

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра интеллектуальных информационных технологий

В.П. КАЧКОВ, С. А. САМОДУМКИН, Д.Г. КОЛЬ

АППАРАТНОЕ ОБЕСПЕЧЕНИЕ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

КОНСПЕКТ ЛЕКЦИЙ

под редакцией
профессора В.В. Голенкова

Минск 2009

УДК 004+004.8 (075.8)

ББК 32.973+32.813 я73

К 30

Качков В.П

К 30 Аппаратное обеспечение интеллектуальных систем. Часть 1: Конспект лекций для студ. спец. "Искусственный интеллект" / В.П. Качков, С.А. Самодумкин, Д.Г. Колб; Под ред. В.В. Голенкова. – Мн.: БГУИР, 2009. – с.

ISBN

В лекциях рассматриваются этапы развития ЭВМ (компьютеров) и эволюция их интеллектуальных свойств, принципы построения и функционирования, направления развития архитектуры и структуры.

УДК 004+004.8 (075.8)

ББК 32.973+32.813 я73

ISBN

© Коллектив авторов, 2009

© БГУИР, 2009

СОДЕРЖАНИЕ

1. Этапы развития ЭВМ. Эволюция интеллектуальных свойств ЭВМ ...	3
1.1. Этапы развития ЭВМ. Особенности традиционных и интеллектуальных ЭВМ.....	5
1.2. Эволюция интеллектуальных свойств ЭВМ	6
2. Арифметические основы ЭВМ	10
2.1. Системы счисления, применяемые в ЭВМ.....	10
2.2. Формы представления чисел в ЭВМ	13
2.3. Кодирование чисел в ЭВМ	16
2.4. Выполнение арифметических операций в ЭВМ	18
2.4.1. Выполнение операций сложения/вычитания чисел, представленных в форме с фиксированной точкой	18
2.4.2. Выполнение операций умножения чисел, представленных в форме с фиксированной точкой	21
2.4.3. Выполнение операций деления чисел, представленных в форме с фиксированной точкой	24
2.4.4. Выполнение операций сложения/вычитания чисел, представленных в форме с плавающей точкой	26
2.4.5. Выполнение операций умножения/деления чисел, представленных в форме с плавающей точкой	27
3. Логические основы ЭВМ	29
3.1. Основные положения алгебры логики	29
3.2. Булевы функции основного функционально полного набора (ФПН)	29
3.3. Основные законы алгебры логики и их следствия	32
3.4. Синтез комбинационных устройств	35
3.5. Сложность логических формул и методы минимизации логических функций.....	39
3.5.1. Общие понятия	39
3.5.2. Расчетный метод минимизации	40
3.5.3. Расчетно-табличный метод минимизации	42
3.5.4. Табличный метод минимизации.....	43
3.6. Функциональная полнота различных наборов	

элементарных логических функций.....	50
3.6.1. Полная совокупность элементарных логических функций.	50
3.6.2. Абсолютная полнота наборов логических функций.....	52
3.6.3. Синтез комбинационных устройств в различных базисах..	55
3.7. Некоторые частные случаи синтеза комбинационных схем ..	60
3.7.1. Синтез комбинационных схем с несколькими	
выходами.....	62
3.7.2. Синтез комбинационных схем, характеризующихся	
не полностью определенной функцией	64
3.8. Принципы построения функциональных устройств	67
3.8.1. Накапливающие схемы	67
3.8.2. Построение различных функциональных устройств	
с использованием триггеров	73
3.9. Синтез цифровых автоматов	77
3.9.1. Определения.....	77
3.9.2. Задание ЦА с памятью	78
3.9.3. Структурный синтез ЦА с памятью	79
4. Принципы построения и функционирования ЭВМ	91
4.1. Общие понятия о принципах организации ЭВМ	91
4.2. Принцип программного управления ЭВМ.....	92
4.3. Состав и порядок функционирования ЭВМ	92
4.4. Функциональная организация ЭВМ	92
4.5. Режимы работы ЭВМ	97
4.6. Принципы построения обрабатывающих подсистем	104
4.7. Принципы построения устройств памяти.....	112
4.7.1. Общие сведения и классификация устройств памяти ..	113
4.7.2. Адресная, ассоциативная и стековая	
организация памяти	116
4.7.3. Организация памяти в многопрограммных системах...	122
4.7.4. Организация виртуальной памяти	123
4.7.5. Совершенствование иерархической структуры	
памяти.....	128
5. Развитие структуры ЭВМ	132
5.1. Основные тенденции развития структуры ЭВМ.....	132

5.2. Основные направления развития подсистемы ЭВМ.....	133
5.2.1. Развитие обрабатывающих подсистем	133
5.2.2. Развитие подсистемы памяти.....	135
5.2.3. Развитие подсистемы ввода-вывода	135
5.2.4. Развитие подсистемы управления и обслуживания	140
5.3. Развитие операционных сред (архитектур) в ЭВМ.....	141
5.3.1. Архитектура виртуальных ЭВМ	141
5.3.2. Архитектура объектной ЭВМ.....	143
5.3.3. Архитектура интеллектуальной ЭВМ	144
5.4. Параллельные компьютеры для интеллектуальных систем ...	148
5.4.1. Классификация параллельных архитектур	148
5.4.2. Многопроцессорные системы.....	154
5.4.3. Графодинамические параллельные асинхронные машины .	160
5.4.4. Семейство суперкомпьютеров СКИФ»	164

1. ЭТАПЫ РАЗВИТИЯ ЭВМ. ЭВОЛЮЦИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СВОЙСТВ ЭВМ

1.1. Этапы развития ЭВМ. Особенности традиционных и интеллектуальных ЭВМ

Традиционно этапы развития ЭВМ классифицируют по поколениям. Поколения характеризуются следующими показателями:

- внутренняя организация (архитектура, программное обеспечение);
- средства взаимодействия пользователя с ЭВМ (языки и формы общения);
- техническая реализация (элементная база, технические параметры).

Развитие этих показателей происходило неравномерно, поэтому деление на поколения достаточно условно (иногда говорят даже о промежуточных поколениях). Тем более ошибочно отдавать предпочтение одному какому-либо показателю (например элементной базе). Некоторые сведения о поколениях ЭВМ и их основных характеристиках приведены в табл. 1.1 [1].

Таблица 1.1

Поколения ЭВМ и их основные характеристики

Характеристики ЭВМ	Значения характеристик ЭВМ поколений			
	1-го (1949-1958)	2-го (1959-1963)	3-го (1964-1976)	4-го (1977-...)
1	2	3	4	5
Элементная база ЭВМ	Электронные лампы, реле	Транзисторы, параметроны	Интегральные схемы, большие интегральные схемы (БИС)	Сверхбольшие интегральные схемы (СБИС)
Производительность центрального процессора (ЦП)	До $3 \cdot 10^5$ оп/с	До $3 \cdot 10^6$ оп/с	До $3 \cdot 10^7$ оп/с	Более $3 \cdot 10^7$ оп/с
Тип оперативной памяти (ОП)	Триггеры, ферритовые сердечники	Миниатюрные ферритовые сердечники	Полупроводниковые БИС	Полупроводниковые СБИС
Объем ОП	До 64 Кбайт	До 512 Кбайт	До 16 Мбайт	Более 16 Мбайт
Характерные типы ЭВМ поколения	-	Малые, средние, большие, специальные	Большие, средние, мини- и микроЭВМ, специальные ЭВМ.	СуперЭВМ, персональные компьютеры, специальные, общие, сети

			Ряды ЭВМ	ЭВМ
--	--	--	----------	-----

Окончание табл.1.1

1	2	3	4	5
Типичные модели поколения	EDSAC, ENIAC, UNIVAC, БЭСМ, Минск-1,11,...,14, УРАЛ	RCA-501, IBM 7090, БЭСМ-6, Минск-2/22, 23,32	IBM/360, IBM/370, PDP, VAX, ЕС ЭВМ, СМ ЭВМ	IBM/390, SX-2, IBM PC/XT/AT, PS/2, Cray, сети, модели «СКИФ»
Характерное программное обеспечение	Коды, автокоды, ассемблеры	Языки программирования, диспетчеры, АСУ, АСУТП	ППП, СУБД, САПР, ЯВУ, операционные системы	БЗ, ЭС, системы параллельного программирования

Обозначения, использованные в таблице:

АСУ – автоматизированная система управления;

АСУТП – автоматизированная система управления технологическим процессом;

ППП – пакеты прикладных программ;

СУБД – система управления базами данных;

САПР – система автоматического проектирования;

ЯВУ – язык высокого уровня;

БЗ – база знаний;

ЭС – экспертные системы.

ЭВМ первого, второго и третьего поколений имели в основном архитектуру фон Неймана, для которой традиционно было характерно:

- единственный вычислительный элемент, в состав которого входят: процессор, каналы передачи данных, память;
- линейная организация ячеек памяти фиксированного размера;
- одноуровневая адресация ячеек памяти;
- машинный язык низкого уровня (команды, приводящие к выполнению простых операций над простыми операндами);
- последовательное централизованное управление вычислениями (поток команд);

- ограниченные возможности ввода-вывода.

Такие ЭВМ часто называют *традиционными*. Они имели достаточно низкий интеллектуальный уровень.

ЭВМ четвертого поколения обладают большими функциональными возможностями и имеют более высокий интеллектуальный уровень.

ЭВМ пока еще не появившегося пятого поколения должны стать интеллектуальными системами (ИС).

Что вообще понимается под *интеллектуальной системой*?

Это понятие возникло как результат обобщения моделей поведения биологических и искусственных систем, проявляющих способность к активному восприятию и избирательному запоминанию информации с целью формирования модели собственного поведения (или принятия решения).

Любая интеллектуальная система должна обладать набором следующих интеллектуальных качеств:

- свобода выбора цели и способа достижения этой цели;
- возможность корректировать свое поведение при изменении обстоятельств;
- способность сохранять свои функциональные свойства при изменении окружающей среды.

Сложившееся в настоящее время представление о функциональных средствах интеллектуальных систем предполагает наличие функциональных механизмов (процессоров) и механизмов памяти (баз знаний).

Под *знаниями* понимается любая информация, хранящаяся в системе и используемая ею для выполнения функционального назначения.

В принципе *знания* – это структурированные данные, относящиеся к различным предметным применениям, а также процедурно заданные способы (*алгоритмы*) поведения системы.

Очевидно, что *уровень интеллекта* во многом определяется содержательностью представленных в системе знаний, которые разделяются на [2]:

- интерфейсные;
- прикладные;
- системные.

Интерфейсные знания – это знания о принципах взаимодействия ЭВМ с внешней средой.

Они трактуют генетический аспект развития системы и применительно к ЭВМ включают знания:

- о языках общения (лексика, синтаксис);
- о способах коммуникации (диалог);
- о формах представления информации;
- о средствах общения.

Прикладные знания – это формализованные знания (информация), содержащая смысловые понятия, отношения, образы поведения, а также способы формирования новых знаний.

Применительно к ЭВМ это процедурные знания об умениях (модели типовых решений, способы доказательств, способы преобразования информации и т.д.), а также проблемные данные о внешней среде.

Наличие этих знаний определяет во многом интеллектуальную способность системы.

Системные знания – это знания о самой системе (ресурсы, состояние, способы доступа, алгоритмы управления, контроля и т.д.).

Функциональные механизмы (процессоры) включают средства (подсистемы) общения, логического вывода, решения задач.

Подробнее эти механизмы будут рассмотрены в п. 5.3.3 данного пособия.

Для определения системы как интеллектуальной необходимо и достаточно наличия всех трех типов знаний: интерфейсных, прикладных и системных.

1.2. Эволюция интеллектуальных свойств ЭВМ

Если представить процесс эволюции ЭВМ как повышение уровня ее интеллекта, то можно по-другому посмотреть на этапы развития и поколения ЭВМ, начиная от первых решающих устройств до вычислительных систем (ВС) обозримого будущего [2, 9].

Цель, преследуемая развитием средств вычислительной техники (СВТ), – это прежде всего повышение производительности интеллектуального труда за счет (или путем) увеличения доли машинного труда в общем процессе постановки и решения задач на ЭВМ, т.е. повышение ее интеллектуальных способностей.

Особенностью интеллектуальной системы является участие человека в процессе переработки информации.

Уровень интеллектуальности системы как человеко-машинной системы можно определить отношением времени решения задачи человеком ко времени решения этой же задачи в человеко-машинной системе [2].

$$\eta = \frac{\tau}{\alpha + (1 - \alpha) \cdot \tau}, \quad (1.1)$$

где τ – показатель относительной производительности ЭВМ (отношение времени решения некоторой типовой задачи человеком ко времени решения этой же задачи ЭВМ);

α – вес машинных операций в общем процессе решения задачи.

Очевидно, что только рост производительности ЭВМ не приводит к значительному росту показателя интеллектуальности η , если α мало. Отметим, что с развитием ЭВМ величина α постоянно увеличивалась. В этом можно убедиться, если посмотреть тенденции в изменении разделения труда между человеком и машиной при решении задач на традиционных ЭВМ общего назначения (табл. 1.2).

Интеллектуализация ЭВМ – это устойчивая тенденция в развитии средств вычислительной техники (СВТ). Все более сложные процедуры передаются от человека к ЭВМ [2, 9].

В табл. 1.3 показана классификация поколений ЭВМ с точки зрения развития свойств поколений, определяющих уровень их интеллекта.

Таблица 1.2

Разделение труда между человеком и машиной

Исполнитель работ	Выполняемые процедуры	Используемые знания	Изменения в разделении труда
Пользователь ЭВМ ↓ Системный аналитик ↓ Прикладной программист ↓ Системный программист ↓ Программист-кодировщик ↓ Оператор ввода-вывода ↓ Транслятор и редактор ↓ Решательный интерпретатор	Формирование условия задачи, цели, ТТ, ожидаемые результаты Выбор алгоритма решения, спецификация структуры и функций программных модулей Разработка логической структуры программных модулей, подготовка текста программы Привязка программ к операционной среде, спецификация процессов и ресурсов Перенос программы на носители для ввода в ЭВМ, проверка информации Контроль ввода и выполнение заданий, сервисные функции Преобразование текста программы, привязка к ресурсам ЭВМ Исполнение программы, редактирование и вывод результатов	Проблемные знания постановки подобных задач Процедурные знания о вычислительных методах и типовых алгоритмах Процедурные знания о языках программирования, проблемные знания о предметной области Системные знания о ресурсах системы и средствах управления процессами Интерфейсные знания о правилах кодировки Системные знания о средствах ввода-вывода и принципах управления процессами обработки Интерфейсные знания о входных языках, принципах кодирования информации Процедурные знания о стандартных функциях и процедурах обработки	<div> <div>ЧЕЛОВЕК</div> <div>6</div> <div>5</div> <div>4</div> <div>3</div> <div>2</div> <div>1</div> <div>МАШИНА</div> </div>

Таблица 1.3

Поколения ЭВМ в зависимости от свойств, определяющих уровень
их интеллекта

Определяющие свойства поколений ЭВМ	Поколения ЭВМ					
	1	2	3	4	5	6
Операционная среда:						
– искусственный «разум»	0	0	0	0	0	1
– интеллектуальная машина	0	0	0	0	1	х
– объектная машина	0	0	0	1	х	х
– виртуальная машина	0	0	1	х	х	х
– процедурная машина	0	1	х	х	х	х
– реальная машина	1	х	х	х	х	х
Виды знаний:						
– мета	0	0	0	0	0	1
– общие	0	0	0	0	1	1
– проблемные	0	0	0	1	1	1
– системные	0	0	1	1	1	1
– интерфейсные	0	1	1	1	1	1
– процедурные	1	1	1	1	1	1
Языки общения:						
– естественные	0	0	0	0	0	1
– прикладные	0	0	0	0	1	1
– функциональные	0	0	0	1	1	х
– диалоговые	0	0	1	1	х	х
– процедурные	0	1	1	х	х	х
– машинные	1	1	х	х	х	х

Примечание. х – данное свойство является неопределяющим

2. АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЭВМ

2.1. Системы счисления, применяемые в ЭВМ

Системой счисления называется совокупность цифровых знаков и правил их записи, применяемых для однозначного изображения чисел.

Системы счисления подразделяются на непозиционные и позиционные.

Непозиционными называются такие системы, в которых применяется неограниченное количество цифр, причем значение каждой цифры не зависит от позиции ее в числе. Например, римская система. Эти системы неудобны и применяются редко.

Позиционными называются системы, в которых применяется ограниченный набор символов (цифр), причем значение каждой из них находится в строгой зависимости от ее позиции в числе.

Количество различных цифр, применяемых в данной системе, называется ее *основанием*. Например, десятичная система $\{0,1,2, \dots, 9\}$.

Любое число в позиционной системе можно представить в следующем виде:

$$x = \sum_{i=-K}^{N-1} x_j \cdot 10^i \quad (2.1)$$

где x_j – некоторая цифра этой системы;

N – количество разрядов в целой части числа;

K – количество разрядов в дробной части числа (после точки);

10^i – разрядный вес некоторой цифры;

10 – основание системы счисления.

Величиной основания может быть любое другое число.

В ЭВМ применяются следующие позиционные системы счисления:

двоичная – используются 2 цифры: $\{0,1\}$;

восьмеричная – используются 8 цифр: $\{0,1, 2, \dots, 7\}$;

шестнадцатеричная – используются 16 цифр: $\{0,1, 2, \dots, 9, A, B, C, D, E, F\}$;

десятично-двоичная – используются 10 цифр: $\{0,1, 2, \dots, 9\}$, но каждая цифра представлена 4-мя двоичными разрядами.

Основной системой счисления, применяемой в ЭВМ на физическом уровне, является двоичная система. Этот выбор обоснован следующими факторами:

- простота технической реализации;
- наибольшая помехоустойчивость кодирования цифр;
- минимум оборудования;
- простота арифметических действий;
- простота формального аппарата для анализа и синтеза цифровых устройств (алгебра логики).

Для перевода чисел из одной позиционной системы счисления в другую применяются табличный и расчетный методы.

В первом случае используется таблица, содержащая прямое соответствие чисел в различных системах счисления.

При *расчетном* методе используются разные правила перевода целых и дробных чисел.

Для перевода целого числа в новую систему счисления его надо последовательно делить на основание новой системы счисления до тех пор, пока не получится частное, у которого целая часть равна 0. Число в новой системе записывается из остатков от последовательного деления, причем последний остаток будет старшей цифрой нового числа.

Например, необходимо перевести из десятичной системы счисления в двоичную целое число $x = 119$.

Производим последовательное деление исходного числа:

$$\begin{array}{r}
 119 \overline{) 2} \\
 \underline{118} \quad 59 \overline{) 2} \\
 1 \quad \underline{58} \quad 29 \overline{) 2} \\
 \quad \quad 1 \quad \underline{28} \quad 14 \overline{) 2} \\
 \quad \quad \quad 1 \quad \underline{14} \quad 7 \overline{) 2} \\
 \quad \quad \quad \quad 0 \quad \underline{6} \quad 3 \overline{) 2} \\
 \quad \quad \quad \quad \quad 1 \quad \underline{2} \quad 1 \overline{) 2} \\
 \quad \quad \quad \quad \quad \quad 1 \quad \underline{0} \quad 0 \\
 \quad \quad \quad \quad \quad \quad \quad 1 \text{ – старшая цифра}
 \end{array}$$

Стрелкой показан порядок записи числа в новой системе счисления.

Ответ: $x = 119_{(10)} = 1110111_{(2)}$.

Для перевода правильной дроби из одной позиционной системы счисления в другую ее (дробь) надо последовательно умножать на основание

новой системы счисления до тех пор, пока в новой дроби не будет нужного количества цифр, которое определяется требуемой точностью представленной дроби.

Например, необходимо перевести из десятичной системы в двоичную правильную дробь $x = 0,375_{(10)}$, ограничиваясь четырьмя значащими цифрами.

Производим последовательное умножение:

$$\begin{array}{r}
 0,375 \\
 \times \underline{2} \\
 0,750 \\
 \times \underline{2} \\
 1,500 \\
 \times \underline{2} \\
 1,000
 \end{array}
 \begin{array}{c}
 | \\
 \downarrow
 \end{array}$$

Стрелка показывает порядок записи дроби в новой системе.

Ответ: $x = 0,375_{(10)} = 0,0110_{(2)}$.

Правильная дробь в новой системе счисления записывается из целых частей произведений, получающихся при последовательном умножении (причем первая целая часть будет старшей цифрой новой дроби).

Если на некотором шаге дробная часть становится равной нулю, процесс преобразования заканчивается.

При переводе в новую систему счисления неправильных дробей целую часть переводят как целые числа, дробную – как правильные дроби. При этом все действия производят в той системе счисления, из которой переводят в другую систему счисления.

Рассмотренные методы перевода из одной системы счисления в другую удобны, когда исходной системой счисления является десятичная. Если же перевод осуществляется из недесятичной системы, то ввиду ее непривычности для человека выполнение в ней арифметических действий затруднено. В этом случае для преобразования чисел можно воспользоваться формулой 2.1. При этом расчеты ведутся в новой системе счисления.

Для примера переведем из двоичной системы в десятичную число $x = 1110111_{(2)}$.

Запишем число x в виде суммы произведений степеней основания на соответствующую цифру по формуле (2.1) в десятичной системе счисления,

после чего произведем необходимые расчеты (умножение и сложение полученных произведений):

$$x = 1110111_{(2)} = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = \\ = 64 + 32 + 16 + 0 + 4 + 2 + 1 = 119.$$

Ответ: $x = 1110111_{(2)} = 119_{(10)}$.

Аналогично можно перевести из двоичной системы в десятичную и правильную дробь, например:

$$x = 0,011_{(2)} = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = 0,375_{(10)}.$$

2.2. Формы представления чисел в ЭВМ

Для представления чисел в ЭВМ применяется две формы:

- естественная, или форма представления чисел с фиксированной точкой (ФТ);
- нормальная, или форма представления чисел с плавающей точкой (ПТ).

В *естественной* форме представления число записывается в соответствии с выражением (2.1) только при помощи набора значащих цифр без явного указания их весов и знаков сложения между ними.

В *нормальной* форме число представляется как произведение некоторой целой степени основания системы счисления и цифровой части, являющейся правильной дробью. Показатель степени основания называется *порядком*, а цифровая часть – *мантиссой*. При записи числа достаточно указать только порядок и мантиссу, не фиксируя в явном виде основание системы. Например,

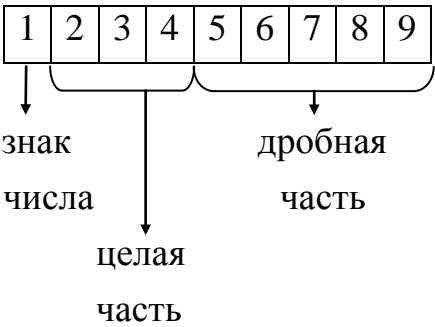

$$10^3 \cdot 0,118375 = \underbrace{3}_{\text{порядок}} \cdot \underbrace{0,118375}_{\text{знак мантиссы}}$$

Обе формы представления чисел имеют достоинства и недостатки [5]. В современных ЭВМ применяются обе формы.

Одним из очевидных преимуществ формы представления чисел с ПТ является значительно больший диапазон возможных представляемых чисел по сравнению с формой с ФТ при равном количестве разрядов. Предположим, ЭВМ имеет 9 десятичных разрядов (табл. 2.1).

Таблица 2.1

Пример представления числа с фиксированной и плавающей точкой

Показатели, характеристика	Число с ФТ	Число с ПТ
Назначение разрядов числа		
$+ x_{max} $	+ 999,99999	$+ 0,99999 \cdot 10^{+99}$
$+ x_{min} $	+ 000,00001	$+ 0,10000 \cdot 10^{-99}$
$- x_{max} $	- 999,99999	$- 0,99999 \cdot 10^{+99}$
$- x_{min} $	- 000,00001	$- 0,10000 \cdot 10^{-99}$

При работе над числами с ФТ в ЭВМ могут возникнуть два нежелательных результата:

- 1) $|x_{\text{маш ФТ}}| < |x_{\text{min}}|$ – обнуление,
- 2) $|x_{\text{маш ФТ}}| > |x_{\text{max}}|$ – переполнение разрядной сетки, т.е. теряются старшие разряды.

Для устранения этих случаев применяют *масштабирование*, которое заключается в замене чисел, участвующих в операции, произведениями чисел на так называемые *масштабные коэффициенты*:

$$x_{\text{маш}} = m \cdot x,$$

где m – масштабный коэффициент (МК).

Цель масштабирования заключается в том, чтобы во время вычислений не возникло переполнения разрядной сетки ни в промежуточных, ни в конечных результатах.

Выбор МК зависит от вида конкретных чисел и вида операции (табл. 2.2).

Определение масштабных коэффициентов

Вид операции	Значение МК
Сложение- вычитание	$ x_3/_{масш} = m_3 x_3 = m_1 x_1 + m_2 x_2,$ но т.к. $x_3 = x_1 + x_2$, находим, что должно выполняться условие $m_3 = m_1 = m_2 = m_{общ}$
Умножение	$ x_3/_{масш} = m_3 x_3 = (m_1 x_1) \cdot (m_2 x_2),$ учитывая, что $x_3 = x_1 \cdot x_2$, получим $m_3 = m_1 \cdot m_2$
Деление	$ x_3/_{масш} = m_1 x_1 / m_2 x_2,$ зная, что $x_3 = x_1 / x_2$, получим $m_3 = m_1 / m_2$

Нетрудно увидеть, что процесс подборки МК является сложным и трудоемким.

Обычно МК выбирают таким образом, чтобы он имел вид целой степени основания системы счисления, т.е.

$$m = 2^P,$$

где P – некоторое целое число.

Это облегчает выравнивание МК.

Для облегчения процесса масштабирования применяют *предварительное масштабирование*. Обычно используют 2 вида предварительного масштабирования:

1) точка фиксируется перед старшим цифровым разрядом (дробные числа):

$$[-1 < |x_{масш}| < 1];$$

2) точка фиксируется после младшего разряда (целые числа):

$$-2^n < |x_{масш}| < 2^n.$$

Точность представления информации в ЭВМ

При записи чисел в ограниченную разрядную сетку возникает ошибка округления. Если округление сделано правильно, то ошибка не должна превышать $\frac{1}{2}$ веса самого младшего разряда. Величина ошибки зависит от формы представления чисел, вида чисел и количества разрядов (табл. 2.3).

Таблица 2.3

Точность представления чисел в ЭВМ

Вид ошибки	Форма ФТ		Форма ПТ
	Дробные числа	Целые числа	
Абсолютная ошибка	$[\Delta_{\Phi T}] = 1/2 \cdot 2^{-n} = \text{const}$	$[\Delta_{\Phi T}] = 1/2 = \text{const}$	$[\Delta_{ПТ}] = 1/2 \cdot 2^{-n} \cdot 2^p = \text{var}, (*)$ где n – количество значащих цифр в мантиссе; p – порядок числа. Из выражения (*) видно, что $[\Delta_{ПТ}]$ изменяется (варьируется) в пределах от $[\Delta_{ПТ \min}] = 1/2 \cdot 2^{-n} \cdot 2^{- P_{\max} }$ до $[\Delta_{ПТ \max}] = 1/2 \cdot 2^{-n} \cdot 2^{ P_{\max} }$
Относительная ошибка	$\delta_{\Phi T \min} = (1/2 \cdot 2^{-n}) / (1 \cdot 2^{-n}) \approx 2^{-(n+1)}$ при $1 \gg 2^{-n}$; $\delta_{\Phi T \max} = (1/2 \cdot 2^{-n}) / 2^{-n} = 1/2 = 50 \%$	$\delta_{\Phi T \min} = (1/2) / (2^n - 1) \approx 2^{-(n+1)}$ при $2^n \gg 1$; $\delta_{\Phi T \max} = (1/2) / 1 = 1/2 = 50 \%$	$\delta_{ПТ \min} = \frac{ \Delta_{ПТ \max} }{ x_{ПТ \max} } \approx 2^{-(n+1)}$; $\delta_{ПТ \max} = \frac{ \Delta_{ПТ \min} }{ x_{ПТ \min} } \approx 2^{-n}$

Из табл. 2.3 видно, что:

1) точность записи чисел с ФТ в ограниченную разрядную сетку не зависит от величины масштабного коэффициента и способа масштабирования;

2) в ЭВМ с ПТ ошибка округления не зависит от величины порядка чисел, а зависит от количества разрядов мантиссы n .

2.3. Кодирование чисел в ЭВМ

В ЭВМ применяются 3 вида кодирования чисел:

- прямой код (ПК);
- дополнительный код (ДК);
- обратный код (ОК).

Особенности представления чисел в указанных кодах приведены в табл. 2.4.

Таблица 2.4

Виды кодирования чисел в ЭВМ

Знак числа x	Значение числа		
	в прямом коде $[x]_{пр}$	в дополнительном коде $[x]_{доп}$	в обратном коде $[x]_{обр}$
$x \geq 0$	x	x	x
$x < 0$	$10^N + x $	$x_{гр} - x $	$x_{max} - x $

В табл. 2.4 применены следующие обозначения:

x – значение числа,

N – количество разрядов в целой части числа,

10 – основание системы счисления (но не число «10»),

$x_{гр}$ – граничное число $= |x_{max}| + 10^{-k} = 10^N$,

x_{max} – максимальное число $= |x_{max}| = 10^N - 10^{-k}$,

k – количество разрядов в дробной части числа.

Как видно из табл. 2.4, положительные числа во всех кодах записываются одинаково.

Для перевода отрицательного числа в *дополнительный код* необходимо:

- записать в знаковом разряде 1 (–);
- во всех других разрядах цифры заменить на взаимно обратные;
- добавить «1» к младшему разряду.

Для перевода отрицательного числа в *обратный код* необходимо выполнить 2 первых шага:

- записать в знаковом разряде 1 (–);
- заменить цифры во всех остальных разрядах на взаимно обратные.

Применение дополнительного или обратного кода позволяет:

- заменить операцию вычитания на операцию сложения;
- автоматически определить знак результата, так как знаковые разряды также участвуют в операции сложения. При этом при возникновении переноса при сложении знаковых разрядов при использовании дополнительного кода единица переноса отбрасывается, а при использовании обратного кода единица переноса из знакового разряда циклически добавляется к младшему разряду суммы.

2.4. Выполнение арифметических операций в ЭВМ

2.4.1. Выполнение операций сложения/вычитания чисел, представленных в форме с фиксированной точкой

Алгебраическое сложение чисел с ФТ в ЭВМ может производиться в одном из трех кодов: прямом (ПК), дополнительном (ДК) или обратном (ОК). Результат получается в этих же кодах.

Чаще всего используются дополнительный и обратный коды. При этом знаковый разряд и цифровая часть числа рассматриваются как единое целое, в результате чего с отрицательными числами ЭВМ оперирует как с положительными, независимо от того, в каком виде они представлены – правильных дробей или целых чисел.

Основные достоинства дополнительного и обратного кодов рассмотрены в подразд. 2.3 «Кодирование чисел в ЭВМ».

Суммирование двоичных чисел производится в соответствии с таблицей сложения:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = \underline{1}0, 1 \text{ переносится для сложения в старший разряд.}$$

Фактически выполняется сложение трех чисел – операндов и переноса.

Алгебраическое сложение многоразрядных чисел обычно организуется как регулярный процесс, состоящий из n одинаковых операций поразрядного сложения-вычитания (где n – количество разрядов в каждом операнде).

Возможны 4 случая (табл. 2.5).

Таблица 2.5

Алгебраическое сложение чисел

	x_1	x_2	$x_3 = x_1 + x_2$
1	> 0	> 0	> 0
2	> 0	< 0	> 0
3	> 0	< 0	< 0
4	< 0	< 0	< 0

Пример выполнения операции сложения/вычитания чисел с ФТ, представленных в двоичной системе счисления, с применением трех видов кодирования, приведен в табл. 2.6.

Таблица 2.6

Пример сложения/вычитания чисел в форме с ФТ

Обычная запись	Код		
	прямой	дополнительный	обратный
<u>1-й случай: $x_1 > 0, x_2 > 0, x_3 > 0$</u>			
$x_1 = 0,10101$	$[x_1]_{\text{пр}} = 0,10101$	$[x_1]_{\text{доп}} = 0,10101$	$[x_1]_{\text{обр}} = 0,10101$
$x_2 = 0,00101$	$[x_2]_{\text{пр}} = 0,00101$	$[x_2]_{\text{доп}} = 0,00101$	$[x_2]_{\text{обр}} = 0,00101$
$x_3 = 0,11010$	$[x_3]_{\text{пр}} = 0,11010$	$[x_3]_{\text{доп}} = 0,11010$	$[x_3]_{\text{обр}} = 0,11010$
<u>2-й случай: $x_1 > 0, x_2 < 0, x_3 > 0$</u>			
$x_1 = 0,10101$	$[x_1]_{\text{пр}} = 0,10101$	$[x_1]_{\text{доп}} = 0,10101$	$[x_1]_{\text{обр}} = 0,10101$
$x_2 = -0,00101$	$[x_2]_{\text{пр}} = 1,00101$	$[x_2]_{\text{доп}} = 1,11011$	$[x_2]_{\text{обр}} = 1,11010$
$x_3 = 0,10000$	$[x_3]_{\text{пр}} = 0,10000$ (вычитание)	$[x_3]_{\text{доп}} = 10,10000$ (1 отбрасывается) = 0,10000	$[x_3]_{\text{обр}} = 10,01111$ (1 добавл.) +1 = = 0,10000
<u>3-й случай: $x_1 < 0, x_2 > 0, x_3 < 0$</u>			
$x_1 = -0,10101$	$[x_1]_{\text{пр}} = 1,10101$	$[x_1]_{\text{доп}} = 1,01011$	$[x_1]_{\text{обр}} = 1,01010$
$x_2 = 0,00101$	$[x_2]_{\text{пр}} = 0,00101$	$[x_2]_{\text{доп}} = 0,00101$	$[x_2]_{\text{обр}} = 0,00101$
$x_3 = -0,10000$	$[x_3]_{\text{пр}} = 1,10000$	$[x_3]_{\text{доп}} = 1,10000$ преобраз. кода $[x_3]_{\text{пр}} = 1,10000$	$[x_3]_{\text{обр}} = 1,01111$ преобраз. кода $[x_3]_{\text{пр}} = 1,10000$
<u>4-й случай: $x_1 < 0, x_2 < 0, x_3 < 0$</u>			
$x_1 = -0,10101$	$[x_1]_{\text{пр}} = 1,10101$	$[x_1]_{\text{доп}} = 1,01011$	$[x_1]_{\text{обр}} = 1,01010$
$x_2 = -0,00101$	$[x_2]_{\text{пр}} = 1,00101$	$[x_2]_{\text{доп}} = 1,11011$	$[x_2]_{\text{обр}} = 1,11010$
$x_3 = -0,11010$	$[x_3]_{\text{пр}} = 1,11010$	$[x_3]_{\text{доп}} = 11,00110$ (1 отбрасывается) преобраз. кода $[x_3]_{\text{пр}} = 1,11010$	$[x_3]_{\text{обр}} = 11,00100$ (1 добавл.) +1 = = 1,00101 преобраз. кода $[x_3]_{\text{пр}} = 1,11010$

Переполнение разрядной сетки

При выполнении операций сложения/вычитания может возникнуть *переполнение разрядной сетки*. На практике одним из наиболее

распространенных методов выявления переполнения является метод, основанный на применении так называемых модифицированных кодов.

Модифицированные коды отличаются от обычных тем, что у них знаки изображаются двумя цифрами:

00 – положительное число (+);

11 – отрицательное число (–).

01 и 10 – запрещенные комбинации.

Модифицированные коды используются для простоты индикации переполнения разрядной сетки. В основе метода выявления переполнения лежит анализ сочетания цифр переносов, поступающих в знаковый разряд и выходящих из знакового разряда. Можно составить вспомогательную таблицу соответствия режимов работы сумматора и сочетаний указанных цифр переносов (табл. 2.7).

Таблица 2.7

Таблица переносов

Режим работы ЭВМ	Сочетание цифр переносов	
	из знакового разряда	в знаковый разряд
1-й случай: $x_1 > 0, x_2 > 0, x_3 > 0$		
Нормальная работа	0	0
Работа с переполнением	0	1
2-й случай: $x_1 > 0, x_2 < 0, x_3 > 0$		
Нормальная работа	1	1
Работа с переполнением	невозможна	
3-й случай: $x_1 < 0, x_2 > 0, x_3 < 0$		
Нормальная работа	0	0
Работа с переполнением	невозможна	
4-й случай: $x_1 < 0, x_2 < 0, x_3 < 0$		
Нормальная работа	1	1
Работа с переполнением	1	0

2.4.2. Выполнение операций умножения чисел, представленных в форме с фиксированной точкой

Операция умножения чисел с ФТ выполняется в 2 этапа:

1) определение знака произведения при помощи сложения знаковых разрядов по модулю 2:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0 ;$$

2) перемножение сомножителей последовательным сложением частичных произведений со сдвигом (как это выполняется в десятичной системе счисления).

Таблица умножения:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1.$$

Обычно для выполнения операции умножения применяются 2 способа, различающиеся тем, с какого разряда множителя (с младшего или со старшего) начинается умножение множимого на множитель.

Операция умножения является одной из самых весомых операций в ЭВМ. Применяются различные способы ускорения операции умножения – аппаратные (обеспечивающие совмещение во времени тех или иных операций, умножение на 2 – 4 разряда одновременно и др.) и логические (за счет усложнения схем управления). Подробнее с этими методами можно познакомиться в [5].

Рассмотрим один из примеров выполнения операции умножения чисел с ФТ.

Пример 2.1

Предположим, необходимо перемножить числа $[x_1]_{\text{ПР}} = 0,1010_{(2)}$ и $[x_2]_{\text{ПР}} = 1,1101_{(2)}$.

1 этап: Определяем знак произведения.

$$0 \oplus 1 = 1$$

2 этап: Перемножим множители.

Порядок перемножения определяется нумерацией цифр сомножителя.

1-й способ

Начиная с младшей цифры

$$\begin{array}{r}
 1010 = |x_1| \\
 \underline{1101} = |x_2| \\
 \text{номера цифр} - (4321) \\
 \begin{array}{r}
 1010 \\
 0000 \\
 1010 \\
 \underline{1010} \\
 1.10000010
 \end{array}
 \end{array}
 \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \begin{array}{l} \text{частичные} \\ \text{произведения} \end{array}$$

знак

2-й способ

Начиная со старшей цифры

$$\begin{array}{r}
 1010 \\
 \underline{1101} \\
 (1234) - \text{номера цифр} \\
 \begin{array}{r}
 1010 \\
 1010 \\
 0000 \\
 \underline{1010} \\
 1.10000010
 \end{array}
 \end{array}
 \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \begin{array}{l} \text{частичные} \\ \text{произведения} \end{array}$$

знак

Как видим, результаты совпадают при выполнении операции умножения обоими способами.

Отметим, что в ЭВМ выполнение операции умножения имеет свою особенность. В ЭВМ нельзя сразу просуммировать n частичных произведений, поэтому производится последовательное накопление суммы частичных произведений. При этом возможны 2 способа:

- производится сдвиг множимого на требуемое количество разрядов (влево или вправо) и добавление полученного таким образом частичного произведения к ранее накопленной сумме;

- производится сдвиг ранее накопленной суммы частичных произведений на каждом шаге на 1 разряд вправо (или влево) и последовательным добавлением к сдвинутой сумме частичных произведений несдвинутого множителя.

Если сомножители хранятся в памяти ЭВМ не в прямом коде, то и умножение производится в ДК или ОК. Предпочтительнее умножение выполнять в ДК. Возможны 4 случая:

$$1) x_1 > 0; x_2 > 0; x_3 = |x_1|/|x_2| = |x_1 x_2|_{\text{дон}} = |x_1|_{\text{дон}} |x_2|_{\text{дон}};$$

$$2) x_1 > 0; x_2 < 0; |x_1|_{\text{дон}} / |x_2|_{\text{дон}} = |x_1| / (10^N - |x_2|) = 10^N |x_1| - |x_1|/|x_2|.$$

Это (модуль псевдопроизведения) значительно отличается от модуля кода истинного произведения

$$|x_1 x_2|_{\text{дон}} = \begin{cases} 1 - (|x_1| |x_2|) - \text{для дробных чисел;} \\ 10^{2N} - (|x_1|/|x_2|) - \text{для целых чисел.} \end{cases}$$

Для получения модуля дополнительного кода истинного произведения $|x_1 x_2|_{\text{дон}}$ необходимо к псевдопроизведению добавить величину

$\Delta_2 = 1 - |x_1|$ – для дробных чисел;

$\Delta_2 = 10^N (10^N - |x_1|)$ – для целых чисел.

3) $x_1 < 0; x_2 > 0$.

$\Delta_3 = 1 - |x_2|$ – для дробных чисел;

$\Delta_3 = 10^N (10^N - |x_2|)$ – для целых чисел;

4) $x_1 < 0; x_2 < 0$.

$\Delta_4 = |x_1| + |x_2| - 1$ – для дробных чисел;

$\Delta_4 = 10^N (|x_1| + |x_2| - 10^N)$ – для целых чисел.

Пример 2.2

Перемножить числа $[x_1] = 0,1010$ и $[x_2] = 1,1101$ в дополнительном коде.

$[x_1]_{\text{доп}} = 0,1010$ и $[x_2]_{\text{доп}} = 1,0011$.

Знак произведения равен

$0 \oplus 1 = 1$.

В результате перемножения сомножителей в дополнительном коде получим

$$\begin{array}{r} .1010 \\ \underline{.0011} \\ 1010 \\ 1010 \\ 0000 \\ \underline{0000} \\ 1.0011110 \end{array}$$

Как видим, результат перемножения модулей чисел в дополнительном коде не совпадает с результатом перемножения этих чисел в прямом коде (пример 2.1).

Необходима корректировка результата на величину:

$\Delta_2 = 1 - |x_1| = 0,01100000$, т.к. $x_1 > 0, x_2 < 0$.

Просуммируем значения псевдопроизведения и Δ_2 :

$$\begin{array}{r} .00011110 \\ .01100000 \end{array}$$

$x_3 = 1.01111110$ – в дополнительном коде.

знак

Переведем x_3 в прямой код.

$$[x_3]_{\text{пр}} = 1.10000001 + 1 = 1.10000010.$$

Это совпадает с результатом перемножения этих чисел в прямом коде (пример 2.1). Отметим, что при умножении дробных чисел с ФТ невозможно переполнение разрядной сетки.

2.4.3. Выполнение операций деления чисел, представленных в форме с фиксированной точкой

При выполнении деления дробных чисел с ФТ в отличие от умножения возможно переполнение разрядной сетки. Поэтому необходимо следить, чтобы делимое по абсолютной величине было меньше делителя.

Признаком переполнения является «1» в знаковом разряде модуля частного, поскольку обычно операция деления производится только над модулями исходных чисел. Операция деления выполняется в 2 этапа:

- 1) определение знака частного при помощи операции \oplus над знаками чисел;
- 2) деление модулей исходных чисел, затем округление, присвоение знака.

При выполнении операции деления в двоичной системе счисления (как и в десятичной) процесс деления сводится к многократному вычитанию сначала из делимого, а затем из остатков, умноженных на основание системы счисления (приписывание нуля справа), произведений делителя на некоторую цифру частного.

