, чтобы адресовать следующую команду, которая будет выполняться. Эта особенность является следствием опережающей выборки команд, реализованной в BM86. Непосредственный доступ к IP имеют команды передачи управления.

Раздел 3. Организация ЭВМ

Тема 3.1 Архитектура ЭВМ

ЭВМ (компьютер) – это совокупность технических средств, создающая возможность проведения обработки данных и получения результата в необходимой форме, основные функциональные устройства которой выполнены на электронных компонентах (ГОСТ 15971-90).

Вычислительная система (ВС) - это совокупность ЭВМ и программного обеспечения, предназначенная для организации вычислительного процесса (последовательности событий решения прикладной задачи).

Процессор - это устройство или функциональная часть цифровой ВС, предназначенная для интерпретации программ (ГОСТ 15971-84).

Процессор - это функциональная единица, которая распознает и выполняет команды (ISO 2382/10 79).

Организация ЭВМ - это структура основных управляющих и информационных связей между основными устройствами, узлами и блоками ЭВМ, обеспечивающая выполнение заданных функций.

Архитектура ЭВМ – это концепция взаимосвязи элементов и частей ЭВМ, включающая компоненты логической, физической и прграммной структур.

История развития архитектуры ЭВМ

Поколения, годы	Элементная база		Аппаратные средства	Программные средства
	логические элементы	элементы памяти		
I 1945-1954 IAS (Нейман), УРАЛ, БЭСМ	вакуумные лампы	ультразвуковые линии задержки	операционный блок с фиксированной запятой, управляющий блок с жёсткой логикой	машинный язык, ассемблер, неразвитые средства ОС
II 1955-1964 БЭСМ-6, МИНСК-32	транзисторы	магнитные сердечники	операционный блок с плавающей запятой, индексный регистр	языки высокого уровня (Фортран), библиотеки подпрограмм, монитор пакетного режима
III 1965-1974 ЕС-IBM (IBM/360), СМ-ЭВМ (PDP-4)	МИС, СИС (ТТЛ)	магнитные сердечники, интегральные схемы статической памяти	микропрограммное управление, конвейерная обработка, кэш-память, микропроцедуры	мультипрограммирование, ОС с разделением времени, виртуальная память, структурное программирование

Поколения, годы	Элементная база		Аппаратные средства	Программные средства
	логические элементы	элементы памяти		
III,5 1975-1984 Эльбрус 2, IBM PC/XT	БИС (ТТЛШ)	БИС динамической памяти	параллелизм, мультиплексный конвейер, матричные и другие специализированные системы, специализированные процессоры, мультипроцессорные системы	параллельное программирование, виртуальные машины, СУБД, исскуственный интеллект (Prolog)
IV 1985-?	СБИС (КМОП, БиКМОП)	СБИС динамической памяти	компьютерные сети, автоматизированное рабочее место, нетрадиционная архитектура, аппаратная поддержка для языков высокого уровня и ОС	объектно-ориентированное программирование, виртуальные устройства, распределённые ОС и СУБД, искусственный интеллект (экспертные системы)

Классификация ЭВМ по Когану

1. ЭВМ общего назначения:
   - многопользовательские системы (mainframe);
   - серверы глобальной компьютерной сети (site).
   - (site занимается удаленным доступом к данным, а mainframe занимается еще удаленным выполнением программ).
2. Малые ЭВМ:
   - специализированные ЭВМ;
   - автоматизированные рабочие места.
3. Мини ЭВМ:
   - персональные компьютеры;
   - notebook.
4. МикроЭВМ:
   - карманные ЭВМ;
   - PDA, Palm.

Структурная организация ЭВМ

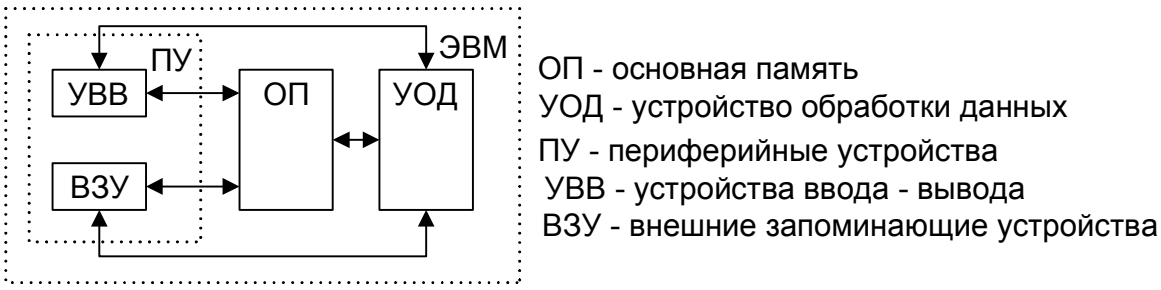


Рис. 3.1. Структурная организация ЭВМ

Тема 3.2 Системный интерфейс

Понятие интерфейса

Рассматривая ЭВМ, как класс устройств, мы пришли к такому выводу, что наша ЭВМ изначально содержит, помимо той упрощенной схемы, которую мы рассматривали раньше, еще некие дополнительные устройства. Первоначально схема была следующей:

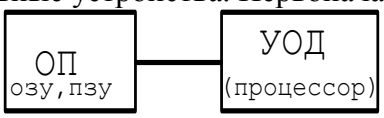


Рис. 3.2. Первоначальная схема интерфейса

Имеется устройство обработки данных (УОД) или процессор. Процессор обращается к основной памяти, которая представляется в виде ОЗУ или ПЗУ или программируемого ЗУ, главное чтобы эти устройства имели линейную организацию памяти. Основная память (ОП) – это память, где хранятся команды и данные для устройства обработки данных, а ОЗУ - это узел этой памяти. Нам потребуется класс устройств, называемых периферийные устройства (ПУ), которые позволяют нам подготовить программу и данные, необходимые для работы процессора. ПУ являются интерфейсом между ОП и теми объектами, для которых предназначены данные, получи в результате счета. ПУ в свою очередь разделяются на 2 части:

Внешние запоминающие устройства (ВЗУ) - служат для долговременного хранения программ и данных.

Устройства ввода-вывода (УВВ) - служат для связи человека и управляемого объекта с вычислительным устройством.

Основой для обработки данных является ОП и УОД. Это машина Тьюринга. Но на ленту машины Тьюринга надо что-то записать, а потом данные как-то прочитать. Прочитать мы не умеем непосредственно из памяти, и записывать туда мы тоже не умеем (у нас нет магнитных органов или электрических органов, которые позволяют туда что-то записать с такой точностью). Поэтому к совокупности ОП и УОД добавляются ПУ. Причем внешним интерфейсом являются УВВ. Поэтому под ЭВМ будем понимать следующую схему:

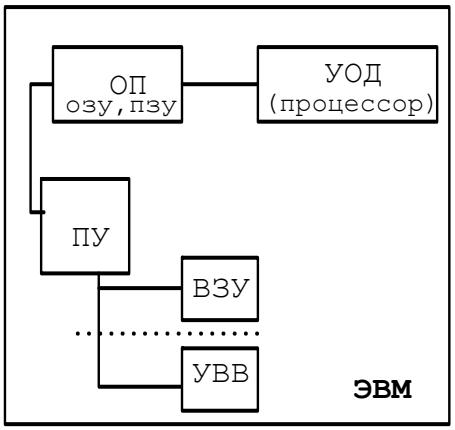


Рис. 3.3. Схема ЭВМ

Возникают следующие проблемы: это устройство универсально-моделирующее, называемое ЭВМ. Мы его декомпозировали на 3 части, в результате чего у нас есть связи разнородных объектов и ясно, что в результате декомпозиции архитектурной мы должны сказать об интерфейсе, то есть, как нам подключать эти устройства. Понятно, что для каждого типа устройств можно было разработать различные схемы такого рода связи. Хотелось бы это каким-то образом унифицировать, чтобы была некая взаимозаменяемость. Неважно, каковы характеристики ВЗУ и какова его модель, важно, что он вставляется в соответствующий разъем или подключается к соответствующей ЭВМ независимо от того, кем изготовлен, когда изготовлен, как изготовлен. Это хорошее архитектурное решение, к которому надо стремиться. Исходя из таких представлений, нам необходимо унифицировать интерфейс и это унифицирование осуществляется на различных уровнях по-разному. Есть унифицированные интерфейсы для взаимосвязи ПУ-ОП-УОД**,** которые называются системными интерфейсами, а есть унифицированные интерфейсы для подключения ВЗУ и УВВ, называемые интерфейсы ПУ.

Почему мы подключили ПУ к ОП? К УОД мы ПУ подключить не можем, потому что процессор работает только с ОП. Значит УОД каким-то образом и будет работать с ПУ**,** но только через ОП, а более точно через линейное адресное пространство, которое для УОД является его ОП.

Существует проблема обеспечения взаимодействия функциональных (структурных) элементов сложной системы (ЭВМ), заключающаяся в необходимости обеспечения локального дистанционного взаимодействия элементов и унификации принципов взаимодействия с целью повышения эффективности.

Проблема обеспечения взаимодействия функциональных (стандартных) элементов в системе

Виды связи между структурными и функциональными частями ЭВМ:

Информационные. Используются для обеспечения согласованного взаимодействия функциональных элементов в соответствии с совокупностью логических условий, определяющих в целом структурную и функциональную организацию системы (кодирование алфавитов данных, синхронизация работы в автоматной сети).

Электрические или энергетические. Определяют согласованность параметров электрических сигналов в системе с учетом ограничений на пространственное размещение элементов (представление знаков алфавитов в виде сигналов, синхронизация работы с учётом времени распространения сигналов).

Конструктивные. Определяют согласованность взаимодействия устройств с учётом пространственного и конструктивного размещения.

Интерфейс – это совокупность физических и логических принципов взаимодействия элементов сложной структуры, где под логическим, в данный момент, понимается

обеспечение информационной совместимости, а под физическим – электрическая и конструктивная совместимость.

Стандартный интерфейс – совокупность унифицированных технических, программных и конструктивных средств, необходимых для реализации взаимодействия функциональных элементов в системах обработки данных при условиях предписанных стандартом и направленных на обеспечение информационной, электрической и конструктивной совместимости.

Основные характеристики интерфейса

Пропускная способность - это количество единиц информации, передаваемых между элементами в единицу времени [Бит/с]. Различают максимальную, минимальную и номинальную пропускную способность.

Замечание: Единица измерения **Бод** никакого отношения к пропускной способности не имеет. **Бод** - это скорость манипуляции в канале. Она связана с пропускной способностью, но никак ее не определяет. Скорость манипуляции - это число изменений параметров сигнала в единицу времени.

Вместимость интерфейса (конструктивная характеристика) - максимальное число подключаемых элементов.

Топология интерфейса:

- двухточечная (радиальная) – это взаимодействие двух устройств через выделенную этим двум устройствам среду;

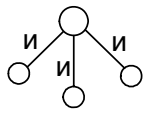


Рис. 3.4. Радиальная топология интерфейса

- многоточечная (магистральная) – это взаимодействие только двух устройств одновременно через общую для всех среду передачи данных;

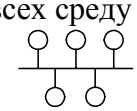


Рис. 3.5. Магистральная топология интерфейса

- шинная – это взаимодействие одного устройства со всеми через общую среду передачи данных.

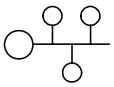


Рис. 3.6. Шинная топология интерфейса

Каждая из этих топологий делится по способу передачи данных на последовательные и параллельные.

Режимы работы:

- синхронный – каждое из устройств имеет свой временной или пространственный интервал, когда нужно передавать данные.
- асинхронный – каждое из устройств путем взаимодействия с другими устройствами выбирает пространственный или временной интервал передачи данных.

Способы адресации устройств в многопоточных интерфейсах:

- логический – адрес устройства задаётся в виде номера, из которого интерфейс, как устройство, определяет его физический адрес;
- географический – устройство адресуется по месту подключения к интерфейсу.

4. **Протокол взаимодействия** – это строго заданная процедура или совокупность правил, определяющая способ выполнения определённого класса функций взаимодействия элементов сложной структуры.

Классификация интерфейсов ЭВМ

Системные интерфейсы: радиальные (ISA, MCA, EISA), совмещенные (PCI, IEEE-1196).
Интерфейсы программируемых подсистем: IEEE – 488, SCSI.
Интерфейсы внешних запоминающих устройств (ВЗУ):
- накопитель на гибком магнитном диске (ST-506);
- накопитель на жёстком магнитном диске (IDE, EIDE);
- накопитель на магнитной ленте или стример (QIC-36).

Интерфейсы устройств ввода/вывода (УВВ)**:** параллельные (Centronics), последовательные (RS232, USB).
Интерфейсы сетей общего пользования:
- телефонный канал (V.24, V.32, V.42, V.90);
- согласованная физическая линия (RS422);
- несогласованная физическая линия (RS423).

Интерфейсы локальных и глобальных компьютерных сетей:
- системы передачи данных общего пользования (X21,X25);
- локальные сети общего назначения (Ethernet IEEE-805.3);
- малые локальные сети (TokenRing).

История развития системных интерфейсов

)д	Интерфейс	Разрядность, бит	Частота, МГц	Пропускная способность, Мбайт/с
)81	XT-Bus	8	4,77	6 Мбит/с
)84	AT-Bus	16	6/8	12
)87	MCA (PS2)	16/32	10	12/24
)89	EISA	32	8	25
)92	VL-Bus	32	33	100
)93	PCI	32	33	132
)95	PCI-2	64	33/66	264/528

Архитектура ЭВМ с множественным интерфейсом

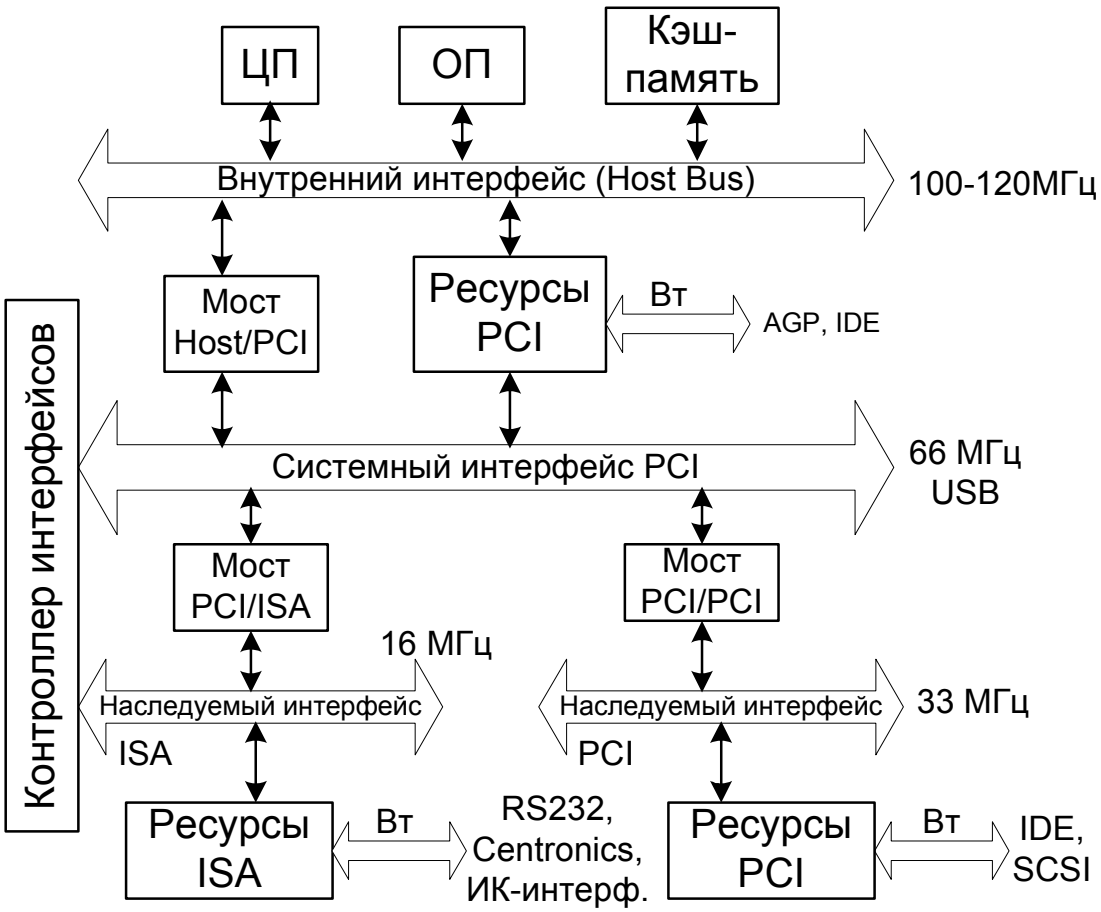


Рис. 3.7. Платформа с множественным интерфейсом

ЦП – центральный процессор;

Вт – вторичные интерфейсы, к которым подключаются периферийные устройства.

Host Bus – внутренний интерфейс, связывающий наиболее быстродействующие устройства системы. Имеет очень высокую пропускную способность, которая соответствует пропускной способности связи CPU-КЭШ. Кэш-память имеет время доступа 5-10нс. Динамическая память имеет время доступа 40нс. ЦП с ОП в современных системах вообще не работает: процессор не “знает”, что есть основная память, он “знает”, что есть кэш-память. Связь ОП с внутренним интерфейсом необходима для того, чтобы в эту память можно было добавить какие-то данные из ВЗУ или УВВ. Понятно, что невозможно обеспечить колоссальную пропускную способность, например 400 МгБ/с, работая на клавиатуре, поэтому для таких менее быстродействующих устройств необходимо предусмотреть другие интерфейсы, согласование между которыми делается при помощи специальных устройств, которые называются мосты (Bridge - устройство сопряжения двух интерфейсов).

Оказывается интерфейс PCI довольно скоростной с пропускной способностью 132-528 Мбит/с, с тактовой частотой 66 МГц, с 64 разрядами. Понятно, что за 5-10нс мы должны осуществить процесс передачи данных. Но реально мы не можем бесконечно повышать тактовую частоту, потому что уже на расстоянии 10см сказываются эффекты конечной скорости распространения сигналов в системе. Оказывается, что такая пропускная способность очень большая для 90% случаев применений. Поэтому основной интерфейс разделился на 2 части: более быстродействующий и менее быстродействующий.

PCI с тактовой частотой 33МГц – наследуемый интерфейс, с пропускной способностью 132Мбит/с. К этой шине подключаются медленные устройства.

AGP – интерфейс для подключения графических устройств.

IDE никогда не подключается к внутреннему интерфейсу.

Самые медленные устройства подключаются к ISA.

Тема 3.3 Системный интерфейс PCI

PCI – Peripheral Component Interconnect (Соединение периферийных компонент)

Группы сигналов интерфейса PCI

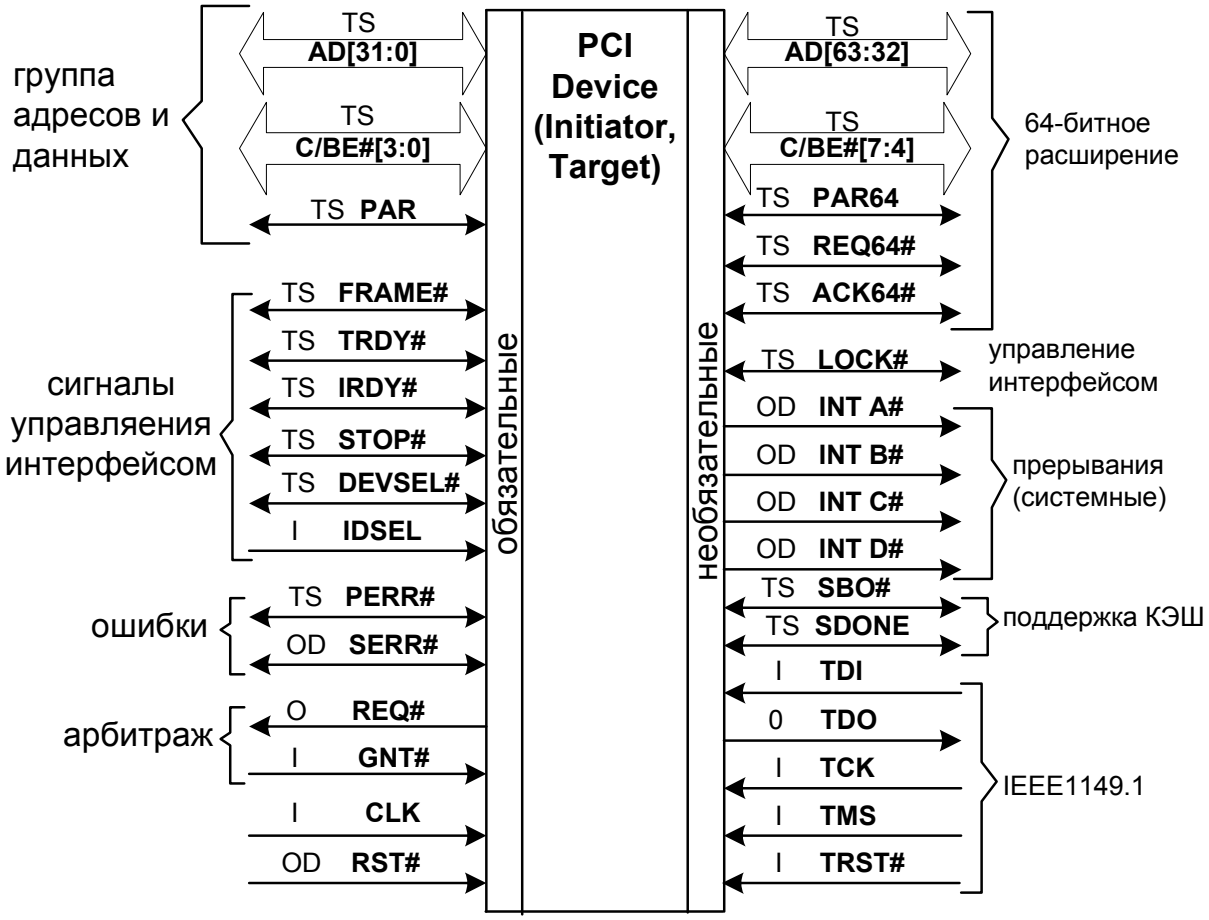


Рис. 3. 8. Интерфейсная микросхема PCI

TS – трехстабильное состояние;
I – вход;
O – выход;
OD (Open Drive) – открытый коллектор;
- сигнал активен низким уровнем.

Описание обязательных сигналов интерфейса PCI

AD[31÷0] (Address/Data) – ***мультиплексируемые линии для конвейерной передачи адреса и данных. Сигналы активны по переднему фронту тактового сигнала и на всем его протяжении.***

C/BE[3÷0] (Command/Byte Enable). ***В начале части цикла обмена передаётся тип обмена или команды.***

Таблица 3.1. Циклы системного интерфейса PCI

C/BE[3÷0]	**Тип команды**
0000	**Interrupt Acknowledge** – ответ на прерывание
0001	**Special Cycle** - специальный цикл
0010	**I/O Read** – чтение порта ввода/вывода
0011	**I/O Write** – запись в порт ввода/вывода
0100	Зарезервировано
0101	Зарезервировано
0110	**Memory Read** – чтение памяти
0111	**Memory Write** – запись в память
1000	Зарезервировано
1001	Зарезервировано
1010	**Configuration Read** – конфигурационное чтение
1011	**Configuration Write** – конфигурационная запись
1100	**Multiple Memory Read** – множественное чтение памяти
1101	**Dual Address Cycle** – двухадресный цикл
1110	**Memory-Read Line** – чтение строки памяти
1111	**Memory Write and Invalidate**– запись в память и аннулирование

Интерфейс PCI поддерживает 3 группы адресуемых объектов:

- адресное пространство ввода/вывода;
- адресное пространство памяти;
- конфигурационное адресное пространство.

Во второй части цикла обмена интерфейса PCI передаются разряды разрешения байт, которые определяют позиции байт на интерфейсе по разрядам.

PAR (паритет) ***– бит контроля. Дополняет число единиц в группе AD до нечётного.***

FRAME# ***- идентефикация начала цикла системного интерфейса. Как правило, когда он активен, передаются адрес и команда в группе адрес/данные***

TRDY# (Target Ready) ***- готовность целевого устройства.***

IRDY# (Initiator Ready) ***- готовность инициатора.***

На интерфейсе PCI имеется 2 типа устройств: инициатор (активное) и цель (пассивное). Активное устройство управляет обменом по интерфейсу PCI. Пассивное устройство участвует в этом обмене. Как активное, так и пассивное устройство могут принимать или передавать данные.

STOP# ***- Ресурс (устройство) требует завершение цикла обмена преждевременно. Выдаётся пассивным устройством, когда оно не в состоянии завершить обмен.***

DEVSEL# (Device Select) ***- сигнал выдается, когда целевое устройство декодировало адрес и претендует на цикл обмена по интерфейсу. То есть, устройство сообщает, что по тому адресу, который был выставлен до начала обмена, может быть осуществлен обмен, потому что этот адрес опознан. Но здесь есть тонкость. Допустим у нас есть 4 разъема интерфейса PCI. Можно вставить в эти разъемы устройства, которые откликаются на один и тот же адрес. Поэтому устройство обрабатывает этот сигнал реально столько, сколько разъемов. Значит, возможно, что несколько устройств заявят о том, что они претендуют на цикл системного интерфейса.***

IDSEL (Initialization Device Select) ***– выдается активным устройством при доступе в конфигурационное адресное пространство. Является результатом дешифрации разрядов адреса AD[31÷11]. 10 разрядов, которые не подвергаются дешифрации, фактически, не участвуют в формировании этого сигнала. Это означает, что все***

конфигурационное адресное пространство разбивается на 2^{20} частей, каждая размером по 2048 байт.

Сигналы ошибок:

- **PERR# (Parity Error)** – сигнал об ошибке паритета. Возникает в фазе передачи данных. Вырабатывается устройством, принимающим данные.
- **SERR# (System Error)** – системная ошибка. Вырабатывается контроллером.

Сигналы арбитража:

- **REQ# (Request)** – сигнал запроса системного интерфейса. Каждый инициатор имеет свой сигнал запроса. Этот сигнал используется для запроса цикла системного интерфейса, так как системный интерфейс – это разделяемое устройство (возможна только одна передача данных в каждый момент времени).
- **GNT# (Grant)** - каждый инициатор имеет свой сигнал GNT#, который сигнализирует о том, что интерфейс предоставлен запросившему его активному устройству.

RST# (Reset) ***– сброс (инициализация) всех устройств, подключенных к системному интерфейсу (кнопка RESET).***

CLK (Clock) ***– сигнал тактовой частоты. Интерфейс PCI синхронный, поэтому все сигналы воспринимаются (стробируются) при переходе сигнала CLK от низкого уровня к высокому.***

Описание необязательных сигналов интерфейса PCI

Сигналы 64-битного расширения:

- **REQ64# -** выставляется инициатором для осуществления 64-х битного обмена в адресной фазе цикла и остается активным в течение фазы данных.
- **ACK64#** - подтверждение 64-битного обмена. Выставляется целевым устройством для указания того, что оно поддерживает расширенный обмен данными. Если целевое устройство не поддерживает такой обмен, то инициатор должен перестроиться на 32-битный обмен.
- **AD[63÷32], C/BE[7÷4], PAR 64** – аналогичны соответствующим сигналам 32-битного обмена.

Сигналы поддержки кэш-памяти:

- **SDONE (Snoop Done)** – сигнал завершенности цикла слежения для текущей транзакции. Низкий уровень указывает на незавершенность цикла слежения за когерентностью памяти и кэш-памяти. Кэш-контроллер использует SDONE при сквозной записи
- **SBO# (Snoop Back Off)** – попадание текущего обращения к памяти абонента шины в модифицированную строку кэш-памяти. Используется только абонентами шины с кэшируемой памятью при алгоритме обратной записи (WB).

LOCK# ***- используется инициатором для блокирования доступа к памяти. Сигнал означает, что никакое другое устройство не может обращаться в адресное пространство памяти до снятия этого сигнала. Этот сигнал необходим для реализации механизмов взаимного исключения. Другой способ аппаратной реализации механизма взаимного исключения – использование цикла чтение/модификация/запись, как единого неделимого действия.***

INT# (Interrupt) ***- асинхронные сигналы прерывания, выдаваемые ресурсом для запроса на обслуживание процессором. Процессор отвечает выполнением цикла подтверждения прерывания, в котором устройство передаёт номер вектора прерывания.***

*Стандарт IEEE 1149.1****. - это совокупность сигналов последовательного интерфейса, предназначенных для тестирования линий системного интерфейса PCI. В эту группу входят следующие сигналы:***

- **TDI (Taste Date In)** – последовательные входные данные.

- **TDO (Taste Date Out)** – последовательные выходные данные.
- **TCK** –тактовая частота синхронизации обмена.
- **TMS (Taste Mode Select)** – выбор режима. Происходит запись в регистр команд, когда сигнал активен.
- **TRST#** – сброс системы тестирования.

Сигналы **TMS** и **TCK** определяют синхронную передачу или прием данных по контактам **TDI** и **TDO**.

Зачем нужен тестовый интерфейс?

Пусть есть устройство, которое работает на частоте 33МГц. Если взять обычную ТТЛ микросхему (например, инвертор) и соедините вход с выходом, то получится генератор. Обычно, отдельно стоящая микросхема порождает сигнал близкий к синусоидальному. Период такого сигнала для этой микросхемы равен 7-10 нс. Оказывается, что невозможно подключить внешнее устройство (типа осциллографа), который смог бы показать эти сигналы на системном интерфейсе, так как в результате такого подключения будут внесены такие искажения в процесс работы системного интерфейса, что он не будет работать. Следовательно, нет средств подключиться к интерфейсу для наблюдения сигналов. Поэтому и предусмотрен тестовый интерфейс внутри интерфейса PCI для его тестирования.

Циклы системного интерфейса PCI

Любой цикл системного интерфейса состоит из двух фаз: фазы адреса и фазы данных. Т.к. интерфейс синхронный, то длительность адресной фазы и фазы данных задается тактовым сигналом.

Системный интерфейс PCI может работать в 3 режимах:

1. IDLE - пассивный режим.
2. ADR - режим передачи адреса или командная часть цикла.
3. DATA - часть цикла связанная с обменом данных.

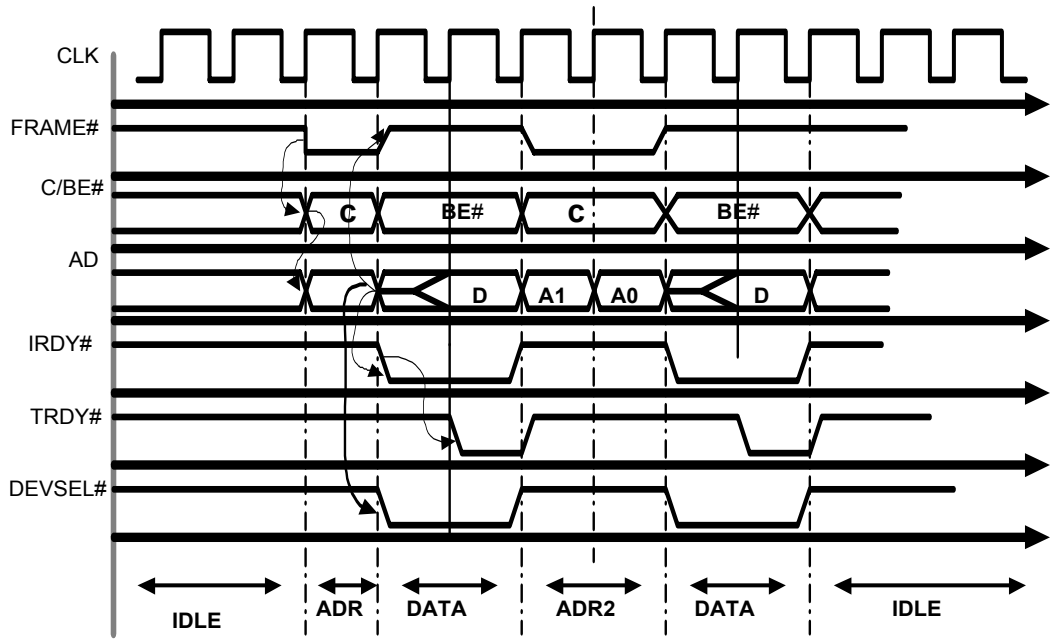


Рис. 3. 9. Диаграмма общего способа обмена данными по интерфейсу PCI

D – данные.

Первый режим - это пассивный (IDLE). На него отводится 2 такта.

При этом режиме: FRAME# – высокий уровень

C/BE# - безразличный уровень (или 0 или 1).

AD – безразличный уровень.

IRDY#, TRDY#, DEVSEL# - высокий уровень.

Цикл системного интерфейса начинается с сигнала FRAME#, поэтому сигнал фиксируется по переднему фронту вплоть до постоянного уровня. Значит, по переднему фронту происходит изменение всех сигналов так, чтобы, когда сигнал CLK станет вплоть до постоянного уровня, сигналы могут быть зафиксированы, то есть должны быть воспринимаемы на интерфейсе.

Под адресный режим (ADR) отводим 1 такт. Далее в этой части передается также еще и команда, то есть, какой тип цикла в данный момент будет выполняться. Причем команда передается ровно 1 такт, после этого передаются сигналы BE# и наступает фаза передачи данных (отводится 2 такта). Далее в AD выставляется адрес, который сменяется данными. При чтении данных данные выставляются позже, а при записи выставляются сразу.

Понятно, что при записи данных, данные сразу же готовы, а при чтении – выдали команду, выдали адрес; устройство должно иметь время сообразить каким образом, какие данные нужно передать – на это дается целый такт. Далее IRDY# сообщает о том, что он готов передать или принять данные, то есть сразу после адреса. TRDY# сообщает о своей готовности конечно позже. В данном случае с опозданием на 1 такт. Как только устройство дешифрировало адрес, то есть опознало, что идет обращение в его адресное пространство, оно выставляет сигнал DEVSEL#.

Далее идет цикл с двойным адресом (ADR2). В этом случае сигнал FRAME# ведет себя точно также и в AD передаются 2 порции адреса, если это 32 разрядный интерфейс - это 64 бита адреса передается, если 64 разрядный, то 128 бит, сначала младшая часть адреса (A0), потом старшая часть (A1). Далее все идет стандартным образом: 2 периода тактового сигнала идет на фазу передачи данных и происходит тоже самое. Потом идет IDLE.

Всего существует порядка 12 типов обмена по системному интерфейсу PCI.

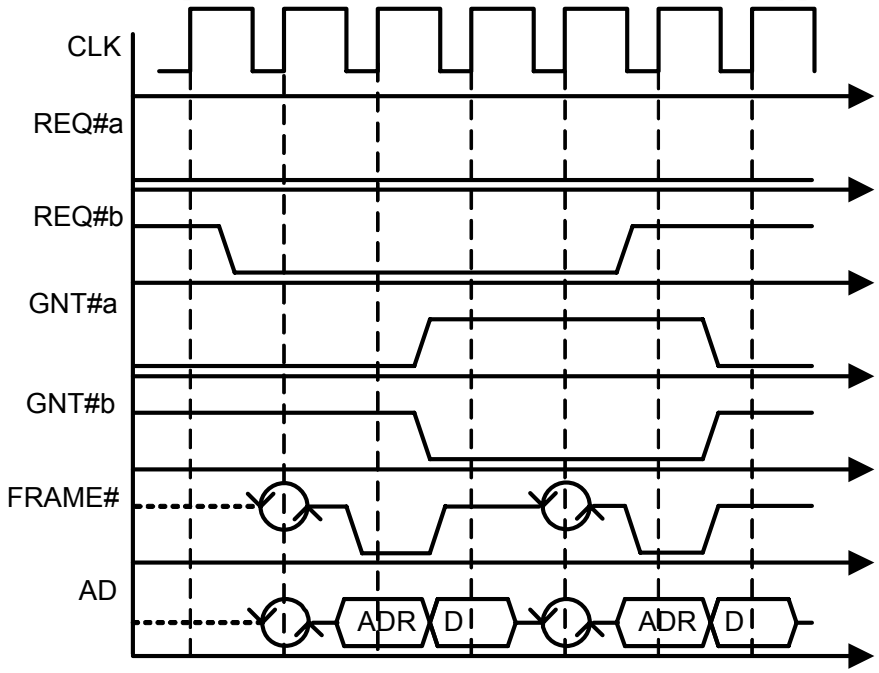


Рис. 3. 10. Протокол арбитража

ADR – адрес

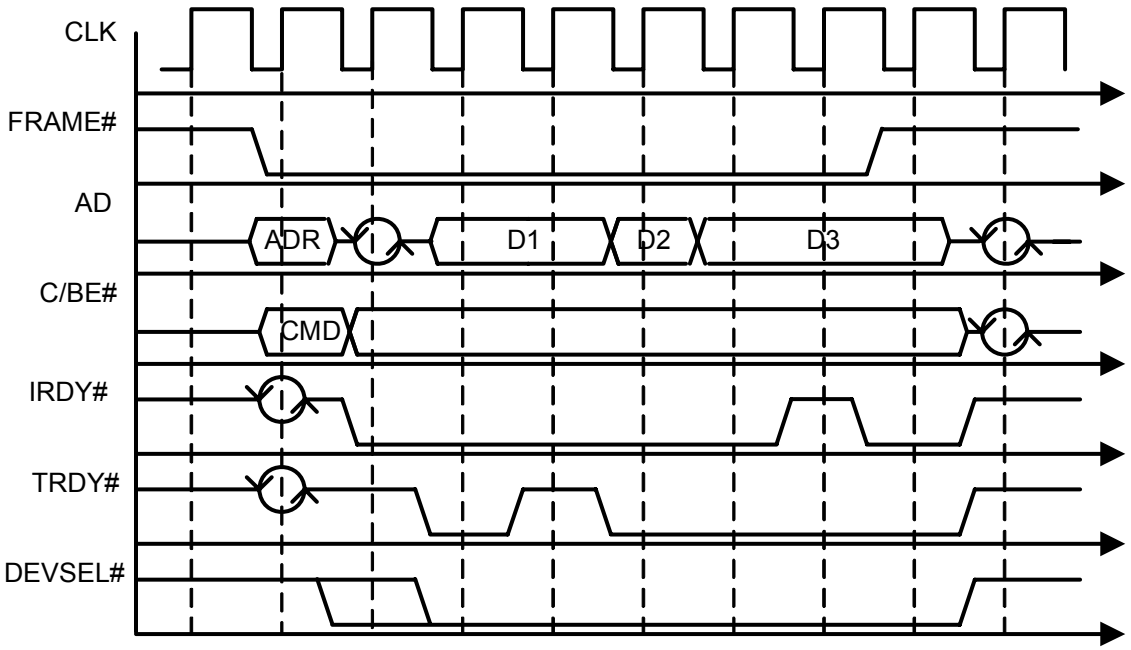


Рис. 3. 11. Цикл чтения

CMD – команда.

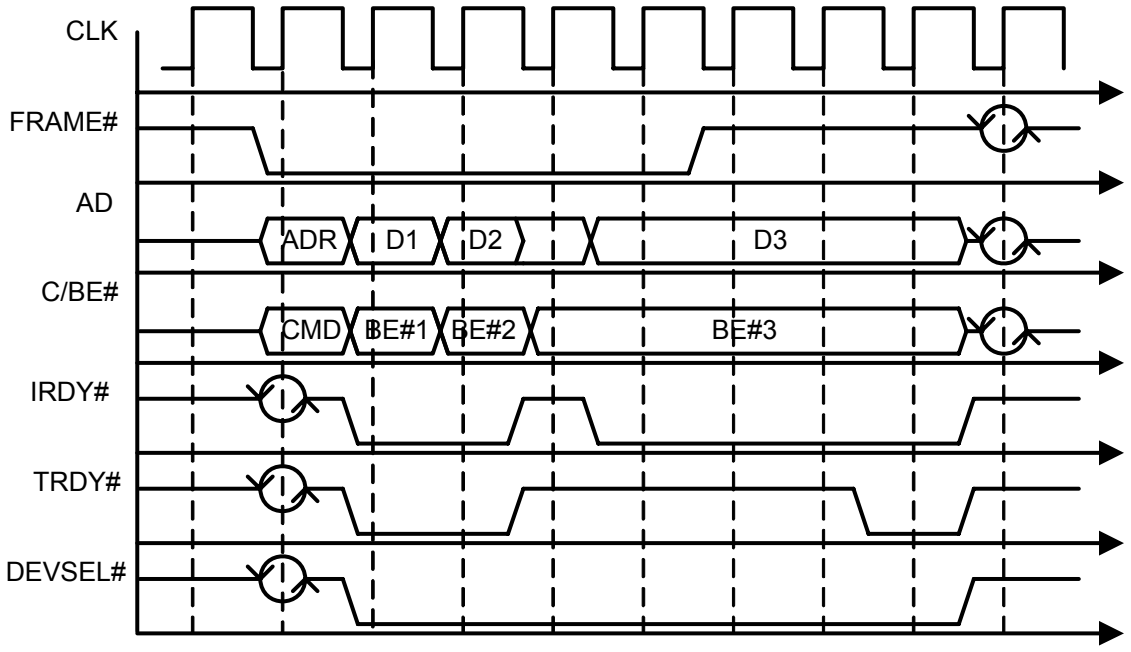


Рис. 3. 12. Цикл записи

Сигнал FRAME# сигнализирует о завершении транзакции на системной шине. В момент перехода в пассивное состояние передаётся последняя порция данных. На системном интерфейсе PCI выставляется адрес байта и увеличивается адрес.

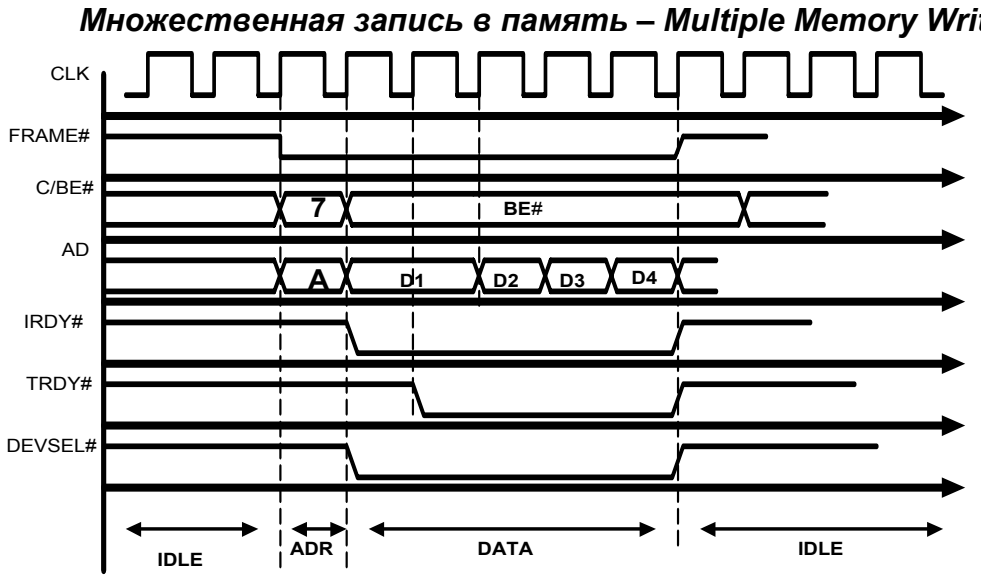


Рис. 3. 13. Множественная запись в память - Multiple Memory Write

Запись в память это 7-й номер обмена на системном интерфейсе. Он интересен в следующем: после выставления команды тут же выставляются данные, но так как это запись, инициатор уже имеет эти данные для того, чтобы их передавать. Понятно, что он при этом говорит, что он готов и выдал данные. Далее через некоторое время устройство отвечает, что оно эти данные приняло и предварительно сообщает, что оно найдено. Для памяти существуют 2 варианта чтения или записи: обычное (похожее на чтение ввода/вывода) и множественное. Управляет множественным доступом к системному интерфейсу сигнал FRAME#: если сигнал FRAME# инициатором не восстанавливается после адресной части цикла, то это означает, что режимом работы этого системного интерфейса является множественный обмен, то есть выставляются какие-то данные D1 и, как правило, чтобы завершился обмен между инициатором и ресурсом требуется не один, а два цикла, поэтому минимальная длительность этой части по передачи первой порции данных равна двум тактам. После этого, если сигнал FRAME# не установлен в исходное состояние и остался активным, считается множественная запись когда, не передавая адреса, выставляется следующая порция данных. Данные меняются каждый такт, пока сигнал FRAME# активен. Это позволяет увеличить на 30% эффективность передачи данных по интерфейсу. Оказывается, есть такая локальная парадигма, которая часто используется в области вычислительной технике, а именно принцип локальности – когда часто данные, которые требуются для следующего обмена передачи использования обработки, находятся рядом. Поэтому подразумевается, что мы передали начальную часть адреса массива, а дальше со смещением 0, со смещением 1 и так далее. Тоже самое происходит при множественной записи. Единственное отличие – данные появляются не сразу, а с опозданием. Подразумевается, что адреса увеличиваются на размер передаваемых данных, то есть адреса увеличиваются на 4 для 32 разрядного интерфейса и на 8 для 64 битного интерфейса.

Обработка сигналов PCI на платах расширения

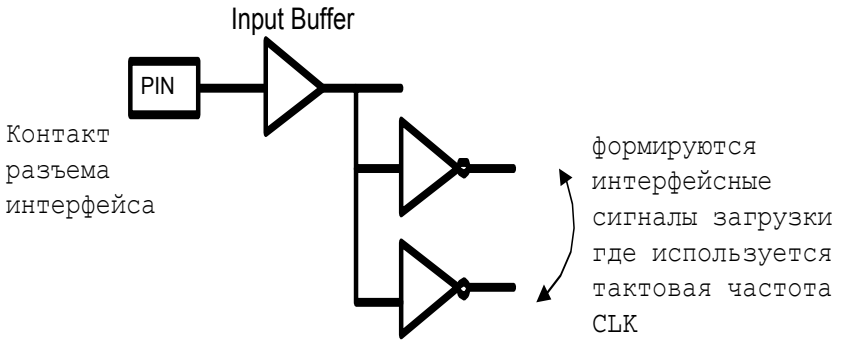


Рис. 3. 14. Схема обработки сигналов PCI

Input Buffer – входной буфер для обеспечения требуемой нагрузочной способности интерфейса.

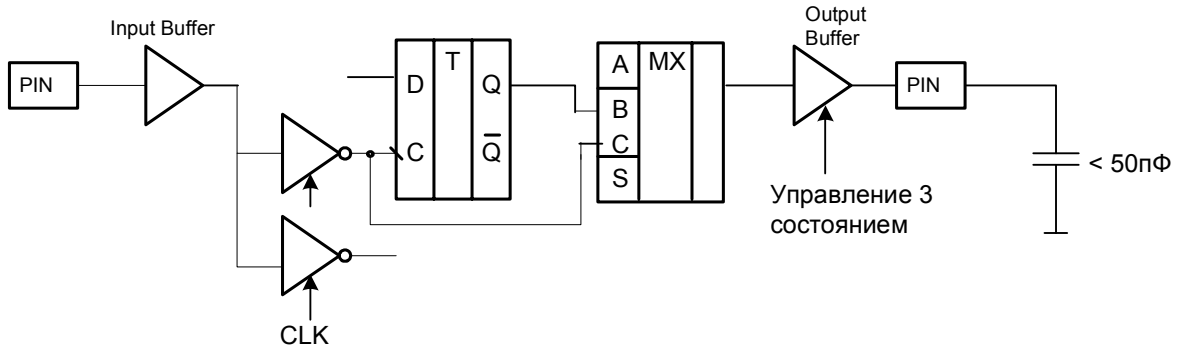


Рис. 3. 15. Схема обработки сигналов PCI

CLK – привязка информационного сигнала к тактовому.

Output Buffer – выходной буфер для обеспечения отключения и подключения к интерфейсу.

Триггер предназначен для запоминания состояния. Паразитная емкость всех подключений к PCI должна быть меньше 50пФ. Если будет больше, то происходит такое искажение импульсов, при котором они перестают выполнять свою роль, то есть обмен данных по интерфейсу PCI невозможен.

Компоновка контроллера PCI на платах расширения

Компоновка микросхем, которые взаимодействуют с сигналами интерфейса, должна делаться в виде БИС и только сигналы, идущие от мультиплексора могут быть вне БИС – выводится наружу. Учитывая эти ограничения, нельзя располагать микросхему контроллера в любом месте, а необходимо располагать её в непосредственной близости от контактов разъёма PCI. Причём сигналы, расположенные ниже пунктирной линии нельзя переносить в верхнюю часть.

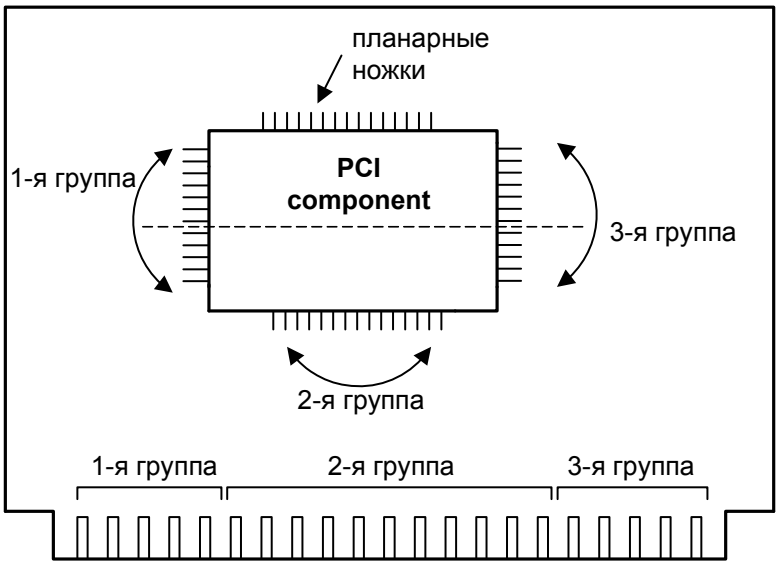


Рис. 3. 16. Компоновка контроллера PCI на платах расширения

1-я группа (сверху вниз)	2-я группа (слева направо)	3-я группа (сверху вниз)
JTAG RST# CLK GNT# REQ# - - - - - - - - - - AD[31÷24] C/BE[3]# IDSEL	AD[23÷16] C/BE[2]# FRAME# IRDY# TRDY# DEVSEL# STOP# LOCK# PERR# SERR# PAR C/BE[1]# AD[15÷8]	PAR64 AD[32÷63] C/BE[4÷7]# - - - - - - - - - - REQ64# ACK64# AD[0÷7] C/BE[0]#

Сигналы IRDY#, DEVSEL#, TRDY#, как правило, активны низким уровнем. Это связано с тем, что ток логического 0 в микросхемах шинных формирователей гораздо больше, чем ток логической 1.

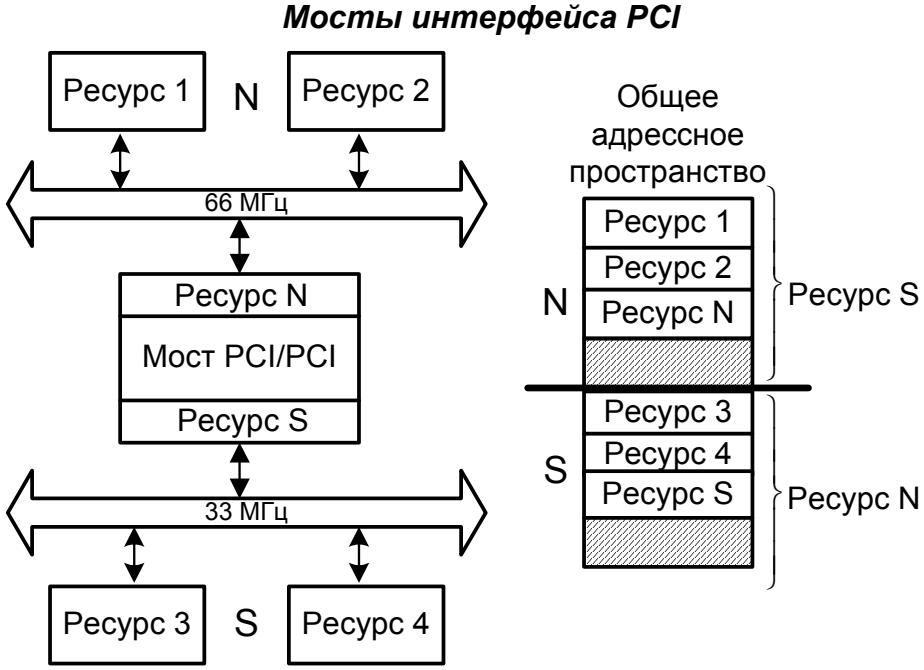


Рис. 3. 17. Мосты интерфейса PCI

Как правило, мосты конструктивно изготавливаются в контроллере интерфейса **PCI,** а он в свою очередь в **БИС,** которая интегрирует все компоненты какой-то вычислительной системы или же платформы.

Для того чтобы дать возможность ресурсу на северном интерфейсе обратиться к ресурсу на южно интерфейсе при использовании моста адресное пространство делится между интерфейсами. Ресурс N откликается на всё адресное пространство южного интерфейса, а ресурс S откликается на всё адресное пространство северного интерфейса

Возможно ли в одном цикле сразу обратится к интерфейсу вышележащего уровня? Эта проблема решена с помощью буферизированного обмена между различными уровнями интерфейса. Буферизированный обмен означает, что запрос на доступ

воспринимается от 1-го интерфейса, PCI component ждет завершения операции, мост получает запрос и ставит его в очередь, получив доступ ко 2-му интерфейсу, он повторяет этот запрос 2-му интерфейсу, выполняет обмен, ставит в очередь ответ, PCI component все еще ждет, а в этот момент 1-й интерфейс может использоваться для других обменов и только потом, когда очередь дойдет до нужного, отвечает ему.

Структура моста

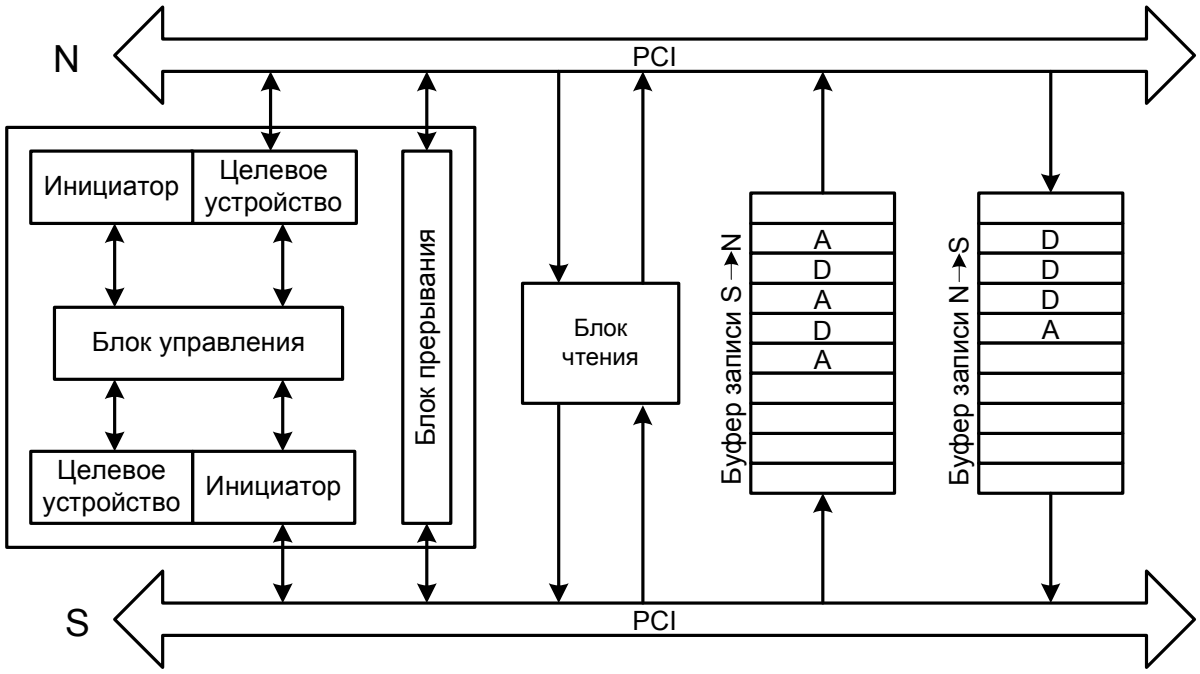


Рис. 3. 18. Структура устройства моста интерфейса PCI/PCI

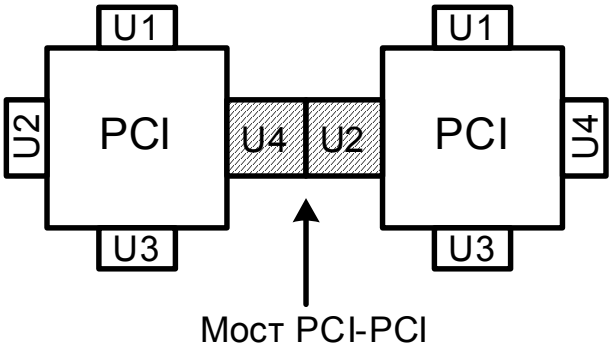


Рис. 3. 19. Представление моста PCI/PCI

Тема 3.4 Подсистема памяти

Принципы построения подсистемы памяти

Подсистема памяти ЭВМ – это совокупность технических средств, предназначенных для организации хранения, записи и чтения (доступа к данным).

Подсистема памяти должна позволять процессору иметь доступ к ячейкам для хранения команд и данных. Зная современные процессоры, можно заметить, что адресное пространство для такого рода процессоров колоссально по своему объему - это 2^{32} байт или 2^{64} байт. Известно, что чем больше объем устройства по объему хранимых данных, тем менее оно быстродействующее, а чем меньше объем, тем быстрее доступ к данным.

Оказалось что 99% адресного пространства процессора, которое можно заполнить - это память с очень низким быстродействием. Современный процессор - достаточно скоростное устройство, и поэтому медленный доступ к данным не даёт ему реализовать свои возможности. Выходом из этого положения стала виртуализация памяти, то есть применение таких методов управления доступом к памяти, которые позволяли бы:
наиболее часто используемы данные, хранить в более быстродействующей памяти; использовать блочную пересылку, когда за один такт обмена передается не одна порция данных, а много, и в конечном итоге, улучшая физические характеристики устройства.

Эволюция принципов построения подсистемы памяти

Логическое адресное пространство – адресное пространство, доступное программе (элементы адресуются через поля в коде команды).

Физическое адресное пространство – адресное пространство доступное процессору через системный интерфейс.

Адресация – это установление соответствия между множеством объектов ячеек памяти и множеством их адресов.

Рассмотрим этапы развития методов адресации:

1. ***В самом начале логическое адресное пространство равнялось физическому адресному пространству: ЛАП = ФАП, то есть адрес на внешнем интерфейсе процессора совпадал с тем адресом, который использовался при написании программ.***
2. ***ЛАП << ФАП. Это было связано с мультизадачностью и уменьшением длины команды. Реализуется с помощью базового метода адресации.***

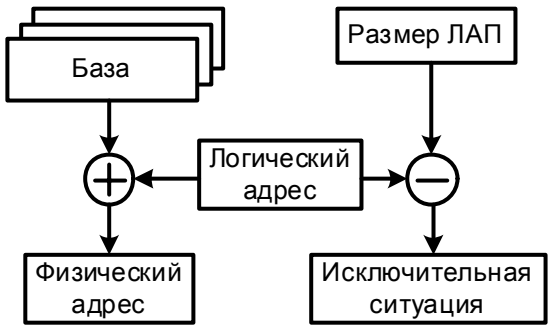


Рис. 3. 20. Базовый метод адресации

3. ***ЛАП >> ФАП. Было связано с тем, что задачи стали сложные и требовали большее число ячеек памяти, физическая память была дорогая. Этот метод реализуется с помощью виртуализации памяти.***
4. ***ЛАП >> ФАП, но объём физической памяти намного больше объёма физического адресного пространства. Этот метод реализуется с помощью страничной адресации памяти (EMS 4.0, XMS 2.0).***

Архитектурные методы повышения быстродействия памяти

1. **Блочная пересылка.** Чтение последовательно расположенных в памяти данных происходит быстрее, чем доступ к данным, которые расположены произвольно.
2. **Использование кэш-памяти**. Наиболее часто используемые данные оседают в наиболее быстродействующей памяти.
3. **Пакетный доступ.** Имеется 3 банка памяти с ячейками. На эти 3 банка подается один и тот же адрес. На выходе по одному и тому же адресу содержимое ячеек записываются в многоразрядное выходное слово. Это, так называемые, параллельные вычисления в пространстве.

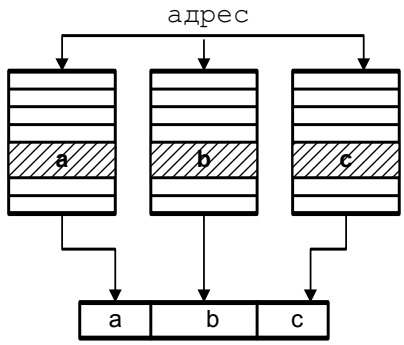


Рис. 3. 21. Пакетный доступ к памяти

4. **Конвейерный доступ.** Используется и конвейеризация и распараллеливание. Имеется 3 банка памяти. Имеется очередь запросов, где записываются адреса. Эта очередь запросов поступает на все 3 банка памяти последовательно. Имеется выходная очередь, в которую передаются данные с выходов банков памяти. Первые данные получаются через время Δt, равное времени доступа к банку памяти. Каждый последующий через время, равное времени извлечения данных из входной очереди, но только в том случае, если запрашиваемые данные располагаются в разных банках.

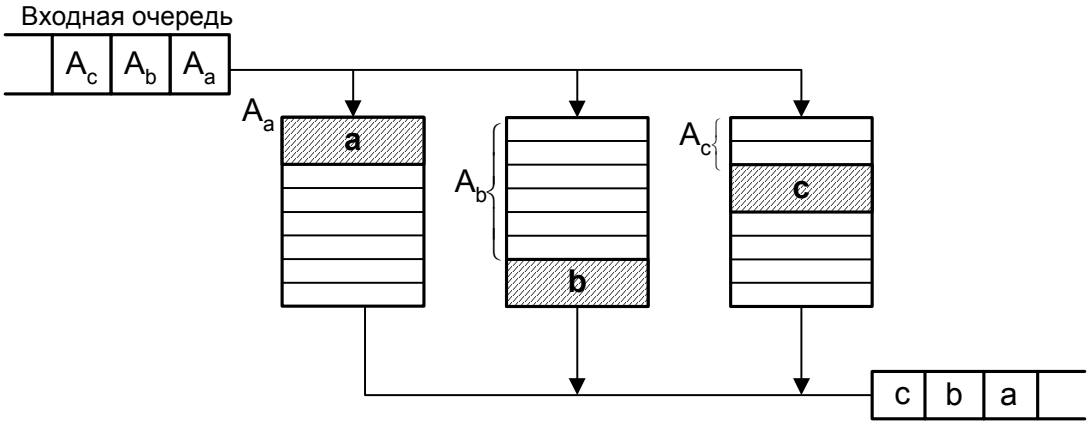


Рис. 3. 22. Конвейерный доступ к памяти

Принципы построения запоминающих устройств произвольной выборки (ЗУПВ)

ЗУПВ – это устройства, предназначенные для хранения, чтения и записи данных.