Особенность процесса деления в двоичной системе счисления состоит в том, что произведение делителя на любую цифру системы счисления (0 или 1) в этом случае равно либо 0, либо самому делителю.

Так как вычитание заменяется сложением, то можно сделать вывод, что деление в двоичной системе счисления сводится к двум микрооперациям: сложению некоторых чисел и их сдвигу.

В ЭВМ процесс деления выполняется следующим образом:

- 1) производится сравнение модулей делимого и делителя, а затем – остатков и делителя при помощи операции вычитания. Соотношение сравниваемых величин определяется по знаку разности;
- 2) производится умножение делимого, а затем – остатков на основание системы счисления сдвигом исходного числа влево на 1 разряд.

В ЭВМ применяется в основном 2 способа деления:

- 1) с восстановлением остатка;
- 2) без восстановления остатка.

Рассмотрим второй способ:

– чтобы определить цифру частного в некотором разряде, необходимо сдвинуть логически предыдущий остаток влево на один разряд, а затем алгебраически прибавить к нему модуль делителя, которому присваивается знак, противоположный знаку предыдущего остатка;

– если полученный текущий остаток положителен, то в частном проставляется 1, а если остаток отрицателен, то – 0.

Микрооперации сдвигов и алгебраических сложений повторяются до тех пор, пока в частном не получится требуемое количество цифр.

Пример 2.3

Разделить число $[x_1] = 0,101$ на $[x_2] = 0,110$.

Решение выполняется в два этапа:

1-й этап – определение знака частного: $0 \oplus 0 = 0$;

2-й этап – деление модулей $|x_1|/|x_2|$.

Таблица 2.8

Пример деления чисел с ФТ

№ шага	Арифметические действия	Цифры модуля частного	Пояснения
0	$\begin{array}{r} 0.101 \\ \underline{1.010} \text{ (доп. код)} \\ 1.111 \end{array}$	$\begin{array}{r} 0.110 \\ \underline{0.1101} \end{array}$	0-е сложение 0-й остаток
1	$\begin{array}{r} 1.110 \\ \underline{0.110} \text{ (прямой код)} \\ 0.100 \end{array}$		1-й сдвиг 1-е сложение 1-й остаток
2	$\begin{array}{r} 1.000 \\ \underline{1.010} \text{ (доп. код)} \\ 0.010 \end{array}$		2-й сдвиг 2-е сложение 2-й остаток
3	$\begin{array}{r} 0.100 \\ \underline{1.010} \text{ (доп. код)} \\ 1.110 \end{array}$		3-й сдвиг 3-е сложение 3-й остаток
4	$\begin{array}{r} 1.100 \\ \underline{0.110} \\ 0.010 \end{array}$		4-й сдвиг 4-е сложение 4-й остаток
Ответ $[x_3] = x_1/x_2 = 0.1101 \approx 0.111$			

Выполнение операций деления чисел с ФТ, представленных в ДК или ОК, имеет свои особенности. Подробно это рассмотрено в [5].

2.4.4. Выполнение операций сложения/вычитания чисел, представленных в форме с плавающей точкой

Если имеется два числа, представленных в форме с ПТ,

$$x_1 = m_1 \cdot 10^{P_1} \text{ и } x_2 = m_2 \cdot 10^{P_2},$$

то для того чтобы их сложить, нужно предварительно привести их к одному общему порядку, т.е. преобразовать одно из слагаемых, например первое, следующим образом:

$$x_1 = m_1 \cdot 10^{P_1} = m_1^* \cdot 10^{P_{общ}}, P_2 = P_{общ},$$

$$\text{тогда } x_1 + x_2 = m_1^* \cdot 10^{P_{общ}} + m_2 \cdot 10^{P_{общ}} = (m_1^* + m_2) \cdot 10^{P_{общ}}.$$

Преобразовывать всегда нужно меньшее слагаемое, т.к. в противном случае произойдет переполнение разрядной сетки мантиссы преобразуемого числа.

Операция сложения двух чисел с ПТ выполняется в 4 этапа.

1. Выравнивание порядков:

- меньший порядок выравнивается до большего;
- мантисса сдвигается вправо на соответствующее количество разрядов.

Практически в ЭВМ производится вычитание порядков операндов. Знак и модуль разности $P_1 - P_2$ определяют соответственно, какое из слагаемых нужно преобразовать и на сколько порядков (единиц).

2. Производится преобразование мантисс в один из модифицированных кодов.

3. Мантиссы слагаемых суммируются по правилам сложения дробных чисел, в форме с ФТ.

4. В случае надобности мантисса переводится в прямой код и производится нормализация суммы, округление мантиссы.

На последнем этапе некоторые действия могут отсутствовать, например перекодировка, если в памяти числа хранятся в ДК.

Пример 2.4

Используя модифицированный дополнительный код, сложить 2 числа:

$$[x_1]_{np} = 0.101; \quad 1.10101;$$

$$[x_2]_{np} = 0.100; \quad 1.11001.$$

Решение:

1-й этап $[x_2]_{np}^* = 0.101$; 1.011001 – выравнивание порядка x_2 до x_1 ;

2-й этап $[m_1]_{don} = 11.01011$; $[m_2]_{don} = 11.100111$;

3-й этап $[m_1]_{don} = 11.010110$

$$[m_2]_{don} = 11.100111$$

$$[m_3]_{don} = 110.111101.$$

В знаковых цифрах мантиисы **1** отбрасывается, а комбинация цифр 10 показывает, что сумма денормализована влево на один разряд.

4-й этап – нормализация суммы (сдвиг вправо на один разряд):

$$[m_3]_{don} = 10.111101 \rightarrow 1.0111101;$$

$$P_{общ} = 0.101 + 0.001 = 0.11.$$

Переводим сумму в прямой код и производим округление мантиисы до пяти разрядов.

$$\text{Ответ: } [x_3]_{пр} = 0.110; \quad 1.1000011 \approx 0.110; \quad 1.10001.$$

порядок мантииса.

2.4.5. Выполнение операций умножения/деления чисел, представленных в форме с плавающей точкой

Предположим, что надо перемножить два числа

$$x_1 = m_1 \cdot 10^{P_1} \text{ и } x_2 = m_2 \cdot 10^{P_2}.$$

Произведение этих чисел будет равно

$$x_3 = x_1 \cdot x_2 = m_1 \cdot m_2 \cdot 10^{P_1+P_2}.$$

Операция умножения чисел с ПТ выполняется в 4 этапа:

1) определение знака произведения (при помощи операции \oplus знаков мантиис);

2) перемножение мантиис по правилам умножения дробных чисел, представленных в форме с ФТ;

3) определение порядка произведения при помощи операции сложения порядков по правилам сложения дробных чисел с ФТ;

4) нормализация результата и округление мантиисы произведения (поскольку мантиисы сомножителей нормализованные числа, то возможна денормализация не более чем на 1 разряд).

Предположим далее, что надо разделить два числа:

$$x_1 = m_1 \cdot 10^{P_1} \text{ и } x_2 = m_2 \cdot 10^P.$$

Частное от деления этих чисел будет равно

$$x_3 = x_1 / x_2 = (m_1 / m_2) \cdot 10^{P_1 - P_2}.$$

Операция деления чисел с ПТ выполняется также в 4 этапа следующим образом:

- 1) определение знака (операция \oplus над знаками мантисс операндов);
- 2) деление модуля мантиссы делимого на модуль мантиссы делителя по правилам деления дробных чисел с ФТ;
- 3) нахождение порядка результат (путем вычитания порядков делимого и делителя по правилам вычитания дробных чисел с ФТ);
- 4) нормализация результата и округление.

3. ЛОГИЧЕСКИЕ ОСНОВЫ ЭВМ

3.1. Основные положения алгебры логики

Одна из причин широкого распространения ЭВМ заключается в том, что они могут решать не только арифметические, но и логические задачи.

Как известно, *логика* – это наука о законах и формах мышления.

Математическая логика занимается изучением возможностей применения формальных методов для решения логических задач. В ЭВМ применяется главным образом начальный раздел математической логики – *исчисление высказываний*, или *алгебра логики*. (Иногда алгебру логики называют *булевой алгеброй* по имени Д. Буля, разработавшего в XIX в. основные положения исчисления высказываний).

В настоящее время алгебра логики широко применяется при анализе и синтезе узлов ЭВМ, а также при машинном решении логических задач.

Начальным понятием алгебры логики является понятие *высказывание*. Под *высказыванием* понимается любое утверждение, которое оценивается только с точки зрения его истинности или ложности. Высказывание может быть либо *истинным*, либо *ложным* (истинно ложных или наоборот не бывает).

Исчисление высказываний основано на том, что каждое высказывание можно рассматривать как некоторую двоичную переменную, принимающую значение 1 или 0. В соответствии с двоичной природой высказываний условились называть их логическими, или булевыми, переменными.

Высказывания могут быть простыми или сложными. Высказывание называется *простым*, если его значение истинности не зависит от значения истинности других высказываний.

Значение истинности *сложного* высказывания зависит от значений истинности других высказываний. Следовательно, любое высказывание можно считать логической (булевой) функцией (ЛФ) некоторых двоичных аргументов – простых высказываний, входящих в его состав. Сложные высказывания, в свою очередь, могут служить аргументами еще более сложных логических функций, т.е. при построении логических функций справедлив принцип *суперпозиции*. Отсюда следует, что можно построить логическую функцию любой наперед заданной сложности, пользуясь ограниченным числом логических связей и принципом суперпозиции. При этом ограниченный набор логических связей,

обеспечивающий на основе принципа суперпозиции построение логических функций любой наперед заданной сложности, называется *функционально полным*.

3.2. Булевы функции основного функционально полного набора (ФПН)

Из множества возможных ФПН широкое распространение получил один, называемый *основным*.

В его состав входят 3 логические связи (функции):

- 1) *отрицание* (связь НЕ, логическая инверсия),
- 2) *конъюнкция* (связь И, логическое умножение),
- 3) *дизъюнкция* (связь ИЛИ, логическое сложение).

В алгебре логики принято логические функции изображать в виде таблиц истинности.

Рассмотрим особенности функций основного набора.

Отрицание – это такая логическая связь между аргументом x и функцией y , при которой y истинно только тогда, когда x ложно, и наоборот.

Таблица истинности функции НЕ

Аргумент x	0	1
Функция y	1	0

Обозначение функции НЕ: $y = \bar{x} = \neg x$.

Свойства функции НЕ:

- 1) $\overline{\bar{x}} = x$;
- 2) $\overline{\bar{z}} = y = \overline{\bar{x}} = \bar{x}$;
- 3) если $f_1 = f_2$, то $\bar{f}_1 = \bar{f}_2$.

Конъюнкцией n высказываний называется такое сложное высказывание y , которое истинно только тогда, когда истинны все входящие в него простые высказывания. В остальных случаях оно ложно.

Таблица истинности функции И для двух аргументов

1-й аргумент x_1	0	0	1	1
2-й аргумент x_2	0	1	0	1
Функция y	0	0	0	1

Обозначение функции И: $y = x_1 \wedge x_2 = x_1 \cdot x_2 = x_1 \& x_2$.

Свойства функции И:

1) $x \cdot x \cdot x \dots = x$;

2) $1 \cdot x_2 = x_2$;

3) $0 \cdot x_2 = 0$;

4) $x_i \cdot \bar{x}_i = 0$.

Дизъюнкцией n высказываний называется такое сложное высказывание y , которое ложно только тогда, когда ложны все входящие в него высказывания. В остальных случаях y истинно.

Таблица истинности функции ИЛИ для двух аргументов

1-й аргумент x_1	0	0	1	1
2-й аргумент x_2	0	1	0	1
Функция y	0	1	1	1

Обозначение функции ИЛИ: $y = x_1 \vee x_2 = x_1 + x_2$.

Свойства функции ИЛИ:

1) $x + x + x + \dots = x$;

2) $0 + x_2 = x_2$;

3) $1 + x_2 = 1$;

4) $x_i + \bar{x}_i = 1$.

В 1910 г. петербургский ученый Эренфест предложил одну из самых ранних интерпретаций логических высказываний. В ней была использована аналогия между высказываниями и электрическими переключателями:

«истина» – замкнуто – «1»;

«ложно» – разомкнуто – «0».

На основе принятой аналогии любую логическую функцию (ЛФ) можно изобразить в виде некоторой переключательной схемы (вот почему ЛФ иногда называют *переключательными функциями*).

Фактически эта аналогия используется и в современных ЭВМ, хотя переключательные схемы прошли большой путь эволюции:

- реле;

- лампа;
- полупроводник;
- интегральная схема.

Любая логическая функция может быть представлена в виде *переключательной схемы* (рис. 3.1).



Рис. 3.1

3.3. Основные законы алгебры логики и их следствия

В алгебре логики имеются 4 **основных закона**, приведенных в табл. 3.1.

Таблица 3.1

Основные законы алгебры логики

Наименование закона	Пример выражение закона для	
	конъюнкции	дизъюнкции
<i>Переместительный</i> (коммутативный)	$x_1 \cdot x_2 = x_2 x_1$	$x_1 + x_2 = x_2 + x_1$
<i>Сочетательный</i> (ассоциативный)	$x_1 \cdot x_2 \cdot x_3 = x_1 \cdot (x_2 \cdot x_3) =$ $= x_3 \cdot (x_1 \cdot x_2)$	$x_1 + x_2 + x_3 = x_1 + (x_2 + x_3) =$ $= x_3 + (x_1 + x_2)$
<i>Распределительный</i> (дистрибутивный)	Закон 1-го рода $(x_2 + x_3) \cdot x_1 = x_1 \cdot x_2 + x_1 \cdot x_3$	Закон 2-го рода $x_2 \cdot x_3 + x_1 = (x_1 + x_2) \cdot (x_1 + x_3)$
<i>Закон инверсии</i> (правило де Моргана)	$\overline{x_1 \cdot x_2 \cdot \dots x_n} = \bar{x}_1 + \bar{x}_2 + \dots + \bar{x}_n$	$\overline{x_1 + x_2 + \dots + x_n} = \bar{x}_1 \cdot \bar{x}_2 \cdot \dots \cdot \bar{x}_n$

Ниже приведены определения указанных законов алгебры логики.

Переместительный закон – от перемены мест слагаемых (сомножителей) сумма (произведение) не изменяется.

Сочетательный закон – при логическом сложении (логическом умножении) нескольких аргументов любую группу слагаемых (сомножителей) можно заменить их логической суммой (логическим произведением).

Распределительный закон 1-го рода – для логического произведения относительно логической суммы.

Произведение суммы нескольких аргументов на какую-нибудь логическую переменную равно сумме произведений каждого слагаемого на эту переменную.

$$(x_2 + x_3) \cdot x_1 = x_1 \cdot x_2 + x_1 \cdot x_3 - \text{практически это раскрытие скобок.}$$

Распределительный закон 2-го рода – для логической суммы относительно логического произведения.

Сумма некоторой логической переменной и произведения нескольких переменных равно произведению сумм каждого сомножителя и этой переменной:

$$x_2 \cdot x_3 + x_1 = (x_1 + x_2) \cdot (x_1 + x_3).$$

Доказательство:

$$\begin{aligned} (x_1 + x_2) \cdot (x_1 + x_3) &= x_1 \cdot x_1 + x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3 = (x_1 + x_1 \cdot x_2 + x_1 \cdot x_3) + x_2 \cdot x_3 = \\ &= x_1 \cdot 1 + x_2 \cdot x_3 = x_1 + x_2 \cdot x_3 = x_1 + x_2 \cdot x_3. \end{aligned}$$

Закон инверсии (см. табл. 3.1)

Доказательством справедливости этого закона являются:

- условия обращения в «0» для дизъюнкции: $\overline{x_1 + x_2 + \dots + x_n} = \bar{x}_1 \cdot \bar{x}_2 \cdot \dots \cdot \bar{x}_n$;

- условия обращения в «1» для конъюнкции: $\overline{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \bar{x}_1 + \bar{x}_2 + \dots + \bar{x}_n$.

В алгебре логики имеется также 4 следствия. Для рассмотрения их приведем некоторые понятия:

1. Логическая функция (умножение, сложение) называется *элементарной*, если все ее аргументы являются единичными аргументами или отрицаниями одиночных аргументов.

2. Элементарное произведение (конъюнкция), являющееся функцией всех аргументов заданного набора, называется *конституентой единицы* (от английского слова constituent – составная часть чего-нибудь).

3. Элементарная дизъюнкция, являющаяся функцией всех аргументов заданного набора, называется *конституентой нуля*.

4. Количество аргументов в функции (сомножителей или слагаемых) называется *рангом*.

5. Две элементарные функции (дизъюнкция или конъюнкция) одинакового ранга называются *соседними*, если они являются функциями одних и тех же аргументов и отличаются только знаком отрицания у одного из аргументов.

С учетом приведенных выше понятий в табл. 3.2 даны следствия из законов алгебры логики. Подробнее они рассмотрены в [3].

Таблица 3.2

Следствия из законов алгебры логики

Наименование следствия	Примеры следствия для	
	дизъюнкции	конъюнкции
1. <i>Правило старшинства</i>	Вначале выполняется функция НЕ, затем «Λ» и «V»	
2. <i>Правило склеивания</i> (для соседних элементарных функций)	$\bar{x}_1 \cdot x_2 \cdot x_4 \cdot x_5 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_4 \cdot x_5 =$ $= \bar{x}_1 \cdot x_4 \cdot x_5$	$\begin{aligned} & \bar{x}_1 + \bar{x}_2 + \bar{x}_3 + x_6 \text{ } \overline{\wedge} \\ & \times \bar{x}_1 + x_2 + \bar{x}_3 + x_6 \text{ } \overline{\wedge} \\ & = x_1 + \bar{x}_3 + x_6 \end{aligned}$
3. <i>Правило поглощения</i> (для элементарных функций разного ранга, меньшая по рангу из которых является частью большей)	$\bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \cdot x_6 + x_2 \cdot \bar{x}_3 =$ $= x_2 \cdot \bar{x}_3$	$\begin{aligned} & \bar{x}_1 + x_2 + x_4 + x_5 \text{ } \overline{\wedge} \\ & \times \bar{x}_1 + x_5 \text{ } \overline{\wedge} = \bar{x}_1 + x_5 \end{aligned}$
4. <i>Правило разворачивания</i> функции в совокупность конститuent 1 или 0	$\begin{aligned} y &= x_1 + \bar{x}_2 + x_5 = \\ &= x_1 + \bar{x}_2 + 0 + 0 + x_5 = \\ &= x_1 + x_2 + x_3 \cdot \bar{x}_3 + \\ &+ x_4 \cdot \bar{x}_4 + x_5 = \\ &= (x_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_6 + x_5) \times \\ &\times (x_1 + \bar{x}_2 + \bar{x}_3 + x_4 + x_5) \times \\ &\times (x_1 + \bar{x}_2 + x_3 + \bar{x}_4 + x_6) \times \\ &\times (x_1 + \bar{x}_2 + x_3 + \bar{x}_4 + x_5) \end{aligned}$	$\begin{aligned} y &= x_1 \cdot \bar{x}_2 \cdot x_5 = \\ &= x_1 \cdot \bar{x}_2 \cdot 1 \cdot 1 \cdot x_5 = x_1 \cdot \bar{x}_2 \times \\ &\times \bar{x}_3 + \bar{x}_3 \text{ } \overline{\wedge} \bar{x}_4 + \bar{x}_4 \text{ } \overline{\wedge} x_5 = \\ &= x_1 \cdot \bar{x}_2 \cdot x_3 \cdot x_4 \cdot x_5 + \\ &+ x_1 \cdot \bar{x}_2 \cdot x_3 \cdot \bar{x}_4 \cdot x_5 + \\ &+ x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot x_4 \cdot x_5 + \\ &+ x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot \bar{x}_4 \cdot x_5 \end{aligned}$

3.4. Синтез комбинационных устройств

Комбинационным устройством (КУ) будем называть устройство, не обладающее памятью и в котором выходные сигналы в любой момент времени определяются только комбинацией входных сигналов, реализующее любой наперед заданный закон преобразования двоичной информации.

Синтез комбинационного устройства подразделяется на 4 этапа:

- 1) составление таблицы истинности синтезируемого КУ согласно его определению, назначению и описанию принципов работы;
- 2) составление математической формулы для логической функции, описывающей работу синтезируемого КУ согласно таблице истинности;
- 3) анализ полученной функции с целью построения различных вариантов ее математического выражения (на основании законов алгебры логики) и нахождение наилучшего из них в соответствии с тем или иным критерием;
- 4) составление функциональной (логической) схемы КУ из элементов И, НЕ, ИЛИ.

Рассмотрим синтез КУ на примере синтеза полусумматора, имеющего 2 входа (слагаемые x_1 и x_2) и 2 выхода (цифры суммы S и переноса P).

1-й этап. Составим таблицу истинности КУ (табл. 3.3).

Таблица 3.3

Таблица истинности комбинационного устройства

1-я цифра – слагаемое x_1	0	0	1	1
2-я цифра – слагаемое x_2	0	1	0	1
Цифра переноса P	0	0	0	1
Цифра суммы S	0	1	0	1

2-й этап

Методика перехода от таблицы истинности к аналитическому выражению подробно изложена в [5]. Приведем только конечную форму и правила записи аналитического выражения функции на основании таблицы истинности.

Аналитическое выражение функции записывается в одной из двух форм:

а) *совершенной дизъюнктивной нормальной форме (СДНФ)* –

$$f_{\text{СДНФ}} \llbracket x_1, x_2, \dots, x_n \rrbracket \stackrel{\sim}{=} \bigvee_{\substack{\text{для } \\ f_j=1}} \bigwedge_{i=1}^n x_i^{[x_i]_j}, \quad (3.1)$$

где $x_i (x_1, x_2, \dots, x_n)$ – двоичные переменные (входные аргументы),

j – номер набора аргументов в таблице истинности,

$[x_i]_j$ – индекс, определяющий, что значения переменной x_i берутся из таблицы истинности для j - набора;

б) совершенной конъюнктивной нормальной форме (СКНФ) –

$$f_{\text{СКНФ}} \llbracket x_1, x_2, \dots, x_n \rrbracket \stackrel{\sim}{=} \bigwedge_{\substack{\text{для } \\ f_j=1}} \bigvee_{i=1}^n x_i^{[\bar{x}_i]_j}. \quad (3.2)$$

Эти формы называются *нормальными*, потому что все члены функций в данном случае имеют вид элементарных дизъюнкций или конъюнкций.

Они называются *совершенными*, потому что все члены этих выражений имеют высший ранг, т.е. являются конституентами.

Напомним правила получения аналитической записи логической функции для некоторого КУ.

Правило 1

Для того чтобы получить аналитическое выражение функции, заданной таблично, в СДНФ, необходимо составить сумму (V) конституент единицы для всех наборов входных двоичных переменных, для которых реализация функции $f_j = 1$, причем символ любой переменной в некоторой конституенте берется со знаком отрицания, если конкретное значение переменной $[x_i]_j$ в рассматриваемом наборе равно 0.

Применив это правило для функций P и S полусумматора, на основании табл. 3.3 получим:

$$P_{\text{СДНФ}} \llbracket x_1, x_2 \rrbracket \stackrel{\sim}{=} x_1 \cdot x_2; \quad (3.3, \text{ а})$$

$$S_{\text{СДНФ}} \llbracket x_1, x_2 \rrbracket \stackrel{\sim}{=} \bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2.$$

Правило 2

Для того чтобы получить аналитическое выражение функции, заданной таблично, в СКНФ, необходимо составить произведение (конъюнкцию) конституент нуля для всех наборов входных двоичных переменных, для которых реализация функции $f_j = 0$, причем символ любой переменной в некоторой конституенте берется со знаком отрицания, если ее конкретное значение $[x_i]_j$ в рассматриваемом наборе равно 1.

Применив это правило для функций P и S полусумматора, на основании табл. 3.3 получим:

$$\begin{aligned} P_{СКНФ} x_1, x_2 &= x_1 + x_2 \cdot x_1 + \bar{x}_2 \cdot \bar{x}_1 + x_2 ; \\ S_{СКНФ} x_1, x_2 &= x_1 + x_2 \cdot \bar{x}_1 + \bar{x}_2 . \end{aligned} \quad (3.3, б)$$

Формы представления функций СДНФ и СКНФ являются каноническими. К ним можно перейти не только от табличной, но и от произвольной аналитической записи функции.

В общем случае переход от произвольной формы к СДНФ или СКНФ производится в 3 шага.

1. С помощью многократного применения законов инверсии (НЕ) общие и групповые отрицания снимаются так, чтобы инверсия оставалась только у одиночных переменных.

2. С помощью распределительных законов производится переход к одной из нормальных форм:

- для перехода к ДНФ – применяется закон 1-го рода;
- для перехода к КНФ – закон 2-го рода.

3. Производится преобразование членов ДНФ или КНФ в соответствующие конституенты при помощи правила разворачивания.

3-й этап

Анализ и оптимизация (минимизация) логических функций являются важнейшими в синтезе КУ. Поэтому эти вопросы будут рассмотрены отдельно (см. подразд. 3.5).

4-й этап

Построение схемы основано на прямом замещении элементарных произведений, сумм и отрицаний соответственно конъюнкторами (&), дизъюнкторами (1) и инверторами (1).

На рис. 3.2 и 3.3 приведены функциональные схемы полусумматора, реализующие функции P и S , представленные в СДНФ и СКНФ соответственно.

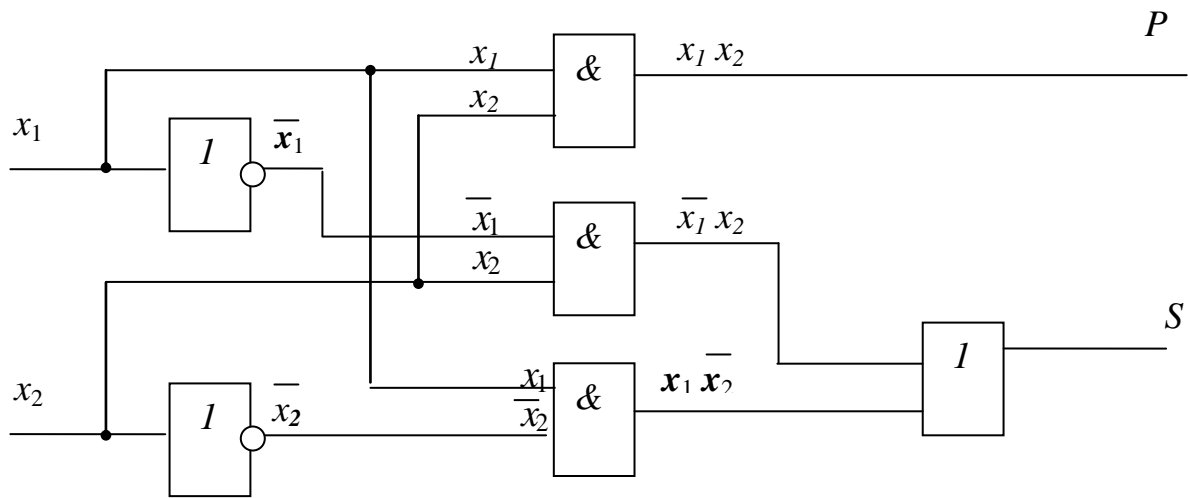


Рис. 3.2

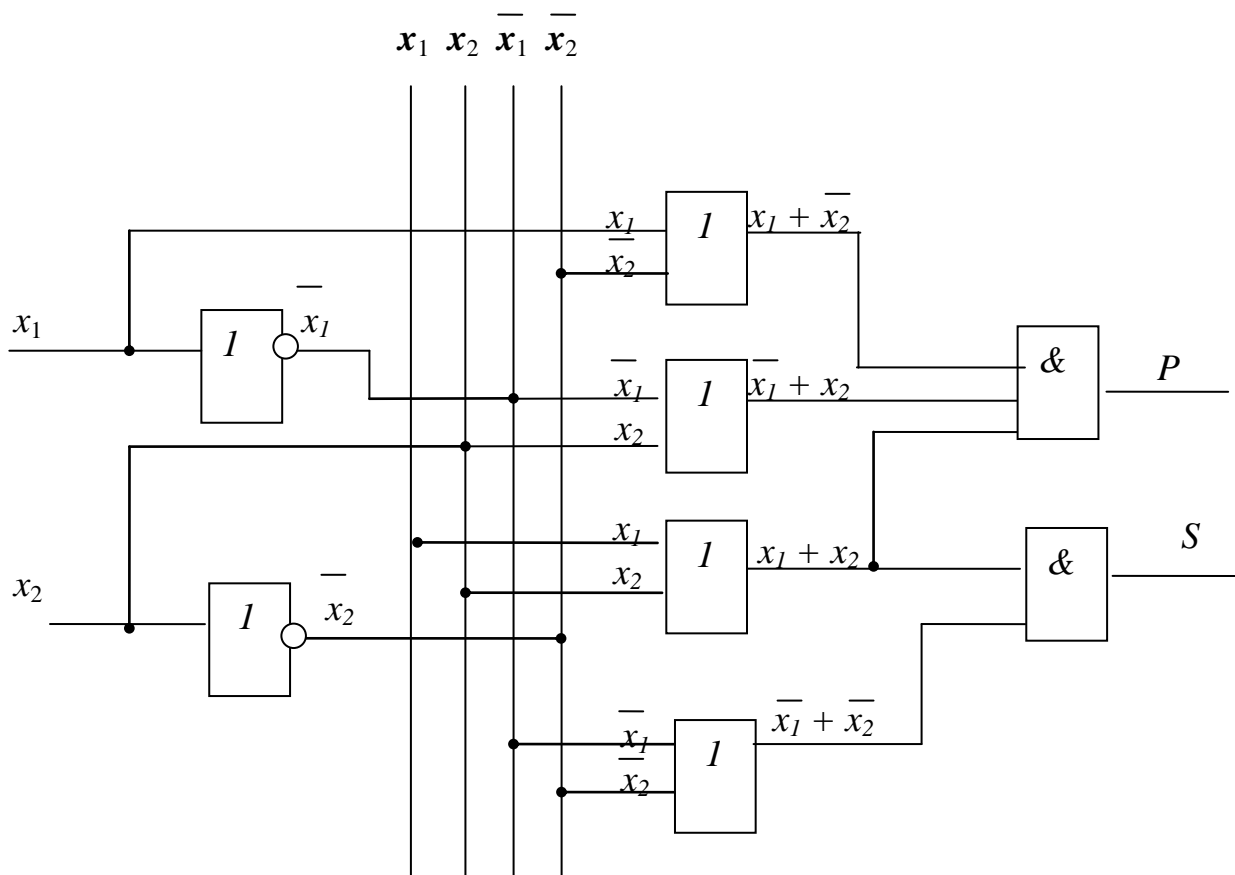


Рис. 3.3

3.5. Сложность логических формул и методы минимизации логических функций

3.5.1. Общие понятия

Ориентировочно сложность логической схемы можно оценить по количеству требуемого оборудования, подсчитав суммарное количество входов всех элементов, входящих в эту схему. Каждая логическая схема содержит примерно столько условных транзисторов, сколько у нее входов. Так, схема, представленная на рис. 3.2, требует 10 условных транзисторов, а на рис. 3.3 – 15 условных транзисторов.

Каждый логический элемент формирует выходной сигнал с некоторой задержкой, причем задержки логических элементов НЕ, И и ИЛИ отличаются незначительно:

$$\tau_{\text{НЕ}} \simeq \tau_{\text{И}} \simeq \tau_{\text{ИЛИ}} \simeq \tau_{\text{ЛЭ}}.$$

Следовательно, любую логическую схему можно оценить по быстродействию, подсчитав суммарное количество ступеней из логических элементов, которое проходит сигнал от входа схемы до ее выхода. Так, для схем, представленных на рис. 3.2 и 3.3, суммарная задержка составляет $3 \tau_{\text{ЛЭ}}$.

Сравнивая обе схемы (см. рис. 3.2 и 3.3), можно оценить основные параметры схемы – необходимое количество оборудования (и, следовательно, стоимость) и быстродействие – до построения схемы. Это значит, что инженер-разработчик может предопределить параметры проектируемого устройства путем целенаправленного подбора той или иной формы представления логической функции, описывающей его работу.

Однозначность соответствия формы ЛФ и параметров реальной электронной схемы приводит к необходимости оптимизации функции. Известно, что добиться оптимизации по всем параметрам практически невозможно, тем более, что некоторые из них противоречат друг другу (например производительность и количество оборудования).

Чаще всего решается задача оптимизации по одному параметру (например количеству оборудования). Такая частная оптимизация называется минимизацией.

Таким образом, возникает задача нахождения из всех возможных форм ЛФ ее так называемой минимальной формы, обеспечивающей минимум затрат оборудования синтезируемого узла, если имеется заданный набор ЛЭ (НЕ, И,

ИЛИ) с определенными техническими характеристиками (например количество входов у элементов И, ИЛИ).

Имеется несколько методов минимизации ЛФ. Наиболее известны три:

- 1) расчетный (метод непосредственных преобразований);
- 2) расчетно-табличный (метод Квайна – Мак-Класки);
- 3) табличный (метод Вейча–Карно).

В общем случае минимизация проводится в три этапа.

1. Переход от СДНФ (или СКНФ) к *сокращенной* ДНФ (КНФ) путем производства всех возможных склеиваний друг с другом, сначала конститuent, потом всех производных членов более низкого ранга (импликант).

2. Переход от сокращенной к тупиковой нормальной форме.

Тупиковой (Т) называют такую ДНФ (КНФ), членами которой являются простые импликанты, среди которых нет ни одной лишней (в прямом смысле), т.е. такой, удаление которой не влияет на значение истинности этой функции.

3. Переход от тупиковой формы функции к ее *минимальной* форме (факторизация). Этот этап не является регулярным (формальным), а требует интуиции, опыта.

Различные методы минимизации отличаются друг от друга путями и средствами реализации того или иного этапа.

3.5.2. Расчетный метод минимизации

При расчетном методе минимизации все этапы выполняются расчетно.

Пример:

Пусть задана некоторая функция в СДНФ, которую необходимо минимизировать:

$$f_{\text{СДНФ}} = \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot x_3 + x_1 \cdot \bar{x}_2 \cdot x_3. \quad (3.4)$$

1-й этап

Переход от СДНФ к сокращенной ДНФ, применяя склеивание. Получим

$$f_{\text{СДНФ}} = x_1 \cdot x_3 + \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot x_2. \quad (3.5)$$

2-й этап

Переход к тупиковой форме путем проверки каждой импликанты в выражении (3.5):

На значение истинности функции влияет только та импликанта, которая сама равна 1. Любая импликанта становится равной 1 лишь на одном вполне определенном наборе значений истинности своих аргументов. Но если именно на этом наборе сумма остальных членов тоже обращается в «1», то рассматриваемая импликанта не влияет на значение истинности функции в этом единственном случае, т.е. она является *лишней*.

Применив это правило к выражению (3.5), получим, что импликанта $\bar{x}_1 \cdot x_3$ является лишней и тупиковая форма выражения (3.4) будет иметь вид

$$f_{\text{ТДНФ}} = \bar{x}_1 \cdot x_2 + \bar{x}_2 \cdot x_3. \quad (3.6)$$

Примечание. Может оказаться, что лишних членов будет больше 1. В этом случае нельзя все лишние импликанты сразу отбрасывать: вначале можно исключить только один член выражения, а затем снова проверить все оставшиеся импликанты на «лишний член».

При использовании в качестве исходной формы представления минимизируемой функции СКНФ все этапы выполняются аналогично минимизации СДНФ, но 2-й этап (переход к ТКНФ) имеет некоторую специфику: на значение истинности функции в КНФ влияет только та импликанта, которая сама равна 0. Но любая импликанта становится равной 0 только на одном наборе своих аргументов. Следовательно, если именно на этом наборе произведение остальных импликант тоже равно 0, то рассматриваемая импликанта является лишней.

С учетом этой особенности, выполнив минимизацию функции, представленной в СКНФ и имеющей вид

$$f_{\text{СКНФ}}(x_1, x_2, x_3) = \overline{x_1 + x_2 + x_3} \supseteq \overline{x_1 + x_2 + x_3} \supseteq \overline{x_1 + \bar{x}_2 + x_3} \supseteq \overline{x_1 + \bar{x}_2 + \bar{x}_3}, \quad (3.7)$$

получим тупиковую форму этой функции:

$$f_{\text{ТКНФ}}(x_1, x_2, x_3) = \overline{x_2 + x_3} \supseteq \overline{x_1 + \bar{x}_2}. \quad (3.8)$$

3-й этап

Переход от тупиковой к минимальной форме, как уже отмечалось, не является регулярным и зависит от опыта и навыков разработчика.

3.5.3. Расчетно-табличный метод минимизации

Минимизация этим методом (способом) отличается от расчетного методикой выявления лишнего члена в сокращенной ДНФ или КНФ. Этапы 1 и 3 идентичны расчетному методу.

Нахождение ТДНФ (ТКНФ) производится при помощи специальной таблицы, предложенной У. Квайном. В графах (колонках) указываются конститутенты, в строках – импликанты сокращенной формы (табл. 3.4). Исходная форма минимизируемой функции – СДНФ (или СКНФ).

Таблица 3.4

Пример таблицы Квайна для расчетно-табличной минимизации функции

Импликанты	Конститутенты			
	$\bar{x}_1 \cdot \bar{x}_2 \cdot x_3$	$\bar{x}_1 \cdot x_2 \cdot \bar{x}_3$	$\bar{x}_1 \cdot x_2 \cdot x_3$	$x_1 \cdot \bar{x}_2 \cdot x_3$
$\bar{x}_1 \cdot x_3$	+		+	
$\bar{x}_2 \cdot x_3$	+			+
$\bar{x}_1 \cdot x_2$		+	+	

Процесс минимизации осуществляется путем последовательного сопоставления каждой импликанты со всеми конститутентами. Если импликанта является частью конститутенты, то в таблице в соответствующей клетке ставится какой-нибудь знак (например «+»).

Каждая конститутента может покрываться несколькими импликантами. Задача состоит в том, чтобы вычеркиванием некоторых (лишних) импликант попытаться оставить в каждой колонке только один знак «+» или, по крайней мере, – минимальное количество. Из табл. 3.4 видно, что лишней является импликанта $\bar{x}_1 \cdot x_3$, и тупиковая форма будет иметь вид

$$f_{\text{ТДНФ}} = \bar{x}_1 \cdot x_2 + \bar{x}_2 \cdot x_3,$$

что совпадает с выражением (3.6) для расчетного метода.

Возможность использования ЭВМ в этом методе связана с именем Мак-Класки. Практическая реализация усовершенствования Мак-Класки основана на числовом представлении логических функций. Десятичный индекс j у символа реализации функции f_j , развернутой в n -разрядное двоичное число, позволяет

полностью восстановить вид набора аргументов $[x_i]_j$, который соответствует данной реализации. А вид набора значений аргументов позволяет восстановить вид соответствующей ему конституенты единицы или нуля.

Например, относительно громоздкую буквенную запись функций $P_{\text{СДНФ}}$, $S_{\text{СДНФ}}$ в виде выражения (3.3, а) можно заменить более компактной числовой записью:

$$\left. \begin{aligned} P_{\text{СДНФ}}(x_1, x_2) &= 11_{(2)} = 3_{(10)}; \\ S_{\text{СДНФ}}(x_1, x_2) &= 01_{(2)} \vee 10_{(2)} = 1_{(10)} \vee 2_{(10)} = \vee (1, 2). \end{aligned} \right\} \quad (3.9)$$

Запись (3.9) представляет собой модификацию записи (3.3, а) с применением переходов $x_i \rightarrow 1$, $\bar{x}_i \rightarrow 0$.

При таком представлении логических функций их можно вводить в ЭВМ, обрабатывать в соответствии с правилами булевой алгебры.

Отметим, что любая переменная x_i в этом случае будет иметь не два, а три состояния: 0, 1 и отсутствие переменной.

3.5.4. Табличный метод минимизации

При этом методе два первых этапа производятся при помощи специальной таблицы, называемой *диаграммой Вейча–Карно* (или картой Карно).

В принципе диаграмма Вейча–Карно является разновидностью табличной записи некоторой функции, заданной в СДНФ или СКНФ. Она имеет в общем виде вид прямоугольника, разбитого на 2^n малых квадрата. При нечетном n одну сторону образуют $2^{(n-1)/2}$ малых квадрата, а другую – $2^{(n+1)/2}$. При четном n каждая сторона прямоугольника образует из $2^{n/2}$ квадратов.

Каждый малый квадрат соответствует одному определенному набору аргументов и, следовательно, одной определенной конституенте.

Для того чтобы при помощи таблиц Вейча–Карно произвести минимизацию, необходимо выполнить одно важное условие: **в соседних клетках (в физическом смысле) должны находиться соседние конституенты.**

Это условие называется *условием совмещенного соседства*.

Для нахождения квадрата, соответствующего определенному набору аргументов, в качестве координат квадратов используются значения аргументов.

Строится плоская прямоугольная система координат, ось ординат которой совпадает с левым вертикальным краем диаграммы, а ось абсцисс – с нижним краем. Пронумеруем ряды и столбцы клеток на диаграмме двоичными числами, причем каждый двоичный разряд сопоставим с определенной логической переменной. Тогда ось ординат явится осью изменения сложного аргумента, состоящего из $n/2$ первых логических переменных, а ось абсцисс – $n/2$ последних логических переменных. Если n нечетное, то по оси ординат $(n - 1)/2$ – логических переменных, а по оси абсцисс – $(n + 1)/2$ переменных. Тогда каждый сложный аргумент изменяется дискретно вдоль соответствующей координатной оси, пробегая все числовые значения от $0 = 000\dots 0$ до $2^{n/2} - 1 = 111\dots$

Следовательно, в построенной системе координат можно положение любой клетки на диаграмме точно и однозначно определить парой полуслов с разрядностью $n/2$ (или $(n-1)/2$ и $(n+1)/2$). Каждое слово, состоящее из двух координатных полуслов, будет соответствовать определенной конституэнте, т.к. всегда можно сделать обратный переход от цифр 0 и 1 к символам \bar{x}_i и x_i соответственно ($i = 1, 2, \dots, n$).

Таким образом, имея на диаграмме по оси ординат и абсцисс номера – координаты клеток в двоичной системе счисления, не заполняя самих клеток, всегда можно быстро и легко указать, какой конституэнте соответствует та или иная клетка и наоборот. Однако, если эти координаты будут проставляться в обычном двоичном коде, то условие совмещенного соседства будет выполняться не всегда. Для выполнения этого условия координаты следует проставлять в двоичном циклическом *коде Грея*. Этот код имеет такую особенность, что представленные в нем изображения двух чисел, имеющих различие по величине на единицу младшего разряда (или просто на 1 для целых чисел), отличаются друг от друга только в одном разряде. Для примера в таблице 3.5 приведены первые десять чисел в двоичном коде и коде Грея.

Таблица 3.5

Примеры чисел в коде Грея

Десятичное число	Двоичное число	Число в коде Грея
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Анализ табл. 3.5 показывает, что код Грея можно интерпретировать как двоичную систему счисления с весами, равными

$$W_{i(u)} = (-1)^j (2^i - 1),$$

где $i = 1, 2, \dots, n$ – номера разрядов, считая справа налево;

j – количество единиц слева от данного разряда с номером i .

Такая интерпретация позволяет сформулировать правило преобразования любого двоичного числа в код Грея:

1) самая старшая цифра (единица числа) в коде Грея совпадает с самой старшей значащей цифрой этого же числа в двоичном коде;

2) цифра в любом другом, более младшем, разряде числа в коде Грея:

а) совпадает с соответствующей цифрой числа в двоичном коде, если слева от данной цифры в коде Грея имеется четное число единиц;

б) совпадает с отрицанием соответствующей цифры в двоичном коде, если слева от данной цифры в коде Грея имеется нечетное количество единиц.

Пример 3.1

Перевести из двоичной системы в код Грея целое число $x = x_3 x_2 x_1 = 111$.

Решение:

$$Y_3 = x_3 = 1; Y_2 = \overline{x_2} = 0; Y_1 = \overline{x_1} = 0.$$

$$\text{Ответ: } Y = Y_3 Y_2 Y_1 = 100.$$

Применение кода Грея для нумерации клеток по осям ординат и абсцисс в диаграмме Вейча–Карно обеспечивает автоматическое размещение в соседних

клетках соседних членов СДНФ (или СКНФ) любой логической функции. В самих клетках необходимо только проставить значение истинности реализации функции на каждом из набора аргументов. Причем в случае задания функции в СДНФ – достаточно проставить только «1», а в случае СКНФ – только «0».

Так, например, на рис. 3.4 и 3.5 приведены таблицы Вейча–Карно для функций P и S полусумматора и функций от трех переменных, рассмотренных в подразд. 3.5.2.

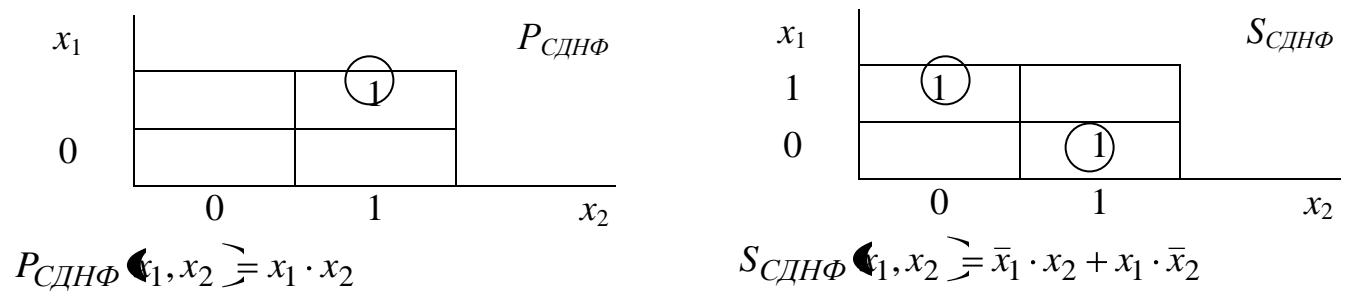


Рис. 3.4

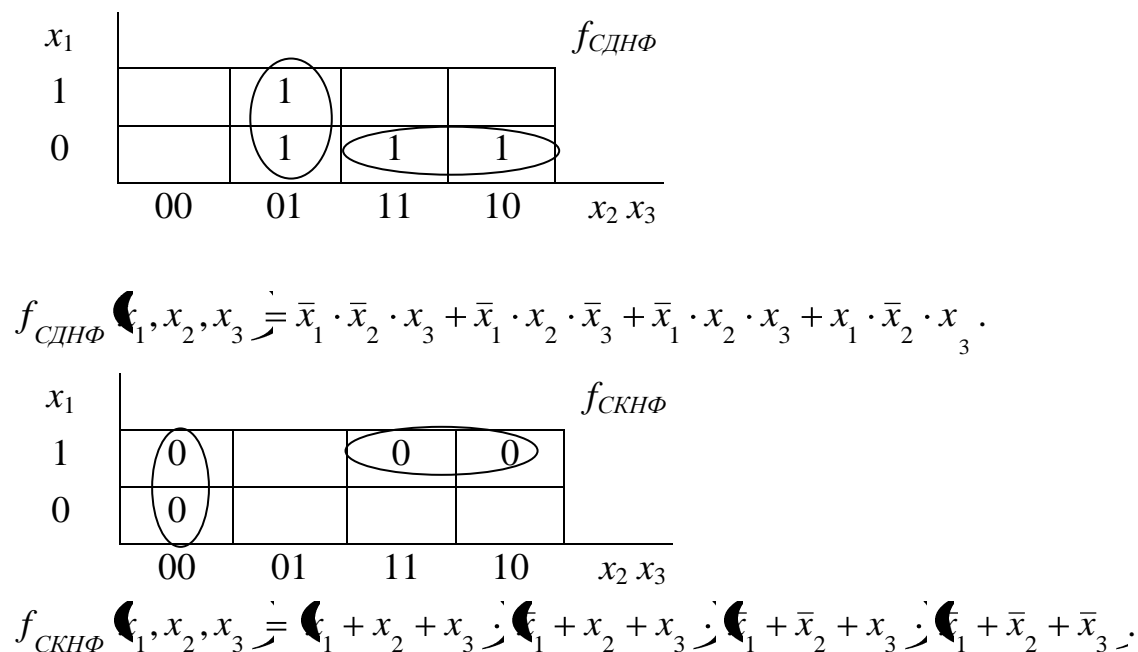


Рис. 3.5

Процесс минимизации с помощью диаграмм Вейча–Карно подчиняется следующим правилам.

Правило 1

2^i смежных клеток, расположенных в виде прямоугольника, соответствуют одной элементарной конъюнкции (дизъюнкции), ранг которой r меньше ранга конститутенты n на i единиц.

Правило 2

В любой диаграмме соседними клетками являются не только смежные клетки, но и клетки, расположенные на противоположных концах любой строки и любого столбца.

Импликанта, соответствующая некоторой группе заполненных клеток, будет содержать в себе символы тех переменных, значения истинности которых совпадают у всех объединенных клеток. Пользуясь переходом $0 \rightarrow \bar{x}_i$ и $1 = x_i$ (для функций, представленных в СДНФ) или $0 \rightarrow x_i$ и $1 = \bar{x}_i$ (для СКНФ), можем записать выражения для каждого члена функции или импликанты.

Так, для функций, представленных на рисунке 3.5, запишем

$$f_{\text{ДНФ}} = \bar{x}_2 \cdot x_3 + \bar{x}_1 \cdot x_2,$$
$$f_{\text{КНФ}} = (x_2 + x_3) \cdot (x_1 + \bar{x}_2),$$

что совпадает с выражениями (3.6) и (3.8) для других методов минимизации.

Отметим, что этот метод минимизации очень удобен при $n \leq 4$.

Рассмотрим еще несколько примеров по минимизации логических функций и синтезу комбинационных схем.

Пример 3.2

Минимизировать функцию 4-х переменных

$$f_{\text{СДНФ}} = \vee \{4, 5, 7, 9, 13, 14, 15\} \text{ методом Вейча–Карно.}$$

Решение

1. Определим количество переменных и составим таблицу истинности для исходной функции. Из исходных данных видно, что максимальный номер набора переменных равен 15, следовательно, количество переменных равно 4.

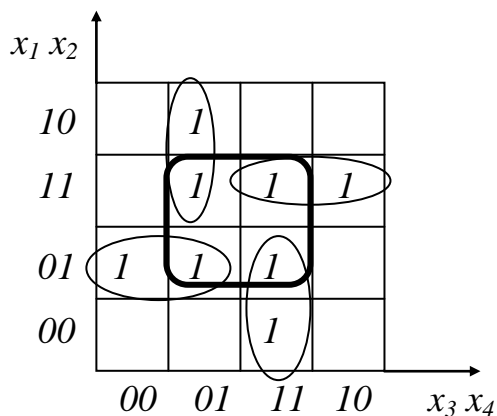
Тогда таблица истинности для заданной функции будет иметь вид

x_1		0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
x_2		0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
x_3		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
x_4		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
f_j		0	0	0	1	1	1	0	1	0	1	0	0	0	1	1
j		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

2. Запишем функцию в СДНФ на основании таблицы истинности:

$$\begin{aligned}
 f_{\text{СДНФ}}(x_1, x_2, x_3, x_4) &= \bigvee_{f_j=1} \bigwedge_{i=1}^n \bar{x}_i^{j_i} = \\
 &= \bigvee_{f_j=1} \bar{x}_1 \bar{x}_2 x_3 x_4 + \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4 + \bar{x}_1 x_2 \bar{x}_3 x_4 + \\
 &+ \bar{x}_1 x_2 x_3 x_4 + x_1 \bar{x}_2 \bar{x}_3 x_4 + x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 + \\
 &+ x_1 \bar{x}_2 x_3 x_4 + x_1 x_2 \bar{x}_3 \bar{x}_4 + x_1 x_2 \bar{x}_3 x_4 + \\
 &+ x_1 x_2 x_3 \bar{x}_4 + x_1 x_2 x_3 x_4.
 \end{aligned}$$

3. Составим таблицу Вейча–Карно для функции $f_{\text{СДНФ}}$:



Отметим, что таблицу Вейча–Карно можно составить сразу на основании таблицы истинности без представления функции в СДНФ или СКНФ.

4. Запишем тупиковую форму исходной функции на основании таблицы Вейча–Карно:

$$f_{\text{ТДНФ}}(x_1, x_2, x_3, x_4) = x_1 \cdot \bar{x}_3 \cdot x_4 + x_1 \cdot x_2 \cdot x_3 + \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_3 \cdot x_4.$$

Пример 3.3

Синтезировать одноразрядный двоичный сумматор с тремя входами (ОДС-3) в СДНФ.

Решение

1-й этап

Составим таблицу истинности ОДС-3:

i -я цифра первого операнда x_{1i}	0	0	0	0	1	1	1	1
i -я цифра второго операнда x_{2i}	0	0	1	1	0	0	1	1
Цифра переноса из ($i-1$)-го разряда x_{3i}	0	1	0	1	0	1	0	1
Цифра переноса в ($i+1$)-й разряд P_i	0	0	0	1	0	1	1	1
i -я цифра суммы S_i	0	1	1	0	1	0	0	1

Уточним, что в данной таблице

$P_i(x_{1i}, x_{2i}, x_{3i})$ – перенос в старший разряд;

$S_i(x_{1i}, x_{2i}, x_{3i})$ – цифра i -го разряда сумматора;

$$x_{3i} = P_{i-1}$$

2-й этап

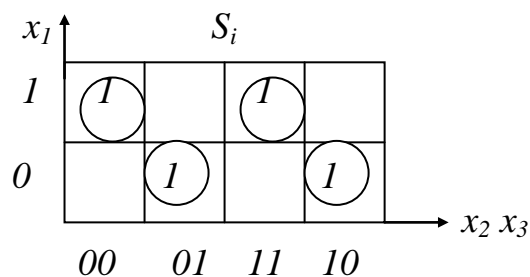
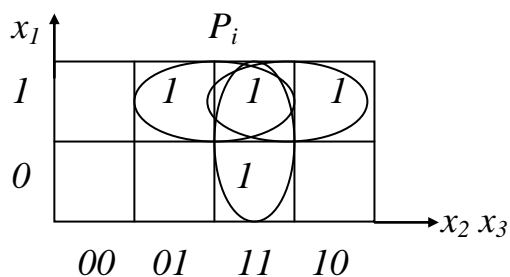
Запишем функции P_i и S_i по условиям истинности:

$$P_i \text{ СДНФ } = \overline{x_{1i}} \cdot x_{2i} \cdot x_{3i} \vee \overline{x_{1i}} \cdot \bar{x}_{2i} \cdot x_{3i} \vee \overline{x_{1i}} \cdot x_{2i} \cdot \bar{x}_{3i} \vee \overline{x_{1i}} \cdot x_{2i} \cdot x_{3i}$$

$$S_i \text{ СДНФ } = \overline{x_{1i}} \cdot \bar{x}_{2i} \cdot x_{3i} \vee \overline{x_{1i}} \cdot x_{2i} \cdot \bar{x}_{3i} \vee \overline{x_{1i}} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i} \vee \overline{x_{1i}} \cdot x_{2i} \cdot x_{3i}$$

3-й этап

Построим диаграмму Вейча–Карно для P_i и S_i .



Получаем

$$P_i \text{ СДНФ } (x_1, x_2, x_3) = x_1 \cdot x_2 + x_1 \cdot x_3 + x_2 \cdot x_3;$$

$$S_i \text{ СДНФ } (x_1, x_2, x_3) = x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 + x_1 \cdot x_2 \cdot x_3.$$

4-й этап

Синтезируем схему ОДС-3 (рис. 3.6):

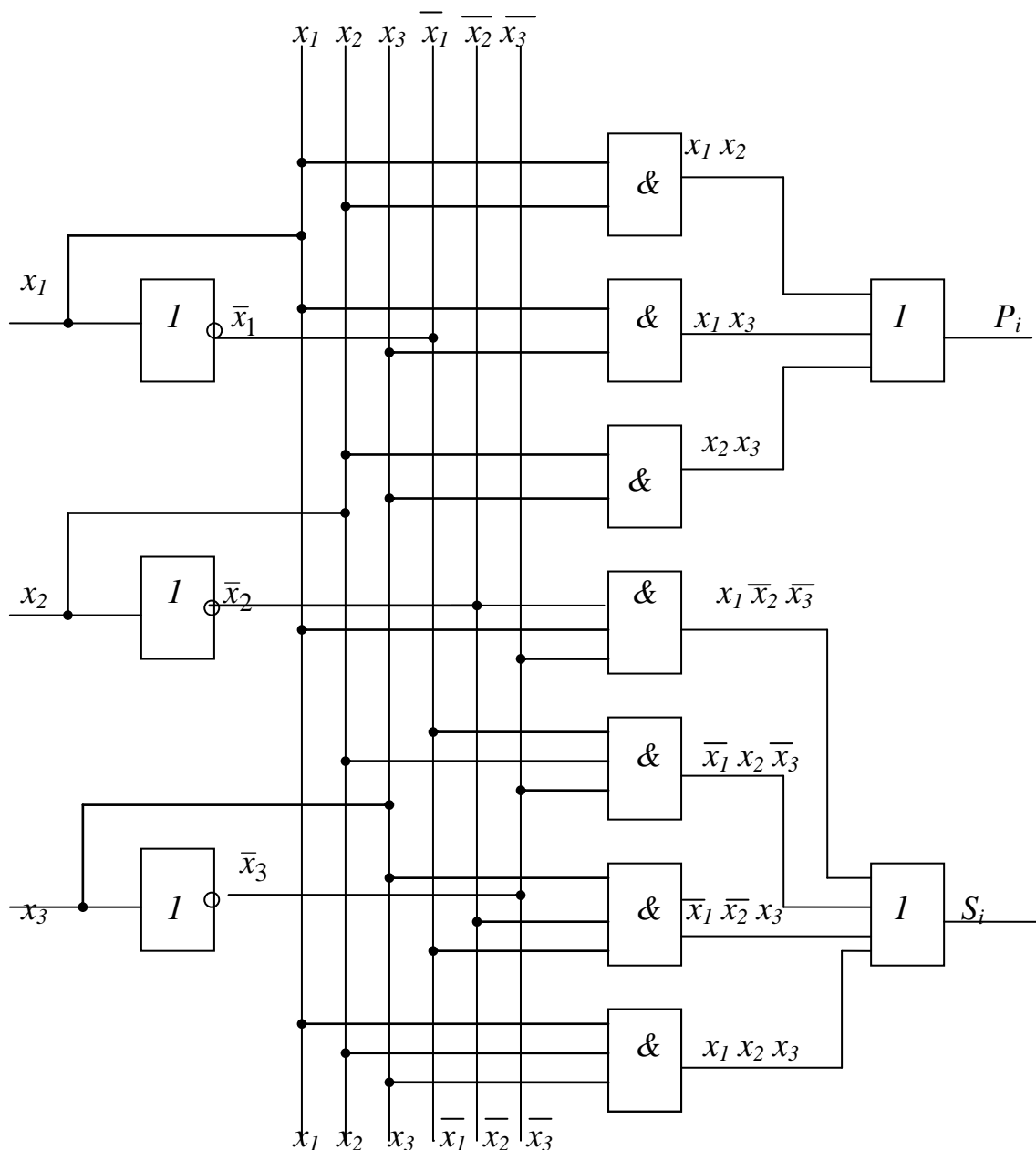


Рис. 3.6

3.6. Функциональная полнота различных наборов элементарных логических функций

3.6.1. Полная совокупность элементарных логических функций



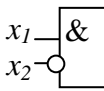

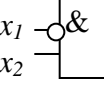

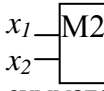

Под *элементарными* будем понимать логические функции (ЛФ) одного и двух аргументов.

При 2-х аргументах возможны 4 различных конкретных реализации любой функции, которые можно интерпретировать как единое 4-разрядное двоичное









число. Следовательно, всего существует $N = 2^4 = 16$ различных двухаргументных ЛФ (табл. 3.6).

Таблица 3.6

Логические функции двух переменных

Логические аргументы и функции	Значение истинности				Наименование функций	Запись функции с помощью		Обозначение
						самостоятельной нотации	нотации ОФПН (НЕ, И, ИЛИ)	
Аргумент x_1	0	0	1	1				
Аргумент x_2	0	1	0	1				
1	2	3	4	5	6	7	8	9
Функция f_0	0	0	0	0	Константа 0 (функция 0)	$f_0 = 0 = \bar{f}_{15}$	$f_0 = 0$	 f_0 генератор 0
Функция f_1	0	0	0	1	Конъюнкция (функция И)	$f_1 = x_1 \& x_2 = \bar{f}_{14}$	$f_1 = x_1 \cdot x_2$	 f_1 конъюнктор
Функция f_2	0	0	1	0	Запрет первого аргумента (функция НЕТ)	$f_2 = x_1 \leftarrow x_2 = \bar{f}_{13}$	$f_2 = x_1 \cdot \bar{x}_2$	 f_2 элемент запрета
Функция f_3	0	0	1	1	Повторение первого аргумента (функция ДА)	$f_3 = x_1 = \bar{f}_{12}$	$f_3 = x_1$	 f_3 повторитель
Функция f_4	0	1	0	0	Запрет второго аргумента (функция НЕТ)	$f_4 = x_2 \leftarrow x_1 = \bar{f}_{11}$	$f_4 = \bar{x}_1 \cdot x_2$	 f_4 элемент запрета
Функция f_5	0	1	0	1	Повторение второго аргумента (функция ДА)	$f_5 = x_2 = \bar{f}_{10}$	$f_5 = x_2$	 f_5 повторитель
Функция f_6	0	1	1	0	Неравнозначность (функция ИЛИ-ИЛИ)	$f_6 = x_1 \oplus x_2 = \bar{f}_9$	$f_6 = \bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2$	 f_6 сумматор по модулю 2
Функция f_7	0	1	1	1	Дизъюнкция (функция ИЛИ)	$f_7 = x_1 \vee x_2 = \bar{f}_8$	$f_7 = x_1 + x_2$	 f_7 дизъюнктор

Окончание табл. 3.6

1	2	3	4	5	6	7	8	9
Функция f_8	1	0	0	0	Операция Пирса (функция ИЛИ-НЕ)	$f_8 = x_1 \downarrow x_2 = \bar{f}_7$	$f_8 = \overline{x + x_2}$	 элемент Пирса
Функция f_9	1	0	0	1	Эквивалент- ность (функция И-И)	$f_9 = x_1 \propto x_2 = \bar{f}_6$	$f_9 = x_1 \cdot x_2 + \bar{x}_1 \cdot \bar{x}_2$	 эквивалентор
Функция f_{10}	1	0	1	0	Отрицание второго аргумента (функция НЕ)	$f_{10} = \bar{x}_2 = \bar{f}_5$	$f_{10} = \bar{x}_2$	 инвертор
Функция f_{11}	1	0	1	1	Импликация от второго аргу- мента к первому (функция НЕТ-НЕ)	$f_{11} = x_2 \rightarrow x_1 = \bar{f}_4$	$f_{11} = x_1 + \bar{x}_2$	 импликатор
Функция f_{12}	1	1	0	0	Отрицание первого аргумента (функция НЕ)	$f_{12} = \bar{x}_1 = \bar{f}_3$	$f_{12} = \bar{x}_1$	 инвертор
Функция f_{13}	1	1	0	1	Импликация от первого аргу- мента ко второму (функция НЕТ-НЕ)	$f_{13} = x_1 \rightarrow x_2 = \bar{f}_2$	$f_{13} = \bar{x}_1 + x_2$	 импликатор
Функция f_{14}	1	1	1	0	Операция Шеффера (функция И-НЕ)	$f_{14} = x_1 / x_2 = \bar{f}_1$	$f_{14} = \overline{x_1 \cdot x_2}$	 элемент Шеффера
Функция f_{15}	1	1	1	1	Константа 1 (функция 1)	$f_{15} = 1 = \bar{f}_0$	$f_{15} = 1$	 генератор 1

3.6.2. Абсолютная полнота наборов элементарных функций

Для того чтобы подбирать логические операции в функционально полные наборы (ФПН) с использованием принципа необходимости и достаточности, необходимо предварительно изучить их свойства.

В математической логике все элементарные функции подразделяют по своим свойствам на линейные, сохраняющие «0», сохраняющие «1», самодвойственные и монотонные.

1. ЛФ называется *линейной*, если ее можно представить полиномом первой степени вида

$$f(x_1, x_2, \dots, x_n) = a_n x_n \oplus \dots \oplus a_2 x_2 \oplus a_1 x_1 \oplus a_0,$$

где $a_i \in \{0, 1\}$, $i = 0, 1, 2, \dots, n$.

Существует только 8 линейных функций от 2-х переменных – $f_0, f_3, f_5, f_6, f_9, f_{10}, f_{12}, f_{15}$, что нетрудно видеть из табл. 3.7.

Таблица 3.7

Линейные функции двух переменных

Функции	a_2	a_1	a_0	Линейная функция 2-х аргументов $a_2 x_2 \oplus a_1 x_1 \oplus a_0$	
$const = 0$	0	0	0	$0 = f_0$	$f_0 = 0$
$const = 1$	0	0	1	$1 = f_{15}$	$f_{15} = 1$
Повторение x_1	0	1	0	$x_1 = f_3$	$f_3 = x_1$
Отрицание x_1	0	1	1	$x_1 \oplus 1 = \bar{x}_1 = f_{12}$	$\bar{x}_1 \cdot \bar{x}_2 \cdot x_3$
Повторение x_2	1	0	0	$x_2 = f_5$	$f_5 = x_2$
Отрицание x_2	1	0	1	$x_2 \oplus 1 = \bar{x}_2 = f_{10}$	$f_{10} = \bar{x}_2$
Неравнозначность	1	1	0	$x_2 \oplus x_1 = f_6$	$f_6 = \bar{x}_1 \cdot x_2 + x_1 \cdot \bar{x}_2$
Эквивалентность	1	1	1	$x_1 \oplus x_2 \oplus 1 = \overline{x_2 \oplus x_1} = f_9$	$f_9 = x_1 \cdot x_2 + \bar{x}_1 \cdot \bar{x}_2$

2. Сохраняющие 0 функции 2-х переменных

ЛФ называется *сохраняющей 0*, если на нулевом наборе аргументов ($x_1 = 0, x_2 = 0, \dots, x_n = 0$) функция равна 0, т.е. $f(0, 0, \dots, 0) = 0$.

Сохраняющими 0 являются первые 8 функций табл. 3.7, т.е. функции $f_0 - f_7$.

3. Сохраняющие 1

ЛФ называется *сохраняющей 1*, если на единичном наборе аргументов ($x_1 = 1, x_2 = 1, \dots, x_n = 1$) функция равна 1, т.е. $f(1, 1, \dots, 1) = 1$.

К этой группе относятся нечетные функции: $f_1, f_3, f_5, f_7, f_9, f_{11}, f_{13}, f_{15}$.

4. Самодвойственные функции

ЛФ называется *самодвойственной*, если на каждой паре противоположных наборов аргументов вида x_1, x_2, \dots, x_n и $\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n$ она принимает противоположные значения, т.е.

$$f(x_1, x_2, \dots, x_n) = \bar{f}(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n).$$

В табл. 3.7 есть только две пары противоположных наборов аргументов:

0.0 и **1.1**; **0.1** и **1.0**.

Функций, удовлетворяющих условиям

$$f(0,0) = \bar{f}(1,1),$$

$$f(0,1) = \bar{f}(1,0),$$

всего 4 – это f_3, f_5, f_{10}, f_{12} .

5. Монотонные функции

ЛФ называется *монотонной*, если при возрастании набора аргументов значения этой функции не убывают.

Таких функций всего 6 – $f_0, f_1, f_3, f_5, f_7, f_{15}$.

Проверку на монотонность для каждой ЛФ в таблице можно проводить по двум путям:

$$f(0.0) \Rightarrow f(0.1) \Rightarrow f(1.1)$$

или

$$f(0.0) \Rightarrow f(1.0) \Rightarrow f(1.1).$$

Можно показать, что при суперпозиции любых функций (или линейных, или сохраняющих 0, или сохраняющих 1, или самодвойственных, или монотонных) получаются такие же функции. Следовательно, для того чтобы набор ЛФ был ФПН, необходимо и достаточно, чтобы в него входили:

- хотя бы одна нелинейная функция;
- хотя бы одна не сохраняющая 0;
- хотя бы одна не сохраняющая 1;
- хотя бы одна не самодвойственная;
- хотя бы одна немонотонная.

Это теорема была доказана В.М. Глушковым.

Пользуясь этой теоремой, можно четко и однозначно указать, какие совокупности ЛФ двух переменных составят ФПН (табл. 3.8).

Таблица для выбора ФПН

Наименование функций	Свойства функций					Примечания
	Нелинейность	Несохраняемость 0	Несохраняемость 1	Несамодвойственная	Немонотонная	
Константа 0 (f_0)			+	+		
Конъюнкция (f_1)	+			+		Сходна по свойствам с f_7
Запрет (f_2, f_4)	+		+	+	+	
Повторение (f_3, f_5)						Тривиальная функция
Неравнозначность (f_6)			+	+	+	
Дизъюнкция (f_7)	+			+		Сходна по свойствам с f_1
Операция Пирса (f_8)	+	+	+	+	+	Обеспечивает ФПН
Равнозначность (f_9)		+		+	+	
Отрицание (f_{10}, f_{12})		+	+		+	
Импликация (f_{11}, f_{13})	+	+		+	+	
Операция Шеффера (f_{14})	+	+	+	+	+	Обеспечивает ФПН
Константа 1 (f_{15})		+		+		

3.6.3. Синтез комбинационных устройств в различных базисах

Как видно из табл. 3.8, имеется возможность выбора ФПН (иногда говорят – логических базисов – ЛБ). Для выбора ЛБ для реализации ЭВМ необходимо проанализировать технические свойства логических функций (ЛФ), т.е. реализуемость ЛФ техническими средствами. С этой точки зрения все ЛФ разделим на 3 группы:

- 1) непосредственно реализуемые на современных переключательных устройствах;
- 2) непосредственно реализуемые только на некоторых переключательных устройствах;
- 3) косвенно реализуемые через ЛФ первой и второй группы (т.е. непосредственно нереализуемые).

Такое разделение относительно, так как современный уровень развития элементной базы позволяет реализовать непосредственно то, что вчера было невозможно или неэффективно.

В начале 80-х гг. к ЛФ 1-й группы относились:

- константа 0 (f_0);
- операция Пирса (f_8);
- конъюнкция (f_1);
- отрицание (f_{10}, f_{12});
- повторение (f_3, f_5);
- операция Шеффера (f_{14});
- дизъюнкция (f_7);
- константа 1 (f_{15}).

Из этих функций константы (f_0 и f_{15}) и повторения (f_3 и f_5) являются *тривиальными*, т.е. реализуются простейшими техническими средствами, конъюнкторы и дизъюнкторы – более сложные схемы (примеры их реализации на транзисторах рассмотрены в [5]).

Можно показать, что операции Пирса и Шеффера (ИЛИ-НЕ и И-НЕ) также могут быть реализованы непосредственно (рис. 3.7).

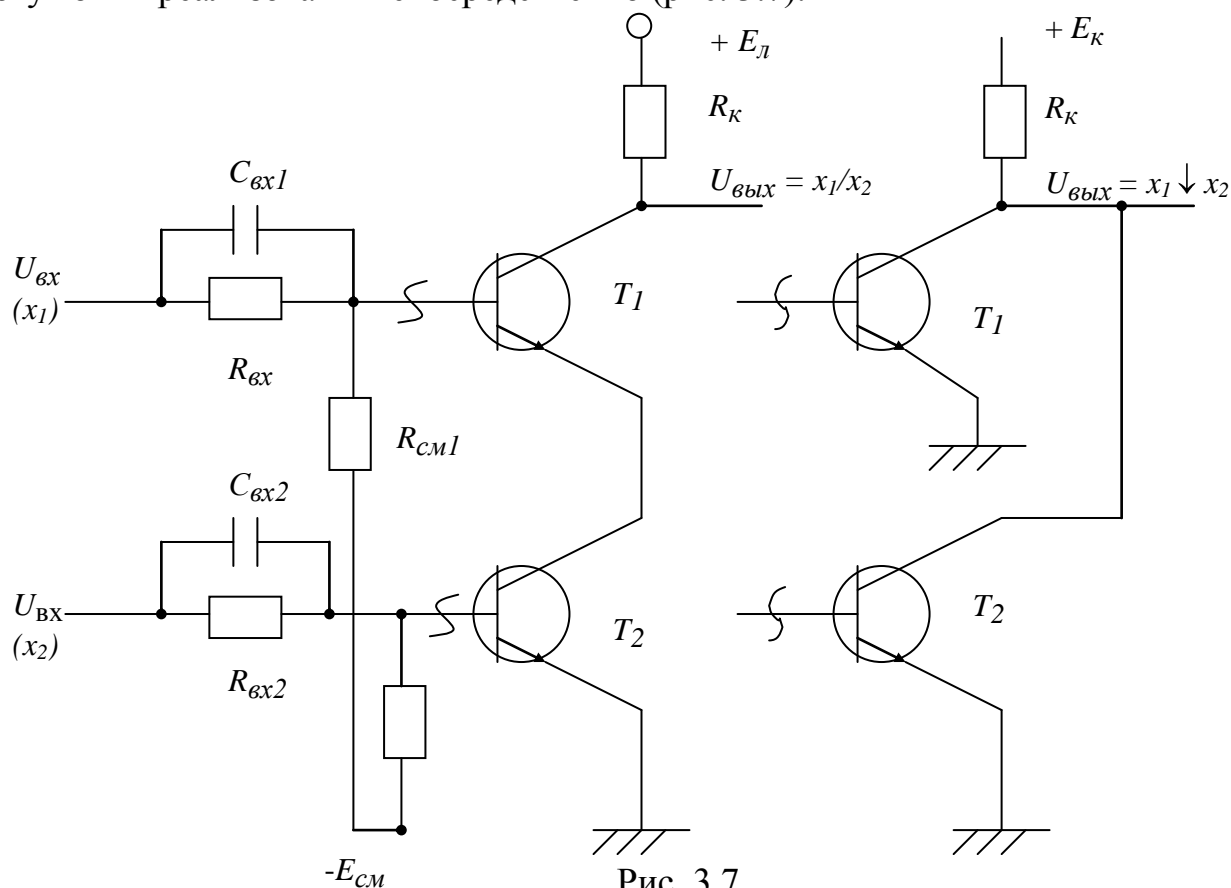


Рис. 3.7

В ЭВМ 1–2 поколений эти функции практически не применялись, хотя требовали транзисторов не более по количеству, чем схемы И и ИЛИ, скорее просто «не дошли» до их применения.

Примеры реализации логических элементов косвенного исполнения приведены на рис. 3.8.

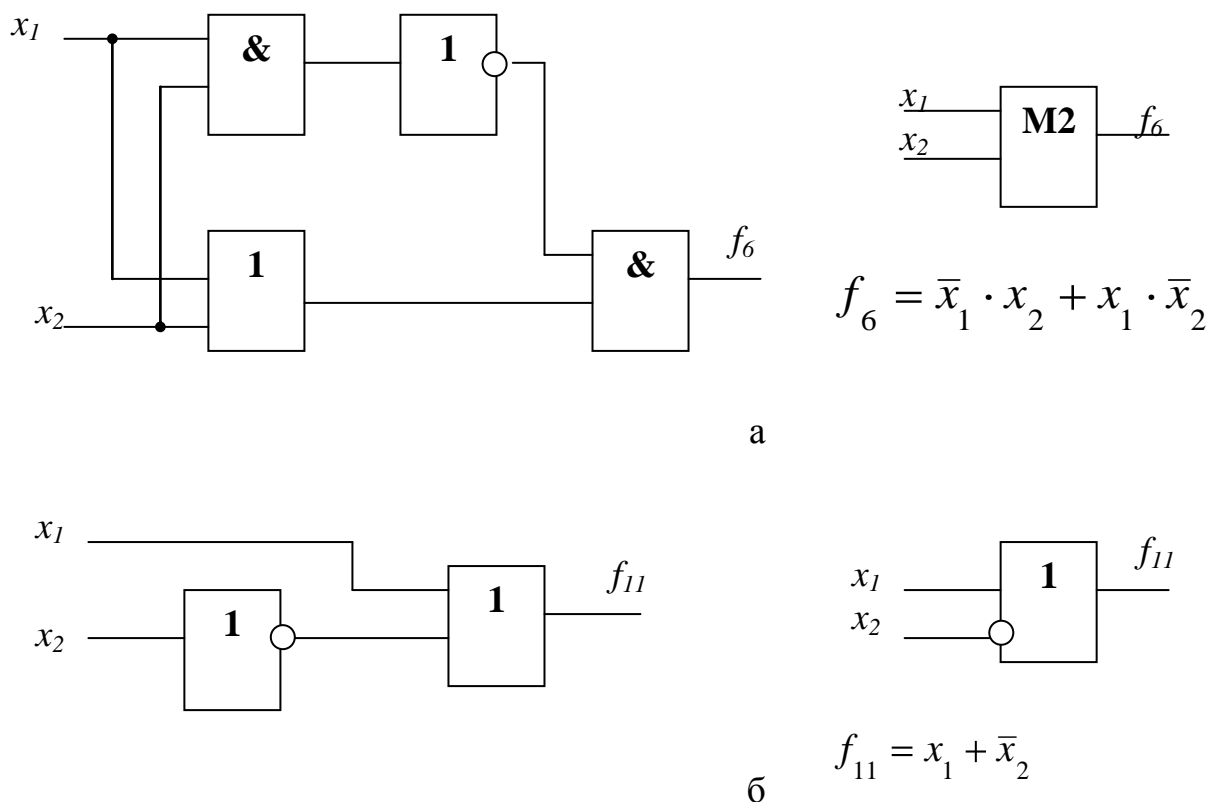


Рис. 3.8

В ЭВМ первого и второго поколения использовался логический базис – НЕ-И-ИЛИ, несколько избыточный (как видно из табл. 3.8, достаточно НЕ-И или НЕ –ИЛИ).

С появлением интегральных схем (ЭВМ третьего поколения) начали применяться и другие базисы, в частности базисы Шеффера и Пирса.

Синтез комбинационных устройств в базисах Шеффера и Пирса имеет некоторые особенности. Рассмотрим кратко возможную методику синтеза в этих базисах. Для этого необходимо ввести некоторые новые понятия и определения.

В частности, будем трактовать операцию Шеффера как логическое умножение специального вида и называть его шефферовым умножением. Для удобства записи заменим символ «штрих Шеффера» («/») значком «точка» («•»).

Тогда отрицание любого аргумента (или функции) можно трактовать как шефферов квадрат этого аргумента (или функции):

$$Y_1 = \bar{x}_i = \bar{x}_i + \bar{x}_i = \overline{x_i \cdot x_i} = x_i / x_i = x_i \cdot x_i = x_i^2, \quad (3.10)$$

где $i = 1, 2, \dots, n$.

Любую элементарную конъюнкцию (например с отрицаниями аргументов с четными индексами) в новых обозначениях можно представить следующим образом:

$$Y_2 = x_1 \& \bar{x}_2 \& x_3 \& \dots \& x_n = \left(x_1 \cdot x_2^2 \cdot x_3 \cdot x_4^2 \cdot \dots \cdot x_n \right)^2. \quad (3.11)$$

Аналогично можно представить любую элементарную дизъюнкцию:

$$Y_3 = x_1 \vee \bar{x}_2 \vee x_3 \vee \dots \vee x_n = x_1^2 \cdot x_2 \cdot x_3^2 \cdot x_4 \cdot \dots \cdot x_n^2. \quad (3.12)$$

Можно ввести понятия: элементарного шефферова произведения, шефферовой конститутенты единицы и соседних шефферовых произведений.

Шефферово произведение имеет следующие свойства:

$$\begin{aligned} x_i \bullet 0 &= 1; & x_i \bullet 1 &= x_i^2; & x_i \bullet x_i^2 &= 1; \\ x_i^2 \bullet 0 &= 1; & x_i^2 \bullet 1 &= x_i; & x_i^2 \bullet x_i^2 &= x_i. \end{aligned}$$

К шефферовым произведениям можно применять правила склеивания и поглощения.

Правило склеивания (теорема 1)

Шефферово произведение двух соседних шефферовых произведений можно заменить общей частью исходных произведений, причем если результат склеивания получился одночленным, то он возводится в квадрат.

Пример (для доказательства)

Требуется упростить выражение

$$y = \left(x_1 \cdot x_2 \cdot x_3 \cdot x_4 \right)^2 \times \left(x_1 \cdot x_2 \cdot x_3 \cdot x_4^2 \right)^2.$$

В соответствии с теоремой 1

$$y = \left(x_1 \cdot x_2 \cdot x_3 \right)^2.$$

Проверка:

$$\begin{aligned} y &= \overline{\left(x_1 \& x_2 \& x_3 \& x_4 \right)^2 \& \left(x_1 \& x_2 \& x_3 \& \bar{x}_4 \right)^2} = \\ &= x_1 \& x_2 \& x_3 \& x_4 \vee x_1 \& x_2 \& x_3 \& \bar{x}_4 = \\ &= x_1 \& x_2 \& x_3 = \left(x_1 \cdot x_2 \cdot x_3 \right)^2. \end{aligned}$$

Правило поглощения (теорема 2)

Шефферово произведение двух элементарных шефферовых произведений, из которых одно является собственной частью другого и имеет ранг больше 1, можно заменить одним меньшим сомножителем, причем если результат получится одночленным, то его следует возвести в квадрат (шефферов).

Подробнее об особенностях алгебры Шеффера можно посмотреть в [5].

Синтез комбинационных схем в базисе Шеффера, как и в случае с ОФПН, проводится в 4 этапа (на примере одноразрядного двоичного сумматора ОДС-3).

1-й этап. Составление таблицы истинности (не зависит от логического базиса).

Таблица истинности ОДС-3

i-й разряд 1-го слова	x_{1i}		0	0	0	0	1	1	1	1
i-й разряд 2-го слова	x_{2i}		0	0	1	1	0	0	1	1
Перенос из i-1 разряда	x_{3i}		0	1	0	1	0	1	0	1
Перенос в i+1 разряд	P_i		0	0	0	1	0	1	1	1
Сумма i-х разрядов	S_i		0	1	1	0	1	0	0	1

2-й этап. Составление логико-математического выражения синтезируемого узла по таблице истинности .

Переход (к формуле) возможен как по условиям истинности (канонический вид), так и по условиям ложности – получим шефферовы конституенты. Будем называть такую форму совершенной шефферовой нормальной формой (СШНФ).

Правило перехода от таблицы истинности к СШНФ

Для того чтобы перейти от таблицы истинности к СШНФ логической функции, **необходимо** для тех наборов значений аргументов, которым в таблице истинности соответствуют единичные значения функции, составить шефферовы конституенты единицы таким образом, чтобы символ некоторого i-го аргумента в

j -й конституенте имел бы знак возведения в квадрат, если значение рассматриваемого аргумента в j -м наборе равно 0, после чего полученные конституенты следует заключить в скобки и соединить между собой знаком шепферова произведения.

Если в СШНФ некоторой функции имеется только одна шепферова конституента, то последняя возводится в квадрат.

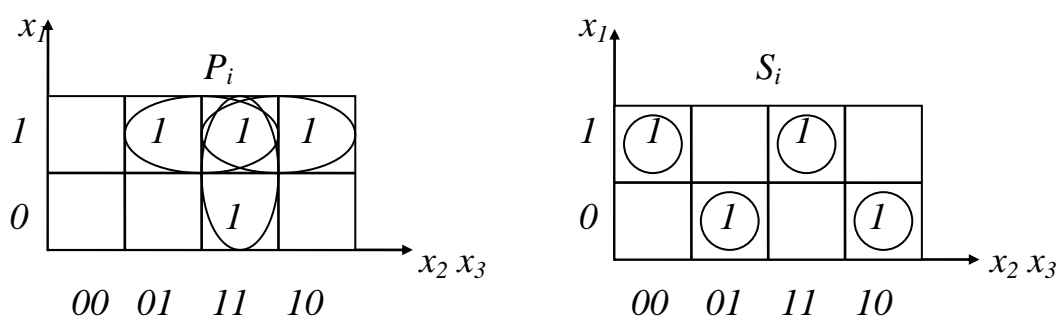
Воспользовавшись этим правилом, можем записать функции P_i и S_i одноразрядного сумматора:

$$P_{i\text{ СШНФ}} = (x_{1i}^2 \cdot x_{2i} \cdot x_{3i}) \cdot (x_{1i} \cdot x_{2i}^2 \cdot x_{3i}) \cdot (x_{1i} \cdot x_{2i} \cdot x_{3i}^2) \cdot (x_{1i} \cdot x_{2i} \cdot x_{3i});$$

$$S_{i\text{ СШНФ}} = (x_{1i} \cdot x_{2i}^2 \cdot x_{3i}^2) \cdot (x_{1i}^2 \cdot x_{2i} \cdot x_{3i}^2) \cdot (x_{1i}^2 \cdot x_{2i}^2 \cdot x_{3i}) \cdot (x_{1i} \cdot x_{2i} \cdot x_{3i}).$$

3-й этап. Оптимизация

Построим таблицы Вейча–Карно для функций P_i и S_i .



В результате минимизации получим тупиковые шепферовы нормальные формы (ТНШФ) исходных функций:

$$P_{i\text{ ТНШФ}} = (x_{1i} \cdot x_{2i}) \cdot (x_{1i} \cdot x_{3i}) \cdot (x_{2i} \cdot x_{3i});$$

$$S_{i\text{ ТНШФ}} = S_{i\text{ СШНФ}}.$$

Отметим, что может быть применен любой из трех методов минимизации.

4-й этап. Построение схемы (рис. 3.9)

Как видно из рис. 3.9, схема на элементах базиса Шеффера будет состоять из элементов одного типа – элемента Шеффера.