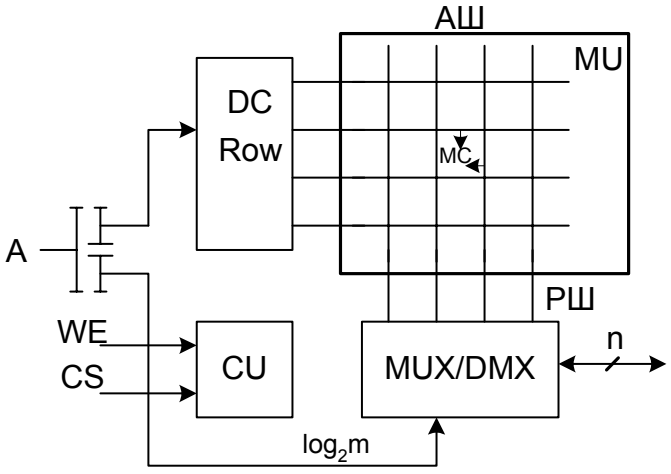


Рис. 3. 23. Функциональная схема ЗУПВ

АШ – адресные шины.

РШ – разрядные шины.

МС – Memory Cell (ячейка памяти).

MU – Memory Unit (память).

Доступ к ячейке памяти осуществляется за счет дешифрации адреса строки и адреса столбца.

Структурная схема ЗУПВ

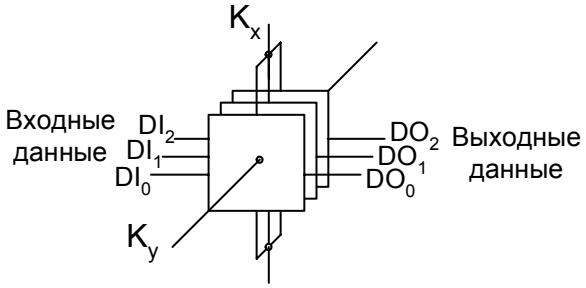


Рис. 3. 24. Структурная схема ЗУПВ

Статическая память SRAM

Статическая память SRAM (Static Random Access Memory) – память, способная хранить информацию в статическом режиме, т.е. сколь угодно долго при отсутствии обращений (при наличии питающего напряжения).

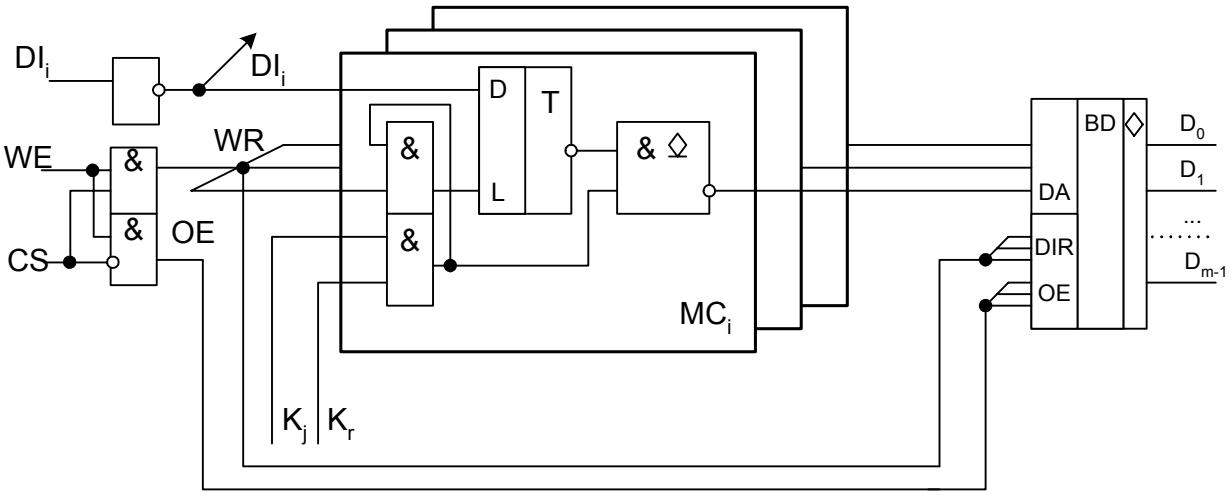


Рис. 3. 25. Функциональная схема статической памяти SRAM

Ячейки статической памяти реализуются на триггерах. Быстродействие и энергопотребление статической памяти определяется технологией изготовления и схемотехникой запоминающих ячеек. Самая экономичная КМОП-память (CMOS Memory) имеет время доступа более 100 нс, но зато пригодна для длительного хранения информации при питании от маломощной батареи, что и применяется в памяти конфигурации PC. Самая быстродействующая статическая память имеет время доступа в несколько наносекунд, что позволяет ей работать на системной шине процессора, не требуя от него тактов ожидания.

Динамическая память DRAM.

Динамическая память DRAM (Dinamic Random Access Memory) получила свое название от принципа действия ее запоминающих ячеек, которые выполнены в виде

конденсаторов, образованных элементами полупроводниковых микросхем. С некоторым упрощением описания физических процессов можно сказать, что при записи логической единицы конденсатор заряжается, а при записи нуля – разряжается. Схема считывания разряжает через себя этот конденсатор, и, если заряд был ненулевым, выставляет на своем выходе единичное значение, и подзаряжает конденсатор до прежнего значения. При отсутствии обращения к ячейке со временем за счет токов утечки конденсатор разряжается и информация теряется, поэтому такая память требует постоянного периодического подзаряда конденсаторов (обращения к каждой ячейке) – память может работать только в динамическом режиме. Этим она принципиально отличается от статической памяти, хранящей информацию сколь угодно долго (при включенном питании). Благодаря относительной простоте ячейки динамической памяти на одном кристалле удается размещать миллионы ячеек и получать самую дешевую полупроводниковую память достаточно высокого быстродействия с умеренным энергопотреблением, используемую в качестве ОП компьютера.

Запоминающие ячейки микросхем **DRAM** организованы в виде двумерной матрицы.

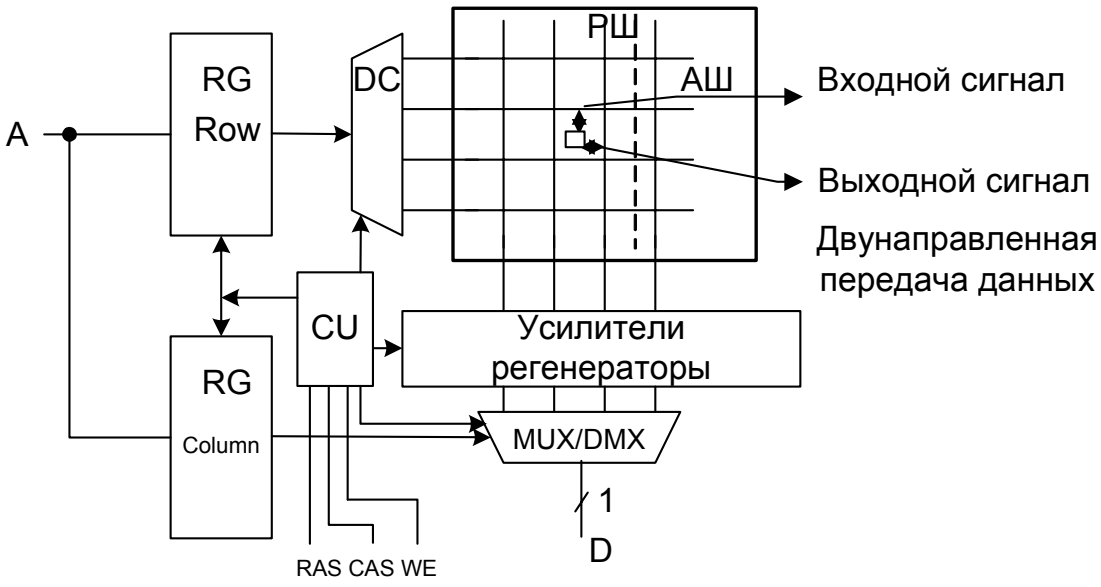


Рис. 3. 26. Функциональная схема динамической памяти DRAM

RAS# (Row Address Select) - выбор адреса строки.
CAS# (Column Address Select) - выбор адреса столбца.
WE – Write Enable.
OE# - Output Enable.
RG Row – регистр строк.
RG Column – регистр столбцов.
MUX/DMX – мультиплексор-демультиплексор.
D – двунаправленный одноразрядный выход.
DC – дешифратор. Формирует адресные линии.

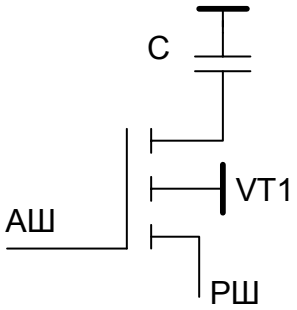


Рис. 3. 27. Схема запоминающей ячейки памяти

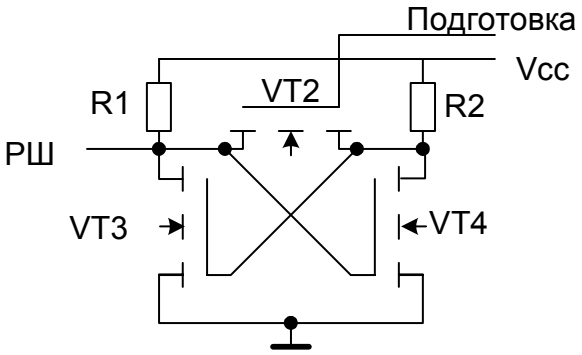


Рис. 3. 28. Схема усилителя-регенератора

Ячейка памяти выполняет только роль ключа в выборке из конденсатора С. Сохранность данных обеспечивается усилителем-регенератором. При отсутствии адресного сигнала АШ ключ VT1 разомкнут и конденсатор С хранит накопленный ранее заряд. В этом случае сигнал “подготовка” активен и напряжения на стоках транзисторов VT3 и VT4 равно Vcc/2, т.к. транзистор VT2 открыт.

При записи данных выходной сигнал низкого или высокого уровня подается на сток VT4, когда снимается сигнал “подготовка”. Это обеспечивает появление несимметричности в схеме. В момент открытия транзистора VT1 сигналом АШ происходит заряд или разряд конденсатора С в зависимости от состояния соответствующей разрядной шины РШ. После снятия сигнала АШ схема возвращается в исходное состояние, а конденсатор С накапливает или же не накапливает заряд.

При чтении данных снимается сигнал “подготовка” и подается сигнал АШ одновременно. Если напряжение на конденсаторе больше Vcc/2, сигнал на стоке VT3 высокий, иначе – низкий. Тем самым данные, записанные на конденсаторе, переписываются в триггер-регенератор с одновременным восстановлением заряда на конденсаторе.

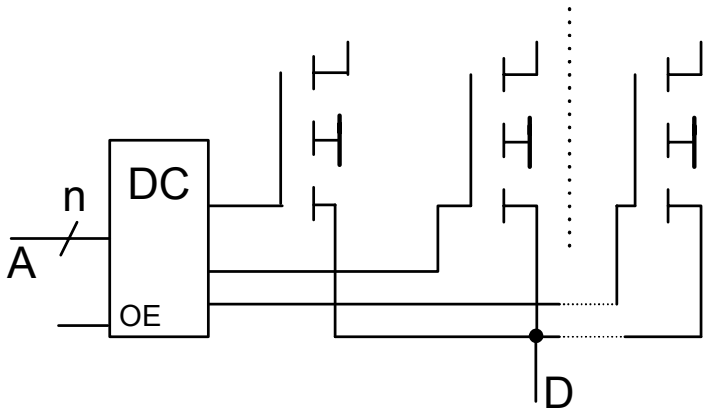


Рис. 3. 29. Схема мультиплексора-демультиплексора

Сначала на микросхему подается адрес строки, потом (через некоторое время, равное считыванию и регенерации всех запоминающих ячеек соответствующей АШ) подается номер столбца, который коммутирует считанный записанный разряд на выход D микросхемы.

Временная диаграмма ИМС КМ48С512

ИМС КМ48С512 – 512 Кслов, разрядность 8 бит, технология изготовления – КМОП. ИМС – типа EDO (extended data output).

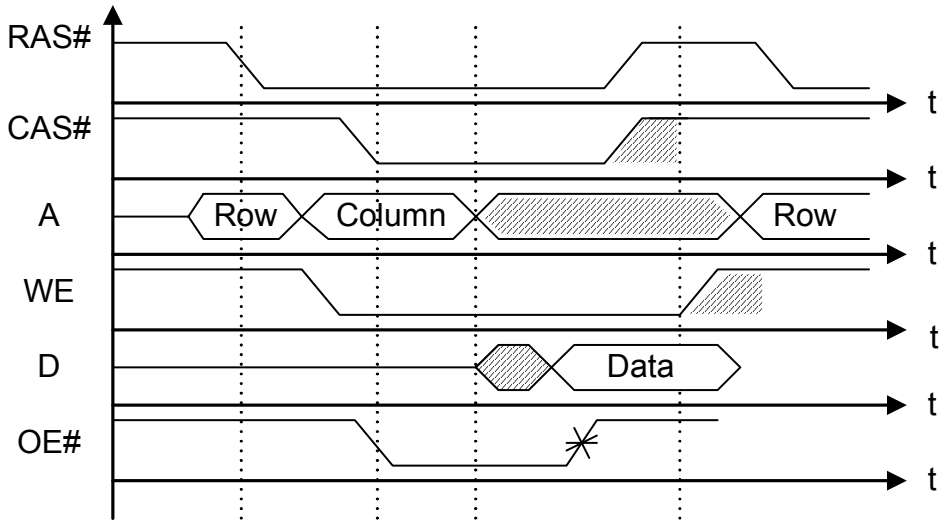


Рис. 3. 30. Временная диаграмма чтения ИМС КМ48С512

Заштрихованные области – безразличное состояние.

***** - Данные находятся на выходе ИМС.

Сигнал OE# символизирует о готовности данных при чтении переходом от низкого уровня к высокому во время активности сигнала CAS#.

Различные моменты появления данных на выходе определяется тем, что, если номер столбца текущего обращения совпал с номером столбца предыдущего обращения, то требуемая строка данных уже находится в усилителях-регенераторах и остается только скоммутировать требуемый триггер на выход, не извлекая данных из матрицы.

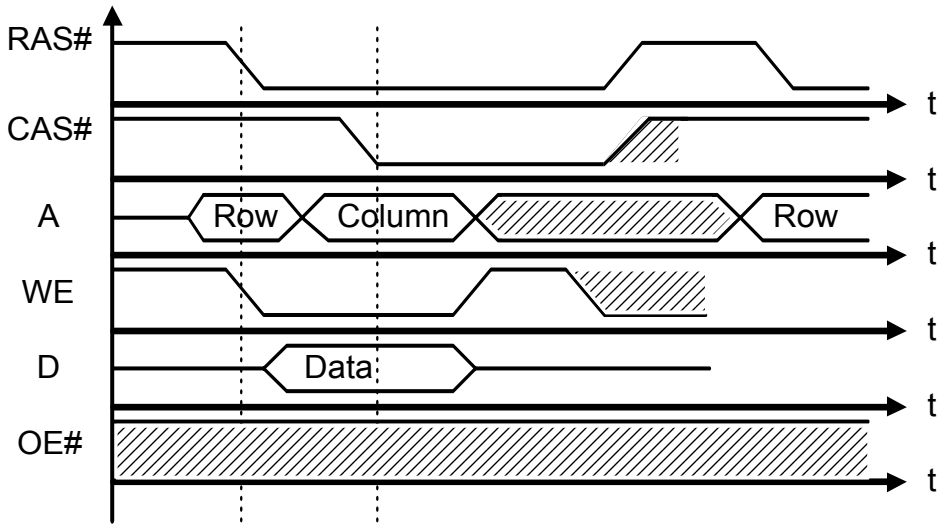


Рис. 3. 31. Временная диаграмма записи ИМС КМ48С512

Динамическая память поддерживает следующие режимы работы:

1. Чтение.
2. Запись.
3. Чтение – модификация – запись. (Сначала данные считываются, а затем на их место записываются новые значения). Эта операция позволяет реализовать семафорный механизм взаимного исключения.
4. Быстрый станичный режим (FPM).

Чтение в режиме FPM (Fast Page Mode).

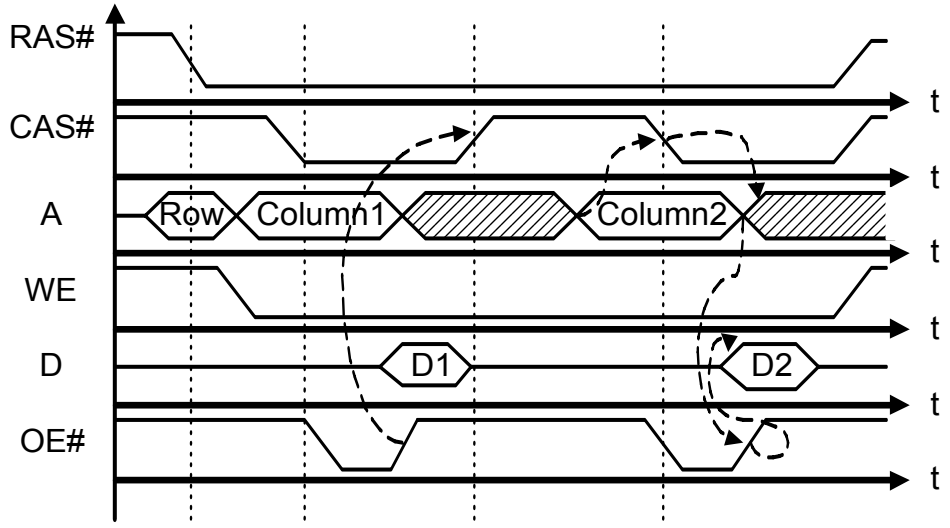


Рис. 3. 32. Временная диаграмма чтения в режиме FPM ИМС КМ48С512

Групповая операция заключается в последующем чтении четырех порций данных из микросхемы динамической памяти. Время групповой операции задается четырьмя числами (целыми): 5-7 –3-3-3 (Для ИМС типа FPM) или 4-3-3-3. Для ИМС типа EDO при 66МГц: К-2-2-2, где К зависит от производителя (например, К=3).

5. Режим регенерации данных. Время регенерации t от 2 до 4 мс (если мы не произвели чтение-запись в течение этого времени, данные потеряются).

Цикл регенерации памяти происходит при любом доступе к строке за время, не превышающее время регенерации. Имеется специальный цикл регенерации. Когда внешнее устройство регенерации через время, не превосходящее время регенерации, выполняет обращение ко всем строкам.

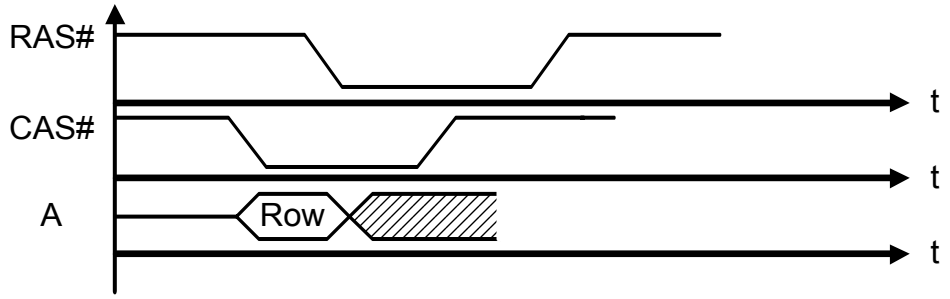


Рис. 3. 33. Временная диаграмма цикла регенерации ИМС КМ48С512

Признаком этого обращения является сигнал CAS# раньше сигнала RAS#. В этом случае простое чтение данных из запоминающей матрицы в триггеры регенерации.

Тип памяти BEDO – расширенный пакетный режим по выходу. Очень похож на страничный режим доступа к памяти со следующими отличиями: каждый сигнал CAS# передает на выход данные из следующего триггера.

Синхронная динамическая память SDRAM (Synchronous DRAM)

Таблица истинности команд

Тип	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	DQM	$A11$	$A10$	$A9-A$
MRSet (установка режима)	L	L	L	L	X	OP Code		
Саморегенерация	L	L	L	H	X	X	X	X
Подзарядка	L	L	H	L	X	BS	L/H	X
Активация банка	L	L	H	H	X	BS	ROW	
Чтение	L	H	L	H	X	BS	L/H	COL
Запись	L	H	L	L	X	BS	L/H	COL
NOP (нет операции)	L	H	H	H	X	X	X	X
Suspend (приостанов)	CLE = L							

Для пакетного обмена данными для 66МГц памяти время групповой операции: 3-1-1-1.

***Структурная схема** SDRAM*

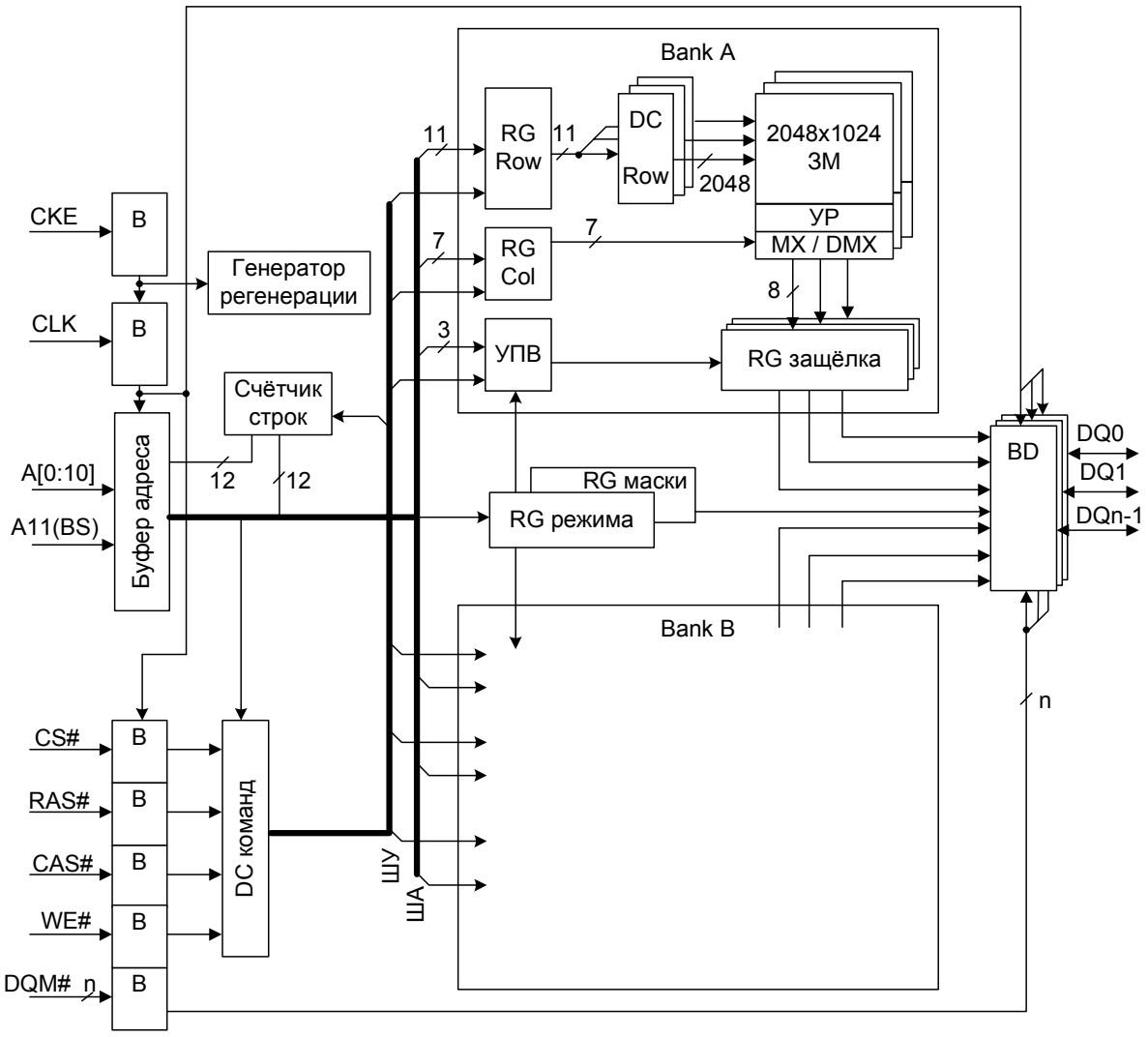


Рис. 3. 34. Структурная схема SDRAM

УР – усилитель регенератор.

УПВ – управление последовательностью вывода разрядов.

Схема регенерации включается, когда нет синхронизации.

CLC – входной импульсный сигнал. Все сигналы микросхемы воспринимаются в центре переднего фронта этого сигнала.

CKE (Clock Enable) – входной, потенциальный. Активизирует восприятие сигнала CLK, когда высокий уровень. При низком уровне переводит микросхему в режим отключения питания, спящий режим или режим саморегенерации.

CS# - входной, импульсный. Разрешает декодирование команд дешифратором команд, когда низкий уровень.

RAS#, CAS#, WE# - входные, импульсные. Определяют 3 разряда команды.

A[0:10] - входные, потенциальные. Во время цикла активации банка A[0-10] определяет адрес строки в момент активности сигнала CLK. Во время циклов чтения и записи A[0:9] определяет столбец, а разряд A10 используется для инициализации операции автоподзарядки в конце пакетного цикла чтения-записи. Если A10 активен, происходит подзарядка банка, определяемого разрядом A11. Во время цикла подзарядки A10 в конъюнкции с A11 определяют банк, подлежащий подзарядке. Если A10 активен, то подзаряжаются два банка, если не активен, подзаряжается банк A11.

DQ[0-15] – вход/выход, потенциальный. Определяют входные/выходные данные.

DQM# - входной, импульсный, активен низким уровнем. Сигнал маскирования входов/выходов, когда они переводятся в высокоимпедантное состояние. Определяет совместно с регистром маски, какие разряды переводятся в отключенное состояние.

Регистр режима

11	10	9	8	7	6	5	4	3	2	1	0
Mode					CAS# задержка			Пакетный режим	Длина максимального пакета		

Mode - режим. 00000 – нормальный, xx100 – пакетный.

CAS# задержка – число тактов по прошествии которых после начала обмена появляются первые данные.

Пакетный режим – Если 0, то последовательный пакетный режим, а если 1, то интерлив.

Длина пакета – 1, 2, 4, 8, full.

Временная диаграмма цикла активации данных

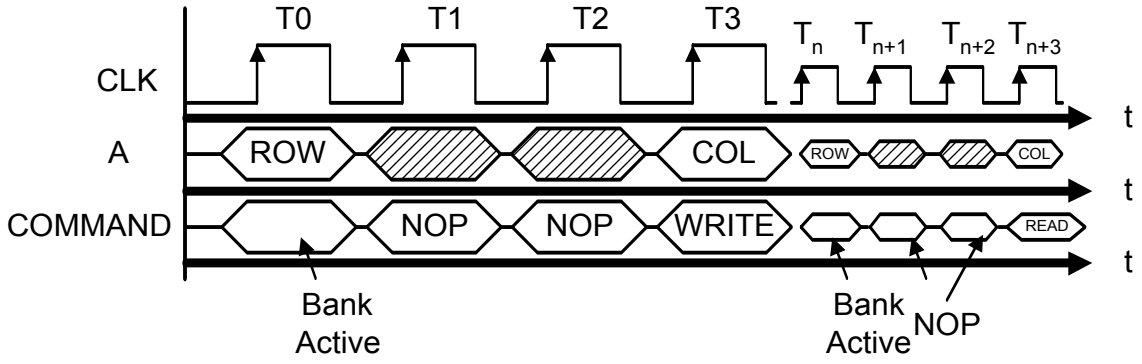


Рис. 3. 35. Временная диаграмма цикла активации данных

Временная диаграмма пакетного чтения данных

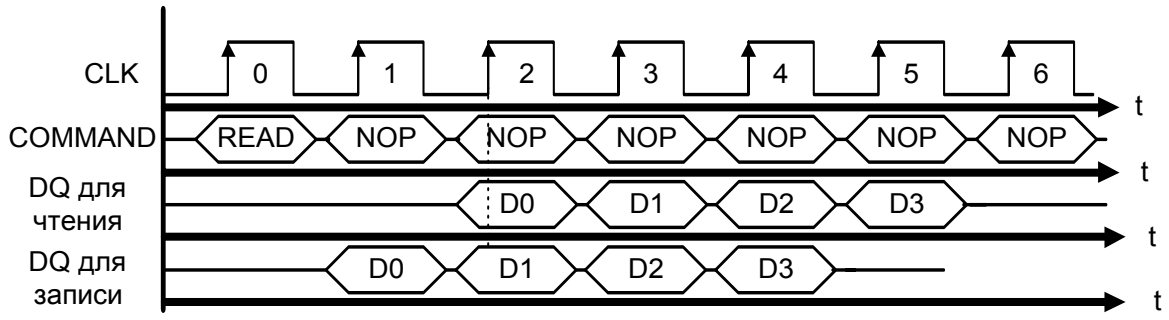


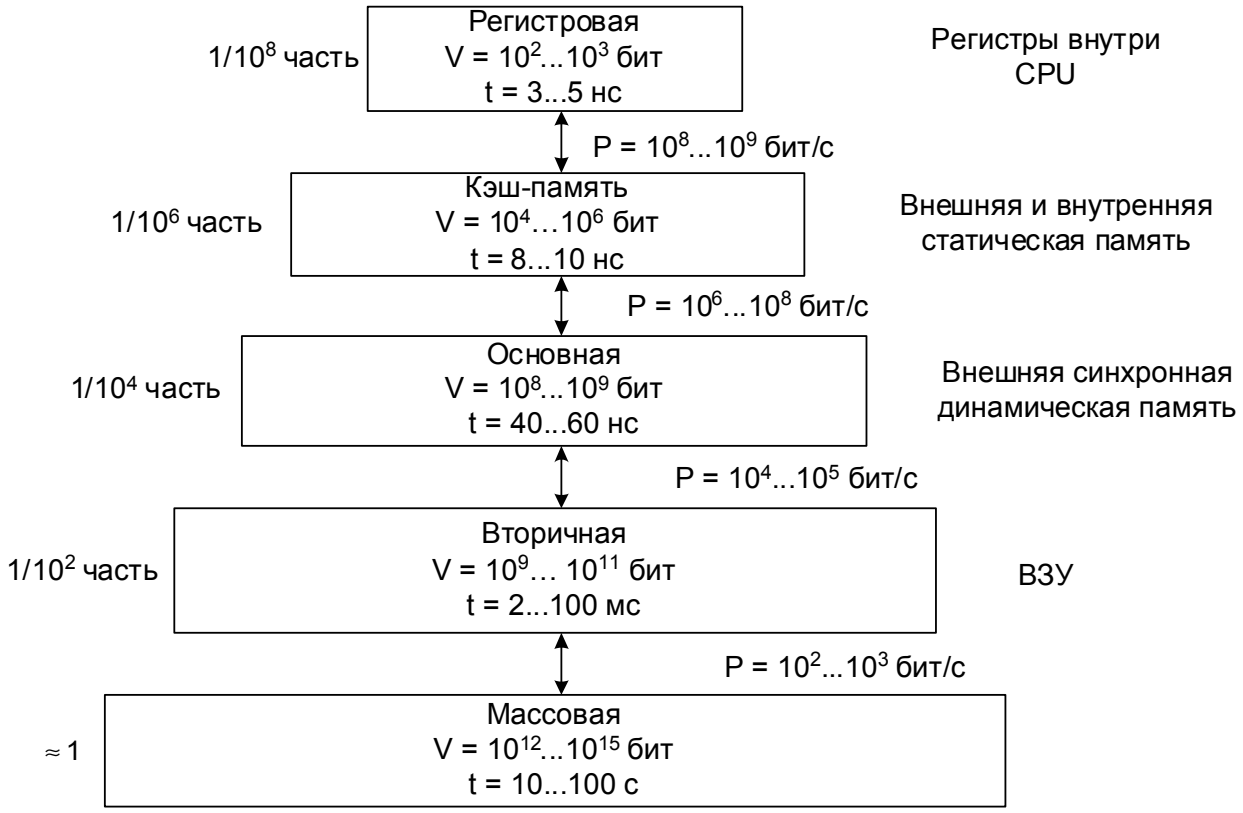
Рис. 3. 36. Временная диаграмма пакетного чтения данных

Автоподзарядка

Если при передаче номера столбца A10 равен 1, то команды чтения и записи выполняются с автоподзарядкой (регенерация данных). Автоподзарядка заключается в следующем. После переписи данных в регистр защёлку происходит регенерация строки запоминающей матрицы. При этом используется адрес строки, переданный при активации банка. В отличие от динамической памяти, где автоподзарядка производится в процессе чтения, здесь гарантируется, что в течение 3-х тактов не изменён банк.

Тема 3.4 Кэш-память

Иерархическая организация памяти



V- объём, t - быстродействие, P - пропускная способность

Рис. 3. 37. Иерархическая организация памяти

Иерархическая организация памяти предусматривает согласование времени доступа, объёма и пропускной способности памяти на различных уровнях. Основная задача, стоящая перед подсистемой памяти – это её виртуализации, т.е. метод автоматического управления иерархической памятью, т.о. что реализуется единая быстродействующая память большого объёма. Учёт локальности потока команд и данных в пространстве и времени и частоты использования данных позволяет построить виртуальную память, в которой :

- используется блочная пересылка между уровнями;
- наиболее часто используемые данные располагаются в более быстродействующей памяти.

Кэш-память - это быстродействующая память, расположенная между процессором и основной памятью, между дисковой и основной. Процессор осуществляет обмен только с кэш-памятью, а кэш-контроллер выполняет обмен между кэш-памятью и ОП.

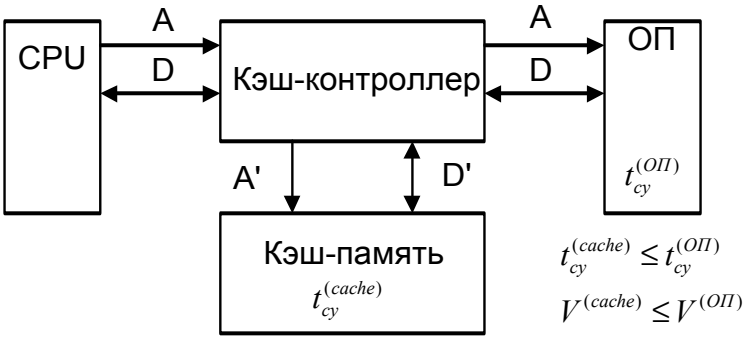


Рис. 3. 38. Организации кэш-памяти

Процессор, когда обменивается по системному интерфейсу, выставляет адрес (A) и записывает (читает) данные (D) по этому адресу. Кэш-контроллер обращается к кэш-памяти и поставляет эти данные в процессор, если в кэш памяти эти данные есть, если их нет, то кэш-контроллер обращается в ОП и записывает эти данные в кэш-память, а потом дает разрешение процессору на получение этих данных. В результате такой схемы в кэш-памяти накапливаются наиболее используемые данные, потому что объем кэш-памяти меньше, чем объем ОП и она состоит из кусочков, которые имеют следующий вид:

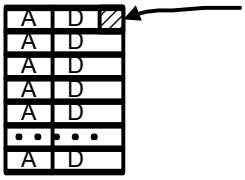


Рис. 3. 39. Кэш-память

A – адрес блока;

D – смещение внутри блока.

Если порция данных по указанному адресу находится в кэш-памяти, то из нее извлекаются данные и передаются в процессор, если этих данных нет, то кэш-контроллер должен выбрать какую-то порцию или место, которое он должен освободить и записать требуемые данные в кэш-память, причем запись требуемых данных осуществляется как при чтении данных процессором, так и при записи, потому что на самом деле процессор записывает только часть этого блока (на рис. 3.39. заштриховано), а не весь. А значит, чтобы записать эту часть, надо прочитать всю порцию данных, которая находится по некоторому адресу и лишь только потом осуществить запись в этот маленький кусочек.

Ёмкость кэш-памяти составляет 1/500…1/100 ёмкости кэшируемой памяти, а её быстродействие 5…10 раз выше нижележащей памяти.

Кэш-память основана на предвосхищении наиболее вероятного использования процессором данных из основной памяти, путем копирования их в быстродействующую память, а также сохранение наиболее часто используемых данных в быстродействующей памяти. Основой для построения кэш-памяти служит использование частотного принципа и принципа локальности.

Локальность потока команд и данных, заключающийся в том, что взаимосвязанные данные в потоке команд и взаимосвязанные команды в потоке данных находятся на небольшом расстоянии друг от друга, как во времени, так и в пространстве.

Организация обмена между основной памятью и кэш-памятью основана на блочной пересылке, когда основная память разбивается на блоки равной длины (4, 8, 16 и т.д. байт) и обмен между уровнями иерархий происходит этими блоками. Т.е. принцип локальности

позволяет, используя блочную пересылку предвосхитить следующее обращения процессора к основной памяти.

Частотный принцип реализуется путём накопления в кэш-памяти наиболее часто используемых блоков. Если блок долго не используется, при очередной блочной пересылке его место занимают новые данные. Блок заведомо в несколько раз больше разрядности нижележащей памяти.

Мы будем использовать следующую модель адресного пространства кэш-памяти и ОП:

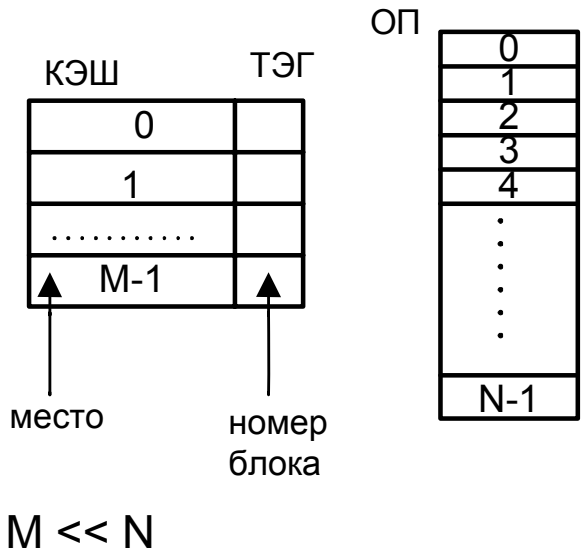


Рис. 3. 40. Адресное пространство кэш-памяти

Имеется адресное пространство кэш-памяти, которое разбито на М-мест. Имеется ОП, которая разбивается на блоки (блоков больше чем мест). Причем, в одном месте в кэш-памяти помещается один блок. В кэш-памяти каждый блок снабжается тэгом, в котором записывается, хранящийся в ОП, номер блока. Это означает, что в 0-м месте может находиться 3-й блок, на 1-м месте может находиться N-1 блок в месте М-1 может ничего не находится и т.д. Возникает проблема: если вся кэш-память занята и процессор затребовал данные, которые в данный момент не находятся в кэш-памяти. Необходимо освободить место, которое занято и в него переписать, используя блочный обмен, блок который содержит кусочек требуемых данных, а потом разрешить процессору работать с этими данными в кэш-памяти. Но как выбрать это место в кэш-памяти, ведь можно попасть в то место, которое потребуется в следующий момент времени, поэтому при рассмотрении кэш-памяти говорят о стратегии замещения места. Стратегия обновления основной памяти определяет метод замещения старых данных в основной памяти новыми. Допустим, процессор записал в кэш-память какие-то данные. Освободить это место, не сохранив измененные данные в основной памяти нельзя, то есть должен быть флаг, указывающий на то, что эти данные изменены. Если необходимо освободить место, надо до этого освобождения сохранить эти данные в основной памяти, чтобы не нарушилась целостность данных, иначе процессор, программист или программа будут считать, что они эту ячейку изменили, а окажется, что они изменили кэш-память, а до основной памяти это изменение не дошло. Для решения такой проблемы необходимо выполнить стратегию обновления.

Стратегии:

1. **Стратегия обновления.** Определяет метод замещения старых данных в основной памяти при их модификации в кэш-памяти: сквозная и обратная запись.
2. **Стратегия выборки.** Определяет метод выбора места для размещения считанного из основной памяти блока. В разных типах памяти стратегия разная.
3. **Стратегия замещения.** Определяет метод определения места, куда необходимо разместить новый блок данных, с учётом того, что все доступные места заняты (какой блок выгрузить из кэш-памяти).

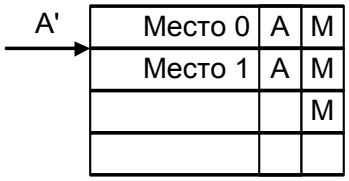


Рис. 3. 41. Кэш память

Бит А периодически сбрасывается в “0”. При обращении к блоку бит А устанавливается в “1”. Если флаг сброшен, то это место – кандидат в замещение. Если флаг установлен, то это место – кандидат в замещение в последнюю очередь.

Разумно использовать следующую стратегию выборки и замещения: если имеется свободное место, то оно выбирается для записи нового блока из ОП. Если свободных мест нет, то замещению подлежит место со сброшенным флагом доступа, например, имеющий наименьший номер. Если флаг М взведен, то этот блок подлежит сохранению (обновлению) в ОП. Самая простая стратегия - это стратегия псевдослучайного выбора места.

Далее стратегия выбора наиболее редко используемого места.

Стратегия выбора немодифицируемого места, то есть то место, где блок не изменился. Из всех редко используемых блоков выбирается любой немодифицированный блок (определяется по флагу). Объем тэговой памяти занимает 10-20% от объема кэш-памяти.

Эффективность кэш-памяти определяется отношением числа успешных обращений (кэш-попаданий) к общему числу обращений к основной памяти процессором. Кэш-промах – это когда требуемая процессором порция данных не находится в кэш-памяти.

Направление просмотра определяет, какой следующий блок будет считан из ОП. Блочная выборка может доставить данные, расположенные как до, так и после затребованной порции процессором. Для учета этого вводится специальный тэг.

Если окажется, что процессор затребовал данные, которые ведут в сторону уменьшения адресов, то вполне вероятно, что контроллеру надо считать следующий блок. Поэтому кэш-контроллер должен фиксировать направление просмотра данных процессором для каждого блока.

Организация кэш-памяти

Рассмотрим три организации кэш-памяти:

- Полностью ассоциативная кэш-память.
- Кэш-память с прямым отображением.
- Ассоциативная по множеству кэш-память.

Они отличаются механизмом преобразования адресов и стратегиями замещения и обновления.

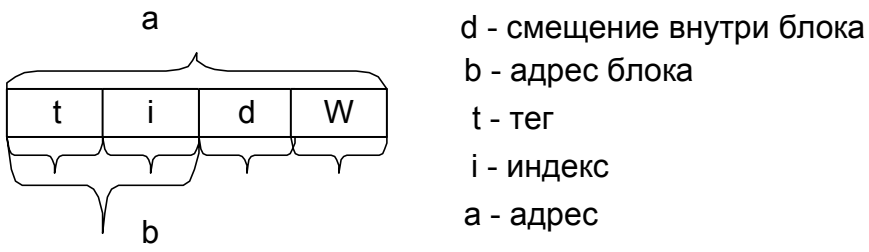


Рис. 3. 42. Структура системного адреса

Рассмотрим структуру системного адреса, который важен для кэш-контроллера при его обмене с процессором и ОП (рис. 3. 42.).

Индекс имеет диапазон изменения такой, что число индексов равно числу мест в кэш-памяти и соответствует номеру места, т.е. индекс однозначно определяет место блока в кэш-памяти.

Тэг – это отличительный признак блока, определяет, какой конкретно блок находится в этом месте.

Общий адрес процессора может быть разбит на 2 части: адрес блока в ОП и смещение требуемых данных внутри блока. W – определяет те разряды, которые не нужны для адресации слова и вместо них процессор выдает разряды BE#. Например, пусть есть 32 разрядный процессор по системному интерфейсу. Это означает, что процессор выставляет адрес 32-х разрядного слова, а не адрес байта, а вместо адреса байта он передает сигналы BE#, которые говорят о том, что в этом слове интересует только какая-то часть.

Полностью ассоциативная кэш-память

При полностью ассоциативном размещении блоков в кэш-памяти допускается размещение каждого блока b из ОП в любом месте m кэш-памяти.

Механизм преобразования адресов должен быстро дать ответ, существует ли копия блока b в кэш-памяти, и, если существует, то в каком месте m она находится.

Исходя из этого рассмотрим структуру полностью ассоциативной кэш-памяти.

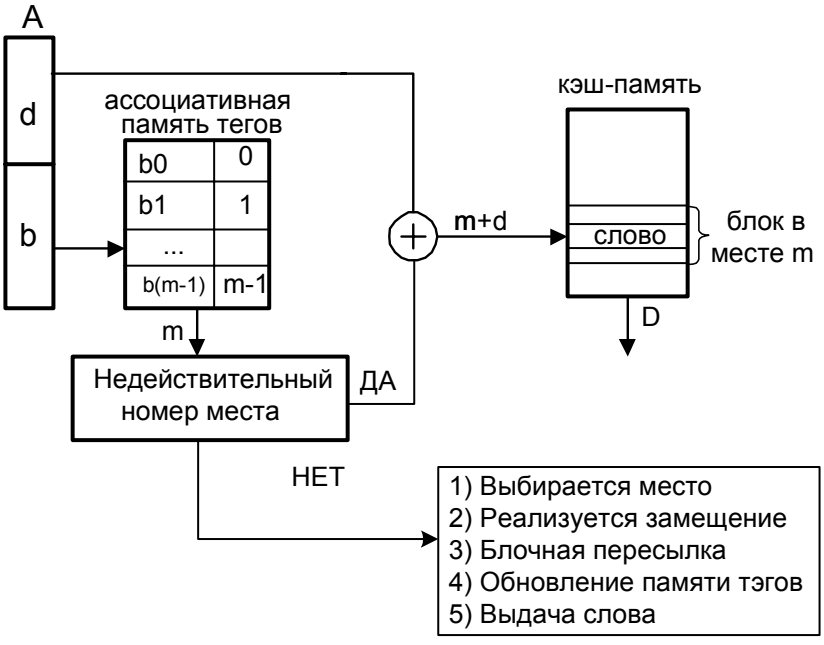


Рис. 3. 43. Структура полностью ассоциативной кэш-памяти

А – это адрес, состоящий из двух частей: номера блока и смещения, требуемых данных внутри этого блока. Часть адреса b передается в ассоциативную память тегов. Блок поступает и на выходе этой памяти имеется схема опознавания действительного или недействительного номера места. Число ячеек в памяти тегов равно числу мест в кэш-памяти с линейной организации, но эта кэш-память возвращает номер места, где хранится нужный блок.

Недостаток: необходимо использовать ассоциативную память, реализующую большое число сравнений для определения кэш-попаданий и место блока в кэш-памяти.

В ассоциативной памяти тегов фактически существует две части: первая часть, которая подвергается ассоциативному сравнению, то есть предъявили входные данные b**,** а она отвечает путем одновременного сравнивания всех тегов ячеек, есть ли ячейка, которая ассоциирована с этим номером блоком или нет. Для последовательного сравнивания нет времени (10нс). Получается, сколько ячеек, столько компараторов адреса. Вся ассоциативная память строится по принципу распараллеливания в пространстве.

Если у нас размер кэш-памяти 512Кб, а размер слова 8 Байт, то число мест равно М=512к/8=64к. Впервые полностью ассоциативная память была применена в Intel4086 и объем ее был 4 места.

Кэш-памяти с прямым отображением

Для размещения в кэш-памяти данных блока b в место m используется часть разрядов адреса блока i. Т.к. мы не можем осуществлять полностью ассоциативное сравнение, то часть разрядов адреса блока разбивают на 2 части, где первая часть будет ячейка ассоциативной памяти, где хранится место требуемого блока (i – индекс) и вторая часть t – тег, определяющий отличие одного блока от другого при его хранении в одном и том же месте. d – смещение внутри блока. Преобразование адреса блока в место М сводится к выборке разрядов i из адреса.

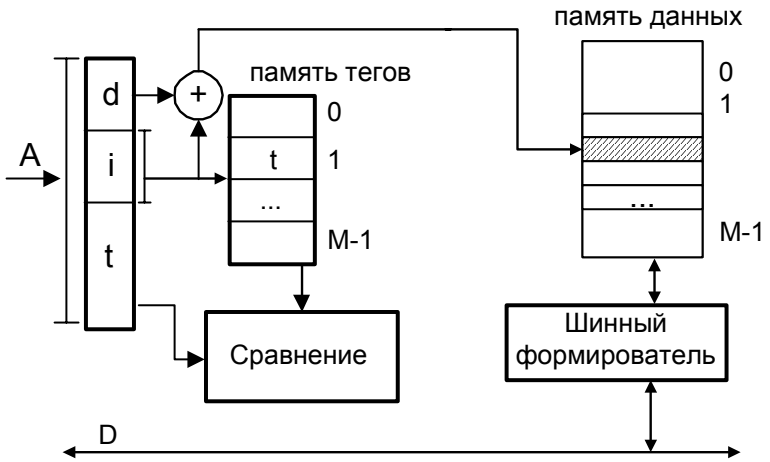


Рис. 3. 44. Структура кэш-памяти с прямым отображением

Индекс и смещение внутри блока суммируются и являются адресом блока в памяти данных. Разряды адреса фактически являются номером ячейки, где хранится тег того блока, который находится в кэш-памяти. Из памяти тегов, которая имеет обычную линейную организацию, выбирается значение, называемое тегом, хранящимся в кэш-памяти. Этот тег поступает в устройство сравнения, которое проверяет, равен ли этот тег тегу, хранящемуся в этом блоке. Если они совпадают, то кэш-попадание, иначе кэш-промах.

Пусть у нас есть 6 блоков в ОП, которые присутствуют в кэш-памяти. Память тегов по числу ячеек совпадает с числом мест в кэш-памяти (Рис. 3. 45).

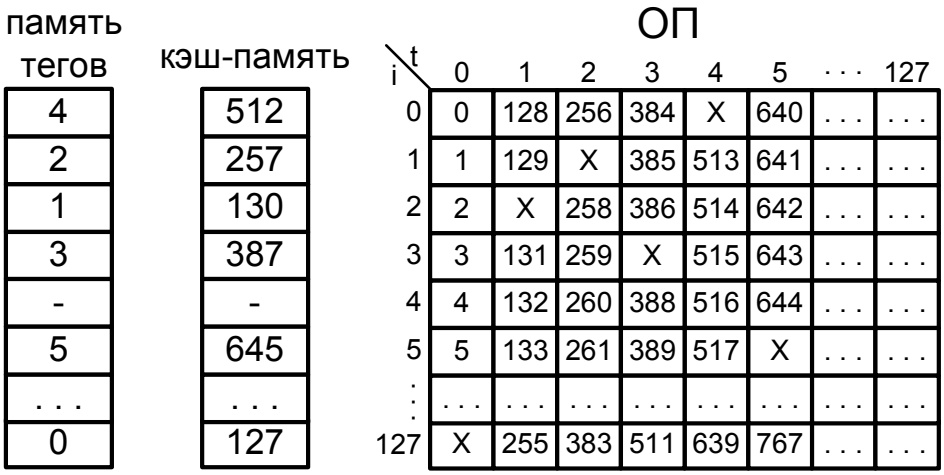


Рис. 3. 45. Структура кэш-памяти

Недостаток: При частом обращении в разные области основной памяти, отображающиеся в одном и том же месте, эффективность кэш-памяти (Рис. 3. 46).

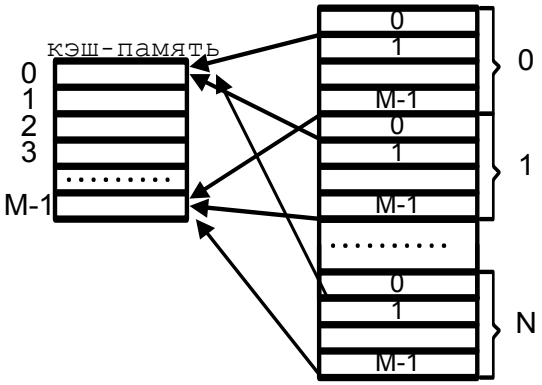


Рис. 3.46. Расслоение памяти

Разбиваем ОП на области так, чтобы в область помещалась кэш-память. Получается, что всегда в кэш-памяти может находиться непрерывный участок основной памяти размером с кэш-память.

Ассоциативная по множеству кэш-память

Реализован компромисс между полностью ассоциативной кэш-памятью и кэш-памятью с прямым отображением: ОП разбивается на блоки, объединенные в области (как при прямом отображении), КП разбивается на группы мест по числу блоков в области, т.е. для каждого индекса имеется несколько мест, но в разных группах (Рис. 3. 47). Тэги в этом случае для определения кэш-попадания подвергаются ассоциативному сравнению (как при полностью ассоциативной памяти).

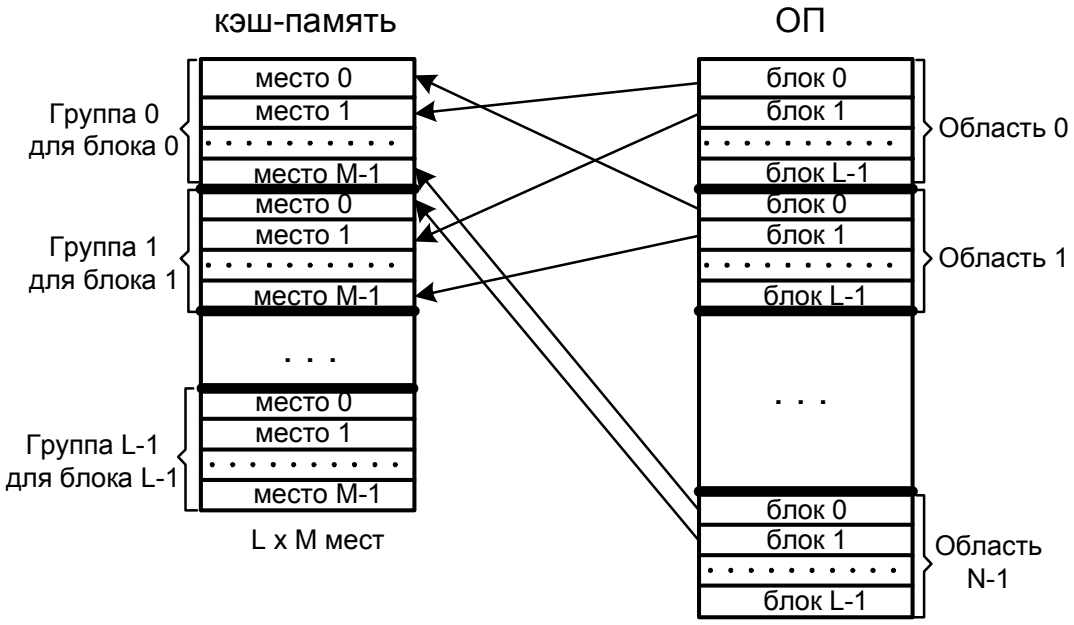


Рис. 3.47. Расслоение ОП по КП

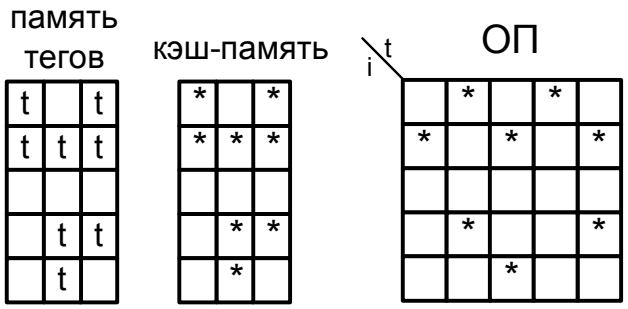


Рис. 3. 48. Структура кэш-памяти

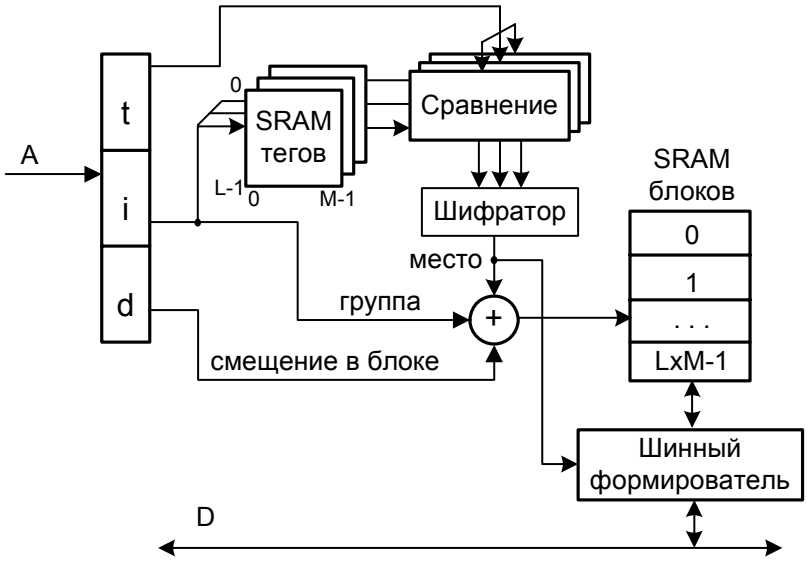


Рис. 3. 49. Структура ассоциативной по множеству кэш-памяти

Как и в памяти с прямым отображением имеется ЗУ с линейной организацией памяти, которое называется статическое ЗУ тегов. Индекс определяет группу тегов, то есть произошло как бы расслоение кэш-памяти. Эти теги подвергаются ассоциативному сравнению с тегом находящимся в адресе. Теги поступают на цифровые компараторы сравнения тегов. После сравнения теги поступают в шифратор, который говорит о том, в

каком месте группы находится требуемый блок или что этого блока нет. Если блок имеется, выдается номер места в этой группе. С шифратора снимаются 2 сигнала, т.е. решаются две задачи:

5. Выбор свободного места в кэш-памяти, если оно есть.
6. Если все места заняты, выбирается замещаемое место путем анализа тегов, содержащее бит обращения (модификации) и формирует номер места, которое можно использовать для размещения нового блока, предварительно сохранив хранящийся, если установлен бит модификации в теге.

Допустим контроллер обнаружил кэш-промах, т.е. сработала эта схема и обнаружила, что нет требуемого блока в кэш-памяти, но вся кэш-память свободна или занята, и имеются некие свободные места в той группе, которая предназначена для хранения этого блока. Для выбора свободных мест шифратор должен выдавать не только сигнал о том, что блока нет, но и сигнал, определяющий свободно место или занято. Если свободных мест нет, то в этом случае можно использовать в памяти тегов дополнительные теги, т.е. хранить там не только теги мест, но еще и специальные флаги: W – в это место произведена запись, F – частота обращения к блоку (контроллер должен с некой периодичностью сбрасывать этот флаг в 0), B.

Шифратор дополнительно решает выбор свободного места в кэш-памяти. Для этого может использоваться специальный бит в теговой памяти B – место занято. Если все места заняты и установлен бит записи W, шифратор реализует стратегию замещения. Для этого используются такие биты, как W - место модифицировано и F – к месту было обращение процессора. Биты F контроллером периодически сбрасываются в 0.

Недостатки:

7. Эта организация самая сложная, но она реально используется.
8. Требуется большой объем памяти тегов.
9. Усложнена реализация стратегий обновления, замещения и выбора, т.е. необходимы специальные схемы для определения свободных мест в кэш-памяти, для выбора места подлежащего замещению (обновлению), а также схем, сохраняющих изменённый блок кэш-памяти в основной памяти.

Стратегии обновления кэш-памяти

Реально у любой системы с кэш-памятью для одних и тех же данных имеется две копии: 1-я копия в кэш-памяти, а вторая в ОП и эти копии могут различаться (когда взведен флаг W**,** то копия в кэш-памяти отличаются от копии в ОП). Поэтому возникает проблема обеспечения целостности данных. Кэш-контроллер должен реализовывать некую стратегию обновления ОП для предотвращения использования устаревших данных в качестве достоверных. На практике используется сквозная (TWR) и обратная (BWR) записи.

- **Сквозная запись –** одновременно с обновлением кэш-памяти происходит запись в ОП.
-

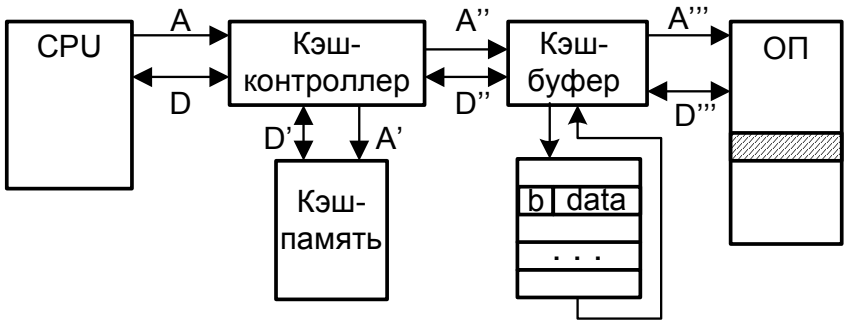


Рис. 3. 50. Структура буферизированной сквозной записи

- **Буферизированная сквозная запись -** обновляемый блок передаётся в кэш-буфер и транзакция к памяти завершается, т.е. процессор продолжает работу получив требуемые данные. Блок, записанный в кэш-буфере независимо от работы кэш-контроллера по мере освобождения интерфейса с ОП записывается в неё. Т.к. кэш-буфер в общем случае может содержать быстродействующую память, достаточную для хранения не одного, а нескольких блоков, организована эта память в виде очереди. Каждое место этой памяти состоит из 2-х частей: адреса блока b и данных блока data.
- Недостаток: Реализуется еще одно устройство кэширования в виде кэш-буфера.
- **Обратная запись** – для каждого места в кэш- памяти вводится в поле тегов специальный бит изменения данных блока, который устанавливается, если блок обновлён, т.е. является более поздним, чем его исходная копия в ОП. Перед освобождением места в кэш-памяти, занятого этим блоком контроллер проверяет этот бит. Если он установлен, то перед загрузкой в это место новых данных выполняется запись содержимого этого места в ОП.
- Недостаток: При использовании энергонезависимой памяти отключение питания не позволяет сохранить изменённые данные в этой памяти.
- **Буферизированная обратная запись -** изменяется порядок обращения к ОП. Замещаемый блок сохраняется в быстродействующей памяти – кэш-буфере вместе со своим адресом, далее читается замещающий блок из ОП в освобождённое место кэш-памяти, а затем по мере освобождения интерфейса с ОП замещённый блок передаётся в ОП. Очевидно, при этом не происходит задержка процессора при получении требуемых данных, такая, которая была бы при отсутствии буфера.

Организация кэш-памяти в многопроцессорных системах (МПС)

Проблема обеспечения целостности данных хранящихся в ОП для многопроцессорных систем, заключающаяся в том, что трудно определить, где находятся не актуализированные данные - в кэш-памяти процессоров или в ОП.