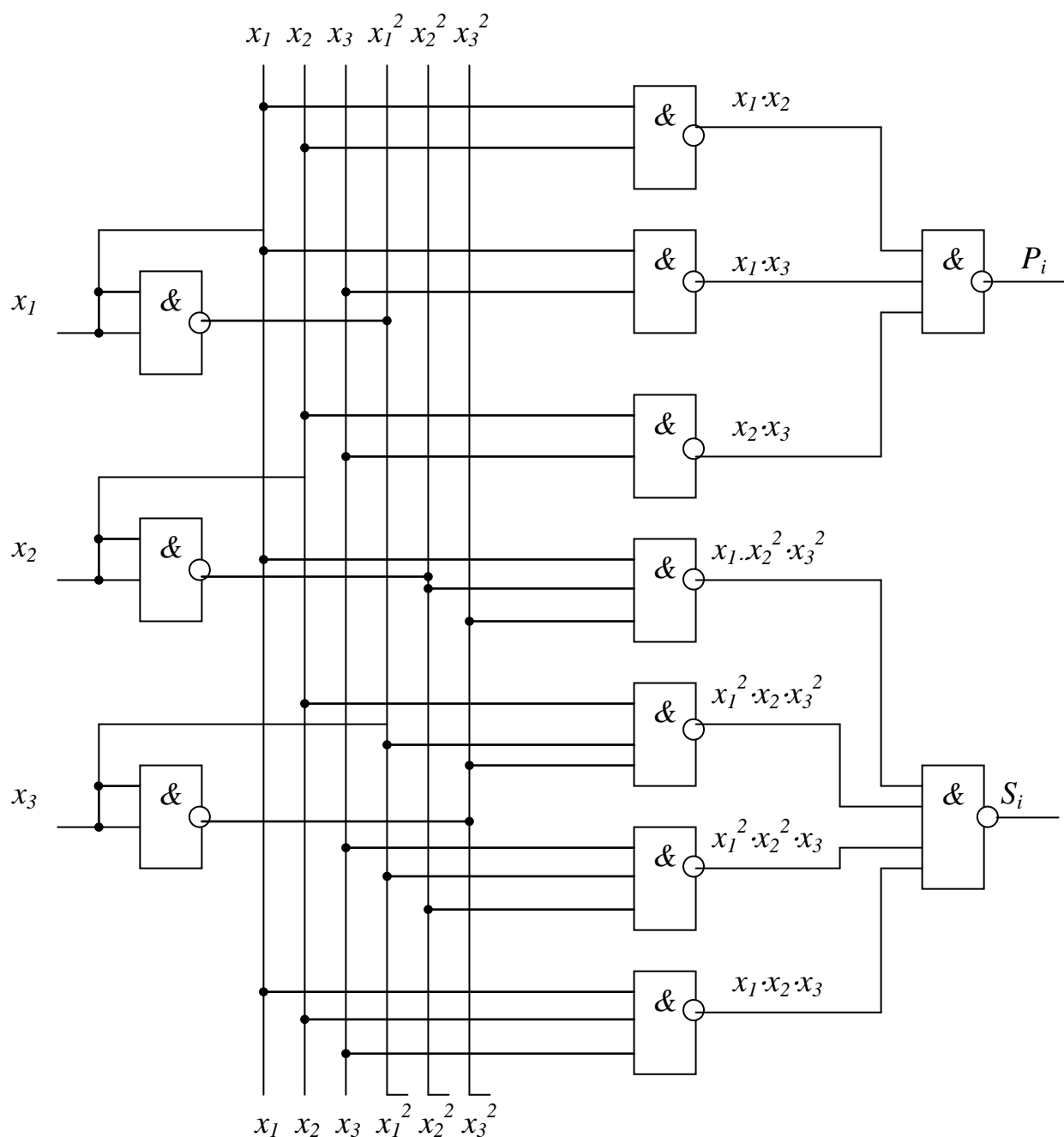


Рис. 3.9

Аналогично можно проводить синтез узлов в базисе Пирса, введя понятия логического пирсова умножения, пирсова квадрата, элементарного пирсова произведения, пирсовой конститuentы нуля (ПК 0), совершенной пирсовой нормальной формы (СПНФ) и тупиковой пирсовой нормальной формы (ТПНФ).

Отметим, что функции (P и S в частности), представленные в СПНФ, будут иметь такой же вид, как в СШНФ, но базис здесь разный.

3.7. Некоторые частные случаи синтеза комбинационных схем

3.7.1. Синтез комбинационных схем с несколькими выходами

На практике часто бывает, что проектируемый узел имеет несколько выходов и, следовательно, описывается не одной, а системой функций.

Тогда кроме индивидуальной минимизации каждой ЛФ появляется возможность совместной минимизации совокупности функций с целью уменьшения оборудования за счет применения одних и тех же логических элементов для реализаций одинаковых компонент различных функций.

Типичный пример: ОДС-3 с двумя выходами (P и S).

$$P_i \text{ СДНФ} = \bar{x}_{1i} \cdot x_{2i} \cdot x_{3i} + x_{1i} \cdot \bar{x}_{2i} \cdot x_{3i} + x_{1i} \cdot x_{2i} \cdot \bar{x}_{3i} + x_{1i} \cdot x_{2i} \cdot x_{3i},$$

$$S_i \text{ СДНФ} = x_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot x_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot x_{3i} + x_{1i} \cdot x_{2i} \cdot x_{3i} \quad (3.13)$$

Эти функции имеют только одну общую конституенту 1:

$$k_7 = x_{1i} \cdot x_{2i} \cdot x_{3i}.$$

Это наводит на мысль сравнить между собой функции \bar{P}_i и S_i :

$$\bar{P}_i \text{ СДНФ} = x_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot x_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot x_{3i} + \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i}. \quad (3.14)$$

В этом случае совпадают уже 3 конституенты, и функция \bar{P}_i является как бы частью функции S_i , если бы не было конституенты $k_0 = \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i}$, т.е. частью S_i является некоторая другая функция P_i^* , равная \bar{P}_i без конституенты k_0 :

$$P_i^* = x_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot x_{2i} \cdot \bar{x}_{3i} + \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot x_{3i}, \quad (3.15)$$

что условно, т.к. в алгебре логики нет операции вычитания, можно записать

$$P_i^* = \bar{P}_i - \bar{x}_{1i} \cdot \bar{x}_{2i} \cdot \bar{x}_{3i}. \quad (3.16)$$

Но это можно заменить другими функциями, например, за счет умножения P_i на конституенту нуля, равную $x_{1i} + x_{2i} + x_{3i}$.

Действительно, эта конституента в $2^n - 1$ случаях из 2^n равна 1. В случае, когда эта конституента нуля равна 0, как раз и обеспечивается значение $P_i^* = 0$ на наборе переменных $x_{1i} + x_{2i} + x_{3i} = 0$.

Следовательно, умножение \bar{P}_i на логическую сумму $x_{1i} + x_{2i} + x_{3i}$ обеспечивает полную нейтрализацию ненужной конституенты k_0 , что эквивалентно ее вычитанию:

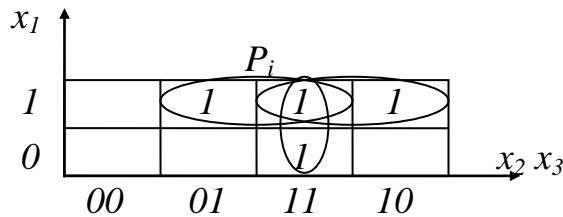
$$P_i^* = \bar{P}_i \cdot (x_{1i} + x_{2i} + x_{3i}). \quad (3.17)$$

Тогда можно записать

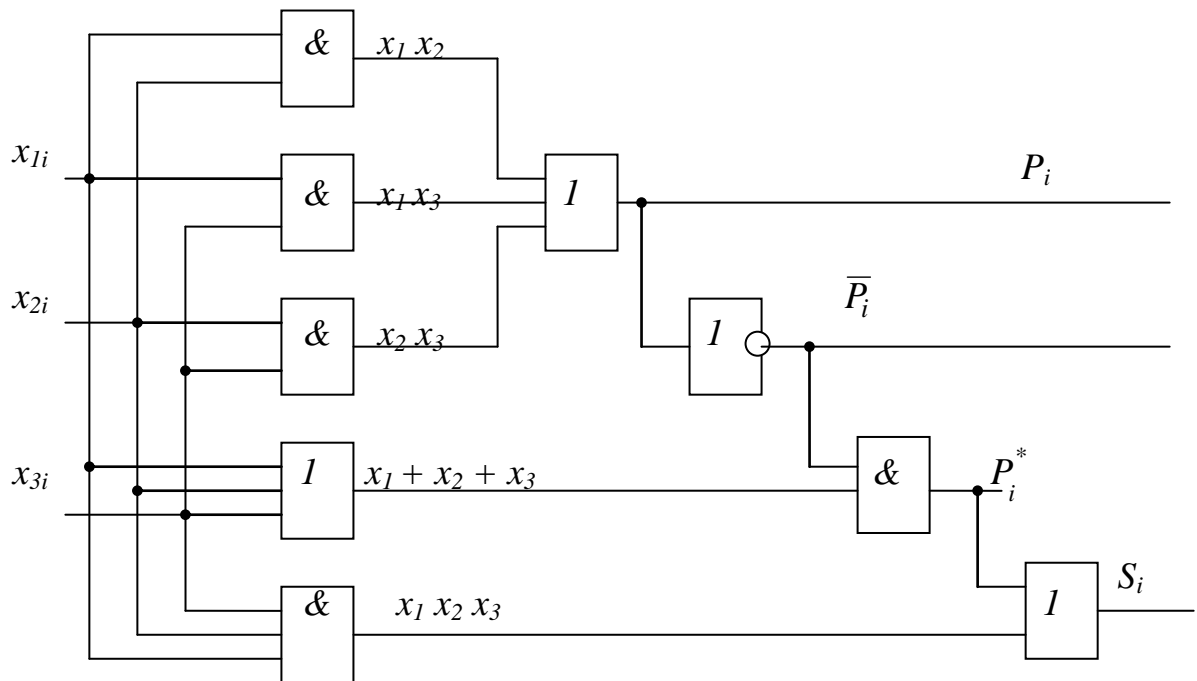
$$S_i = P_i^* + (x_{1i} \cdot x_{2i} \cdot x_{3i}) = \bar{P}_i \cdot (x_{1i} + x_{2i} + x_{3i}) + (x_{1i} \cdot x_{2i} \cdot x_{3i}), \quad (3.18)$$

а P_i в соответствии с таблицей Вейча–Карно, приведенной ниже, определяется выражением

$$P_i = x_{1i} \cdot x_{2i} + x_{1i} \cdot x_{3i} + x_{2i} \cdot x_{3i}. \quad (3.19)$$



Полученные выражения дают высокую степень совмещения логических элементов в схеме ОДС-3 и, следовательно, экономию оборудования (рис. 3.10).



(20 транзисторов)

Рис. 3.10

Эта схема получила название *сумматор фон Неймана*.

Примечание.

В обычной (минимизированной по отдельным выходам) схеме было представлено (см. рис. 3.6):

3	схемы НЕ	– 3 входа,
3	2-входовые схемы И	– 6 входов,
4	3-входовые схемы И	– 12 входов,
1	3-входовая схема ИЛИ	– 3 входа,
1	4-входовая схема ИЛИ	– 4 входа.

Всего 12 схем 28 входов (транзисторов).

В общем случае рекомендуется следующий порядок совместной минимизации нескольких функций:

- 1) производится раздельная минимизация каждой функции (одним из 3-х методов);
- 2) все функции (после шага 1) располагаются по степени сложности, начиная с простой, далее – преобразование каждой функции для выражения ее через более простую соседнюю, применяя различные логические преобразования;
- 3) на каждом этапе преобразования следует проводить сравнение получающихся вариантов с целью контроля эффективности действий.

3.7.2. Синтез комбинационных схем, характеризующихся не полностью определенной функцией

На практике бывают случаи, когда ЛФ определена не на всех наборах аргументов, т.е. *ЛФ не полностью определена*.

Комбинации значений x_1, x_2, \dots, x_n , на которых ЛФ не определена, называются *избыточными*.

Например, требуется синтезировать преобразователь тетрад десятично-двоичного кода 8421 в двоичные комбинации 8421 + 6.

1. Обозначим входные тетрады через

$$X = x_4 x_3 x_2 x_1,$$

а выходные – через

$$Y = y_4 y_3 y_2 y_1,$$

где x_i, y_i – двоичные цифры (переменные).

Следовательно, у рассматриваемого устройства будет 4 входа и 4 выхода.

2. Построим таблицу истинности. Ясно, что возможных наборов из 4-х переменных будет 16, а количество тетрад десятично-двоичных цифр – 10. Следовательно, шесть старших наборов $x_4x_3x_2x_1$ будут избыточными. В соответствующих наборах $y_4y_3y_2y_1$ ставятся прочерки.

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_4		0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1
x_3		0	0	0	0	1	1	1	1	0	0	0	0	1	1	0	1
x_2		0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_1		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
y_4		0	0	1	1	1	1	1	1	1	1	-	-	-	-	-	-
y_3		1	1	0	0	0	0	1	1	1	1	-	-	-	-	-	-
y_2		1	1	0	0	1	1	0	0	1	1	-	-	-	-	-	-
y_1		0	1	0	1	0	1	0	1	0	1	-	-	-	-	-	-

Избыточные наборы

Так как избыточные наборы никогда не появляются на входе преобразователя, то им можно условно в таблице истинности ставить в соответствие любые значения истинности функции (0 или 1), так как это никак не повлияет на работу устройства. Говорят, что в этом случае ЛФ *доопределяется*. Можно доопределить наборы целенаправленно, чтобы формула доопределенной функции максимально упростилась. Это наиболее удобно производить при помощи таблицы Вейча–Карно.

При этом можно рекомендовать следующий порядок действий для каждой функции:

- произвести запись значений ЛФ (как 1, так и 0) для всех неизбыточных наборов аргументов в таблицу Вейча–Карно;
- доопределить ЛФ по таблице либо 1, либо 0 так, чтобы отмеченные клетки составляли по возможности наиболее крупные совокупности прямоугольников по 2^i клеток. Необходимо, чтобы в отмеченные конфигурации

обязательно входила хотя бы одна реализация функции, соответствующая неизбыточному набору;

- по таблице Вейча–Карно произвести запись тупиковой ДНФ или КНФ доопределенной ЛФ обычными способами.

В соответствии с этими рекомендациями построим таблицы Вейча–Карно для каждой из функций y_4, y_3, y_2, y_1 .

$y_4(x_4 x_3 x_2 x_1)$

$x_4 x_3$	10	11	01	00
10	1	1		
11				
01	1	1	1	1
00	0	0	1	1
	00	01	11	10

$x_2 x_1$

$y_3(x_4 x_3 x_2 x_1)$

$x_4 x_3$	10	11	01	00
10	1	1		
11			1	1
01	0	0	1	1
00	1	1		
	00	01	11	10

$x_2 x_1$

$y_2(x_4 x_3 x_2 x_1)$

$x_4 x_3$	10	11	01	00
10	1	1		
11	1	1		
01	1	1	0	0
00	1	1	0	0
	00	01	11	10

$x_2 x_1$

$y_1(x_4 x_3 x_2 x_1)$

$x_4 x_3$	10	11	01	00
10		1	1	
11		1	1	
01	0	1	1	0
00	0	1	1	0
	00	01	11	10

$x_2 x_1$

3. Проведем минимизацию функций преобразователя кода.

Из таблиц Вейча–Карно для функций y_1, y_2, y_3, y_4 видно, что можно эффективно доопределить 3 функции: y_1, y_2, y_3 . Единицы («1») в доопределенных клетках таблиц показаны жирным шрифтом.

Функцию же y_4 оказалось целесообразнее записать по нулям (т.е. в СКНФ) без доопределения.

Тогда

$$y_4 = x_2 + x_3 + x_4;$$

$$y_3 = \bar{x}_2 \cdot \bar{x}_3 + x_2 \cdot x_3;$$

$$y_2 = \bar{x}_2;$$

$$y_1 = x_1.$$

4. Схема синтезированного преобразователя тетрад десятично-двоичного кода 8421 в двоичные комбинации 8421+6 приведена на рис. 3.11.

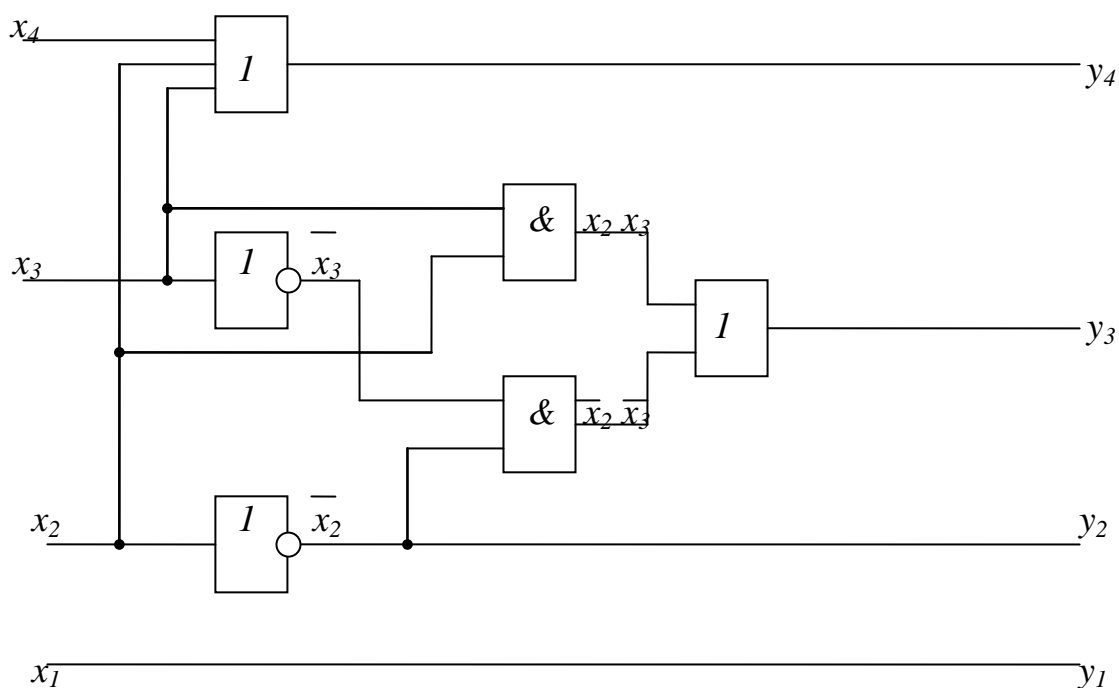


Рис. 3.11

3.8. Принципы построения функциональных устройств

3.8.1. Накапливающие схемы

Основным элементом накапливающих схем (некомбинационных) является *триггер (T)* – элемент с двумя устойчивыми состояниями. Переход триггера из одного состояния в другое происходит скачкообразно, с изменением уровня выходного напряжения.

Имеется несколько типов триггеров (табл. 3.9).

Все типы триггеров могут быть *асинхронными*, *синхронизируемыми*, *однотактными* или *двухтактными* [3].

Для описания функционирования триггеров используются таблицы переходов. Пример переходов для *R-S*-триггера приведен в табл. 3.10, а функциональные схемы и графические обозначения *R-S*-триггеров различных типов (однотактных асинхронного и синхронизируемого, двухтактного синхронизируемого) – на рис. 3.12.

Таблица 3.9

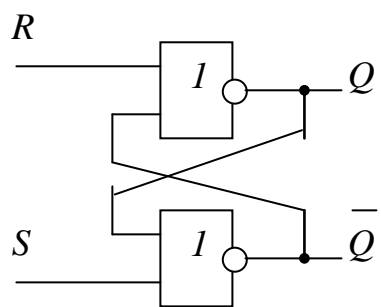
Типы триггеров

Наименование триггера	Обозначение триггера	Уравнение, описывающее функционирование триггера	Примечания
1 Триггер с установочными входами	R-S -триггер	$Q(t+1) = S(t) \vee Q(t) \cdot \bar{R}(t)$ Причем $S(t) R(t) = 0$	$Q(t)$ – выход триггера в момент t S – установка в 1 R – установка в 0
2 Триггер со счетным входом	T -триггер	$Q(t+1) = Q(t) \cdot \bar{T}(t) \vee \bar{Q}(t) \cdot T(t)$	$T(t)$ – счетный вход триггера
3 Триггер с функцией задержки	D -триггер	$Q(t+1) = D(t)$	$D(t)$ – входной сигнал
4 Универсальный триггер	J-K -триггер	$Q(t+1) = \bar{K}(t) \cdot Q(t) \vee J(t) \cdot \bar{K}(t) \cdot Q(t)$	У J-K -триггера есть возможность отдельных установок в 1 и 0 (R и S) При $J = K = 1$ J-K -триггер работает как T -триггер

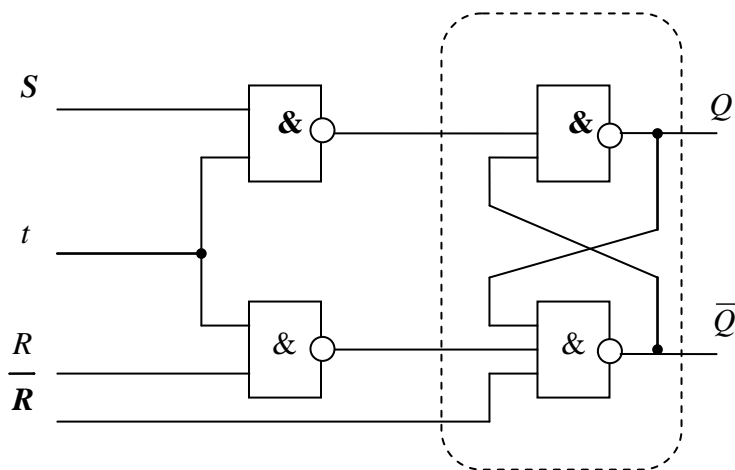
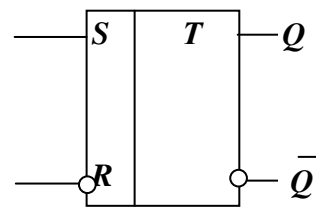
Таблица 3.10

Таблица переходов для R-S-триггера

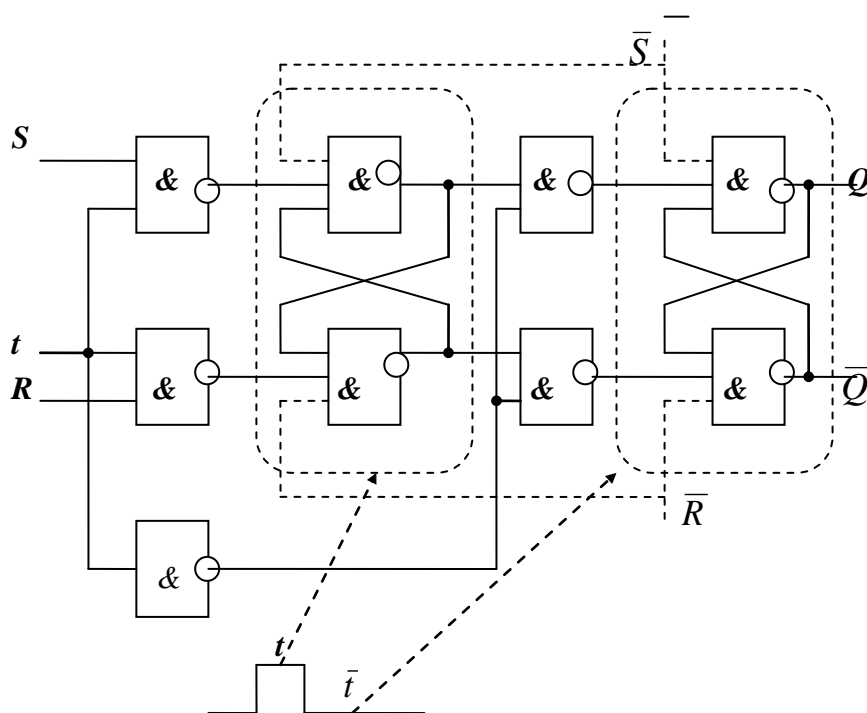
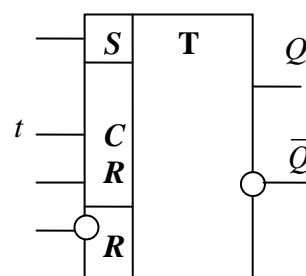
t		$t + 1$	Примечания
R	S	Q	
0	0	$Q(t)$	Хранение
0	1	1	Установка 1
1	0	0	Установка 0
1	1	–	Запрещено



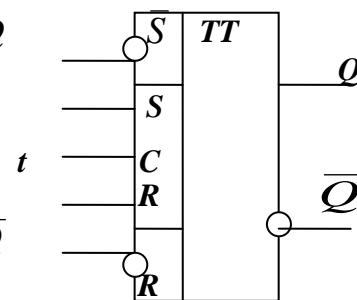
a



б



в



Двухтактный
синхронизируемый
R-S-триггер
на элементах И-НЕ

Рис. 3.12

На рис. 3.12 приняты следующие обозначения:

S – раздельная установка триггера в 1;

R – раздельная установка триггера в 0;

C – вход синхронизации;

Q – единичный выход триггера;

\overline{Q} – нулевой выход триггера.

Схема R - S -триггера составляет основу (базу) для построения других триггерных схем, таких как T -, D - и J - K -триггеры.

Рассмотрим кратко особенности этих триггеров.

T -триггер – триггер со счетным входом (от слова *toggle* – переключать).

Как видно из табл. 3.9, T -триггер должен суммировать по М2 (\oplus) сигналы состояния триггера Q и входного сигнала T . Следовательно, единичный входной сигнал T меняет состояние триггера на противоположное, а нулевой (\overline{T}) – не изменяет состояние триггера.

Переходы T -триггера приведены в табл. 3.11.

Таблица 3.11

Таблица переходов для T -триггера

t	$t+1$	Примечания
T	Q	
0	$Q(t)$	Сохранение состояния $Q(t)$
1	$\overline{Q}(t)$	Изменение состояния на \overline{Q}
0	$\overline{Q}(t)$	Сохранение состояния \overline{Q}
1	$Q(t)$	Изменение состояния на $Q(t)$

Функциональные схемы и условные обозначения T -триггеров различного типа (асинхронного и синхронизируемого) приведены на рис. 3.13

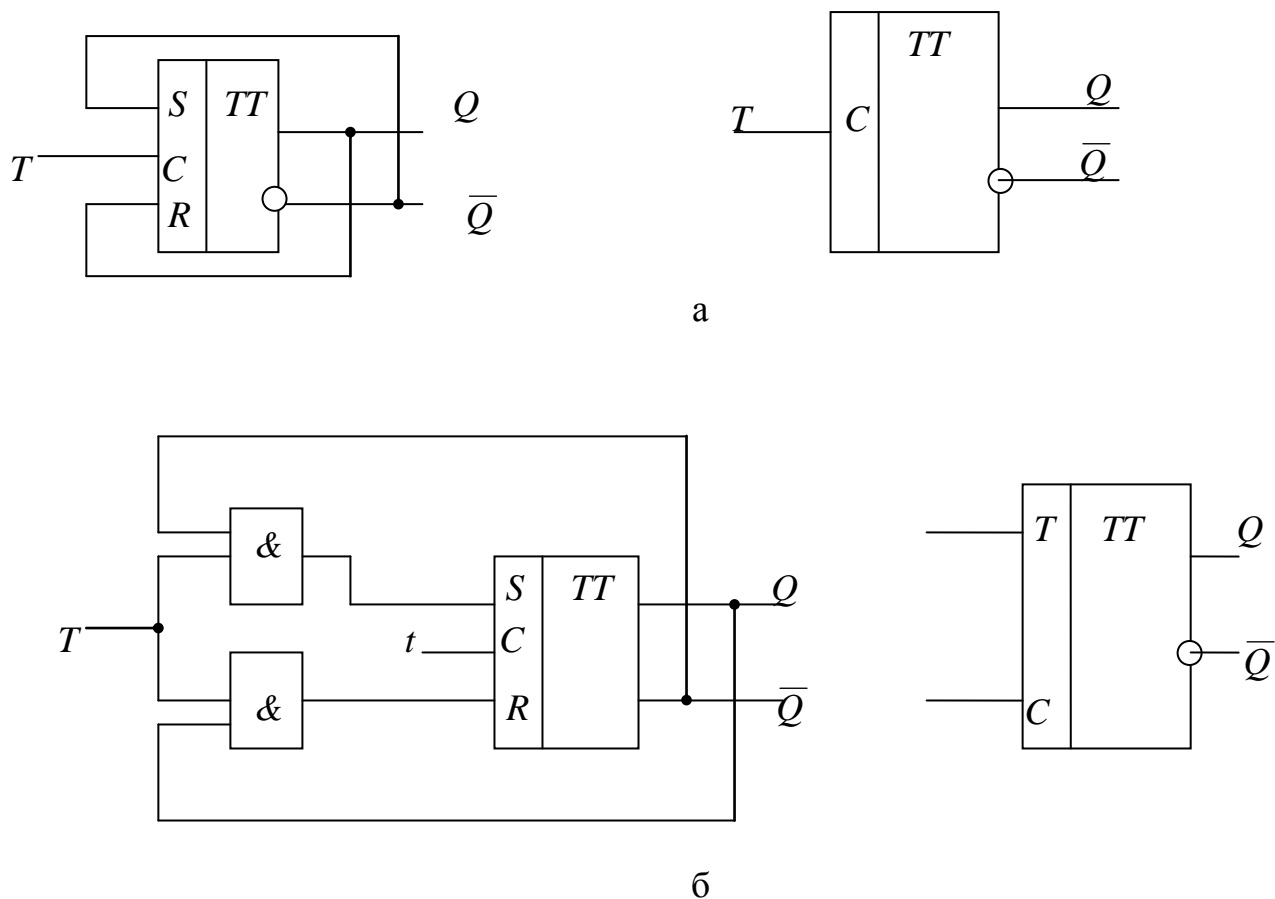


Рис. 3.13

D-триггер (от слова *delay* – задержка) реализует функцию задержки. В отличие от *R-S*-триггера, имеющего режимы установки «1», установки «0» и хранения предыдущего состояния, *D*-триггер имеет режимы только установки «1» и установки «0». Можно считать, что *D*-триггер соответствует *R-S*-триггеру, работающему только в режиме установки, т.е. с $R = 1$ и $S = 0$ или с $R = 0$ и $S = 1$. Применяется синхронизируемый одноклапный или двухклапный *D*-триггер, обеспечивающий задержку установки триггера на половину или полный такт синхронизации.

Функциональные схемы и условные обозначения *D*-триггеров (синхронизируемых одноклапного и двухклапного) приведены на рис. 3.14.

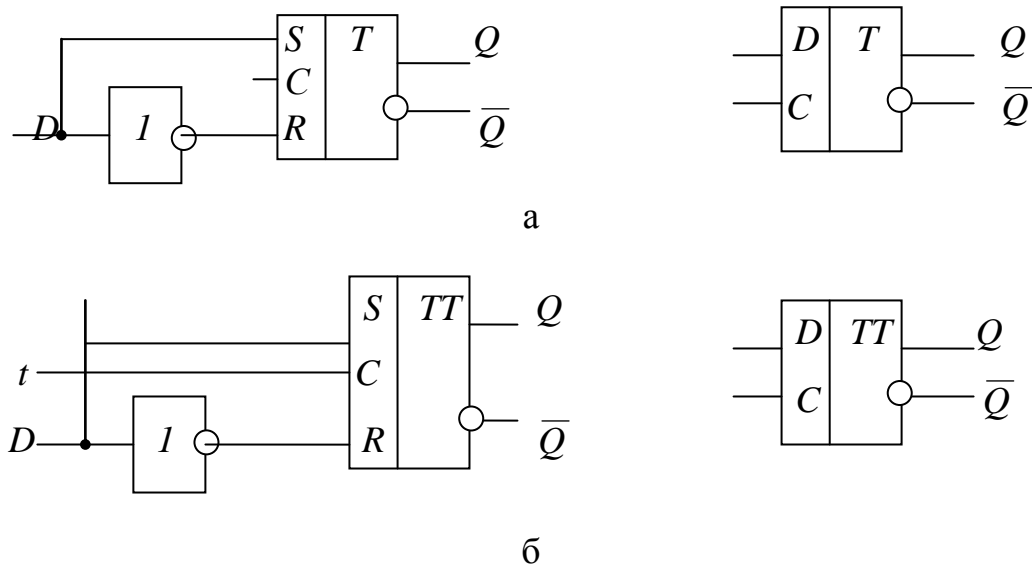


Рис. 3.14

J-K-триггер является универсальным триггером. У этого триггера имеются входы несинхронизируемой установки R и S , с помощью которых при $C = 0$ триггер может быть установлен в состояние «1» путем подачи $R = 0$ и $S = 1$, либо в состояние «0» путем подачи $R = 1$ и $S = 0$. При подаче сигналов $J = K = 1$, не меняющих состояние схемы, работа триггера осуществляется под воздействием синхронизируемых входов $C = 1$.

Входы J и K соответствует входам S и R R - S -триггера, однако в отличие от R - S -триггера в J - K -триггере единичные сигналы могут одновременно прийти на входы J и K , при этом J - K -триггер работает как T -триггер.

Переходы J - K -триггера приведены в табл. 3.12.

Таблица 3.12

Таблица переходов для J - K -триггера

t		$t+1$	Примечания
J	K	Q	
0	0	$Q(t)$	Хранение
0	1	0	Установка в 0
1	0	1	Установка в 1
1	1	$\overline{Q}(t)$	Инверсия

Функциональная схема и условное графическое обозначение J - K -триггера приведены на рис. 3.15.

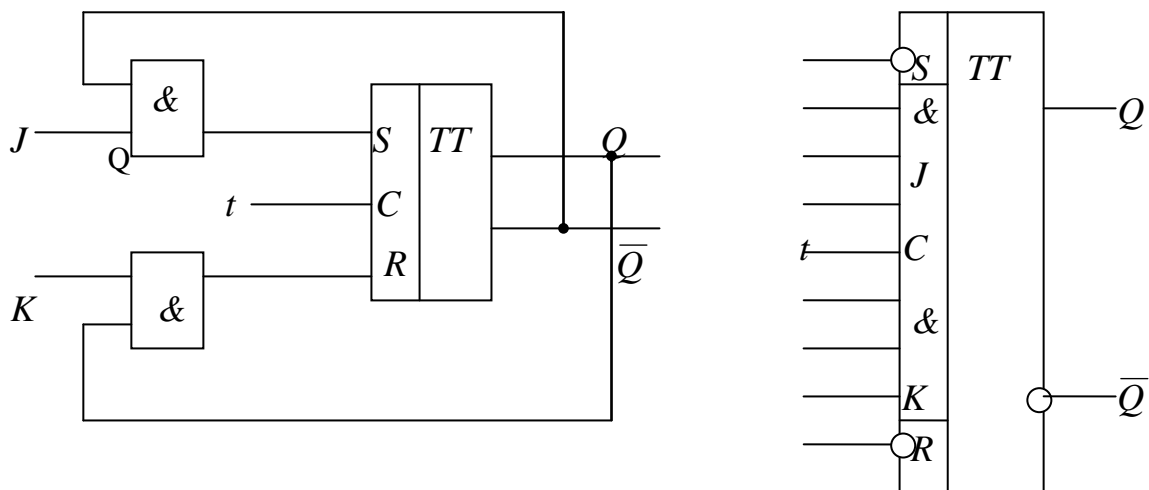


Рис. 3.15

J-K-триггер может быть использован для построения триггеров других типов. Примеры применения *J-K*-триггера для построения других триггеров (*D*, *T*, *R-S*) приведены на рис. 3.16

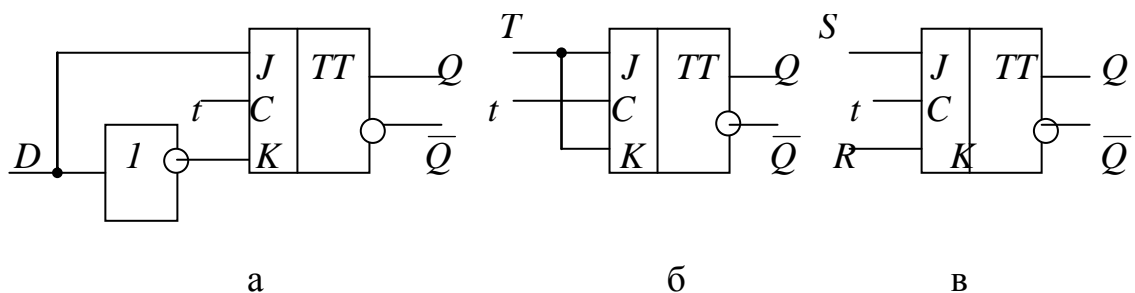


Рис. 3.16

3.8.2. Построение различных функциональных устройств с использованием триггеров

С использованием накопительных элементов (триггеров) строятся различные функциональные устройства, такие, как *регистры* с различными функциями, *сдвигатели*, *счетчики*. Широко применяются также *дешифраторы*, которые относятся к комбинационным схемам.

Рассмотрим примеры применения элементов памяти для построения различных функциональных устройств.

1. Регистр с приемом и предварительным сбросом

Схема регистра такого типа приведена на рис. 3.17.

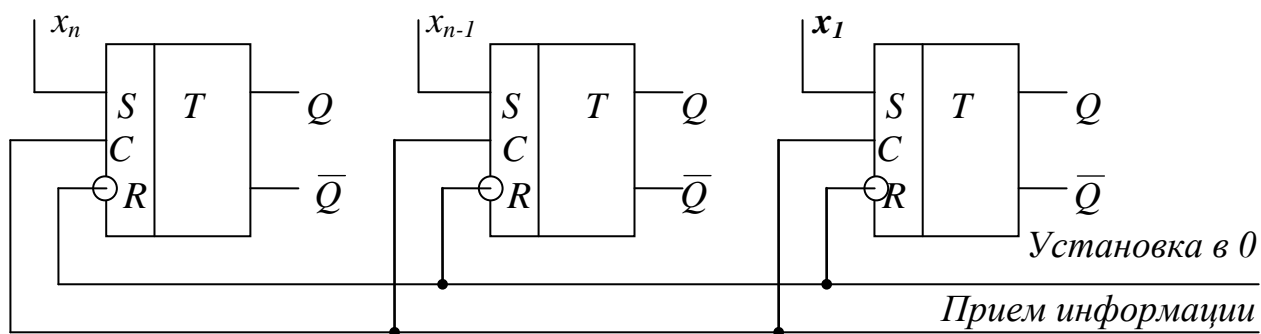


Рис. 3.17

Примечание.

Такой регистр может быть и без предварительного сброса.

На схемах применяется условное обозначение регистров, приведенное на рис. 3.18.

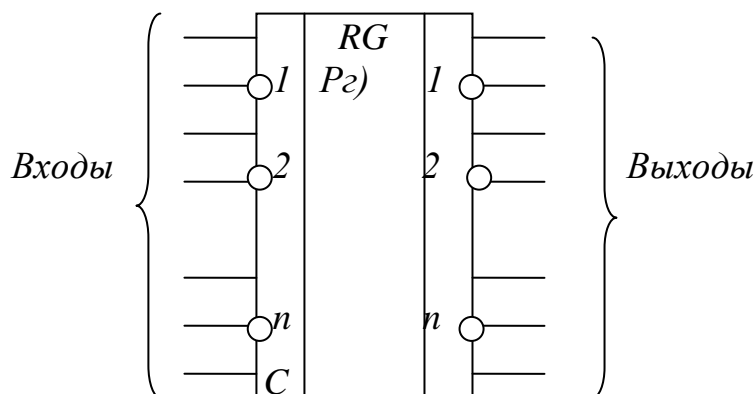


Рис. 3.18

2. Сдвигающий регистр

Пример схемы сдвигающего регистра на R - S -триггерах приведен на рис. 3.19.

3. Счетчики

Пример построения суммирующего 4-разрядного счетчика на J - K -триггерах приведен на рис. 3.20.

Таблица 3.13

Таблица переходов триггеров счетчика

$x_{сч}$	Q_4	Q_3	Q_2	Q_1					$x_{сч}$	Q_4	Q_3	Q_2	Q_1				
0	0	0	0	0					8	1	0	0	0				
1	0	0	0	1					9	1	0	0	1				
2	0	0	1	0					10	1	0	1	0				
3	0	0	1	1					11	1	0	1	1				
4	0	1	0	0					12	1	1	0	0				
5	0	1	0	1					13	1	1	0	1				
6	0	1	1	0					14	1	1	1	0				
7	0	1	1	1					15	1	1	1	1				

4. Дешифраторы

Напомним, что дешифраторы относятся к комбинационным схемам. Здесь мы их приведем как пример широко применяемых функциональных устройств.

Функциональная схема дешифратора и его условное графическое обозначение приведены на рис. 3.21.

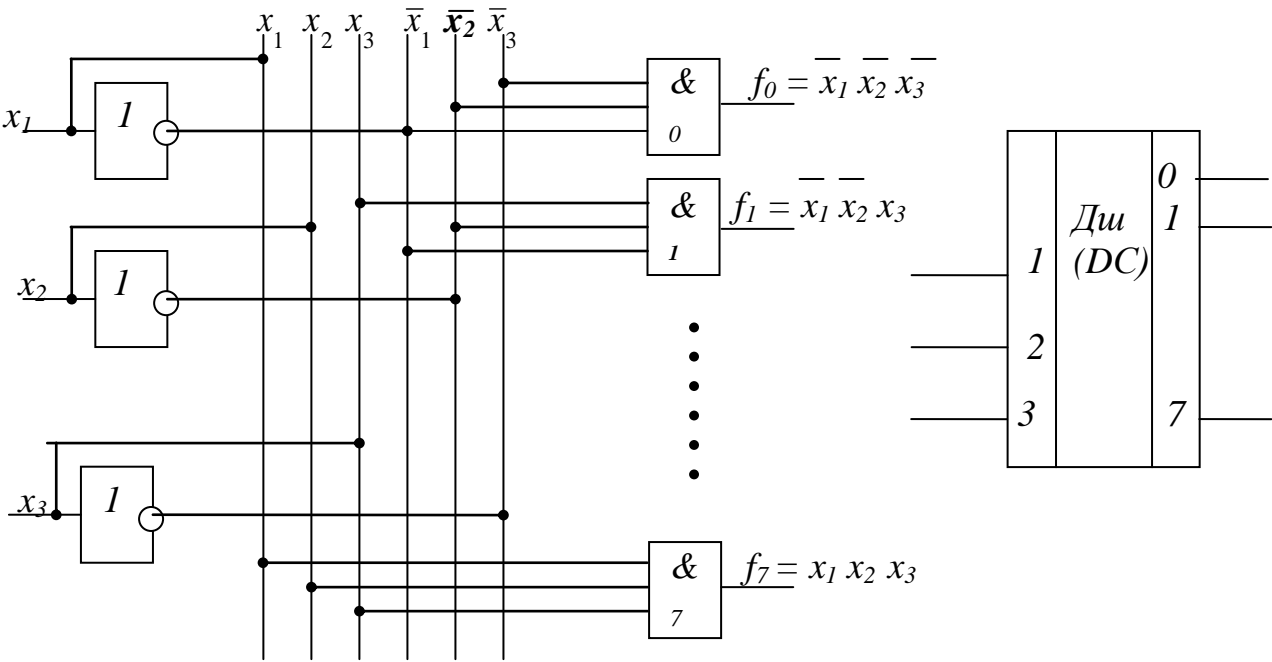


Рис. 3.21

Могут быть также и каскадный дешифратор, дешифратор с инверсными выходами и другие.

Подробнее накапливающие элементы и различные функциональные устройства рассмотрены в [3].

3.9. Синтез цифровых автоматов

3.9.1. Определения

Цифровой автомат (ЦА) – это обобщение для всех видов устройств обработки цифровой информации, имеющих программное управление.

Математически ЦА определяется как множество из 5-ти элементов:

$$A = \{ X, S, Y, f, \varphi \}, \quad (3.20)$$

где $X = \{ X_1, X_2, \dots, X_k \dots X_K \}$ – множество входных сигналов – входной алфавит автомата;

$S = \{ S_1, S_2, \dots, S_l \dots S_L \}$ – множество внутренних состояний – внутренний алфавит автомата;

$Y = \{ Y_1, Y_2, \dots, Y_m \dots Y_M \}$ – множество выходных сигналов – выходной алфавит автомата;

f – функция переходов, реализующая отображение множества $P_f \in X \times S$ в S ;

φ – функция выходов, реализующая отображение множества $P_\varphi \in X \times S$ в Y .

Автомат называется *конечным*, если конечны образующие его множества.

Отдельные сигналы называются *буквами алфавита*, а конечные совокупности сигналов – *словами* в заданных алфавитах.

Таким образом, цифровой автомат осуществляет переработку слов, представленных в алфавите X , в слова – в алфавите Y . Состояние автомата изменяется скачкообразно (дискретно).

Работа ЦА производится (осуществляется) во времени – по тактам – $0, 1, \dots$

Время, измеряемое тактами, называется *автоматным временем*.

Закон функционирования автомата можно выразить следующим образом:

$$S(t) = f[S(t-1), X(t-1)]; \quad (3.21)$$

$$Y(t) = \varphi[S(t), X(t)]. \quad (3.22)$$

3.9.2. Задание ЦА с памятью

Комбинационные устройства, рассмотренные ранее, являются частным случаем ЦА без памяти.

Для схем, содержащих память, аппарата булевой алгебры недостаточно. На помощь в этом случае приходит математический аппарат общей теории цифровых автоматов (см. формулы 3.20 – 3.22).

Функции переходов $f(t)$ и выходов $\varphi(t)$ дают полное описание ЦА, если они каким-то способом заданы. Наиболее удобно это делать таблично. Строят 2 таблицы – *переходов* и *выходов*. Обе таблицы являются двухкоординатными. Обычно число горизонтальных входов берется равным количеству возможных X_i , а количество вертикальных входов равно количеству внутренних состояний ЦА (табл. 3.15 и 3.16 соответственно).

Таблица 3.14

Таблица переходов

Входные сигналы	Внутренние состояния	
	S_1	S_2
X_1	S_1	S_1
X_2	S_2	S_2
X_3	S_1	S_2

Таблица 3.15

Таблица выходов

Входные сигналы	Внутренние состояния	
	S_1	S_2
X_1	Y_1	Y_1
X_2	Y_1	Y_1
X_3	Y_1	Y_2

В клетках, соответствующих любой паре входных координат (на пересечении строк и столбцов), проставляются значения выходной величины: $S(t+1)$ – в таблице переходов и $Y(t)$ – в таблице выходов.

ЦА, описываемые зависимостями (3.21) и (3.22) и задаваемые таблицами вида табл. 3.14 и 3.15, называются *автоматами Мили*.

На практике иногда удобнее функцию выходов трактовать как функцию только одного аргумента – внутреннего состояния автомата. Тогда можно выразить ее следующим образом:

$$Y(t) = \varphi[S(t)], \quad (3.23)$$

поскольку влияние сигнала $X(t-1)$ все равно скажется через величину $S(t-1)$.

ЦА, описываемые формулами (3.21) и (3.23), называются *автоматами Мура*.

Можно показать, что для любого автомата Мили существует эквивалентный ему автомат Мура и наоборот.

Для автоматов Мура таблица выходов вырождается в одну строчку соответствия внутренних состояний и выходных сигналов. Поэтому для них строится обобщенная таблица, называемая обычно *отмеченной таблицей переходов* (табл. 3.16).

Таблица 3.16

Отмеченная таблица переходов

Входной сигнал	Внутренние состояния и выходные сигналы	
	$S_1(Y_1)$	$S_2(Y_2)$
X_1	S_1	S_1
X_2	S_1	S_1
X_3	S_2	S_2

3.9.3. Структурный синтез ЦА с памятью

3.9.3.1. Выбор функционально полной системы элементов для синтеза ЦА с памятью

Если ЦА без памяти, то может быть применена любая функционально полная система элементов.

Если же ЦА с памятью, то дополнительно необходимо элементарное устройство памяти, которое имеет b состояний (где b – основание системы счисления) и, следовательно, может запомнить любую букву структурного алфавита.

В теории ЦА доказана следующая **теорема**.

Для того чтобы система была функционально полной для структурного синтеза ЦА любой наперед заданной сложности, необходимо и достаточно, чтобы она содержала:

- логические элементы, образующие функционально полную систему для синтеза *комбинационной схемы*;
- запоминающие элементы, обладающие полной системой переходов и выходов.

Считают, что ЦА обладает *полной системой переходов*, если для каждой пары его внутренних состояний S_i и S_j найдется хотя бы один входной сигнал, который переводит автомат из состояния S_i в S_j .

Автомат Мура обладает *полной системой выходов*, если в каждом внутреннем состоянии он вырабатывает выходной сигнал, отличный от всех других своих выходных сигналов.

Нетрудно убедиться, что триггер является простейшим автоматом Мура и при этом обладает *полной системой переходов и выходов*.

Следовательно, любая функционально полная система элементов, дополненная триггером, является функционально полной для синтеза ЦА с памятью. Вместо триггера можно использовать *линии задержки (ЛЗ)* – простейшие запоминающие устройства с *полной системой переходов и выходов*.

Триггер можно использовать как элементарный автомат (с 1, 2, 3 входами).

В связи с особым положением триггеров в ЭВМ напомним таблицы переходов и аналитические описания нескольких типов триггеров (см. п. 3.8.1).

Будем строить отмеченные таблицы переходов аналогично тому, как строили таблицы истинности для логических элементов (табл. 3.17 и 3.18).

Таблица 3.17

Таблица переходов T-триггера

1-й аргумент– $S(t-1)$	Q^*	0	0	1	1
2-й аргумент–сигнал на счетном входе в такте $t-1$	T	0	1	0	1
Функция – $S(t)$	Q	0	1	1	0

Такая форма таблицы дает возможность записать логическое уравнение T-триггера по правилам для комбинационных схем.

В СДНФ T-триггер будет описываться следующей булевой функцией:

$$Q = \overline{Q}^* \cdot T \vee Q^* \cdot \overline{T}. \quad (3.24)$$

Из формулы (3.24) видно, что T-триггер – это сумматор по модулю 2 накапливающего типа, (т.е. для двоичных сигналов, приходящих последовательно во времени).

J-K-триггер описывается следующим выражением:

$$Q = Q^* \cdot \overline{K} \vee \overline{Q}^* \cdot J, \quad (3.25)$$

что показывает, что J - K -триггер является универсальным, имеющим два режима:

- 1) R - S -режим (режим записи и хранения), если $J \neq K$ и триггер заранее обнулен;
- 2) T -режим (счета), если $J = K$.

Таблица 3.18

Таблица переходов J - K -триггера

1-й аргумент – $S(t-1)$	Q^*	0	0	0	0	1	1	1	1
2-й аргумент – $X_1(t-1)$	J	0	0	1	1	0	0	1	1
3-й аргумент – $X_0(t-1)$	K	0	1	0	1	0	1	0	1
Функция – $S(t)$	Q	0	0	1	1	1	0	1	0

3.9.3.2. Канонический метод структурного синтеза ЦА

Этот метод разработан В.М. Глушковым.

Исходные данные :

- словесное описание синтезируемого автомата;
- таблицы переходов элемента памяти;
- логический базис для комбинационной схемы.

Метод основан на разделении автомата на две части:

- комбинационную схему;
- память автомата.

Далее по количеству внутренних состояний ЦА и основанию структурного алфавита определяется количество элементов памяти. В результате синтез ЦА с памятью сводится к синтезу комбинационной схемы. Процесс синтеза ЦА с памятью можно разделить на 4 этапа.

1-й этап

На этапе осуществляется:

а) переход от словесного описания к его формализованному описанию в виде таблицы переходов и выходов с учетом аспектов его функционирования. Далее производится выбор структурного алфавита и кодировка таблиц (в ЭВМ это двоичная кодировка);

б) переход от схемы абстрактного автомата, информация о котором ограничена сведениями о количестве входных сигналов, внутренних состояний и

выходных сигналов, к схеме структурного автомата, у которого может быть определено множество входов $V = \{V_1, V_2, \dots, V_e\}$, множество элементов памяти $Q = \{q_1, \dots, q_n\}$ и множество выходов $Z = \{z_1, \dots, z_w\}$ (рис.3.22).

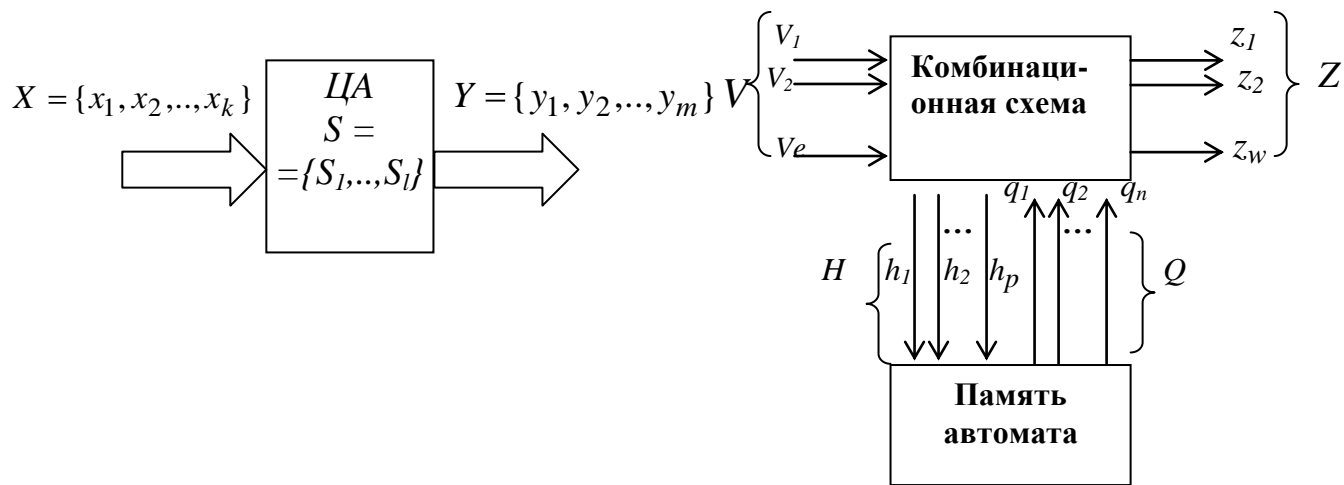


Рис. 3.22

Для этого перехода по количеству входных сигналов k , внутренних состояний l и выходных сигналов m (эти данные выбираются из таблицы переходов и выходов) и по величине основания b рассчитывается количество входов, элементов памяти и выходов из ЦА:

$$\left. \begin{array}{l} \text{количество входов} \quad e \geq \log_b k; \\ \text{количество состояний} \quad n \geq \log_b l; \\ \text{количество выходов} \quad w \geq \log_b m. \end{array} \right\} \quad (3.26)$$

Отметим одно важное обстоятельство.

Для правильного функционирования схемы из двух частей должно выполняться следующее неперенное условие:

- сигналы на входе памяти не должны непосредственно участвовать в образовании ее выходных сигналов.

Как следствие указанного условия – необходимость использования в качестве элементов памяти автомата Мура. Кроме того, количество сигналов на выходе памяти должно всегда равняться количеству ее внутренних состояний.

2-й этап

Определяется количество входов у элементов памяти (количество выходов обычно равно 1):

$$e_{\text{эн}} \geq \log_b k_{\text{эн}} ,$$

где $k_{эн}$ – количество входных сигналов, необходимых для записи информации.

Иногда количество входов у элемента памяти определяется его *типом*.

После этого подсчитывается количество связей, поступающих из комбинационной схемы на память автомата:

$$p = e_{эн} \cdot n \quad (3.27)$$

(количество связей, поступающих из памяти на комбинационную схему, равно n).

Таким образом, этап 2 завершается раскрытием структуры памяти ЦА.

На этом же этапе происходит оформление комбинационной схемы как «черного ящика»: выявляется, что комбинационная схема имеет общее количество входов

$$e_{общ} = e + n,$$

а общее количество выходов

$$W_{общ} = w + p.$$

3-й этап

Строится таблица возбуждения памяти ЦА на основе таблиц переходов автомата и элемента памяти. По сути таблица возбуждения памяти является частью таблицы истинности комбинационной схемы. Она задает соответствие между входными сигналами (словами) ЦА и его памяти. Таблица возбуждения памяти автомата имеет те же входы (а следовательно, и форму), что и таблица переходов автомата, но в ее клетках вместо слов внутреннего состояния автомата проставляются слова из входных сигналов, приходящих на элементы памяти (которые обычно называют *сигналами возбуждения памяти*) и обеспечивающих переход автомата (т.е. его памяти) в соответствующее состояние из предыдущего.

Если обозначить множество сигналов возбуждения памяти через $H = \{h_1, \dots, h_p\}$, то в общем виде можно записать

$$h_j = \Psi(v_1, v_2, \dots, v_e), \quad (3.28)$$

где $j = 1, 2, \dots, p$;

Ψ – символ некоторой булевой функции.

Например, при двоичном структурном алфавите в одной из клеток таблицы переходов ЦА указано внутреннее состояние $S_i = 101$, причем по выходным данным таблицы видно, что в это состояние ЦА переходит из состояния $S_{i-1} = 011$.

Анализ исходного и конечного состояний показывает, что:

1) память автомата состоит из 3-х двоичных элементов памяти;

2) первый элемент памяти (триггер младшего разряда) переходит из состояния 1 снова в состояние 1; второй – из состояния 1 в 0, а третий – из 0 в 1.

Для данного конкретного случая будем считать, что каждый элемент памяти имеет только один вход. Тогда по таблице переходов находим те значения входного сигнала h_j , которые обеспечивают перевод элементов памяти из 1 в 1 (например $h_1=0$), из 1 в 0 (например $h_2=1$) и из 0 в 1 (например $h_3=1$).

Найденные значения сигналов возбуждения (слово 110) записываются в соответствующие клетки таблицы возбуждения памяти.

4-й этап

Осуществляется комбинационный синтез автомата, т.е. логический синтез его комбинационной схемы. В принципе его можно выполнять, пользуясь двумя таблицами, описывающими работу:

- таблицей выходов ЦА;
- таблицей возбуждения памяти.

Эти две таблицы сводят для удобства в единую таблицу истинности, которая имеет $e + n$ входов, $w + p$ выходов.

В любом случае требуется для одного такта автоматного времени составить схему из $w + p$ булевых функций от $e+n$ двоичных переменных:

$$\left. \begin{aligned} Z_i(t) &= \varphi_1[v_1(t), \dots, v_l(t); q_1(t), q_2(t), \dots, q_n(t)], \\ h_j(t) &= \varphi_2[v_1(t), \dots, v_l(t); q_1(t), q_2(t), \dots, q_n(t)], \end{aligned} \right\} \quad (3.29)$$

где $i = 1, 2, \dots, w$;

$j = 1, 2, \dots, p$;

$q_n(t)$ – выходной сигнал n -го элемента памяти.

При практической записи и минимизации функций системы (3.29) идентификатор i у всех символов можно опустить в силу его всеобщности и подразумевать его наличие условно.

Пример 3.4

Синтезировать схему двоичного счетчика накапливающего (суммирующего) типа, имеющего 8 внутренних состояний, в базисе И, ИЛИ, НЕ, Т-триггер.

Решение

1-й этап

Формулируем словесное описание синтезируемого ЦА (счетчика).

Отметим, что структурный алфавит задан не только структурным базисом, но и типом двоичного счетчика. Следовательно, алфавит – двоичный, T -триггер в наибольшей степени подходит для синтеза именно счетчиков, т.к. он имеет счетный вход (хотя можно использовать и другие типы триггеров).

Переходы в T -триггере описываются табл. 3.18.

Под суммирующим счетчиком понимают такое устройство, с выходов которого можно считать число, равное количеству импульсов, поступивших на его вход, после того как оно было приведено в нулевое состояние.

В общем случае сигналы счета ($Cч$) могут поступать на вход через любые интервалы времени. Таким образом, счетчик является автоматом Мура с одним входом и для его описания достаточно составить только одну (отмеченную) таблицу переходов.

Таблицу в этом случае удобнее строить не в классическом виде, а в виде таблицы истинности (табл. 3.17).

Для этого нужно найти число элементов памяти в автомате по формуле

$$n > \log_b l = \log_2 8 = 3,$$

где l – количество внутренних состояний счетчика.

Таким образом, в таблице переходов (истинности) ЦА будут фигурировать 4 аргумента (табл. 3.19):

1-й аргумент – выходной сигнал 3-го триггера (состояние самого старшего разряда q_3^* в предыдущем такте (т.е. в такте $t-1$);

2-й аргумент – выходной сигнал второго триггера q_2^* ;

3-й аргумент – выходной сигнал триггера самого младшего q_1^* ;

4-й аргумент – входной сигнал автомата ($Cч$).

Таблица 3.19

Таблица переходов двоичного 3-разрядного суммирующего счетчика

Номер такта		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Состояние T_3 в такте $t-1$ – 1-й аргумент	q_3^*	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Состояние T_2 в такте $t-1$ – 2-й аргумент	q_2^*	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Состояние T_1 в такте $t-1$ – 3-й аргумент	q_1^*	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Входной сигнал в такте $t-1$ – 4-й аргумент	V	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Окончание табл. 3.19

Номер такта		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1-я функция – состояние T_3 в такте t	q_3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0
2-я функция – состояние T_2 в такте t	q_2	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
3-я функция – состояние T_1 в такте t	q_1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

Нетрудно видеть, что двоичное число $Q^* = q_1^* q_2^* q_3^*$ есть не что иное, как содержимое (внутреннее состояние) счетчика в такте $t-1$.

Количество функций в табл. 3.19 равно 3.

q_3 , q_2 и q_1 – состояния Т-триггеров в такте t .

Следовательно, двоичное число $Q = q_3 q_2 q_1$ есть содержимое счетчика в следующем такте, причем

$Q = Q^*$, если $V = 0$,

$Q = Q^* + 1$, если $V = 1$.

В случае, если в счетчике в такте $t-1$ было записано максимальное число (111), то при очередном $V = 1$ счетчик обнуляется.

Кроме таблицы переходов, на первом этапе составляется упрощенная схема в виде двухкомпонентного структурного автомата (рис. 3.23).

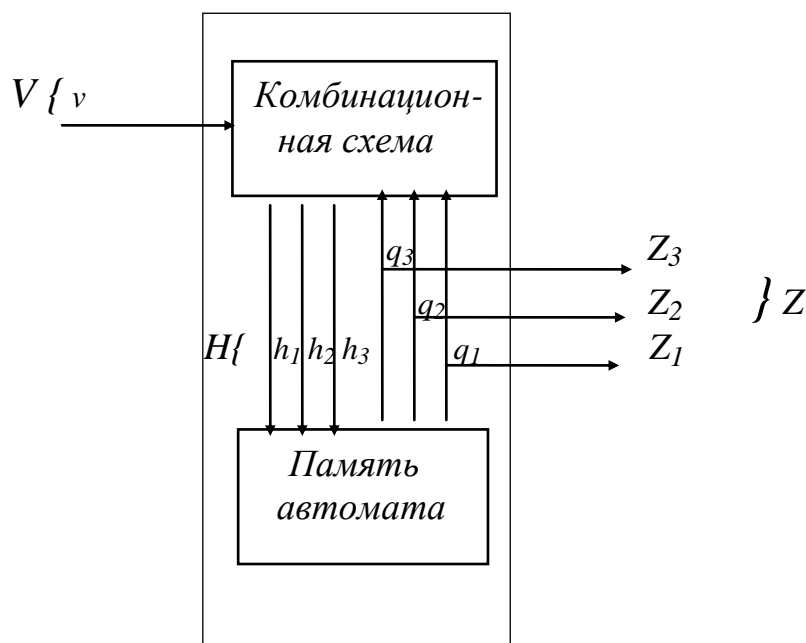


Рис. 3.23

2-й этап

Учитывая, что количество входов у выбранного элемента памяти равно 1, по формуле $p = e_{\text{эн}} n$ находим количество выходных связей, идущих на память:

$$p = e_{\text{эн}}, n = 3.$$

Количество входных связей, идущих от памяти, равно n (в данном случае 3). Следовательно, комбинационная схема имеет общее количество входов:

$e_{\text{общ}} = e + n = 1 + 3 = 4$, а общее количество выходов $W_{\text{общ}} = w + p = 3$, так как счетчик является автоматом Мура ($w = 0$).

3-й этап

Строим таблицу возбуждения памяти ЦА (табл. 3.20). Аргументы, а также форма таблицы будут такими же, как у табл. 3.19 (переходов). Заполнять ее будем, пользуясь данными табл. 3.19 и табл. 3.17 (переходов Т-триггера).

Для примера рассмотрим 3-й столбец таблицы переходов.

Номер столбца, как обычно, определяется двоичным числом, состоящим из значений истинности аргументов, записанных по вертикали сверху вниз.

Таблица 3.20

Таблица возбуждения памяти

1-й аргумент (t-1)	q_3^*	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
2-й аргумент (t-1)	q_2^*	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
3-й аргумент (t-1)	q_1^*	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
4-й аргумент (t-1)	V	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Функция воз- буждения T_3	h_3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
Функция воз- буждения T_2	h_2	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
Функция воз- буждения T_1	h_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Заметим, что триггер T_3 перешел из состояния $q_3^* = 0$ в состояние $q_3 = 0$.

Из табл. 3.17 следует, что Т-триггер из состояния $q^* = 0$ в состояние $q = 0$ переводится сигналом возбуждения $T = 0$. Следовательно, на T_3 надо подать сигнал возбуждения $h_3 = 0$. Записываем 0 в 3-м столбце таблицы возбуждения в строке с h_3 .

Далее анализируем поведение T_2 . Замечаем, что он перешел из состояния $q_2^* = 0$ в состояние $q_2 = 1$. В соответствии с табл. 3.17 для такого перехода на T_2 требуется подать сигнал $h_2 = 1$. Записываем 1 в 3-м столбце таблицы возбуждения в строке с h_2 .

И, наконец, анализируем T_1 . Этот триггер совершает переход из $q_1^* = 1$ в $q_1 = 0$. Табл. 3.17 показывает, что для такого перехода нужно на T_1 подать сигнал $h_1 = 1$. Записываем 1 в 3-м столбце таблицы возбуждения в строке h_1 .

Следуя указанной методике, аналогично проставляют значения истинности функций возбуждения h_3, h_2, h_1 во всех 16-ти столбцах табл. 3.20.

Чтобы облегчить эту процедуру, заметим (см. табл. 3.17), что для перевода T -триггера в противоположное состояние ($0 \rightarrow 1$ или $1 \rightarrow 0$) требуется сигнал возбуждения $T=1$. Если триггер не меняет своего состояния (переходы вида $0 \rightarrow 0$ или $1 \rightarrow 1$), то требуется сигнал $T=0$.

Далее, воспользовавшись тем, что при входном сигнале $V = 0$ слова $Q^* = q_3^* q_2^* q_1^*$ и $Q = q_3 q_2 q_1$ равны между собой, во всех столбцах таблицы возбуждения памяти (табл. 3.20), имеющих четные номера (в том числе и в нулевом столбце), во всех строках проставляем нули ($H = h_3 h_2 h_1 = 000$).

Остальные столбцы заполняем по стандартной методике.

4-й этап

В связи с тем, что синтезируемый счетчик является автоматом Мура, его комбинационная схема описывается только одной таблицей возбуждения памяти. Следовательно, табл. 3.20 является таблицей истинности синтезируемой комбинационной схемы автомата.

Основываясь на данных табл. 3.20, проводим далее комбинационный синтез автомата.

В данном случае удобнее всего построить для каждой из функций $h_3(q_3^*, q_2^*, q_1^*, V)$, $h_2(q_3^*, q_2^*, q_1^*, V)$, $h_1(q_3^*, q_2^*, q_1^*, V)$ диаграмму Вейча–Карно. Эти диаграммы приведены на рис. 3.24

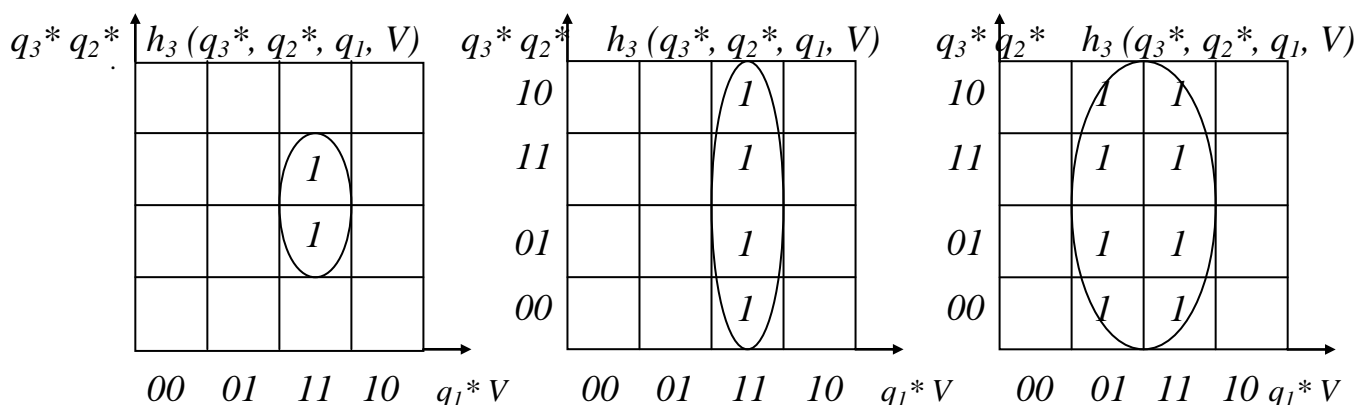


Рис. 3.24

Диаграмма Вейча–Карно позволяет представить искомые логические функции в ТДНФ:

$$\left. \begin{aligned} h_3 &= q_2^* q_1^* V; \\ h_2 &= q_1^* V; \\ h_1 &= V. \end{aligned} \right\} \quad (3.30)$$

Функциональная схема суммирующего счетчика, комбинационная схема которого построена в соответствии с системой выражений (3.30), а память состоит из 3-х триггеров, приведена на рис. 3.25.

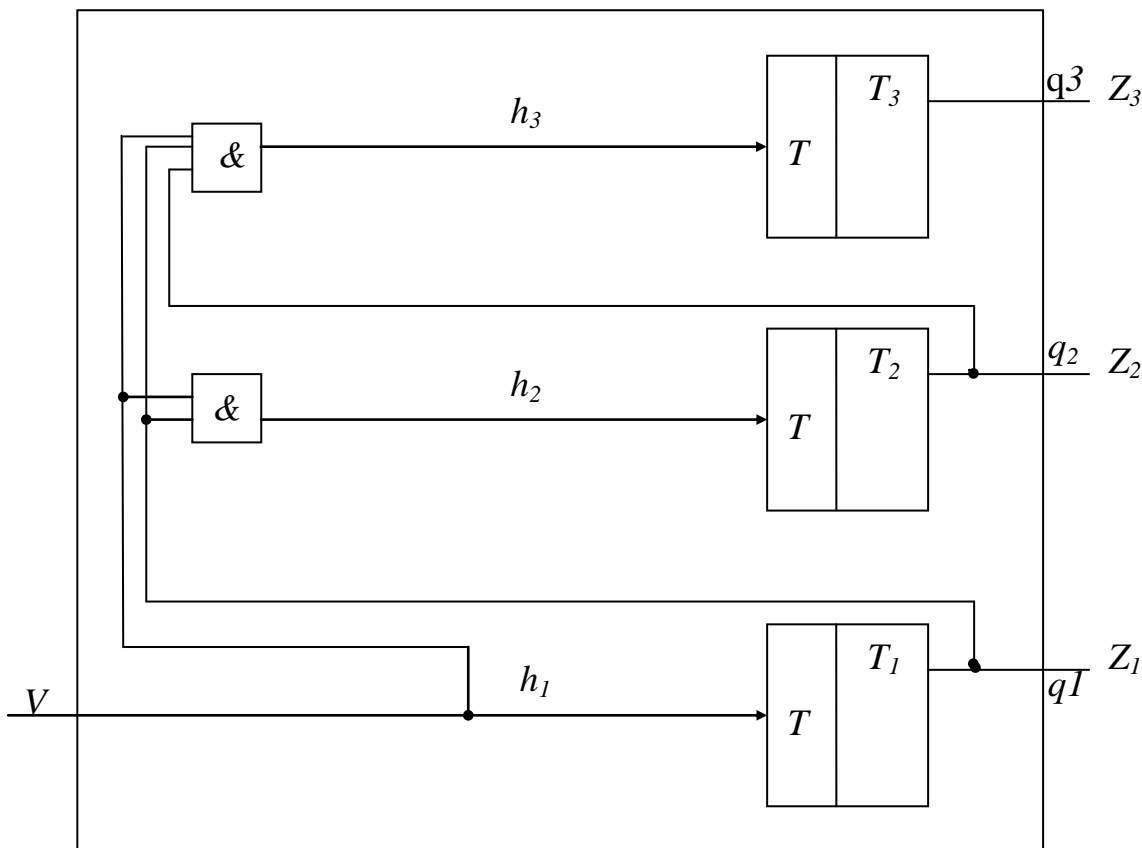


Рис. 3.25

Это счетчик параллельного действия. Сигнал счета V поступает одновременно на все триггеры. Время срабатывания счетчика

$$\tau_{сч (пар)} = \tau_{н.н.к} + \tau_{н.н.т},$$

где $\tau_{н.н.к}$ – время переходных процессов в конъюнкторе (&);

$\tau_{н.н.т}$ – время переходных процессов в триггере (T).

Для реализации такой комбинационной схемы требуется $N_{сч (пар)} = \lceil (n2 + n - 2) \rceil / 2$ условных транзисторов.

Количество оборудования можно уменьшить, если перейти к схеме

последовательного действия, т.е. за счет ухудшения быстродействия. Для этого преобразуем систему выражений (3.30) следующим образом:

$$\left. \begin{aligned} h_3 &= q_2^* h_2; \\ h_2 &= q_1^* h_1; \\ h_1 &= V. \end{aligned} \right\} \quad (3.31)$$

Схема счетчика последовательного действия приведена на рис. 3.26.

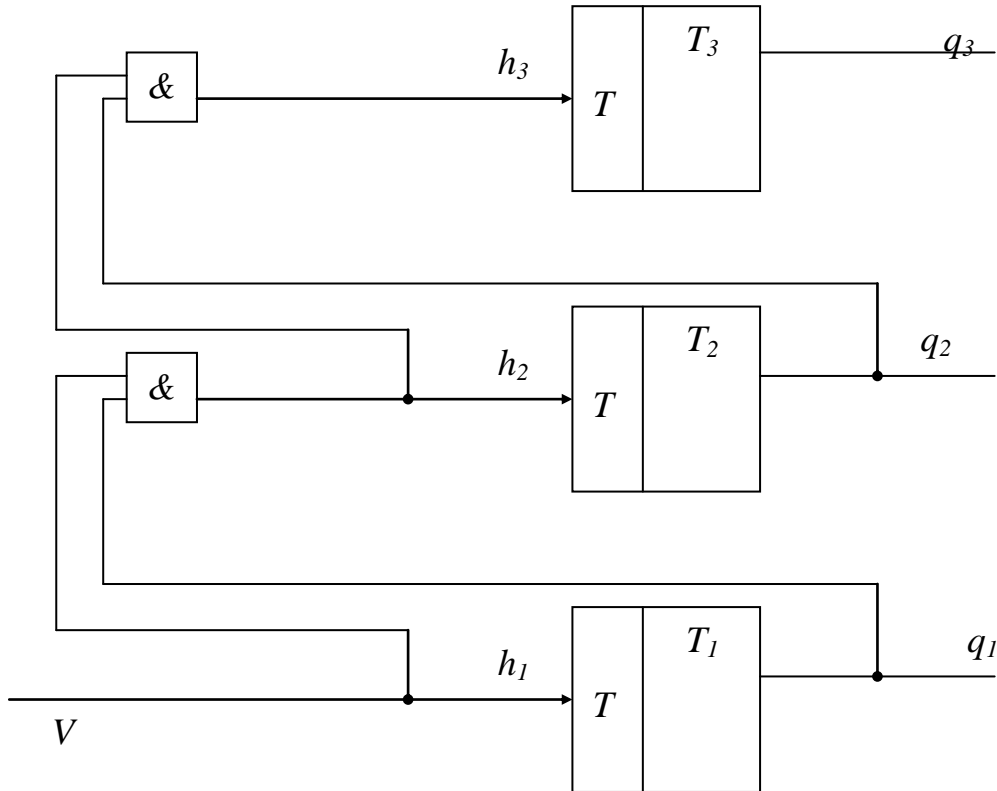


Рис. 3.26

Максимальное время срабатывания этого счетчика будет больше:

$$T_{сч(max)} = (n-1) (\tau_{н.н.к} + \tau_{н.н.м}),$$

но зато оборудование, необходимое для реализации счетчика, будет меньше:

$$N_{сч(посл)} = 2 (n-1) \text{ условных транзисторов.}$$

Отметим одну важную закономерность в построении разрядов. Используя метод математической индукции, логическое выражение для функции возбуждения любого i -го разряда счетчика ($i = 1, 2, \dots, n, n > 3$):

$$h_i = q_{i-1}^* q_{i-2}^* \dots q_2^* q_1^* V; \quad (3.32)$$

$$h_i = q_{i-1}^* h_{i-1}. \quad (3.33)$$

Эти выражения позволяют строить любые суммирующие счетчики при $n > 0$.

Отметим, что аналогичным образом можно синтезировать и вычитающий счетчик.

4. ПРИНЦИПЫ ПОСТРОЕНИЯ И ФУНКЦИОНИРОВАНИЯ ЭВМ

4.1. Общие понятия о принципах организации ЭВМ

Приведем некоторые определения [7,17,18].

Функция – правило получения результатов, определяемых назначением системы. Другими словами, это описание процессов, которые имеют место в системе.

Формы описания функций – словесная, формулы, алгоритмы (чаще всего).

Структура – совокупность элементов и связей между ними.

Математической формой изображения структуры является *граф*.

Схема – это тоже граф (инженерная форма).

Системам присуще следующее качество: свойство совокупности элементов, объединенных в одну систему, не является суммой свойств элементов, а имеет новое качество, отсутствующее в отдельных элементах.

Принцип, по которому объединение элементов приводит к новому качеству, называется *принципом организации*.

Функция и структура – это конкретизация принципа организации.

Когда говорят о принципе порождения функций, пользуются термином «*функциональная организация*». *Функциональная организация – это принципы построения абстрактных структур*.

Когда речь идет о порождении структур, необходимых для выполнения заданных функций, пользуются термином «*структурная организация*». *Структурная организация – это принципы перевода абстрактных систем, заданных в виде функций, в материальные системы, состоящие из физически существующих элементов*.

Рассмотрим основные факторы, влияющие на принципы построения ЭВМ.

Напомним, что *ЭВМ – это искусственная система, предназначенная для выполнения вычислений на основе алгоритмов*. Следовательно, свойства алгоритмов предопределяют принципы построения, т.е. организацию ЭВМ.

К важнейшим свойствам алгоритмов, определяющих организацию ЭВМ, относятся:

- 1) *дискретность информации*, с которой оперируют алгоритмы;
- 2) *конечность и элементарность* набора операций, выполняемых при реализации алгоритмов;

3) *детерминированность* вычислительных процессов (ВП), порождаемых алгоритмами.

Определяя назначение ЭВМ, указывают не только класс алгоритмов (задач), на выполнение которых ориентирована ЭВМ, но и требования к основным параметрам: производительности, надежности и т.д.

4.2. Принцип программного управления ЭВМ

Современные ЭВМ строят по одному принципу – *принципу программного управления*.

В основе этого принципа лежит представление алгоритма в форме операторной схемы, которая задает правило вычислений как композицию операторов (операторов преобразования информации и операторов анализа для определения порядка выполнения операторов).

Реализация этого принципа может быть различной. Один из способов был предложен в 1945 г. Дж. фон Нейманом. Этот способ заключается в следующем:

- информация кодируется в двоичной форме и разделяется на единицы (элементы) информации, называемые *словами*;
- разнотипные слова информации различаются по способу использования, но не способами кодирования;
- слова информации размещаются в ячейках памяти ЭВМ и идентифицируются номерами ячеек, называемых *адресами слов*;
- алгоритм представляется в форме последовательности управляющих слов, которые определяют наименование операций и слова информации, участвующие в операции, и называются *командами*.

Алгоритм, представленный в терминах команд, называется *программой*.

Выполнение вычислений, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определенном программой.

4.3. Состав и порядок функционирования ЭВМ

Определим номенклатуру устройств, из которых должна состоять традиционная ЭВМ [7].

Номенклатура устройств вытекает из фон-неймановского принципа программного управления и свойств технических средств, обеспечивающих

хранение, обработку, ввод и вывод информации. В соответствии с этим в состав традиционных ЭВМ входят (рис. 4.1): обрабатывающая подсистема (или центральный процессор), подсистема памяти, подсистема ввода-вывода и сервисная подсистема (пульт управления).



Рис. 4.1

Из рис. 4.1 видно, как разделены функции между устройствами ЭВМ.

Функционирование ЭВМ производится в следующей последовательности:

- ввод программы и данных в память;
- запуск программы;
- выполнение программы;
- вывод результатов.

4.4. Функциональная организация ЭВМ

Как уже отмечалось, ЭВМ можно рассматривать в двух аспектах: функциональном и структурном.

С функциональной точки зрения ЭВМ – это система абстрактных элементов, в терминах которых представляется информация, относящаяся к данным и алгоритмам, и процессы вычислений, порождаемые алгоритмами.

К числу таких элементов относятся:

- наборы символов и машинные элементы информации;
- машинные операции и команды;
- адреса и способы адресации;
- управляющие слова (слово состояния программы (ССП), адресное слово канала (АСК), командное слово канала (КСК), слово состояния канала (ССК), управляющее слово устройства (УСУ) и др.).

Структура ЭВМ – это состав элементов и конфигурация связей между ними, определяемая функциями ЭВМ.

Структуры могут различаться при одинаковых функциях. Так, например, структура ЭВМ, приведенная на рис. 4.1, и структура персональной ЭВМ, имеющей шинную структуру, имеют одинаковую функциональную организацию. Различные структуры определяют различные временные характеристики и затраты оборудования. Следовательно, функции являются первичными по отношению к структуре, и поэтому проектирование ЭВМ должно начинаться с конкретизации функций, т.е. с принципа программного управления в плане способов представления информации, способов адресации, типов команд и т.д.

Для обозначения круга вопросов, относящихся к функциональной организации ЭВМ, часто применяют термин *архитектура*. Есть несколько разных определений архитектуры:

Архитектура вычислительной системы – общая логическая организация вычислительной системы (ВС), определяющая процесс обработки данных в конкретной ВС и включающая методы кодирования данных, состав, назначение, принципы взаимодействия технических средств (ТС) и программного обеспечения (ПО).

Фон-неймановская архитектура – организация ЭВМ, при которой ЭВМ состоит из двух основных частей – *линейно-адресуемой памяти*, слова которой хранят команды и элементы данных, и *процессора*, выполняющего эти команды.

Архитектура ЭВМ – совокупность основных устройств, узлов и блоков ЭВМ, а также структура основных управляющих и информационных связей между ними, обеспечивающая выполнение заданных функций.

В [6] приведено такое определение: *архитектура – это распределение функций, реализуемых системой, по отдельным уровням и точное определение границ между уровнями.*

Рассмотрим кратко абстрактные элементы, определяющие функциональную организацию ЭВМ.

1. *Машинные элементы информации включают:*

- двоичные коды;
- байты, слова;
- представление чисел в форме с фиксированной точкой, плавающей точкой, двоично-десятичной форме, символьном виде.

2. *Машинная операция – это действие, инициируемое одной командой и реализуемое оборудованием ЭВМ.*

С набором операций связано понятие «системы команд». В систему команд ЭВМ обычно входят следующие типы команд:

- арифметико-логические;
- посылочные (пересылочные);
- переходы (условные и безусловные);
- ввода-вывода;
- системные (управление режимами работы, системой времени и др.).

3. *В ЭВМ применяются следующие способы адресации памяти [3, 7]:*

- непосредственная (константа в команде);
- прямая адресация;
- косвенная (базирование и индексирование);
- неявная адресация.

Особенности двух первых способов адресации понятны из их названия и дополнительного пояснения не требуют.

Косвенная адресация базируется на *страничной организации памяти.*

Множество 2^m ячеек оперативной памяти разделяются на группы, состоящие из $S = 2^p$ соседних ячеек и называемые *страницами*, при этом $p, m, m/p$ – целые числа и $p \leq m$.

Количество ячеек в странице называют *размером* страницы.

Для уменьшения длины команд адреса в команде представляются не полностью, а только младшими p -разрядами (т.е. смещением). Для определения

полного адреса, указываемого в команде, используются относительная адресация и страничная адресация.

1. *Относительная адресация* при страничной организации памяти выполняется путем сложения m -разрядного базового адреса, указанного в поле B относительного адреса, с p -разрядным смещением (рис. 4.2).

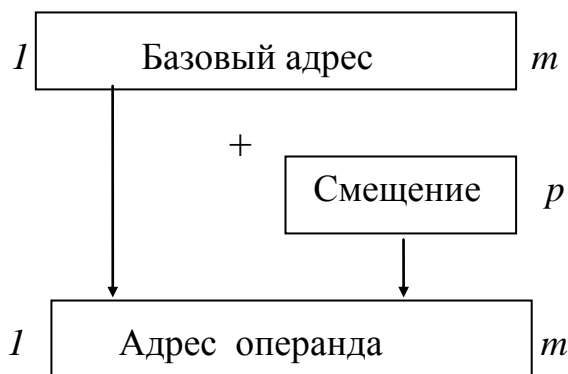


Рис. 4.2

При этом обращение возможно только к ячейкам $l-p$ в группе базового адреса.

2. При *страничной* адресации 2^m ячеек основной памяти разделяются на страницы с фиксированными базовыми адресами:

ячейки с адресами $0, 1, \dots, S-1$ – страница 0;

$S, S+1, \dots, 2S-1$ – страница 1 и т.д.

$m-S, m-S+1, \dots, m-1$ – страница $\frac{m}{S} - 1$.

Номера $0, 1, \dots, \frac{m}{S} - 1$ называются *номераами страниц*.

Поскольку длина страницы равна степени двойки ($S = 2^p, p = 1, 2, \dots, m$), m -разрядный адрес памяти, имеющей емкость 2^m слов, можно рассматривать как состоящий из двух полей (рис. 4.3): P – q -разрядный адрес страницы, равный $0, 1, \dots, 2^q - 1$, D – p -разрядный адрес ячейки в странице, равный $0, 1, \dots, 2^p - 1$.

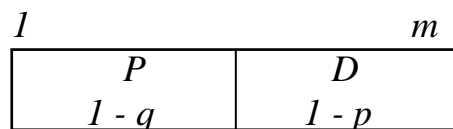


Рис. 4.3

В командах указываются p -разрядные адреса слов (ячеек) в странице.