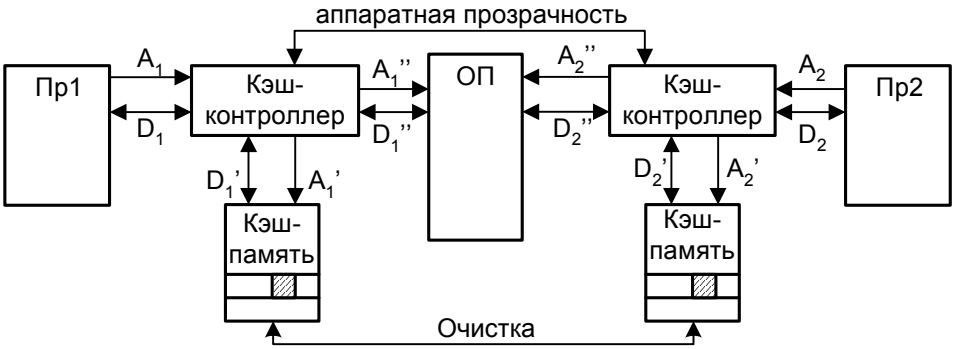


Рис. 3. 51. Организация кэш-памяти в многопроцессорных системах

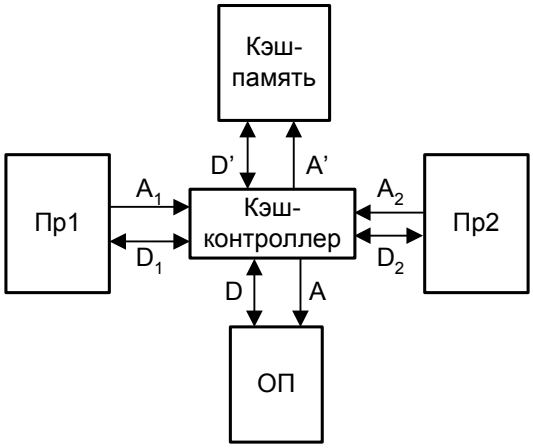
У нас имеется 2 процессора и имеется единая ОП, но у каждого процессора имеется своя кэш-память. Допустим у нас есть некая область в ОП. Процессор 1, обратившись к этой области ОП, получает в кэш-памяти копию этих данных. Если процессор изменил данные кэш-памяти, при использовании метода обратной записи, то возникает нарушение целостности данных, то есть данные кэш-памяти и ОП различны. Другой процессор, который использует туже самую область памяти «уверен», что данные в этой области целостны и получает эту копию в свою кэш-память. Это приводит к тому, что данные обрабатываются неправильно.

Существуют 3 метода обеспечения целостности данных в ОП:

10. **Очистка кэш-памяти.** Заключается в том, что после изменения данных в кэш-памяти место освобождается. Для обеспечения целостности данных необходимо, чтобы кэш-контроллеры были связаны и очистка происходила во всех кэш-памятях.

При этом данные очищаются не в том месте, где произошло изменение, а в тех местах, которые содержат этот блок - у других кэш-контроллеров.

11. **Аппаратная прозрачность.** Аппаратная прозрачность организуется двумя способами:
    - запись в кэш-память приводит к записи изменённых данных во все другие кэш-памяти. Идея заключается в том, чтобы обеспечить связь двух кэш-контроллеров. Каждый хранит адреса блоков под управлением другого кэш-контроллера. В случае доступа процессора, к одному и тому же блоку действия контроллеров согласуются;
    - Использование единой кэш-памяти для всех процессоров, которая работает с основной памятью с двумя портами (входами) по одному для каждого из процессоров. В этом случае кэш-память называется многовходовой.



    - Рис. 3. 52. Использование единой кэш-памяти

12. **Некэшируемая область в ОП.** Т.к. имеется канал прямого доступа к памяти, который является специализированным процессором по пересылке данных и вполне возможны различные конфликты, то общесистемные переменные хранятся в области, которая не подлежит кэшированию. Все запросы к некэшируемой области ОП вызывают кэш-промахи. Повышение быстродействия в некэшируемой области возможно путем копирования программным обеспечением этой области в кэшируемую память. В этом случае нужно использовать семафорный механизм взаимного исключения.

Тема 3.5. Подсистема прерываний

Прерывание - это временное прекращение вычислительного процесса, вызванное событиями, внешними по отношению к этому процессу.

Прерывание – это операция процессора, состоящая в регистрации предшествовавшего прерыванию состояния процессора и установлению нового состояния (ГОСТ).

- **Состояние процессора** – содержимое его внутренней памяти (регистры общего назначения, регистры флагов, регистр команд, счетчик команд, регистр микрокоманд, счетчик микрокоманд).

Состояние процессора определяется содержимым этой памяти. В момент прерывания эту память мы должны где-то, сохранить, чтобы в момент восстановления прерванного процесса мы могли восстановить эту память и продолжить выполнение с того места, где был процесс остановлен. Современный процессор имеет внутреннюю память порядка 100 бит и больше. Для того чтобы эту память уменьшить, надо запретить прерывание процессора не в любой момент, а только в некие стандартные моменты, когда состояние большинства памяти процессора является стандартным. Если разрешить прерывание процессора после окончания исполнения текущей команды, то некоторая

память процессора имеет стандартное значение (регистр микрокоманд либо счетчик микрокоманд) или значение которой в памяти потеряли свою актуальность (регистр команд). Т.е. мы фактически сократили объем сохраняемой памяти, обеспечив возникновение прерывания в момент смены команд процессора.

Регистры общего назначения не сохраняются в стеке в момент прерывания, для того чтобы упростить выполнение команды прерывания, поэтому, если мы в процедуре обработки прерывания вынуждены использовать регистры общего назначения, то мы должны выдать специальные команды, которые сохранят содержимое этих регистров в стеке, а перед возвратом из процедуры обработки прерывания эти регистры восстановить.

Для уменьшения объёма сохраняемых данных процессор прерывается не в любой момент, а в те моменты, у которых состояние части внутренней памяти предопределено. Чаще всего это происходит при окончании выполнения текущей команды, состояние управляющего блока можно исключить из текущего состояния.

Прерывание возможно только после полного завершения текущей команды.

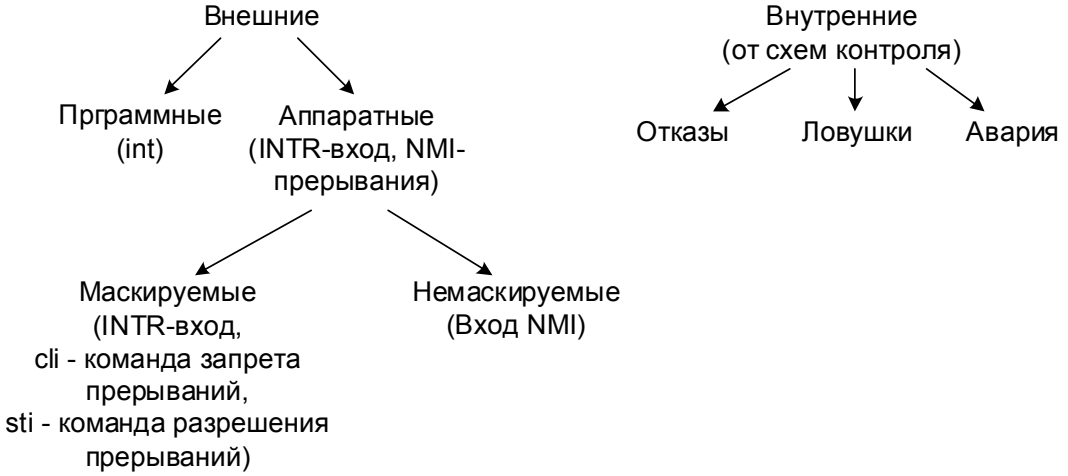


Рис. 3. 53. Источники прерываний

-
- **Маскируемые** прерывания можно запретить специальной командой, а **немаскируемые** - это те прерывания, которые нельзя запретить программными средствами и на вход немаскируемого прерывания подаются сигналы от схем контроля вычислительной системы. Например, если память ненадежно работает или возникла ошибка памяти, то понятно, что необходимо каким-то образом прервать текущий исполняемый процесс и обработать эту ситуацию.
- **Отказы** – возникают до или во время выполнения команды до её завершения. После возникновения отказа возможно восстановить прерванное состояние процессора и повторить.
- **Ловушки –** возникают после завершения выполнения команды и предназначены, например, для отладочных действий.
- **Авария** – неисправность или некорректность работы схем процессора, которые не входят в его ядро, например, 3-й отказ является аварией.
- Под программным прерыванием понимается команда, которая запускает процедуру обработки прерывания, но в момент выдачи этой команды эта процедура не исполняется, поэтому в чистом виде это прерыванием не называется, а прерывание - это внешняя команда процессора, причем асинхронная по отношению к основному циклу работы процессора. Программные прерывания представляют собой специфический способ вызова процедур – не по адресу, а по номеру в таблице.

Последовательность прерывания

Последовательность прерывания – это действие, выполняемое процессором для реализации прерывания текущего процесса:

1. Получает запрос на прерывание.
2. Ожидание завершения текущей команды.
3. Приём типа или номера прерывания.
4. Сохранение в стеке минимально необходимых данных состояния.
5. Запрет повторных прерываний.
6. Установление нового состояния процессора, задаваемого типом или номером прерывания.
7. Выполнение процедуры обработки прерывания.
8. Восстановление состояния процессора.

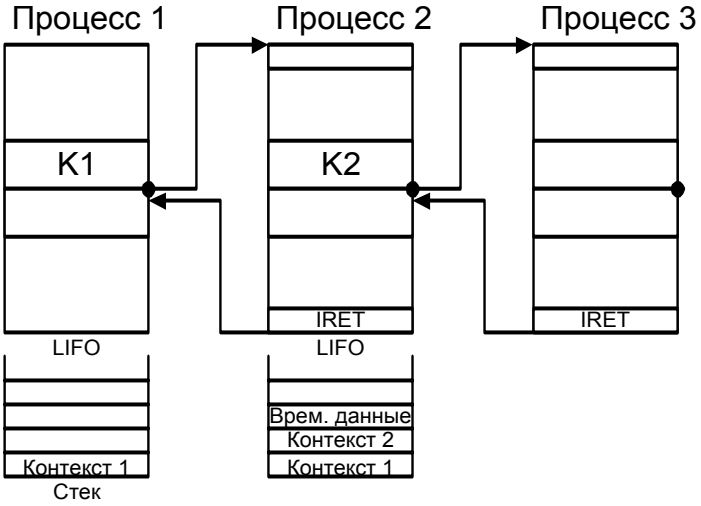


Рис. 3. 54. Последовательность выполнения прерываний

При обработке прерывания процессор сохраняет в стеке слово состояния, сбрасывает флаг разрешения прерываний IF и вызывает процедуру обслуживания, точка входа в которую описана в таблице прерываний, хранящейся в ОЗУ. При сохранении контекста осуществляется запрет прерываний от того же источника.

Процедура обработки завершается командой IRET, по которой из стека восстанавливаются автоматически сохраненные регистры (в регистре флагов прерывания разрешены) и процессор начинает выполнение команды, следующей за той, после которой исполнялось прерывание.

Конечно, программно во время обслуживания прерывания возможно умышленное или случайное изменение указателя или содержимого стека, и тогда команда IRET “отправит” процессор по другому адресу, в результате чего компьютер может зависнуть. Использование стека для сохранения состояния процессора является обязательным и обеспечивает организацию вложенной обработки прерывания, если процедура обслуживания установит флаг IF. Тогда возникает опасность переполнения стека, поскольку каждое “вложение” будет использовать его для своих целей. Переполнение стека также может стать причиной зависаний. Длинные процедуры со сброшенным флагом IF могут привести к потере системного времени, поскольку “часы” операционной системы используют аппаратные прерывания от таймера.

Системный интерфейс PCI: Цикл ответа на прерывание

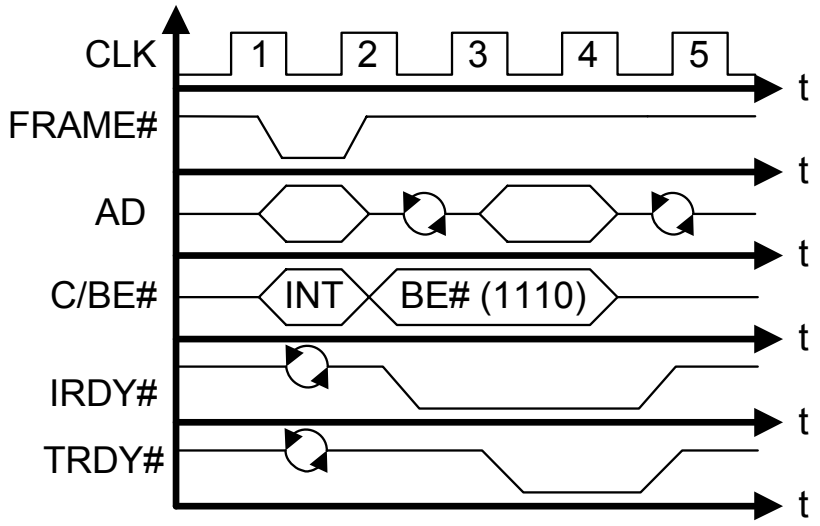


Рис. 3. 55. Цикл PCI – ответ на прерывание

Аппаратные способы реализации подсистемы прерываний

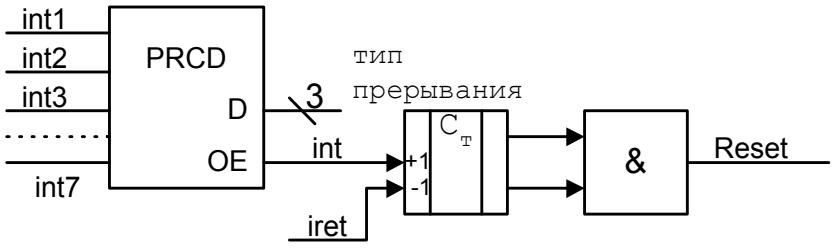


Рис. 3. 56. Схема аппаратного способа реализации подсистемы прерываний

PRCD – приоритетный шифратор, на который заводятся источники прерываний. Одновременно может присутствовать несколько активных сигналов запроса на прерывание от различных источников. PRCD кодирует номер источника прерывания с учетом приоритетов: самым высокоприоритетным является источник 0, а самым низкоприоритетным – источник 1. На выходе появляется 3-разрядный код номера источника прерывания и сам факт возникновения прерывания. Далее сигнал int поступает на счетчик уровня вложенности прерывания. Как только возникает 4-й уровень вложенности, происходит сброс счетчика. Тип прерывания определяет начальный адрес процедуры обработки прерывания. Для каждого источника он свой. Счетчик уменьшается командой возврата из прерывания.

Организация внешних прерываний

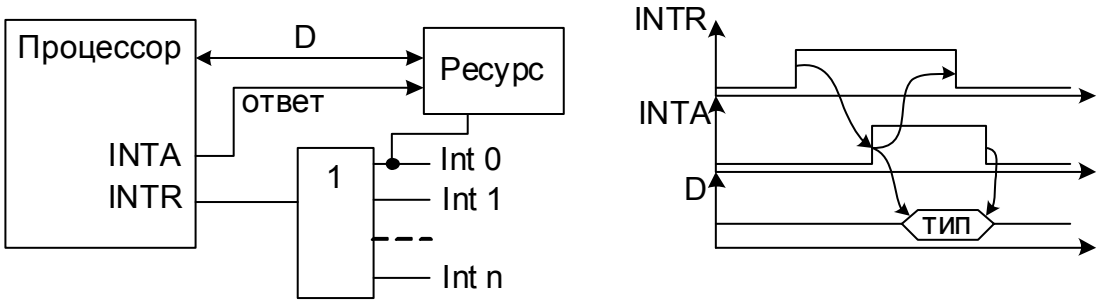


Рис. 3. 57. Организация внешних прерываний

У нас есть запрос на прерывание INTR. Он возникает асинхронно в некий момент времени. При возникновении этого сигнала внешние устройства, по отношению к

процессору, выставляют сигнал прерывания. Процессор на этот сигнал отвечает – INTA (необязательно сразу). Как только процессор ответил, что он перешел в состояние прерывания, наступает соответствующий цикл прерывание и в этом цикле внешнее устройство, вызвавшее прерывание передает тип прерывания. Поэтому по шине данных передается тип прерывания. Всего может быть 256 внешних источников прерывания. После того, как тип данных выставлен, процессору сообщается в рамках того же самого системного интерфейса, что данные выставлены. Через какое-то время заканчивается цикл прерывания.

Характеристики подсистемы прерываний

13. Время реакции на прерывание (в мкс) – определяет максимальное время от возникновения сигнала прерывания до запуска процедуры обработки прерывания (от 10 до 100 мкс – типовое время).
14. Число источников прерывания (типов).
15. Тип смены приоритетов линий прерывания.
16. Инициализация прерывания фронтом или уровнем.
17. Возможность маскирования источников.
18. Возможность получить состояние сигнала на входе прерывания.

Контроллер приоритетных прерываний

В связи с тем, что устройство процессора является сложным автоматом и требуется наличия у него очень много входов и выходов, а число входов (выходов) для изготавливаемых микросхем ограничено, подсистема прерываний строится следующим образом. Имеется один (в лучшем случае два) сигнал у процессора, который называется прерывание, а источник прерывания передается сложным образом в рамках целого цикла системного интерфейса (временная диаграмма на рис. 3.57). В связи со сложными действиями, которые надо организовать по этому взаимодействию, каждому устройству, которое хочет быть источником прерывания реализовывать эти функции становиться не целесообразно. Поэтому в системе выделяют контроллеры, которые называются контроллеры приоритетных прерываний, так как у них входные линии имеют некий приоритет.

Контроллер приоритетных прерываний – это пассивное устройство на системном интерфейсе. Передает номер источника прерывания, которое возникло, но перед тем как передать, оно должно зафиксировать запросы и сформировать по какому-то приоритету этот запрос.

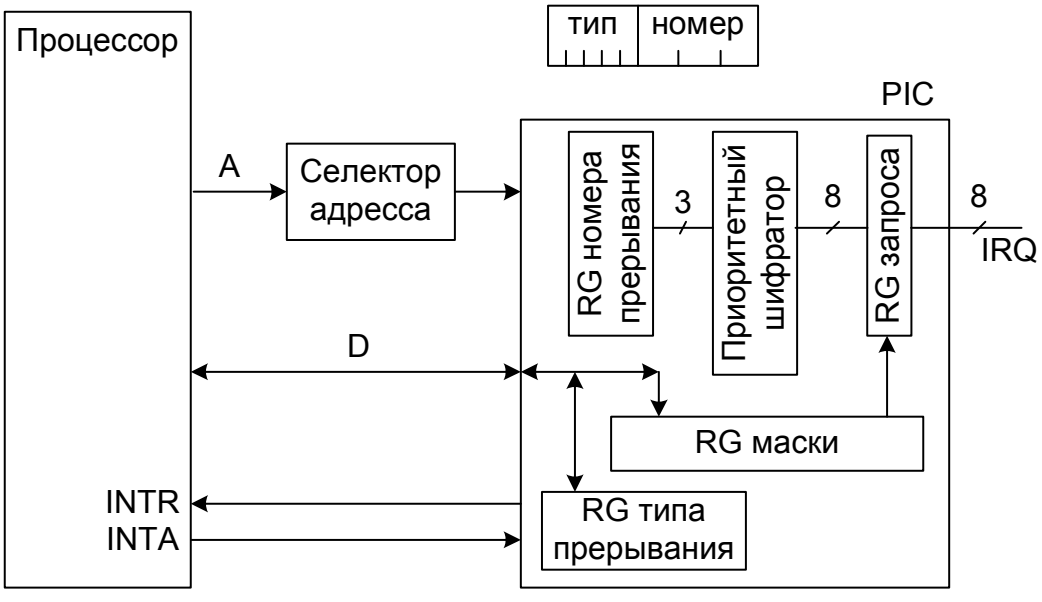


Рис. 3. 58. Контроллер приоритетных прерываний

Селектор адреса необходим для опознавания обращения процессора в адресное пространство контроллера прерываний, содержащий как минимум 2 регистра (ячейки): маска прерываний и начальный тип прерывания. В простейшем случае селектор адреса это дешифратор.

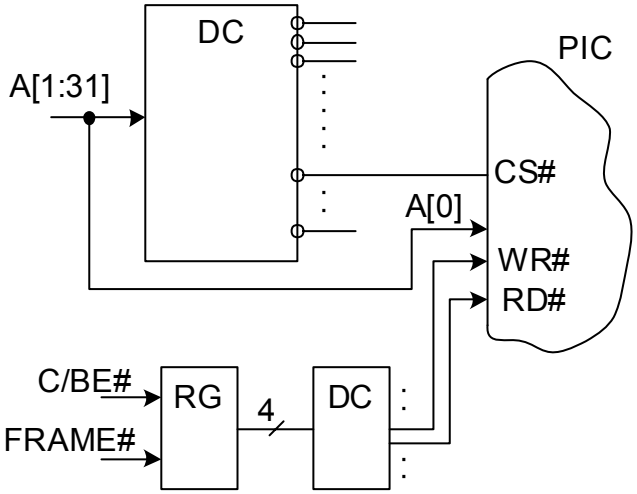


Рис. 3. 59. Структура селектора адреса

Программируемый контроллер прерываний (PIC)

Рассмотрим пример программируемого контроллера прерываний для многопроцессорных систем.

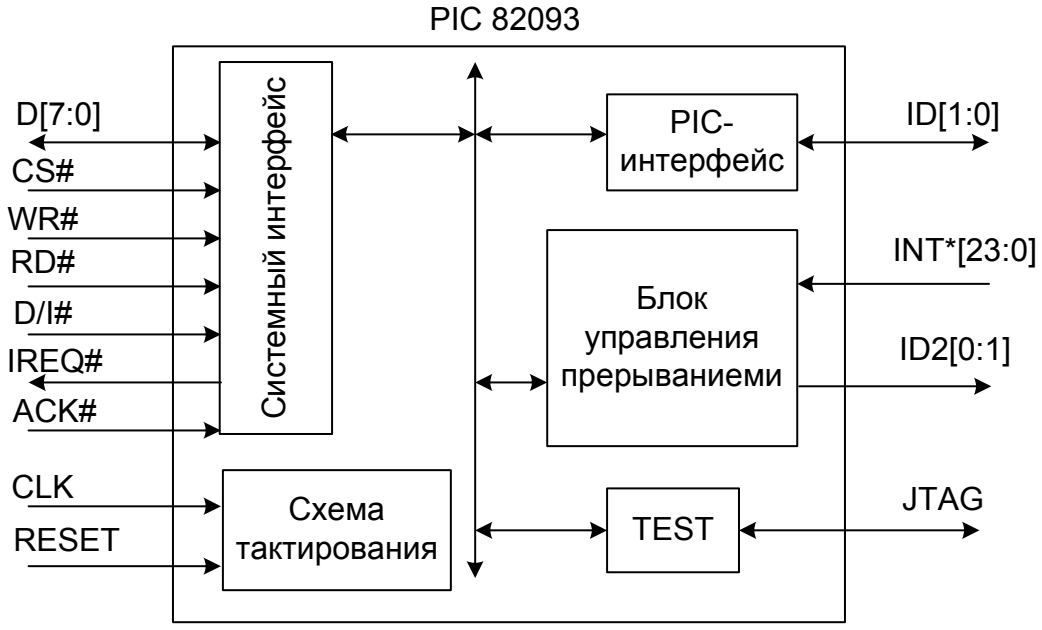


Рис. 3. 60. Структурная схема программируемого контроллера прерываний

1. **D[7:0]** –шина данных контроллера. Используется для выдачи номера типа прерывания, а также для обмена данными между системным интерфейсом и устройствами.
2. **CS#** - выбор контроллера. Когда сигнал активен, контроллер воспринимает управляющие сигналы системного интерфейса.
3. **RD#** и **RW#** - сигналы чтения и записи регистров контроллеров.

4. **D/I#** - высокий уровень – чтение/запись данных, низкий уровень – чтение/запись индексного регистра. Заводится на этот вход младший разряд шины адреса интерфейса.
5. **IREQ#** и **ACK#** - запрос на прерывание и разрешение прерывания (смотри временную диаграмму выше).
6. **INT*** - 24 линии запроса на прерывание от устройств.
7. **ID2** – шина идентификации процессора, который должен обработать прерывание.
8. **JTAG** – последовательный интерфейс тестирования контроллера в рамках интерфейса PCI.
9. **ID1** – шина идентификации ведомых (ведущих) контроллеров при расширении подсистемы прерываний.

Рассмотрим структурную схему многопроцессорной системы с несколькими контроллерами прерываний.

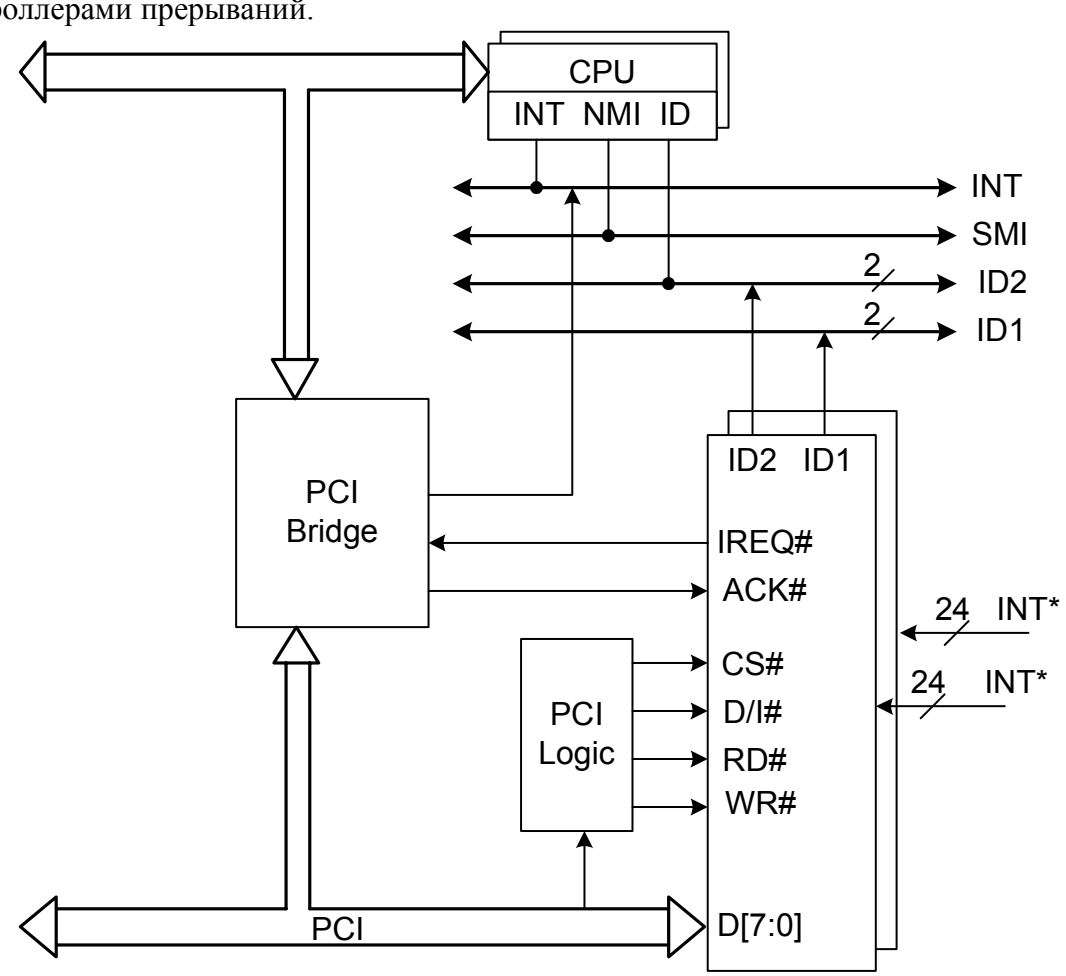


Рис. 3. 61. Структурная схема подсистемы прерываний многопроцессорной системы

Каждый процессор имеет свою логику обработки запросов на прерывание.

INT – обычный вход прерывания.

NMI – немаскируемый вход прерывания.

Логика работает тогда, когда на шине идентификации процессора ID2 выставлен номер процессора. Контроллер прерываний PIC является пассивным устройством на интерфейсе PCI. Для связи контроллера с интерфейсом PCI используется блок PIC-logic, который формирует сигналы чтения/записи регистров, тип регистра (данные или индекс), а также сигнал выбора контроллера, который формируется селектором адреса путём обработки адреса обращения с интерфейса PCI. По получению запроса на прерывание INTx от внешнего устройства контроллер обрабатывает его, как описано ранее,

выставляет сигнал запроса на прерывание IREQ# и ID2 и после подтверждения обработки прерывания сигналом ACK на шине данных PCI выставляет номер типа прерывания.

При каскадировании контроллеров, когда в системе используется несколько контроллеров к ведущему контроллеру подключаются ведомые (вторичные), выход IREQ которых подключается на один из входов ведущего (главного) контроллера. В этом случае используется шина идентификации контроллеров ID1. По получению сигнала ACK главный контроллер выставляет на шине ID1 номер того ведомого контроллера прерываний, который должен передать тип прерывания на шину данных PCI. Все контроллеры доступны через адресное пространство интерфейса PCI.

Внутренние регистры контроллера приоритетных прерываний

INDEX	Описание	Доступ
00h	ID	R/W
01h	Версия	RO
02h	Архитектурный идентификатор	RO
10÷8Fh	Таблица переадрессации (24x8)	R/W

Формат таблицы переадресации

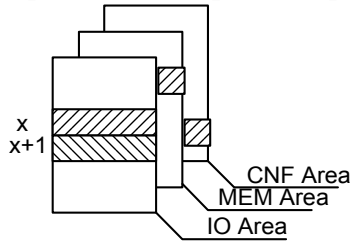
7:0	Вектор прерывания, чтение/запись (Interrupt vector R/W).
10:8	Режим обработки, чтение/запись. (Delivery mode R/W): 000 - фиксированный приоритет; 001 - наименьший приоритет (Lowest Priority); 010 - системное прерывание (SMI); 100 - немаскируемое прерывание (NMI); 111 - внешнее прерывание (ExtINT).
11	Тип контроллера: 0 – главный; 1 – ведомый.
12	Занятость контроллера: 1 - обработка прерываний; 0 - пассивное состояние.
13	Полярность сигнала прерывания (Pin Polarity R/W): 0 - активен высоким уровнем; 1 - активен низким уровнем.
14	Запрет передачи типа прерывания.
15	Триггерный режим (Trigger Mode): 0 - чувствительность к уровню (Level sensitive); 1 - чувствительность к переходу от 0 к 1 или от 1 к 0 (Edge sensitive).
16	Маска прерывания (Interrupt Mask R/W): 0 - прерывание обрабатывается (processity); 1 - замаскировано (masked).
17:55	Зарезервировано.
56	Назначение. В зависимости от типа контроллер содержит или идентификационный номер процессора (если тип 0), или свой идентификационный номер (если тип 1 - ведомый).

Тема 3.6. Подсистема ввода-вывода

Под **процессом ввода-вывода** понимают пересылку данных между оперативной памятью ЭВМ и периферийными устройствами с целью реализации иерархической организации памяти (виртуализации).

Принципы построения подсистемы ввода-вывода

1. Использование специальных устройств (мостов), называемых контроллерами периферийных устройств (КПУ), которые предназначены для сопряжения ПУ с системным интерфейсом требуемого уровня.
2. Архитектурно КПУ представляют собой совокупность трех адресных пространств (для самого сложного контроллера). Более простые контроллеры могут содержать одно или два адресных пространства, например, в интерфейсе ISA присутствуют только пространство ввода/вывода и пространство памяти, а в интерфейсе PCI присутствует еще и конфигурационное адресное пространство.



IO Area - пространство ввода/вывода
MEM Area - пространство памяти
CNF Area - конфигурационное пространство

Рис. 3. 62. Адресные пространства КПУ

В каждом из адресных пространств размещаются регистры (ячейки памяти контроллера), доступные через соответствующую область физического адресного пространства интерфейса.

Через пространство ввода вывода осуществляется пересылка данных. Содержимое регистра КПУ пересылается в ячейку основной памяти и обратно.

В адресном пространстве памяти контроллера отображаются в последовательных ячейках данные, полученные от ПУ и в случае необходимости пересылаются в основную память.

Конфигурационное пространство памяти используется для настройки и конфигурирования ПУ. Обмен данными через конфигурационное пространство не осуществляется. На интерфейсе ISA из-за отсутствия конфигурационного адресного пространства используется спецификация Plug and Play, где каждое устройство “говорит” само за себя.

3. Способы ввода/вывода:

) Программный. При котором программа (драйвер) исполняя команды ввода/вывода процессора читает или записывает регистры контроллера периферийных устройств. Реализуется средняя скорость передачи данных. Как правило, любой контроллер имеет 3 типа регистров:

- регистры команд, куда записываются коды действий, которые должен выполнить контроллер;
- регистр состояния. Программа, читая содержимое этого регистра, определяет состояние контроллера и ПУ;
- регистры дынных, через которые происходит обмен данными с ПУ.

Недостатки:

- Неэффективное использование процессора, т.к. процессор занимается вводом-выводом и вынужден ожидать готовность ПУ.
- Быстродействующие ПУ не могут быть обслужены через этот способ ввода-вывода, т.к. процессор исполняет программу, которая располагается в кэш-памяти.

- Для каждого контроллера и ПУ необходима своя, отличная от других программа ввода-вывода.

Достоинства:

- Требуется минимальный объем оборудования.
- Простота КПУ.
- Простота подключения различных типов ПУ.

) Ввод/вывод по прерываниям. При таком вводе/выводе драйвер устройства взаимодействует, как с контроллером приоритетных прерываний, так и с КПУ. Ввод/вывод по прерываниям используется, когда устройство обеспечивает среднюю скорость передачи данных, но с большими задержками появления 1-й порции данных. КПУ по завершению операции подготовки данных выставляет сигнал прерывание через системный интерфейс, к которому он подключён. Для того, чтобы сигнал привёл к прерыванию процессора, этот же драйвер программирует контроллер приоритетных прерываний, зная места физического подключения КПУ и заносит в таблицу векторов прерываний адрес процедуры обработки прерываний, чтобы продолжить операцию вводы/вывода.

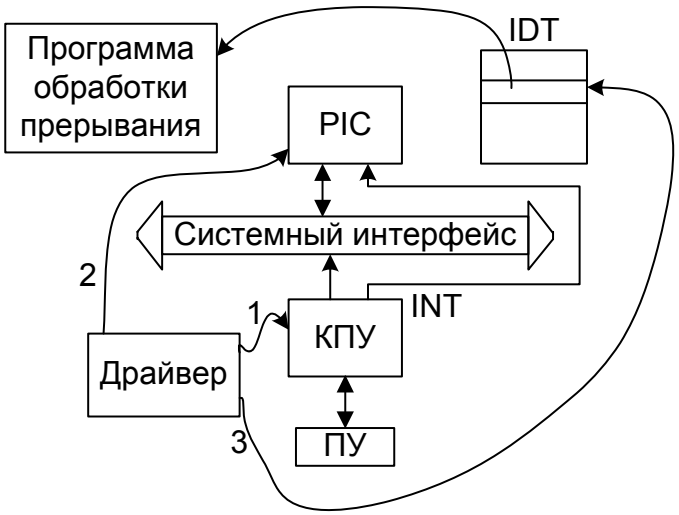


Рис. 3. 63. Структурная схема ввода-вывода по прерываниям

Недостатки:

- Усложняется КПУ.
- Может быть использован этот ввод/вывод для реакции на достаточно редкие события.
- Усложняется программа обработки прерывания, т.к. вынуждена реагировать и анализировать большое число событий ввода-вывода.

Достоинства:

- Позволяет разгрузить процессор от ожидания редко возникающих событий.
- Позволяет разделить процедуру ввода-вывода на относительно независимые части. Например: процесс обмена данными, процедура обработки событий от КПУ, инициализация и завершение ввода-вывода.

) Ввод/вывод через прямой доступ к памяти. При традиционной реализации используются специальные процессоры, по пересылке данных, называемые каналами DMA.

- В связи с развитием интегральной схемотехники КПУ иногда реализуются совместно с каналом прямого доступа и этот ресурс на интерфейсе называется активным.

1-й способ. Использование канала прямого доступа для организации ввода/вывода через DMA.

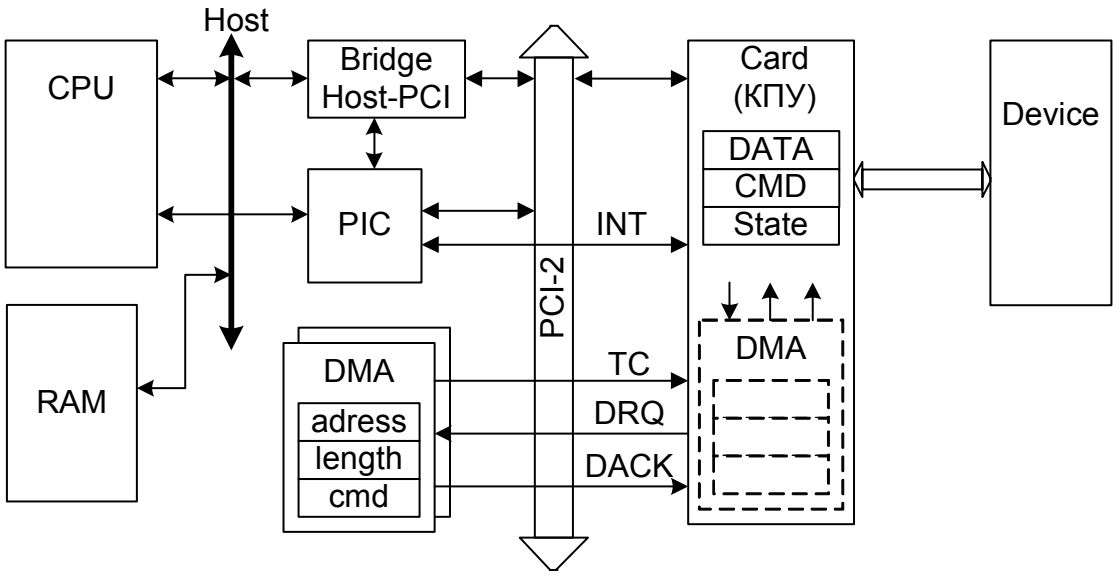


Рис. 3. 64. Структурная схема ввода-вывода через прямой доступ к памяти

DRQ – запрос прямого доступа.

DACK – ответ канала DMA после захвата системного интерфейса контроллеру ПУ.

- Контроллер DMA перед началом обмена программируется путём задания команды (чт/зап), задаётся начальный адрес ОП и длина пересылаемых данных. Получив команду на начало работы, канал ожидает прихода сигнала DRQ. Получив этот сигнал, канал захватывает, как активное устройство системный интерфейс выставляет текущий адрес ОП и выдаёт команду чтения или записи ОП. По готовности данных при чтении из ОП выдаётся сигнал DACK КПУ, который означает, что контроллер может выставить или считать данные с системного интерфейса. Получив сигнал DACK, КПУ считывает или записывает данные в соответствии с их назначением (задаётся при программировании КПУ перед вводом/выводом через DMA). Получив данные, устройство снимает сигнал DRQ и соответственно снимается сигнал DACK.
- После завершения обмена канал DMA увеличивает адрес и уменьшает размер своих регистров и переходит в состояние ожидания следующего сигнала DRQ.
- **TC** – окончание счёта. При передаче последней порции данных канал DMA формирует сигнал TC, после получения которого устройство не выставляет сигнал DRQ, т.к. обмен не завершён. Если данные при пересылке исчерпаны контроллером ПУ, то для остановки процесса ввода/вывода через DMA устройство генерирует прерывание. Процедура обработки прерывания завершает ввод/вывод через DMA. Прерывание генерируется контроллером ПУ также и по завершению обмена по инициативе канала DMA (при получении сигнала TC).
- Достоинства:

- значительно упрощается КПУ, т.к. оборудование DMA размещается как устройство на системно интерфейсе и используется многими устройствами;
- при пересылке данных в память из памяти не задействуется процессор, который может решать другую задачу.

- Недостатки:

- системный интерфейс должен содержать дополнительные сигналы;
- если нет ни одного устройства, которое работает через DMA, множество каналов DMA как ресурсы на системном интерфейсе не используются.

2-й способ. Использование канала прямого доступа для организации ввода/вывода через DMA.

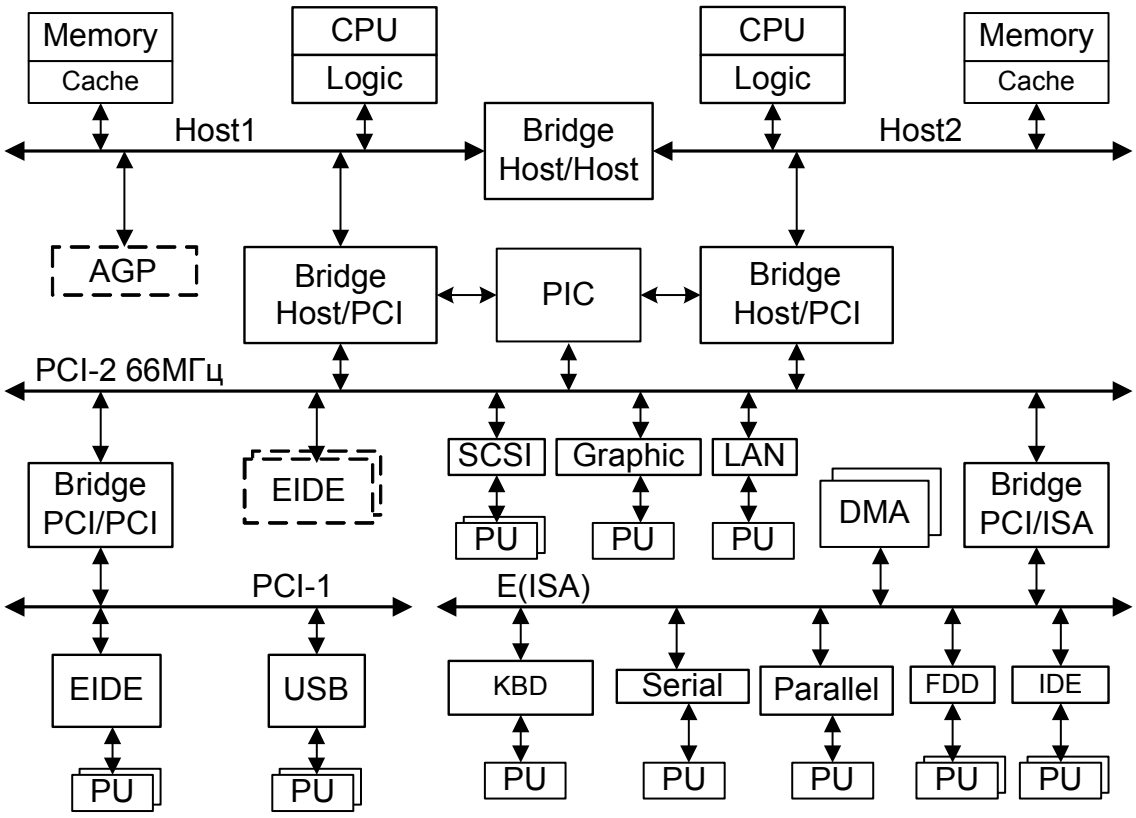


Рис. 3. 65. Подсистема ввода вывода

Схемотехника КПУ

Основные элементы:

1. Конфигурационное адресное пространство (при использовании ИМС PC-Card можно опустить).
2. Пространство ввода/вывода (есть всегда).
3. Пространство памяти (ПЗУ, ОЗУ).
4. Управление доступом (селекторы, защёлки, шинные формирователи).
5. Управление прерываниями (не всегда).
6. Управление прямым доступом.
7. Сопряжение с интерфейсом ПУ.

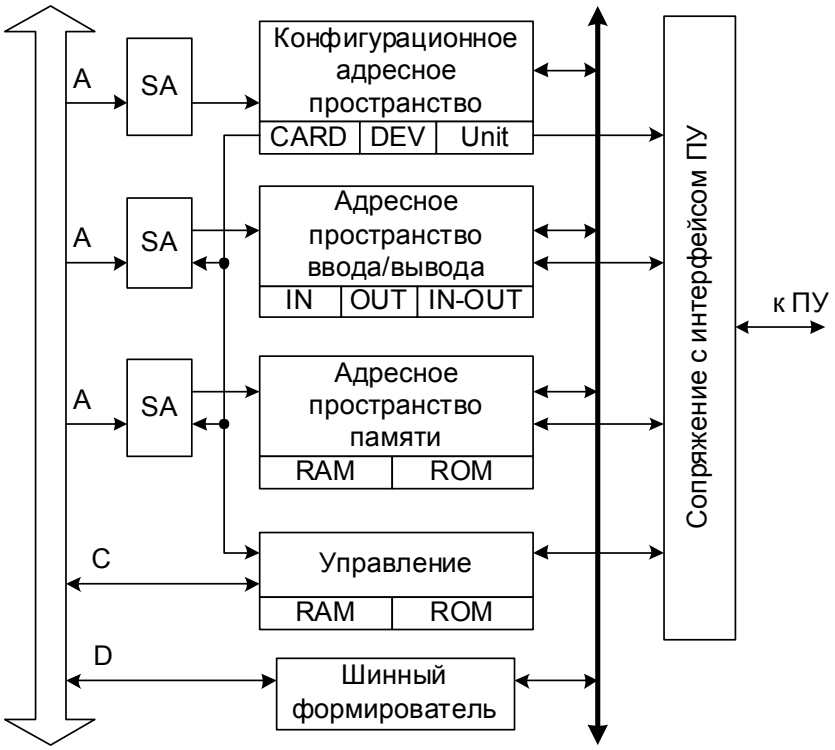


Рис. 3. 66. Структура КПУ

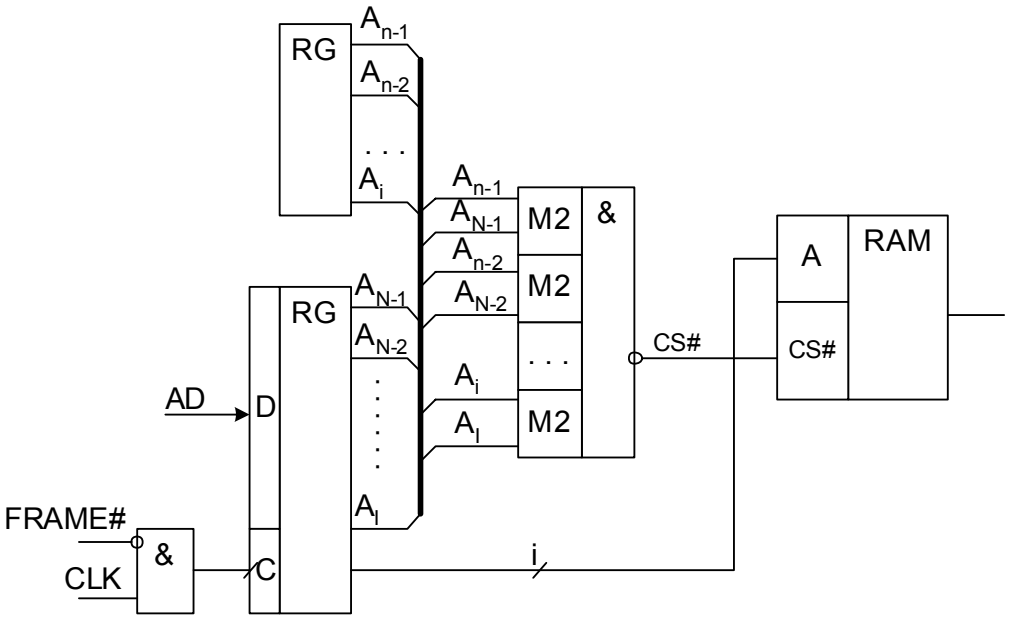


Рис. 3. 67. Схемотехника селектора адреса

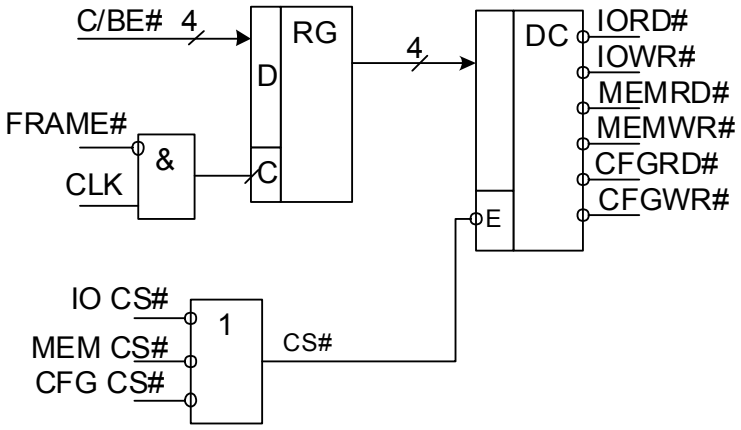


Рис. 3. 68. Схема обработки команды интерфейса

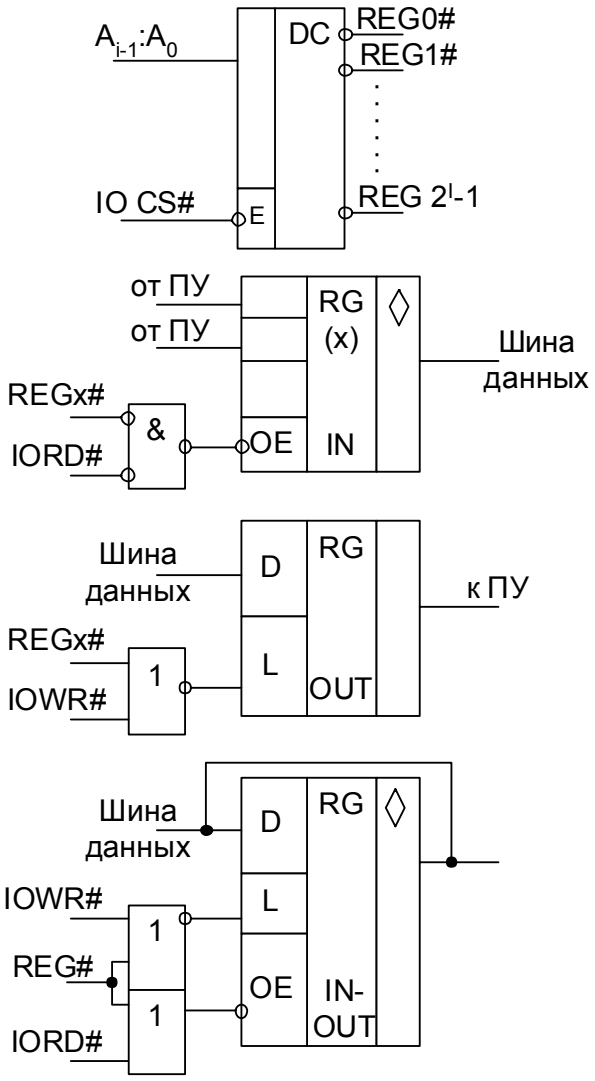


Рис. 3. 69. Адресное пространство ввода/вывода

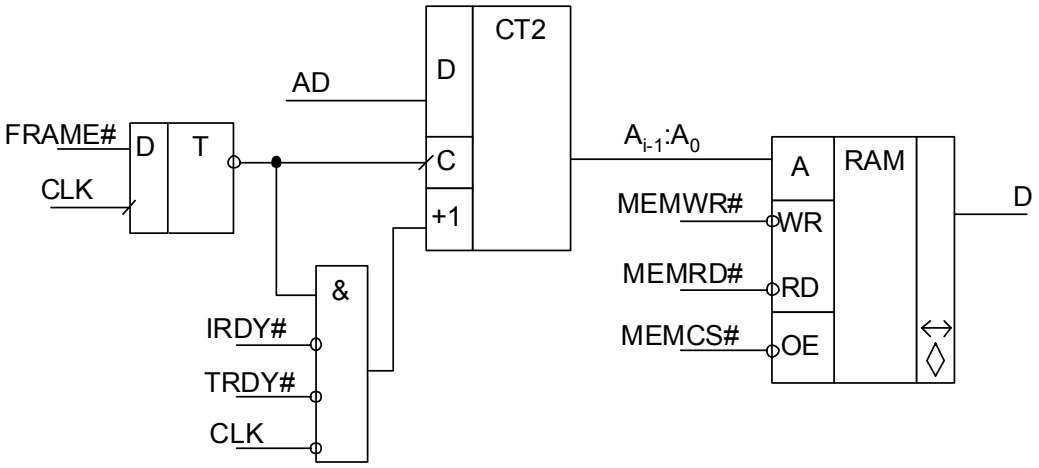


Рис. 3. 70. Адресное пространство памяти

Тема 3.7. Конфигурация и инициализация ЭВМ

Проблема конфигурации, заключается в следующем: может оказаться, что у нас имеется несколько однотипных устройств, которые находятся в системе, например, несколько однотипных контроллеров, которые требуют одних и тех же ресурсов. Каждому КПУ выделяется место в памяти с определенными адресами. Но, если нам потребуется включить два, три или четыре однотипных устройства (например, две видеокарты), тогда может возникнуть конфликтная ситуация. Например, если есть видеоадаптер на интерфейсе ISA, то ему выделяется память начиная с адреса A000 до C000, и, понятно, две видеокарты подключить нельзя, т.к. возникает проблема конфликта. Решением такой проблемы может быть использование перемычек. Так же проблема конфигурации возникает тогда, когда ресурсы на СИ (активные или пассивные) могут использовать одно и то же адресное пространство, одни и те же линии прерывания.

Проблема инициализации заключается в следующем. Оказывается мы должны при включении питания каким-то образом подготовить ЭВМ или комплекс, причем заведомо известно, что содержимое ОП теряется, поэтому мы должны предусмотреть некие ПЗУ, которые содержат код, исполняемый процессором, потому что изначально этот код не предусмотрен, который позволит, с учетом конкретной архитектуры ЭВМ, запустить (инициализировать) устройства входящие в систему. А устройств таких много: начиная от СИ, заканчивая котроллерами или ресурсами, работающими на СИ, причем многоуровневыми. Но может оказаться, что изменяется состав вашей вычислительной системы.

Первое и самое простое решение было реализовано на интерфейсе ISA Plug&Play. Это было сделано путём отключения всех устройств и подключения одного. Затем оно инициализируется и временно отключается и т.д.

Сейчас решение проблемы инициализации решается через конфигурационное адресное пространство интерфейса PCI следующим образом. Каждому ресурсу на интерфейсе PCI выделяется 2048 байт в этом адресном пространстве. Разные устройства на интерфейсе имеют различные подключения, а это значит, что по месту подключения можно выделить пересекающиеся конфигурационные адресные пространства. Спецификация PCI требует, чтобы при обработке обращения в конфигурационное адресное пространство ресурс на интерфейсе дешифрировал только младшие 11 разрядов адреса. Старшие разряды адреса зависят от места подключения к интерфейсу, могут быть различными и устройством не обрабатываются. Для предотвращения одновременной обработки запросов в конфигурационное пространство несколькими устройствами на интерфейсе PCI имеется сигнал IDSEL. И только, когда он активен, устройство обладает доступом в его конфигурационное пространство. Очевидно, этот сигнал заводится к

каждому месту подключения с дешифраторов старшей части адреса, находящегося в контроллере системного интерфейса.

Выделение конфигурационного адресного пространства на интерфейсе PCI

Каждое устройство на PCI имеет свою область памяти, используемую строго для конфигурации. Для каждого устройства изготовитель предусматривает свое адресное пространство.

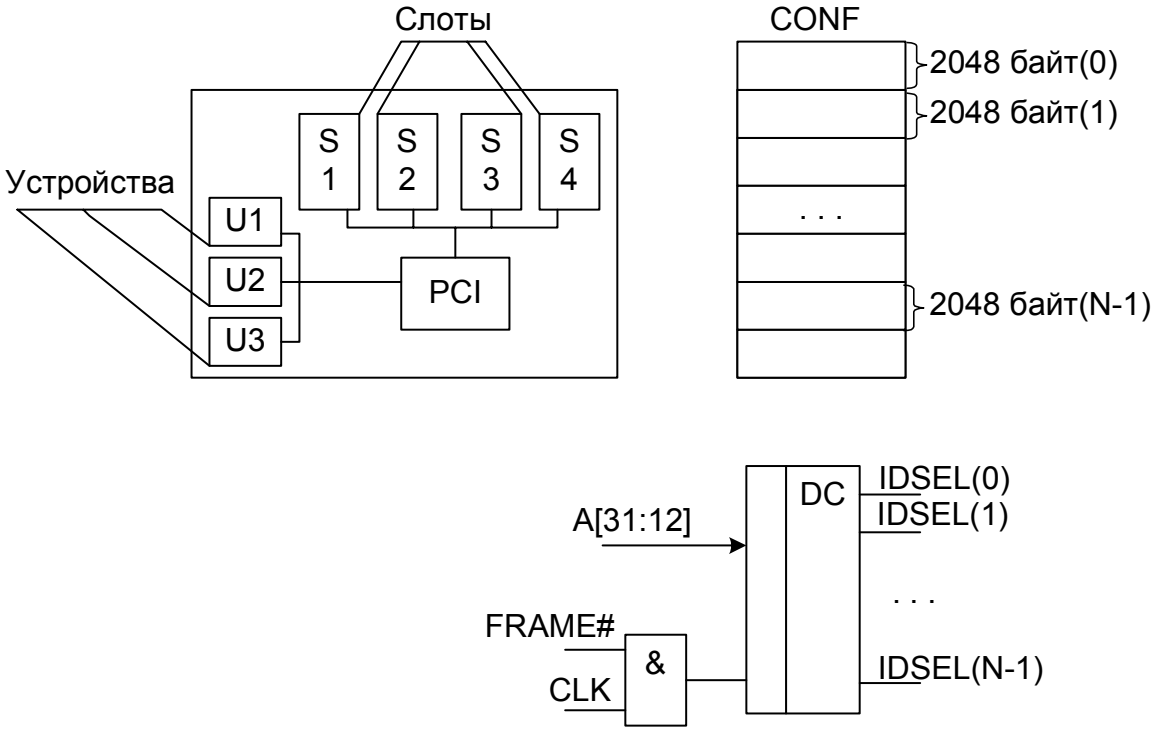


Рис. 3. 71. Выделение конфигурационного адресного пространства на интерфейсе PCI

Структура конфигурационного адресного пространства

Может оказаться, что на одной плате существуют многофункциональные устройства. Пример, на звуковой карте есть очень много устройств: входной и выходной каналы, регулятор громкости, микшер и т.д. и т.п. А конфигурационное адресное пространство одно. Каждое из этих устройств независимо: микшер может быть, а может и не быть, также и регулятор. Т.е. на контроллере может быть несколько логических устройств.

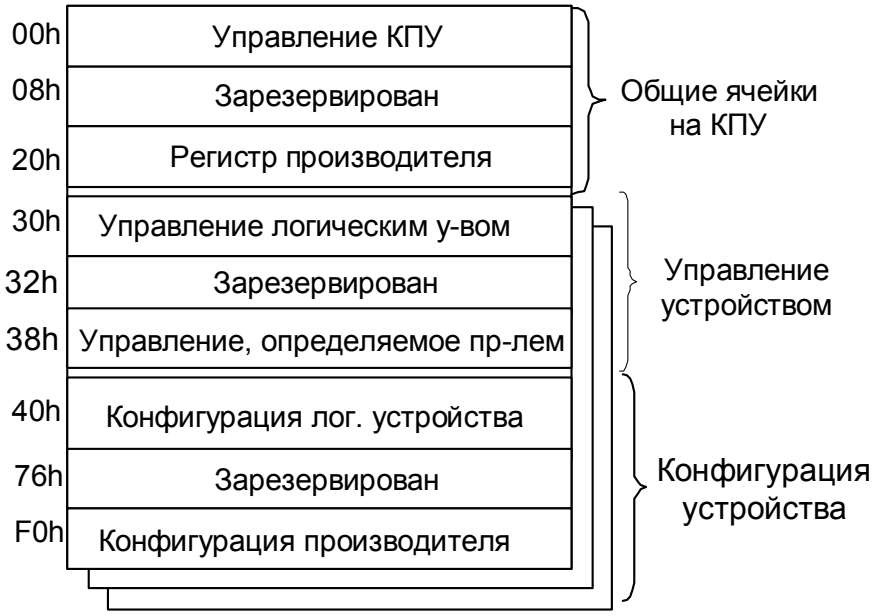


Рис. 3. 72. Конфигурационное адресное пространство

Это те регистры КПУ, которые он должен отобразить в адресном пространстве (АП) СИ PCI. Эти регистры разделены на две большие части. К первой части относятся те регистры, которые используются в одном экземпляре для всего контроллера, а ко второй части – ячейки для каждого из устройств, подключаемых к контроллеру.

На контроллере есть общее адресное пространство, независимое от числа логических устройств и входящих в контроллер и части, которые дублируются на каждом из устройств. Существуют специальные команды, которые отображают эти части в общее адресное пространство, путем занесения номера логического устройства в специальную ячейку. В свою очередь каждая из этих частей делится на 3 типовых части. Первая часть – управление КПУ в целом; вторая часть зарезервирована; третья часть – регистры, определяемые производителем. Последние части: управление, определяемое производителем и конфигурация производителя, т.е. те данные, которые использует драйвер производителя.

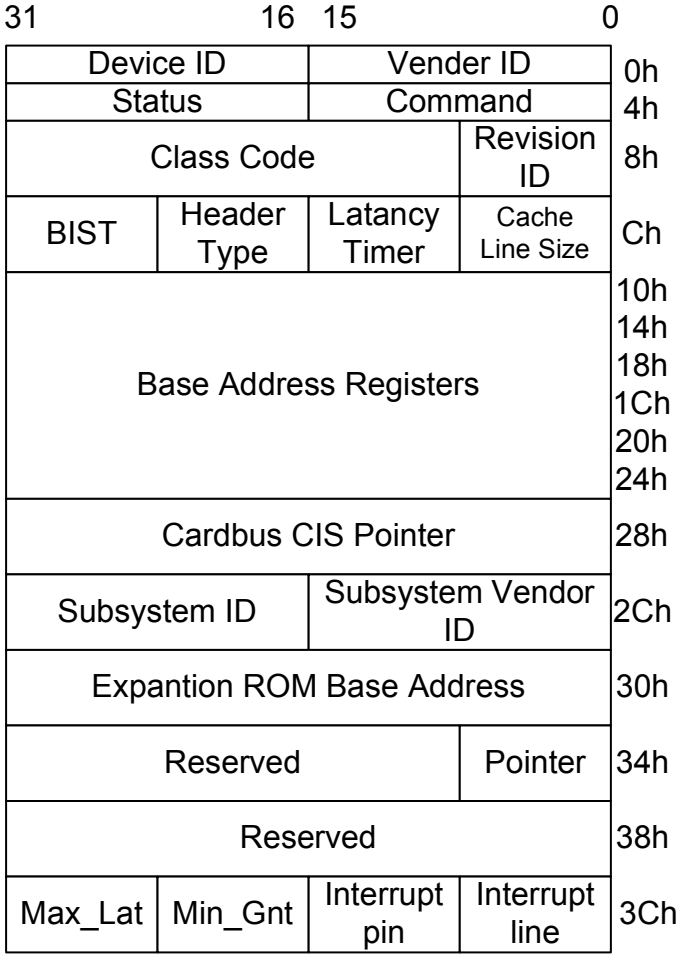


Рис. 3. 73. Регистры конфигурационного адресного пространства

Здесь используется последовательность байт в слове, называемая Big Endian (по младшему адресу выводится более важная информация, чем по старшему).

1. **Vender ID** – Идентификатор производителя. 0FFFFh – это признак того, что идентификатор производителя не определен, т.е. такого значения быть не может.
2. **Device ID** – идентификатор устройства, выделяемый производителем.
3. **Revision ID** – модификация устройства. Определяется производителем самостоятельно.
4. **Header Type** – тип заголовка. Имеет следующую структуру.

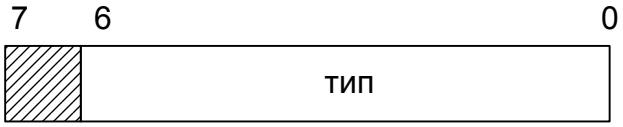


Рис. 3. 74. Структура Header Type

Если седьмой разряд равен, “1” – устройство многофункциональное, если – “0”, устройство не многофункциональное.

5. **Class Code** – Определяет тип устройства, необходимый для правильного выбора стандартного драйвера.
   - 0Bh – базовый класс.
   - 0Ah – подкласс устройства.

– 09h – тип программного интерфейса. Есть специальные приложения, в которых оговорены эти типы.
– К базовым классам:
– 0 – устаревшие устройства.
– 1 – контроллер массовой памяти.
– 2 – сетевой контроллер.
– 3 – контроллер дисплея.
– 4 – мультимедийные устройства.
– 5 – контроллер памяти.
– 6 – мост.
– 7 – коммуникационный контроллер.
– 8 – BIOS.
– 9 – входное устройство.
– 10 – контроллер криптографического закрытия данных.
– B – процессор.
– C – последовательный контроллер.
– D – беспроводный контроллер.
– FF – устройство неопределенного класса, т.е. не может быть отнесено ни к одному из этих классов.

6. **Command** – управление устройством. Структура показана на рис. 3. 75.

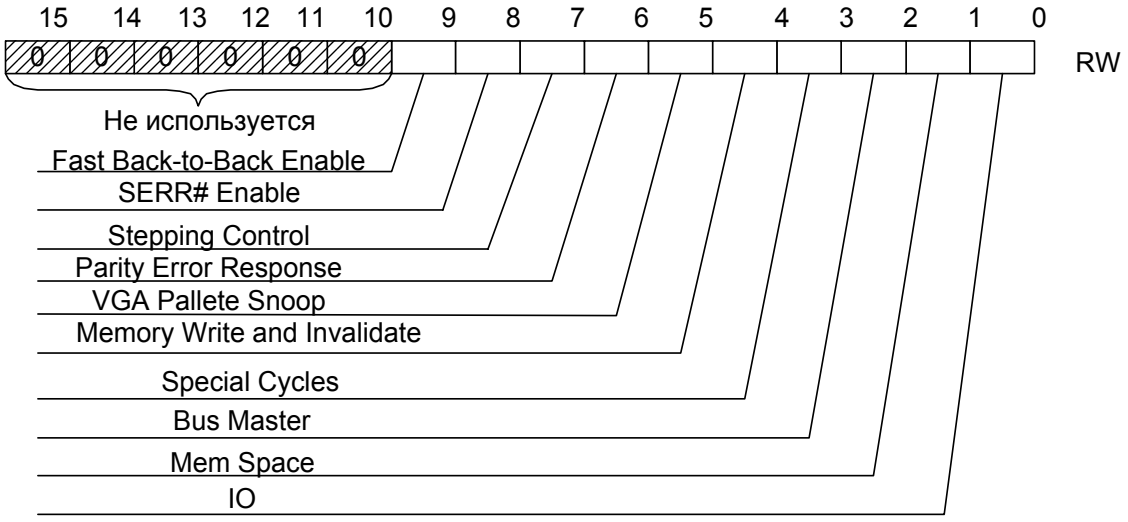


Рис. 3.75. Структура Class Code

– Разряд 0: Если “1”, то устройство логически отключается от СИ PCI. (Устройство не реагирует по вводу-выводу).
– Разряд 1: Пространство памяти.
– Разряд 2: Если “1”, то устройство логически отключается (не может запрашивать циклы СИ).
– Разряд 3: Если “1”, то запрещение поддержки специальных циклов.
– Разряд 4: Запись в память с проверкой. Если “1”, то контроллер может генерировать эти команды на СИ.
– Разряд 5: Для VGA определяет должно ли это устройство работать с палитрами или не должно. “1” – разрешение работы с регистрами палитры, “0” – запрещение.

– Разряд 6: Разрешение (“1”) или запрещение (“0”) генерации ошибки по паритету. Если “1”, то мы обязаны информировать об ошибке.

– Разряд 7: Управление конвейеризацией. Этот бит используется для определения, поддерживает ли конвейеризацию адреса данных. Если устройство не поддерживает этот режим, должно выставлять в этом разряде “0” независимо от передаваемых данных.

– Разряд 8: Разрешает (“1”) или запрещает (“0”) выдачу сигналов системной ошибки (т.е. устройству можно запретить выдавать этот сигнал на СИ).

– Разряд 9: Устанавливается для активных устройств (Master) и целевых устройств для выполнения ускоренного обмена (транзакций) по СИ, когда в одном цикле происходит не только чтение, но и запись данных.

7. **Status –** Состояние устройства.

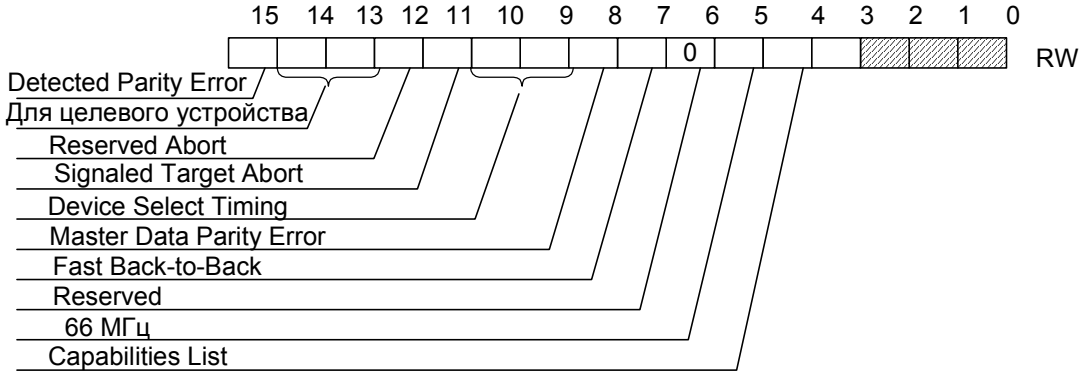


Рис. 3. 76. Структура Status

– Разряд 4: Список совместимости. Означает присутствие (“1”) или отсутствие (“0”) в конфигурационном пространстве списка совместимых устройств. Если список существует, то по смещению 34h находится указатель на этот список в конфигурационной памяти логического устройства.

– Разряд 5: Если (“1”), то устройство поддерживает 66МГц обмен данными, иначе 33Мгц.

– Разряд 6: Зарезервирован (всегда “0”).

– Разряд 7: “1” – поддержка режима Fast Back-to-Back.

– Разряд 8: “1” – означает, что во время работы устройства в качестве активного на СИ обнаружена ошибка по паритету.

– Разряд 9 и 10: Устройство сообщает, о своей скорости дешифрации адреса.

- 00 – Fast – быстрая.
- 01 – Medium – средняя.
- 10 - Slow – медленная.
- 11 – не оговорено.

– Разряд 11: “1” – устройство сигнализирует о преждевременном прекращении обмена с ресурсом на СИ.

– Разряд 12: “1” – был принят сигнал преждевременного прекращения обмена с ресурсом на СИ.

– Разряд 13 и 14: Похожи на 11 разряд и 12 разряд, только для активного устройства.

- Разряд 15: “1” – устройство обнаружило ошибку по паритету.

8. **Cache Line Size** – (4 разряда читаются или записываются). Определяет объем обмена в двойных словах для множественного чтения или записи. (Тут записывается не сама длина, а степень числа 2).
9. **Latency Timer –** (RO). Определяет, на сколько тактов устройство задерживает выдачу данных при пакетном обмене (т.е. на сколько это устройство задерживает получение 1-го байта при пакетном обмене).
10. **BIST (Built-in-Self-Test)** – Используется для управления и состояния встроенного тестирования.

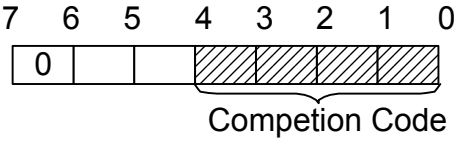


Рис. 3. 77. Структура BIST

- С 1-го по 4-й разряд – код завершения. 0000 – нет ошибки, остальные значения – это код ошибки.
- Разряд 7: Определяет, устройство поддерживает самотестирование (“1”) или не поддерживает (“0”).
- Разряд 6: Используется (если устройство самотестируемое) для перевода в состояние самотестирования (“1”) и выставляет через некоторое время код завершения. Устройство сбрасывает этот бит, когда тест завершен.

11. **Cardbus CIS Pointer** – Этот регистр используется для устройств, которые “хотят” (или “могут”) разделять устройство между внутренней шиной контроллера и PCI.
12. **Interrupt line** – Содержит номер прерывания контроллера прерываний (RW). Устройством этот регистр не используется (оно только хранит этот регистр). Этот регистр предназначен для драйверов и ОС. В него записывается значения процедурой начальной загрузки.
13. **Interrupt pin** – Здесь записывается та линия прерывания, которая разрешена на СИ (RO). 0 – линия не выделена. 1, 2, 3, 4 – соответственно INT A#, INT B#, INT C#, INT D#. Остальные числа не используются. Этот регистр только для чтения. Производитель устройства привязывает каждое логическое устройство к своей линии прерывания или же записывает 0, если линия прерывания не требуется.
14. **Min_Grnt и Max_Lat (R/O)** – значения в этом поле задаются изготовителем. Определяет времена в четвертях микросекунды, необходимых для Min_Grnt – пакетного обмена и Max_Lat – задает максимальный период (во времени) обращения к интерфейсу PCI, необходимый устройству.
15. **Subsystem Vendor ID (Device ID)** – идентификаторы подсистем. Эти регистры используются для уникальной идентификации устройств на интерфейсе PCI. Это означает, что даже для двух одинаковых плат совокупность этих регистров не могут совпадать, т.е. это индивидуальный серийный номер.
16. **Capabilities pointer** – содержится смещение, относительно начала, где находится список совместимых устройств (R/O). Каждый элемент списка состоит из восьмибитного поля идентификатора, распределенного PCI SIG (специальная организация) и восьмибитного указателя в конфигурационном пространстве для следующего элемента списка.

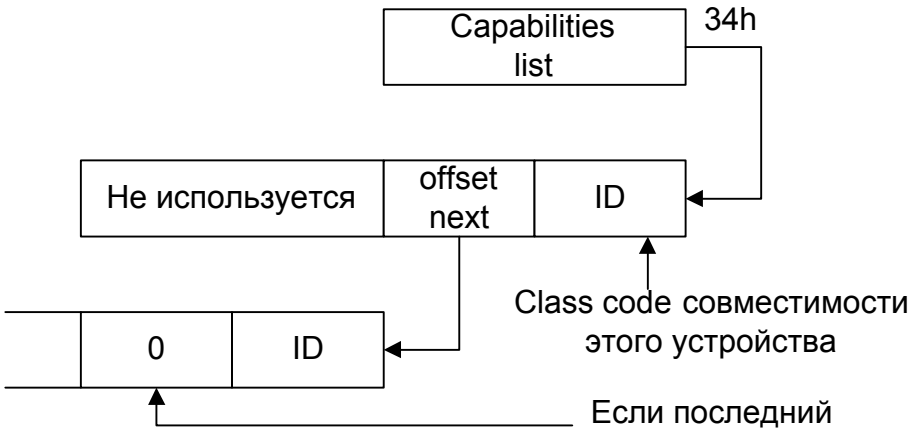


Рис. 3. 78. Структура Capabilities pointer

17. **Base Address Register** – программа начальной инициализации и конфигурации определяет через базовый регистр объем памяти и тип адресного пространства, требуемые устройством. Для этого в регистр записывается 32-х разрядное число 0FFFFFFFFh, которое опознается устройством, и следующее чтение из этого регистра содержит тип адресного пространства и его объем.

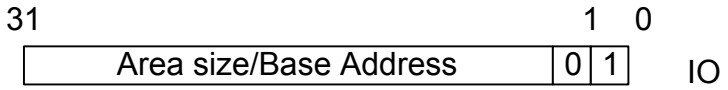


Рис. 3. 79. Структура Base Address Register (адресное пространство ввода-вывода)

Если младший разряд содержит “1”, то это адресное пространство ввода-вывода (IO). Второй разряд всегда “0” и не используется (Reserved). В остальных содержится размер области или же базовый адрес.

Если младший разряд содержит “0” , то это адресное пространство памяти.

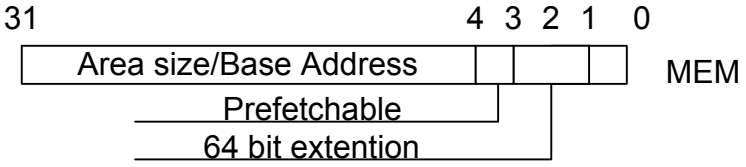


Рис. 3. 80. Структура Base Address Register (адресное пространство памяти)

Тогда первый и второй разряды содержат тип адресного пространства. 00 – 32-х разрядное адресное пространство, 10 - 64-х разрядное адресное пространство. В случае 10 следующий 32-х разрядный регистр после текущего участвует в определении размера и базового адреса.

Третий разряд устанавливается в “1”, если устройство выполняет перестановку байт, т.е. возможно обращение по чтению слова по любому адресу, не кратному слову. Устройство также поддерживает разряды разрешения байт на системном интерфейсе. “0” – в противном случае.

После выделения памяти в эти регистры записываются начальные адреса соответствующих адресных пространств. Устройства могут запрашивать адресные пространства большего объема, чем им необходимо для ускорения дешифрации адреса. В этом случае программа инициализации и конфигурации выделяет базовые адреса так, что устройству необходимо дешифрировать только соответствующую часть старших адресов. Программа инициализации и конфигурации определяет объем адресного пространства следующим образом: очищаются два (четыре) разряда после

чтения, инвертируются все биты и прибавляется “1”, т.о. обеспечивается автоматическое невыделение памяти.

18. **Expantion ROM Base Address** - (базовый адрес расширяющей памяти (R/O)). Содержит индивидуальный код, который применяется только для данного устройства (например, для видеокарты – код настройки цветов, и т.д.).

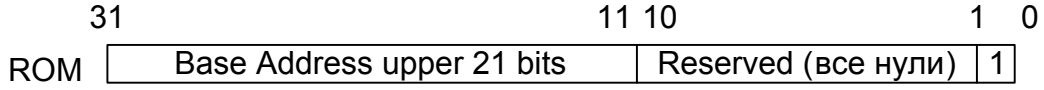


Рис. 3. 81. Структура Expantion ROM Base Address

Нулевой бит используется для управления доступом в память ROM. Если он равен “0” – ПЗУ запрещено (отключено), если равен “1” – ROM Enable, то при обращении по базовому адресу в этом регистре программа конфигурации и инициализации может читать данные из этого ПЗУ. Если память разрешена, то в разрядах с 11 по 31 определяется ее размер.

Рассмотрим структуру ПЗУ (рис. 3.82).

Рис. 3.82. Структура ПЗУ

ПЗУ содержит произвольное число образов памяти для каждой конфигурации в отдельности. Каждый образ содержит заголовок. Структура заголовка показана на рис. 3. 83.

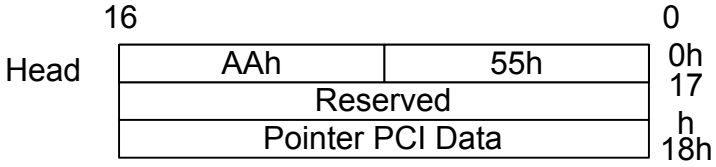


Рис. 3. 83. Структура заголовка

55h = 01010101

AAh = 10101010.

Эти значения записаны для того, чтобы узнать правильность чтения памяти, т.к. вероятность чтения этого числа при неисправности очень мала.

По смещения 18h содержится указатель специальной структуры данных, которая описывает идентификаторы устройства, изготовителя, длину области, длину образа, тип кода.

Известны следующие типы кодов:

- 0 – PC Intel x86;
- 1 – Sun;
- 2 – Hp.

Остальные зарезервированы.

После выбора и загрузки образа в ОП в область базового адреса записывается адрес этой области, чтобы драйвер знал, где эти данные расположены. При этом ПЗУ в устройстве запрещается путем записи “0” в нулевой бит. Для каждого типа этих образов имеется своя структура.

Процедура начальной загрузки.

1. Включение питания (power-on). Тестируется процессор, запуская внутри себя микрокод. Затем чтение команды, записанной по жёско-фиксированному адресу.
2. Power-On-Test – этот тест находится по начальному адресу в ПЗУ и выполняет:
   - расширенную диагностику процессора;
   - тест ядра аппаратной платформы (внутренний интерфейс, RAM, Cache II, системные устройства: контроллер прерывания и т.д.).
3. BIOS выполняет инициализацию конфигурацию системы и периферийных устройств:
   - поиск устройств и их изоляция;
   - настойка системных устройств (моты, системные контроллеры);
   - перенос BIOS в ОП;
   - процедура распределения ресурсов и инициализации устройств, которые были обнаружены путём сканирования конфигурационного адресного пространства. В BIOS есть специальная память (энергонезависимая), в которой хранится список обнаруженных устройств и распределённых ресурсов;
   - поиск и активация загрузочных устройств (boot device);
   - подготовка векторов прерывания для обращений к загрузочному устройству (чтение данных загрузочного устройства осуществляется через программное прерывание, т.к. они имеют абсолютные адреса). Выполняется чтение загрузочного сектора по адресу, фиксированному для каждой платформы и передаётся управление на считанный загрузчик ОС.
4. Работа загрузчика ОС:
   - загрузчик ОС получает списки устройств и распределённых им ресурсов из энергонезависимой памяти BIOS;
   - выполняет перераспределение ресурсов если необходимо;
   - загружает драйвера устройств и ядра ОС;
   - инициализация ОС (строятся системные таблицы, определяются системные процессы, устанавливаются процедуры обработки прерываний, инициализируются драйвера и т.д.).
   -

Раздел 4. Аналоговые и гибридные ЭВМ

Тема 4.1. Теория подобия и электрическое моделирование

Аналоговая величина

Отличительная особенность АВМ - аналоговая или непрерывная форма представления информации, переменных, использование аналоговых величин.

Аналоговая величина - непрерывная физическая величина, заменяющая искомую или заданную в решаемой задаче, связанная с ней масштабным соотношением.

АВМ - вычислительная машина, производящая операции над аналоговыми величинами.

В процессе вычисления или управления ЭВМ приходится решать разные задачи. Некоторые задачи предполагают получение числового результата, результатом решения других задач являются функции, т. е. математическая зависимость выходной переменной от входных. В простейшем случае функции одной переменной результат решения задачи может быть представлен непрерывной линией на листе бумаги.

Аналоговая форма представления информации позволяет получить значения функции в любой точке при любых значениях независимых переменных, причем всякое изменение переменных практически мгновенно отражается на значении функции. АВМ способна выдать результат в виде непрерывной линии на экране индикатора или на бумаге самописца в реальном масштабе времени, виде непрерывной зависимости напряжения от времени.

В АВМ функция формируется как результат взаимодействия отдельных блоков, соединенных друг с другом в соответствии с формулой, выражающей значения функции.

Машинной переменной может быть масса и давление, температура и ток. Но наибольшее применение нашли АВМ, в которых машинные переменные представляются в виде электрических напряжений. Напряжение легко формировать и менять, например, с помошью потенциометров, напряжение легко измерить и стабилизировать; напряжение используется как носитель информации в системах автоматического регулирования или управления; в виде напряжения обычно выдается сигнал с датчиков; напряжение может управлять двигателем. Используя сигналы в виде напряжений, легко создать блоки сложения и умножения, интегрирования и дифференцирования.

Решающий усилитель

Операционный усилитель (ОУ) – усилитель электрических сигналов, предназначенный для выполнения различных опреаций над аналоговыми величинами при работе в схеме с отрицательной обратной связью. В качестве ОУ обычно используют усилитель постоянного тока, способный усиливать как переменный, так и постоянный ток, т.е. сигнал, в спектральный состав которого может входить нулевая частота.

В интегральном исполнении современный ОУ обычно имеет один выход и два входа (прямой и инвертирующий) (рис. 4.1).

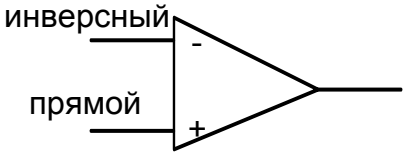


Рис. 4.1. Схема решающего усилителя

Решающий усилитель – функциональный узел, состоящий из ОУ и внешних элементов, образующих цепи отрицательной и положительной обратной связи, и выполняющий операции над аналоговыми веоичинами.

В общем виде схема решающего усилителя на базе дифференциального ОУ представлена на рис. 4.2.

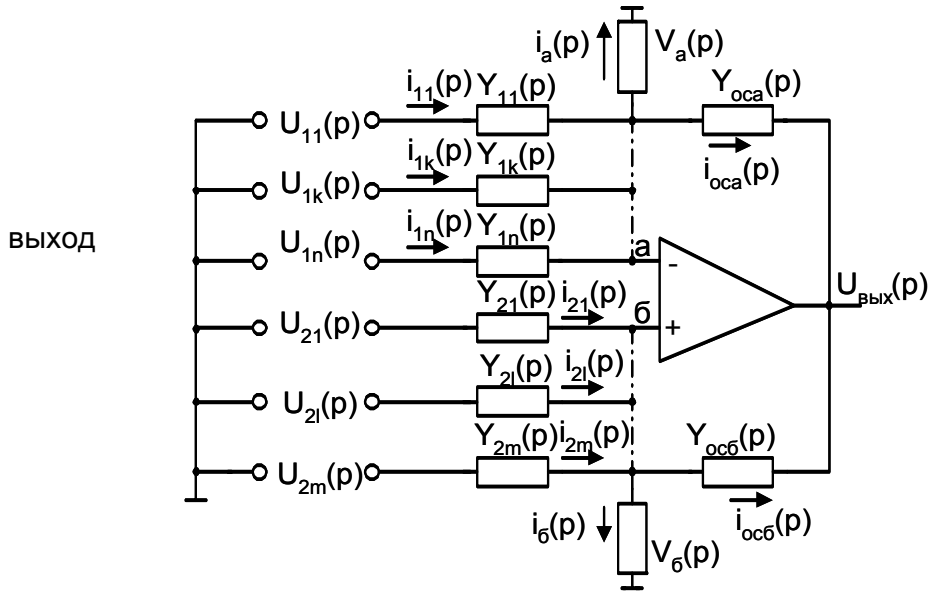


Рис. 4.2. Схема решающего усилителя в общем виде

Используя закон Кирхгофа для алгебраической суммы токов в узловой точке, составим уравнение, описывающее происходящие в схеме процессы:

$$\sum_{k=1}^{n} i_{1k}(p) = i_a(p) + i_{oca}(p); \quad \sum_{l=1}^{m} i_{2l}(p) = i_б(p) + i_{осб}(p).$$

Выражая токи через соответствующие разности напряжений и проводимости цепей $Y(p)$ и учитывая формулу $U_+ \approx U_-$, получим систему уравнений для решающего усилителя.

$$\sum_{k=1}^{n} \left[U_{1k}(p) - U_a(p)\right] \cdot Y_{1k}(p) = U_a(p) \cdot Y_a(p) + \left[U_a(p) - U_{вых}(p)\right] \cdot Y_{oca}(p)$$

$$\sum_{l=1}^{m} \left[U_{2l}(p) - U_б(p)\right] \cdot Y_{2l}(p) = U_б(p) \cdot Y_б(p) + \left[U_б(p) - U_{вых}(p)\right] \cdot Y_{осб}(p)$$

Из этой системы уравнений определим

$$U_{вых}(p) = \sum_{l=1}^{m} \left[k_{2l}(p) \cdot U_{2l}(p)\right] - \sum_{k=1}^{n} \left[k_{1k}(p) \cdot U_{1k}(p)\right]$$

, где

$$k_{2l} = \frac{Y_{2l}(p) \cdot \left[Y_a(p) + Y_{oca}(p) + \sum_{k=1}^{n} Y_{1k}(p)\right]}{Y_{oca}(p) \cdot \left[Y_б(p) + Y_{осб}(p) + \sum_{l=1}^{m} Y_{2l}(p)\right] - Y_{осб}(p) \cdot \left[Y_a(p) + Y_{oca}(p) + \sum_{k=1}^{n} Y_{1k}(p)\right]},$$

$$k_{1n} = \frac{Y_{1k}(p) \cdot \left[Y_б(p) + Y_{осб}(p) + \sum_{l=1}^{m} Y_{2l}(p)\right]}{Y_{oca}(p) \cdot \left[Y_б(p) + Y_{осб}(p) + \sum_{l=1}^{m} Y_{2l}(p)\right] - Y_{осб}(p) \cdot \left[Y_a(p) + Y_{oca}(p) + \sum_{k=1}^{n} Y_{1k}(p)\right]}$$

Т.о., решающий усилитель на базе дифференциального ОУ выполняет операцию вычитания алгебраических сумм входных напряжений, подаваемые через входные проводимости $Y_{1k}(p)$ на инвертирующий вход ОУ. Причем каждое напряжение одновременно умножается на свой собственный коэффициент передачи $k_{1k}(p)$ или $k_{2l}(p)$. Выбирая соответствующим образом электронные цепи для реализации проводимостей рассматриваемого решающего усилителя, можно осуществлять сложные преобразования зависящих от времени входных напряжений $U_{1k}(t)$ и $U_{2l}(t)$ в зависящее от времени выходное напряжение $U_{вых}(t)$.

Наличие в схеме цепей обратной связи, в том числе положительной, требует решение в каждом конкретном случае проблемы устойчивости создаваемой схемы. Это несколько затрудняет широкое использование возможностей схемы, но при создании специализированных аналоговых устройств не является существенным ограничением, поскольку достигаемый выигрыш в простоте реализации требуемой функции может быть существенным.

Можно заметить, что коэффициенты передачи $k_{2k}(p)$ и $k_{2l}(p)$ описываются громоздкими выражениями. Существенно упростить формулы и облегчить задачу практического использования схемы можно, если выполнить следующие равенства:

$Y_{осб}(p) = Y_a(p) = Y_б(p) = 0$, т.е. из схемы убирается цепь положительной обратной связи и несколько других проводимостей. В этом случае для коэффициентов справедливы равенства:

$$k_{2l} = \frac{Y_{2l}(p) \cdot \left[Y_{oca}(p) + \sum_{k=1}^{n} Y_{1k}(p) \right]}{Y_{oca}(p) \cdot \sum_{l=1}^{m} Y_{2l}(p)}, \; k_{1k} = \frac{Y_{1k}(p)}{Y_{oca}(p)}.$$

Операции над аналоговыми величинами

В качестве аналоговых величин в АВМ используют электрические напряжения. Электрическое напряжение, как известно, легко преобразовать в другое используя потенциометр. Последний находит широкое применение для умножения напряжения (машинной переменной) на постоянный коэффициент, лежащий в диапазоне от 0 до 1. Выполняющие функцию умножения потенциометры называют операционными. Схема включения операционного потенциометра представлена на рис. 4.3. Поскольку выходное сопротивление потенциометра обычно велико, то подключение нагрузки изменяет коэффициент передачи α. Чтобы исключить влияние этого недостатка на точность решения задачи, установку требуемого коэффициента производят для нагруженного потенциометра.

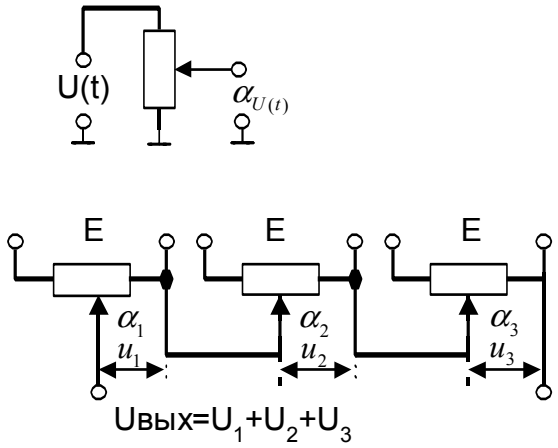


Рис. 4.3. Схема включения операционного потенциометра

Часто используются однооборотные потенциометры, не имеющие шкалы значений коэффициента передачи. Для установки требуемого коэффициента передачи на вход потенциометра подают стабилизированное известное напряжение (например, 10 В). Контролируя напряжение на выходе связанного с потенциометром усилителя с помощью вольтметра относительно общей шины или более точным компенсационным методом, добиваются требуемого коэффициента вращением движка потенциометра.

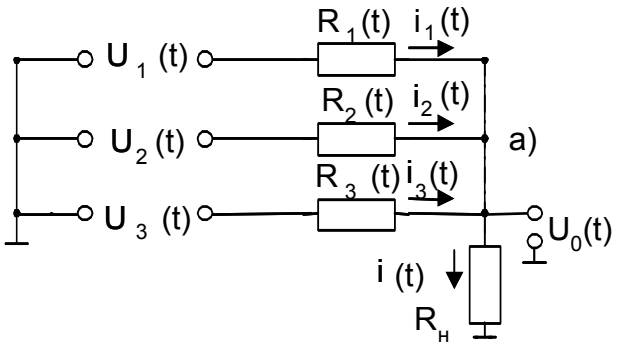


Рис. 4.4. Схема выполнения математических операций в АВМ

Чтобы выполнить операцию сложения электрических напряжений, задаваемых с помощью потенциометров, достаточно создать последовательную цепь (рис. 4.4). Выходное напряжение окажется суммой взвешенных с помощью потенциометров напряжений. Но применение данной схемы предполагает наличие большого числа независимых источников напряжений Е, что в большинстве случаев оказывается неудобным. Поэтому наибольшее применение нашел метод суммирования токов.

Задача создания аналогового устройства, реализующего принцип суммирования токов, формулируется следующим образом:

простейшим образом преобразовать каждое входное напряжение в пропорциональный ток, просуммировать полученные токи и выходной ток преобразовать в пропорциональное выходное напряжение.

В данной схеме выходное напряжение формируется как падение напряжения на сопротивлении нагрузки Rн и равно Ua(t). Следуя закону Кирхгофа, для узловой точки а справедливо $\sum_{l=1}^{n} i_l(t) = i(t)$.

Выражая токи через напряжения и сопротивления, получим $\sum_{l=1}^{n} \frac{U_l(t) - U_a(t)}{R_l} = \frac{U_a(t)}{R_H}$.

Отсюда $U_a(t) = \sum_{l=1}^{n} \frac{1}{R_l\left(1/R_H + \sum_{l=1}^{n} 1/R_l\right)}$, $U_l(t) = \sum_{l=1}^{n} k_l U_l(t)$.

Существенным недостатком схемы, ограничивающим ее применение, является зависимость коэффициентов передачи напряжений как от сопротивления нагрузки, так и от числа суммируемых напряжений. Исключить зависимость коэффициентов передачи от числа суммируемых напряжений можно, если при любом числе переменных обеспечивать потенциал точки а равный нулю, т. е. создать потенциально заземленную точку. В этом случае для выходного тока суммирующей цепи справедливо: $i(t) = \sum_{l=1}^{n} U_l(t) R_l$.

Но возникают две проблемы на пути использования данного уравнения при реализации суммирующей схемы:

- создание потенциально заземленной точки;
- превращение пропорционального сумме входных напряжений выходного тока в пропорциональное выходное напряжение, которое не будет зависеть от сопротивления нагрузки.

Все проблемы легко решаются использованием на выходе суммирующей цепи специального усилителя постоянного тока, который называется операционным усилителем. При решении дифференциальных уравнений с помощью АВМ возникает проблема интегрирования зависящего от времени напряжения по времени. Эта задача может быть решена с помощью конденсатора, емкость которого равна С. Напряжение на конденсаторе связано с током через него следующей зависимостью:

$U_C(t) = \frac{1}{C}\int_0^t i(t)dt + U_C(0)$, где $U_C(0)$ - начальное напряжение на конденсаторе.

Чтобы использовать эту способность конденсатора, необходимо пропускать через него ток $i(t)$, пропорциональный входному напряжению. А преобразование напряжения в ток возможно с помощью резистора при условии создания потенциально заземленной точки на втором конце этого резистора. В случае $i(t) = U(t)/R$ напряжение на конденсаторе равно $U_C(t) = \frac{1}{RC}\int_0^t i(t)dt + U_C(0)$, т. е. и при создании интегрирующей схемы требуется использование операционного усилителя как для создания потенциально заземленной точки, так и для обеспечения низкого выходного сопротивления у схемы, для обеспечения возможности подключения к схеме низкоомной нагрузки.

Реализация суммирования и масштабирования

Суммирование аналоговых величин в АВМ осуществляется путем суммирования пропорциональных аналоговым напряжениям аналоговых токов, которые формируются с помощью резисторов. Схема блока суммирования представлена на рис. 4.5.

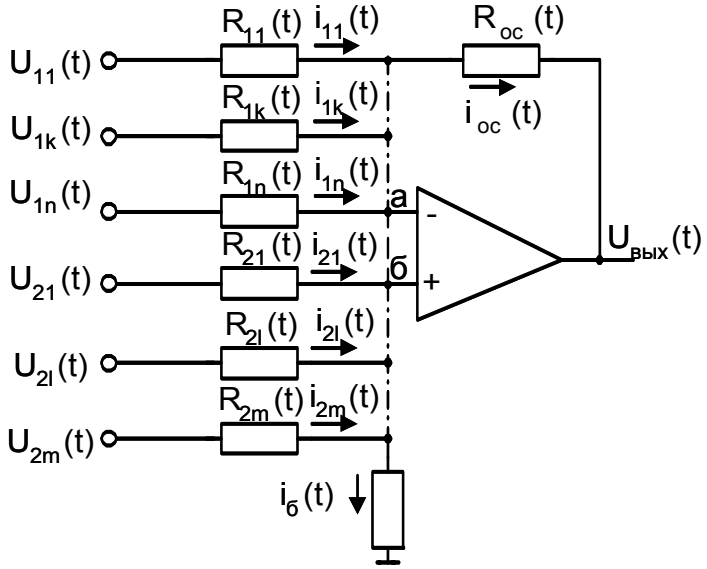


Рис. 4.5. Общая схема блока суммирования

Все напряжения в схеме формируются относительно общей шины нулевого потенциала. Используя первый закон Кирхгофа для узлов а и б и учитывая равенство напряжений в узлах, получим систему уравнений:

$$\sum_{k=1}^{n}\frac{[U_{1k}(t)-U_a(t)]}{R_{1k}}=\frac{U_a(t)-U_{вых}(t)}{R_{oc}};\ \sum_{l=1}^{m}\frac{[U_{2l}(t)-U_б(t)]}{R_{2б}}=\frac{U_б(t)}{R_б};\ U_a(t)\approx U_б(t).$$

Т.к. входные и выходные напряжения и токи в схемах зависят от времени, с целью упрощения записи формул время в обозначениях напряжений и токов будем опускать.

Перегруппируем члены в уравнениях системы:

$$\sum_{k=1}^{n}\frac{U_{1k}}{R_{1k}}+\frac{U_{вых}}{R_{oc}}=U_a\left(\frac{1}{R_{oc}}+\sum_{k=1}^{n}\frac{1}{R_{1k}}\right);\ \sum_{l=1}^{m}\frac{U_{2l}}{R_{2l}}=U_б\left(\frac{1}{R_б}+\sum_{l=1}^{m}\frac{1}{R_{2l}}\right);\ U_a\approx U_б.$$

Выразим напряжение в точке *а* из последних двух уравнений и подставим полученное выражение в первое уравнение. Это позволит определить выходное напряжение схемы:

$$U_{вых}=\sum_{l=1}^{m}k_{2l}U_{2l}-\sum_{k=1}^{n}k_{1k}U_{1k}\text{, где }k_{2l}=\frac{1/R_{oc}+\sum_{k=1}^{n}\frac{1}{R_{1k}}}{1/R_б+\sum_{l=1}^{m}\frac{1}{R_{2l}}}\cdot\frac{R_{oc}}{R_{2l}},\ k_{1k}=\frac{R_{oc}}{R_{1k}}\ (*).$$

Схема выполняет вычитание алгебраических сумм взвешенных входных напряжений. Можно заметить, что при разрыве цепи отрицательной обратной связи, т. е. при $R_{oc}\to\infty$, коэффициенты передачи по всем входам бесконечно растут (при бесконечно большом коэффициенте усиления ОУ), что приводит к появлению на выходе ОУ предельного значения выходного сигнала, и выходу ОУ из линейного режима. Такой режим работы ОУ обычно не допускается.

Непосредственная подача входного напряжения на инвертирующий вход ОУ также недопустима, поскольку приводит к большим коэффициентам передачи и к выходу ОУ из линейного режима. Это явление наблюдается при уменьшении входного сопротивления на одном из инвертирующих входов суммирующего блока до нуля. Выходное напряжение рассматриваемой схемы могло быть получено из основного уравнения для решающего усилителя **(см. Решающий усилитель)**, если сделать соответствующие подстановки значений проводимостей:

$Y_a(p) = Y_б(p) = Y_{осб}(p) = 0$; $Y_{оса}(p) = 1/R_{ос}$; $Y_{1k}(p) = 1/R_{1k}$; $Y_{2l}(p) = 1/R_{2l}$; $k = \overline{1,n}$; $l = \overline{1,m}$.

Частные решения блока суммирования.

Пусть неинвертирующий вход ОУ заземлен, т. е. его потенциал равен нулю (рис. 4.6). При этом $k_б = 0$; $U_{вых} = -\sum_{k=1}^{n}(R_{ос}/R_k)\cdot U_k$. Схема осуществляет алгебраическое суммирование входных напряжений с инвертированием знака результата.

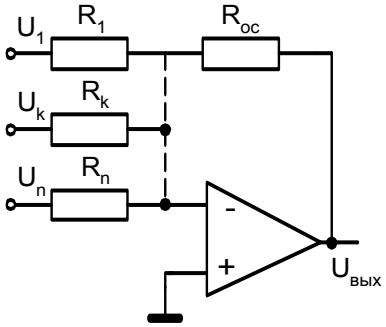


Рис. 4.6. Схема блока суммирования

Схема, представленная на рис. 4.7, осуществляет умножение входного напряжения на постоянный коэффициент $k = -R_{ос}/R_1$ с одновременным инвертированием полярности сигнала. Если $R_{ос} = R_1$, то коэффициент передачи схемы равен -1. Схема инвертирует полярность сигнала, не меняя его абсолютного значения, и называется инвертором или инверсным усилителем.

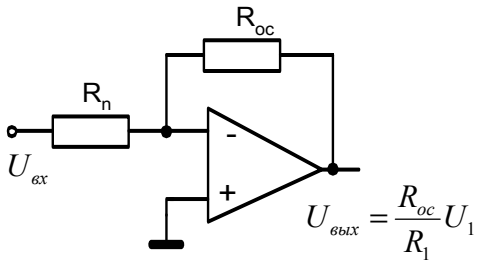


Рис. 4.7. б) Схема блока суммирования

На рис. 4.8 представлен блок суммирования, который не инвертирует знак результата. Формула для выходного напряжения получается из выражения (*), если $U_{11} = 0$, $R_{12} = R_{13} = ... = R_{1n} \to \infty$.

$$U_{вых} = \sum_{l=1}^{m} \frac{1 + R_{ос}/R_{11}}{1/R_б + \sum_{l=1}^{m} 1/R_{2l}} \cdot \frac{1}{R_{2l}} U_{2l}\ .$$

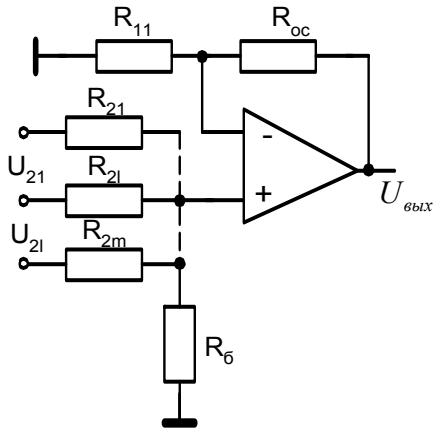


Рис. 4.8. Схема блока суммирования

Можно заметить, что коэффициент передачи каждого входного напряжения достаточно сложно выражается через сопротивления резисторов. Представляют интерес отдельные схемы, в которых используется только один входной сигнал.

На рис. 4.9, г, д представлены схемы, не инвертирующие полярность входного напряжения, но осуществляющие масштабное преобразование.

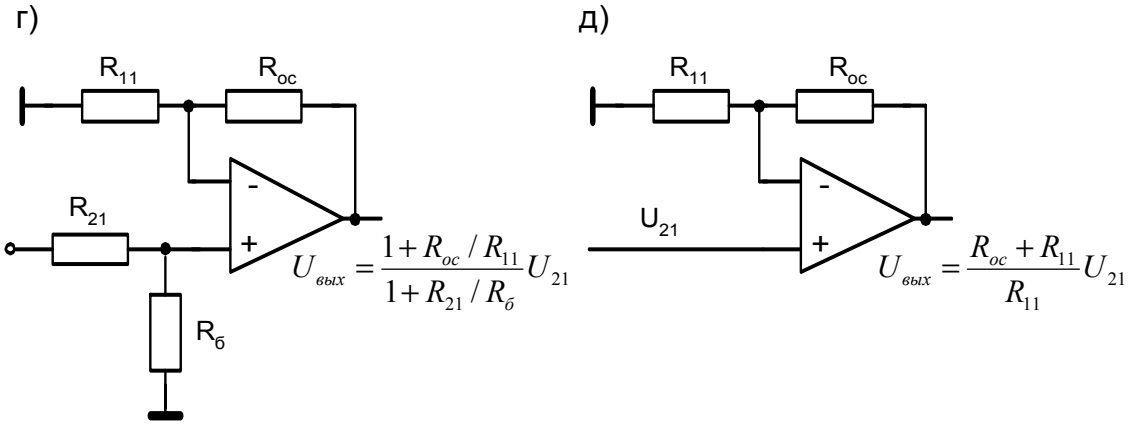


Рис. 4.9. Схема блока суммирования

Реализация интегрирования

В блоках интегрирования (интеграторах) используется способность емкости формировать напряжение на своих обкладках, пропорциональное интегралу тока через конденсатор по времени. На рис. 4.10 показана схема интегратора, который выполняет операцию интегрирования суммы взвешенных входных напряжений.

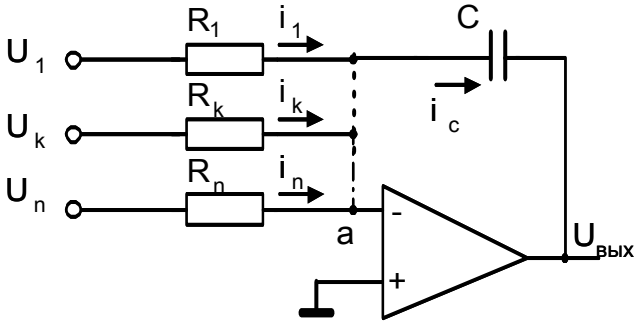


Рис. 4.10. Схема блока интегрирования

Для узла а справедливо уравнение $\sum_{k=1}^{n} i_k = i_c$.

Учитывая, что потенциал точки *a* близок к нулю $i_k = U_k / R_k$, $k = \overline{1,n}$, $i_c = -c(dU_{вых} / dt)$.

После подстановки значений токов в уравнения для узла а получим: $\sum_{k=1}^{n} U_k / R_k = -CdU_{вых} / dt$. Интегрирование полученного уравнения позволяет получить выходное напряжение схемы: $U_{вых} = -\sum_{k=1}^{n} \frac{1}{CR_k} \int_0^t U_k dt + U_{вых}(0)$.

Интегрирование только одного входного напряжения осуществляет интегратор, представленный на рис. 4.11.

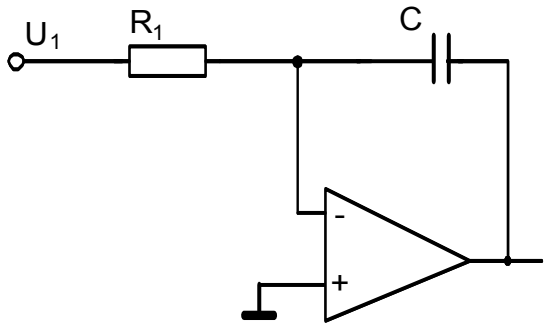


Рис. 4.11. Схема блока интегрирования

Выходное напряжение данного интегратора $U_{вых} = -\frac{1}{CR_1} \int_0^t U_k dt + U_{вых}(0)$.

Перед началом интегрирования необходимо задать начальные условия в виде напряжения $U_{вых}(0)$, до которого заряжается емкость интегратора. Для этого используются специальные цепи (рис. 4.12).

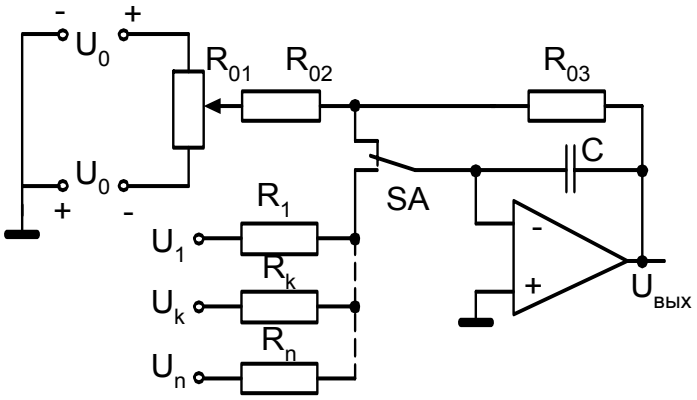


Рис. 4.12. Схема блока интегрирования

В режиме установки начальных условий переключатель *SA* подключает инвертирующий вход ОУ к делителю R_{02}, R_{03}. При этом схема превращается в масштабирующий усилитель, у которого входным сигналом является напряжение с движка потенциометра R_{01}. Источники напряжения, подключенные к потенциометру, позволяют задавать любое напряжение в диапазоне от $-U_0$ до $+U_0$. Это позволяет на выходе интегратора формировать напряжение начального условия в рабочем диапазоне выходных напряжений ОУ. В момент начала интегрирования переключатель подключает к входу ОУ входные резисторы и тем самым разрешает интегрирование суммы входных

напряжений. На практике в качестве переключателя используют контактные группы реле или аналоговые полупроводниковые ключи. А момент начала интегрирования определяется, например, моментом нажатия на кнопку “Пуск” АВМ.

Реализация дифференцирования

Блок дифференцирования (дифференциатор) формирует производную от входной величины по времени. Схема блока дифференцирования представлена на рис. 4.13.

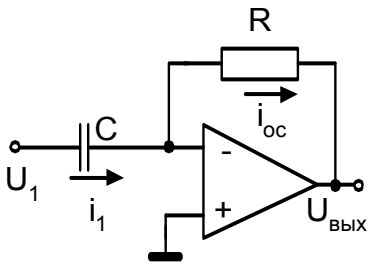


Рис. 4.13. Схема блока дифференцирования

Для инвертирующего входа ОУ справедливо: $i_1 = i_{ос}$; $U_- \approx 0$; $i_1 = C(dU_1 / dt)$; $i_{ос} = -U_{вых} / R$. Отсюда выходное напряжение $U_{вых} = -CR(dU_1 / dt)$.

Блок дифференцирования очень чувствителен к высокочастотным помехам даже при их небольших амплитудах, поэтому редко используется. Однако иногда без него нельзя обойтись, например при интегрировании некоторой функции по произвольной переменной *f{x).* В этом случае используется следующее равенство:

$$\int_0^x f(x)dx = \int_0^x f(x)\frac{dx}{dt}dt .$$

В интеграл входит производная от произвольной переменной х по времени. Определив эту производную с помощью дифференциатора, легко заменить интегрирование по произвольной переменной интегрированием по времени, которое просто реализуется в блоках интегрирования. Однако в процессе определения интеграла требуется перемножение двух зависящих от времени функций.

Реализация умножения

Блок умножения (БУ) предназначен для перемножения двух или большего числа аналоговых величин. Он находит широкое применение не только в АВМ, но и в системах автоматического регулирования, радиоаппаратуре, при создании перестраиваемых фильтров, в измерительных приборах. Широкому применению БУ способствует возможность их создания в интегральном исполнении.

Различают БУ с **прямым** и **косвенным** умножением. **Прямое умножение** реализуется с использованием датчиков Холла, выходное напряжение которых пропорционально произведению протекающего тока и магнитной индукции; с использованием электронно-лучевых трубок, отклонение луча в которых пропорционально произведению напряжения на. отклоняющих пластинах и тока через дополнительную отклоняющую катушку; с использованием дифференциальных усилителей управляемых сопротивлений; с использованием амплитудно-широтной импульсной модуляции.

Блок умножения, косвенного умножения реализуется с использованием параболических и логарифмических перемножителей.

В параболических перемножителях результат формируется как сумма или разность некоторых вспомогательных функций квадратов от исходных аналоговых величин. Реализуется в подобных БУ одна из следующих математических зависимостей:

$xy = (1/4)[(x+y)^2 - (x-y)^2]$; $xy = (1/2)[(x+y)^2 - x^2 - y^2]$;

$xy = (1/4)[(a+x+y)^2 - (a-x+y)^2]$; $xy = (1/2)[(x+y)^2 - (x-y)^2]$.

Возведение в квадрат выполняется с помощью ФП с кусочно-линейной аппроксимацией на диодах, которые при десяти точках излома позволяют получить погрешность не более ± 0,1% от полной шкалы измерений (рис. 4.14).

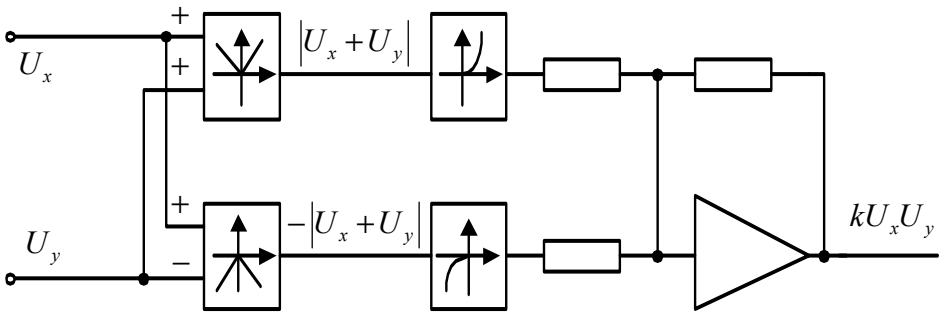


Рис. 4.14. Схема блока умножения

В логарифмических БУ используется логарифмическая зависимость напряжения на полупроводниковом переходе от тока через него и реализуется выражение: $\ln(U_x U_y) = \ln U_x + \ln U_y$.

Использование диода в цепи отрицательной обратной связи ОУ позволяет выполнить логарифмирование входного напряжения, а использование диода на входе ОУ - выполнить антилогарифмирование. Выполнив вычитание логарифмов, легко реализовать деление практически на тех же элементах: $\ln(U_x / U_y) = \ln U_x - \ln U_y$.

Обычный дифференциальный каскад может служить простейшим БУ. Разностный сигнал на коллекторах транзисторов равен произведению $\gamma U_x i_0$, где (i_0 - ток в эмиттерных цепях каскада, U_x - разностное входное напряжение, γ - коэффициент пропорциональности. Используя преобразователь “напряжение - ток” для формирования i_0, можно использовать каскад как перемножитель.

В настоящее время промышленность выпускает микросхемы аналоговых перемножителей 140МА1, 525ПС1, 525ПС2 и др., погрешность перемножения которых составляет ±(1—2)% от полной шкалы при полосе пропускания 1—2 мГц. Если требуется достижение высокой точности при умножении, то используется принцип амплитудно-широтной импульсной модуляции.

Реализация деления

Операция деления может быть реализована или с помощью схем логарифмирования, или с использованием блоков умножения в цепи отрицательной обратной связи ОУ (рис. 4.15).

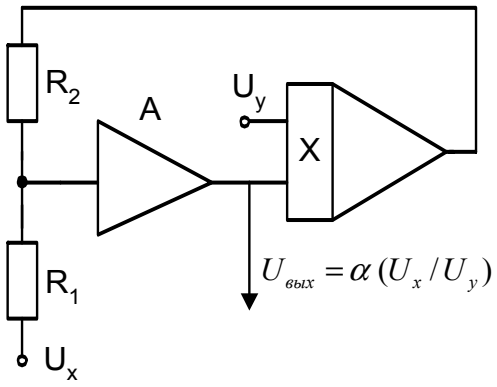


Рис. 4.15. Схема блока деления

Для данной схемы справедливо $U_x / R_1 = -U_{вых} U_y / R_2$. Отсюда выходное напряжение $U_{вых} = -(R_1 / R_2)(U_x / U_y) = \alpha U_x / U_y$. Таким образом, схема, представленная на рис. 4.15, выполняет операцию деления.

Тема 4.2. Аналоговые ЭВМ

Аналоговая ЭВМ (АВМ)

Отличительная особенность АВМ - аналоговая или непрерывная форма представления информации, переменных, использование аналоговых величин.

АВМ - вычислительная машина, производящая операции над аналоговыми величинами.

Аналоговая величина - непрерывная физическая величина, заменяющая искомую или заданную в решаемой задаче, связанная с ней масштабным соотношением.

В процессе вычисления или управления ЭВМ приходится решать разные задачи. Некоторые задачи предполагают получение числового результата, результатом решения других задач являются функции, т. е. математическая зависимость выходной переменной от входных.

В АВМ функция формируется как результат взаимодействия отдельных блоков, соединенных друг с другом в соответствии с формулой, выражающей значения функции.

Состав АВМ

Структурные АВМ представляют совокупность вычислительных и вспомогательных блоков. В зависимости от числа блоков различают малые (до 20 блоков), средние и большие (свыше 60 блоков) АВМ. Кроме вычислительных блоков АВМ обычно содержит источники питания, наборное поле, устройство управления, измерительную и регистрирующую аппаратуру.

Источники питания служат для преобразования переменного напряжения с промышленной частотой (обычно 50 Гц) в напряжения, обеспечивающие питание АВМ. К определенным постоянным напряжениям источника питания предъявляются повышенные требования по стабильности, так как они используются при формировании функций и возмущающих воздействий (±Uo). Всякие изменения этих напряжений воспринимаются АВМ как изменения машинных переменных.

Наборное поле позволяет осуществить коммутацию блоков АВМ в соответствии со схемой моделирования решения задачи. Коммутация производится коммутационными шнурами, каждый из которых представляет отрезок провода с однополюсными вилками на концах. Шнуры своими вилками вставляются в те гнезда наборного поля, которые соответствуют определенным схемой моделирования элементам и блокам.

Устройство управления обеспечивает совместную работу всех блоков и частей АВМ во времени. Режим работы АВМ определяется оператором с помощью органов управления. Различают четыре режима работы АВМ.

- **Режим подготовки**, в котором оператор производит установку требуемых коэффициентов передачи, начальных условий и возмущающих воздействий.
- **Режим решения**, в течение которого осуществляется решение задачи. Решение может производиться однократно или многократно, путем автоматического повторения решения. Переход в режим решения задачи осуществляется с пульта управления.. Процесс решения обычно начинается с момента нажатия оператором на кнопку «Пуск».
- **Режим останова**, в течение которого процесс решения приостанавливается и оператор спокойно может измерить очередные промежуточные результаты. Останов осуществляется с помощью специальной кнопки. Продолжить решения можно повторным нажатием кнопки «Пуск».
- **Режим возврата АВМ в исходное состояние** осуществляется или автоматически при многократном повторении решения, или нажатием на специальную кнопку. После возврата процесс решения может быть повторен.

Измерительная и регистрирующая аппаратура предназначена для регистрации результатов решения. В качестве измерительной и регистрирующей аппаратуры используются стрелочные и цифровые вольтметры, многолучевые осциллоскопы с длительным послесвечением экрана для одновременного наблюдения результатов решения в нескольких точках схемы, шлейфовые осциллографы для графической записи меняющихся во времени машинных переменных на светочувствительную бумагу.

Структурные АВМ представляют собой совокупность жестко не связанных друг с другом вычислительных блоков, что позволяет более эффективно использовать оборудование, но требует кроме обычных подготовительных работ коммутации блоков в соответствии с исходным уравнением. Структурные АВМ нашли наибольшее применение.

Чтобы было возможным осуществить должным образом коммутацию вычислительных блоков АВМ, составляется по исходному уравнению структурная схема или схема моделирования решения задачи - графическое изображение соединений между решающими усилителями, элементами, преобразователями, входящими в схему и обеспечивающими подготовку и решение задачи моделирования.

Существующие условные графические обозначения для блоков АВМ представлены на рис. 4.16.

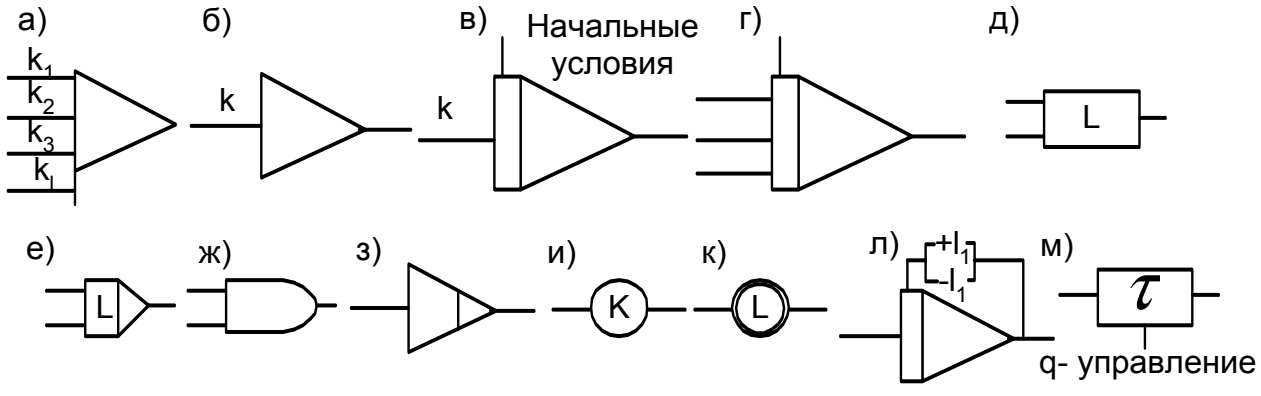


Рис. 4.16. Условные графические обозначения блоков АВМ.

Где, а) Суммирующий усилитель, б) масштабирующий усилитель, в) интегрирующий усилитель, г) интегросуммирующий усилитель, д) нелинейный блок операции L, е) нелинейный блок с собственным выходным устройством, ж) нелинейный блок без собственного выходного усилителя, з) усилитель дифференцирующий, и) блок постоянного коэффициента, зависящего от времени, к) L – идентификатор, л) схема ограничения, м) блок переменного запаздывания.

Перестраиваемый функциональный преобразователь

Блоки функционального преобразования или функциональные преобразователи (ФП) предназначены для моделирования нелинейных зависимостей функций от одной или большего числа переменных. Входными и выходными переменными ФП являются аналоговые величины, представленные электрическими напряжениями, т. е. при входном напряжении $U_{вх}$ на выходе ФП формируется $U_{вых} = F(U_{вх})$.

Функциональные преобразователи делят на **универсальные** и **специализированные**.

Универсальные ФП позволяют воспроизводить широкий класс функций с помощью одной схемы путем ее перестройки. При создании универсальных ФП используется аппроксимация функции с помощью полиномов и функциональных рядов. Наибольшее применение благодаря простоте схемной реализации нашла кусочно-линейная аппроксимация функции.

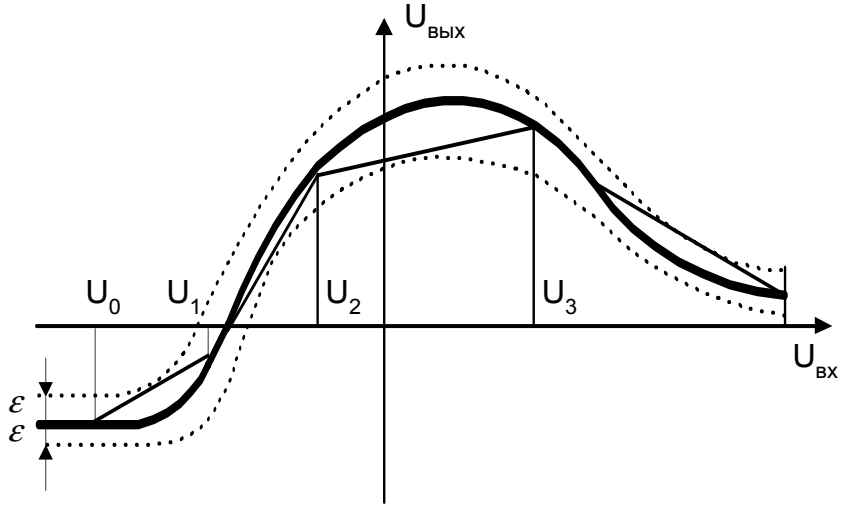


Рис. 4.17. Кусочно-линейная аппроксимация функции

При этом методе (рис. 4.17) диапазон входных напряжений разбивается на интервалы $\Delta l = U_l - U_{l-1}$, длина которых зависит от заданного значения максимальной ошибки аппроксимации ε и известного максимального значения второй производной исходной функции

$$\left|\frac{d^2U_{вых}}{dU^2_{вх}}\right|_{\max}; \Delta l = \sqrt{\frac{8\varepsilon}{\left|\frac{d^2U_{вых}}{dU^2_{вх}}\right|_{\max}}}.$$

Деление на интервалы возможно теоретически при заданном математическом выражении $U_{вых} = F(U_{вх})$ или графически. На каждом интервале исходная функция аппроксимируется полиномом 1-го порядка:

$F(U_{вх}) \approx F(U_{l-1}) + a_0(U_{вх} - U_{l-1})$, где $U_{вх} \in [U_{l-1}; U_l]$,
$a_0 = [F(U_l) - F(U_{l-1})]/(U_l - U_{l-1})$.

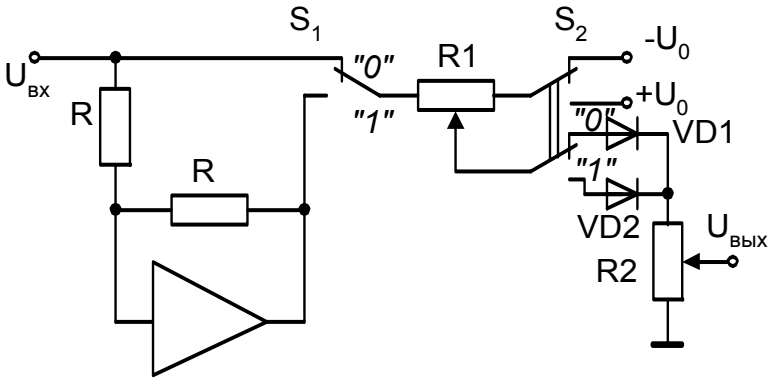


Рис. 4.18. Схема диодного функционального преобразователя

Широкое применение нашли **диодные ФП**, состоящие из нескольких одинаковых диодных нелинейных элементов (ДНЭ) (рис. 4.18), позволяющих при определенном положении переключателей S_1 и S_2 формировать аппроксимирующую функцию на определенном интервале. Выходы всех ДНЭ подаются на входы суммирующего решающего усилителя, выход которого является выходом ФП. Вид воспроизводимых ДНЭ зависимостей представлен на рис. 4.19.Положение точки перегиба U_{l-1} определяется положением движка потенциометра R_1 и полярностью подключенного к R_1 с помощью S_2 опорного напряжения U_0. Угол наклона α_i зависит от положения движка потенциометра R_2; вид воспроизводимой функции - от положения переключателей S_1 и S_2.