Полный адрес формируется приписыванием p -разрядного смещения D , указанного в команде, к q -разрядному адресу страницы, определенному к моменту выполнения команды (программы).

Размер страницы может изменяться от 2^1 до 2^m .

В типовых форматах команд ЭВМ задаются *код операции* (КОП) и *адреса операндов*. В ЭВМ, имеющей внутреннюю регистровую память (регистры общего назначения (РОН) и регистры для выполнения операций с плавающей точкой (РПТ)) (рис. 4.4), в составе системы команд используются операнды, расположенные в регистровой памяти (R) и/или в основной памяти (S), т.е. команды разных форматов $R \rightarrow R$, $R \rightarrow S$, $S \rightarrow R$, $S \rightarrow S$.

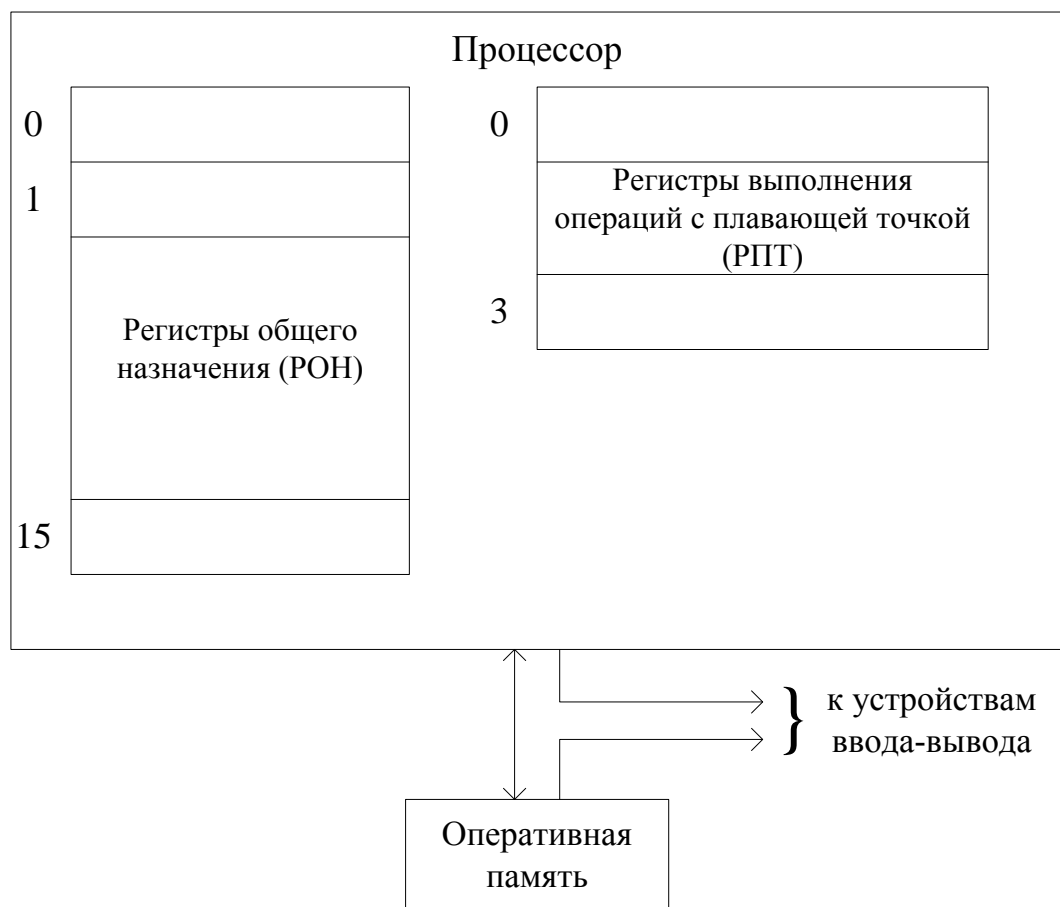


Рис. 4.4

4.5. Режимы работы ЭВМ

ЭВМ могут работать в следующих режимах:

- однопрограммном;
- режиме пакетной обработки задач;
- режиме работы в реальном масштабе времени;
- мультипрограммном.

Особенности режимов работы очевидны из их названий и не требуют дополнительных пояснений. Рассмотрим только наиболее сложный из режимов – мультипрограммный.

Требования, предъявляемые к ЭВМ для обеспечения режима мультипрограммной работы.

Для обеспечения режима мультипрограммной работы в ЭВМ должны быть выполнены следующие требования:

- наличие средств, управляющих порядком выполнения задач;
- емкость памяти, достаточная для размещения информации, относящейся к нескольким задачам;
- защита памяти;
- система прерывания программ (для сообщения об особых ситуациях, возникающих при выполнении программ);
- возможность параллельной работы процессора и системы ввода-вывода.

Для управления порядком выполнения задач обычно используются программные средства. Организация памяти в мультипрограммных ЭВМ рассмотрена в подразд. 3.7. Ниже рассмотрены примеры реализации остальных требований, предъявляемых к мультипрограммным ЭВМ.

Защита памяти

Средства защиты памяти (СЗП) обеспечивают проверку адреса при каждом обращении к памяти. Адрес считается корректным, если он принадлежит области ОП, которая выделена программе, выполняемой процессором или периферийным устройством (ПФУ), и, следовательно, некорректен, если не принадлежит. В этом случае выполнение программы блокируется, и программа прерывается по причине нарушения защиты.

Контроль по защите памяти может происходить в любом режиме работы оперативной памяти – чтении или записи.

Применяются различные способы защиты адреса оперативной памяти:

- 1) *по граничным адресам* – этот способ неудобен, так как требует выделения сплошного участка оперативной памяти;
- 2) *по ключам* (рис. 4.5).

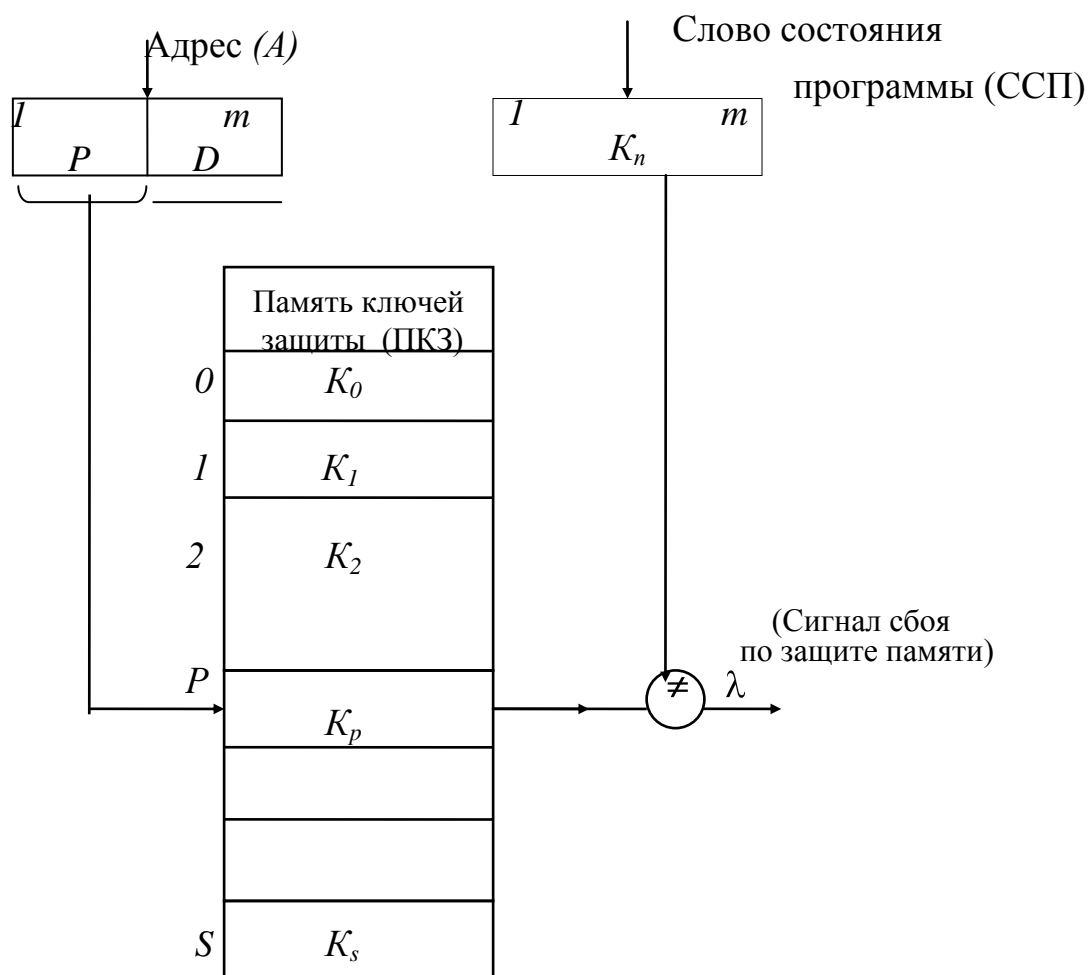


Рис. 4.5

Память разбивается на страницы (блоки) объемом 2–4 Кбайт. Каждой странице присваивается свой ключ, разрядностью, кратной 2 (например 4).

Ключи (K_i) хранятся в *памяти ключей защиты* (ПКЗ):

K_0 – в нулевой ячейке;

K_1 – в первой ячейке и т.д.

Ключи в ПКЗ записывает супервизор (управляющая программа) по привилегированной команде:

«Установить K_n «A», «P»,

где A – адрес ячейки оперативной памяти, где хранится ключ;

P – адрес страницы, которой присвоен ключ.

В результате выполнения этой команды странице P , закрепленной за программой с ключом K_n , присваивается ключ K_n , который становится ключом страницы.

Проверка адресов выполняется следующим образом:

1) ключ K_n из слова состояния программы (ССП) загружается в центральный процессор на время выполнения программы;

2) в адресе A , сформированном программой, выделяется номер страницы P , определяющий адрес страницы оперативной памяти, к которой производится обращение;

3) по адресу P выбирается из ячейки памяти ключей защиты (ПКЗ) ключ K_p и сравнивается с K_n , указанным в ССП.

Если $K_p \neq K_n$, формируются сигнал сбоя по защите и запрос на прерывание (λ).

При работе центрального процессора в режиме супервизора при $K_n = 0$ сигнал " \neq " не формируется.

Организация прерывания

Во время выполнения ЭВМ текущей программы внутри машины или в связанной с ней внешней среде (например во внешнем устройстве) могут возникнуть события, требующие немедленных реакций на них со стороны ЭВМ. Эти реакции состоят в том, что при возникновении подобных событий ЭВМ должна прервать отработку текущей программы и перейти к выполнению другой программы, специально предназначенной для ситуации, связанной с появлением данного события. По завершении этой программы ЭВМ должна вернуться к выполнению прерванной программы.

Если имеется несколько источников прерывания, вырабатывающих свои запросы независимо, должен быть установлен определенный порядок обслуживания запросов. Существуют системы прерывания с последовательным просмотром (*сканированием*) наличия запросов у источников прерывания и с обслуживанием запросов в порядке присвоенного им приоритета.

Системы прерывания выполняют следующие функции:

- 1) запоминание состояния прерываемой программы;
- 2) приоритетный выбор запроса для исполнения из поступивших запросов прерывания и организация перехода к прерывающей программе;
- 3) восстановление состояния прерванной программы и возврат к ней;

4) программное изменение приоритетов программ (выполняемое при помощи механизма маскирования запросов).

В традиционных ЭВМ применялись различные средства для организации прерывания. Наиболее распространенным способом организации прерывания является применение *слов состояния программы (ССП)*, под управлением которых осуществляется выполнение в ЭВМ программ.

Пример структуры слова состояния программы приведен на рис.4.6.

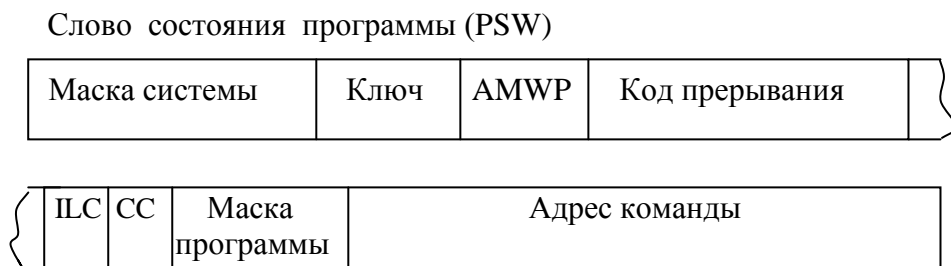


Рис. 4.6

В поле «Маска системы» указываются маски каналов и маска внешних прерываний.

В поле «Ключ» указывается ключ защиты.

В поле AMWP указывается:

A – режим *ASCII*;

M – маска контроля машины;

W – состояние ожидания;

P – режим решения основной задачи – в отличие от режима супервизора.

В поле «Код прерывания» указывается код обрабатываемого вида прерывания:

ILC – код длины команды;

CC – код-условие (или признак результата).

В поле «Маска программы» указываются маски:

- переполнения в операциях с фиксированной точкой;
- переполнения в операциях с десятичными числами;
- исчезновения порядка;
- потери значимости.

В поле «Адрес команды» указывается адрес следующей команды.

В рассматриваемой в данном примере ЭВМ было пять источников (причин) прерывания, в частности, прерывания от схем контроля ЭВМ, программные прерывания, от обращения к супервизору, прерывания от внешних источников и от ввода-вывода. Система прерывания данной ЭВМ – приоритетная, и запрос на

прерывание, имеющий более высокий приоритет, может прервать выполнение программы обработки запроса на прерывание менее приоритетное.

Для каждого вида прерывания в памяти ЭВМ имеются ячейки для хранения слова состояния прерванных программ (старые ССП) и хранятся ССП для программ обработки соответствующих запросов на прерывание (новые ССП).

Организация прерывания с использованием ССП приведена на рис. 4.7.

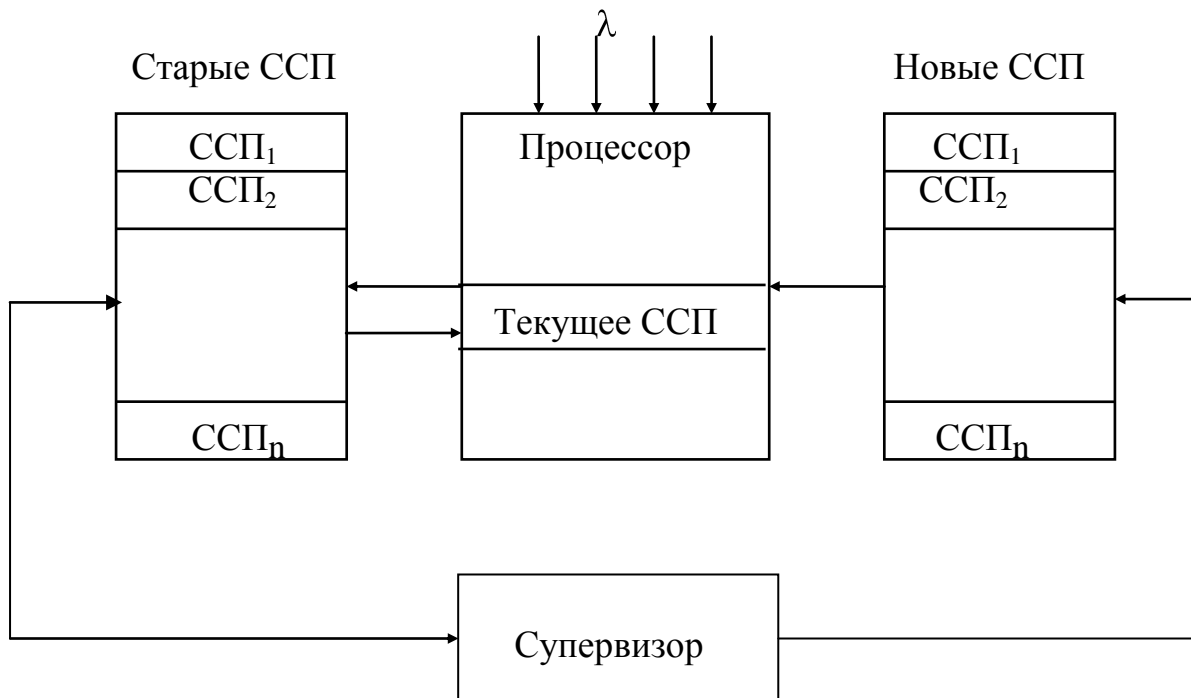


Рис. 4.7

Организация ввода-вывода

Для обеспечения параллельной работы процессора и системы ввода-вывода, начиная с ЭВМ 3-го поколения, был введен ряд специальных управляющих слов для обеспечения работы системы ввода-вывода. В структуре системы ввода-вывода ЭВМ можно выделить следующие уровни (устройства):

- каналы ввода-вывода;
- устройства управления внешними (периферийными) устройствами;
- внешние устройства (механизмы).

Пример структуры слов управления вводом-выводом (АСК, КСК, УСУ и ССК, приведенных в подразд. 4.4) и их взаимосвязь показана на рис. 4.8.

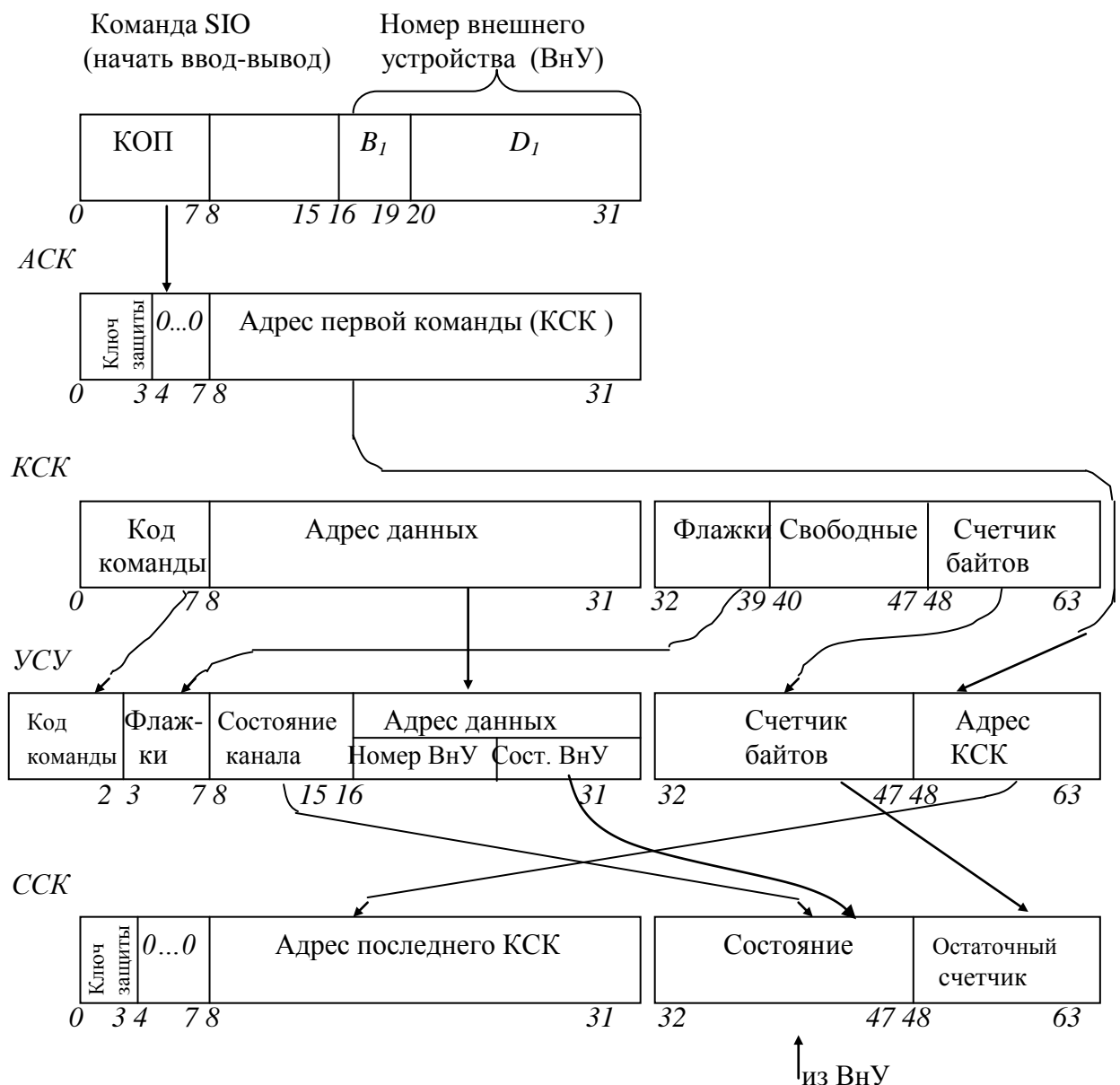


Рис. 4.8

Для управления устройствами ввода-вывода в традиционных ЭВМ используется несколько типов команд, таких, как SIO (начать ввод-вывод), PIO (проверить ввод-вывод), TCH (проверить канал), HIO (остановить ввод-вывод), CLRIO (очистить ввод-вывод), HDV (остановить устройство) и другие.

Как уже отмечалось выше, для организации в ЭВМ параллельной работы устройств ввода-вывода и процессора используется ряд специальных управляющих слов.

Назначение отдельных полей этих управляющих слов видно из рис. 4.8. Требуется немного добавить о назначении и составе полей «Флажки» и «Состояние канала».

В поле «Флажки» указываются специальные признаки, которые позволяют организовывать без прерывания работы процессора последовательности команд обращения к устройствам ввода-вывода (канальных программ) с разными кодами команд или одинаковыми кодами, но с разными адресами памяти, информировать процессор о ходе выполнения канальных программ и организовывать другие режимы.

Процессор участвует в организации ввода-вывода до момента формирования УСУ, далее процессор переходит к выборке и выполнению следующей команды своей программы, а выполнение программы осуществляется параллельно с работой процессора под управлением УСУ. Отметим, что может быть сформировано столько УСУ, сколько операций ввода-вывода необходимо одновременно выполнять.

В процессе выполнения передачи информации между ВнУ и памятью ЭВМ содержимое счетчика байтов уменьшается на величину переданных данных, а после успешного завершения текущей команды обеспечивается выборка следующего КСК канальной программы, если в поле «Флажки» установлен соответствующий признак.

После завершения операции ввода-вывода успешного или вынужденного (например из-за сбоя в устройстве ввода-вывода) канал формирует специальное управляющее слово – ССК, структура которого показана на рис. 4.8. Назначение полей ССК ясно из их названия. Отметим только, что в поле «Состояние» записывается информация о состоянии канала ввода-вывода и устройства ввода-вывода.

Информация, записанная в ССК, используется при выполнении прерывания по вводу-выводу и позволяет точно определить, как завершилась операция ввода-вывода.

4.6. Принципы построения обрабатывающих подсистем ЭВМ

В обрабатывающих подсистемах ЭВМ, как и в любых устройствах обработки цифровой информации, можно выделить *операционный блок (ОБ)* и

управляющий блок (УБ) [3, 7]. Структурная организация обрабатывающей подсистемы показана на рис. 4.9.

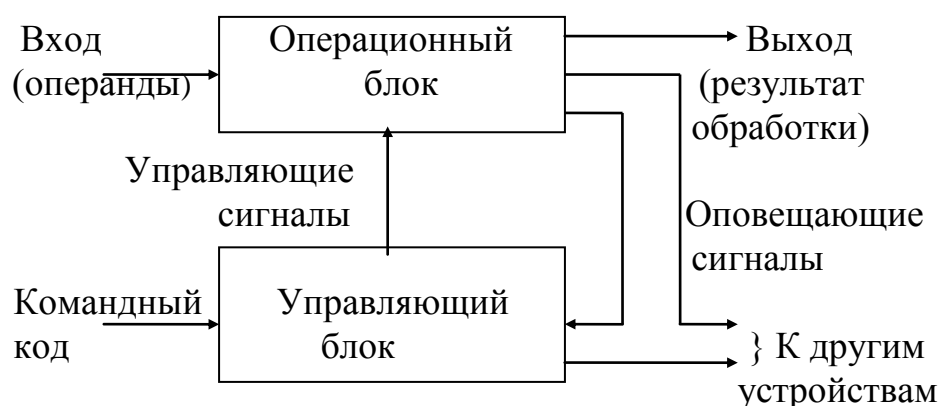


Рис. 4.9

Процесс функционирования во времени обрабатывающей подсистемы состоит из последовательности тактовых интервалов, в которых ОБ производит определенные элементарные операции преобразования слов (например, передача слов из регистра $Pz1$ в регистр $Pz2$, сдвиг и т.д.). Выполнение этих элементарных операций инициируется поступлением в ОБ управляющих сигналов из УБ.

Элементарная функциональная операция (или их комбинация), выполняемая за один тактовый интервал и приводимая в действие одним управляющим сигналом v_i , называется *микрооперацией*.

Может выполняться несколько элементарных микроопераций одновременно. Такая совокупность микроопераций называется *микрокомандой*.

УБ вырабатывает последовательность управляющих сигналов (УС) $v_{t1}, v_{t2}, \dots, v_{tk}$ ($v_{tj} \in V$), порождающих в ОБ требуемую последовательность микроопераций.

Последовательность управляющих сигналов определяется управляющим командным кодом, поступающим в УБ извне, и оповещающими сигналами, зависящими от операндов и промежуточных результатов преобразования информации.

ОБ задается его структурой, т.е. составом узлов и связями между ними, и выполняемым ОБ набором микроопераций.

Последовательность микрокоманд, обеспечивающих выполнение данной операции, называется *микропрограммой данной операции*.

Любая операция в операционном блоке описывается микропрограммой и реализуется за несколько тактов, в каждом из которых выполняется одна или несколько микроопераций. Для реализации команды, операции, процедуры или микропрограммы необходимо на соответствующие управляющие шины подать определенным образом распределенную во времени последовательность управляющих функциональных сигналов.

Рассмотрим пример реализации и функционирования операционного блока для выполнения операций сложения-вычитания чисел, представленных в форме с фиксированной точкой (рис. 4.10).

Напомним, что при операции вычитания

$$Z = X - Y = X + (-Y)$$

производится замена операции вычитания на сложение, и операция производится в дополнительном коде («инверсия» + 1).

Передачи в регистрах производятся по отдельным микрооперациям.

Приведем совмещенное описание операций сложения/вычитания на языке микропрограммирования [3]:

- 1) Пр РгВ: РгВ: = ШИ Вх <прием 1-го операнда>;
- 2) Пр Рг1: Рг1: = ШИ Вх <прием 2-го операнда>;
- 3) Прием: если "сложение", то Пр РгАП: РгА: = Рг1,
иначе Пр РгАИ : РгА = $\overline{\text{Рг1}}$;
- 4) Сумма: если "сложение", то Пр Рг См: Рг См: = РгА + РгВ,
иначе Пр Рг См: + 1 См: Рг См: = РгА + РгВ + 1;
- 5) Пр УБ: если признак результата ПР = 11, то "прерывание",
иначе Пр ШИ Вых: ШИ Вых: = Рг См <выдача результата>

Конец.

Для рассматриваемых операций сложения/вычитания код признака результата операции (ПР) формируется следующим образом (табл. 4.1):

Таблица 4.1

Значение признака результата

Результат	ПР	
0	0	0
< 0	0	1
> 0	1	0
Переполн.	1	1

ПР = 11, если

$$ПнСМ \cdot \overline{ПнСМ} \vee \overline{ПнСМ} \cdot ПнСМ;$$

ПР = 00, если

$$\bigwedge_{i=0}^n \overline{СМ_i} = 1;$$

ПР = 01, формируется, если

$$СМ \cdot \overline{ПнСМ} \cdot \overline{ПнСМ} \vee \overline{ПнСМ} \cdot \overline{ПнСМ};$$

ПР = 10, формируется, если

$$СМ \cdot \overline{ПнСМ} \cdot \overline{ПнСМ} \vee \overline{ПнСМ} \cdot \overline{ПнСМ}.$$

Кратко приведенная выше микропрограмма может быть записана следующим образом (табл. 4.2):

Таблица 4.2

Микропрограмма сложения/вычитания чисел

Такты	Сигналы
1	Пр Рг В
2	Пр Рг 1
3	Пр Рг АП или Пр Рг АИ
4	Пр Рг См или Пр Рг См + 1 См
5	Пр Ш и Вых

Напомним, что каждый управляющий функциональный сигнал поступает в начале некоторого такта на соответствующую ему шину ОБ, вызывая выполнение в ОБ определенной микрооперации (передача слов, суммирование кодов и т.д.).

Управляющие сигналы генерируются УБ. Последовательность этих сигналов задается поступающими на входы УБ:

- 1) кодом операции;
- 2) сигналами из ОБ, несущими информацию об особенностях операндов, промежуточных и конечных результатах;
- 3) синхросигналами, задающими границы тактов.

Формально УБ может рассматриваться как конечный автомат, определяемый:

- а) множеством $Z = \{Z_1, \dots, Z_m\}$ – выходных сигналов, соответствующих множеству микроопераций в обрабатывающем блоке; при $v_i = 1$ возбуждается i -я операция;
- б) множеством $v = \{v_1, \dots, v_n\}$ – входных сигналов, соответствующих задаваемому извне двоичному коду операции, и двоичным значением оповещающих сигналов;

в) множествами подлежащих реализации микропрограмм, устанавливающих в зависимости от входных сигналов управляющие сигналы, выдаваемые блоком в определенные такты.

По множествам входных (v) и выходных (Z) сигналов и микропрограмм определяется *множество внутренних состояний* УБ: $Q = \{Q_0, Q_1, \dots, Q_r\}$, «мощность» которого (объем памяти и оборудование УБ) в процессе проектирования стараются минимизировать.

Сказанное выше объясняет, почему УБ называют *управляющими автоматами*. Поскольку эти автоматы задаются микропрограммами, их часто называют *микропрограммными автоматами*.

УБ может быть задан как автомат *Мура*:

$$Q(t+1) = f[Q(t), v_1(t), v_2(t), \dots, v_n(t)];$$

$$Z_1(t) = \varphi_1[Q(t)];$$

$$Z_m(t) = \varphi_m[Q(t)]$$

или как автомат *Мили*:

$$Q(t+1) = f[Q(t)] = f[Q(t), v_1(t), \dots, v_n(t)];$$

$$Z_i(t) = \varphi_i[Q(t), v_1(t), \dots, v_n(t)],$$

где f – функция переходов,

φ – функция выходов.

Эти функции определяются заданной микропрограммой.

Существуют два основных метода построения логики УБ:

1. *УБ с жесткой или схемной логикой.*

Для каждой операции строится набор комбинационных схем и конечный автомат.

2. *УБ с хранимой в памяти блока логикой* (с запоминаемой или программируемой логикой).

Каждой выполняемой в обрабатывающем блоке операции ставится в соответствие совокупность хранимых в памяти слов – микрокоманд, содержащих информацию о микрооперациях, подлежащих выполнению в течение одного машинного такта, и указание (в общем случае зависящее от значений входных сигналов) – какая должна быть выбрана следующая микрокоманда. Таким образом, в этом случае функции переходов и выходов f и φ управляющего автомата хранятся в памяти в виде совокупности микрокоманд.

Последовательность микрокоманд, выполняющих одну машинную команду или процедуру, образует *микропрограмму*.

Микропрограммы хранятся обычно в отдельной памяти .

УБ, использующие такой принцип управления, называют *микропрограммными* (рис. 4.11).

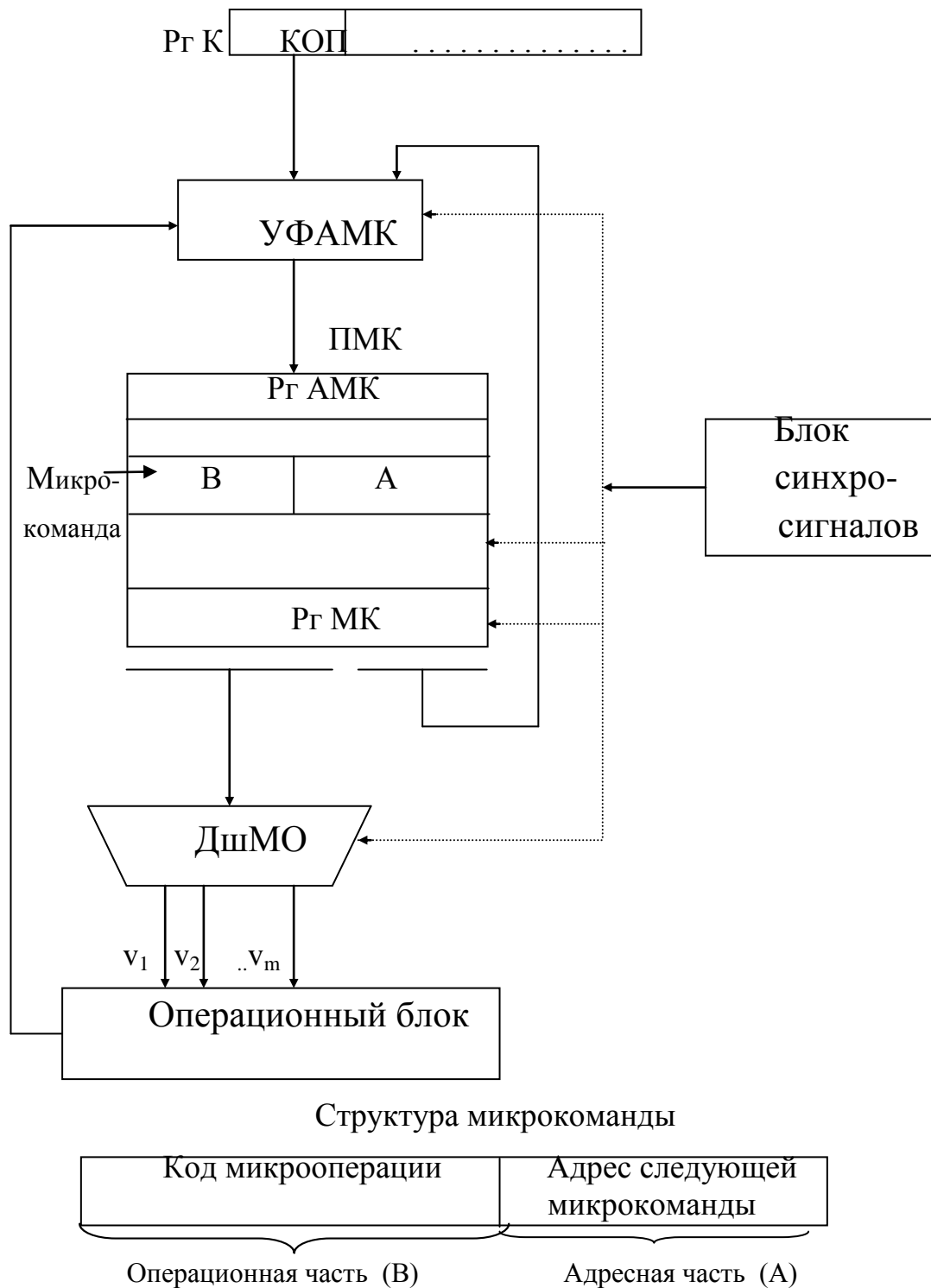


Рис. 4.11

На рисунке использованы следующие обозначения:

Рг К – регистр команды;

КОП – код операции;

УФАМК – узел формирования адреса микрокоманды;

Рг АМК – регистр адреса микрокоманды;

ПМК – память микрокоманд;

ДШМО – дешифратор микроопераций.

Применяются следующие типы структуры микрокоманд:

- вертикальная (одна микрооперация в микрокоманде);
- горизонтальная (все возможные микрооперации в микрокоманде);
- горизонтально-полевая, особенностью которой является разбиение

структуры микрокоманды на отдельные поля, каждое из которых используется для задания микроопераций управления отдельными блоками и устройствами ЭВМ (памятью, арифметико-логическим устройством и т.д.). При этом внутри полей может применяться вертикальное или горизонтальное кодирование [3, 7]. Для каждого из полей в составе УБ имеется отдельный дешифратор, что позволяет формировать сигналы управления различными устройствами одновременно.

Пример структуры УБ с горизонтально-полевой микрокомандой приведен на рис. 4.12.

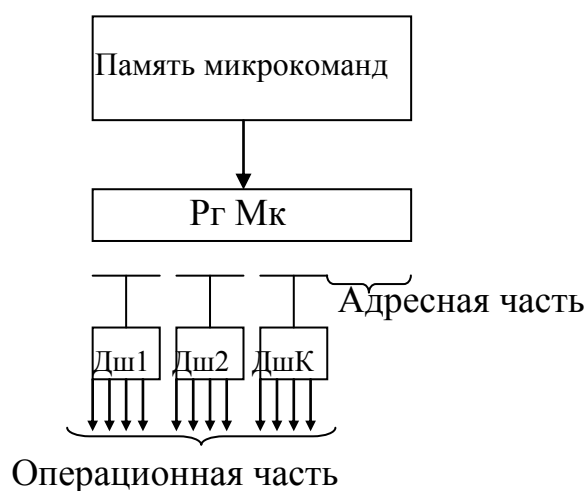


Рис. 4.12

Для хранения микропрограмм применяются различные виды устройств памяти: статические (ПЗУ), динамические (ОП) и статическо-динамические ЗУ.

4.7. Принципы построения устройств памяти

4.7.1. Общие сведения и классификация устройств памяти

Памятью называется совокупность устройств, предназначенных для запоминания и выдачи информации. Отдельные устройства, входящие в эту совокупность, называются *запоминающими устройствами (ЗУ)*, или *памятью того или иного типа* [3].

Эти термины – *ЗУ* и *память* – стали синонимами, теперь термин *ЗУ* чаще используют, когда хотят подчеркнуть принцип построения (*ЗУ на ферритах, ЗУ на магнитных дисках (МД) и т.д.*), а *память* – когда подчеркивают функцию устройства или место его расположения в ЭВМ (*основная память, внешняя память и т.д.*).

Производительность и вычислительные возможности ЭВМ в значительной степени определяются составом и характеристиками ее *ЗУ*.

В составе ЭВМ используется одновременно несколько типов *ЗУ*, отличающихся характеристиками и назначением.

Основные операции, выполняемые в памяти: запись (ввод), считывание (вывод). В зависимости от типа *ЗУ* за одно обращение может считываться или записываться разный объем информации: байт, слово или блок данных.

Важнейшими характеристиками отдельных устройств памяти являются: емкость памяти, удельная емкость и быстродействие.

Емкость – это максимальное количество данных, которое может храниться в памяти.

Емкость измеряется в кило-, мега-, гига-, терабайтах и т.д.

Удельная емкость – отношение емкости к объему памяти (физическому).

Быстродействие – продолжительность операции обращения, т.е. время, затраченное на поиск информации и ее воспроизведение ($t_{обр}^{счит}$ – при чтении и $t_{обр}^{зан}$ – при записи). В некоторых типах *ЗУ* считывание информации стирает ее и требуется регенерация (восстановление).

Продолжительность обращения ($t_{цикла\ памяти}$) при считывании и записи:

$$t_{обр}^{счит} = t_{дост}^{счит} + t_{счит} + t_{рег};$$

$$t_{обр}^{зан} = t_{досм}^{зан} + t_{подг} + t_{зан}.$$

В большинстве случаев

$$t_{досм}^{счит} = t_{досм}^{зан} = t_{досм}.$$

В качестве времени цикла обращения к памяти принимается

$$t_{обр} = \max(t_{обр}^{счит}, t_{обр}^{зан}),$$

которое определяет интервал между двумя последовательными обращениями к памяти.

По принципу действия устройства памяти можно разделить на следующие типы:

- память на электронных схемах;
- на неподвижных ферритовых (магнитных) элементах;
- на подвижных магнитных средах (носителях);
- на лазерных дисках.

В зависимости от реализуемых операций обращения различают:

- память с произвольным обращением (ЧТ/ЗАП);
- память только для считывания или постоянные запоминающие устройства (ПЗУ).

По способу организации доступа память делится:

- на память с непосредственным (произвольным) доступом;
- на память с последовательным доступом.

Эти памяти имеют разное $t_{обр}$.

ЗУ различаются также по выполняемым функциям, зависящим от места расположения в структуре ЭВМ.

Требования к емкости и быстродействию памяти являются противоречивыми. Чем выше быстродействие, тем технически сложнее и дороже достигается большой объем памяти.

Поэтому память в ЭВМ организуется в виде иерархической структуры ЗУ, обладающих различными быстродействием и емкостью (рис. 4.13).

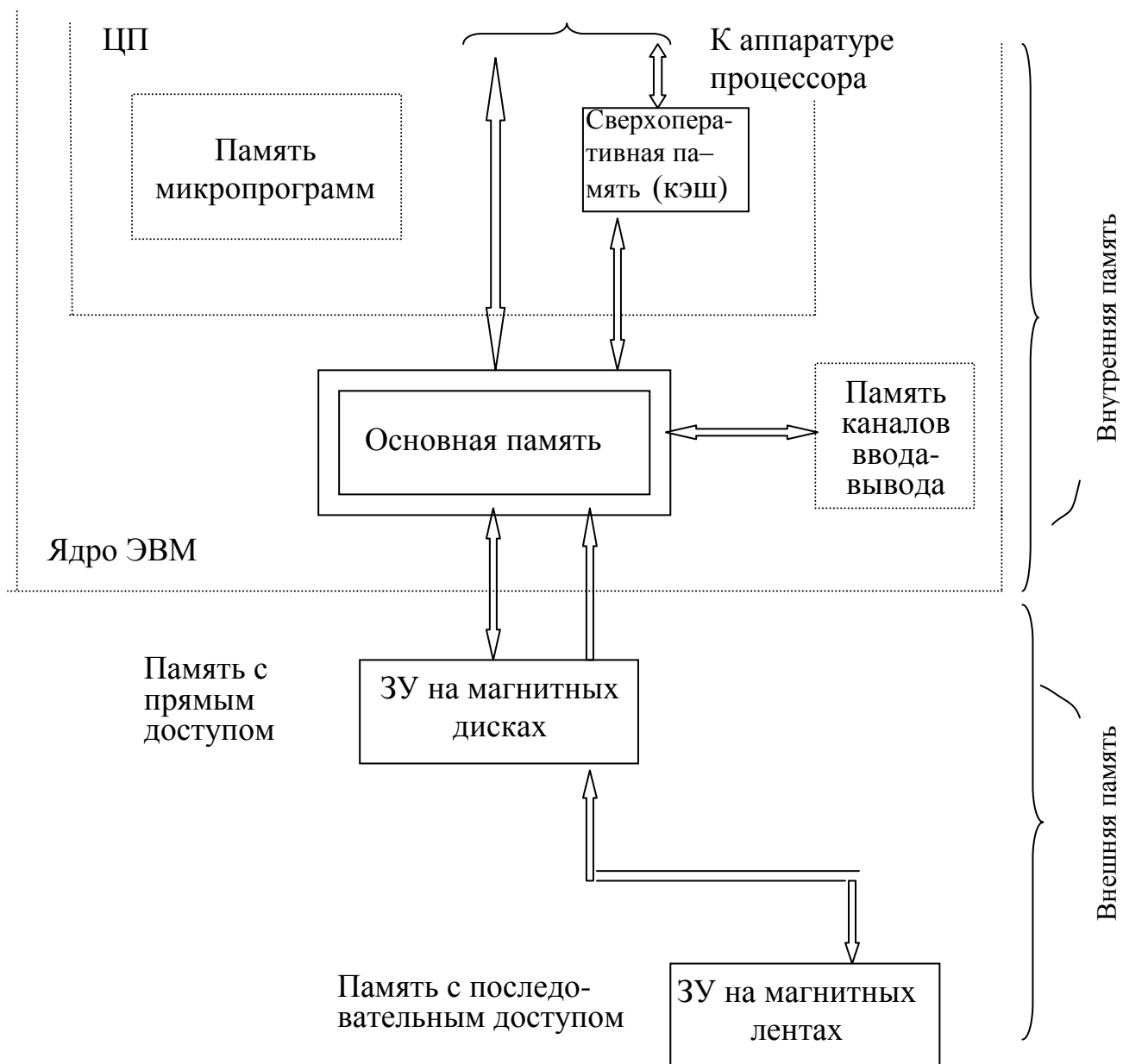


Рис. 4.13

4.7.2. Адресная, ассоциативная и стековая организация памяти

Запоминающее устройство, как правило, содержит множество одинаковых элементов, образующих *запоминающий массив (ЗМ)*. Массив разделен на отдельные ячейки, каждая из которых предназначена для хранения двоичного кода, количество разрядов в котором определяется шириной выборки памяти.