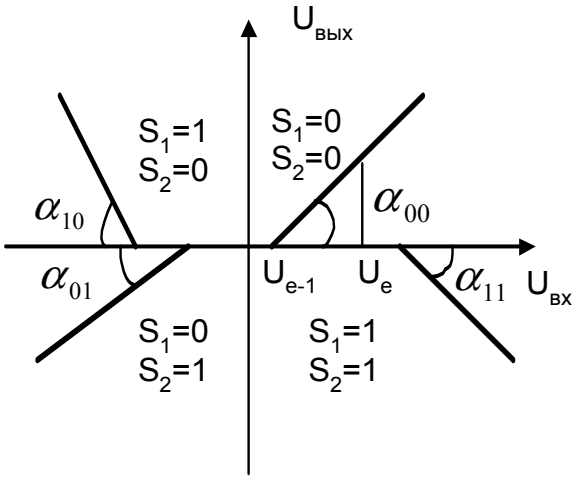


Рис. 4.19. Зависимость, воспроизводимая ДНЭ

Рассмотрим работу ДНЭ для представленного на рис. 4.18 положения, т. е. S_1=0, S_2=0. При $U_{вх} < U_l$ на движке потенциометра R_1 формируется отрицательное напряжение, смещающее диод VD1 в обратном направлении $U_{вых} = 0$ В. При $U_{вх} \geq U_l$ диод VD1 окажется прямосмещен и рост входного напряжения вызывает линейный рост выходного напряжения. В процессе настройки ФП приходится учитывать совместную работу всех входящих в его состав ДНЭ, т. е. наклон α_i конкретного ДНЭ зависит не только от угла наклона аппроксимирующей линии на данном участке, но и от настройки других ДНЭ. Специализированные ФП ориентируются на воспроизведение одной функции и используются в случае, когда реализация этой функции с помощью универсального ФП затруднительна или нецелесообразна.

Схема **специализированного ФП** для воспроизведения параболической зависимости представлена на рис. 4.20.

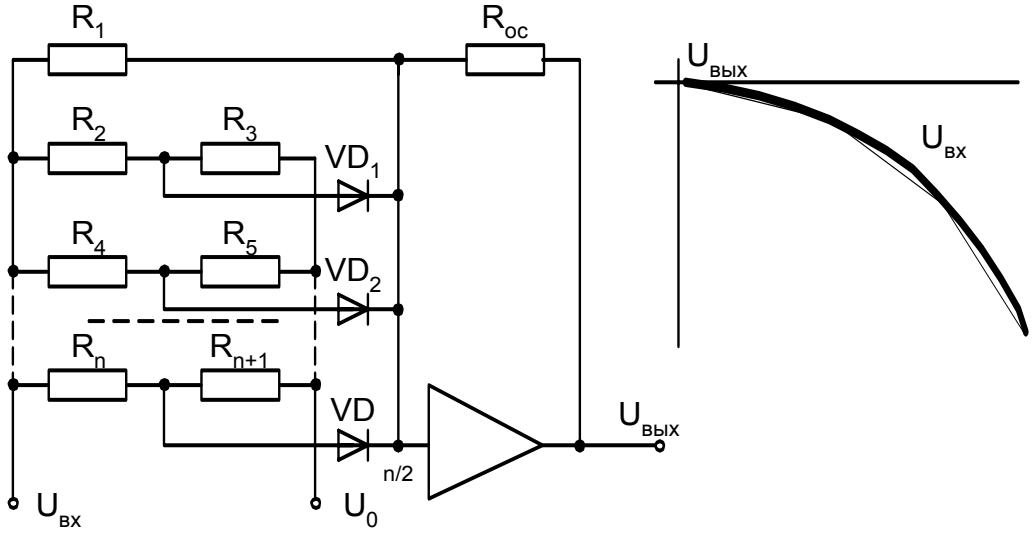


Рис. 4.20. Схема специализированного ФП для воспроизведения параболической зависимости

Особый интерес представляют ФП, воспроизводящие функции времени. Для них справедливо: $U_{вых} = F_1(t)$. Реализация различных функций времени может быть осуществлена или специально созданным для этой цели устройством, или интегрированием вспомогательных дифференциальных уравнений. Например, для формирования функции $U_{вых} = \sin \omega t_M$ достаточно решить дифференциальное уравнение 2-го порядка: $\frac{d^2 U_{вых}}{dt_M^2} = -\omega^2 U_{вых}$.

Программирование АВМ

Широкое распространение получили АВМ для решения обыкновенных дифференциальных уравнений (ОДУ) (дифференциальные анализаторы). Различают АВМ **матричные** и **структурные**.

Матричные АВМ строятся в соответствии с системой дифференциальных уравнений 1-го порядка вида

$$\frac{dx_j}{dt} + \sum_{k=1}^{n} a_{kj} x_j = b_j f_j(t), \quad j = 1,2,\dots,n.$$

Любое дифференциальное уравнение или система приводятся к данному виду и решение сводится лишь к установке постоянных коэффициентов a_{kj}, b_j, вводу возмущающих воздействий $f_j(t)$, начальных условий и решению. Вычислительные блоки АВМ матричного типа жестко соединены друг с другом.

Структурные АВМ представляют собой совокупность жестко не связанных друг с другом вычислительных блоков, что позволяет более эффективно использовать оборудование, но требует кроме обычных подготовительных работ коммутации блоков в соответствии с исходным уравнением. Структурные АВМ нашли наибольшее применение.

Чтобы было возможным осуществить должным образом коммутацию вычислительных блоков АВМ, составляется по исходному уравнению структурная схема или схема моделирования решения задачи - графическое изображение соединений между

решающими усилителями, элементами, преобразователями, входящими в схему и обеспечивающими подготовку и решение задачи моделирования.

Существующие условные графические обозначения для блоков АВМ представлены на рис. 4.16.

Основные способы построения структурных схем

Для решения уравнения на структурной АВМ необходимо привести исходное уравнение к виду, удобному для реализации на АВМ. При этом ставится задача определить такой способ коммутации вычислительных блоков, который позволяет создать на базе АВМ математическую модель исходной системы. В основу построения структурной схемы АВМ могут быть положены два возможных способа:

- **Способ повышения порядка производной** предполагает выделение искомой переменной и использование блоков дифференцирования при создании модели. Этот способ находит ограниченное применение, поскольку дифференцирующие блоки очень чувствительны к помехам.
- **Способ понижения порядка производной** предполагает выделение старшей производной и применение интеграторов при создании модели. Этот способ нашел широкое применение.

Начальные условия

Чтобы обеспечить однозначность решения дифференциальных уравнений, необходимо принимать во внимание начальные условия. обычно в качестве начальных условий используют значения переменных и их производных в начальный момент времени.

Например, для системы уравнений:

$$a_0 \frac{d^2 x_1}{dt^2} + a_1 \frac{dx_1}{dt} + a_2 x_1 + a_3 x_2 = f_1(t)$$

$$b_0 \frac{dx_2}{dt} + b_1 x_2 = f_2(t)$$

могут задаваться значениями функций x_1 и x_2 при t_0; $x_1(0)$; $x_2(0)$ и значением первой производной функции x_1 в тот же момент времени: $x_1'(0)$. Чтобы определить начальные условия для машинного уравнения, необходимо использовать следующие значения масштабов:

$$M_{X1} \geq \frac{|x_1|_{\max}}{|U_{X1}|_{\max}}, \qquad M_{X2} \geq \frac{|x_2|_{\max}}{|U_{X2}|_{\max}}, \qquad M_{X'1} \geq \frac{|x_1'|_{\max}}{|U_{X'1}|_{\max}}, \qquad M_t \geq \frac{t_{\max}}{t_{M\max}},$$

$$M_{f1} \geq \frac{|f_1|_{\max}}{|U_{f1}|_{\max}}, \; M_{f2} \geq \frac{|f_2|_{\max}}{|U_{f2}|_{\max}}, \; M_{f'1} \geq \frac{|f_1'|_{\max}}{|U_{f'1}|_{\max}}, \; t_{M\max} = 100\ c.$$

Используя эти значения масштабов, получим:

$$x_1(0) = M_{x1} U_{x1}(0), \qquad x_2(0) = M_{x2} U_{x2}(0),$$

$x_1'(0) = \left.\frac{dx_1}{dt}\right|_{t=0} = M_{x'1} \left.\frac{dU_{x1}}{dt_M}\right|_{t_M=0} = M_{x'1} U'_{x1}(0)$. Из этих условий выразим значения начальных условий для машинного уравнения:

$U_{x1}(0) = x_1(0)/M_{x1}$, $U_{x2}(0) = x_2(0)/M_{x2}$, $U'_{x1}(0) = x_1'(0)/M_{x'1}$.

Эти значения напряжений задаются в качестве начальных условий на интегрирующие блоки с учетом законов формируемых на выходах блоков значений машинных переменных.

Процесс решения задачи на АВМ

После осуществления программирования АВМ, которое заканчивается привязкой схемы моделирования решения задачи к конкретным элементам, операционным усилителям и потенциометрам, следует процесс решения на АВМ. В процессе решения можно выделить несколько основных этапов:

- Этап 1. Набор задачи на наборном поле АВМ, настройка и установка коэффициентов передачи отдельных блоков. На этом этапе компенсируют при необходимости смещение нуля ОУ, настраивают функциональный преобразователь, устанавливают значения постоянных коэффициентов, соединяют вычислительные блоки в соответствии со схемой моделирования, вводят внешние возмущения, задают начальные условия на интегрирующие блоки.
- Этап 2. Пробное решение задачи, позволяющее уточнить масштабы переменных, контролируя напряжение на выходах всех блоков.
- Этап 3. Проверка правильности и точности решения задачи.
- Этап 4. Снятие результатов решения и их обработка. Используя масштабы машинных переменных, легко преобразовать результаты решения к исходному уравнению в виде соответствующих изменений физических величин.

Гибридные ЭВМ

Существуют задачи, для решения которых применение АВМ или ЦВМ отдельно малоэффективно. К таким задачам можно отнести, например, задачу моделирования в реальном масштабе времени описываемых сложными дифференциальными уравнениями динамических систем. Если в решении задачи можно выделить низкочастотные и высокочастотные процессы, то применение только АВМ может привести к большой погрешности времени, а ЦВМ может не обеспечить необходимого быстродействия при обработке высокочастотных процессов.

Для подобных задач эффективен аналого-цифровой вычислительный комплекс (АЦВК) (гибридные ЭВМ), представляющий в простейшем случае совокупность АВМ, ЦВМ, преобразователей форм представления информации (аналого-цифровых и цифроаналоговых преобразователей) и схем управления и синхронизации. АЦВК позволяет объединить достоинства АВМ и ЦВМ. Если предложенную выше задачу решать с помощью АЦВК, то аналоговая часть комплекса может успешно справиться с высокочастотными процессами, а цифровая часть осуществит обработку в реальном времени низкочастотные процессы.

Применение ЦВМ совместно с АВМ позволяет решать алгоритмическими способами ряд проблем, присущих обычному использованию АВМ. Во-первых, ЦВМ упрощает моделирование логических функций. Во-вторых, ЦВМ может выполнить все подготовительные операции, связанные с программированием АВМ, позволяет значительно упростить процесс подготовки АВМ к решению задачи.

Применение ЦВМ позволяет успешно решать задачу автоматической установки постоянных коэффициентов и начальных условий, автоматизировать процесс оптимального выбора масштабов машинных переменных, осуществлять автоматически (при наличии соответствующих аналоговых коммутаторов) по заданной схеме моделирования (возможно, созданной также с использованием ЦВМ) процесс коммутации вычислительных блоков, упрощать анализ результатов решения, оптимизировать совместное использованние АВМ и ЦВМ при решении задачи. Таким образом, ЦВМ в составе АЦВК может использоваться как на этапе подготовки к решению задачи, так и на этапе решения задачи.

К ЦВМ, используемой в составе АЦВК, предъявляются определенные требования:

19. ЦВМ должна быть достаточно быстродействующей (до $5 \cdot 10^5$ операций/с), чтобы обеспечить работу комплекса в реальном масштабе времени;
20. ЦВМ должна обладать сравнительно большим объемом оперативной памяти;
21. в ЦВМ должна быть предусмотрена возможность расширения состава команд, чтобы управлять АВМ.

Этим требованиям во многих случаях удовлетворяют малые управляющие ЦВМ, имеющие невысокую разрядность (16 - 24 двоичных разряда).

Раздел 5. Архитектуры для языков высокого уровня

Тема 5.1. Явление семантического разрыва

Методы кодирования данных.

Для того чтобы ввести понятие семантического разрыва необходимо вспомнить основные элементы. ЭВМ (вычислительное средство) – это совокупность некоторых элементов, которые составляют единое целое, а архитектура – это концепция этой взаимосвязи. Организация ЭВМ – это конкретные примеры неких архитектурных принципов соединения (организация ЭВМ универсального назначения и т.д.). Что касается ВС, то под архитектурой мы понимаем общую логическую организацию такого рода ВС. Сюда входит несколько таких частей. 1-я часть – это методы кодирования данных. Понятно, что если есть взаимодействующие части, то они должны взаимодействовать каким-то образом. Методы кодирования данных и принципы взаимодействия, как 2-ую часть архитектуры, мы определили в рамках системных интерфейсов. Системный интерфейс занимается 1-ми двумя частями архитектуры. Еще в состав архитектуры входит логическая организация (пример логической организации – это принцип иерархической организации памяти, она не организуется на уровне системного интерфейса).

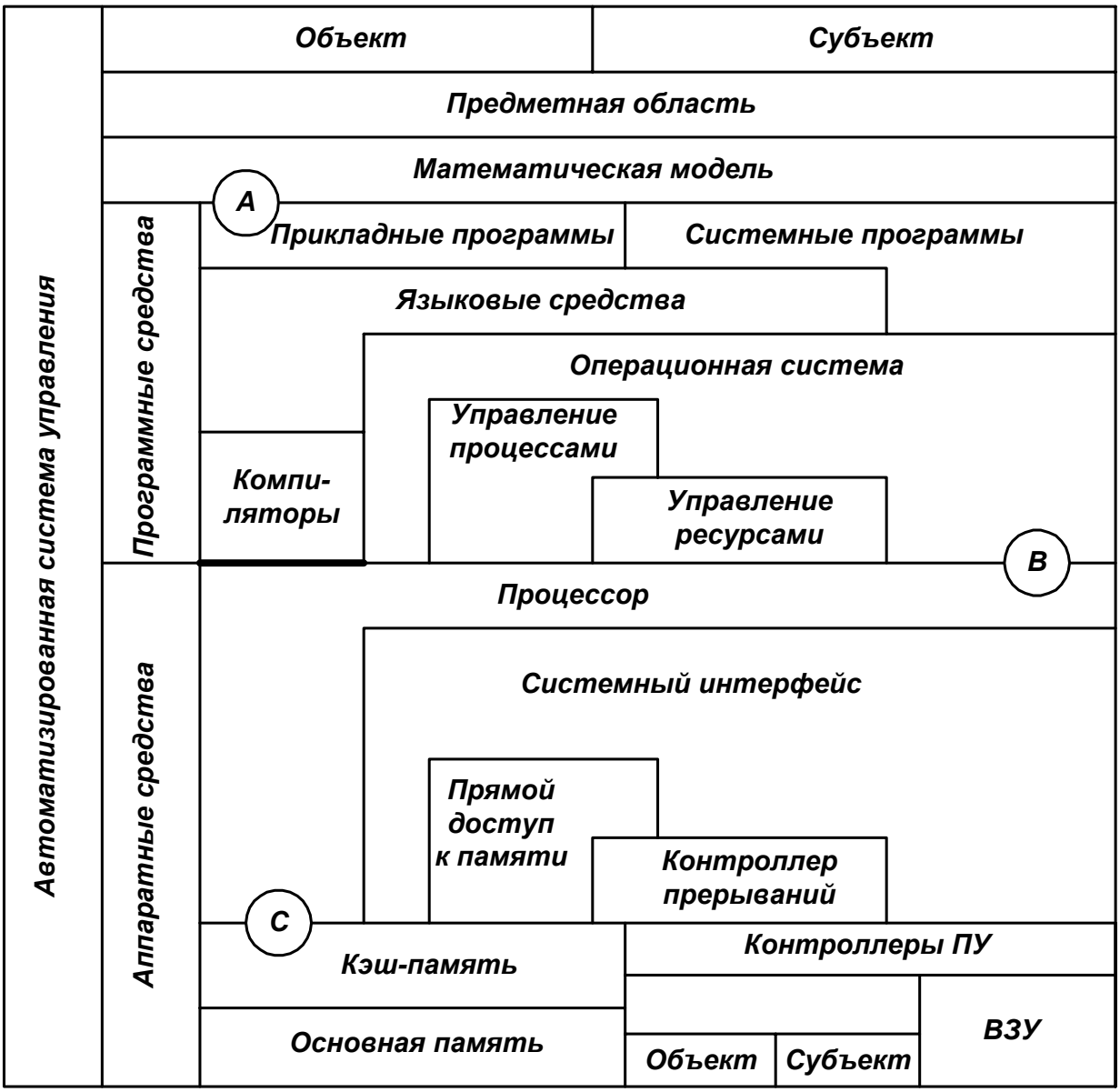


Рис. 5.1. Уточнение понятия архитектуры

Уточним понятие архитектуры, связанное с систематизацией архитектурных элементов.

1-ый уровень – это автоматизированная система. Когда мы рассматриваем автоматизированную систему, мы предполагаем, что есть некоторый объект изучения и есть некоторая техническая система, которая в совокупности с обслуживающим персоналом является автоматизированной системой, предназначенной для решения какой-то конкретной задачи управления или автоматизации. Таким образом, автоматизированная система состоит из технических средств, из технического персонала, а составная 3-я часть – это представление обслуживающего персонала об этом объекте управления. Вы не можете участвовать в автоматизированной системе, ничего не зная об объекте управления. Разобьем автоматизированную систему на 3 части.

1-я часть, в свою очередь, делится на 2 части: объект и субъект. Автоматизации подлежат не все аспекты управления, а только те, которые возможно формализовать, т.е. представить в виде мат. модели. Но, к сожалению, не все можно формализовать, поэтому там и присутствует человек. Т.е. нет математических моделей, которые точно описали бы окружающие нас явления.

В представлении математической модели участвует как объект, так и субъект. Формальная система является математической моделью.

Совокупность ВС, математической модели, объекта и субъекта является автоматизированной системой.

Любая ВС состоит из 2-х частей: программные и аппаратные средства. В качестве аппаратных средств используются ЭВМ и комплексы.

Все программные средства разделяются на 2 класса: прикладное программное обеспечение и системное программное обеспечение.

Семантическим разрывом традиционно называется феномен, определяемый как мера различия принципов, лежащих в основе языковых средств вычислительных систем и тех принципов, которые положены в основу аппаратных средств. Что это означает? Когда мы используем языки высокого уровня, у нас есть абстракции: типы данных (встроенные), они бывают разными (знак, целое, массив и т.д.), для каждого типа данных определяются операции, которые над ними применимы. Но в памяти эти объекты представляются одинаково, как последовательность байт и процессор работает с некоторым набором операций, которые у него есть. Т.е. имеется смысловой или семантический разрыв между теми средствами, которые используются для представления каких-то решаемых задач и теми средствами, что мы вынуждены использовать при реализации этих представлений. Это по декларативной части.

По процедурной части: есть такие операции как DO, WHILE, FOR, IF, но процессор эти операции не выполняет. Т.е. имеется также семантический разрыв по процедурной части. То, что мы пишем на языке программирования, это совсем не то, что мы должны исполнять на аппаратном уровне. Имеется семантический разрыв между языковыми средствами и аппаратными средствами (традиционная трактовка семантического разрыва).

Объекты и соответствующие им операции, реализуемые аппаратными средствами, отличаются от объектов и операций, используемых в языках высокого уровня.

Слой разрыва **А** называется архитектурой вычислительных систем.

Мы вынуждены преобразовывать нашу модель в те объекты и операции над ними, которые поддерживаются вычислительной системой.

Существует семантический разрыв – между прикладными программными средствами и языковыми средствами. Попытка решить проблему этого разрыва привела к созданию более универсальных языков (объектно-ориентированные методологии программирования). Тот же самый разрыв существует между системным программным обеспечением и языковыми средствами.

Третий семантический разрыв – между системным программным обеспечением и операционной системой. Если бы мы могли некие математические модели, связанные с системными вопросами сразу реализовывать в рамках операционной системы, не было бы системного программного обеспечения.

Существует семантический разрыв **В** между аппаратными средствами и тем, на что они опираются. Слой **В** называется архитектурой процессора.

Аппаратные средства можно условно разделить на 2 части. Первая часть называется процессором (т.е. представления, которые мы имеем о процессоре или те объекты и типы данных, которые поддерживает процессор). Вторая часть называется системным интерфейсом, который имеет 2 составные части: канал прямого доступа к памяти и контроллер приоритетных прерываний. При работе канала возникает прерывание, что канал окончил работу. Системный интерфейс относим к процессору, потому что процессор является организующим звеном для всей вычислительной системы и его интерфейс очень жестко связан с системным интерфейсом.

ЭВМ разделена на кэш-память, ОП, внешнее запоминающее устройство и устройство ввода/вывода, но они ничего не могут сделать без контроллера периферийных устройств. УВВ необходимо для управления объекта и взаимодействием с субъектом.

Слой разрыва **C** – архитектура ЭВМ (комплекса).

Последствия семантических разрывов

1. Высокая стоимость программных средств, т.к. необходимо держать постановщика задачи, который преобразует то, чем нам требуется управлять в математическую модель. Также требуется еще ряд специалистов, которые эту математическую модель (абстракцию) смогут интерпретировать в терминах и операциях конкретной ВС, требуется программист.
2. Низкая эффективность и надежность программных средств. Глобальные проблемы мы реализуем в рамках побитовых операций, т.е. получается большой объем кода, а при большом объеме кода и большом количестве специалистов получается много ошибок.
3. Большой объем исполняемого кода.
4. Сложность компиляторов и ОС.

Стимулом для развития архитектур процессора послужило наличие семантического разрыва.

Основные виды семантического разрыва

1. Семантический разрыв между математической моделью и языковыми средствами (устраняется прикладным программным обеспечением с помощью объектно-ориентированного программирования).
2. Семантический разрыв между языковыми средствами и архитектурой процессора (архитектуры для языков высокого уровня). Чтобы решить проблему этого семантического разрыва мы должны сделать процессор, который «ближе» к языкам программирования, т.е. поддерживает различные типы данных и выполняет более сложные операции.
3. Семантический разрыв между ОС и архитектурой ЭВМ (сокращается путем аппаратной поддержки ОС процессором).
4. Семантический разрыв между архитектурой процессора и ПУ (устраняется контроллерами ПУ).
5. Семантический разрыв между предметной областью и архитектурой вычислительной системы.

Цели устранения семантического разрыва

1. Сокращение длительности проектирования программных средств.
2. Повышение надежности программы.
3. Уменьшение сложности компилятора.

Сократить один семантический разрыв можно только за счет другого. Архитектуры процессоров полностью не устраняют семантический разрыв, просто мы приближаем процессор к тем объектам и операциям, которые используются в языках высокого уровня. Сейчас 90% затрат идет на разработку программного обеспечения, а не на разработку ЭВМ.

Методы сокращения семантического разрыва

1. Увеличение мощности системы команд. Команды манипулирования со структурами, наборами (множествами), команды типа DO, WHILE, FOR.
2. Использование новых принципов организации памяти (память с семантической структурой, теговая память).

3. Переложение ряда функций ОС на аппаратные средства (управление процессами, управление и защита памяти, управление ресурсами). Пример: имеются специальные микросхемы процессоров или сопроцессоров, которые управляют ресурсами, процессами, и предназначены для реализации управления операционной системой.

Тема 5.2. Теговая архитектура

В различных архитектурах типы данных определяются программой, иначе говоря, типы данных определяются кодом операции, что приводит к необходимости использовать разные команды для обработки различных типов данных. Каждый тип данных предусматривает наличие специального подмножества команд.

Принцип теговой архитектуры заключается в самоопределение данных, то есть каждая ячейка памяти имеет специальное дополнительное поле (тег), которое служит для описания атрибутов его содержимого. Это позволяет получить инвариантность команд к типам используемых данных. Сокращается число и длина команд. Определение типа данных откладывается на момент выборки этих данных, а не на момент выборки команды. Вполне может быть, что возникают исключительные ситуации, связанные с несоответствием типов командам, как правило, это является ошибкой программирования.

Расширенный принцип тегирования.

Заключается в задании не только типа хранимых данных, но и длины операнда, занятости ячейки и т.д. Используются дополнительные флаги:

1. Бит занятости. Определяет, свободна ячейка или занята.
2. Бит захвата. Необходимость выполнения прерывания при доступе к ячейке.
3. Биты защиты – определяет, какие обращения допустимы к этой «порции» данных (только чтение, только запись, чтение/запись).
4. Длина данных.

Преимущества механизма тегирования

1. Компактная система команд.
2. Строгий контроль типов.
3. Более простые компиляторы. Меньший объем кода.
4. Уменьшение интенсивности пересылки основная память – процессор, т.к. уменьшается код программы, загружающий интерфейс на 50%, а увеличивается объем читаемых данных, загружающий интерфейс на 30%.

Недостатки теговой организации:

1. Строгий контроль типов снижает выразительные способности языка.
2. Уменьшается быстродействие процессора за счет откладывания привязки атрибутов данных на этап выполнения команды. В момент дешифрации команды сразу определяются типы данных, здесь же в момент дешифрации команды типы данных не определяются и поэтому то время, которое связано с подгрузкой операнда могло быть использовано для определения типов данных, что и происходит. Т.е. сокращено время дешифрации команды, но доступ к памяти медленный и снижается быстродействие процессора.
3. Дополнительные расходы основной памяти в области данных для хранения полей тега.

Расширенная теговая организация.

Основана на двух принципах:

1. Самоопределение данных. Заключается в использовании тега не только для простых типов данных, но и составных (массивы, структуры, объединения).

2. Механизм дескрипторов. Дескриптор – это слово в основной памяти, в котором содержится информация об атрибутах данных, а так же об их местоположении в основной памяти. Здесь хранится не само данное, а указатель на то место в основной памяти, где это данное располагается. В команде кодируются только лишь указатели или адресуются только лишь дескрипторы, а потом уже идет обращение к этим данным. В этом случае необходимо под таблицу дескрипторов иметь быстродействующую память внутри процессора. Для доступа к данным необходимо дополнительное обращение к памяти.

   Пример тэговой архитектуры: ЭВМ BURROUGHT 6700

   Реализует механизм дескрипторов, имеющих длину 51 бит.

   X[3] P[1] C[1] I[1] S[1] R[1] T[2] D[1] LNT[20] ADR[20]

   X – это тэг (тип данных)

   P – основная или внешняя память (это фактически бит наличия страницы в памяти, если внешняя память происходит прерывание)

   C – бит копии дескриптора

   I – бит распространения (если I=0, то L – число элементов)

   S – бит непрерывности (говорит о том, что данные занимают непрерывную область или эта область сегментирована)

   F – защита от записи

   T – характер (т.е. данные фиксированного размера или переменного, т.е. строка или слово)

   D – точность данных (одинарная или двойная)

   A – местоположение данных в основной памяти

   Для описания многомерных массивов дескрипторы объединяются в древовидные структуры, т.е. поддерживается тип данных «дескриптор».

Тема 5.3. Объектно-ориентированная архитектура

Объект – это совокупность взаимосвязанных элементов, включающих в себя непосредственно данные (код), а также сведения о состоянии его взаимодействия с другими объектами.

Свойства объекта

1. Элементы объекта создаются и уничтожаются одновременно.
2. Объект является единым целым, и внутренняя структура объекта остается невидимой.
3. Для выполнения операций над объектами имеются специальные команды.
4. Объекты не находящиеся во взаимосвязи по модели данных невидимы друг другу.

Потенциальная адресация

Указывает возможность обращения одного объекта к другому. Каждый объект потенциально адресуем любым другим объектом, но такая адресация не устанавливается автоматически и не может быть создана односторонне (используются специальные процедуры связывания, на уровне команд процессора и функций объектно-ориентированных операционных систем). Иными словами, потенциальная адресация означает, что каждый объект может использовать адрес любого другого объекта, но эта связь устанавливается не автоматически.

Имена объектов отражают их логические, а не физические адреса. При этом на аппаратном уровне формируется таблица соответствия логических и физических адресов.

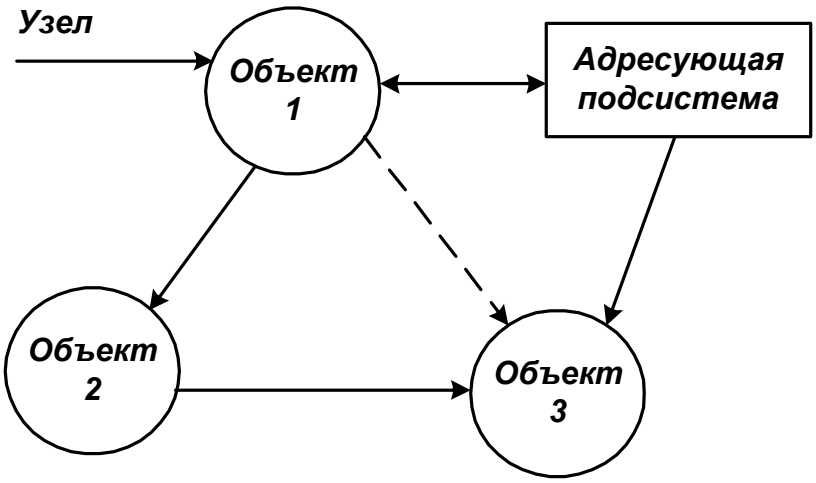


Рис. 5.2. Механизм адресации объектов

Объект 1 порождает в результате каких-то действий объект 2, причем знает о его создании и может к нему обращаться только объект 1, который его создал. Связь между объектом 1 и объектом 3 устанавливается с помощью адресующей подсистемы.

Операционная система поддерживает механизм адресации объектов. Примером служит архитектура SWARD, которая является как объектно-ориентированной, так и тэговой архитектурой. Аппаратно поддерживаются 5 типов объектов:

1. Модули (домен);
2. Процесс (может состоять из многих портов, модулей, данных) – единица работы вычислительной системы;
3. Запись активации (требуется наличие стеков для методов объектов; содержит ресурсы, необходимые для запуска процесса, метода);
4. Порты как пункты взаимодействия и связи объектов;
5. Память данных.

Модуль – это код; процесс – это совокупность объектов. Процесс может содержать несколько подпроцессов. Запись активации обеспечивает повторную входимость какой либо метод. Каждая запись активации – это метод этого объекта. Запись активации содержит в системной области указатель на модуль, который должен быть запущен, когда этот метод активизируется. Память - это выделяемая или высвобождаемая память под адресное пространство ввода/вывода.

Рассмотрим архитектуру Intel APX-432.

Объектно-ориентированное представление в своей основе содержит некую модель и основой для построения этой модели является дерево. На аппаратном уровне поддерживается модель ориентированного графа для структур данных.

В объектно-ориентированном подходе более узко рассматривается базовый объект, который может иметь несколько наследуемых объектов. Это случай с одиночным наследованием: когда один базовый объект может иметь один или несколько наследуемых, причем базовый объект описан некой совокупностью данных всех составляющих его объектов. Наследуемый объект как бы содержит одновременно данные, которые определяет базовый объект и свою часть, т.е. в этом случае наследуемый объект добавляет к структуре данных, описываемую базовым объектом некие свои данные.

При множественном наследовании у нас получается модель в виде ориентированного графа. Необходимо было представить аппаратную поддержку этой модели.

Решение было достигнуто путем представления структур данных в виде наборов независимых адресных пространств, которые назвали объектами.

Соответствие между моделью данных в виде ориентированного графа и моделью данных во время исполнения в виде набора объектов представляется так: узел ориентированного графа есть объект, дуга есть ссылка на другой объект.

Структура адресного пространства объекта

1. Ссылка на объект представляется в виде дескриптора.
2. Сегмент – это независимое адресное пространство. Понятие сегмента вводится из-за того, что у нас есть объект, который состоит из нескольких сегментов: 1-й сегмент – одно адресное пространство, 2-й сегмент – другое адресное пространство, 3-й сегмент содержит ссылки плюс сами данные. Т.е. мы должны ввести понятие сегмента, которое как-то отличается от понятия объекта.
3. Сегменты независимы друг от друга и к каждому сегменту обращение осуществляется через дескрипторную таблицу (таблицу отображения), что обеспечивает полную перемещаемость сегментов в линейном адресном пространстве. Имеется в виду, что есть дескриптор, который является ссылкой на объект. Это дескриптор через таблицу отображения позволяет получить доступ к сегментам, т.е. дескриптор является точкой входа в таблицу отображения, которая в свою очередь содержит адрес объекта в линейной памяти (или сегменте).
4. Сегменты могут быть различной (переменной) длины, размеры устанавливаются при создании объектов или сегментов и могут изменяться в процессе выполнения. В последнем случае используется механизм виртуализации памяти (с помощью жесткого диска). В таблицу отображения заносится длина сегмента L.

Мандатный доступ

Защита сегментов производится с помощью мандатов и доступов. Дескриптор является мандатом, который помимо точки входа в таблицу отображения, содержит требуемый доступ к сегменту. В таблице отображения указывается разрешенный доступ к сегменту. В случае их не противоречия друг к другу разрешается обращение к сегменту.

Каждый объект разделен на 2 части: системная часть и пользовательская часть. В системной части записываются мандаты (дескрипторы) для доступа к составляющим этот объект сегментам. Т.е. системная часть состоит из последовательности дескрипторов или мандатов для доступа, причем для одного и того же сегмента они могут быть разные: один сегмент может быть описан с разрешением записи. Другой сегмент может быть описан только с разрешением чтения, а третий сегмент может быть описан с разрешением только проверки о его наличии. Один и тот же сегмент может иметь несколько дескрипторов в таблице отображения.

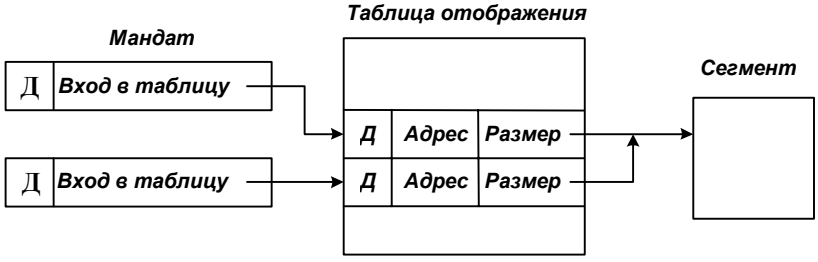


Рис. 5.3. Использование мандатов

Типы данных (объектов). В разных системах поддерживается разное число типов и названия, но основных 3:

1. Домен – это постоянная часть программы.
2. Контекст (запись активации) – сегмент, обеспечивающий повторную входимость в процедуру, функцию (реентерабельность). Фактически это стек процедуры и локальные статические данные процедуры. Здесь создается стек метода и статические данные метода, для того, чтобы обеспечить повторную входимость, чтобы

параллельно один и тот же домен могли использовать различные программы. Запись активации фактически создается при вызове метода и удаляется, когда метод завершил работу.

3. Процесс представляет собой набор объектов, необходимых для создания, выполнения и завершения процесса. Состоит из объектов типа «память», где память есть домен, который описывает временно выделяемую область; объектов, составляющих процесс (домен для кода, несколько записей архивации); контекста (записи архивации процесса); глобальные данные и т.д. Это защищаемая область адресного пространства, где описаны доступы, дескрипторы или мандаты для других сегментов. Тип объекта хранится внутри дескриптора. Системой аппаратно поддерживается (контролируется) различные типы доступа для различных объектов.

Домен защиты (Д) определяет тип операций, применимых к данному сегменту через данный дескриптор.

Типы доменов защиты:

1. Выполняемый (E);
2. Чтение (R);
3. Запись (W).

В связи с тем, что с использованием одного и того же дескриптора или мандата происходит последовательно несколько обращений, и каждый раз этот мандат должен размениваться через дескрипторную таблицу, придумали хитрость: сказано, что мандат или дескриптор может находиться в одном из сегментных регистров (CS, SS, DS, ES, FS, GS). Эти сегментные регистры имеют размер, равный размеру дескриптора какого-то сегмента или объекта. Т.е. в сегментных регистрах содержится дескриптор, и вы обращаетесь к сегменту относительно дескриптора, но на самом деле в дескрипторе хранится не адрес, а номер входа в таблицу дескрипторов. Адрес сегмента в памяти называется базой, длина сегмента называется размером.

При обращении к сегменту через сегментный регистр читается дескрипторная таблица, потом в команде вычисляется смещение относительно начала сегмента, с этим базовым адресом происходит сложение и обращается к области физической памяти. Значит, каждое обращение к памяти сопровождается обращением к памяти. Получается абсурд. Чтобы решить эту проблему, придумали хитрость: каждый сегментный регистр имеет теневой регистр с размером элемента таблицы отображения и фактически адрес сегмента, размер сегмента и т.д. хранится внутри процессора. Как только перезагружается сегментный регистр, процессор, зная адрес дескрипторной таблицы извлекает 64-битное слово и записывает его в теневой регистр. Адрес дескрипторной таблицы хранится в специальном регистре GDT, размером 64 бита, т.е. физический адрес хранится в таблице отображения и поэтому, как только меняется содержимое сегментного регистра, запускается программа, которая попадает в таблицу по заданному новому смещению (по новой точке входа в таблицу). Таким образом, всегда поддерживается соответствие сегментного регистра и теневого.

Тема 5.4. Стековая архитектура

Это родная архитектура всех языков программирования высокого уровня. В курсе теории автоматов обсуждался вопрос о том, что разрешимыми являются 3 типа языков: контекстные, контекстно-свободные и регулярные. Языки произвольные или типа 0 не разрешимы. Не зная наперед какую-то конкретную грамматику языка, вы никогда не сможете определить принадлежность строки языку.

Контекстно-свободные языки могут быть относительно эффективно распознаны магазинными автоматами. Но помимо того, что магазинный автомат необходим для распознавания, код, который порождается такого рода автоматом, тоже очень легко может быть исполнен на магазинном (стековом) автомате. Поэтому есть целый класс архитектур, которые близки к языкам высокого уровня тем, что они являются стековыми.

Чтобы пояснить, что такое стековая архитектура рассмотрим арифметическое выражение: $y = a/(-b * c - d)$.

Синтаксический анализатор или дерево грамматического разбора, которое потом преобразуется в семантическое дерево, определяет порядок выполнения операций для стековой вычислительной машины (для магазинного автомата).

Построим семантическое дерево для данного выражения (оно задает порядок интерпретации тех терминальных знаков, которые встречаются в этой строке).

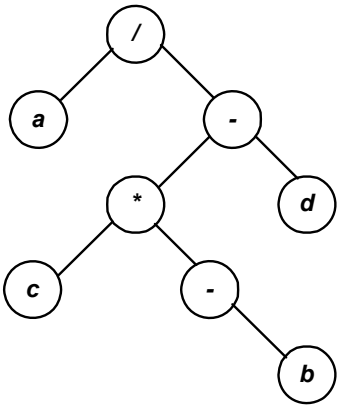


Рис. 5.4. Семантическое дерево

Для стековой архитектуры код может выглядеть следующим образом:

```
push b
not
push c
mul
push d
sub
push d
swap
div
```

Примеры стековых вычислительных устройств:

1. Математический сопроцессор i80x87;
2. FORTH-процессоры, JAVA-процессоры (специализированные процессоры, являющиеся в чистом виде конечными автоматами);
3. Стековая вычислительная система Эльбрус.

Достоинства стековой архитектуры

1. Высокое быстродействие. Оно определяется тем что код команд выполнен очень компактно и данные всегда находятся в регистровой памяти.
2. Уменьшение загрузки системного интерфейса процессора.
3. Языки программирования по своей внутренней структуре ориентированы на стековую архитектуру, что уменьшает сложность компилятора.

Что касается компактности кода: более компактного кода, чем код порождаемый стековой архитектурой ни одна система программирования пока не дает.

Недостатки стековой архитектуры

1. Психологические проблемы при программировании.
2. Необходимость реализации памяти со стековой архитектурой.

Тема 5.5. Перспективные архитектуры

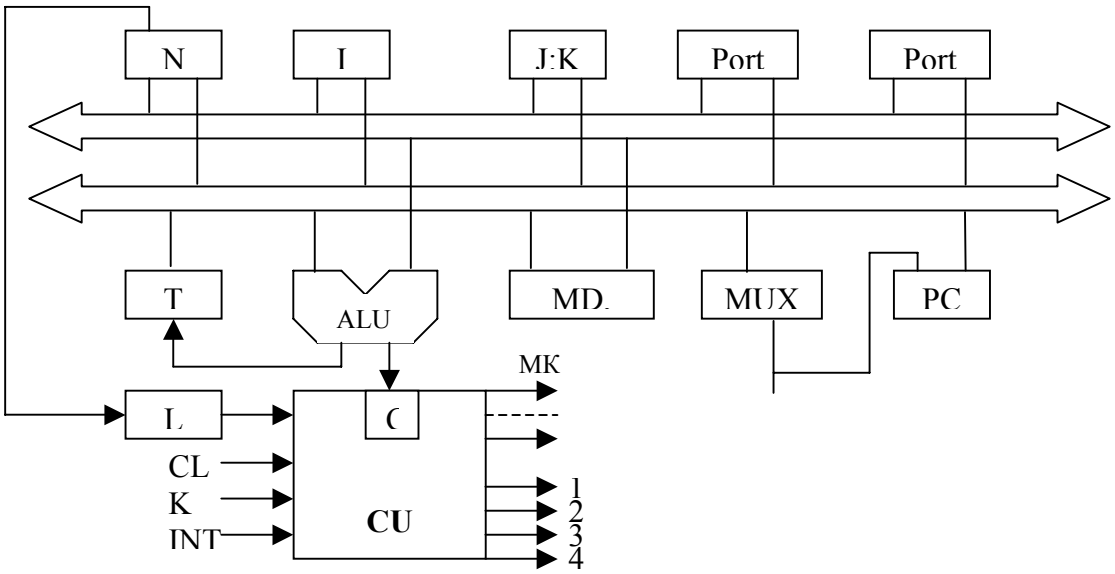
Система программирования FORTH

FORTH является стеково-ориентированной системой. В стеке могут находиться объекты, то есть каждую ячейку стека занимает объект, в более сложных случаях указатель на объект. Имеется входной поток, откуда черпаются данные. Входной поток по модели магазинного автомата является входной очередью. Во входном потоке расположены слова (лексемы языка), то есть входной поток является последовательностью слов.

Эта система является контекстной системой программирования, когда слово воспринимается не абсолютно, а в зависимости от контекста, т.е. в зависимости от объектов, которые находятся в настоящий момент на вершине стека. Чтобы слово можно было интерпретировать, имеется словарь слов. Структура словаря следующая: каждое слово определено не абсолютно, а относительно, поэтому имеется поле, которое называется контекст, что, означает, что это слово используется в этом контексте, и одно и тоже слово может иметь несколько словарных статей с разными контекстами. Входное слово ищется в словаре, причем учитывается текущий контекст компилятора (например, сложение целых и вещественных чисел). Помимо этого, надо определить, какие объекты вместо указанных (эти удаляются) появятся на вершине стека, поэтому каждое слово определяет еще и типы результатов. В режиме компиляции никакой работы с объектами не ведется, а ведется только с типами и стек фактически представляет собой ячейки, где записаны идентификаторы типов. Понятно, что каждое слово, встретившееся в программе, обозначает еще некоторые действия в заданном контексте, поэтому каждое слово имеет код, то есть те действия, которые выполняются над стеком, содержащем операнды, чтобы получить требуемый результат. И вот в момент компиляции в текущую ячейку кода заносится вызов этой функции (кода), который находится в словаре. В свою очередь этот код при компиляции является кодом другого слова, которое компилируется, т.е. как такого кода нет. Код и данные перемешаны. Все находится в словарях, т.е. кода отдельно от описателя кода не существует, поэтому там нет семантического разрыва между типами данных в языках высокого уровня, там все структурировано.

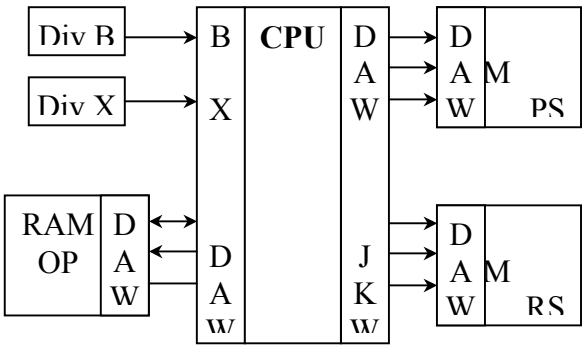
Рассмотрим архитектуру форт процессора на примере К1865 (94г)
Технология: КМОП, БИС.
Временные характеристики: тактовая частота 0-4МГц,
цикл внешнего интерфейса процессора 250нс,
одно и двухтактное выполнение команд,
время реакции на прерывание 250нс,
параллельное выполнение нескольких команд.

Структура процессора.



T – регистр вершины стека данных,
N - регистр под вершиной стека,
I – регистр вершины стека адресов возврата из подпрограмм,
1- запись в стек данных,
2- запись в стек адресов возврата,
3- запись в порт,
J (K) – указатель внешних стеков данных (возврата),
MD, SR – регистры расширенной арифметики,
PC – счетчик команд,
MUX – мультиплексор адреса.

Организация микропроцессорной системы.



PS – стек данных,
RS – стек адресов возврата,
OP – устройство хранения команд.

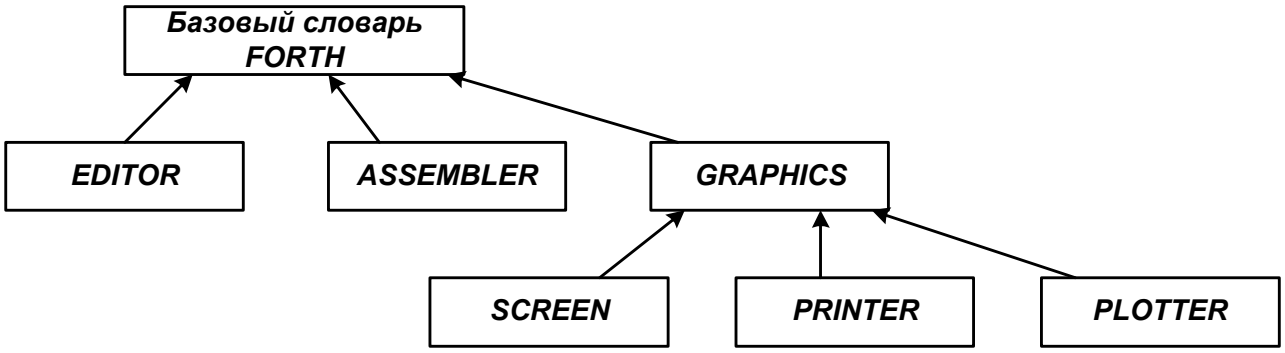


Рис. 5.5. Древовидная структура словаря

В момент компиляции анализируется входной поток, и в такт появления слова во входном потоке создается или абсолютная секция кода исполняемого, или создаются определения новых слов. Определение новых слов выглядит следующим образом:

: <имя> <слово>…<слово> ;

Имя – определяемое слово языка, слова – список слов, которые определены раньше и хранятся в древовидной структуре словаря. Эти слова определяют новое слово – имя, и должны быть исполнены, если в тексте встретится это слово.

Базовые слова:

1. Операции над стеком

SWAP	nm→mn
DUP	n→nn
DROP	nm→n
OVER	nm→nmn

2. Операции работы с памятью

@ a→m(a) по адресу a, находящемуся на вершине стека, читается содержимое

! na→ слово n записывается в память по адресу a.

3. Логические операции

NOT	n→n
AND	nm→p
XOR	nm→p
OR	nm→p

4. Операции над целыми числами

+, -, *, /, MOD, ABS, MIN, MAX, MINUS

5. Управляющие конструкции

IF XXX ELSE YYY
THEN ZZZ

Если на вершине стека истина, то выполняется XXX, иначе YYY, в любом случае – ZZZ.

BEGIN XXX
WHILE YYY REPEAT

Тело цикла XXX исполняется всегда, YYY вычисляет условие завершения.

6. Стек адресов возврата

>R	из стека операндов слово переносится в стек адресов возврата
R>	из стека адресов возврата слово переносится в стек операндов
R	слово из адресов возврата копируется в стек операндов

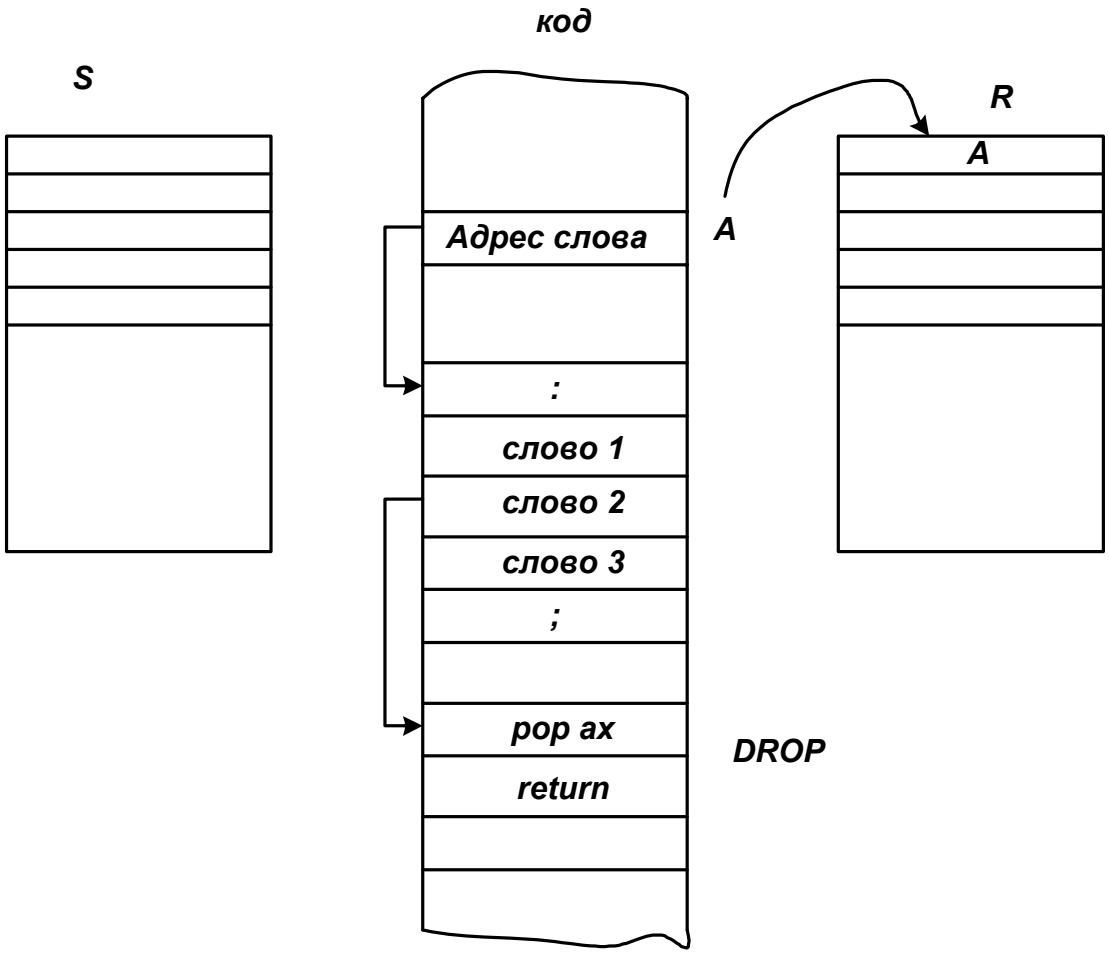


Рис. 5.6. Структура вычислительного механизма

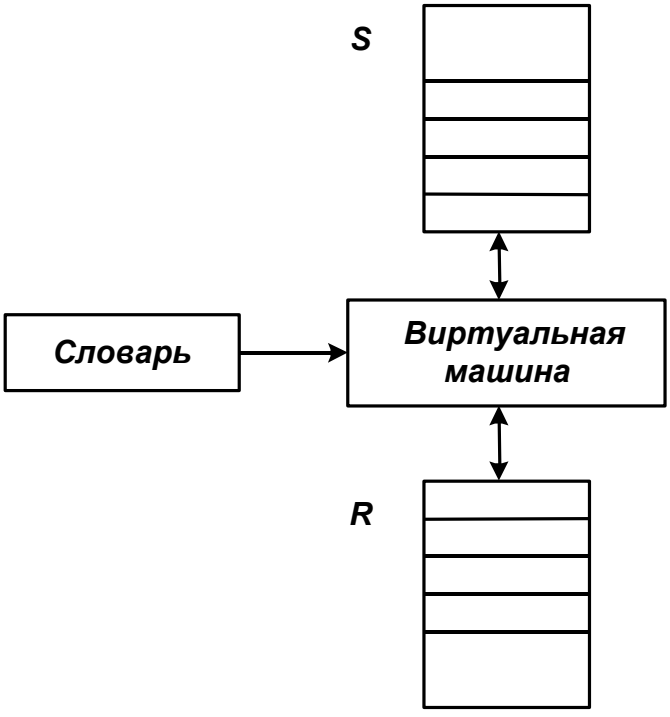


Рис. 5.7. Вычислительный механизм

Объектно-ориентированный FORTH

Недостатками система программирования FORTH являются:

1. Наличие только простых типов данных, с которыми работает виртуальная машина;
2. Бесконтекстное употребление слов.

Изменим структуру словарной статьи:

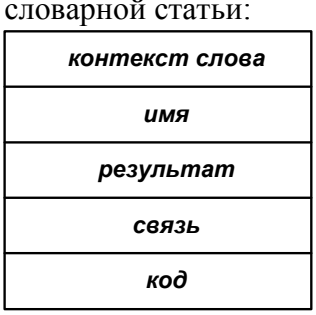


Рис. 5.8. Структура словарной статьи

Контекст слова – это последовательность слов – объектов, после которых употребляется слово – имя. Результат – это тот контекст, который это слово формирует.

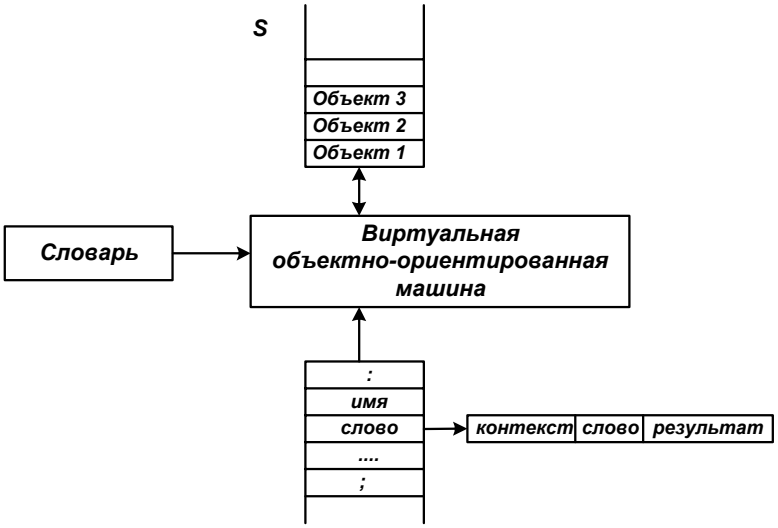


Рис. 5.9. Вычислительный механизм

В стеке операндов уже хранятся не слова, а объекты.

Традиционная система программирования FORTH не имеет контекста в указанном смысле. У нее все объекты одного типа. При компиляции проверяется, только наличие достаточной глубины стека операндов, т.е. чтобы реализовать FORTH – систему необходимо оставить, только, слово и код, и удалить контекст и результат, причем никакой привязки слова к типам объектам на вершине стека не происходит. Поэтому при написании программ на языке FORTH существует большая возможность сделать ошибки.

В объектно-ориентированной системе FORTH используются дополнительные поля в словаре: контекст и результат. Контекст определяет типы объектов и их последовательность, которые находятся в стеке, и при наличии которых может быть применено слово – имя. В результате действия слова вместо объектов контекста в стеке появляются объекты, описываемые полем результата.

При наличии аппаратной поддержки операционного поиска слов, исполнения кода, работы со словарем, может быть получено эффективное техническое решение, в котором встроены возможности использования полу контекстных языков.

Раздел 6. Высокопроизводительная обработка данных

Тема 6.1. Недостатки неймановской архитектуры

Основные черты неймановской архитектуры

В основу функционирования большинства систем обработки данных положена архитектура Неймана, основанная на следующих принципах:

1) Все операции выполняются в единственном устройстве обработке данных последовательно.
2) Программы и данные хранятся в единой последовательно адресуемой памяти с линейной(или одномерной архитектурой).
3) Отсутствие различия между данными и командами при их хранении в памяти – семантический разрыв.

Архитектура Неймана проста и логична, но обладает следующими существенными недостатками:

1) Семантический разрыв между языками высокого уровня и системой команд (обрабатывающего устройства).
2) Проблема согласования пропускной способности процессора и ОП, т.е. узким местом неймановской архитектуры является память.
3) Кризис ПО:
   a) стоимость разработки ПО >> стоимости разработки аппаратных средств
   b) низкая надежность ПО
   c) невозможность полного тестирования программы
4) Достигнут теоретический предел быстродействия.

Вывод: Архитектура фон Неймана представляется неадекватным средством при решении некоторых имеющихся задач обработки данных.

Подходы к повышению эффективности систем обработки данных

На основе анализа недостатков неймановской архитектуры можно три подхода к повышению эффективности систем обработки данных:

1) Совершенствование внутренней структуры (организации) системы обработки данных:

Оставляя неизменную архитектуру, повышение эффективности обработки данных достигается за счет усовершенствования внутренней структуры:

   a) использование кэш-памяти
   b) блочная пересылка
   c) иерархическая организация памяти
   d) использование RISC-архитектуры.

2) Повышение уровня команд:

Предполагает использование усложненной, модифицируемой системы команд:

   a) появление CISC-архитектуры
   b) создание архитектур для языков высокого уровня (стековая, тэговая, ОО)
   c) MISC-архитектура процессора (архитектура с модифицируемой системой команд)
   d) архитектура со сверхдлинным командным словом (VLIW) – суперскалярная архитектура (одновременное выполнение двух и более команд)
   e) конвейеризация и распараллеливание.

3) Использование нетрадиционных архитектур:

Предполагает радикальный отход от неймановской архитектуры.

a) используется распараллеливание вычислительного процесса или распараллеливание вычислений
b) новые вычислительные механизмы.

Примеры: Редукционные машины, нечеткие процессоры, нейрокомпьютеры, систолические (волновые) системы, процессоры логического вывода.

Тема 6.2. Кратные вычисления

Кратные вычисления – это процесс нахождения некоторой дискретной функции на нескольких наборах данных:

$Y = F(X_0, X_1, ..., X_{n-1},)$,

где X_i - переменные $\in N_{k_f} = \{0, ..., k_f - 1\}$ (возможно с различной значностью),

Y - результат вычисления $\in N_{k_f} = \{0, ..., k_f - 1\}$.

Обозначим $X^{(j)}$ - j-ый набор переменных: $X^{(j)} = \left(X_0{}^{(j)}, X_1{}^{(j)}, ..., X_{n-1}{}^{(j)}\right)$. Очевидно в этом случае: $Y^{(j)} = F\left(X^{(j)}\right) = F\left(X_0{}^{(j)}, X_1{}^{(j)}, ..., X_{n-1}{}^{(j)}\right)$.

Когда возникают кратные вычисления

1-ый уровень архитектурной иерархии вычислительных средств, на котором мы оперируем такими понятиями как процессы и входные данные (для этого процесса).

2-ой уровень архитектурной иерархии вычислительных средств, на котором мы оперируем такими понятиями как команда, операнд, т.е. процессор можно рассматривать как устройство кратного вычисления. Имеется последовательность команд и последовательность операндов.

3-й уровень (уровень операционных устройств) рассматривает такие понятия как операция, машинное слово. Имеется последовательность микроопераций и последовательность машинных слов.

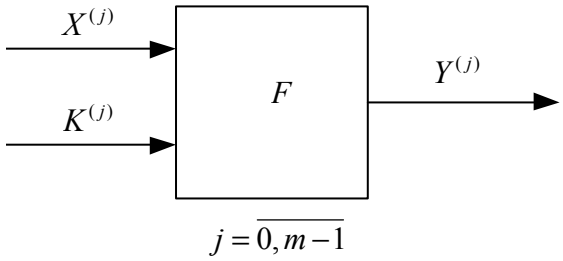


Рис.6.1. Схема кратных вычислений

Критерии эффективности кратных вычислений.

1) Эффективность по времени вычисления ξ:

Определим, как во сколько раз время однократно вычисления больше времени кратных вычислений в пересчете на один набор данных. Время однократных вычислений примем за единицу измерения времени.

$$\xi = \frac{1}{\frac{\tau}{m}} = \frac{m}{\tau},$$

где τ - время кратных вычислений.

$X^{(0)}(t), X^{(1)}(t+1),..., X^{(m-1)}(t+m-1)$

$Y^{(0)}(t+1), Y^{(1)}(t+2),..., X^{(m-1)}(t+m)$

2) Пространственная эффективность η:

Определяет во сколько раз пространственные затраты в однократном случае больше пространственных затрат при m-кратных вычислениях. За единицу измерения пространственных затрат или сложность вычислений, примем пространственные затраты в однократном случае

$$\eta = \frac{1}{\nu_m},$$

где ν_m – пространственные затраты при m – кратных вычислениях.

η – пространственная эффективность.

2) Комплексный критерий эффективности ϑ:

$$\vartheta = \xi \cdot \eta$$

При параллельных вычислениях во времени вычислительный процесс организуется таким образом, что все наборы входных данных одновременно подаются на вход устройства, а результаты вычислений одновременно снимаются с выхода, при чем комплексная эффективность таких вычислений должна быть больше единицы, что достигается путем параллельной декомпозиции вычисляемых функций в отличии от последовательных применяемых конвейерных вычислений.

Структурная организация кратных вычислений

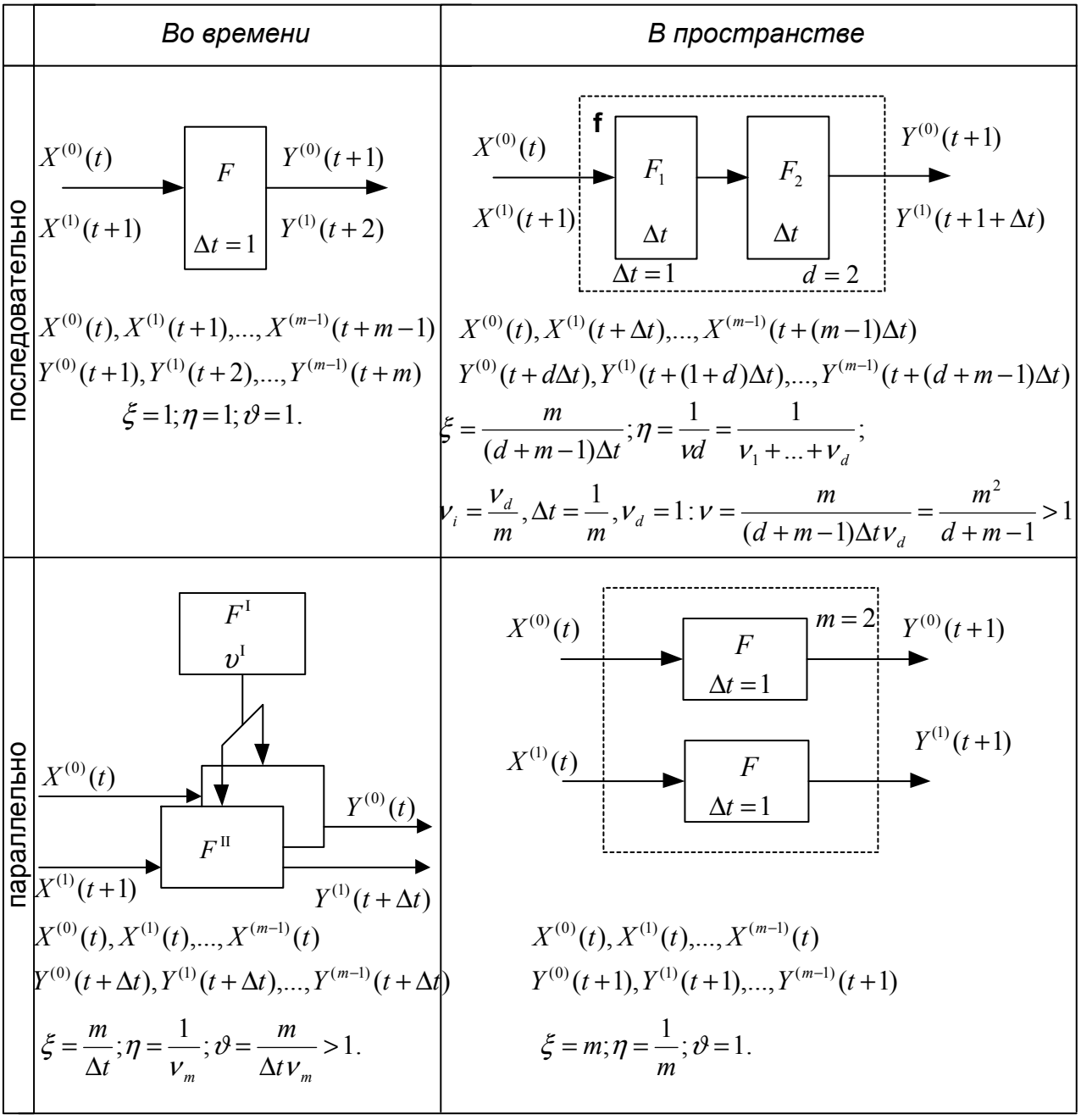


Рис.6.2. Структурная организация кратных вычмслений

Тема 6.3. Конвейерные и параллельные вычисления

Система обработки данных понимается как совокупность аппаратных и программных средств, предназначенная для автоматизации решения задач по обработке данных.

Конвейерная обработка данных

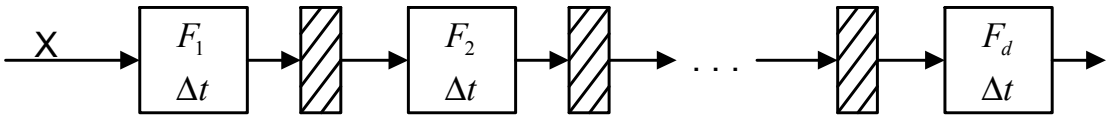


Рис.6.3. Структура конвейерного вычислителя

Функция $F(x)$ декомпозируется на несколько стадий вычислений этой функции. То есть функция $F(x)$ представлена в виде последовательности функций, которая может быть описана следующим выражением: $F(x) = F_d(...F_2(F_1(x))...)$. Если обработку данных можно представить в виде последовательности тактов обработки, таких что результат предыдущего такта обработки является входными данными для следующего, то можно реализовать обработку данных в виде конвейерного устройства.

Конвейерная обработка данных

Конвейерные преобразователи структурно будем различать:

1) По числу стадий в конвейере d.
2) По числу параллельных каналов:

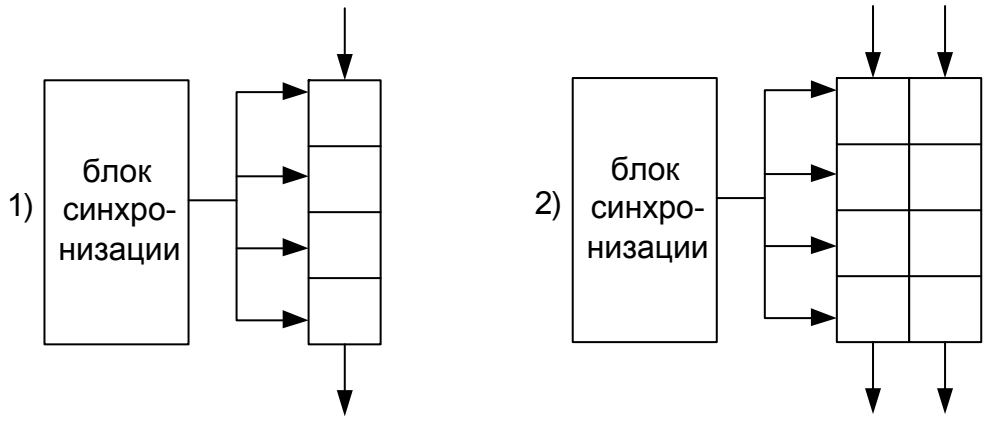


Рис.6.4. Конвейер с одним и двумя параллельными каналами

3) По режиму управления:
   a) Статические конвейеры;
   b) Динамические конвейеры;

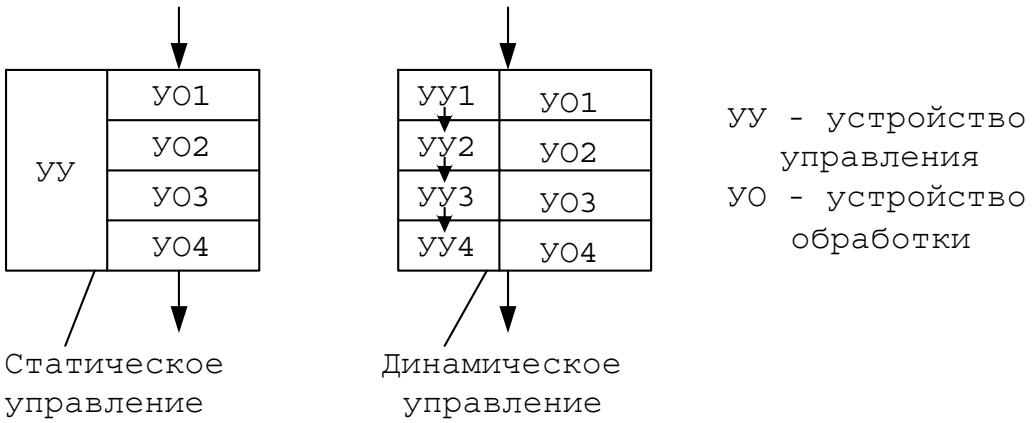


Рис.6.5. Конвейер со статической и динамической схемой управления

При статическом управлении задачи управления решаются статически одновременно для всех стадий.

При динамическом управлении синхронизация работы конвейера происходит на основе взаимодействия управляющих устройств смежных стадий.

4) по степени многофункциональности:
   a) однофункциональные конвейеры;
   b) многофункциональные конвейеры;

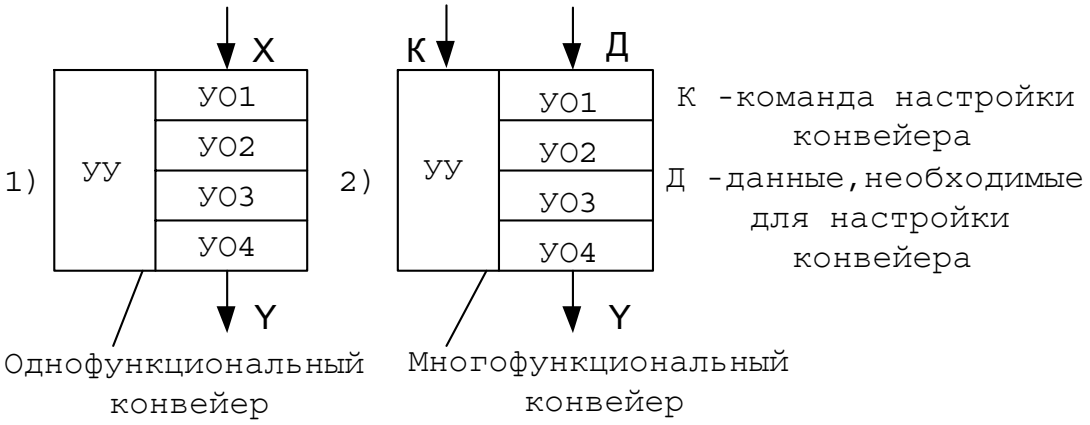


Рис.6.6. Одно- и многофункциональный конвейер

В первом случае происходит выполнение единственной операции конвейерной обработки.

Во втором случае функции, выполняемые конвейерным преобразователем, определяются управляющими данными К (возможно различными для каждого набора данных).

5) По способу организации связей:
   a) конвейерные преобразователи с фиксированной структурой;
   b) конвейерные преобразователи с переменной структурой.

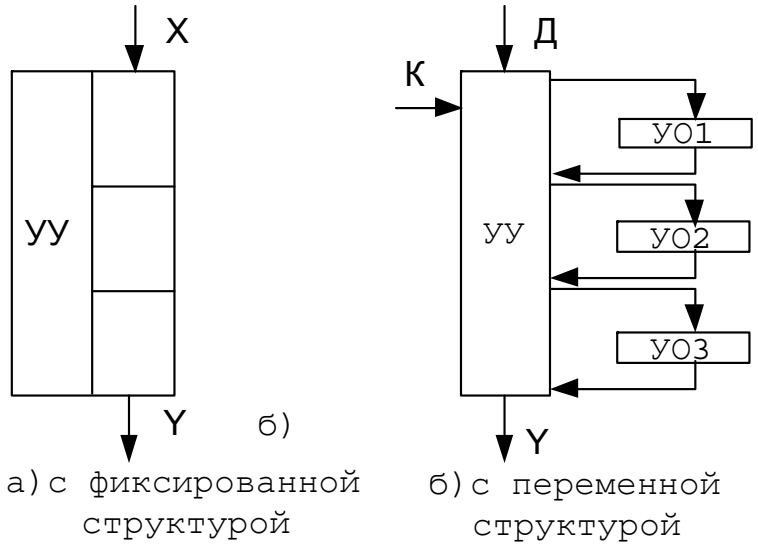


Рис.6.7. Конвейерные преобразователи с фиксированной и переменной структурой

Параллельная обработка данных

Как и последовательное вычисление во времени, параллельное вычисление в пространстве не требует декомпозиции дискретной функции. Конвейерная обработка данных требует последовательной декомпозиции функции. При параллельных вычислениях во времени выполняется параллельная декомпозиция дискретной функции. Каждая функция $F(X)$, заданная сразу на m наборах данных: $X^{(1)}, X^{(2)},..., X^{(m)}$, представляется в таком виде, что существуют общие выражения, участвующие при вычислениях на каждом наборе $X^{(i)}$ и от него независящее.

$$\begin{cases} F(X^{(1)}), \\ F(X^{(2)}), \\ ... \\ F(X^{(m)}) \end{cases}$$

Если окажется, что для вычисления каждой из этих функций необходимо сделать нечто общее, одинаковое, то обозначим это в виде функции F^I, которая не зависит от входных переменных, и функций F^{II} от них зависящие:

$$\begin{cases} F^I(m), \\ F^{II}(X^{(1)}, F^I), \\ ... \\ F^{II}(X^{(m)}, F^I) \end{cases}$$

Структурная классификация параллельных преобразователей данных

1) по числу параллельных каналов.
2) По доле общего оборудования β: $0 \le \beta < 1$

Если $\beta = 0$ - то параллельное вычисление в пространстве.

Если $\beta > 0$ - то параллельное вычисление во времени.

$$\vartheta = \frac{m}{\Delta t \nu_m} = \frac{m}{\nu_m}$$

ν_1 - в однократном случае.

$\nu_m = m\nu_1(1-\beta)$ - в m -кратном случае.

ν - число микросхем.

β - доля общих логических элементов.

$$\vartheta = \frac{m}{\nu_1(1-\beta)} = \frac{1}{1-\beta}.$$

3) По степени многофункциональности:
   a) однофункциональные;
   b) многофункциональные;

   Аналогично конвейерным преобразователям.
4) По способу организации внутренних связей:

С фиксированной или переменной изменяемой структурой.

Примеры параллельной обработки данных

I. Параллельно-конвейерная обработка данных.

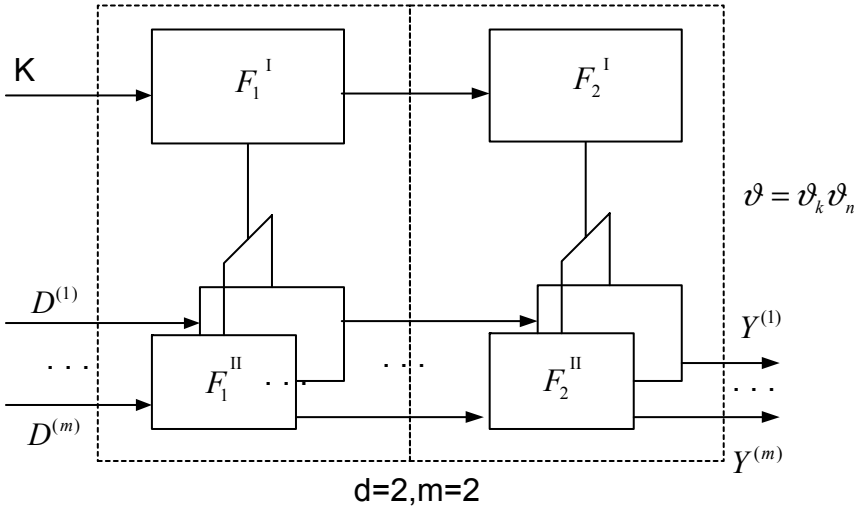


Рис.6.8. Схема параллельно-конвейерной обработки данных

II. Имеется устройство оптической обработки данных, состоящей из пространственного мультиплексора M и пространственного демультиплексора D, и канала передачи данных.

Имеется модулятор A, который изменяет характеристики сигналов, объединяемых пространственным мультиплексором для передачи по одному каналу. В канале смесь объединенных сигналов подвергается одной и той же обработке и на выходе получаем после их разделения пространственным демультиплексором результаты параллельной обработки во времени.

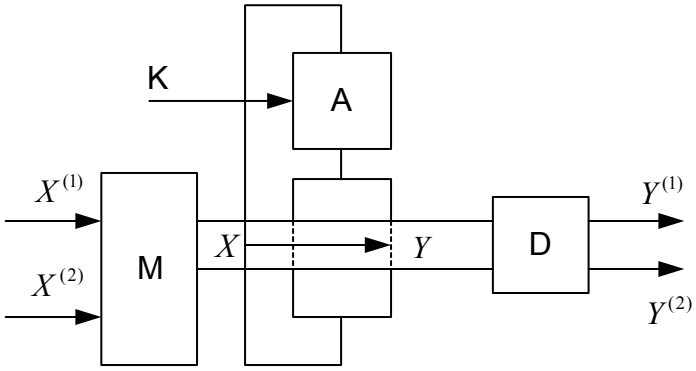


Рис.6.9.

Классификация Флинна (1966г.)

Процесс вычисления можно представить как взаимодействие последовательности команд программы (потока команд) с соответствующей ей последовательностью данных (поток данных), вызываемая этой последовательностью команд.

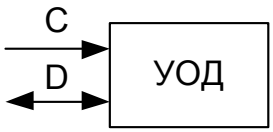


Рис.6.10. Процесс вычисления

Каждая система может включать как одиночные, так множественные потоки команд, данных. Множественный поток команд (данных) определяет наличие в системе нескольких последовательностей команд, находящихся в стадии выполнения или несколько последовательностей данных, подвергающихся обработке.

Выделяют 4 типа устройств (архитектур):

1) Одиночный поток команд, одиночный поток данных (ОКОД).
2) Одиночный поток команд, множественный поток данных (ОКМД).
3) Множественный поток команд, одиночный поток данных (МКОД).
4) Множественный поток команд, множественный поток данных (МКМД).

ОКОД:

Традиционно устройство обработки данных разделяют на 2 устройства: на устройство обработки (УО) и устройство хранения данных (ЗУ). ЗУ может быть разделено на 2 устройства: запоминающее устройство данных (ЗУД) и запоминающее устройство команд (ЗУК).

Устройство обработки можно разделить на 2 части: управляющий блок (УБ), операционный блок (ОБ).

ЗУК поставляет данные для УБ (один поток). ОБ получает данные и возвращает результаты в ЗУД. Это структура с одиночным потоком команд и одиночным потоком данных. Командный поток однонаправленный, поток данных – двунаправленный, но один поток, то есть поток команд вызывает соответствующий поток данных для архитектур с командным управлением, или поток данных вызывает поток команд для архитектур с управлением по данным. Управляющий блок воздействует на операционный. Есть еще и обратная связь. Пример: неймановская архитектура.

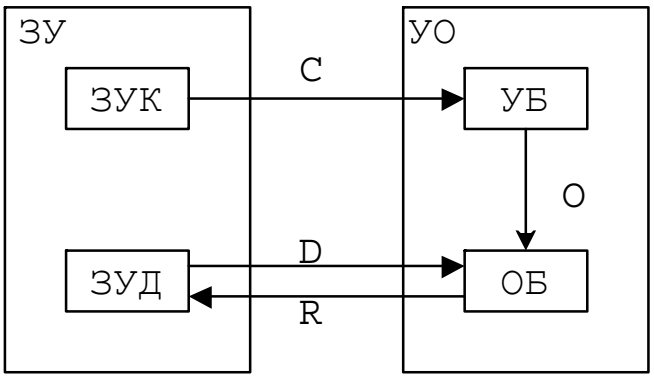


Рис.6.11. ОКОД

ОКМД:

По одной и той же программе обрабатывается несколько наборов данных одновременно. Существует несколько потоков данных и один поток команд. В классическом виде это похоже на параллельные вычисления во времени. Здесь имеется общий УБ (общее оборудование) для нескольких ОБ (для нескольких параллельных каналов). Примеры: матричные и ассоциативные системы обработки данных.

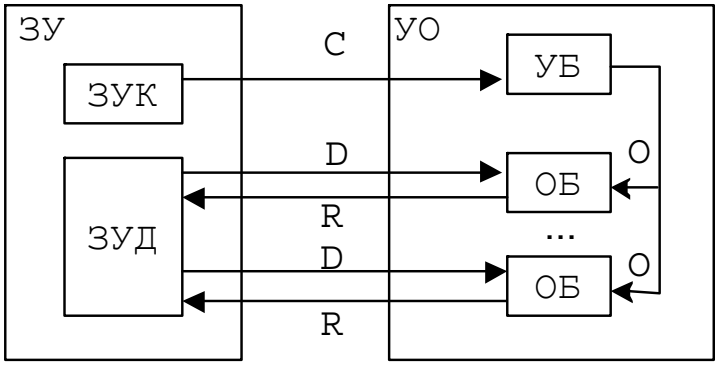


Рис.6.12. ОКМД

МКОД:

У нас имеется множественный поток команд и одиночный поток команд. Тут фактически над одними и теми же данными выполняется разная обработка. Этот класс архитектур считается плохо развитым, потому что такую обработку сложно придумать. Для создания таких устройств нет смысла создавать специализированные вычислительные средства на данном этапе развития. Полная аналогия с конвейерной обработкой данных. Данный класс назовем так – гипотетические конвейерные системы. Условно сюда можно отнести систолические системы.

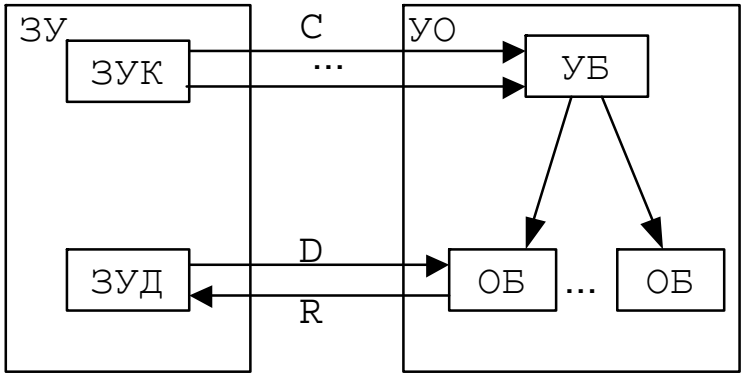


Рис.6.13. МКОД

МКМД:

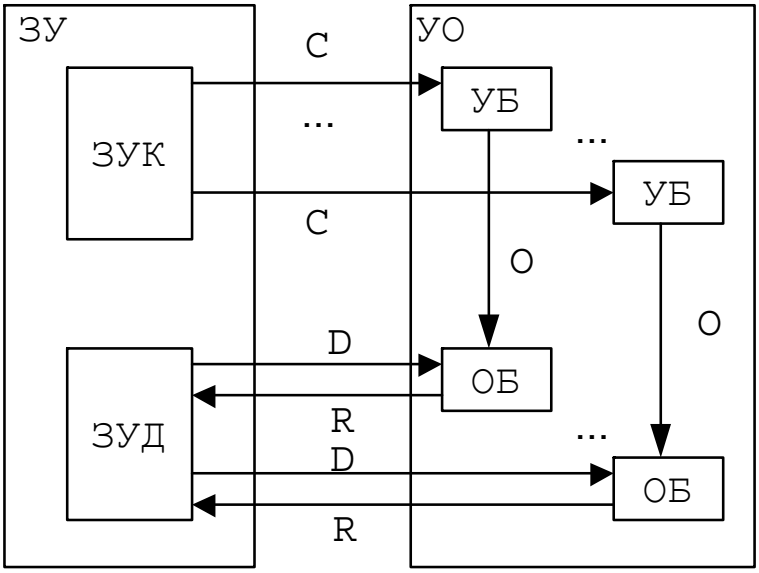


Рис.6.14. МКМД

Каждый поток команд вызывает соответствующий поток данных. Практически это параллельные вычисления в пространстве, когда для каждого параллельного канала есть полный комплект оборудования. Т.к. потоки команд не зависимы, то сложно выделить общее оборудование. К такому классу устройств можно отнести мультипроцессорные комплексы.

Классификация Скилликорна (1989г.)

Предполагается рассматривать архитектуру любого компьютера, как абстрактную структуру, состоящую из четырех компонентов:

1) Процессор команд IP – функциональное устройство, работающее как интерпретатор команд.
2) Процессор данных DP – функциональное устройство, работающее как преобразователь данных в соответствии с операциями, получаемыми от IP.
3) Иерархия памяти. IM - память для процессора команд IP. DM – память для процессора данных DP.
4) Переключатель (коммутатор) К – абстрактное устройство, обеспечивающее связь между CPU и памятью.

Автор зафиксировал четыре типа переключателей:

a) Тип 0: 1-1 переключатель, связывающий пару функциональных устройств какой-то статической связью.
b) Тип 1: n-n связывает i-е устройство из одного множества с i-м устройством из другого множества, т.е. фиксирует попарную связь.
c) Тип 2: 1-n. Соединяет одно устройство со всеми n функциональными устройствами.
d) Тип 3: n x n. Каждое функциональное устройство одного множества может быть связано с любым функциональным устройством другого множества и наоборот.

I-ый уровень классификации Скилликорна:

Классификация осуществляется на основе восьми характеристик:

1) Количество процессоров команд - IP.
2) Количество ЗУК – IM.
3) Тип переключателя между IP и IM - SW_I .
4) Количество процессоров данных – DP.

5) Количество ЗУД – DM.
6) Тип переключателя между DP и DM - SW_D .
7) Тип переключателя между IP и DP - SW_{ID} .
8) Тип переключателя между процессорами данных - SW_{DD} .

Пример: Рассмотрим архитектуру:

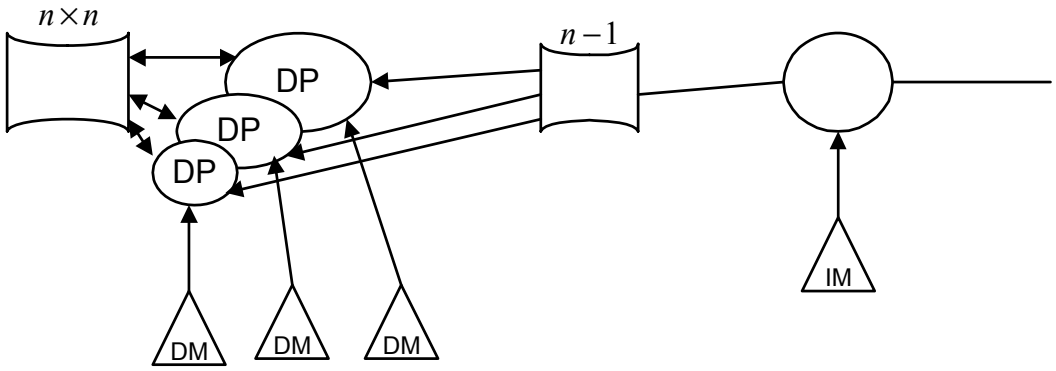


Рис.6.15. Архитектура (1, 1, 1-1, n, n, n-n, 1-n, $n \times n$)

В рамках этой классификации на I-ом уровне возможно 28 классов архитектур. В кассах 1-5 находятся архитектуры редукционных машин и машин, управляемых потоком данных, которые не имеют процессоров команд в обычном понимании этого слова. 6-й класс – это классическая неймановская архитектура. С 7-го по 10-й – всевозможные архитектуры матричных процессоров. С 11-го по 12-й – машины с множественным потоком команд и одиночным потоком данных. С 13-го по 28-й – всевозможные варианты мультипроцессорных систем, причем 13-20 – традиционно понимаемы, а 21-28 – экзотические мультипроцессорные архитектуры.

II-ой уровень классификации Скилликорна предусматривает уточнение описания, сделанного на I-ом уровне путем добавления возможностей конвейерной обработки данных.

Все системы обработки данных подразделяются в зависимости от особенностей взаимодействия составных частей на синхронные и асинхронные системы.

Тема 6.4. Архитектуры перспективных процессоров

Два подхода к повышению производительности процессоров:

1) Конвейеризация.
2) Распараллеливание во времени.

Архитектура процессора MISC

Недостатки CISC – архитектуры:

1) Далеко не все программы используют мощную систему команд универсального процессора полностью, что приводит к простою дорогостоящего оборудования.
2) Даже самые простые команды не могут выполняться за один такт.
3) Ограничение возможности оптимизации кода.
   a) Ограниченный размер СОЗУ
   b) Небольшое число режимов адресации и невозможность создавать новые способы доступа к данным.

Недостатки RISC – архитектуры:

1) Значительная загрузка системного интерфейса, нельзя обойтись без кэш-памяти принципиально.
2) Значительные затраты на программирование.
3) Увеличение доли времени на дешифрацию команды.

MISC (Multipurpose) – процессор с множественной системой команд

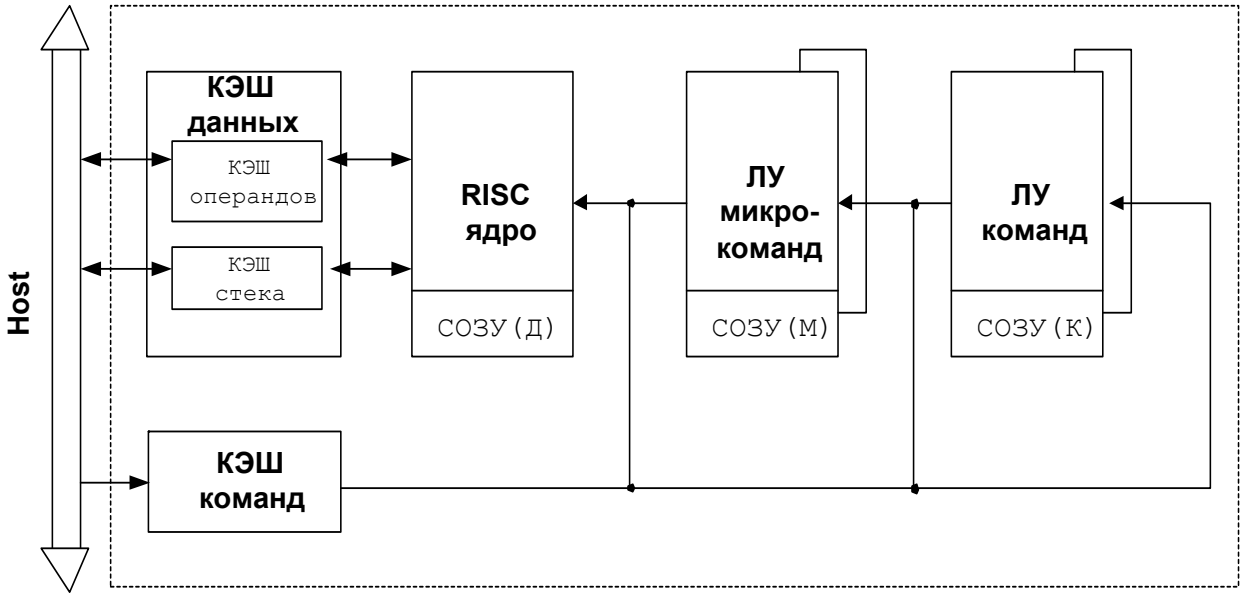


Рис.6.16. Структура MISC-процессора

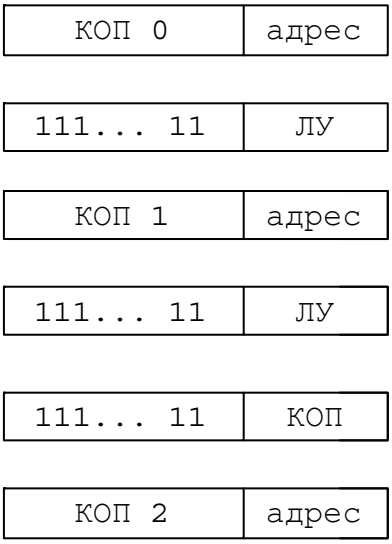


Рис.6.17. Система команд

Достоинства архитектуры MISC:

1) Содержит все достоинства CISC и RISC-архитектуры.
2) Возможность перезагрузки смены системы команд, в том числе и во время работы процессора, можно настроить процессор для выполнения конкретного типа задач.
3) Открытая архитектура процессора, которая позволяет наращивать число логических устройств как по вертикали, так и по горизонтали.
4) Возможность полной оптимизации кода программы с учетом реализуемой иерархической памяти для кода.

Процессор с конвейеризацией команд

Время исполнения произвольной команды разобьем на 6 частей:

G	D	A	O	E	P

G – выборка команд,
D – декодирование команд,
A – преобразование адреса (логический адрес в физический адрес),
O – выборка операндов,
E – выполнение команды,
P – сохранение результата.

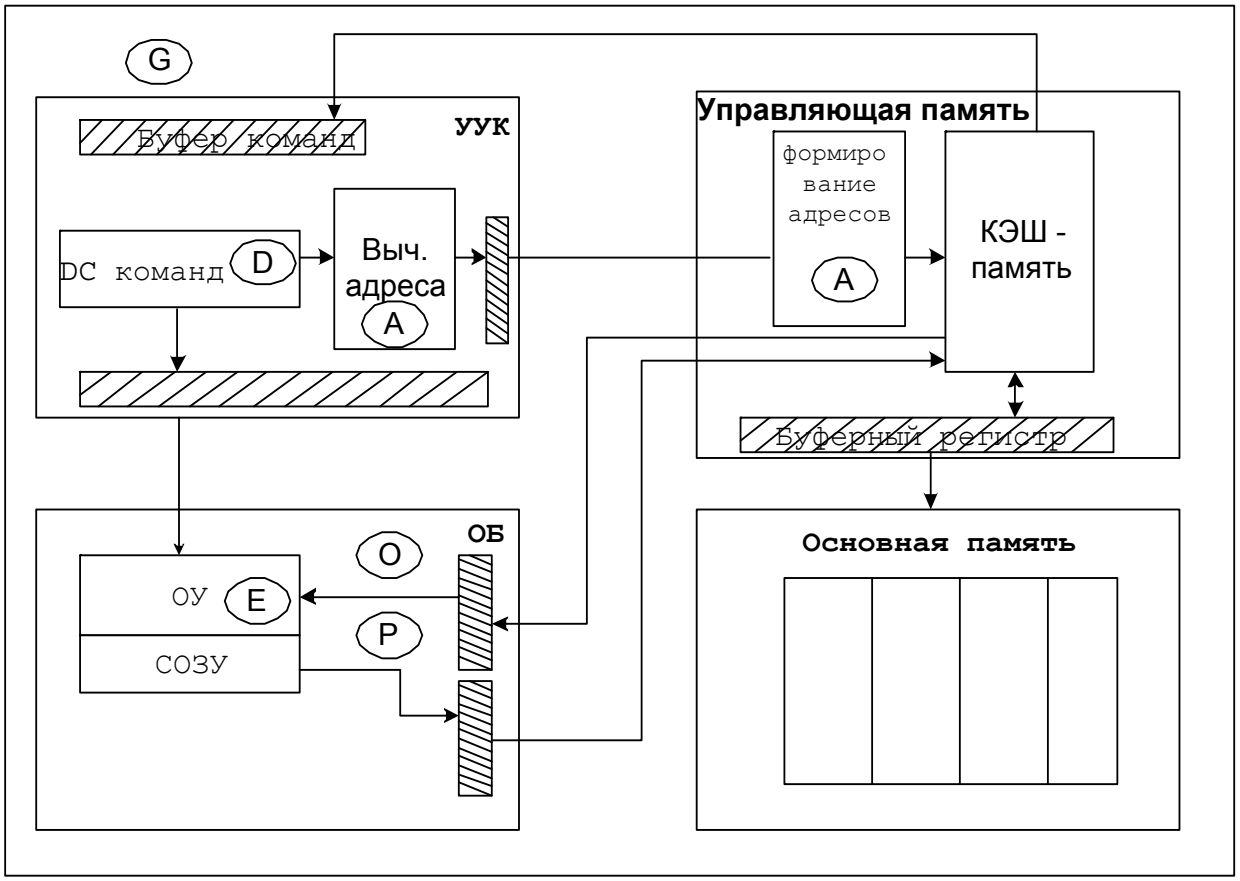


Рис.6.18. Структура процессора с конвейеризацией команд

Причины нарушения работы конвейера:

1) Нет связи по данным, когда используются результаты предыдущей команды в следующей команде. Устраняется путем выявления связи между командами и трассировки результатов на предыдущую стадию. Используются специальные регистры, недоступные программе в регистровом файле (РФ).
2) Связь по адресам. Адрес операнда определяется как результат предыдущей команды в конвейере.
3) Ветвление (нарушение последовательности команд). Приводит к перезагрузке конвейера путем запуска его работы сначала.
4) Конфликты доступа к памяти. 3 стадии одновременно (G, O, P) осуществляют обращение к ОП.
5) Длительность стадий должна быть одинакова, а поэтому длительные во время выполнения команды приводят к приостановке работы конвейера.

Архитектура со сверхдлинным командным словом (VLIW)

Принцип: Распараллеливание выполнение операций, то есть в одном командном слове кодируется несколько различных действий по преобразованию данных, что создает предпосылки для выполнения нескольких операций в одно и то же время.

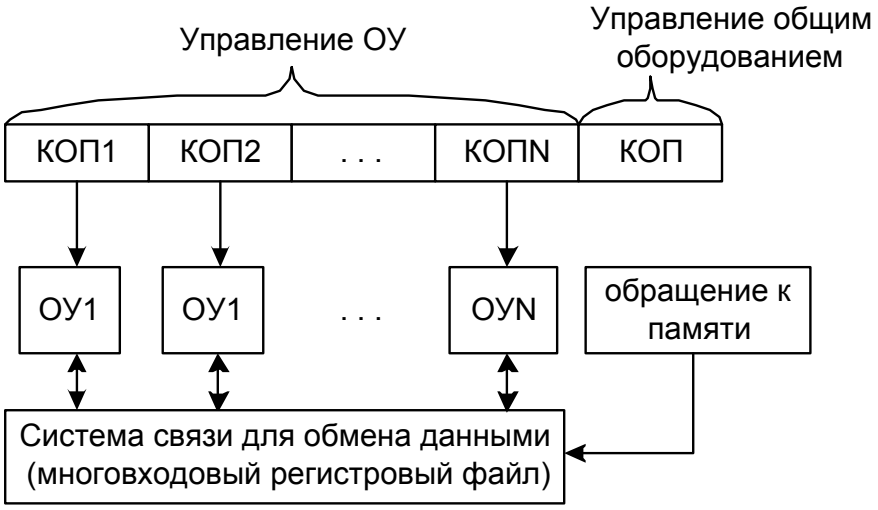


Рис.6.19. Принцип архитектуры VLIW

Для реализации этой архитектуры необходимо наличие нескольких ОУ, возможно различных.

Достоинства архитектуры VLIW:

1) Более простое устройство управления.
2) Компилятор может более эффективно использовать зависимость между командами, и переупорядочивать команды и подкоманды так, чтобы полностью загрузить все функциональные устройства.

Недостатки архитектуры VLIW:

1) снижение производительности из-за ветвления зависимости по данным (когда значения данных становится известны только в момент выполнения данной команды и потому не могут быть использованы в это же самое время другими операционными устройствами).
2) Сложный регистровый файл.
3) Не все команды могут закодировать в произвольной комбинации, т.е. не все подкоманды совместны в одной команде.

Особенностью данной архитектуры является статическая оптимизация кода.

Суперскалярная архитектура

Принцип: Реализовывается динамическая оптимизация кода процессором.

Имеется два подхода к отображению параллелизма задачи среднеблочном уровне:

1) Имеется явное указание на параллелизм в специальных полях команды.
2) Никакого указания на параллелизм в системе команд не содержится, но в процессоре содержится несколько параллельно работающих ОБ. Команды извлекаются последовательно и , в случае, если ОБ не занят, передаются на исполнение этому блоку.

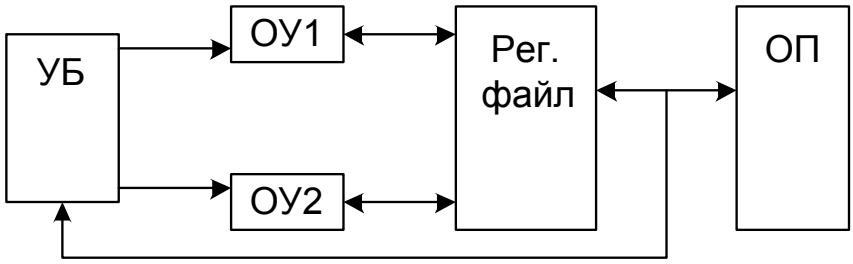


Рис.6.20. Принцип суперскалярной архитектуры

ОБ извлекает из ОП по одной команде, и по мере освобождения ОУ, параллельно управляет работой этих устройств, т.е. суперскалярную обработку данных можно представить себе как последовательность команд, которые извлекаются отдельно, и по мере освобождения ОУ, производится их выполнение.

При суперскалярной обработке происходит использование нескольких параллельно работающих ОУ при сохранении традиционной последовательности команд. Это позволяет реализовать возможности архитектуры со сверхдлинным командным словом при традиционном кодировании команд, когда в каждой команде содержится один код операции.

Компиляторы и аппаратные средства путем изменения последовательности команд позволяют оптимизировать использование процессора.

Основной процесс при суперскалярной обработке данных – это переупорядочивание, которое изменяет последовательность использования команд.

Существует два типа зависимости между командами:

1) Зависимость по управлению (условные и безусловные переходы, вызовы подпрограмм).
2) Зависимость по данным, т.е. когда результаты выполнения одной команды являются операндами для выполнения другой.

«Окно исполнения» – равно максимальному числу одновременно исполняемых команд.

Классификация зависимостей по данным:

1) RAR (Read After Read) – зависимость по данным отсутствует.
2) WAR (Write After Read) – фиксированная зависимость по данным.
3) WAW – фиксированная зависимость по данным.
4) RAW – истинная зависимость по данным.

Рассмотрим фрагмент кода:

```
label:  mov bx, addr    (1)
        mov ax, [bx]    (2)
        add bx, 4       (3)
        mov dx, [bx]    (4)
        loop label      (5)
```

1 и 3 – WAW,
1 и 2 – RAW,
2 и 3 – WAR.

Предположим имеется процессор с окном исполнения равным 2 для фрагмента вышеуказанного кода:

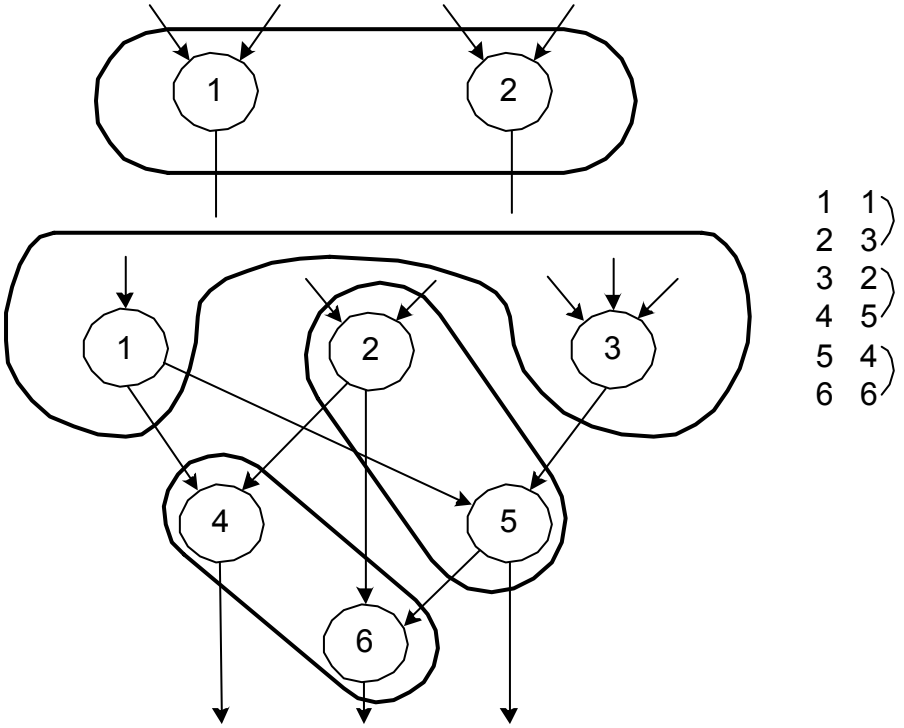


Рис.6.21. Граф зависимостей команд

Рассмотрим структурную схему суперскалярной обработки.

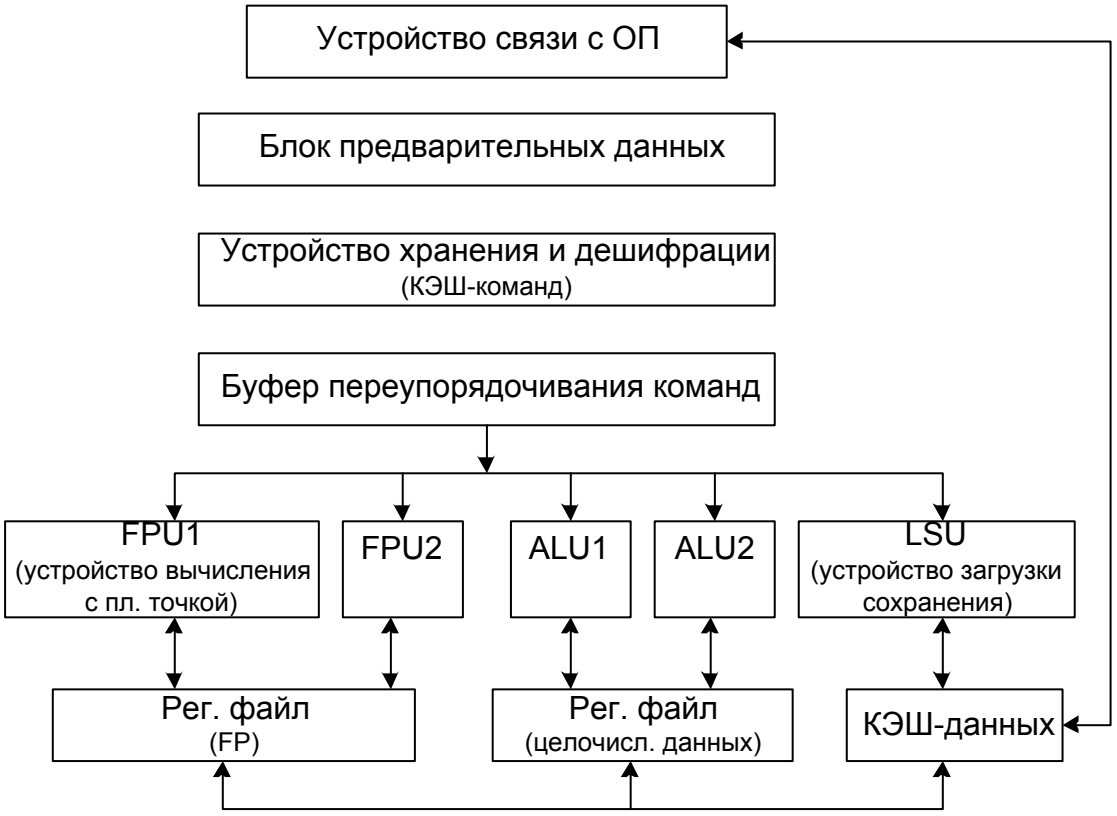


Рис.6.22. Структурная схема суперскалярной обработки

Мультискалярная архитектура

Принцип: Программа разбивается на совокупность задач (блоков) с помощью программных средств, каждой из которых назначается свой процессорный элемент для исполнения.

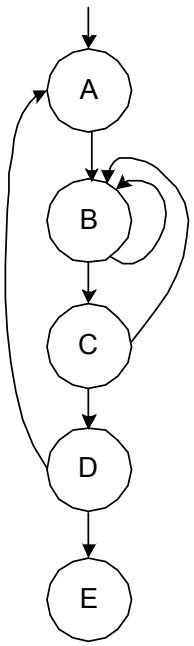


Рис.6.23. ГУЗ

Отсюда видно, что программу можно структурировать. Все программы могут быть представлены в виде графа. Зависимость между операторами программы по управлению представляется как граф управляющих зависимостей или информационный граф. В вершинах этого ориентированного графа располагаются базисные блоки, где под базисным блоком понимается последовательность команд, исполняемая на одном процессорном элементе.

Рассмотрим схему процессора, имеющего мультискалярную архитектуру.

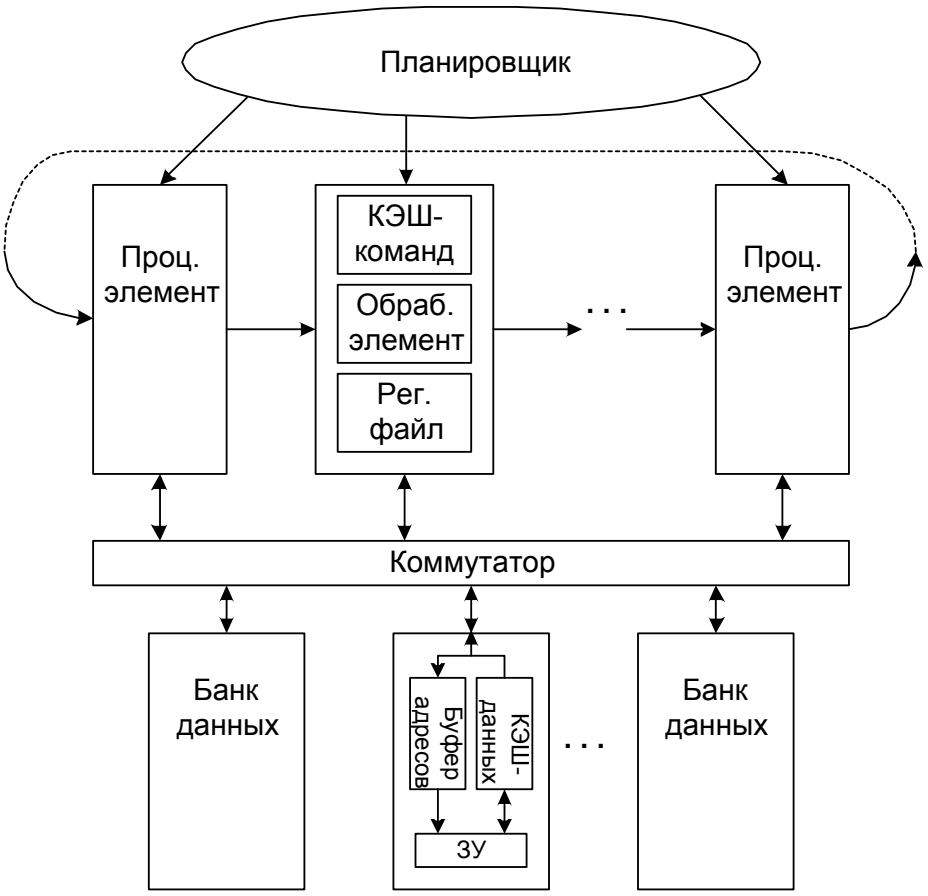


Рис.6.24. Структурная схема процессора, имеющего мультискалярную архитектуру

1) Каждый процессорный элемент выполняет команды своего базисного блока, находящегося в кэш-памяти команд.
2) Значение разделяемых процессорными элементами регистров копируется в каждый процессорный элемент. В этом случае может оказаться, что не все процессорные элементы разделяют один и тот же регистр. Результат модификации регистра динамически направляется через однонаправленное кольцо множеству процессорных элементов, но модификация регистров осуществляется только в том процессорном элементе, который не замаскирован для этой операции. По однонаправленному кольцу проходят команды, имеющие следующий формат:

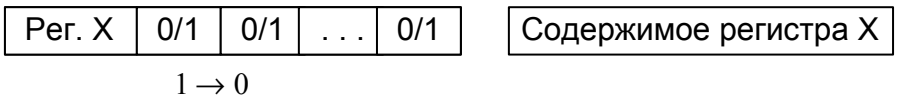


Рис.6.25. Формат команд, проходящих по однонаправленному кольцу

Маски генерируются статически компилятором, а динамически во время исполнения – планировщиком, - это оперативная связь между процессорными элементам.

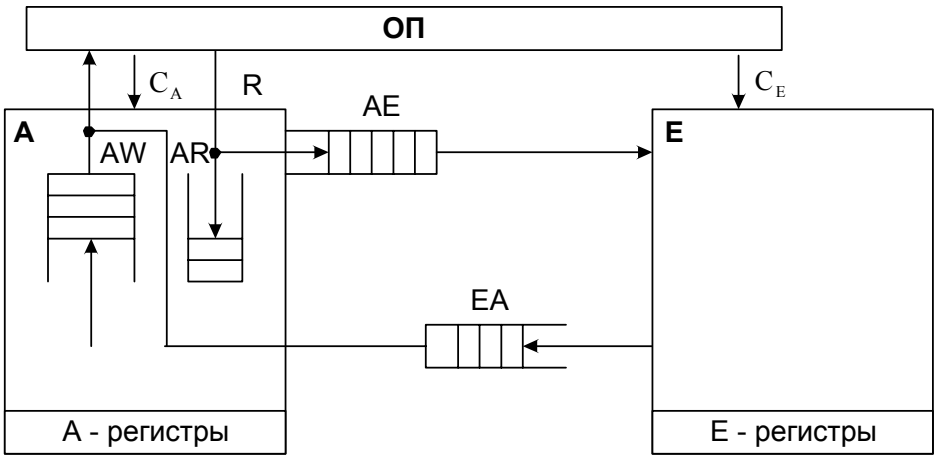
3) Обращение к данным каждым процессорным элементом осуществляется параллельно через коммутационную среду, обеспечивающую его связь с требуемым банком данных. Банк данных, помимо ЗУ, содержит кэш-данных и буфер адресов (очередь запросов). Планировщик выполняет назначение базисных блоков процессорным элементам и синхронизацию их исполнения.

Использование мультискалярной архитектуры возможно только при наличие распараллеливающего компилятора с языков высокого уровня.

Разнесенная архитектура (Decoupled)

Принцип: Используется естественный параллелизм вычисления выражений.

Под выражением понимается совокупность операндов, разделенных одноместными и двуместными операторами. Каждая операция ассоциируется с командой. Имеется проблема, связанная со следующим: содержимое ячейки памяти надо доставить к месту выполнения операции. Для получения значения операнда используются механизмы преобразования логического адреса в физический, которые требуют времени для выполнения.



где C_A - команды A-процессора,

C_E - команды E-процессора.

Рис.6.26. Структурная схема процессора, имеющего разнесенную архитектуру

Процессор состоит из двух связанных подпроцессоров, каждый из которых управляется собственным потоком команд – адресный процессор A (Address) и исполнительный подпроцессор E (Execute).

Адресный подпроцессор A выполняет адресные вычисления и формирует обращение к памяти по чтению/записи.

Исполнительный подпроцессор E выполняет операции по преобразованию данных.

Данные из ОП используются либо A- , либо E – процессором и помещаются в очередь AR или AE. Когда E-процессору требуются данные, он берет их из AE-очереди. Когда он вычисляет результат, то помещает их в очередь EA для записи в ОП. Так как A-процессор является целочисленным процессором (вычисляет адреса операндов), то имеется специальная очередь AW для сохранения результата вычисления.

Достоинства разнесенной архитектуры:

1) Достигается производительность векторных процессоров за счет предвыборки из памяти и автоматической развертки нескольких витков цикла.
2) Реализуется транзакционное чтение и запись, что позволяет частично решить проблему взаимосвязи по управлению.
3) Архитектура позволяет расположить между процессором и ОП коммутационную среду, что позволит легко построить многопроцессорную систему.

Раздел 7. Системы числовой обработки данных

Тема 7.1. Векторно-конвейерные системы (ВКС)

Принцип: Имеется две типа систем высокопроизводительной числовой обработки:
Матричные системы.
Векторно-конвейерные системы.

Итак, имеется некоторый конвейер с несколькими стадиями. Данные берутся из памяти и подаются на конвейер, и результат выполнения возвращается в память:

Рис.7.1.

Рассмотрим **пример**:

Предположим A, B и C – вектора (или одномерные массивы чисел). Тогда векторной операцией будет поэлементное сложение чисел, составляющих эти вектора и присвоенные соответствующим элементам вектора С:

$$C = A + B$$

$$A = [a_i],\ i = \overline{0, n-1}$$

$$c_i = a_i + b_i,\ i = \overline{0, n-1}$$ -

два n-чтений из ОП, n-сложений, n-присвоений (запись).

Рис.7.2.

Эту схему можно усложнить. Пусть A, B, C и D – вектора.

$$C = A + B$$

$$E = C \times D.$$

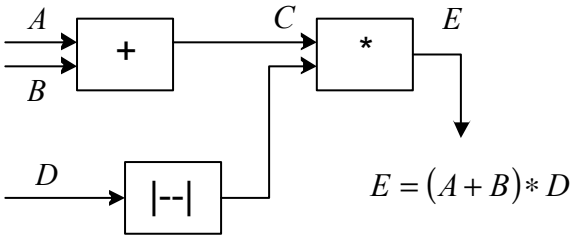


Рис.7.3.

Особенностью ВКС должна быть перенастройка конвейера для различных выражений. Рассмотрим **архитектуру ВКС**.

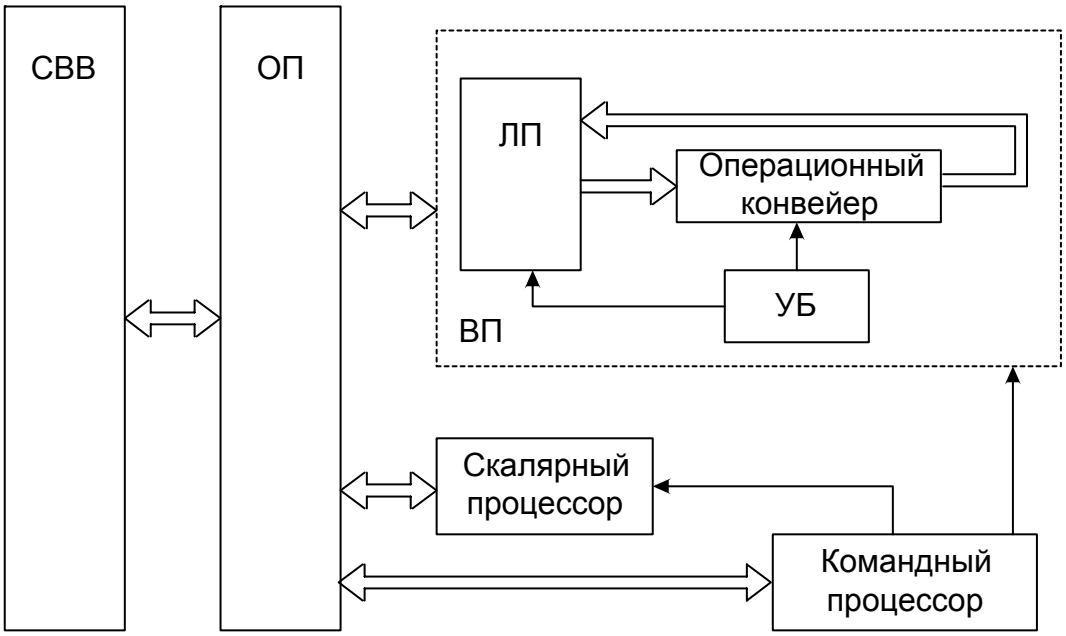


Рис.7.4. Архитектура ВКС

В векторных архитектурах используются многофункциональные конвейеры, которые имеют схемы статической или динамической настройки на определенную операцию (операции). При статической настройки конвейер настраивается перед началом выполнения векторной команды. При динамической настройки – в процессе ее исполнения путем перестройки конвейера.

Пример:

$$\begin{cases} C = A + B, & 1/2\,A,B \\ C = A - B, & 1/2\,A,B \end{cases}$$

Различают линейные и нелинейные конвейеры:

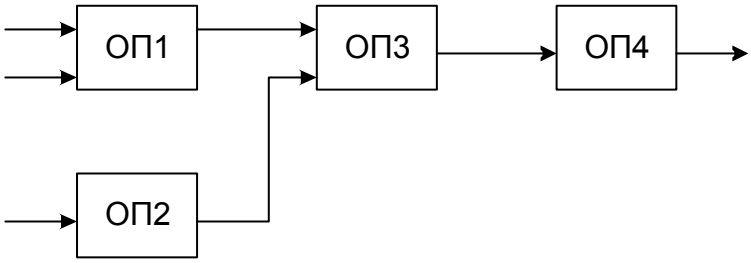


Рис.7.5. Линейный конвейер

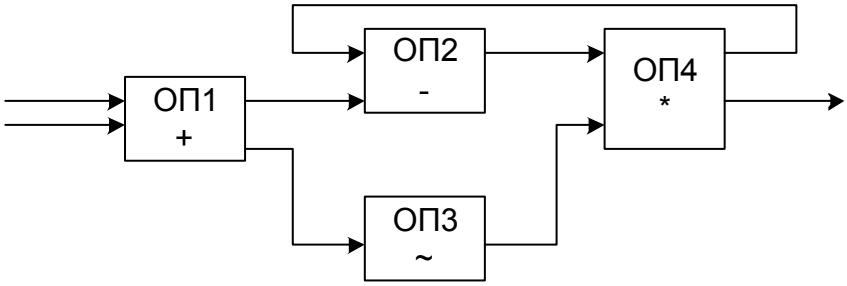


Рис.7.6. Нелинейный конвейер

Нелинейный конвейер позволяет организовать вычисление рекуррентных выражений:

$\sim (A+B)*(-A\_B)=C$ (без обратной связи)

Рассмотрим сцепление и распределение данных:

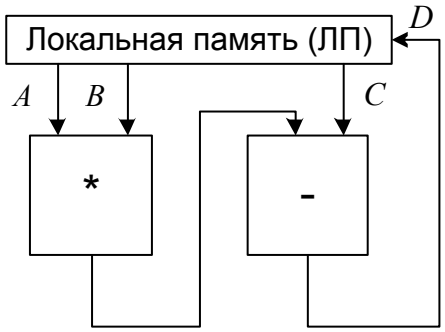


Рис.7.7. Сцепление данных

При **сцеплении данных** результат, полученный на предыдущих стадиях конвейера, поддается на вход следующих стадий конвейера, куда также передаются данные из ЛП (локальная память).

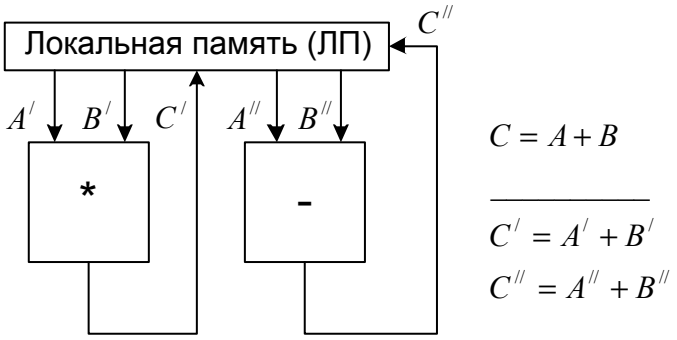


Рис.7.8. Распределение данных

При **распределении данных** векторная команда реализуется посредством нескольких однотипных операционных конвейеров.

В мире существует несколько десятков **Супер-ЭВМ**, построенных по векторно-конвейерному принципу. Понятно, что эти Супер-ЭВМ очень дорогостоящие. Приведем таблицу наиболее известных типов Супер-ЭВМ, организованных по векторно-конвейерному принципу:

Таблица 7.1.