Способ организации памяти зависит от методов размещения и поиска информации в ЗМ. По этому признаку различают адресную, ассоциативную и стековую (магазинную) память [3].

Адресная память.

В такой памяти размещение и поиск информации в ЗМ основаны на использовании номера ячейки ЗМ (адреса) хранения слова (команды, числа и т.д.), в которой это слово размещается.

Для записи/чтения слова в ЗМ иницирующая эту операцию команда должна указать адрес (номер ячейки), по которому производится обращение (ЧТ/ЗП).

Схема памяти с адресной структурой (организацией) изображена на рис. 4.14. *Емкость ЗМ – N n-разрядных слов.*

Аппаратное обрамление памяти имеет:

Рг А – регистр адреса;

Рг И – регистр информационный;

БАВ – блок адресной выборки;

Дш А – дешифратор адреса;

БУС – блок усилителей считывания;

БУЗ – блок усилителей записи;

БУП – блок управления памятью;

ША – шина адреса.

Цикл работы памяти начинается с поступления в БУП сигнала «Обращение».

Общая часть цикла «Обращение» включает:

- прием в Рг А адреса ячейки памяти с ША;
- прием в БУП и расшифровку типа операции (ЧТ/ЗП);
- операцию (ЧТ/ЗП).

Если физически запись осуществляется со стиранием, то производится регенерация содержимого ячейки ЗУ.

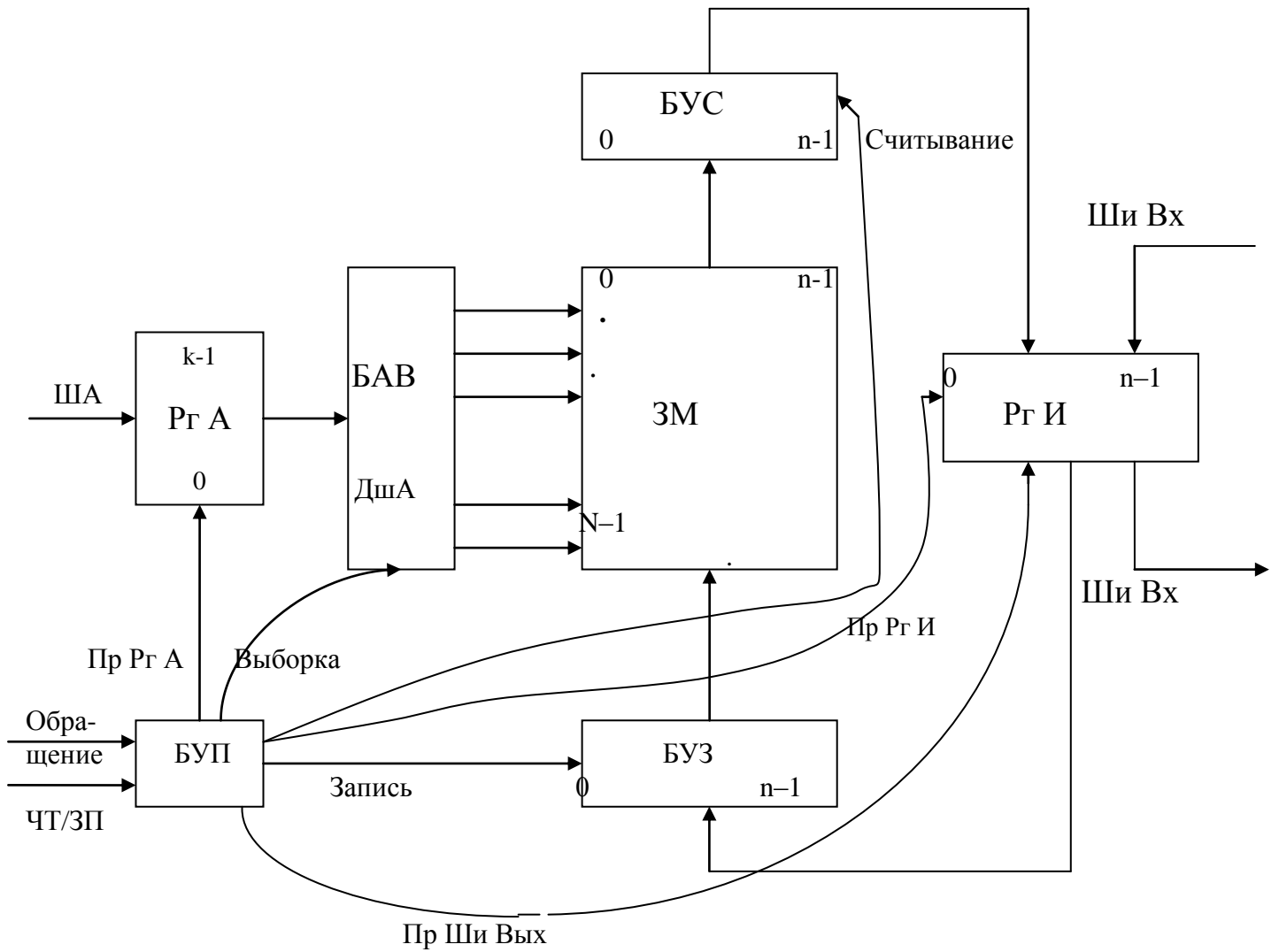


Рис. 4.14

При считывании из памяти выполняется следующая последовательность действий:

$$\text{Дш А} \rightarrow \text{ЗМ} \rightarrow \text{Рг И (через БУС)} \rightarrow \text{Ши Вых}.$$

При записи в память последовательность действий следующая:

$$\begin{array}{ccc} \text{Дш А} \rightarrow (\text{ЗМ} [\text{Дш А}] := 0) \rightarrow [\text{Ши Вх} \rightarrow \text{Рг И}] \rightarrow (\text{Рг И} \rightarrow \text{ЗМ}) \\ \text{Очистка} & & \text{Через БУЗ} \\ \text{ячейки} & & \end{array}$$

Ассоциативная память

В памяти этого типа поиск информации производится не по адресу ячейки, а по содержанию (по ассоциативному признаку (АП)). При этом поиск по АП производится параллельно во времени для всех ячеек ЗМ.

Во многих случаях поиск по АП позволяет существенно упростить и ускорить обработку данных. Это осуществляется за счет того, что в памяти этого типа операция считывания информации совмещена с выполнением ряда логических операций.

Структура ассоциативной памяти изображена на рис. 4.15.

ЗМ памяти АП содержит $N(n+1)$ -разрядных ячеек. Для указания занятости ячейки ЗМ используется дополнительный – *служебный* – n -й разряд (0 – ячейка свободна, 1 – ячейка занята (записано слово)).

По входной информационной шине Ши Вх в РгАП (регистр ассоциативного признака) в разряды $[0 - n-1]$ поступает n -разрядный ассоциативный запрос, а в регистр маски (Рг М) код маски поиска, при этом n -й разряд Рг М устанавливается в 0. Поиск по АП производится лишь для совокупности разрядов РгАП, которым соответствуют 1 в Рг М (незамаскированные разряды Рг АП).

Для слов, в которых цифры разрядов совпали с незамаскированными разрядами в Рг АП, КС устанавливает в «1» соответствующие разряды в регистре совпадения (Рг СВ) и в 0 – остальные разряды.

Таким образом, значение j -го разряда Рг СВ определяется выражением

$$\text{РгСВ } j = \bigwedge_{i=0}^{n-1} \overline{\text{РгАП } i \oplus \text{ЗМ } j,i \vee \text{РгМ } i}, \quad 0 \leq j \leq N-1$$

где Рг АП[i], Рг М [i] и ЗМ [i,j] – значения i -го разряда соответственно Рг АП, Рг М и j -й ячейки ЗМ.

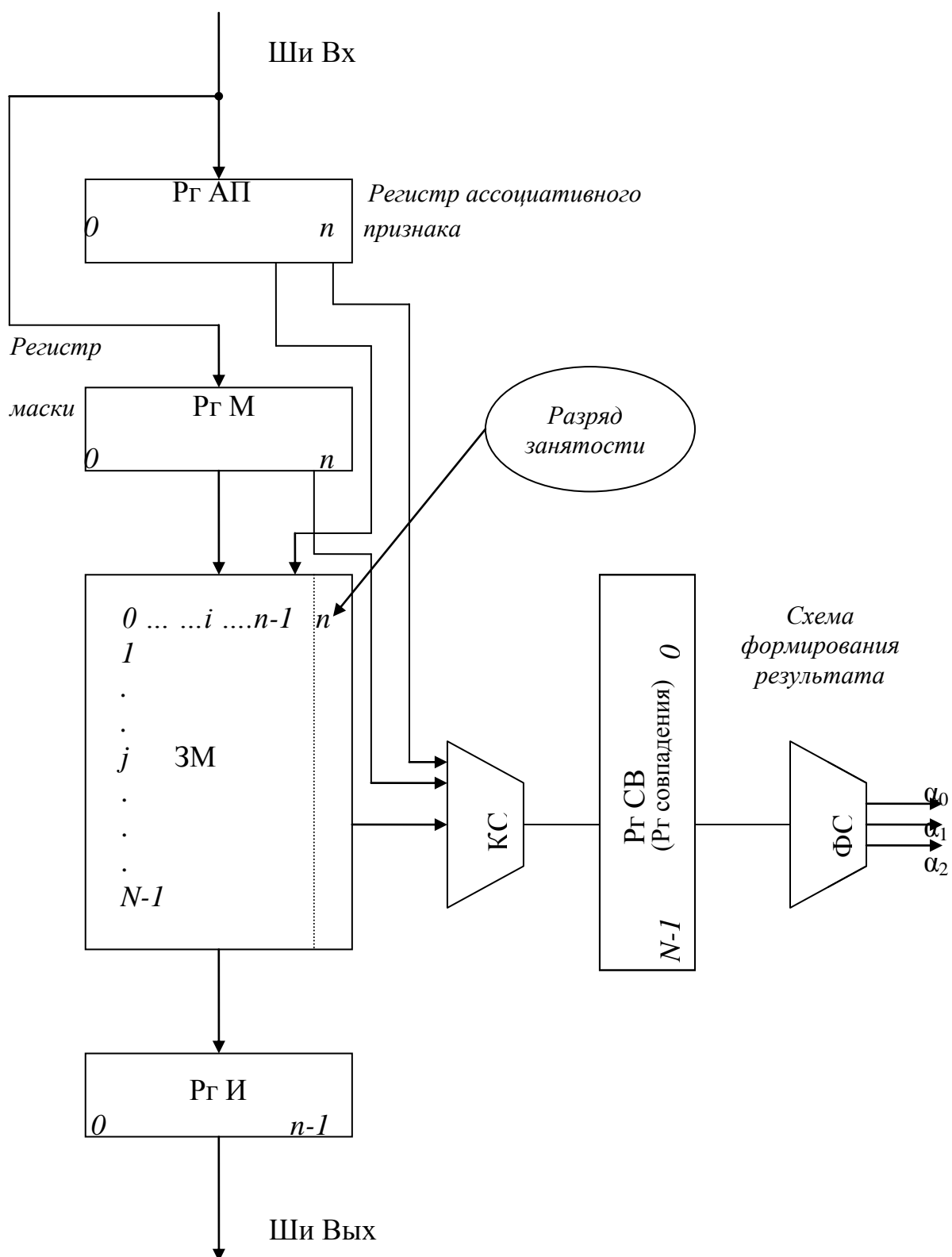


Рис. 4.15

Комбинационная схема (КС) формирования результата ассоциативного обращения в блоке формирования сигналов (ФС) формирует из слова, образовавшегося в Рг СВ, сигналы α_0 , α_1 , α_2 — соответствующие случаям отсутствия слов в ЗМ, удовлетворяющих АП, наличию одного и более одного такого слова:

$$\alpha_0 = \bigwedge_{j=0}^{N-1} \overline{\text{РгСВ}}[j] \text{ — нет такого слова ;}$$

$$\begin{aligned} \alpha_1 &= \text{РгСВ}[0] \cdot \overline{\text{РгСВ}}[1] \cdot \overline{\text{РгСВ}}[2] \cdot \dots \cdot \overline{\text{РгСВ}}[N-1] \vee \overline{\text{РгСВ}}[0] \cdot \text{РгСВ}[1] \cdot \overline{\text{РгСВ}}[2] \cdot \dots \cdot \overline{\text{РгСВ}}[N-1] \times \\ &\times \dots \vee \overline{\text{РгСВ}}[0] \cdot \overline{\text{РгСВ}}[1] \cdot \overline{\text{РгСВ}}[2] \cdot \dots \cdot \overline{\text{РгСВ}}[N-2] \cdot \text{РгСВ}[N-1] \text{ — имеется одно слово;} \\ \alpha_2 &= \overline{\alpha_0} \cdot \overline{\alpha_1} \text{ — имеется несколько слов.} \end{aligned}$$

Формирование содержимого Рг СВ и сигналов α_0 , α_1 , α_2 по содержимому Рг АП, Рг М и ЗМ является операцией контроля ассоциации. Эта операция является составной частью операции считывания и записи, хотя она имеет и самостоятельное значение.

При считывании:

вначале выполняется контроль ассоциации по АП в Рг АП. Затем при $\alpha_1 = 1$ считывается в Рг И найденное слово, а при $\alpha_2 = 1$ в Рг И считывается слово из ячейки, имеющей наименьший адрес из числа найденных в Рг СВ. Это слово из Рг И \rightarrow Ши Вых.

При записи:

сначала отыскиваются свободные ячейки. Для этого выполняется операция контроля ассоциации при Рг АП = 111...10 и Рг М = 000...01. При этом свободные ячейки отмечаются «1» в Рг СВ. Для записи выбирается свободная ячейка с наименьшим номером, в нее записывается информация, поступившая из Ши Вх в Рг И.

При помощи операции контроля ассоциации можно, не считывая слов из ЗМ, определить по содержимому Рг СВ, сколько в памяти слов, удовлетворяющих АП.

При использовании соответствующих КС в ассоциативной памяти можно выполнять достаточно сложные логические операции, например, такие, как:

- поиск $>$ ($<$) числа;
- поиск числа в границах.

Для ассоциативной памяти нужны ЗУ без разрушения информации при считывании.

Стековая (магазинная) память СП (МП)

СП (МП) так же, как и ассоциативная память, является безадресной.

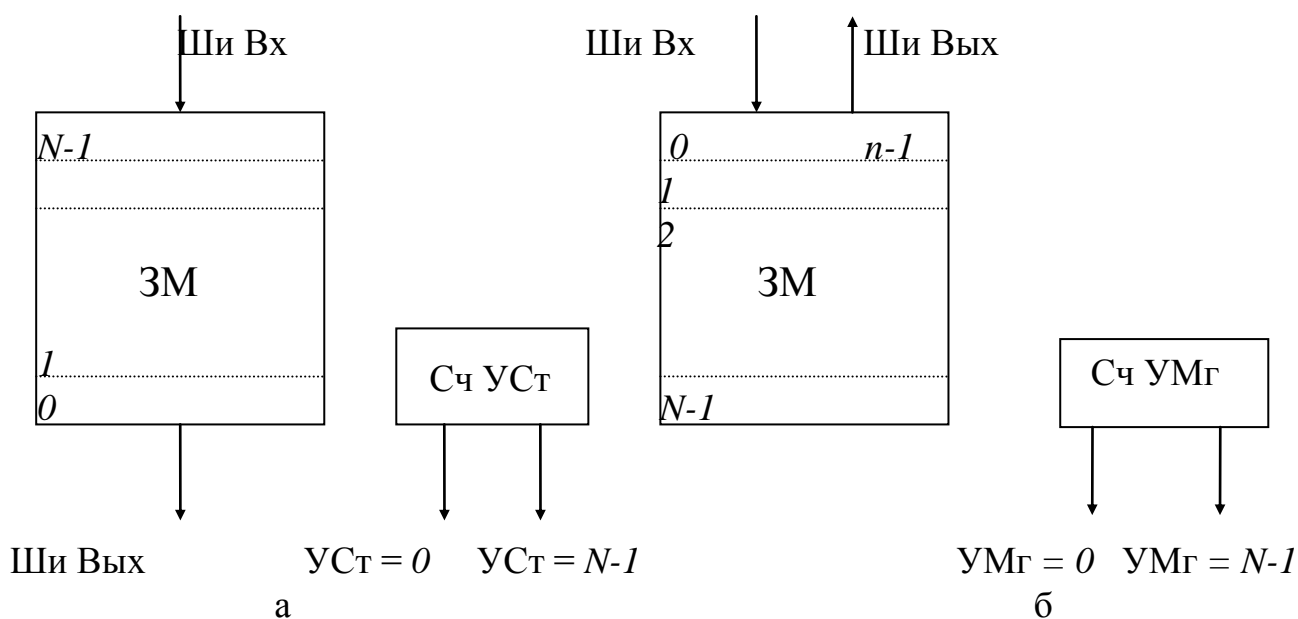


Рис. 4.16

Стековая память (рис. 4.16, а) представляет собой одномерный массив, в котором соседние ячейки связаны друг с другом разрядными цепями передачи слов.

Стек заполняется с одной стороны (Ши Вх), при этом ячейки записываются, начиная с адреса $N-1$, а считывание – с обратной стороны. При этом остальные слова сдвигаются в соседние ячейки с меньшими номерами. Стековая память работает в режиме «первым пришел – первым обслуживают» (FIFO).

В состав стековой памяти входит счетчик-указатель стека Сч УСт. При записи нового слова в стек содержимое Сч УСт увеличивается на 1, при считывании – уменьшается на 1.

Стековая память используется в ЭВМ для аппаратной организации различных очередей.

В магазинной памяти (рис. 4.16, б) запись нового слова производится в верхнюю ячейку (0), при этом все ранее записанные слова, включая и слово, находящееся в ячейке 0, сдвигаются вниз на один адрес.

Считывание производится только из верхней ячейки МП. При этом если считывание с удалением, то все оставшиеся в МП слова сдвигаются на один адрес вверх (к 0-й ячейке).

Режим работы МП – «последним пришел – первым обслуживают».

МП имеет счетчик-указатель магазина – Сч УМг.

Магазинная память используется эффективно при обработке вложенных структур данных.

4.7.3. Организация памяти в многопрограммных системах

В многопрограммных системах (МПС) размещение всех исполняемых программ (команд и данных) полностью в основную часть внутренней оперативной памяти (ОП) во многих случаях невозможно (из-за ограниченного ее объема), учитывая также, что в оперативной памяти должна располагаться резидентная часть операционной системы (ОС). Но в этом нет необходимости, так как в каждый момент работа программы производится с определенным ограниченным объемом памяти. Поэтому в оперативной памяти нужно хранить только используемые в данный момент данные, а остальные могут храниться во внешней памяти (ВЗУ на магнитных дисках или магнитных лентах).

Из этого следует:

- 1) в МПС должны быть ВЗУ большого объема со сравнительно быстрым временем доступа;
- 2) целевые программы не должны привязываться к определенному месту (адресам) ОП.

При подготовке целевых или пользовательских программ используют *условные* адреса. Позднее, в процессе подготовки к запуску (выполнению) программ СУПЕРВИЗОР присваивает целевым адресам *исполнительные* адреса. Эта процедура получила название *динамического распределения памяти* [3, 9].

Для перевода условных адресов в исполнительные применяются следующие способы:

- базирование;

- организация виртуальной памяти.

При использовании базирования свободная память может состоять из несвязанных областей (фрагментация памяти), и для ввода нужной программы может понадобиться сдвиг содержимого памяти. На рис. 4.17 приведен пример такого способа. В ОП размещены блоки (фрагменты) выполняемых программ (А, В, С и D). Предположим, что программы А и D завершены, и есть необходимость начать выполнять программу Е, для которой нужен фрагмент памяти Е, больший по объему, чем А или D. В этом случае необходим сдвиг фрагментов В и С вверх или вниз для того, чтобы в ОП мог разместиться фрагмент памяти программы Е.

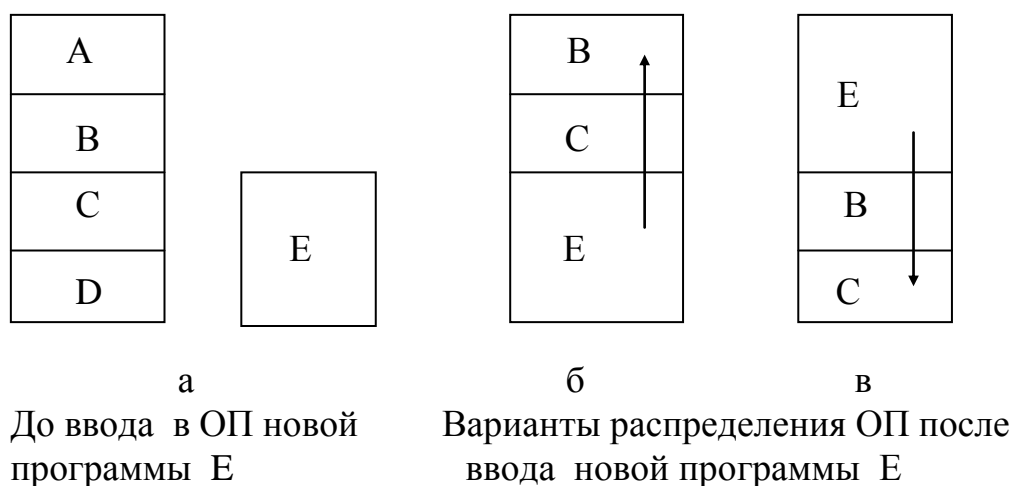


Рис. 4.17

4.7.4. Организация виртуальной памяти

Виртуальная память (ВП) – это способ организации памяти в МПС, при котором достигается гибкое динамическое распределение памяти, устраняется ее фрагментация и создаются значительные удобства для программистов [3]. Это удастся достичь без снижения производительности ЭВМ путем усложнения аппаратуры и операционной системы и процессов их функционирования.

Программист имеет дело не с реальной, а с виртуальной (т.е. кажущейся) памятью, объем которой равен адресному пространству ЭВМ (в ЕС ЭВМ и S/370 – это $2^{24} = 16777216$ битов, позже оно увеличилось до 2^{32}).

На всех этапах подготовки программ, включая загрузку в ОП, программа представляется в виртуальных адресах, и лишь при самом выполнении машинной команды производится преобразование виртуальных адресов в реальные физические адреса. Пользователь не знает об этом и о том, где (в ОП или на ВЗУ) в настоящий момент находятся его данные (и команда). Это устанавливается автоматически путем динамического распределения памяти в ходе вычислительного процесса.

Преобразование виртуальных адресов (ВА) в реальные (физические) адреса (ФА) упрощается и устраняется фрагментация памяти, если физическая память (ФП) и виртуальная память разбиты на блоки, называемые в этом случае *страницами*, содержащими одинаковое количество байтов.

Страницам ФП и ВП присваивают номера, называемые *номера* *физических и виртуальных страниц*. Каждая физическая страница способна хранить одну из виртуальных страниц. Порядок расположения байтов (нумерация байтов) в физических и виртуальных страницах сохраняется одним и тем же (рис. 4.18).

Если таблица страниц указывает, что требуемый адрес находится в ВЗУ, то вначале производится обмен между внешней и внутренней памятью. В этом случае значение $P(p)$ является для супервизора указанием для поиска информации в ВЗУ.

При многопрограммной работе ЭВМ таблица страниц должна указывать на принадлежность страниц различным пользователям. Тогда адрес P должен быть представлен как функция двух переменных:

- номера виртуальной страницы;
- номера программы N .

Следовательно, виртуальный адрес, представляющий собой число $2^\beta P + l$, где β – количество разрядов номера байта в странице, преобразуется в физический адрес $2^\beta P(P, N) + l$ (рис. 4.19).

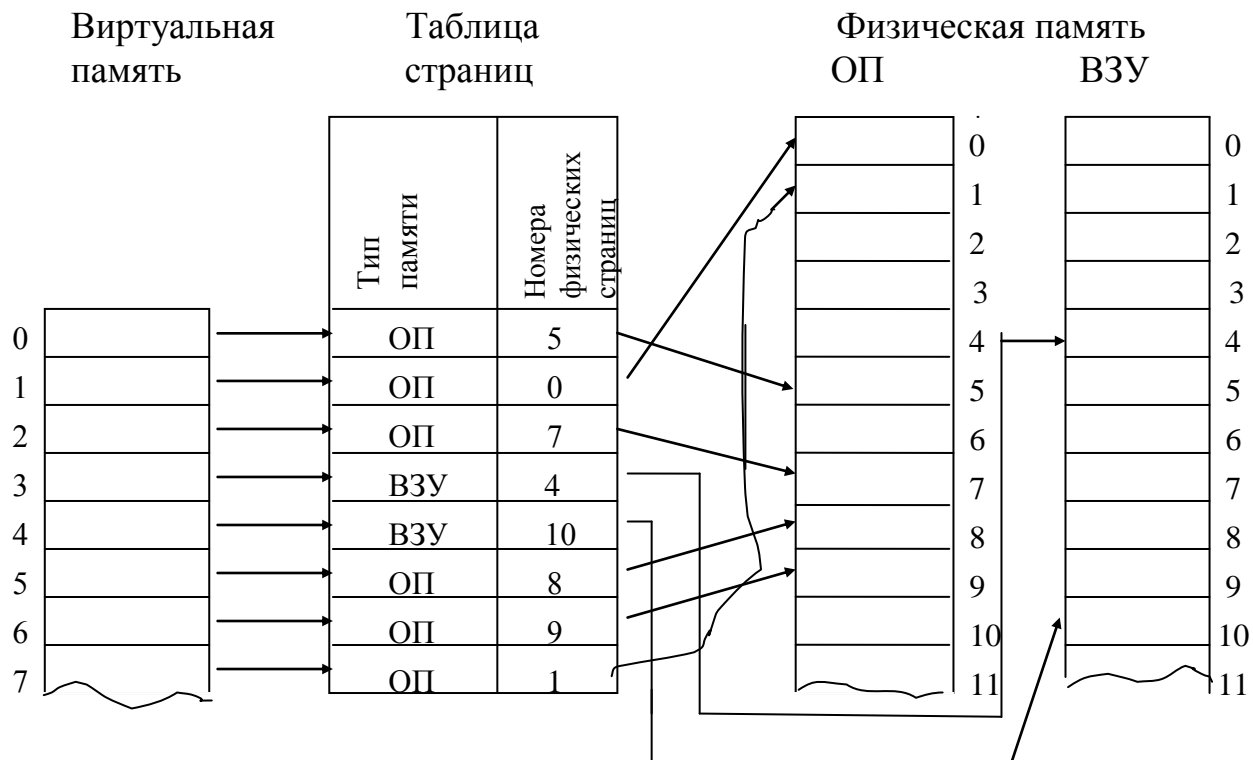


Рис. 4.18

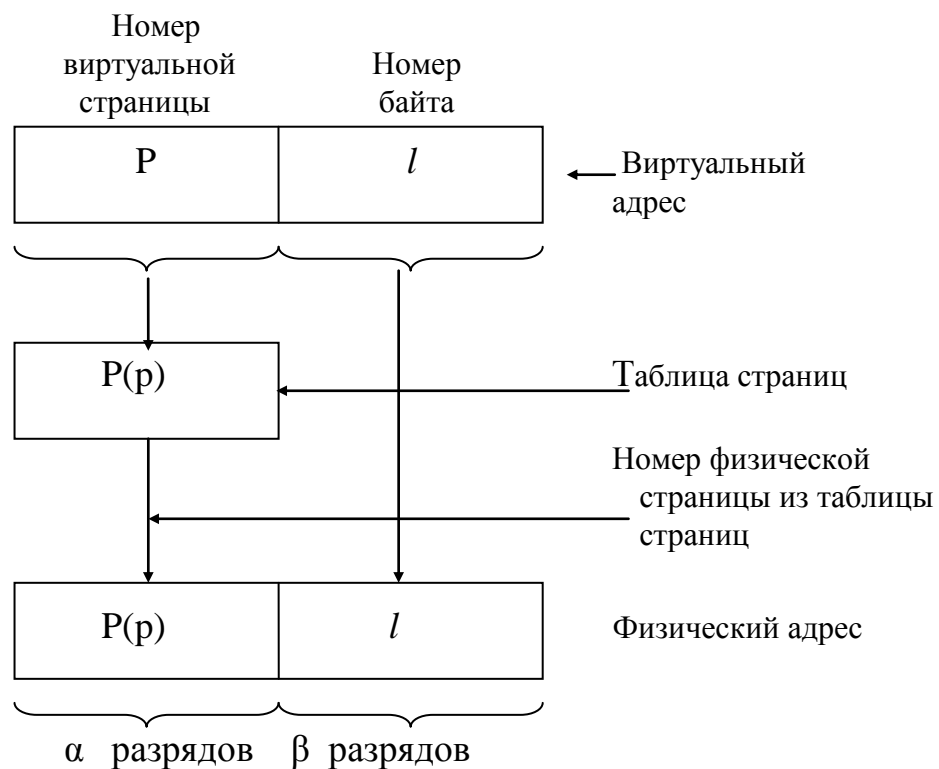


Рис. 4.19

Аппаратная реализация преобразования адресов

Известны две основные структуры таблиц страниц:

1. Каждой виртуальной странице (ВС) соответствует одна строка таблицы страниц. Строка содержит номер физической страницы, хранящей данную виртуальную страницу.

2. Каждой физической странице (ФС) соответствует одна строка таблицы. Строка содержит номер виртуальной страницы, хранимой в данной физической странице.

В реальных ЭВМ применяется сочетание обеих этих структур. При этом для реализации таблицы страниц используется ассоциативная организация памяти.

Обычно программа состоит из нескольких массивов и подпрограмм. Удобнее, если при разработке программ каждый массив имеет независимую нумерацию байтов, начиная с 0. Для динамического преобразования адресов в этом случае используют особый метод преобразования ВА в ФА, называемый *сегментной организацией памяти*.

Структура такой организации памяти еще более усложняется, так как вводится еще один уровень – *таблица сегментов*.

Для ускорения процесса переадресации используется *буфер быстрой переадресации (ББП)*, содержащий информацию о соответствии логических адресов и эквивалентных им реальных адресов, по которым производится обращение к ОП. Таким образом, учитывая, что обращение процессора к ОП обычно осуществляется по последовательным адресам и с большой вероятностью в пределах одной страницы, выборка строк таблиц из ОП производится только один раз.

В дальнейшем полученная при первом обращении информация остается в буфере и все последующие обращения к ОП, использующие строки таблиц переадресации из той же области ОП, выполняются с использованием ББП. Такой способ преобразования виртуальных адресов в физические получил название *динамической трансляции адресов (ДТА)*, схема которого приведена на рис. 4.20 [9].

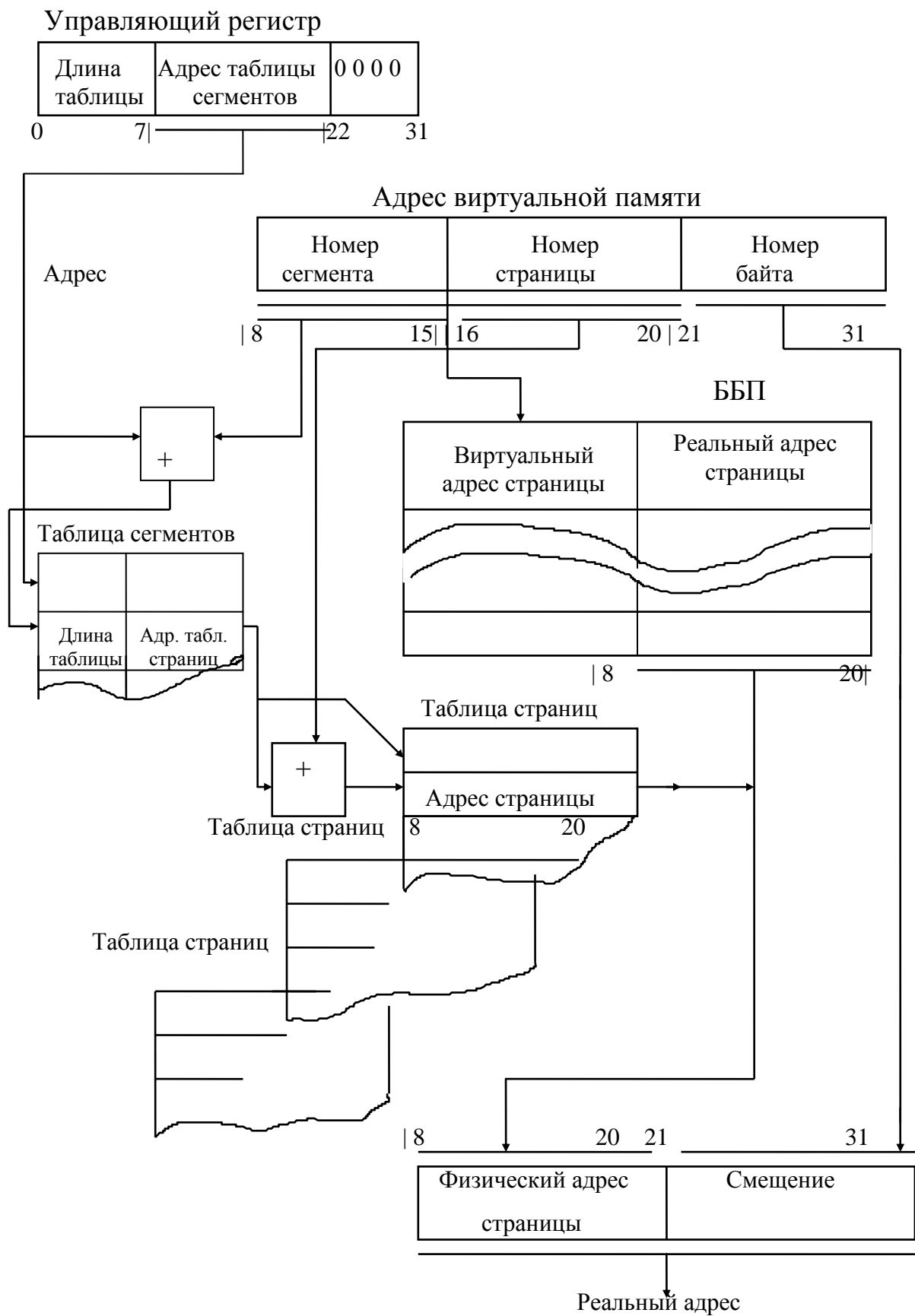


Рис. 4.20

4.7.5. Совершенствование иерархической структуры памяти

Использование иерархических систем внутренних памяти в вычислительной технике является устойчивой тенденцией. Иерархические системы памяти характеризуются следующим образом [9]:

1. Имеется несколько иерархических уровней хранения организованной в блоки информации.
2. Иерархические уровни отличаются по быстродействию и емкости, более быстродействующие памяти имеют меньшую емкость и располагаются либо в самом процессоре, либо ближе к нему – в устройстве управления (УУ) памятью.
3. Первое обращение к блоку информации приводит, как правило, к перемещению блока с более медленного уровня иерархии на более быстрый – последующие обращения к этому блоку приводят к выборке только из быстродействующей памяти.

Обычно внутренняя память имеет два уровня:

- собственно ОП;
- быстрая сверхоперативная память (БП).

БП предназначена для хранения отдельных, наиболее часто используемых участков программы и данных, что обеспечивает быстрый доступ к ним со стороны процессора. Эффект применения БП определяется временем цикла и вероятностью нахождения запрашиваемой информации в БП, что, в свою очередь, зависит от ее емкости, способа адресации и размера блока данных, которыми ОП и БП обмениваются между собой.

В идеальной системе БП должна размещать всю информацию, за которой процессор обращается в ОП. При этом вероятность обращения к БП равна 1, а эффективное время доступа к ОП равно времени цикла БП.

Однако в реальной системе вероятность обращения к БП $P \neq 1$ и изменяется в зависимости от параметров БП, подчиняясь закону, близкому к экспоненциальному:

$$P = (1 - \exp(-v/v_0)),$$

где v_0 – средний размер программы и ее рабочей области;

v – емкость СБП.

Максимальная эффективность применения БП достигается при $P=0,95-0,99$, что соответствует емкости БП=16–128 Кбайт.

Применяется 2 типа адресации БП:

- *прямая* – на месте определенного блока БП могут размещаться блоки ОП, кратные 64 ($k, k + 64, k + 128, \dots$);
- *ассоциативная* – любой блок ОП может размещаться на место любого блока БП, но для этого необходимы специальные средства.

Наибольшее распространение получила смешанная – *адресно-ассоциативная* – организация памяти (*типа КЭШ*), рис. 4.21.

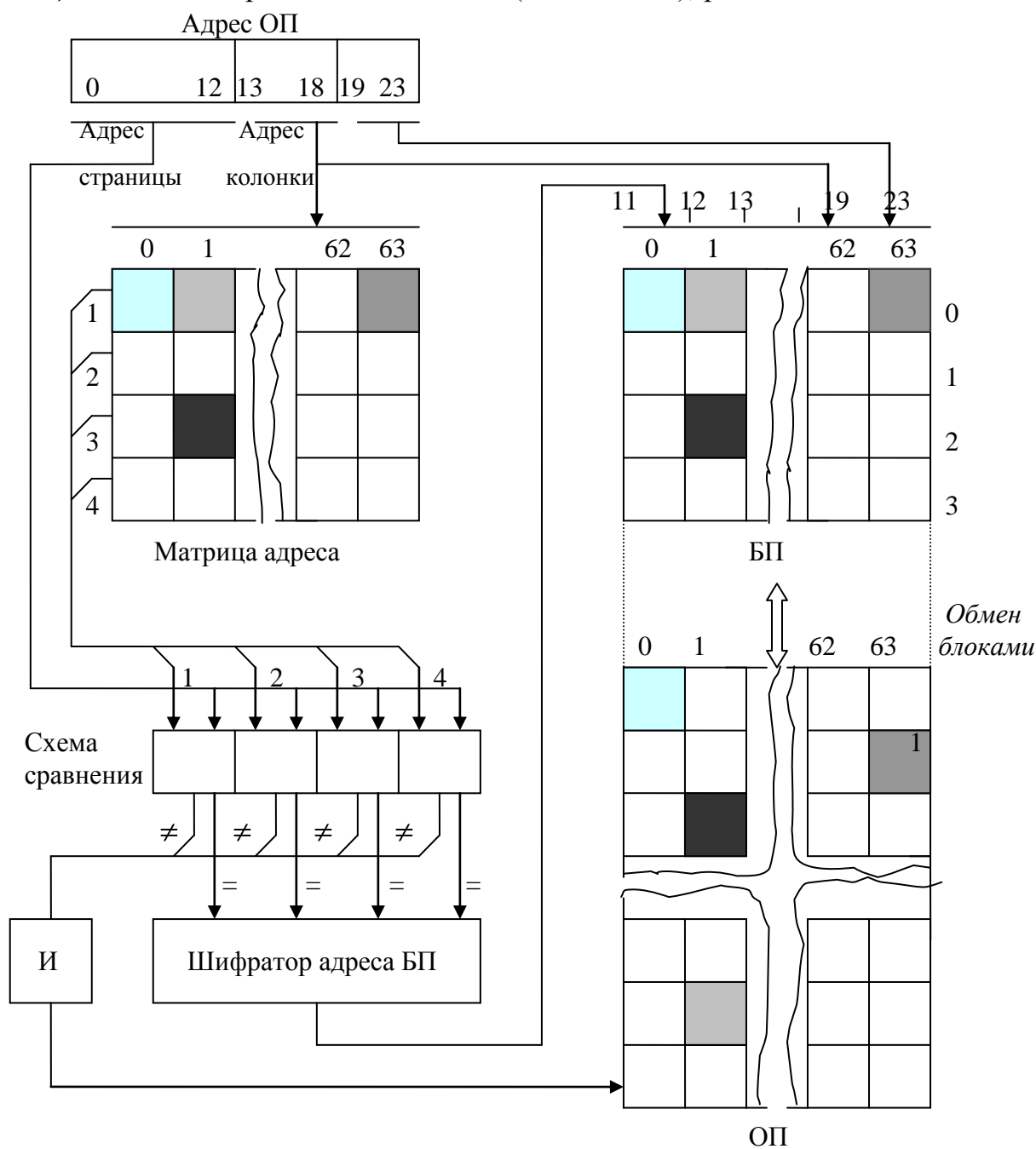


Рис. 4.21

На рис. 4.21 показано размещение информации в ОП и БП.

Содержимое БП однозначно отображает содержимое отдельных фрагментов ОП. Фрагменты определяют адресную единицу в БП: чем больше фрагмент, тем меньше различных адресов в БП, при этом уменьшается объем эффективно используемой БП.

Все поля ОП разбиты (для иллюстрации) на отдельные *страницы* – по горизонтали и *колонки* – по вертикали. Количество страниц зависит от $E_{оп}$. Так, при $E_{оп} = 8$ Мбайт ОП может содержать 4096 страниц по 2 Кбайта. При блоке информации, равном 32 байта, количество колонок постоянно и равно 64. В результате разбиения каждая страница содержит 64 блока данных ОП по 32 байта. Обмен между ОП и БП производится *блоками*.

БП тоже делится на блоки. БП может содержать блоки информации разных страниц ОП. Любой блок данных из определенной колонки ОП можно разместить в одном из блоков той же колонки БП.

БП управляется двумя массивами: *матрицей адресов (МА)* и *таблицей активности*.

Деление МА на строки и колонки соответствует делению БП.

В МА хранятся адреса тех блоков информации ОП, которые находятся в БП.

Структура адреса в МА:

- адрес страницы ОП, из которой пересылается блок данных;
- разряды достоверности данных.

Рассмотрим алгоритм работы БП-ОП.

При обращении центрального процессора (ЦП) на выборку из ОП по адресу колонки из МА одновременно считываются адреса всех блоков данных, содержащихся в данной колонке (адресуемой колонке). Эти адреса сравниваются с адресом страницы ОП.

Если требуемая информация находится в БП (сравнение произошло), то данные выбираются из БП и передаются в ЦП. При этом адрес БП определяется следующим образом:

- номер блока данных определяется сигналом с соответствующей схемы сравнения;
- адрес колонки – разрядами 13 – 18;

- адрес байта – разрядами 19 – 23 адреса обращения.