Модель Супер-ЭВМ	Произв. В мегафлопп[1]	Число ПЭ	Длительность такта (в нс)		Объем цикла	Элементная база
			Для вект. Операций	Для скал. Операций		
CRAY-2	487	4	4.1	4.1	1Г/38нс	ЭСЛ[2] СИС[3] 0,35нс[4]
VP2000 Fujitsu	533	1	7.5	15.0	256Мб/ 60нс	ЭСЛ БИС 0,35нс
S-810 Hitachi	840	1	14.0	28.0	256Мб/ 70нс	ЭСЛ БИС 0,35нс
CYBER-205 CDC	40	2	20.0	40.0	1Г/50нс	ЭСЛ БИС 0,75нс

Пример ВКС:

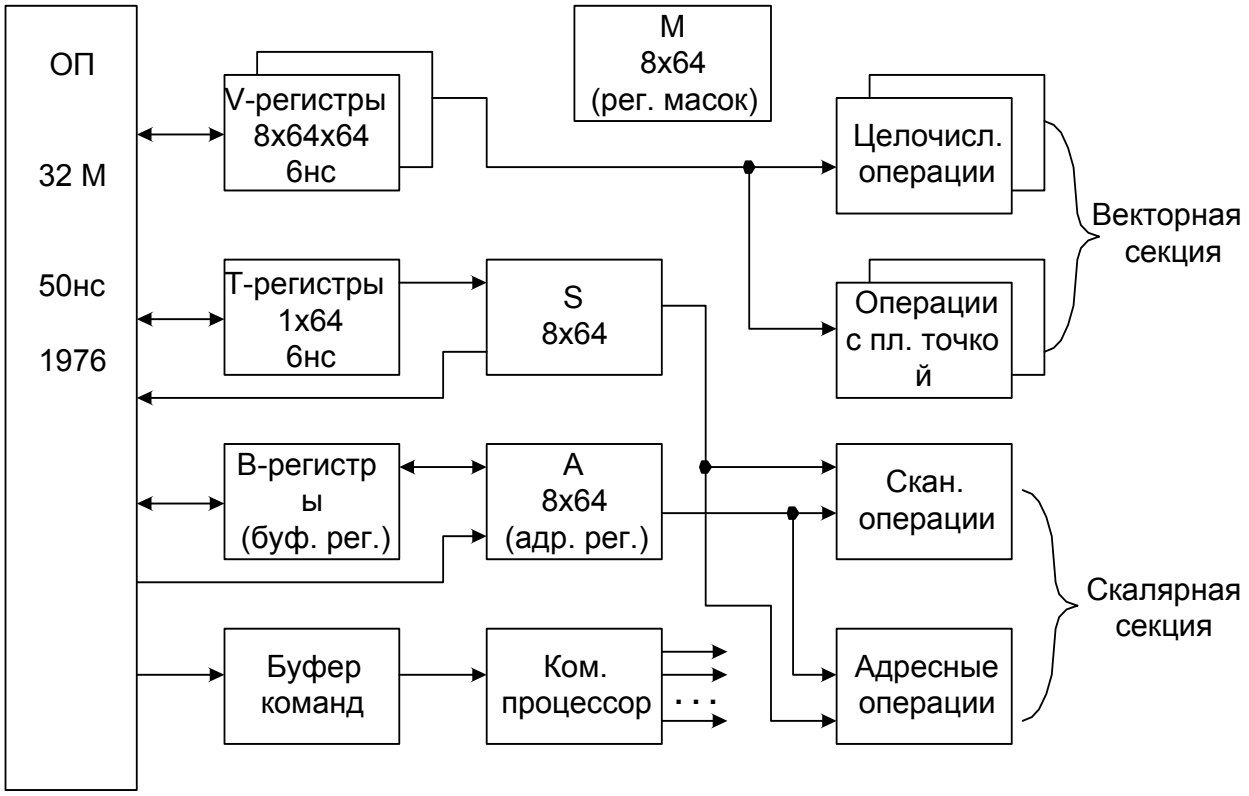


Рис.7.9. Супер-ЭВМ CRAY

[1] 1 мегафлопп – это 1млн. операций в сек.
[2] ЭСЛ – эмитерно-связанная логика
[3] СИС – средняя интегральная схема
[4] 0,35нс – базовый эл-нт переключается за 0,35нс

Конструктивной особенностью Супер-ЭВМ CRAY является то, что расстояние между функциональными узлами сказывается на его быстродействии. Поэтому она сделана в виде такого тора.

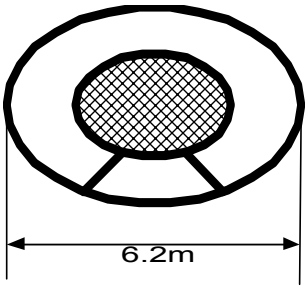


Рис.7.10.

В связи с тем, что если сделать ее вытянутую в линию, то окажется, что между двумя концами распространение сигнала столь длительное, что очень сложно синхронизировать работу частей, находящихся наиболее удаленно. Поэтому сделали в виде «бублика», чтобы эту проблему решить. Получили макс. длину связи 6.2 м.

Тема 7.2. Матричные системы (МС)

В матричных системах доля распараллеливания, как правило, превосходит долю вычислений выполняемых при конвейеризации, т.е. параллельные вычисления преобладают над конвейерными. Матричная система имеет много ПЭ, которые соединяются через коммутационную сеть для обмена данными или промежуточными результатами вычислений.

Матричные системы – это синхронные системы обработки данных типа **ОКМД**.

Принцин матричной обработки

Принцип матричной обработки: Несколько одинаковых обрабатывающих устройств выполняют одну и ту же последовательность команд над различными последовательностями данных.

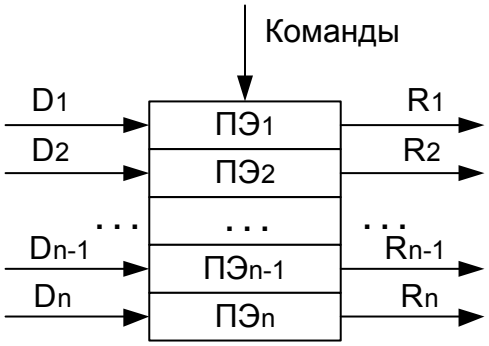


Рис.7.11. Матричный процессор

Структура матричной системы

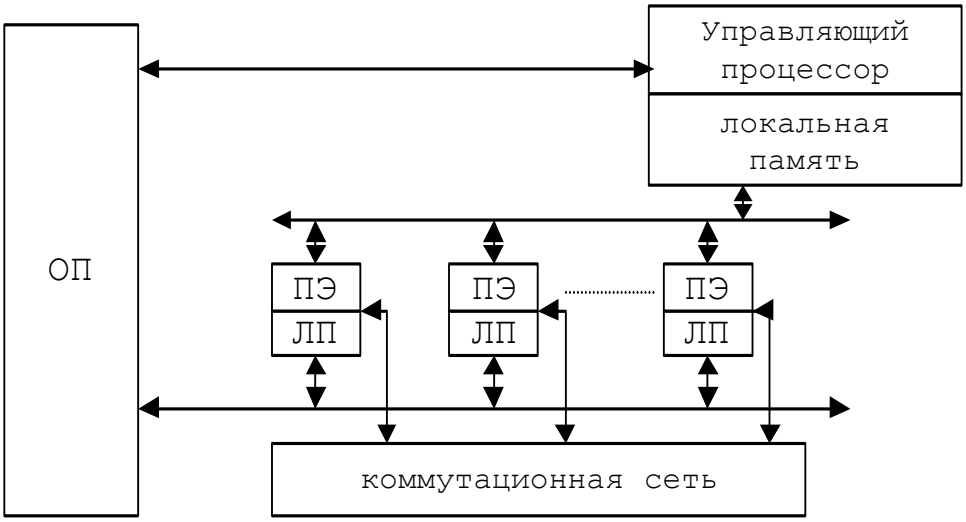


Рис.7.12. Структура матричной системы

Центральной частью любой ВС является основная память. Обязательно наличие специального управляющего процесса, потому что такая матричная система не может выполнять скалярную обработку данных. Поэтому есть специализированный скалярный процессор, называемый управляющим процессором, и он имеет свою локальную память. Он обменивается с основной памятью и выполняет скалярные операции. Сам управляющий процессор может иметь свою кэш-память, взаимодействующую с локальной памятью, и может иметь регистровый файл. Управляющей процессор управляет работой нескольких процессорных элементов, которые составляют основу этой матричной системы. Каждый ПЭ также имеет свою локальную память. Управляющий процессор выдает команды или настраивает вычислительную матрицу, образованную ПЭ. Поэтому есть некий интерфейс управления и в рамках этого интерфейса происходит обмен по управляющим воздействиям. Понятно, что для работы ПЭ, помимо доступа к ячейкам локальной памяти, необходимо чтобы осуществлялся доступ к основной памяти, поэтому есть еще один СИ, который обеспечивает доступ ПЭ к основной памяти через локальную память. Но этого оказывается мало. К сожалению, существуют задачи, которые не распараллеливаются, потому что иногда бывают связи, то есть рез-тат работы одного ПЭ должен подаваться на вход другого. Здесь же такой возможности нет, есть возможность сделать это ч/з ОП. Ясно, что это не мало эффективно, потому что доступ к памяти всегда проблематичен, поэтому в матричных системах используется специальная коммутационная сеть для обмена данными м/у ПЭ. Т.о. существует двусторонний обмен ПЭ-тов через коммутационную сеть. Управляющийпроцессор, получив команду, организовывает внутреннюю шину управлния. Данные по специальной шине параллельной загрузки (ШД) поступают в ЛК ПЭ-тов (загружаются последовательно). Скалярные операции реализовываются управляющим процессором. Матричные операции – массивом ПЭ-тов. Управляющий процессор передает одну и ту же команду на все ПЭ. Загрузка данных в ЛК ПЭ-тов осуществляется последовательно из ОП. Матричные команды управляющим процессором выделяются из потока команд, преобразуются и передаются на исполнение всем ПЭ.

Матричные команды

Рассмотрим **матричные команды**:

Сложение матриц:

Пусть заданы две матрицы A и B. Необходимо вычислить матрицу C (n x m), которая равна поэлементной сумме этих матриц: $C = A + B$. Элемент i,j матрицы C есть сумма соответствующих элементов матриц A и B:

$$c_{ij} = a_{ij} + b_{ij},$$

где $i = \overline{1,n}$, $j = \overline{1,m}$.

а) Пусть имеется n x m ПЭ. Элементы матриц A и B можно разделить по локальной памяти ПЭ-тов, т.е. процессорный элемент i, j содержит соответствующие элементы a_{ij}, b_{ij}, c_{ij}.

- Управляющему процессору для выполнения операций сложения достаточно передать ПЭ команды:

ADD A,B,C.

- Рассмотрим случай, когда ПЭ меньше, чем число элементов в матрицах:

При меньшем числе ПЭ-тов можно разделить матрицы по строкам, чтобы в ЛП $ПЭ_i$ находились:

a_{ij}, b_{ij}, c_{ij} $i = \overline{1,n}$, $j = \overline{1,m}$.

Тогда должно быть n ПЭ. В этом случае управляющий процессор передает m – команд, при выполнении каждой такой команды в ПЭ будет формироваться по одному элементу c_{ij}, а все ПЭ одновременно один k-ый столбец матрицы C.

- Использование индексных регистров:

При наличие индексных регистров управляющий процессор передает одну команду с указанием индекса и количества сложений.

Пример:

ADD A, B, C, ind (, cnt) ,

где ind – индексный регистр, cnt – число операций сложения.

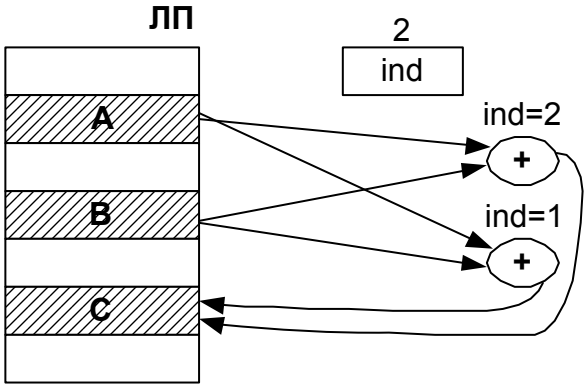


Рис.7.13.

Каждый массив хранится в своей области памяти. Индексный регистр определяет число складываемых элементов и одновременно смещение в соответствующем массиве текущего элемента. После выполнения индексный регистр уменьшается, что обеспечивает доступ к следующим элементам массива. Окончание вычисления, когда индексный регистр равен нулю.

- Загрузка ЛП:

Load A1,A2,ind

Поток данных, расположенных начиная с адреса A1 в ОП, передается по адресу A2 в ЛП ПЭ-тов последовательно(по порядку) в каждый ПЭ. ПЭ принимает данные по адресу A2, если его номер равен содержимому индексного регистра, а если нет, то данные не принимаются. После передачи очередного слова ПЭ-ты уменьшают индекс на единицу.

Замечание: *В каждом из рассмотренных случаев отсутствует взаимодействие между ПЭ, т.е. исходные данные выбираются из ЛП и туда же помещаются.*

- Скалярное произведение векторов:

$$c = \sum_{i=1}^{n} a_i b_i = \sum_{i=1}^{n} c_i$$

Элементы a_i, b_i распределены по ЛП ПЭ. Вычисление C требует передачи данных между ними. Управляющий процессор выдает индексную команду в ПЭ, которая заставляет все ПЭ, кроме одного, выдавать данные из ячейки c_i на ШУ. Все поступившие данные УП суммирует. В более сложных случаях обеспечивается пересылка данных через коммутационную сеть с использованием режима маскирования ПЭ.

- Рассмотрим матричные операции условной обработки:

Стоит задача вычислить следующую систему выражений:

$$\begin{cases} c_i = a_i + b_i, a_i > 5 \\ c_i = a_i - b_i, a_i \le 5 \end{cases}$$

Необходимо обеспечить выполнение различными ПЭ разных операций.

Режим маски:

Прием команд от УП осуществляется только в том случае, если специальный триггер маски ПЭ сброшен в 0(ПЭ - не замаскирован).

```
        TEST A,5
MASK LE  (устанавливает маску, если A>5)
        ADD A,B,C
        COMP MASK  (инвертирование масок ПЭ на противоположные)
        SUB A,B,C
```

В ПЭ используются такие разряды, как маскирование, конфигурация и флаги.

Матричная система ПС-2000

Примером МС является **ПС-2000.**

ПС-2000 состоит из 8 модулей, управления каналом и мониторной системой.

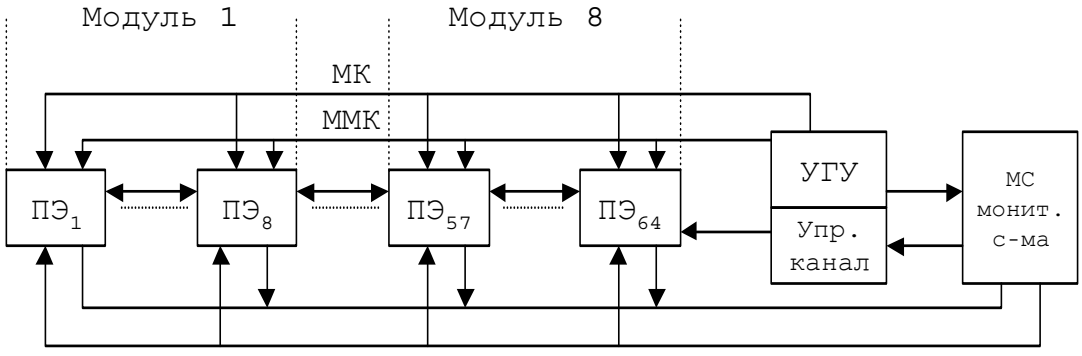


Рис.7.14. ПС2000

УГУ – устройство группового управления.

Имеется линейка 8 модулей по 8 ПЭ. Также есть магистраль команд(МК), которая используется для передачи команд на ПЭ. Есть хорошее архитектурное решение в этой системе: оказывается в модули ПЭ передаются определения некоторых команд в виде микрокода. Поэтому рисуем еще одну магистраль – магистраль микрокоманд(ММК). Т.е.

есть базовый набор команд, но можно задавать с помощью микропрограмм новые команды. Имеется двунаправленный регулярный канал(РК) для передачи данных между соседними ПЭ. Данные поступают из мониторной системы. Имеется два независимых канала при магистрали ввода и магистрали вывода. Мониторная система – это какой-нибудь персональный компьютер. Матричная система не является самодостаточной, она встраивается, как некий блок в ВС универсального назначения. Она обеспечивает загрузку данных в локальную память ПЭ и организует работу УГУ.

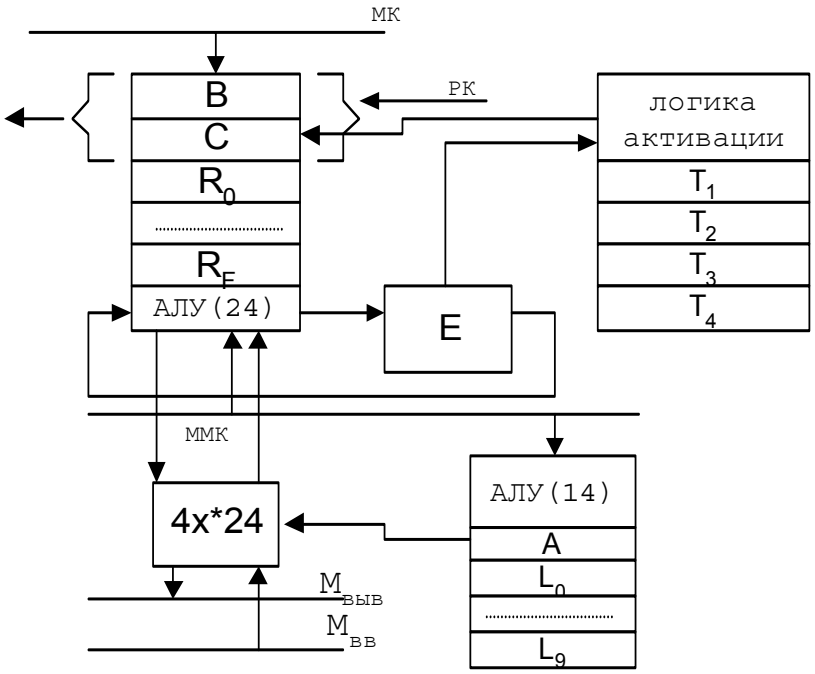


Рис.7.15. Процессорный элемент

Есть блок регистров, называемых регистры арифметико-логического устр-ва:B,C,R_0,..R_F. Эти регистры 24-х разрядные. Они подключаются к АЛУ. Понятно, что регистры В и С участвуют в обмене по регулярному каналу. Имеется специальный регистр Е, называемый регистром расширения для выполнения операций с двойной точностью, т.е. 48 разрядов. АЛУ имеет вход с модуля микрокоманд и фактически операции АЛУ изменяются. Имеется память 4 КБ 24-х разрядовых слов. К этой памяти подключается магистраль ввода и магистраль вывода. Имеется 14 разрядное АЛУ для адресной арифметики, которая управляется на уровне микрокоманд и позволяет эффективно реализовывать сложные алгоритмы пересчета индексов: А – для задания адреса локальной памяти, L_0-L_9 – регистры, используемые АЛУ. Имеется логика активации, т.е. активируется этот ПЭ или нет, служащая стеком признаков. Содержит 4 набора признаков: Т1, Т2, Т3, Т4. Логика активации должна получать признаки рез-тата из АЛУ.

Тема 7.3. Ассоциативные системы (АС)

Принцип Ассоциативной обработки

Одним из принципов Неймана является хранение данных в ЗУ с линейной организацией, и что приводит к тому, что содержимое памяти носит субъективный характер, т.е. определяется самой программой. Это является узким местом архитектуры Неймана, так как ОП используется лишь для пассивного хранения данных.

Основной принцип ассоциативной обработки: приблизить обработку данных к месту их хранения, и тем самым решить проблему доступа к памяти.

Рассмотрим **структурную схему** АС:

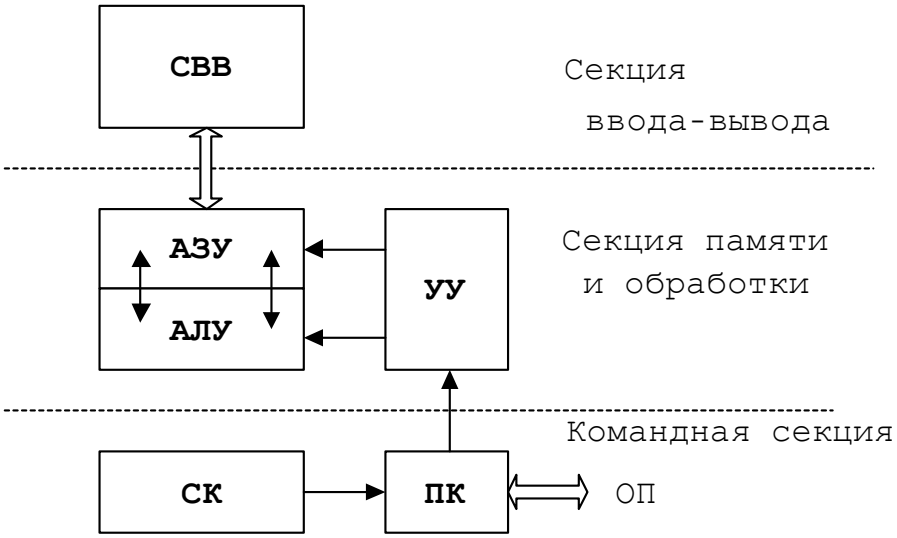


Рис.7.16. Структурная схема АС

СВВ – система ввода-вывода
АЗУ – ассоциативное запоминающее устройство
АЛУ – арифметико-логическое устройство
УУ – устройство управления
СК – счетчик команд
ПК – процессор команд
1-й блок - секция ввода-вывода, средняя секция - секцией памяти и обработки и последняя – командная секция. Эта структурная схема показывает нам, что обработка данных осуществляется в месте их хранения. В данном случае как-то разделена область для хранения данных и устройство для обработки данных, но нужно представить следующим образом: каждая ячейка (или группа ячеек) содержит некий обрабатывающий элемент – АЛУ. Понятно, что мы должны управлять этим процессом обработки и для этого служит УУ, которое управляет как ассоциативными ЗУ так и теми элементами, которые выполняют обработку данных. Понятно, что данные перед обработкой должны быть как-то поданы или записаны в ассоциативную память, для этого служит СВВ. Так как у нас бывают различные программы обработки, т.е. ПК, который управляет выборкой команд из ОП, понятно, чтобы знать какую команду выбирать текущей – должен быть СК. Т.е. обработка данных и обработка потока команд разнесены.

Данные из памяти выбираются по содержимому или части содержимого, а не по адресу, т.е. не используется линейная организация памяти для хранения данных.

Операция обработки осуществляется одновременно над несколькими элементами данных под управлением одной команды. Т.е. ассоциативные системы относятся к классу по Клину: одиночный поток команд и множественный поток данных. Особенность заключается в следующем: выборка идет не по адресу, а по содержимому этих данных.

Система команд должна предусматривать выборку значений указанного диапазона.

Свойства АС:

Данные из памяти выбираются по содержимому или части содержимого, т.е. отсутствует адресация ячеек памяти.

Операции обработки осуществляются одновременно над несколькими элементами данных под управлением одной команды.

Наиболее часто реализуемыми операциями является сравнение, максимум (max), минимум (min), попадание в интервал или между с заданием маскированных разрядов.

Замечание: Высокая стоимость ячейки памяти приводит к тому, что они редко являются полностью параллельными. Более распространены поразрядно-последовательные системы, в которых одновременно обрабатывается битовый или поразрядный срез, но разные битовые срезы обрабатываются последовательно.

Битовый срез – это совокупность разрядов всех ячеек АП.

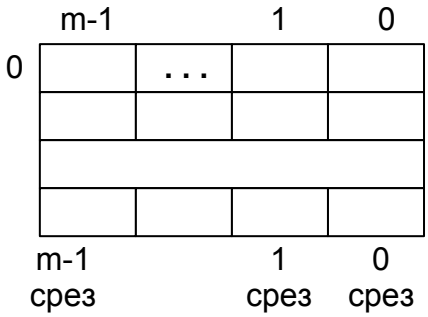


Рис.7.17. Битовые срезы

Поразрядно-последовательные системы

Так как каждое слово, которое может быть записано в ячейку АЗУ может быть разделено на разряды. Число разрядов, предположим, ограничено и равно m. М/б, есть смысл не обрабатывать сразу параллельно все m разрядов для ассоциативного поиска, а этот процесс разделить на m частей, т.е. обрабатывать разряды последовательно, но параллельно по всем словам. Это означает, что нам требуется не АЛУ m-разрядное, а битовый ПЭ. Ведь когда организуется операция сложения, то осуществляется поразрядное сложение с учетом переносов. Когда мы обрабатываем поразрядно каждое слово, то такая ассоциативная система называется **поразрядно-последовательной**.

Рассмотрим **функциональную схему поразрядно-ассоциативной системы:**

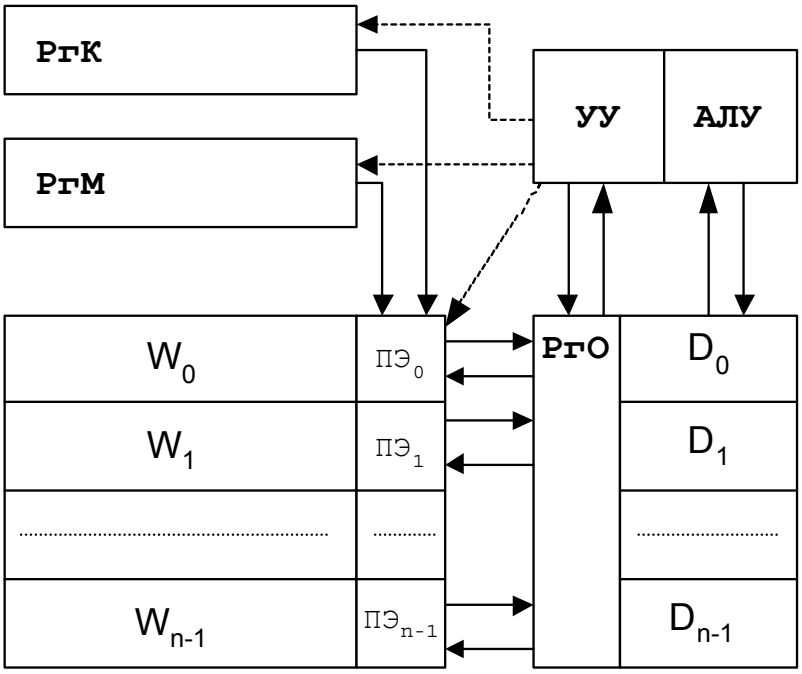


Рис.7.18. Функциональная схема АС

РгО – регистр отклика, в котором хранятся результаты битового среза.
РгК – компаранда.

Описание работы схемы:

Схема управления позволяет сдвигать слово, что обеспечивает параллельную передачу битового среза в массив ПЭ. Одновременно с передачей в ПЭ битового среза передаются одноименные разряды регистра компаранда и маски.

Каждый ПЭ формирует частичный результат – бит отклика, который используется для следующей операции в ПЭ. Здесь возможность использовать это для следующей операции после всех n сдвигов или же после всех операций, когда пришли данные очередного битового среза.

Регистр маски позволяет исключать некоторые разряды слов из обработки.

В РгО формируется также результат ассоциативной обработки всего массива, который, например, может использоваться для маскирования некоторых слов ассоциативной памяти при выполнении следующей команды УУ.

РгК содержит данные о ассоциативной обработке. Например число с которым сравнивается каждая ячейка.

Анализ всех n слов будет завершен для m тактов и не зависит от ячеек ОЗУ. Обработка битовых срезов дает значительный выигрыш в быстродействии при n >> m.

Ассоциативный процессор реализует функционально-полную систему операций над многозначными данными. Можно увидеть, что за m тактов выполняется сразу n операций.

Ортогональные системы

В ассоциативных системах формат слова (данных) имеет два поля: поле данных и поле тега. В поле тега находятся управляющие данные (тип, индекс и т.д.). Ассоциативной обработке подвергается только поле тега. Поле данных обрабатывается традиционным способом. Поиск требуемого слова (слов) осуществляется путем обработки битовых срезов поля тегов. Возможен множественный отклик, когда в регистре отклика несколько разрядов содержат требуемые значения. УУ выбирает данные из поля данных и передает их в АЛУ. Понятно, что в этом случае используется состояние регистра отклика.
Запись данных в АЗУ производится ассоциативно, т.е. безадресно. Это можно осуществить следующим образом: если регистр компаранд совпал с содержимым ячейки, то эту ячейку можно назвать свободной или выделить какой-то разряд, что эта ячейка не занята.
1 из разрядов тега используется для указания занятости ячейки. Сначало читается битовый срез занятости, а потом осущесствляется запись, как в поле тега так и в поле данных. В этом случае система ввода-вывода(СВВ) имеет возможность адресного обращения к АЗУ.
Ассоциативные системы используются:

- в кэш-памяти;
- в памяти для обработки таблиц (машины баз данных);
- в организации виртуальной памяти.

Пример АС: GoodYear Aerospace 70гг.

АЗУ имеет организацию 256 слов, а каждое слово имеет 256 бит.

ПЭ – 256, которые выполняют арифметико-логические операции.

Система передач между АЗУ и ПЭ позволяет перемещать слова между ячейками, что позволяет производить перегруппировку слов.

УУ работает по командам длиной 32 бита, хранящихся в специальной памяти объемом 32К слов.

Цикл памяти 200нс, т.е. одна команда выполняется за 200нс.
Общее управление ассоциативной системы осуществлялось ЭВМ PDP-11.

Тема 7.4. Систолические и волновые системы (АС)

Систолическая система(СС) – это числовой обработки данных.
Особенностью является то, что она занимается проблемой хранения промежуточных данных. Проблема, решаемая в рамках СС, заключается в устранении или отсутствии дополнительных обращений к ОП для хранения промежуточных результатов при вычислении, тем самым устраняется традиционные недостатки конвейерной и матричной обработки. Используются как распараллеливание, так и конвейеризация – одновременно.

Принцин систолической обработки

Он заключается в выполнении всех стадий обработки каждого элемента данных, извлеченных из памяти, прежде чем результат этой обработки поместить обратно в память.
Этот принцип реализуется систолической матрицей или массивом ПЭ.
В СС данные извлекаются из ОП и последовательно передаются между ПЭ, которые выполняют их обработку, и последний ПЭ сохраняет в ОП окончательный результат.

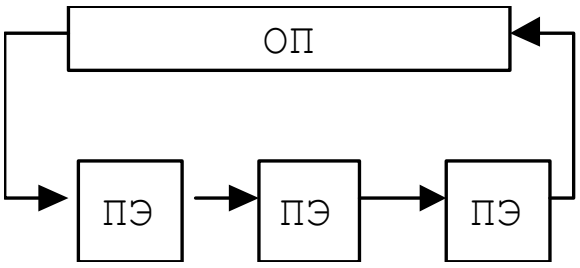


Рис.7.19. Линейная СС

Данные передаются в общем случае произвольно.

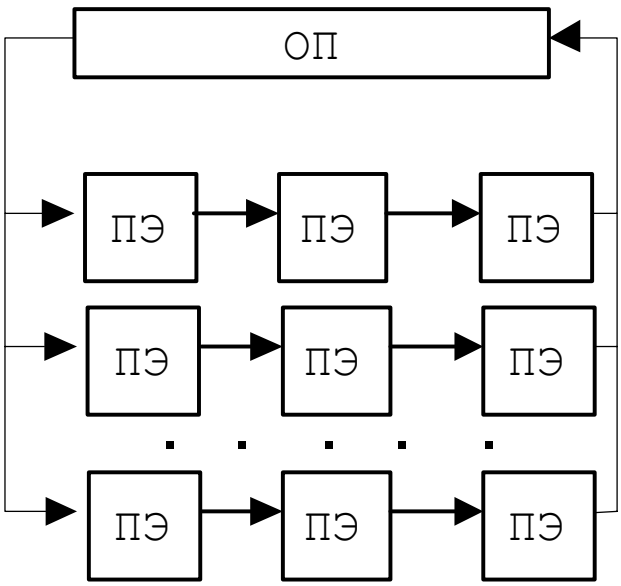


Рис.7.20. Объемная СС

По этим параллельным конвейерам как бы «прокачиваются» операнды, т.е. каждый элемент данных извлекается из ОП и ритмически продвигается по матрице ПЭ-тов, прежде чем опять попадет в память.
СС является синхронной системой, т.е. потоки данных синхронизованы единой для всех ПЭ-тов системой тактовых сигналов.

Преимущество систолической обработки:

Минимизация обращений к ОП:

Это позволяет согласовать скорость работы ОП со скоростью обработки данных, что в известных системах – несогласованно.

- Облегчается решение проблемы В/В.
- СС легко реализовывается по технологии СБИС.
- Минимизации межсоединений:

70% затрат на изготовление микросхемы идет на транспортировку данных. Ближлежайшие ПЭ связаны между собой.

Проблема использования СС:

Для каждой задачи требуется нахождение своей систолической матрицы. Для реализации преимущества систолической обработки необходимо найти для каждой задачи (класса задач) соответствующие систолические алгоритмы и отобразить их на систолическую структуру.

Примеры:

Систолические алгоритмы найдены для широкого спектра задач числовой обработки, обработки сигналов, символьной обработки.

b) умножение и обращение матриц;
c) решение систем линейных уравнений;
d) дискретное преобразование Фурье;
e) кодирование и декодирование числовых последовательностей.

Систолические алгоритмы

Систолические алгоритмы:

Символьная обработка: поиск вхождения подстроки в строку.

Предположим у нас имеется две строки:

$A = a_1 a_2 ... a_n$,

$B = b_1 b_2 ... b_n, n \geq m.$

Ставится задача поиска вхождения строки B в строку A. Оказывается этих точек вхождения может быть не одна. Упростим задачу и будем считать, что b_i и a_i это двоичные цифры, т.е. $a_i \in \{0,1\}$, $b_i \in \{0,1,?\}$. Рез-тат мы должны получить в следующем виде:

$$r_i = \begin{cases} 0, c_i несопоставлена \\ 1, c_i сопоставлена \end{cases} \quad i = \overline{0, n-m}$$

Эта задача решается с помощью линейной систолической структуры:

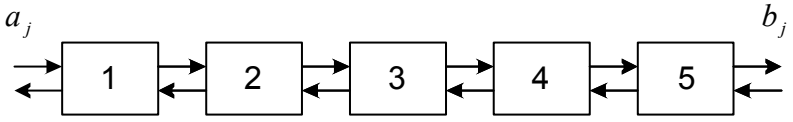


Рис.7.21.

Всего пять ПЭ. И на эту систолическую структуру мы будем подавать данные в разных направлениях. Приведем временную диаграмму функционирования этой системы.

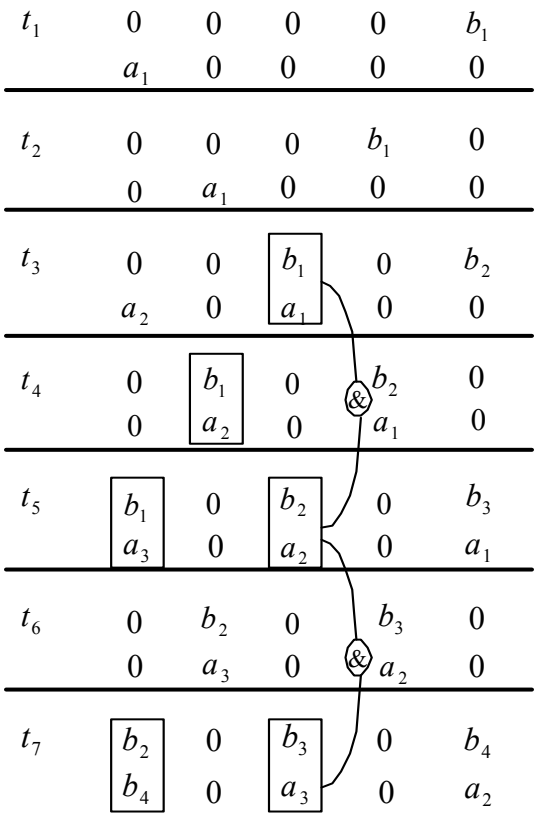


Рис.7.22. Временная диаграмма функционирования системы

В первый момент времени считаем, что никаких данных не содержится, т.е. поступает a_1, далее ничего не содержится, а в конце b_1.

$r_j = \& a_s b_t$

(s,t)

В результате получаем значение множества r. За $4n$-тактов ищется вхождение подстроки в строку.

- **Умножение квадратных матриц:**

Заданы две матрицы A и B. Пусть известно, что надо найти матрицу C, равную произведению матриц A и B, и известен элемент матрицы:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad i,j = \overline{1,n}$$

Запишем рекуррентное выражение:

$c_{ij}(0) = 0$,

$c_{ij}(k) = c_{ij}(k-1) + a_{in} b_{kj}$.

Как только у нас появилось рекуррентное выражение или система рекуррентных выражений такого типа, то мы можем представить это вычисление в виде систолической структуры, где $k = \overline{1,n}$. Рисуем структуру ПЭ:

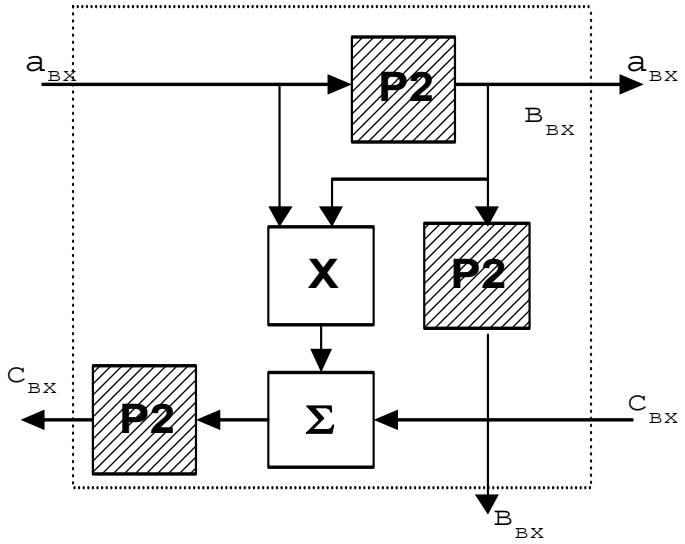


Рис.7.23.

Всего будет три регистра, умножитель и сумматор.

Видно, что отразили рекуррентное выражение в структуру ПЭ. А т.к. это конвейерное устройство, то каждое поступаемое данное подаем для временного хранения в буферный регистр.

А теперь хотим организовать так, чтобы каждая итерация реализовывалась своим ПЭ: сколько итераций – столько ПЭ. Нарисуем процессорную матрицу для случая n=2:

$$A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}.$$

Изображаем четыре ПЭ: c_{11}, c_{12}, c_{21}, c_{22}.

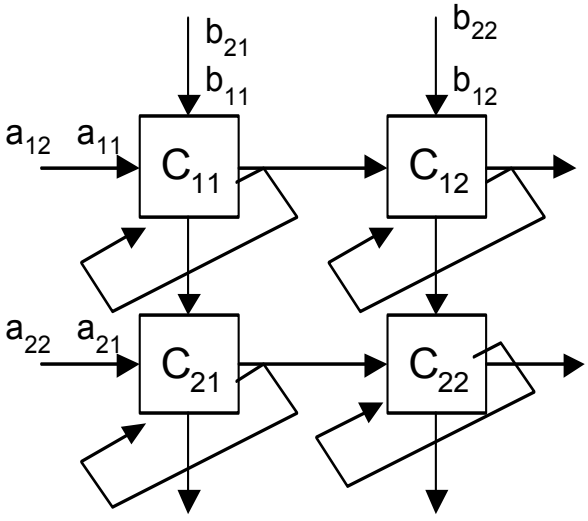


Рис.7.24.

Для того чтобы эта систолическая структура смогла правильно вычислить произведение мы должны подавать элементы на вход в нужное время и в нужной последовательности:

a_{11} - b_{21}, $\alpha_{12} - b_{22}$.

В результате получили следующую систолическую структуру: подавая нужные данные в нужное время, мы как бы вычисляем в каждом ПЭ элемент результирующего произведения.

К перемножению м-ц nxn на систолической структуре требует n^2 тактов работы. На последовательной ЭВМ требуется выполнить n^3 команд.

Для представления систолических структур используется описание ПЭ и граф систолической матрицы и структуры. $Y = AX$ $X = A^{-1}Y$

Волновые системы (ВС)

Систолические структуры – это системы синхронной обработки данных. Для них используется единая система тактовых сигналов, т.е. пульсация и продвижение данных происходит согласованно. Если на один элемент поступила порция данных, то только следующий такт меняет одну входную порцию на другую, причем это происходит по всей матрице одновременно. Но иногда, когда мы будем использовать универсальные систолические системы с командным управлением, может оказаться, что сейчас подали команду сдвига, на другую матрицу подали команду сложения, а на третью команду умножения. Время выполнения этих команд различное. Мы должны выполнить времена выполнения. ВС, в отличие от традиционных систолических конвейеров отличается тем, что у них идет асинхронное взаимодействие рядом стоящих элементов. Т.е. этот элемент выполняет умножение, сколько ему надо, а только потом сообщает, что данные готовы. И поэтому, если мы находим ту же самую задачу, например, умножение матрицы на волновой системе, то вполне возможно, что умножение на 2 занимает меньше времени, чем на, предположим, 511. Поэтому в этом случае можно представить себе при асинхронном взаимодействии эту матрицу, и для умножения, как некую систему, по которой одноименные данные прокачиваются. И если мы зафиксируем какие-нибудь данные, то мы увидим какой-нибудь промах. Так вот в синхронных системах фронты представляют собой прямые линии, а в волновых системах это может быть некая изогнутая кривая, причем она с течением времени форму свою меняет, в зависимости от тех команд, что выполняются и сколько времени они выполняются.

Волновые системы (ВС) – это асинхронные системы обработки данных, основой построения которой служит регулярная коммутационная сеть (жесткие связи между ПЭ), обеспечивающая локальные связи между соседними ПЭ.

Механизм кретирования. Т.к. есть асинхронное взаимодействие, то между ПЭ идет некий диалог и каждое данное, если оно получено кретируется. Механизм кретирования состоит в следующем: обеспечивает асинхронное взаимодействие связанных ПЭ по обмену данными.

Волны, фронт которых образуется одновременно введенными исходными данными, передвигается по волновой структуре и обеспечивают синхронизацию выполнения последовательности операций.

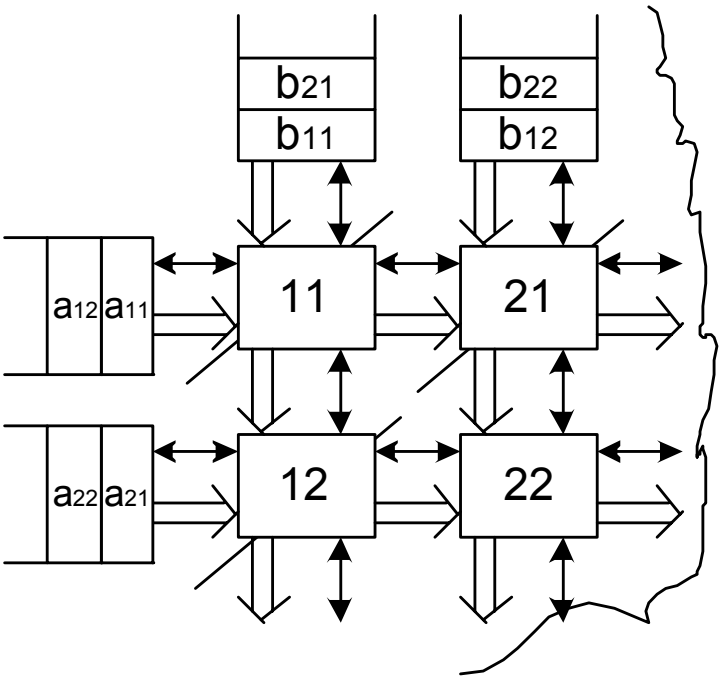


Рис.7.25.

Преимущества волновых систем:

Только здесь можно использовать сложные ПЭ, которые требуют различного времени выполнения для различных команд.

- Высокая отказоустойчивость и достоверность. А потому что, если хотя бы один элемент не работает, то рез-тат невозможно получить.
- Более низкое быстродействие, потому что механизм кветирования занимает определенное время.

Тема 7.5. Мультипроцессорные комплексы

МПВК (мультипроцессор) – это комплекс, состоящий из нескольких процессоров, работающих под управлением единой ОС, организующей весь процесс обработки данных в комплексе.

Классификация МПВК

МПВК могут быть разделены на 3 типа:

1). С общей памятью
2). С распределенной памятью
3). С динамической структурой

Структура МПВК с общей памятью:

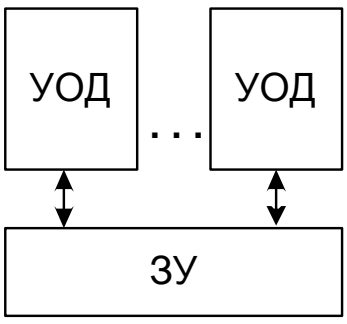


Рис.7.26.

Структура МПВК с распределенной памятью:

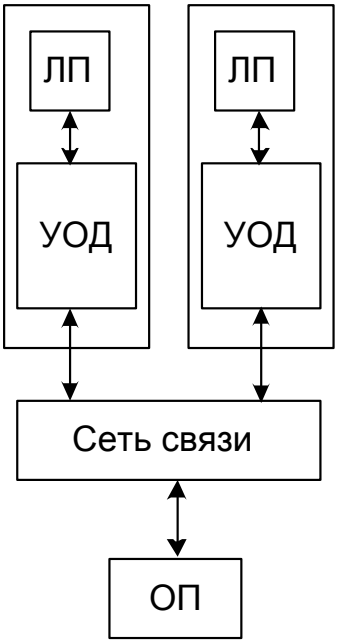


Рис.7.27.

Если в системе объем общей памяти больше чем локальной, то такая система относится к МПВК с общей памятью, а если меньше, то это МПВК с распределенной памятью.

Мультипроцессорный комплекс с общей памятью

Типы структурной организации МПВК с общей памятью:

4). С общей шиной;
5). С перекрестной коммутацией;
6). С многовходовыми ОЗУ.

С общей шиной (системным интерфейсом):

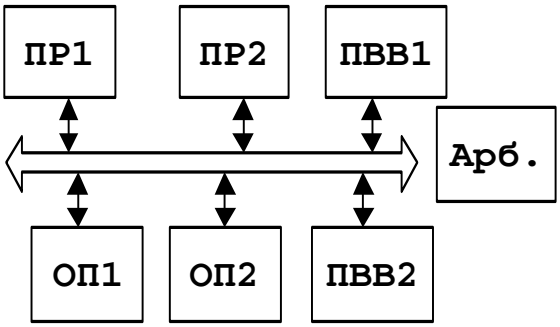


Рис.7.28.

ПР1 и ПР2 – это процессоры вычислительного назначения.
ОП1 и ОП2 – это блоки основной памяти.
ПВВ – это процессоры ввода/вывода.

Имеется один СИ, через который объединяются все устройства системы, в том числе, может быть и трехуровневый СИ. На интерфейсе PCI нет понятия процессор и ОП, а есть активные и пассивные устройства. Больше двух процессоров к этой схеме не подключается, поэтому у нас идет расслоение этого общего интерфейса на 3 уровня, потому что не только есть процессоры активные в современном ПК, но еще есть и другие активные устройства, которые захватывают СИ.

Достоинства:

Простота реализации

Низкая стоимость

Простота наращивания.

Недостатки:

Невысокое быстродействие

Низкая надежность объясняется тем, что в МП системе есть много блоков памяти и много процессоров, но есть единая общая среда передачи данных, которая может выйти из строя, - и все процессоры остановятся.

МПВК с перекрестной коммутацией:

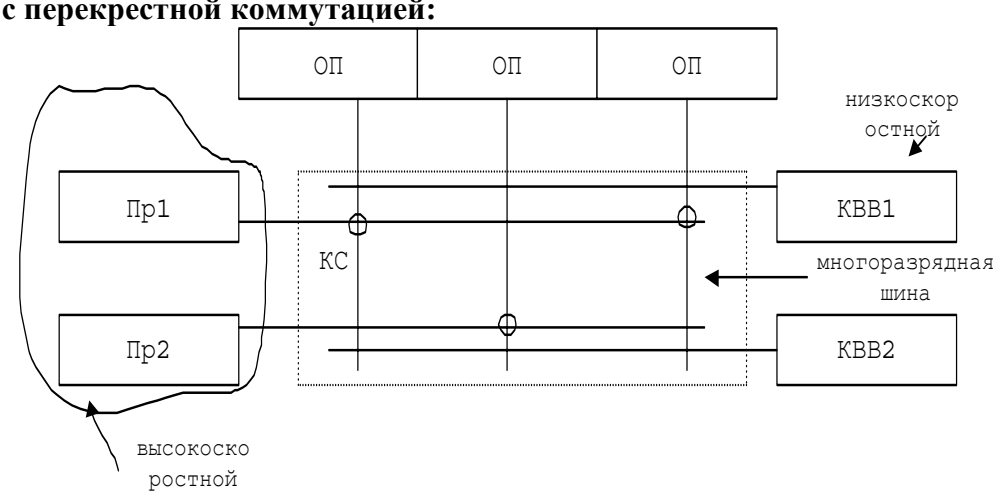


Рис.7.29.

КС – коммутационная среда
КВВ – канал ввода-вывода
Пр – универсальный процессор

Одновременный доступ 3 устройств. Многоразрядная шина.

Дорогостоящий и ложный коммутатор

Невысокая надежность

Необходимость средств разрешения конфликтов(аппаратных средств)

Для решения этих проблем коммутатор делают многоступенчатым или используют два коммутатора – высокоскоростной (для быстродействующих устройств) и низкоскоростной (для медленно действующих устройств).

С многовходовыми ОЗУ:

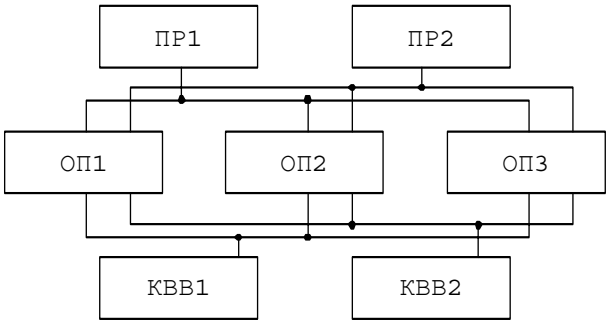


Рис.7.30.

Достоинство: упрощена логическая организация ВС, путем локального разрешения конфликтов доступа к ОП (каждая ОП имеет внутри, как минимум 4 блока (независимых).
Недостаток: необходимость коммутации внутри ЗУ-в, т.к. одновременный доступ к одной ячейке нескольких процессоров – невозможен.

Мультипроцессор с распределенной памятью

Разделяются на:

Многопроцессорные ВС(тип 1) – многомашинные;

Многопроцессорные ВС(тип 2) – многопроцессорные.

МПВК (тип 1) – историчесткий экскурс:

f) С общим ВЗУ;
g) С общей основной памятью;
h) С каналами передачи данных.

Типы многомашинных ВК:

Слабосвязанные - глобальные компьютерные сети;

Прямосвязанные - локальные(корпоративные) сети (используются высокоскоростные каналы передачи данных) или одноранговые сети;

Сателитные – локальные или глобальные сети, в узлах которого находятся неодинаковые устройства(клиент-сервер или сервер-приложений).

Многомашинный ВК – комплекс, включающий в себя 2 или более ЭВМ, каждая из которых работает под управлением собственной ОС и связи между которыми обеспечивают выполнение функции, возложенных на комплекс в целом (повышение производительности, повышение надежности). (Крупно-блочный уровень).

Микролокальные вычислительные сети. (Средне-блочный уровень).

Симметричные и несимметричные МПВК.

Симметричная обработка данных – распределение задач (процессов) по процессорам независимо от типа процессора и типа процессов.

При несимметричной обработке выделяется класс задач процессов, которые привязываются (распределяются) на исполнение на заданных процессорах.

Микролокальные сети:

Это ММВК – состоящий из элементарных вычислительных машин, объединенных в единую систему (сеть) по средствам коммутационных средств (КС).

Элементарная машина, как правило, состоит из процессора, локальной памяти, средства ввода/вывода и которая может независимо от других элементарных машин исполнять программу в собственной локальной памяти. Взаимодействие процессов в микролокальной сети осуществляется путем обмена сообщений по коммутационной сети.

В качестве элементарных машин используются модули, построенные на базе процессоров с архитектурой RISC, CISC.

Пример:(кластерные арх-ры, ЭВМ с массовым параллелизмом, транспьютеры).

ЭВМ с массовым параллелизмом(транспьютерные с-мы).

Транспьютер (Transmission Computer) – Микро ЭВМ.

Фирма Inmos T-800 – полная однокристальная микро-ЭВМ: 30МГц, 15Mips, 3.3Mflops.

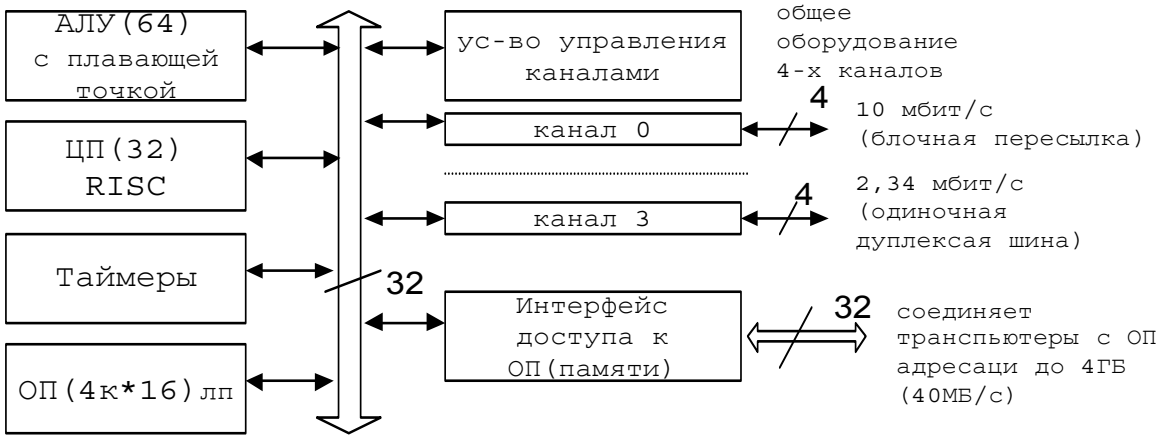


Рис.7.31. Структурная схема транспьютера фирмы

Система команд включает в себя 16 однобайтовых к-нд.

0000	Загрузка локальная
0001	Загрузка указателя локальная
0010	Загрузка нелокальная
0011	Загрузка указателя нелокальная
0100	Запись локальная
0101	Запись
0110	Запись
0111	Запись
1000	Загрузка константы
1001	Запись константы
1010	Сложение с константой
1011	Сложение с памятью
1100	Безусловный переход
1101	Условный переход
1110	Вызов подпрограммы
1111	Настройка рабочей области
....	Префикс расширения

Регистры: A,B,C(могут образовать министы)
PC – указатель текущей к-нды
PM – указатель рабочей области

Длину любого операнда можно увеличить за счет команд префисикации. Префиксная команда загружает свои 4 разряда в регистр данных, и сдвигает содержимое этого регистра (используя 3 префиксные команды увеличиваем разрядность до 16 бит).

Безразрядные команды содержат в поле данных код управления.

Последовательные каналы имеют по 2 однонаправленные линии связи, по которым передаются данные и подтверждение о приеме данных. Каждая линия позволяет осуществлять блочную пересылку в режиме «память - память», как для локальной , так и для дополнительной памяти. Функционирует каждый канал полностью асинхронно и независимо от ЦП.

В состав T800 входят дополнительные вспомогательные БИС (адаптеры последовательного канала, коммутаторы, сигнальные процессоры, контроллеры ПЗУ).

Рассмотрим **топологию коммуникационных сетей (КС)**:

Необходимо обеспечить передачу данных между процессами, исполняемых на каждом транспьютере.

Необходимо обеспечить синхронизацию этих процессов, т.е. реализовать заданную логику их порождения и запуска.

Так как интерфейс дополнительной памяти является разделяемым между транспьютерами, то это является слабым звеном такого рода систем.

Возможна как статическая, так и динамическая надстройка. В природе существует только динамическая надстройка.

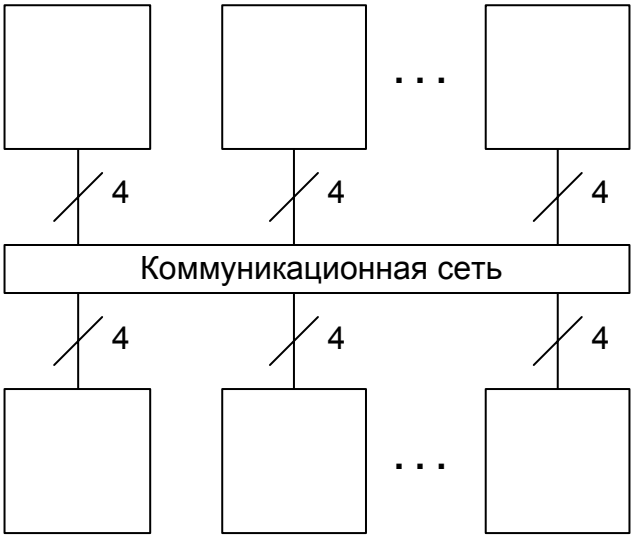


Рис.7.32.

Типы топологий КС:

n – число уровней в сети;

N – число элементарных машин в сети;

d – коммуникационный диаметр.

Кольцевая:

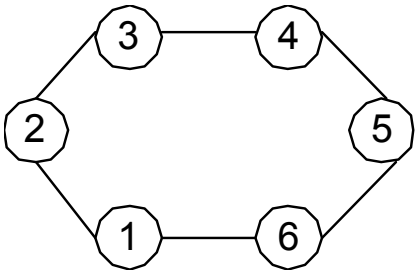


Рис.7.34.

Транспьютеры соединяются только с двумя соседними.

Пересылка сообщений из-за отсутствия прямых связей осуществляется через промежуточные элементарные машины – ретрансляторы.

Коммуникационный диаметр – это расстояние между наиболее удаленными элементарными машинами в КС. Коммуникационный диаметр:

$$d = \frac{N}{2}.$$

- Двумерная решетка:

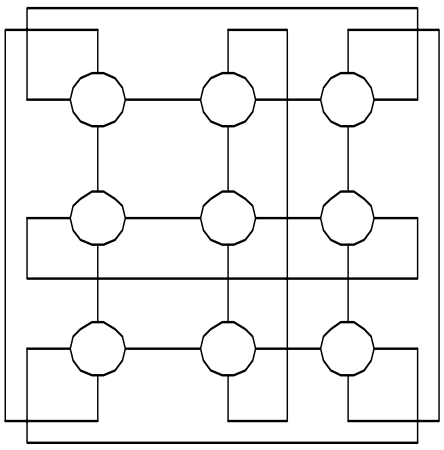


Рис.7.35.

Каждая элементарная машина связана с четырьмя соседними.
Коммуникационный диаметр:

$d = 2\left(\sqrt{N} - 1\right)$-для незамкнутой.

- Древовидная:

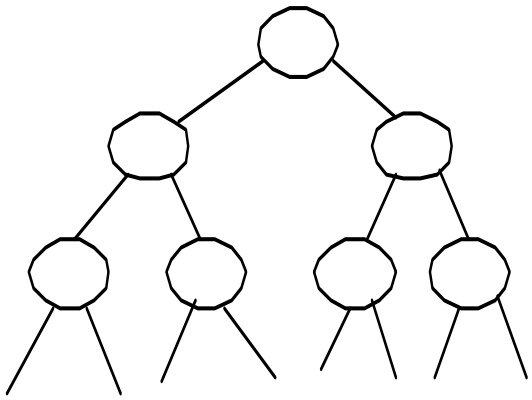


Рис.7.36

$d = 2(N - 1)$

$N = 3^{n-1}$.

- Гиперкубическая:

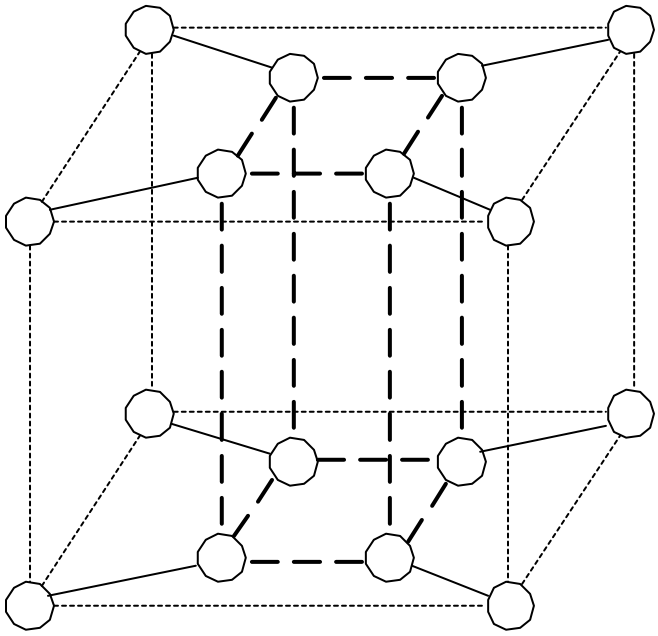


Рис.7.37.

$d = 4$

$N = 16$

Рассмотрим ***перестраиваемую топологию:***

Для каждой задачи может быть выбрана своя топология, при реализации которой можно получить наибольшую эффективность вычислений.

Ставится задача создания систем обработки данных с перестраиваемой топологией КС.
Существует два типа систем с перестраиваемой топологией:

Транспьютерные сети (кластерные системы).

Системы обработки с массовым параллелизмом.

Рассмотрим структуру обработки данных с массовым параллелизмом:

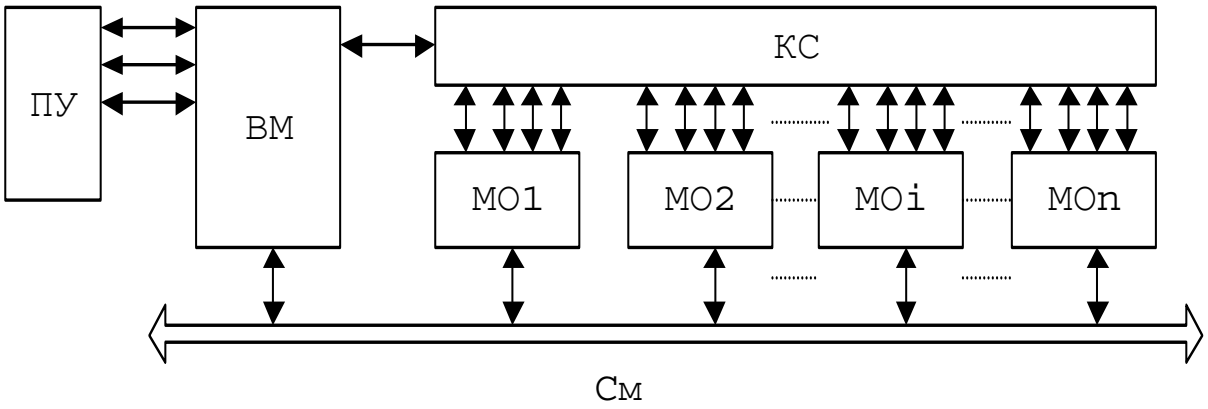


Рис.7.37. Система обработки с массовым параллелизмом

ВМ – ведущий модуль
МО – модуль обработки
ПУ – переферийные устр-ва
СМ – супервизорная магистраль
С-ма GC(Gega Claster)Parsejtec
Т9000 от 64 – до 16384.

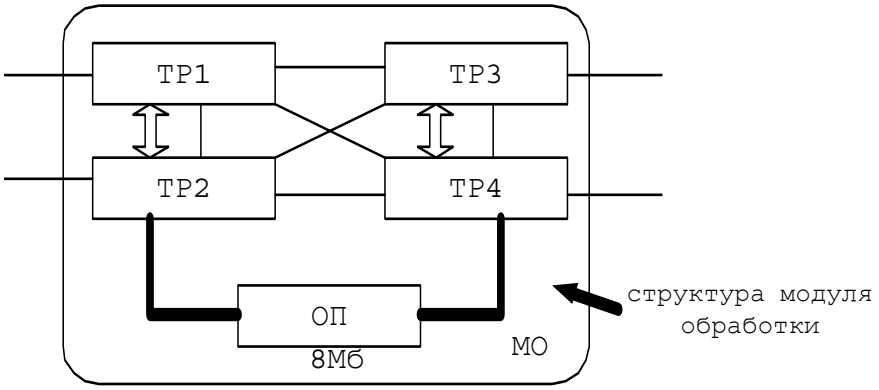


Рис.7.38. Meiko CS(Computing Surface)

Кластерные системы строятся на базе кластера, т.е. модуля следующего уровня по отношению к транспьютеру (элементарной машине).
Рассмотрим структуру типового кластера
Структура кластера:
С104 – коммутатор
Т9000 – транспьютер

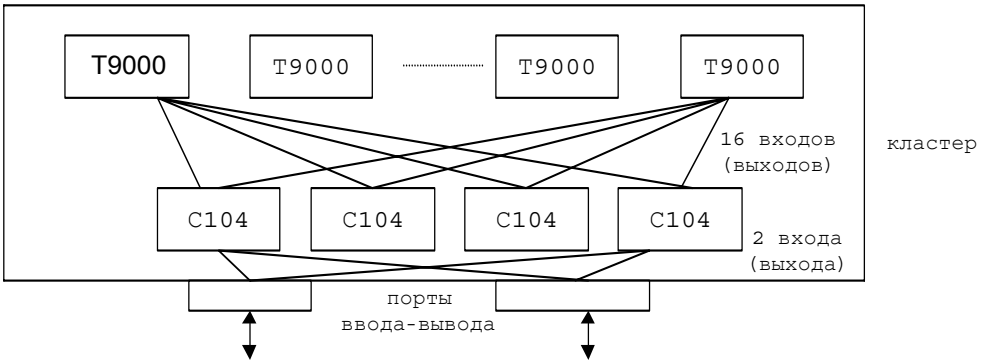


Рис.7.39. Транспьютер T9000

Имеется четыре специальных модуля (коммутатора), каждый из них имеет четыре входа (выхода). Образуем четыре пункта объединения. Кластеры объединяются в поверхности. Четыре таких кластера образуют Гигакуб.

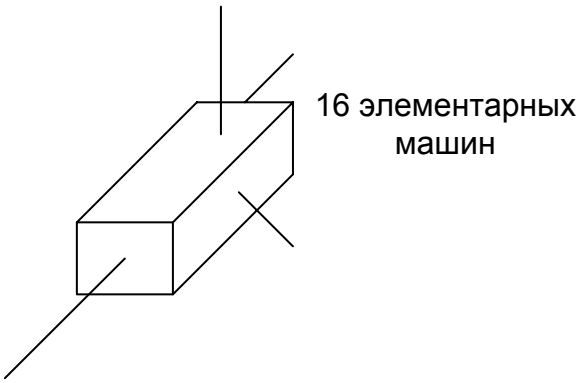


Рис.7.40. Гигакуб

Тема 7.6. Обзор языка OCCAM

Парадигма программирования

Это набор параллельных процессов, взаимодействующих через программные каналы путем **столкновения** в канале.
Явление столкновения является одновременно механизмом синхронизации процессов: Один процесс выводит в канал, а другой выводит из канала. Вывод в канал при отсутствии данных в канале приводит к остановке читающего процесса.

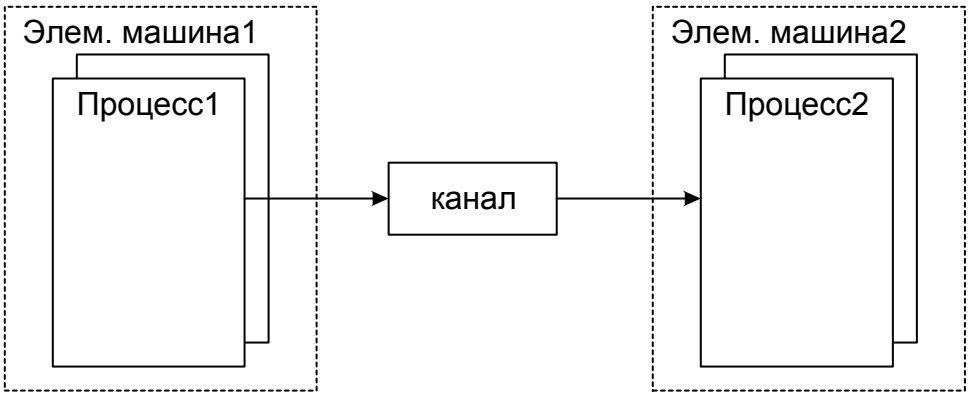


Рис.7.41.

Каждый процесс существует на отдельной элементарной машине. На одной элементарной машине может быть несколько процессов.
Функции по организации канала берет на себя распределенная (сетевая) ОС, которая является достаточно сложной.

Типы данных языка OCCAM

OCCAM – является языком строгой типизации, т.е. отсутствует неявное преобразование типов. **Пример:** язык программирования Паскаль.

7). BOOL (логический) – двоичный тип.
8). BYTE – байтовый тип.
9). INT16, INT32, INT64 – целый тип.
10).REAL16, REAL 32, REAL 64 – тип с плавающей запятой.

[< размерность >] <тип> <идентификатор>
Пример:
[16]BOOL x – x является массивом бит длиной 16.
Каждый из базовых типов имеет ключевое слово CHAN OF (модификатор).
Пример:
CHAN OF INT ch // канал для передачи целых данных

Операторы присваивания и выражения

Оператор присваивания состоит из трех частей:
< леводопуст >: = < выражение >
где леводопуст – это выражение, которое ссылается на ячейку памяти. Самым простым является идентификатор переменой:
x [0]: = 0.
Выражение представляет собой переменные или константы, разделенные знаками арифметических операций (унарные или бинарные: +, -, *, /, (<тип>)).
Пример:
x[1] : = (BOOL) 32.3
Существует две специальные операции, предназначенные для работы с каналами:

i) ! – запись в канал (вывод в канал):
Пример:
C[3]!15 – в канал C3 записывается число 15

- ? – чтение из канала
Пример:
C[3]!x – чтение из канала C3 и результат присваивается переменной x

Параллельные и последовательные процессы

SEQ – означает секцию последовательных процессов.
PAR – означает секцию параллельных процессов.
Пример:

```
SEQ
    x: = 0
    y: = 1 – означает последовательное выполнение процесса
    z: = 2 – выход из секции последовательных процессов

PAR
    a: = 1
    b: = 2
```

Здесь неизвестно какое присваивание будет первым(может одновременно, может с опозданием), но известно, что тело этого цикла закончится, когда выполнится 2 присвоения.
Пример: драйвер оператора
ALT – данная конструкция используется только при работе канала
Пример:
ALT
WHILE TRUE (скобок никаких нет, но то что имеет отношение к телу WHILE имеет отступ 2 пробела от 1-го знака слова WHILE)
ALT // альтернативное выполнение процессов
left.chan?x // в левый канал вводим число x
output!x // тело процесса left.chan
right.chan?x //
output!x
Эта конструкция непрерывного объединения 2-х потоков данных.
Есть некий процесс alt:

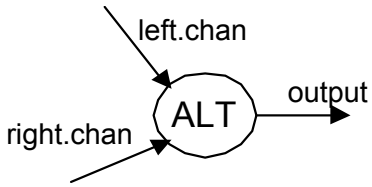


Рис.7.42.

ALT конструкция означает, что войдя в тело этого процесса не ожидает готовности данных, а проверить: есть ли готовность данных – выполняется этот процесс, если нет готовности, то выполняется второй процесс.
Секция или конструкция ALT позволяет выполнять несколько альтернативных процессов в зависимости от готовности того или иного канала.
Пример:
Имеется три процесса функции:
edit(in, out)
kb(in) - процесс ввода данных с клавиатуры
screen(out) - процесс вывода данных на экран
Потребуется три канала (байтовые):
CHAN OF BYTE key // канал, откуда поступают нажатия от клавиш клавиатуры
CHAN OF BYTE scr // канал вывода на экран
CHAN OF BYTE in, out // каналы целых чисел
Нарисуем схему, которую хотим реализовать. Т.е. у нас имеется драйвер клавиатуры, драйвер экрана, как некие процессы и имеется редактор, к-рый вводит с клавиатуры и выводит на экран.
Схема, которая позволит организовать диалог с оператором.

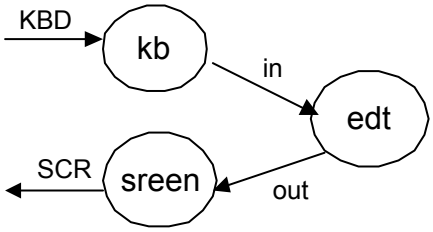


Рис.7.43.

Код этой программы следующий: параллельно запускается процесс редактора. Ему в качестве параметра передаются 2 канала: входной и выходной. Далее запускается драйвер клавиатуры, которому передается канал от оборудования и входной канал. И запускается 3 канал, к-рый получает рез-тат обработки данных редактором. Эти процессы могут тоже, в свою очередь, порождать процессы: например редактор может порождать процесс, занимающийся формированием какого-нибудь окна.

```
PAR
  edt(in,out)
  kb(in)
  screen(out)
```

Организация конвейерных вычислений

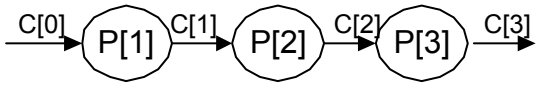


Рис.7.44.

Конвейер будет состоять из 3 стадий, т.е. будет 3 независимых процесса, которые будут распределены на разные процессоры в этой многопроцессорной сети. Данные подаем на один процесс, он их вычисляет, затем результат подаем на 2 процесс, она этот результат должен опять преобразовать и свой результат передать 3-му процессу. Каналы организуем в виде массива.

Рассмотрим программу, которая реализует эту конвейерную обработку. Время на последовательную обработку (есть один процесс и он будет последовательно выполнять эту работу) будет гораздо больше, чем мы организуем постоянный поток данных на эту схему, то комплексная эффективность будет больше 1, потому что эти процессы существуют параллельно и независимо, т.е. утверждается, что комплексная эффективность этого кода >1 (реализуется конвейерное вычисление).

```
[4]CHAN OF INT C
  PAR
    P[1](C[0], C[1])
    P[2](C[1], C[2])
    P[3](C[2], C[3])
```

Систолические вычисления

Пример:

```
 [3][2]CHAN OF INT X
 [2][3]CHAN OF INT Y
    PAR i=0 FOR 2
      PAR j=0 FOR 2
        P[i,j](X[i,j,Y[i,j],
X[i+1,j],Y[i,j+1])
```

Автоматически осуществляется синхронизация передачи данных, как в волновой системе. Возможно, что каждый процесс имеет еще один канал куда передается окончательный рез-тат для формирования итогового массива.

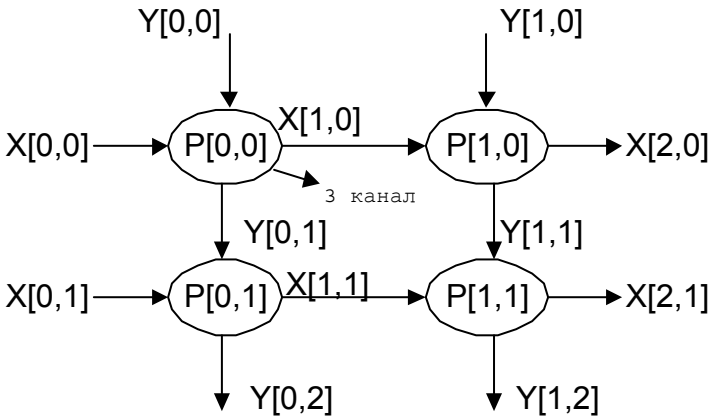


Рис.7.45.

Мультипроцессоры с динамической архитектурой

До сих пор мы изучали подходы, к-рые обеспечивали нам способ решения возникающих перед нами задач, причем заведомо мы предполагали что задачи универсальны, т.е. заведомо не ограничивая круг решаемых задач. Но оказывается встречаются задачи, к-рые моделируют с-мы, стр-ра к-рых меняется по ходу решения этой задачи или же по ходу ф-ционирования этого моделируемого объекта. Например задача клетки жизни.

Проблема заключается в том, что мы исполняем заведомо параллельно ф-ционирующий механизм или модель на последовательные данные.

Хотелось бы создать некую арх-ру, которая позволяла бы образовывать параллельно взаимодействующие ф-ционирующие объекты, т.е. некую предметную область стр-ра и связи м/у к-рыми изменяются.

Пример: задача «клетки жизни»

a) Жизнь – это многоклеточное сообщество.

b) Колония клеток – часть n-мерного пространства, разделенная на ячейки в к-рых может находиться 1 клетка.

c) Клетка – конечный автомат(группы состояний клетки: рождение, существование, размножение и смерть)

d) Мера времени – смена поколений

e) Смена поколений:

– Соседи – это клетки расположенные на расстоянии ρ в n-мерном пространстве

– Условия смерти , если клетка имеет менее S соседей, то погибает от одиночества, а если более чем t соседей, то от перенаселения

– Условия рождения, если рядом с пустой ячейкой на расстоянии r имеется k соседей.

– Рождение и смерть происходит в моменты смены поколений.

– Эпидемия(катаклизмы). Моделируются распределением как во времени так и в пространстве величин ρ, r, s, t.

В основу МДА положен подход, использующий представление задачи в виде стр-рного описания предметной области(семантические сети). При традиционном подходе основная форма представления вычислений – алгебраическое выражение, которая неэффективна для решения задач типа клеток жизни.

Область применения МДА:

– Задачи управления объектами с изменяемыми стр-рами. Если объект является стр-рно изменяемым, то и модель должна обладать такими же свойствами.

– Планирование, проектирование, моделирование, связанные со стр-рными преобразованиями в предметной области.

МДА непосредственно реализуют стр-рный м-д представления задачи без перехода к алг-мам в их традиционной форме.

Основа построения МДА.

Это автоматный способ орг-ции вычислений, основанный на теории динамических автоматных сетей.

Пр-мма МДА – это сеть, каждому узлу которой соответствует нек-рый автомат, способный не только изменять свое состояние и перерабатывать данные, но также изменять свои связи с другими узлами, порождать новые автоматы(узлы), уничтожать самого себя или узлы с к-рыми он связан. Заметьте, что с-м в окружающем нас мире не существует, мы просто их наблюдаем. Выполнение пр-ммы сводится к автотрансформации сети, входе выполнения которой исходная сеть растет(развивается) и в конечном итоге теряет способность к изменению, что соответствует окончанию решения задачи.

Как нам реализовать эти представления? В жизни реализовать динамическую автоматную сеть (ДАС) в чистом виде очень сложно, т.е. устр-во, к-рое явл. вычислительным одновременно является и устр-вом, которое производит технологически эти автоматы, более того уничтожает или утилизирует продукты их распада. В биологических с-мах такое наблюдается. Т.к. у нас получается некие отличные от биологических поэтому нам надо это вместить в рамки известных технологий. Как это сделать? Есть 2 подхода:

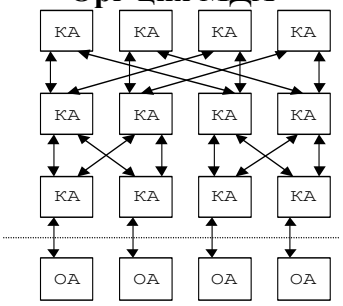
Упростить на одном уровне

Приведем этот уровень к почти традиционным с-мам с известыми ранее пр-ммами, процессами и т.д.

Виртуальная динамическая автоматная сеть(ВДАС).

Любой ДАС поставим в соответствие виртуальную машину, орг-ция к-рой динамически изменяется в соответствии с изменением стр-ры сети в ходе реш-ия зад.

Орг-ция МДА



Нам необходимо обеспечить связи «каждый с каждым» независимо от числа ранее установленных связей. Поэтому мы должны предусмотреть коммутационные устр-ва, к-рые должны обеспечить эти связи и назовем из коммутационными автоматами. Понятно, что обмен сообщениями будет двухсторонний. Связи могут устанавливаться м/у любой парой автоматов, существовать некоторое время и разрываться. Это мы рассмотрели задачу при ограниченных ресурсах.

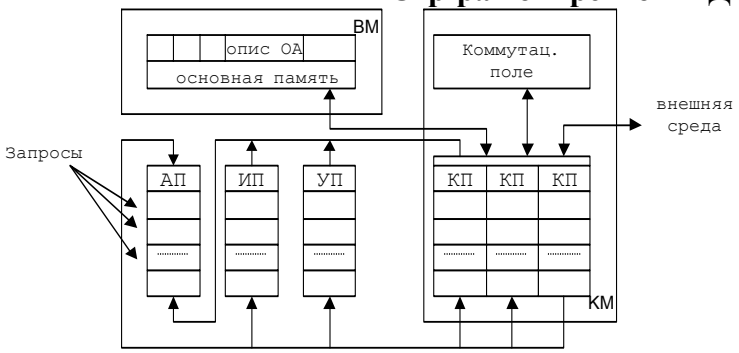
Недостаток этой с-мы в том, что если есть 4 автомата, то их можно только уменьшать, увеличить их нельзя.

Состав МДА: сост. из операционной и коммутационной частей. Операционная часть обеспечивает хранение данных и реал-цию автоматных операторов. Коммутационная часть обеспечивает динамическое изменение связи м/у операционными автоматами. Состоит из специализированных коммутационных автоматов.

Недостаток: ДАС обладает способностью ограниченного роста.

Вывод: необходимость виртуализации коммутационной и операционной части.

Стр-ра конкретной МДАС.



Рисуем блок операционных автоматов. Т.е. операционная часть нашего мультипроцессора представляет собой мультипроцессорную с-му с общей памятью. Далее нарисуем правее коммутационное поле (позволяет соединить каждого с каждым, но кратковременно). Потребуется 4 типа процессоров: административный, исполнительный, управляющий, коммутационный. Каждый процессор имеет очередь запросов на обработку, т.е. у нас процессор один, а моделирует он произвольное число автоматов. Поэтому у нас есть блоки данных, к-рые передаются от одного процессора к другому и представляют собой задание на выполнение порождения, уничтожения автомата в автоматной сети и т.д. Существуют следующие связи: АП определяет тип элементарного задания, которое необходимо выполнить и ставит их в ту или иную очередь. Т.е. АП получает задания или сообщения и смотрит: это сообщение связано с управлением или исполнением и взависимости от этого ставит в очередь эти задания к УП или ИП. Может оказаться, что текущий акт элементарной обработки является актом коммутации возникновения связи или же ее уничтожения, поэтому вполне возможно, что разобравшись с тем, что за задание текущее или стр-ра данных находится в его очереди направляет это задание одному из коммутационных процессоров. Понятно что ИП или УП, отработав свое задание могут породить какое-то свое задание и поэтому рисуем так, чт.е. выход и блоки данных с этого выхода поступают на вход АП. Также вполне возможно, что нам потребуется точно такой же выход от КП. КП, в свою очередь, записывает или читает эти блоки данных из основной памяти в область того или иного ОА. КП-ы обеспечивают возможность взаимной адресации двух таких ОА ч/з коммутационное поле, т.е. в коммутационном поле хранится текущее состояние связей м/у этими автоматами. Требуется внешний выход, чтобы с-ма не была замкнута – внешняя среда.

ОА-ты отображаются в вычислительные модули, каждый из к-рых состоит из процессоров(реализация автоматных ф-ций), основная и внешняя память, хранение описания автоматов и их состояний, каналы ввода/вывода взаимодействия автоматов м/у собой и с внешними устр-вами. Коммутационная часть отображается в коммутационные модули.

АП управляет ресурсами и организует совместное ф-ционирование процессоров управляющего модуля и коммутационного модуля.

ИП реализует выполнение автоматных операторов.

УП выполняет интерпритацию эл-нтов динамической автоматной сети.

КП реализует внутреннюю, межмодульную и внешнюю коммутацию.

Ситуация след.: мы должны виртуализовать наше представление о ДАС. Сделаем это сл. обр.: введем мн.тво специализированных процессоров, к-рые будут обмениваться сообщениями, но эта специализация процессоров обеспечит нам эффективное реш-е задачи связанное с представлением ДАС. Мы берем описываем каждый ОА(они не обязательно одинаковы). Описание состоит из автоматной ф-ции и состояния или на языке пр-ммирования – это тело ф-ции и те статические данные к-рые она использует. АП

формирует очереди заданий для каждого процессора в с-ме по типам. Самая 1-я к-нда АП в момент нажатия кнопки “пуск” – выбрать описание 1-го ОА и “посмотреть”, что он должен делать. АП выдает к-нду УП: выбери описание автомата ч/з коммутационную сеть из ОП. Это описание выбирается и поступает в УП. Он смотрит, что это описание автомата и надо его исполнить. Он говорит АП-ру: “посмотри есть ли свободный ИП и поставь его в очередь там. Далее после того как вычислилась ф-ция надо восстановить описание, потому что его может кто-то использовать, поэтому этот ИП говорит: “ сохрани новое состояние такого-то автомата в ОП”, что он ч/з КП, отправляя туда новое состояние. Далее предположим надо установить связь м/у 2-мя автоматами. ИП видит, что автомат породил новую связь и говорит тогда АП: “сохрани состояние и установи новую связь м/у двумя узлами”.

Раздел 8: Проблемно-ориентированные системы обработки данных

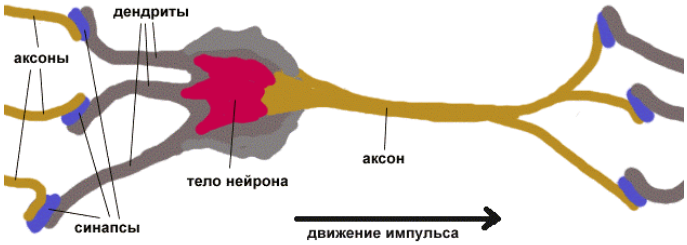
Тема 8.1. Нейронные сети и нейрокомпьютеры

Биологический нейрон.

Мы с вами обсудили проблемы, к-рые возникают при решении нек-рых прикладных задач. Мы видим примеры, когда реш-е этих задач реализуется эффективно, а с помощью технических средств мы эту реализацию достигнуть не можем. И мы должны разобраться в чем природа этой эффективности. Конечно, биологической с-ме или мозгу тяжело сравниться со скоростью вычислений, которую делают современные вычислительные средства. Но есть категория задач для к-рых современные средства являются несостоятельными, например, задача распознавания образов. И вот мы должны выяснить природу этого феномена.

Нейрон – это основная стр-рная ф-циональная единица нервной с-мы, обладающая специфическим проявлением возбудимости.

Рис.



Главная особенность нейрона – наличие специальных отростков: дендридов и аксонов. Воспринимающая часть нейрона – дендриды, снабженные рецепторной мембраной. Нейрон воспринимает нервные сигналы с помощью специальных образований в мембране. Синапс – это специализированный ф-циональный контакт м/у нервными клетками, служащий для передачи преобразования нервных импульсов. Передача возбуждения и торможения м/у нейронами происходит ч/з химические, электрические и смешанные синапсы. Передача нервного раздражения к области синаптического контакта осуществляется распространением нервного импульса по нервному волокну аксону. Длительность 1 импульса – 1млсек. Интенсивность раздражения определяется скважностью импульса – чем чаще импульсы, тем более интенсивные сигналы. При подходе нервного импульса к синапсу происходит химическая реакция и выброс из аксона биологически активного вещества медитатора, которое дифундируя в синаптическую щель в мембране дендрида достигает нервной клетки и реализует синаптическую передачу. В рез-тате суммирования местных процессов возбуждения и тарможения в дендриде в наиболее возбудимой части нейрона – тригерной зоне, возникают нервные импульсы и т.д. Исходя из ф-ций нейроны подразделяются на сенсорные, ассоциативные и эффекторные. Последовательность синаптического объединения нейронов сенсорного, ассоциативного и эффекторного нейронного слоя образуют рефлекторную дугу. В рез-тате чего образуется НС.

PS 1 к биологическому нейрону: явление торможения возбуждения нейрона может быть смоделировано наличием некоторой пороговой величины интенсивности нервных импульсов: ниже порога – торможение, выше – возбуждение.

PS 2: наблюдается явление усталости нейрона, когда его сильное возбуждение(торможение) со временем ослабевает, что согласуется с известными представлениями о природе информации. В современных сетях это не реализуется.

Что такое современное представление о природе информации? Наверняка вы все знаете, что количество информации по Шенону есть величина пропорциональная логарифму

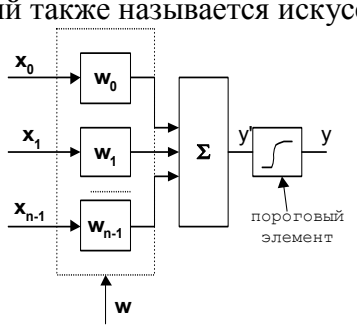
вероятности появления этого сообщения. Т.е., когда нейрон получает одни и теже импульсы и постоянно раздражается, то нейрон переходит в состояние насыщения.

PS 3: память нейрона определяется величиной его внутреннего возбуждения и изменяется с течением времени.

PS 4: память НС опред-ся межнейронными связями, к-рые постоянно перестраиваются количественно и качественно в процессе ф-ционирования НС.

Искусственные нейроны.

То что мы сказали попробуем смоделировать. Представим нейрон в виде нек-рого материального устр-ва или объекта. Базовым эл-нтом искуственных нейронных сетей явл. линейный пороговый эл-нт, к-рый также называется искусственным нейроном.



У нас имеются входы у ЛПЭ, к-рый моделирует синаптический контакт и дендриды. Имеется умножители $w_0...w_{n-1}$. Моделируется внутренняя память нейрона. У нас, фактически, все НС явл. НС кратковременной памяти. Умножители, к-рые реализуют умножение входных сигналов на весовой коэф-нт w_i. w – вектор весовых коэф-нтов. Сумматор моделирует тригерную зону, т.е. реализуется линейная комбинация входных сигналов. Представить произвольную ф-цию с помощью линейной комбинации не сможем. А мы знаем, что нейрон помимо суммирования входных сигналов с учетом кратковременной памяти, он еще обладает способностью иногда эти сигналы просто не воспринимать, когда они очень часто возникают или когда возникают очень редко. Это м/б смоделировано обязательно наличием некоего нелинейного эл-нта – пороговый эл-нт.

Значит любой нейрон или пороговый эл-нт реализует ф-цию: $y = \rho(\sum_{i=0}^{n-1} w_i x_i)$

Линейный пороговый эл-нт моделирует нейрон с учетом его внутренней памяти.

Классификация искусственных нейронов.

1. По виду входных и выходных сигналов
2. По виду ф-ции архивации

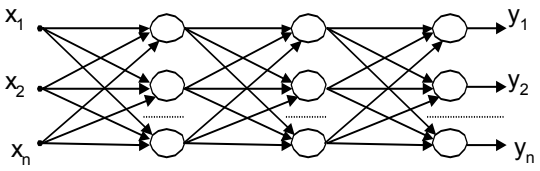
Есть 2 типа нейронов по виду входных и выходных сигналов: дискретные и аналоговые, хотя на самом деле все дискретное или все аналоговое. Дискретные, когда входные сигналы принимают значение на конечном множестве чисел. Аналоговые, когда значения принимаются на некотором отрезке числовой оси. В технике аналоговые нейроны моделируются использованием чисел с плавающей точки, а дискретные - с помощью целых чисел. Шкала целых чисел от шкалы чисел с плавающей точкой отличается линейностью шкалы, у целых чисел линейная шкала, а у нелинейных – нелинейная шкала.

2 ф-ции архивации: сигмоидальные и колокоидальные.

1-я нелинейность представляется, как правило, следующей ф-цией, а 2-я – это следующая ф-ция.

Нейронные сети.

НС, как правило, сост. из 3 слоев: сенсорный слой или предворительная обр-ка (подготовка) данных, ассоциативный слой – обр-ка данных, эффекторный слой – формирование рез-тата. Имеется несколько входных сигналов. Понятно, что каждый нейрон должен получать по возможности большее число сигналов. Это покажем след. обр.



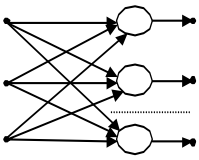
Сенсорный слой формирует рез-таты предварительной обработки, к-рые попадают на нейроны ассоциативного слоя. Понятно, что нейроны ассоциативного слоя «сообщают» что-то нейронам эффекторного слоя, к-рые уже начинают «дергать» за нужные «веревочки».

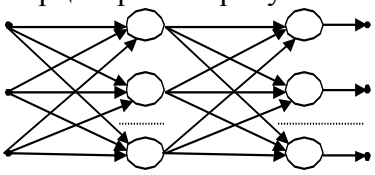
НС на рисунке является полносвязной нейронной сетью, когда каждый входной сигнал попадает на все нейроны сенсорного слоя и каждый выход нейрона поступает на все нейроны след. слоя. Есть 2 причины ограниченного использования полносвязных сетей: отсутствие «долговременной памяти» и сложность технической реал-ции. Помимо этого сущ. сети, как с прямыми, так и с обратными связями.

Арх-ры нейронных сетей.

НС будут разделятся на 2 класса: с прямыми связями и с рекурентными(обратными) связями.

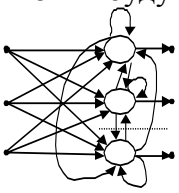
1-ый тип называется перцептрон и многослойный перцептрон. Нарисуем 2 слоя.



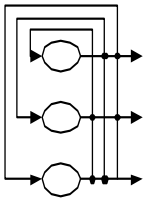


В перцептроне, как правило, есть возможность установления произвольных связей м/у нейронами и входными и выходными сигналами.

Рекурентные или с обратными связями будут 4-х типов:

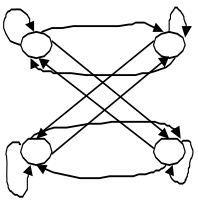


1. Соревновательные.
2. Сеть Кохонена. Есть связи только близлежащих нейронов м/у собой.
3. Сеть Хопфилда.



Входные сигналы не будем показывать, а покажем только обратные связи. Связи обратные являются регулярными. Эти связи показаны так: в слое из 3 нейронов каждый нейрон имеет связь с двумя какими-то другими. Обратные связи здесь задаются с помощью м-цы коммутации или м-цы соединения.

4. Модель .



М/у нейронами устанавливаются всевозможные перекрестные двусторонние связи. Каждый из этих нейронов имеет сильную рефлексию.

Обучение нейронных сетей.

Фундаментальным св-вом биологич. НС явл. способность к обучению. Самой интересной особенностью того, чт.е. на практике явл. возможность самообучения.

Процесс обучения состоит из:

1. Настройка (выбор арх-ры сети). Когда возникает биологическая НС, то там выбора арх-ры нет. Здесь мы вынуждены выбирать арх-ру, потому что реализация связей каждого нейрона с каждым является технически нереализуемой задачей.
2. Настройка весов по обучающей выборке, где обучающую выборку представляют в виде множества упорядоченных пар $O=\{(X,R)\}$, где X – это входной вектор НС, а R – это тот рез-тат, к-рый должен быть, когда этот входной вектор подан. Обучение состоит из предъявления образов и тот рез-тат, к-рый должен быть на выходе.
3. Итеративная подстройка весовых коэф-нтов.

PS 1(плохое): биологические НС в процессе обучения-ф-ционирования изменяют свою арх-ру, что в искусственных НС отсутствует напрочь.

*PS*2(хорошее): св-во НС обучаться на пр-рах делает их привлекательными по сравнению с с-мами у к-рых правила ф-рования формируются заранее экспертами.

Парадигма обучения.

Парадигма обучения – это модель внешней среды, отражаемая в арх-ре сети и м-ды модификации весовых коэф-нтов. Т.е., парадигма обучения сост. из 2 частей: модели внешней среды, к-рая отражается в арх-ре. Это обязательно потому что у нас невозможно реализовать произвольную связь нейронов в НС, а значит надо ограничить класс решаемых задач, поэтому надо составить модель до решения задачи на нейрокомпьютере, а уже потом когда модель построена можно говорить о том, что эта арх-ра НС наиболее удачно приспособлена для решения этих задач. И только когда эта предварительная часть сделана можно приступать к выбору м-да модификации весовых коэф-нтов по обучающей выборке.

М-ды обучения (м-ды модификации весовых коэф-нтов):

«с учителем», когда по обучающей выборке настраиваются весовые коэф-нты.

«без учителя». Нет необходимости знать правильный ответ. Обучение происходит в рез-тате раскрытия внутренней стр-ры и взаимосвязей входных данных(корелляция м/у образами).

Смешанное обучение. Часть весов определяется посредством обучения с учителем, а часть весов – в процессе самообучения.

Правила обучения.

- Коррекция по ошибке. Заключается в следующем: происходит использование разностей Y-R НС для модификации весов, которая заключается в постепенном уменьшении ошибки при многократной подаче образов из обучающего множества. Что это означает? – подали вы какой-то образ; вычислили ошибку реальной отклик сети Y и тот, к-рый д/б R; зная эту ошибку, мы ее разбрасываем по всем весовым коэф-нтам т.о., чтобы не устранить эту ошибку, а уменьшить. И после того как мы подали следующий обучающий предмет, мы опять чуть-чуть уменьшаем ошибку и это повторяем до тех пор, пока на обучающей выборке эта ошибка не станет приемлемой.
- Обучение Больцмана. Это стахостическое правило обучения, к-рое следует из инф-ных (кол-во инф-ции, содержащееся в повторяющихся сообщениях уменьшается) и термодинамических принципов (с-ма стремится к минимуму энергии). Но в НС принцип Больцамана используется подход, когда мы беспорядочно изменяем коэф-нты так, чтобы уменьшить ошибку, возникающую при использовании обучающей послед-ти. Под ошибкой понимается

расхождение корреляций состояний сети в 2 режимах: до и после корекции коэф-нтов. Состояние сети – это ее весовые коэф-нты.

– Правило Хебба. Наиболее близка к биологическим НС. Основано на нейрофизиологических наблюдениях. Если в нейроне активизируется по нескольким входам одновременно и регулярно, то сила синаптической связи возрастает. К примеру, если по некоторому входу идет постоянная подача нервного импульса, то синапс устроен так, что он начинает более активно реагировать на сигнал той же интенсивности, к-рый прийдет следующим, но не забывайте, что ч/з какое-то время наступает насыщение. Правило сост. в том, что изменение синаптического веса w_i зависит только от активности нейрона, к-рые связаны с данным. Например корреляция по ошибке может быть реализована каким образом: любой нейрон может быть изменен, вы же не знаете стр-ру сети, т.е. она не привязана ни к какой арх-ре; в обучении Больцмана вообще случайно выбирается коэф-нт, а здесь – нет: выбираются только те весовые коэф-нты и меняются только там, где есть связь м/у нейронами. Нейрон активен, если он не находится в состоянии насыщения. Рис. Получ-ся такая же картина, к-рая была при рассмотрении биологического нейрона: мы начинаем подавать входные сигналы и если значение сигнала по этому входу увелич-ся, то в соответствии с правилом Хебба мы увеличиваем коэф-нт, но это увеличение коэф-нта приводит к тому, что ч/з нек-рое время нейрон становится пассивным, т.е. он не будет влиять на те нейроны с к-рыми он связан.

– Обучение м-дом соревнования. В отличие от правила Хебба, в котором мн-во выходных нейронов могут возбуждаться одновременно, при соревновательном обучении выходные нейроны соревнуются м/у собой за активацию. Модифицируются только веса победившего нейрона.

Таблица: известные алг-мы обучения.

	Правило	**Арх-ры**	**Алг-м**	**Задачи**
С учителем	Коррекция	Перцептрон(без обратн. связей)	Обрат. распрстранение ошибки	Классиф-я Апроксим-я управление
	Больцмана	Рекурентные(с обратн.связями)	Обучение Больцмана	Для решения задач классификации
	Хебба	Прямого распространения	Линейный дискриминантный анализ	Классиф-я Анализ данных
	Соревнование	Соревновательная арх-ра	Векторное квантование	Категор-я Сжатие данных

Без учителя	Коррекция	Прямое распространение	Обратное распростр.	Классиф-я Апроксим-я Управление
	Хебба	Соревноват.	Анализ главных компонентов	Анализ и сжатие данных
		Хопфилда	Ассоц.память	Ассоциации
	Соревнование	Кохонена, Соревноват.	Векторное квантование, SOM	Категориз-я Сжатие данных

PS: есть такое понятие, как емкость НС. Она означает следующее – это максимальное число образов, которое она может запомнить. Если вы возмете простую нейронную сеть и подадите 1000 образов, то она эту обучающую выборку на 99% проигнарирует, она просто переобучится и будет фактически различать все, что мы хотим, но не т.к. мы хотим. Поэтому в зависимости от задачи должна быть использована НС соответствующей емкости.

Алг-м обратного распространения ошибки.

Этот алг-м явл. аналогом м-да градиентного спуска. В этом случае для градиентного спуска мы должны знать как нам изменить коэф-нты, чтобы получить требуемое увеличие или уменьш-ие выходов на нейроны. Псть есть нейрон (рис). Он описывается след. ур-ем: $y=\rho(\Sigma w_i x_i)$. Если мы знаем, что ф-ция больше требуемого знач-ия, то зная производную при заданных x_i, мы будем знать что нам делать с коэф-нтами (рис). (к рис.) вот есть 2 ф-ции. Если ошибка больше 0, мы должны уменьшить y, а увеличить x, т.е. увеличить весовой коэф-нт. Чтобы уменьшить ошибку мы должны знать в какую сторону изменять весовой коэф-нт. Далее, как правило в м-дах, к-рые используют обратное распространение ошибки исп-ся простые аналитич. ф-ции архивации: $Y = \frac{1}{1+e^{-x}}$, $X = \sum_{i=0}^{n-r} w_i x_i$.

В общем случае надо использовать частные производные, потому что у нас много переменных. Но для нашего упрощенного рассмотрения будет достаточно вычисления производной по $X = \sum_{i=0}^{n-r} w_i x_i$. $\frac{dY}{dX} = Y(1-Y)$.

Цель обучения:

Подстройка весов так, чтобы на обучающей выборке минимизировать суммарную ошибку распознавания. Это как бы примитивный критерий.

Алг-м состоит из следующих шагов:

1. Перед началом обучения всем весам присвоены небольшие начальные значения, выбранные случайным образом: $w_i \sim 0$.
2. Выбираем очередную обучающую пару(X,R).
3. Вычисляем выход сети Y и вектор ошибки E=Y-R.
4. Выполняем коррекцию весов $w^{(1)} \Rightarrow w^{(2)}$, исходя из примитивных представлений.
5. Повторяем шаги 2-4 до тех пор, пока ошибка распознавания e на всей обучающей выборке не достигнет допустимого уровня величины.

Тема 8.2. Системы цифровой обработки сигналов

Область цифровой обработки сигналов включает специфич. алг-мы обработки числовых данных, получаемых при аналого-цифровом преобразовании (АЦП):

Измерительных (электрических, механических сигналов)

Речевых

Видео

Область применения:

Анализ сигналов (аэрофотосъемка, биология, металлургия). В рез-тате этого анализа мы должны выявить какие-то объекты;

Распознавание (сейморазведка, локация);

Обр-ка.

Основные задачи ЦОС:

Фильтрация;

Спектральный анализ;

Классификация распознавания;

Воспроизведение и конструирование сигналов.

М-ды, используемые при цифровой обработке сигналов:

Дискретное преобразование;

Фильтрация;

Вычисление корреляции.

Оказывается все задачи в указанных областях сводятся к последовательному применению перечисленных м-дов или их комбинаций.

Особенности ЦОС:

ЦОС всегда выполняется в режиме реального времени;

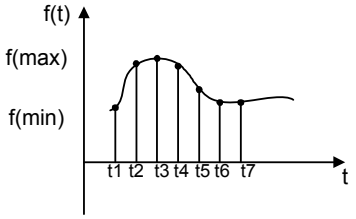
Высокое быстродействие;

Большие объемы данных;

Очень небольшое число неизменяемых алг-мов.

Вывод: как правило, с-мы ЦОС являются специализированными с-мами.

Дискретизация и квантование



Дискретизация – это преобразование аналогового сигнала в последовательность его выборочных значений, отстоящих друг от друга на время интервала дискретизации Δt.

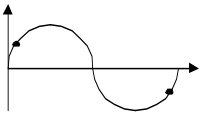
Вот у нас есть некая ф-ция времени, которая представляется в виде ф-ции имеющей непрерывное мн-во значений от f(max) до f(min). Выбираем некий начальный момент времени t1 и относительно него ч/з, в частном случае равные, а в общем – неравные, интервалы времени Δt получаем выборочные значение сигналов, т.е.

$f_i \in [f_{min}, f_{max}]$. Это интервал числовой оси, т.е. при дискретизации мы получаем некую решетчатую ф-цию (задается не во всех точках, а только в точках дискретизации) действительного переменного. в этом случае можно записать: $t_{i+1} = t_i + \Delta t \Rightarrow t_{i+1} = t_0 + i\Delta t$.

Интервал дискретизации может быть переменным. Когда интервал дискретизации переменный, то очевидно, что надо просуммировать все те интервалы на к-рых ранее мы рассматривали ф-цию: $t_{i+1} = t_0 + \sum_{i=0}^{i} \Delta t_i$.

Спрашивается, а как часто нам надо выбирать ф-цию. Вот есть некий сигнал, к-рый мы представляем в виде ф-ции зависящей от времени. Как часто мы должны его оцифровывать? Запишем следующую теорему.

Теорема Котельникова:



Если частотный спектр сигнала f(t) ограничен нек-рым значением F, то этот сигнал может быть восстановлен после дискретизации без погрешностей по выборочным значениям, следующим с интервалом $\frac{1}{\Delta t} > 2F$. Физический смысл теоремы Кательникова очень прост: если у нас имеется в спектре сигналов максимальная гармоника, которая отлична от нуля, равная частоте F, то для того чтобы мы смогли восстановить и частоту этой гармоники и ее амплитуду надо как минимум 2 точки. Поэтому мы должны оцифровывать сигнал с частотой превосходящей двойную частоту спектра. На практике интервал времени Δt мы должны выбирать меньше чем тот, к-рый может быть получен из теоремы Кательникова: $\Delta t < \frac{1}{3 \div 5} F^{-1}$.

Нарисуем таблицу.

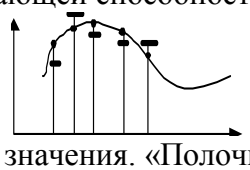
Сигнал	Частота дискретизации(Гц)	Длительность выборки (с)	Разрядность	Объем выборки
Измерительные	$10 \div 10^3$	$100 \div 10$	$22 \div 12$	$10^{-1} \div 10^5$
Речевые	$10^4 \div 10^5$	$1 \div 10^{-1}$	$14 \div 8$	$10^3 \div 10^5$
Изображение	$10^6 \div 10^8$	$10^{-2} \div 10^{-3}$	$8 \div 4$	$10^4 \div 10^6$

Оказ-ся, в завис-ти от полосы частот, к-рые занимают эти сигналы, м/б подсчитаны след. диапазоны дискрет-ции, к-рые использ-ся в этих прикладных обл-тях.

Квантование – это процесс преобразования выборочных значений сигнала в последовательность отсчетов. Идея здесь следующая: мы получаем решетчатую ф-цию, а это означает, что она принимает значение в диапазоне от минимального до максимального. Но т.к. в реальных с-мах у нас всегда задана некая точность измерения (обработки, восприятия), а это означает, что мы можем задать точность δ с которой мы будем обрабатывать наш сигнал. Более того, из-за наличия шумов мы принципиально не можем эту точность устремить к нулю. Поэтому если мы зададим некую точность δ, или ее еще называют разрешающей способностью обработки, то это означает, что наш диапазон от f_{min} до f_{max} разбивается на N частей и значение ф-ций принадлежащей какой-то из этих частей будет нами отождествлено числом, равным номеру этого интервала. Заранее понятно, что мы не можем обрабатывать данные со сколь угодно большой точностью. Значит разрешающая способность любой технической с-мы ограничена. Отсюда следует, что число разрядов, с помощью к-рых мы должны закодировать полученные выборочные значения, может быть представлена, как целая часть от величины: $n = \left\rceil 1 + \log_2 \frac{f\max - f\min}{\delta} \right\lceil$.

Заметьте, у нас тогда были выборочные значения, а после квантования они называются отсчеты сигналов.

Реальный отсчет сигнала может быть или больше или меньше выборочного значения на величину не более половины разрешающей способности.



«Точкой» обозначены выборочные значения. «Полочкой» обозначены отсчеты сигналов.

Разрядность уменьшается из-за того, что объем данных растет, а объем данных связан с таким параметром, как длительность выборки, которая подлежит обработке. Мы должны входные данные делить на части и эти части обрабатывать. А если это так, то исходя из предметных областей есть параметр, называемый длительностью выборки и равен KΔt, где К – это число отсчетов сигналов, к-рые подлежат одновременной обработке.

Длительность выборки должна быть такая, чтобы задержка на это время не вызывала особых слуховых эффектов.

Относительная ошибка, которая получается в рез-тате квантования есть $\sum_0 = 2^{-n}$. Абсолютная ошибка квантования: $\sum_a = \delta \sum_0 = \delta * 2^{-n}$. При проектировании с-м ЦОС добиваются, чтобы эта ошибка была по уровню сигнала не более заданной.

Дискретное ортогональное преобразование.

Любая ЦОС основана на разложении произвольной ф-ции сигнала в ряд по некоторой с-ме ортогональных ф-ций. Например, ф-ции cos и sin являются ортогональными и можно разложить произвольную ф-цию в ряд по этим ф-циям. Оказывается некий произвольный сигнал мы можем представить в виде композиции более простых сигналов. Обобщая это можно записать, что произвольную ф-цию f(t) можно представить в виде ряда некоторой ортогональной ф-ции, но в связи с тем, что мы не можем обрабатывать бесконечные ряды, поэтому мы вынуждены ограничить число членов в этом ряду. А в этом случае мы теряем точность, но т.к. у нас уже задана разрешающая способность, то мы можем ограничиться числом членов в этом ряду, чтобы остаточный член этого ряда был меньше, чем наша разрешающая способность. И, как следствие этого, мы записываем не бесконечную, а конечную ф-цию.

$f(t) = \sum_{i=0}^{k-1} S_i \Theta(i,t)$ где Θ и t – это ортогональные ф-ции, по к-рым раскладывается произвольная ф-ция;

S_i – коэф-нты разложения или же совокупность таких коэф-нтов составляет спектр сигнала по с-ме ф-ций Θ(i,t). Для того чтобы мы могли переходить от спектра к сигналу и от сигнала к спектру мы должны иметь возможность вычислять эти коэф-нты. Поэтому мы должны иметь возможность вычислить эти коэф-нты:

$$S_i = \sum_{t=0}^{k-1} f_t \Theta^{-1}(t,i),$$

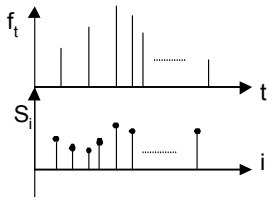
Т.е. с-ма ф-ций будет тогда ортогональна, когда мы, зная отсчеты сигналов в дискретные моменты времени сможем получить спектр этого сигнала, вычисляя указанную сумму.

Усл-ие ортогональности следующее:

$\sum_{i=0}^{k-1} \Theta^{-1}(t,i) * \Theta^{-1}(t,i) = \delta_{t\tau}$,где $\delta_{e\tau} = \begin{cases} 1, t = \tau \\ 0, t \neq \tau \end{cases}$

но т.к. мы работаем не с ф-цией, а с отсчетами, поэтому эта первая сумма будет равна какому-то отсчету в момент времени t, где t – это число от 0 до К-1, где К – число отсчетов сигнала или объем выборки.

Как найти К? Частота дискретизации известна; можем посчитать длительность выборки; можем найти объем выборки, а объем выборки и есть К, т.е. сколько точек мы должны одновременно обработать. В виде графика это выглядит следующим образом:



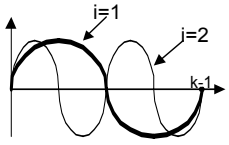
Есть ф-ция f(t), заданная отсчетами и путем выч-ия этой суммы получаем спектр сигнала. Если в кач-ве ортогональных ф-ций использовать sin или cos, то получ-ся Фурье-спектр. Но кто сказал, что не сущ. др. ф-ций кроме sin и cos, для того чтобы эти ф-ции удовлетворяли приведенному ниже дискретному усл-ию ортогональности. Оказ-ся таких ф-ций достаточно много. Выр-ия, к-рые составляют дискретные ортогональные преобразования можно записать в матричном виде:

$$\begin{cases} F = D * S \\ S = D^{-1} * F, \text{где} : D^{-1} * D = E \end{cases}$$

Т.о., задачи цифровой обр-ки сигналов мы можем свести к матричным вычисл-ям. Нам требуется реш-ия след. зад.: умножение век-ра на м-цу и обращение м-ц.

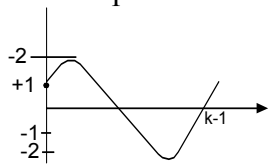
Базисы дискретного преобразования:

– Фурье. В этом случае:



$$\Theta(i,t) = e^{-\sqrt{-1}*\frac{2\pi}{k}*it} = \sin(\frac{2\pi}{k} * it) + \sqrt{-1}\cos(\frac{2\pi}{k} it)$$

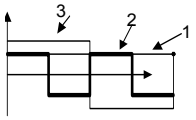
– Хартли



$$cas(i,t) = \sin(\frac{2\pi}{k} it) + \cos(\frac{2\pi}{k} it)$$

Ф-ия Фурье отличается от Хартли тем, что Фурье явл. ф-ей комплексного переменного, а ф-ия Хартли явл. ф-ей действительного переменного. Дискретное преобразование Фурье уходит на задний план, а на передний выходит дискретное преобразование Хартли, т.к. требует в 2 раза меньше операций для вычислений.

– Уолша. Классический базис, предназначенный для выполнения в чистом виде дискретной обработки данных.

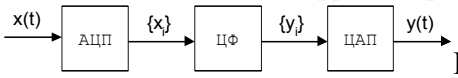


$wal(i,t) = (-1)^{\sum_{i=0}^{n-1} i_j t_j}$, где i_j и t_j – это цифры двоичного представления чисел i и t.

Внешний вид этой ф-ции: ф-ция нулевой частоты(1) равна постоянной частоте.

Цифровая фильтрация

Служит для подавления помех и основана на различиях спектров полезного сигнала и помехи. Общая схема цифровой фильтрации следующая:

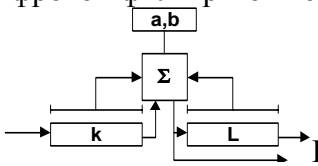


Имеется аналоговый сигнал x(t), к-рый поступает на устр-во дискритизации и квантования АЦП; далее полученные отсчеты сигнала x_i поступают на цифровой фильтр; после обработки ЦФ эти отсчеты искажаются, т.е. появл. др. массив чисел y_i; эта последовательность чисел поступает на ЦАП и уже с выхода получается отфильтрованный сигнал y(t).

Мы выяснили закономерности, к-рые здесь присутствуют и к-рые позволяют нам правильно выбрать как частоту дискретизации, так и количество уровней квантования, для того чтобы не потерять те данные, к-рые были изначально в сигнале.

Цифровая фильтрация есть многомерная и одномерная. В данном случае, цифровая фильтрация одномерного сигнала описывается следующим рекурентным уравнением: выходной отсчет в следующий момент времени есть $Y_{i+1} = \sum_{k=0}^{k-1} a_k X_{i-R} + \sum_{l=0}^{l-1} b_l Y_{i-l}$. Это так

называемый рекурсивный фильтр для фильтрации одномерных сигналов. Изменяя коэф-нты a_k и b_l можно задавать различные филтрации. a_k и b_l – это постоянные коэф-нты. Из пределов суммирования виден диапазон их изменения. K и L – это число ячеек во входной и выходной очереди фильтра. Если посмотреть на это рекурентное выр-ие, то окажется, что следующий отсчет выходного сигнала зависит линейно от предыдущих K отсчетов, т.е. для того чтобы вычислить Y_{i+1} мы суммируем предыдущие входные значения на глубину до K, но также учитываются предыдущие выходные значения на глубину L. Т.е. цифровой фильтр можно образно представить сл. обр.:

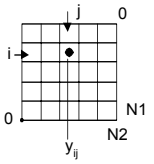


Есть некая входная и выходная очередь, состоящие из K и L ячеек; также есть сумматор и умножитель, к к-рому подключается запоминающее устр-во коэф-нтов a и b. Это линейная модель фильтрации – линейный цифровой фильтр. Более сложную модель фильтрации не реализуют. Иногда выход цифрового фильтра выводится прямо с текущ. полученного отсчета, но это как бы запоминается ранее выданные отсчеты одного сигнала. Для того, чтобы фильтр начал полностью фильтровать, надо записать все входные данные, но если не записывать, то предполагается, что эти данные имеют какие-то четко оговоренные знач-ия. Если коэф-нты b_l положить равными 0, то получается не рекурсивный фильтр, т.е. отсчеты выходного сигнала не зависят от ранее полученных отсчетов сигналов. Мы видим, что в этом случае стр-ра выр-ия такая же, как и в дискретном ортогональном преобразовании, т.е. коэф-нты, к-рые постоянны и известны заранее суммируются с умножением на отсчеты сигналов. Т.е. чтобы выполнить цифровую фильтрацию нужно выполнить тот же набор операций, что и для дискретного преобразования Фурье, Хартли и т.д.

Далее, что произойдет, если надо отфильтровать не одномерный сигнал, а, к примеру, двухмерный? Двухмерный сигнал – это изображение.

Рекурсивный цифровой фильтр для фильтрации изображений.

В этом случае изображение разбивается на эл-нты:

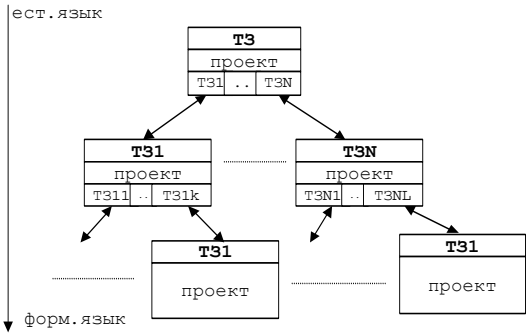


Раздел 9. Автоматизация проектирования

Тема 9.1. Спецификация и требования к САПР.

Ранее были выявлены 3 этапа проектирования ВС: арх-рный (с-мный, стр-рный), логический (ф-циональный), технический (принципиальный).

Рассмотрим эти этапы в контексте автоматизации проектирования. Особенностью здесь является следующее, что имеются входные данные для каждого этапа, а выходными данными являются входные данные следующего этапа. Нарисуем иерархию проекта:



Имеется технич. задание на с-му в целом, к-рое определяем те спецификации, к-рые для этой с-мы д/б выполнены. Технич. задание имеет свою стр-ру, описываемую в соответств. документах. Технич. задание специфицирует требования для какой-то части или целой ВС и явл. входными данными. Проделав какую-то работу над технич. заданием мы получили проект. Этот проект не м/б полностью детализирован, потому что наша с-ма изначально имеет стр-ру, а значит в составе этого проекта есть технические задания или спецификации на составные части. Поэтому выделяем эти технич. задания: ТЗ1...ТЗN. В соответствии с этим технич. заданием проектируют какой-то ф-циональный блок или модуль ТЗ1....ТЗN. На основании этого задания создается некий проект ф-ционального уровня, но т.к. мы все детализировать не можем до принципиального уровня, туда входит технич. задания на составные части этого ф-ционального модуля ТЗ11...ТЗ1К.

Проекты бывают 3 уровней: эскизный, технический и рабочий. Они отличаются степенью детализации.

Используются 2 м-дологии проектирования: нисходящий и восходящий.

Восходящий м-д: при восходящем проектировании сначала проектируется относительно низкий уровень с-мы, определяются технические задания на его модули и части, к-рые используют в качестве стр-рных (ф-циональных) эл-нтов при проектировании на более высоком уровне. Нельзя что-то спроектировать, придерживаясь только восходящей или нисходящей м-дологии.

Вывод: процесс проектирования итерационный.

Для использования САПР необходимо формализовать:

1. Описания или технические задания, т.е. входные данные для каждого этапа.
2. Разработать м-ды (алг-мы) реализации этих описаний.

Стрелка на рисунке показывает в каком уровне увеличивается формализация: от естественного языка до формального языка.

Разработан широкий спектр языковых средств, к-рые применяются на каждом уровне. Они помогают естественный язык частично формализовать, и довести эту частичную формализацию до полностью формализованного описания, к-рый позволяет получить схему принципиальную нашего устр-ва. На стр-рном уровне используется язык SDL – с этим языком работает человек на естественном языке. На ф-циональном уровне используется язык типа DDL – язык цифрового проектирования. На принципиальном уровне используется язык логических выражений LDL.

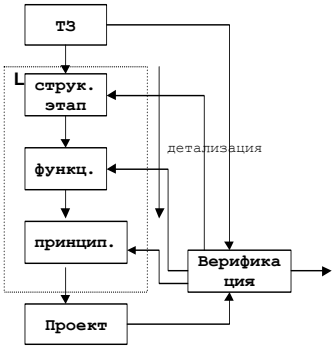
Важно заметить, что проектир-ие на каждом уровне может осущ-ся с 3-мя степенями арх-рной детализации: крупноблочный, среднеблочный и мелкоблочный.

На арх-рном уровне: крупноблочный и иногда среднеблочный.

На логическом уровне: возможны все 3 уровня, но как правило среднеблочный, остальные встречаются редко.

На техническом уровне: мелкоблочный и редко среднеблочный.

На каждом из уровней возможны 3 этапа проектирования. Представим эти 3 этапа в виде следующей диаграммы.



Тех. задание явл. входными данными для нашего этапа. И тут есть 3 этапа: стр-рный, ф-ный, принципиальный. И только после этого получ-ся проект. На каждом из этих этапов степень детализации увелич-ся. На последних 2 этапах возможно появление временных диаграмм. Самое интересное, что после получ-ия проекта мы переходим к очень важному этапу – верификации проекта, к-рая может изменять что-то на каждом их этих уровней.

Верификация рез-татов заключается в проверке выполнения технического задания или спецификации. Для верификации используются следующие м-ды:

1. Натурное моделирование. Создается макет устр-ва и происходит его тестирование.
2. Имитационное моделирование. Путем обработки рез-татов выполняется проверка правильности проектирования.

Вывод: использование САПР предполагает формализацию спецификаций (технич. заданий) и проектов путем формализации описания и ф-рования устр-в.

Стр-рный этап проектирования.

На стр-рном этапе проектир-ия для описания исп-ся язык SDL, к-рый позволяет формализовать не только описания, но и ф-рование устр-ва на стр-рном уровне.

Ф-циональный этап.

Используется яз. ф-ционального проектирования DDL – Digital Design Language.

Язык представляет собой язык спецификации или описания автоматной сети.

Ключевые слова:

SYSTEM - с-ма

TIME – задание синхронизации. Например тактовый генератор CL с частотой 10.

REGISTER – описание регистров. К примеру, регистр данных 16-ти разрядный: DATA(16)

TERMINAL – внешние сигнальные линии. К примеру, 16-ти разрядная шина, которая входит в ваше устр-во: BUS(16). Сигналы, к примеру, чтения и записи.

Основное описание – это описание конечного автомата, которое начинается с AUTOMATION.

AUTOMATION – описание авт-тов, входящих в состав устр-ва. К примеру: AUTOMATION CPU:CL – описание какого-то центрального процесса, причем для его работоспособности необходим тактовый генератор, к-рый можно описать, как отдельный авт-т. Далее описываются регистры: REGISTER IR(16)=OP(4)|D(4)|... Далее описываем сост-ие авт-та, к-рое сост. из след. стр-ры: название сост-ия, усл-ие перехода в это сост-ие (булевое выр-ие), далее перечисляются операции, к-рые выполняются: STATE

Название: условие: операции

END

Пример:

FETCH1:MEMEN:

IR⇐BUS,IP⇐IP+1⇒FETCH2

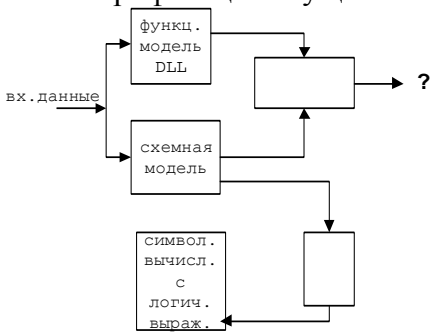
Состояние FETCH наступает, когда входной сигнал MEMEN является активным и состоит из выполнения следующих пересылок: из шины данных данные передаются в регистр к-нд, счетчик к-нд увеличивается на 1 и переходит в состояние FETCH2. В состояние FETCH2 что-то делается и т.д.

Язык объединяет в себе описание и ф-ционирования и входных и выходных данных.

Тема 9.2. синтез и верификация логических схем

Сост. из след. этапов:

1. Синтез схем из лог. эл-нтов. После того как мы проверили правильность ф-ционирования авт-та, мы берем граф перехода, представляем в виде автоматной табл., определяем входные и вых. знаки, кодируем их в булевой алгебре. На основании этого кодирования и описания ф-рования, к-рое проверено на языке ф-цного проектирования составляем лог. ур-ия, к-рые реализуем в виде схемы. Как правило, синтез схем из лог. эл-нтов разделяется на 4 части:
   А.1) проектирование схем передачи данных.
   А.2) проектирование схем управления.
   А.3) проектирование операционных схем
   А.4) оптимизация рез-татов проектирования.
2. Верификация. Цель: проверка соответствия полученных рез-татов описаниям, например пр-мме на языке DDL. Верификация осущ-ся на 2 уровнях:



1-й уровень:ф-циональная модель и схемная модель (логический эмитатор). Мы должны на них подать одни и те же данные. Тут на языке мы уже описали автоматы наши, потом получили схему; схему моделируем с помощью логического эмитатора; рез-таты ф-ционирования исходных данных для этого процесса модели на DDL сравниваются м/у собой на предмет их соответствия и принимается реш-е: схемная модель адекватна ф-циональной или нет.

2-й уровень: когда мы проверяем схемную модель с логическими выражениями, к-рые мы синтезировали, т.е. сравниваем комбинационные схемы с логическими выражениями, к-рые они должны реализовывать.

1. Кремниевый компилятор – это комплекс средств, предназначенный для автоматического размещения на кристалле топологических эл-нтов интегральных схем, работа к-рого базируется на рез-татах схемного проектирования.

Техн.	$t_{зд}$, нс	$P_{потр}$, мВт
ЭСЛ	0,1	5
ТТЛ	0,5	2,5
ТТЛ Ш	0,7	1,5
n-МОП	0,8	0,6
К-МОП	1,0	0,1
БИКМОП	0,5	0,1