Если сравнения не происходит, значит, требуемого блока данных в БП нет и требуется обращение к ОП. В этом случае блок данных считывается из ОП, записывается в БП и передается в ЦП.

Блок данных из ОП записывается в колонку БП, номер которой определяется соответствующими разрядами адреса обращения.

Имеются определенные трудности при обмене информацией между ОП (БП) и ПФУ – должны осуществляться определенные действия по копированию содержимого БП и ОП.

Алгоритмы замещения информации в БП

Применяются в основном два способа (алгоритма):

- *точные* – точно вычисляющие номер блока, к которому долго не было обращения;

- *неточные* – определяющие блок и группу блоков информации, к которым были направлены последние по времени обращения и которые замещать не следует, т.е. номер блока для замещения определяется по некоторому приближенному правилу.

Для оценки неточных алгоритмов вводится *коэффициент качества k*, определяющий математическое ожидание номера замещаемого блока:

$$k = \sum_{i=1}^n i \cdot P_i ,$$

где n – число блоков в колонке БП;

P_i – вероятность замещения i -го блока;

i – номер блока в гипотетически жесткой очередности замещения по точному алгоритму, причем самый активный (новый) блок имеет номер 1, а самый неактивный (старый) – номер n .

Для точных алгоритмов $k = n$.

5. РАЗВИТИЕ СТРУКТУРЫ ЭВМ

5.1. Основные тенденции развития структуры ЭВМ

Развитие архитектуры неизбежно ведет к развитию структуры ЭВМ. Реализация принципов интеллектуализации, которые все больше определяют развитие архитектуры традиционных ЭВМ, возможна при совершенствовании структурной организации, обеспечивающей повышение эффективности вычислительного процесса и, как следствие этого, рост производительности ЭВМ. В конечном счете, условием и критерием развития структуры ЭВМ является рост их производительности.

Эффективность структуры ЭВМ определяется отношением $\text{стоимость} / \text{производительность}$). Требования к ЭВМ разной производительности (малых, средних и высокопроизводительных) различные [9].

Для *ЭВМ малой производительности*, как наиболее массовой, особенно важен вопрос рентабельности (требование минимизации стоимости, габаритов, эксплуатационных расходов). Выполнение этого требования предполагает использование таких методов оптимизации структуры, которые позволяют реализовать простую и надежную ЭВМ с минимальным объемом оборудования. Это достигается за счет разработки структуры на базе универсальных устройств (микропроцессоров, памяти, адаптеров и др.).

ЭВМ средней производительности использует дополнительное специализированное оборудование, ускоряющее выполнение отдельных операций и более высокую разрядность операционных блоков и шин.

Основное требование, предъявляемое к *ЭВМ высокой производительности*, – это обеспечение максимальной производительности и высокой точности вычислений. Для этих машин емкость основного и внешнего ЗУ, пропускная способность каналов ввода-вывода, степень мультипрограммируемости, состав внешних устройств выбираются из условий обеспечения полной загрузки центрального процессора при различных сочетаниях класса задач и режимов использования. Поэтому в ЭВМ высокой производительности используют все методы структурной оптимизации, высокую степень специализации составляющих ее устройств и повышенную разрядность (64–128 битов) операционных блоков.

Основными тенденциями в развитии структуры традиционных ЭВМ общего назначения являются разделение функций системы и максимальная специализация подсистем для выполнения этих функций.

5.2. Основные направления развития подсистем ЭВМ

В общем виде основные направления развития структуры для каждой подсистемы ЭВМ показаны на рис. 5.1.

5.2.1. Развитие обрабатывающих подсистем

Развитие обрабатывающей подсистемы в большей степени, чем всех остальных подсистем, идет по пути разделения функций и повышения специализации составляющих ее устройств. Создаются специальные средства, которые осуществляют функции управления системой, освобождая от этих функций средства обработки. Такое распределение функций сокращает эффективное время обработки информации и повышает производительность ЭВМ. В то же время средства управления, как и средства обработки, становятся более специализированными. Устройство управления памятью реализует эффективные методы передачи больших объемов данных между средствами обработки и подсистемой памяти.

Меняются функции центрального устройства управления – ряд функций передается в другие подсистемы (например функции ввода-вывода), развиваются средства организации многопоточковой обработки команд с одновременным повышением темпа обработки в каждом потоке, для чего применяются методы конвейерной обработки наряду с совершенствованием алгоритмов диспетчеризации и исполнения команд. Развиваются средства управления межпроцессорным обменом.

Арифметико-логические устройства обрабатывающей подсистемы кроме традиционных средств скалярной или логической обработки все шире стали включать специальные средства для векторной обработки. При этом время выполнения операций можно резко сократить как за счет увеличения количества арифметических конвейеров, а также за счет сокращения такта конвейера. Возможности задач к распараллеливанию алгоритма счета снимают потенциальные ограничения к организации существенно параллельной обработки информации и использованию структур с глубокой конвейеризацией.

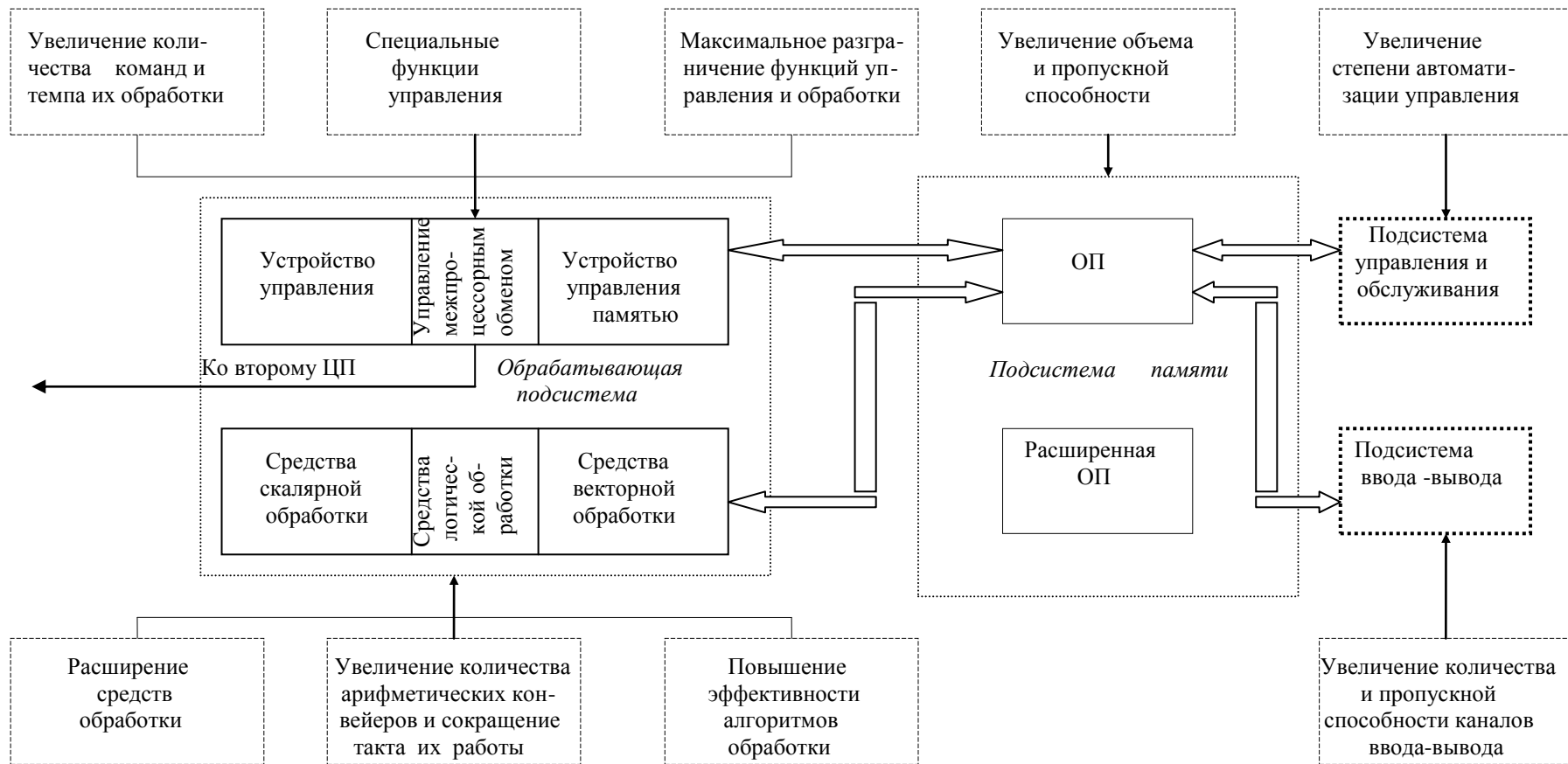


Рис. 5.1

В устройствах скалярной обработки все шире используются специальные операционные блоки, оптимизированные на эффективное выполнение отдельных операций.

Пример структуры обрабатывающей подсистемы ЭВМ общего назначения приведен на рис. 5.2.

5.2.2. Развитие подсистемы памяти

Развитие подсистемы памяти идет в направлении увеличения объема и пропускной способности. Увеличение объема оперативной памяти позволяет существенно сократить потерю времени на обмен обрабатывающей подсистемы с внешней памятью.

Характеристики оперативной памяти можно улучшить, применяя программно-доступную двухуровневую структуру. Первый уровень представляет собой основную оперативную память (емкостью до нескольких гигабайтов), второй уровень – расширенную оперативную память емкостью в сотни гигабайтов, реализованную также на интегральных схемах и являющуюся в общем случае буфером между основной оперативной памятью и подсистемой ввода-вывода. Введение расширенной оперативной памяти позволяет увеличить пропускную способность подсистемы в 300–400 раз по сравнению с пропускной способностью существующих накопителей на магнитных дисках.

Увеличение пропускной способности оперативной и расширенной памяти достигается также за счет увеличения их расслоения и секционирования.

Вопросы построения и функционирования подсистемы памяти подробно рассмотрены в подразд. 4.7 данного пособия.

5.2.3. Развитие подсистемы ввода-вывода

В состав подсистемы ввода-вывода входит набор специализированных устройств, между которыми распределены функции ввода-вывода, что позволяет свести к минимуму потери производительности системы при выполнении операций ввода-вывода.

Краткие сведения об организации ввода-вывода приведены в подразд. 4.5.

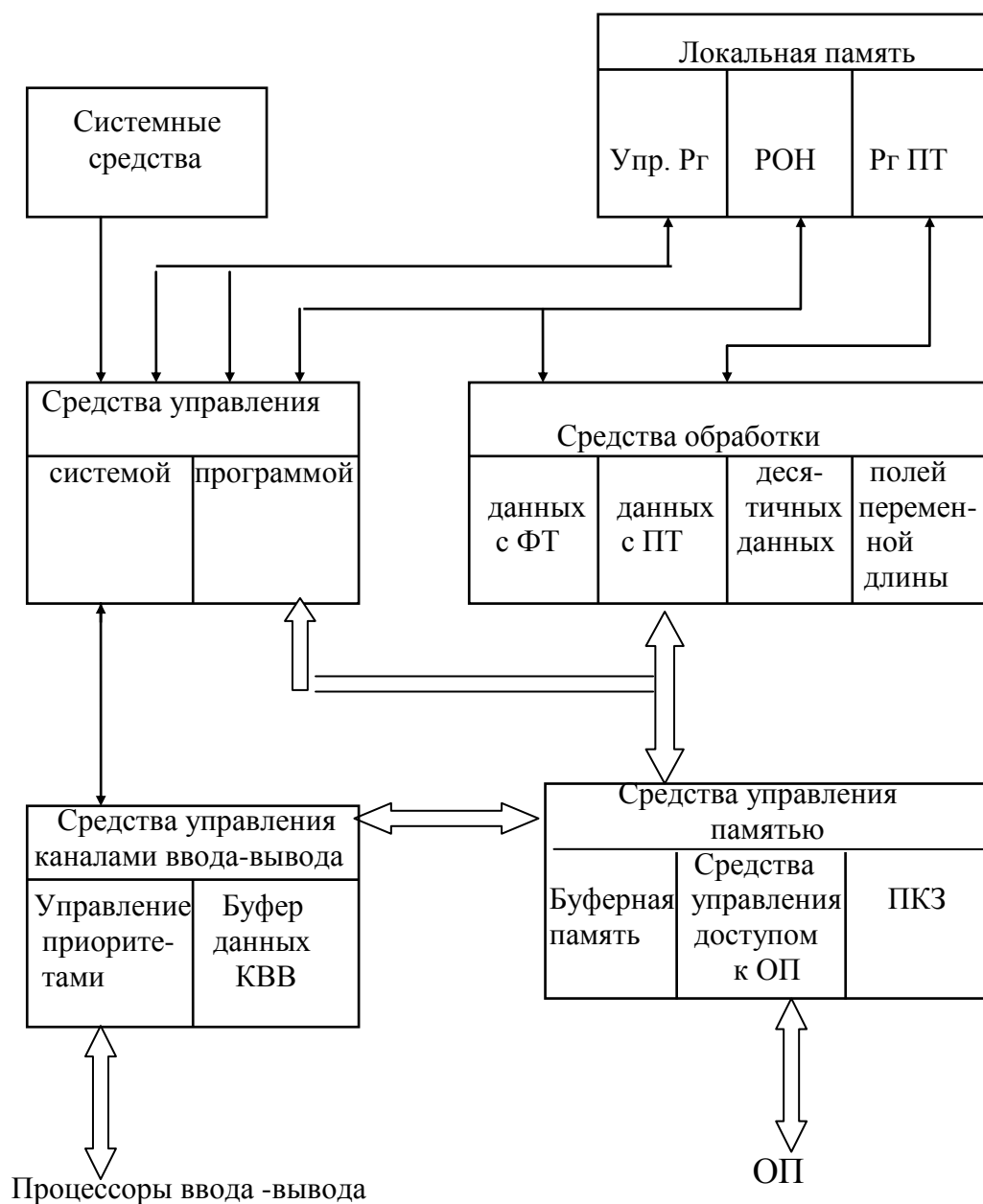


Рис. 5.2

На рисунке использовались следующие обозначения:

Упр. Рг – управляющие регистры;

РОН – регистры общего назначения;

Рг ПТ – регистры для операндов с плавающей точкой;

ПКЗ – память ключей защиты;

КВВ – каналы ввода-вывода.

Отметим, что такая организация предполагает значительную степень участия ЦП в подготовке и выполнении запроса на ввод-вывод. 25-40 % времени ЦП от общего времени обработки задания тратится на управление вводом-выводом. Поэтому одним из направлений развития подсистем ввода-вывода стало перераспределение функций между аппаратными и программными средствами, в частности, реализация части функций супервизора ввода-вывода (часто повторяемых и независимых) аппаратными средствами подсистемы ввода-вывода.

В новой структурной организации подсистемы ввода-вывода уменьшены непроизводительные потери времени ЦП на ввод-вывод за счет:

- объединения всех каналов вычислительного комплекса в единую подсистему каналов;
- исключения функций управления вводом-выводом из программы ЦП;
- исключения логической связи между каналом и ЦП, инициирующим процедуру ввода-вывода;
- увеличения количества ЦП.

Для отличия подсистему ввода-вывода с расширенными функциями будем называть *подсистемой каналов ввода-вывода*. Сравнение подсистем ввода-вывода и подсистемы каналов показано на рис. 5.3. Любой ЦП может запустить операцию ввода-вывода с любого устройства и получить прерывание от любого устройства через каналную систему. Команды ввода-вывода оперируют только с логическими адресами и структурами данных, которые присвоены лишь одному внешнему устройству. Прерывания каналной подсистемы обслуживаются любым ЦП независимо от того, какой из процессоров запускал операцию ввода-вывода.

Новыми функциями подсистемы ввода-вывода являются:

- управление на основе подканалов и канальных путей;
- независимая от канального пути адресация внешних устройств;
- динамическое повторное соединение ЦП с каналной подсистемой и др.

Существенная особенность развитой подсистемы ввода-вывода (подсистемы каналов) – наличие общих подканалов. С точки зрения управления устройство ввода-вывода представляется определенным логическим подканалом, с которым оно связывается однозначно, независимо от количества физических канальных путей, используемых для связи с каналной подсистемой.

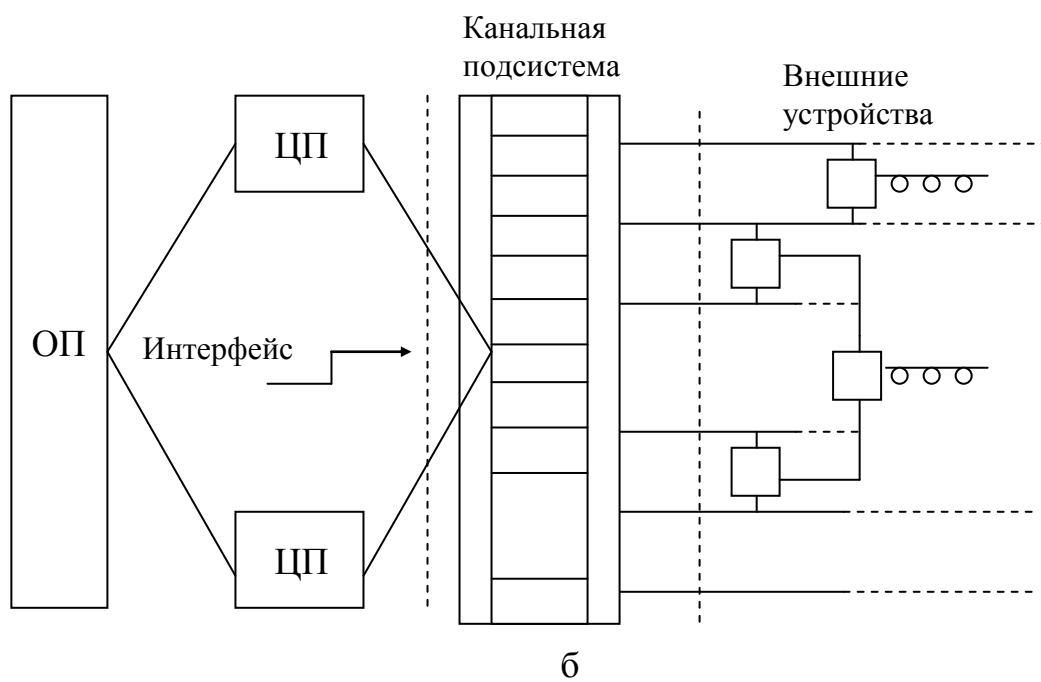
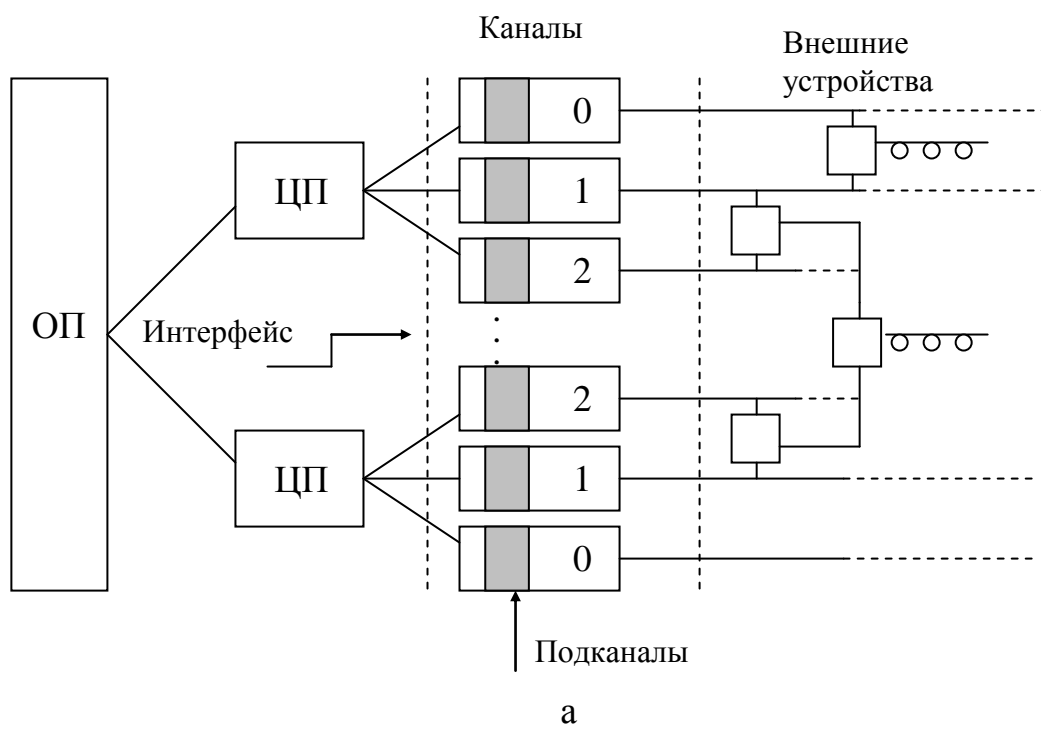


Рис. 5.3

Канальная подсистема выбирает канальный путь и обрабатывает все условия занятости внешних устройств без участия ЦП. Архитектура новых подсистем ввода-вывода рассчитана на тысячи подканалов.

Канальная подсистема формирует и обновляет специальные *таблицы конфигурации*, на основании которых и выполняется поиск требуемого устройства, т.е. реализует возможность динамического выбора пути доступа к внешнему устройству. Динамический выбор может быть использован для продолжения выполнения цепочки команд, повторения команды или подключения устройства к каналу в блок-мультиплексном режиме.

Блок-мультиплексный режим – это режим, при котором производится обмен информацией между каналом и несколькими внешними устройствами, но при этом обмен между каналом и внешним устройством производится блоками данных и в процессе передачи блока данных канал работает в монопольном режиме. Мультиплексирование разрешается только между блоками данных.

Таким образом, приостановленную канальную программу можно снова включить через любой свободный канальный путь.

Подсистема каналов ввода-вывода может работать в различных алгоритмах обмена канальной системы с различными устройствами и обеспечивает открытость подсистемы для реализации новых функций, а также развития принципов виртуальной ЭВМ. Архитектура приспособлена к возможным эволюциям и рассчитана на дальнейшее ее функциональное совершенствование.

Развитие ЭВМ, появление новых поколений требовали не только повышения гибкости подсистемы ввода-вывода, снижения затрат ЦП на организацию ввода-вывода, но и повышения производительности (пропускной способности) собственно подсистемы ввода-вывода. Основным способом повышения пропускной способности подсистемы ввода-вывода является увеличение количества каналов ввода-вывода. В современных ЭВМ используется от 32 до 256 каналов ввода-вывода) и повышение пропускной способности интерфейса ввода-вывода (в современных ЭВМ применяются: параллельный (физический) интерфейс ввода-вывода с пропускной способностью до 4,5 Мбайт/с, оптоволоконный интерфейс (ESCON) – с пропускной способностью до 17 Мбайт/с и возможностью удаления внешнего устройства от канала на

расстояние до нескольких километров, а также новый вид интерфейса – FICON с пропускной способностью до 100 Мбайт/с.

В заключение отметим, что при разработке ЭВМ все подсистемы должны быть сбалансированы между собой. Только оптимальное согласование быстродействия обрабатывающей подсистемы с пропускной способностью системы ввода-вывода позволяет добиться максимальной эффективности использования ЭВМ для всех задач и во всех режимах.

5.2.4. Развитие подсистемы управления и обслуживания

Подсистема управления и обслуживания лишь косвенно влияла на производительность ЭВМ. Однако с включением в ее состав сервисного процессора она все больше начинает принимать участие в вычислительном процессе, так она непосредственно участвует в организации передачи сообщений от одного процессора к другому в мультипроцессорной системе. Сервисный процессор эффективно участвует во взаимодействии устройств обрабатывающей подсистемы, поддерживая быстрое восстановление вычислительного процесса после сбоев, в организации необходимого технического обслуживания отдельных устройств (одновременно и параллельно с решением задач на ЭВМ).

Подсистема управления и обслуживания (ПУО) позволяет поднять на качественно новый уровень эксплуатацию современных ЭВМ, усложнение которых выражается в росте количества уровней иерархии ЭВМ, росте числа элементов на каждом уровне иерархии, усложнение межэлементных связей на каждом уровне, усложнение настройки, усложнение протоколов межуровневых взаимоотношений, жестких временных ограничений снизу на скорость прохождения информационных потоков.

Выделение ПУО в самостоятельную структурную единицу – логическое развитие принципа постоянного усложнения ЭВМ.

ПУО современных ЭВМ представляет собой аппаратно-программный комплекс, выполняющий широкий спектр задач, основными из которых являются:

- инициализация ЭВМ;
- поддержка ручных операций оператора;
- обеспечение автоматической и интерактивной диагностики сбоев и отказов и передача информации для систем обработки сбоев и восстановления после них;

- автоматическая перезагрузка операционной системы и содержимого памяти микропрограммы;
- накапливание долговременных и текущих статистических сведений о загрузке и состоянии вычислительного процесса;
- определение потери производительности как функции деградации вследствие неисправности отдельных устройств;
- обеспечение блокировки несанкционированного доступа при случайных или преднамеренных действиях;
- обеспечение связи с центром техобслуживания через удаленный доступ, по инициативе оператора;
- обеспечение совмещения ремонта отказавших ресурсов с управлением остальной части вычислительного комплекса;
- управление электропитанием, охлаждением, синхронизацией системы;
- поддержка средств мультипроцессирования;
- поддержка справочно-информационной системы;
- предоставление «интеллектуального» интерфейса различным категориям обслуживающего персонала.

Круг задач, решаемых ПУО, постоянно расширяется.

5.3. Развитие операционных сред (архитектур) в ЭВМ

С развитием поколений ЭВМ изменялась их операционная среда (см. табл. 1.3).

Рассмотрим наиболее развитые операционные среды поколений ЭВМ – виртуальную и последующие.

5.3.1. Архитектура виртуальных ЭВМ

Основной концепцией ЭВМ третьего поколения является совместное использование (разделение) ресурсов многими пользователями (заданиями). Дальнейшее развитие архитектуры ЭВМ происходит в направлении более широкого применения принципа *виртуального распределения ресурсов* [9].

Основным свойством архитектуры виртуальной машины (ВМ) является отсутствие в интерфейсе пользователя с ЭВМ ограничений на предоставляемые ресурсы.

Архитектура ВМ обладает рядом важных достоинств:

- одно и то же задание может быть выполнено на ЭВМ с различными конфигурациями устройств;
- возможны дальнейшее развитие ЭВМ и адаптация новых ТС;
- каждая ВМ абсолютно надежна и имеет высокую степень защиты от «вмешательства» извне.

Основным недостатком ВМ являются большие системные потери по организации виртуальных средств.

Концептуальная модель виртуальной ЭВМ приведена на рис. 5.4.



Рис. 5.4

5.3.2. Архитектура объектной ЭВМ

Дальнейшим развитием архитектуры ВМ является архитектура так называемой *объектной ЭВМ (ОМ)*, характерным свойством которой является операционная среда более высокого уровня, определяемая в терминах процессов и данных, что позволяет полностью исключить из интерфейса пользователя с ЭВМ понятие реальной машины [12].

Концепция объектно-ориентированной архитектуры (архитектуры объектной машины) предусматривает:

1) высокоуровневый объектный машинный язык (ОМЯ), использующий единое представление для всех объектов задачи (программы, данные, файлы и др.) и команды высокого уровня для управления объектами;

2) одноуровневую память с практически неограниченным адресным пространством и уникальными идентификаторами объектов, что делает возможным аппаратное управление доступом к информации во внешней памяти;

3) интегрированные в объектный язык средства управления процессами и данными, в том числе организации базы данных, что существенно упрощает программирование;

4) интегрированные в аппаратуру средства управления ресурсами физического уровня, что упрощает функции операционной системы и делает их выполнение более эффективным.

Наиболее значительное отличие архитектуры ОМ от архитектуры ВМ заключается в организации одноуровневой схемы управления памятью, при которой и внешняя, и оперативная памяти рассматриваются как единое целое, одинаково адресуемое пространство, разделенное на *сегменты*.

Объекты, которые представляют собой высокоуровневые конструкции (программы, наборы данных и др.), размещаются в одном или нескольких сегментах.

Сегменты имеют *оглавление*, которое определяет тип и атрибуты объекта, а также дополнительные сегменты данного объекта. В них вместо адресов используются указатели, отсутствуют регистры и не осуществляется адресация отдельных сегментов внутри объектов.

Указатель является логическим адресом – уникальным идентификатором объекта в системе, содержащим виртуальный адрес и другую информацию доступа.

Другое важное отличие ОМ – интеграция основных функций управления на уровень микропрограмм, что следует непосредственно из свойств объектного уровня машинного языка, который не содержит традиционных команд управления ресурсами. Отсутствие прямой связи между объектами и ресурсами делает возможным интерпретацию ОМ на различных ВМ.

Это требует огромных аппаратных средств поддержки на реальной машине (РМ). Эффективность работы системы обеспечивается переносом функций управления системой на специализированные процессоры, например, процессор ввода-вывода, сервисный процессор и др. Поэтому одним из сопутствующих свойств ОМ является *функционально-ориентированная организация (ФО)* – отдельные системные функции выполняются независимыми специализированными устройствами.

Архитектура ОМ еще более приспособлена к возможным эволюциям в будущем. Своеобразная гибкость архитектуры обеспечивается принципиальными преимуществами микропрограммного управления. С внедрением этой архитектуры в системах снижается объем управляющего системного программного обеспечения (СПО), полностью исключаются ресурсы системы из средств программирования, обеспечиваются высокие надежность, уровень защиты, гибкость реализации (рис. 5.5).

5.3.3. Архитектура интеллектуальной ЭВМ

ЭВМ пятого поколения – это естественное развитие ЭВМ в направлении повышения доступности для конечного пользователя, т.е. удовлетворение требований пользователей различных профессий и разной квалификации [9, 10].

Для этого ЭВМ пятого поколения должна иметь следующие основные возможности:

- интеллектуальный интерфейс, обеспечивающий доступ пользователя к ЭВМ на языке проблемной области в естественной для пользователя форме (текст, речь и т.д.);

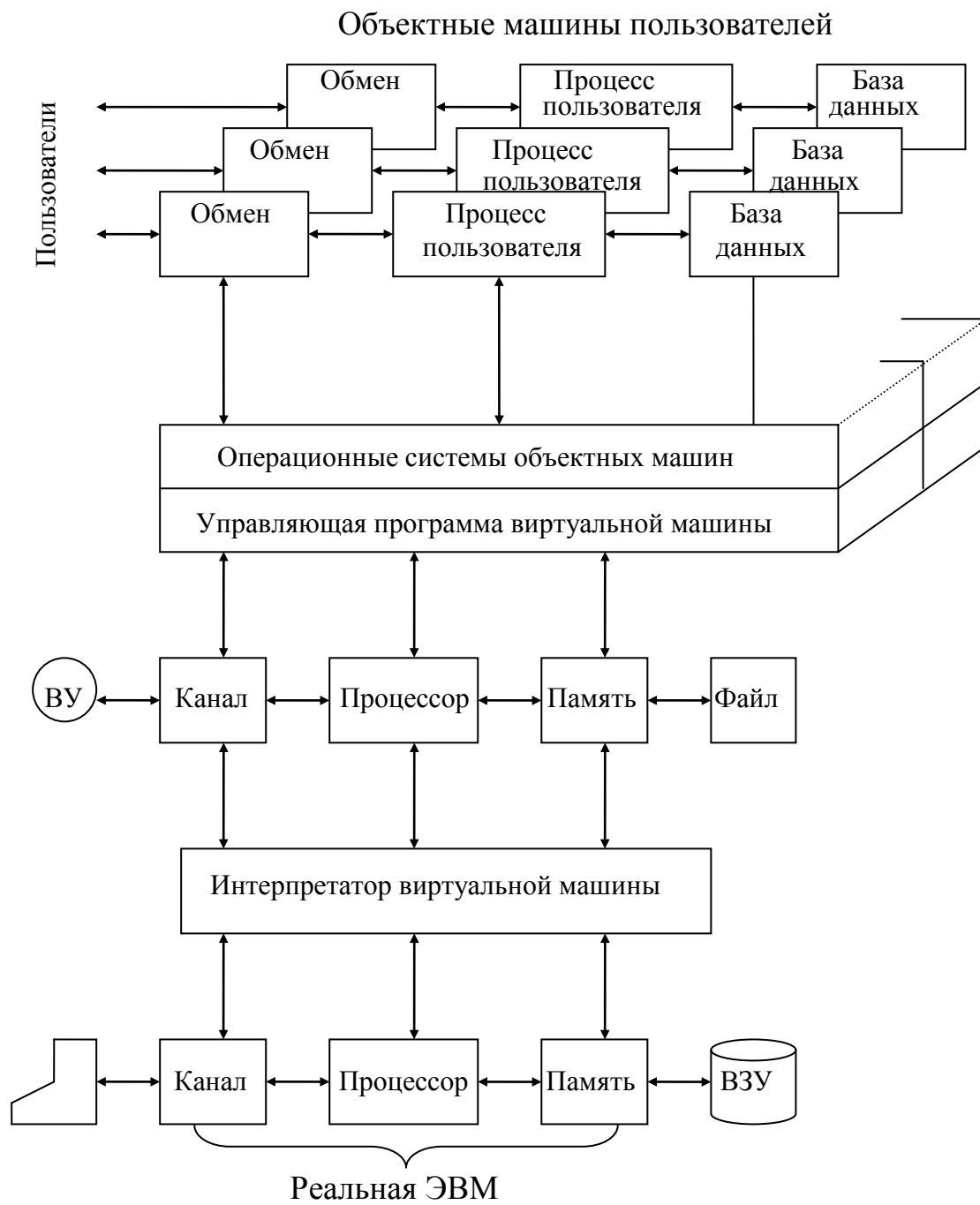


Рис. 5.5

- развитые инструментальные системы программирования, обеспечивающие создание пользователем прикладных систем и их настройку на конкретную предметную область;

- базу разнообразных в семантическом и синтаксическом отношениях знаний с развитыми механизмами упорядочивания и поиска информации;

- средства организации распределенного доступа пользователя к средствам хранения и обработки информации в виде общей сети интеллектуальных персональных ЭВМ, работающих в качестве сетевых абонентских пунктов (рис. 5.6).

ЭВМ пятого поколения должна будет иметь следующие подсистемы:

Подсистема общения с пользователем (интеллектуальный интерфейс).

2. Подсистема анализа и логического вывода, обеспечивающая выбор методов решения и синтез программ с учетом контекста и содержания задачи, которые дополняются знаниями общего характера из *базы знаний (БЗ)* ЭВМ. Результатом является формирование алгоритма решения.

Формирование программы обработки данных осуществляется с использованием методов решения из БЗ. Соответственно осуществляется выбор вычислительных и логических средств ЭВМ.

3. Подсистема решения задач потребует средств высокой производительности.

4. Подсистема управления и обслуживания обеспечивает надежность ВС, в том числе перемещаемость программ и данных, автоматический контроль и восстановление с целью поддержания живучести.

5. База знаний – накопление, обработка и хранение разнообразных знаний, поддерживающих функциональные возможности интеллектуальной ЭВМ: базу интерфейсных знаний, базу проблемных знаний и базу системных знаний.

Знание – это закономерности предметной области (принципы, связи, законы), полученные в результате практической деятельности и профессионального опыта, позволяющие специалистам ставить и решать задачи в этой области [16].

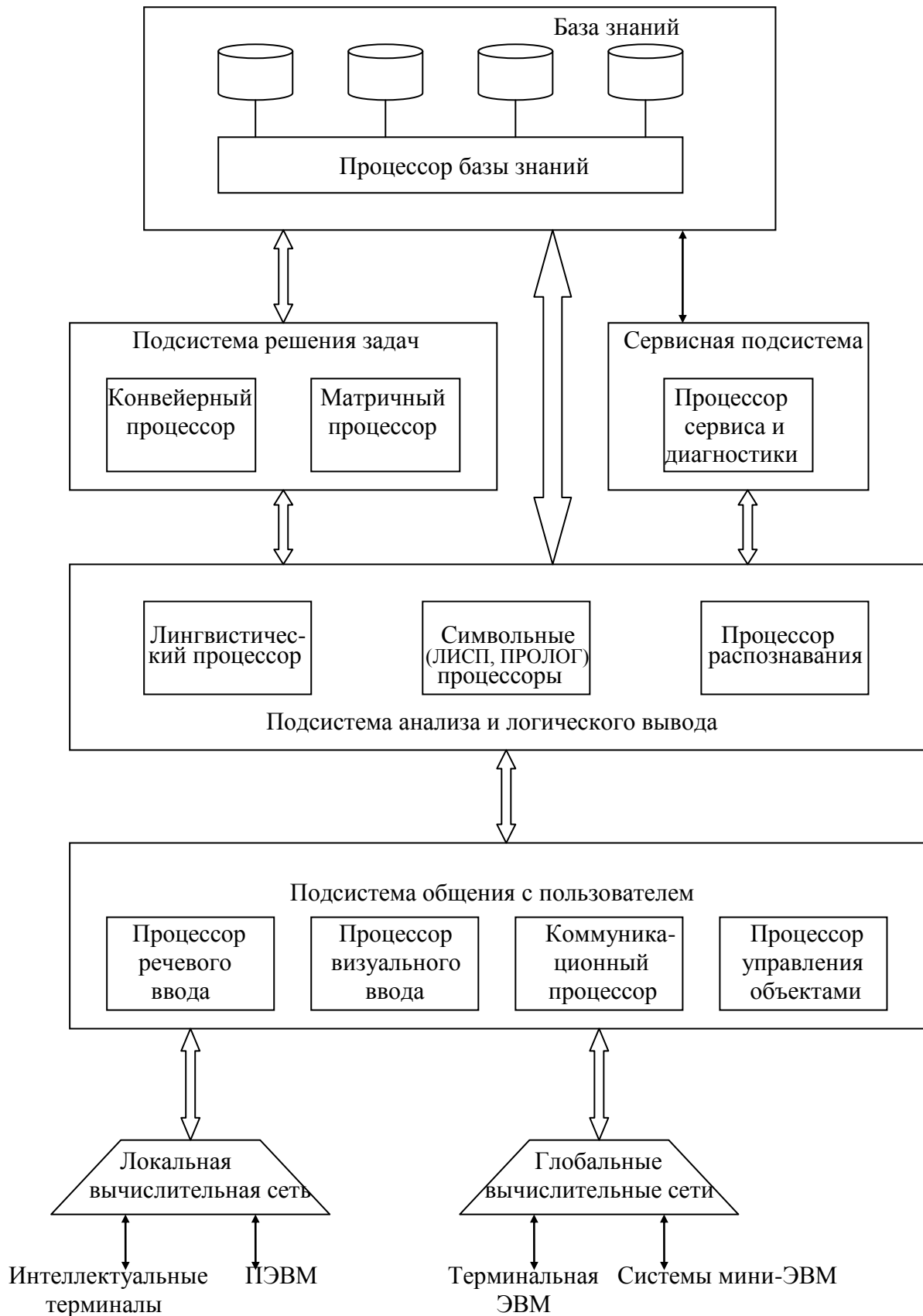


Рис. 5.6

5.4. Параллельные компьютеры для интеллектуальных систем

В последнее время в различных областях получили широкое применение разнообразные интеллектуальные системы: поддержки принятия решений, экспертные, обучающие, тренажерные и др., являющиеся в своей основе системами обработки знаний. Системы обработки знаний имеют свою специфику, обусловленную, во-первых, большими объемами перерабатываемой информации, во-вторых, высоким уровнем сложности структур перерабатываемых данных, в-третьих, трудно формализуемым характером перерабатываемых знаний. Практическая ценность интеллектуальных систем (особенно это касается систем поддержки принятия решений в чрезвычайных ситуациях и в управлении реальными объектами) определяется в значительной мере реальным масштабом времени ее реакции на поступающие запросы. Общеизвестно, что решение проблемы функционирования вычислительных систем обработки знаний в реальном масштабе времени труднодостижимо без использования принципов параллельной обработки и ассоциативного доступа к структурам данных. Разработка подобных систем чрезвычайно трудоемка, поэтому для обеспечения ее экономической целесообразности жизненный цикл функционирования подобных систем должен быть достаточно продолжительным (от десятка лет и более). Сменяемость аппаратных и программных средств вычислительной техники в настоящее время достигает от нескольких лет до нескольких месяцев. Это обуславливает необходимость наличия у интеллектуальных систем таких качеств, как открытость (в плане модифицируемости и добавления новых методов представления и переработки знаний) и интегрируемость (в смысле возможности функционирования в составе комплексов различных вычислительных и исполнительных средств). Анализ современных систем обработки знаний показывает, что ни одна из архитектур, на базе которых они реализованы, не обладает в совокупности всеми указанными выше свойствами.

5.4.1. Классификация параллельных архитектур

В процессе развития вычислительной техники роль параллельной обработки информации менялась. Если ранее применение ЭВМ диктовалось необходимостью увеличения надежности оборудования управляющих систем, то современные параллельные ЭВМ используются для ускорения счета в различных

областях. Особое значение приобретает параллельная обработка для ЭВМ, предназначенных для выполнения алгоритмов искусственного интеллекта. Такие алгоритмы часто носят комбинаторный характер и требуют большой вычислительной мощности [11].

Параллелизм – это возможность одновременного выполнения нескольких арифметико-логических или служебных операций. На стадии постановки задачи параллелизм не определен, он появляется только после выбора метода вычислений. В зависимости от характера этого метода параллелизм алгоритма может меняться в довольно больших пределах. Параллелизм используемой ЭВМ также меняется в широких пределах и зависит в первую очередь от числа процессоров, способов размещения данных, методов коммутации и способов синхронизации процессов. Язык программирования является средством переноса параллелизма алгоритма на параллелизм ЭВМ и тип языка может в сильной степени влиять на результат переноса.

Для сравнения параллельных алгоритмов необходимо уметь оценивать степень параллелизма. Одновременное выполнение операций возможно, если они не зависят друг от друга по данным или управлению.

Можно выделить следующие основные формы параллелизма: естественный, или *векторный параллелизм*; *параллелизм независимых ветвей*; параллелизм смежных операций, или *скалярный параллелизм*.

Отметим, что для скалярного параллелизма часто используют термин *мелкозернистый параллелизм* (МЗП). К *крупнозернистому параллелизму* (КЗП) относят векторный параллелизм и параллелизм независимых ветвей. КЗП оперирует с крупными информационными объектами: ветвями программ и векторами.

В зависимости от стадии разработки полезными оказываются различные характеристики эффективности ЭВМ. Рассмотрим *ускорение* r параллельной системы, которое используется на начальных этапах проектирования или в научных исследованиях для оценки предельных возможностей параллельной архитектуры. Ускорение определяется выражением

$$r = T_1 / T_n,$$

где T_1 – время решения задачи на однопроцессорной системе,

T_n – время решения той же задачи на n -процессорной системе.

Пусть $W = W_{ск} + W_{пр}$,

где W – общее число операций в задаче,

$W_{пр}$ – число операций, которые можно выполнять параллельно,

$W_{ск}$ – число скалярных (не распараллеливаемых) операций.

Обозначим также через t время выполнения одной операции. Тогда получаем закон Амдала :

$$r = \frac{W \cdot t}{(W_{ск} + \frac{W_{пр}}{n}) \cdot t} = \frac{1}{a + \frac{1-a}{n}} \xrightarrow{n \rightarrow \infty} \frac{1}{a}.$$

Здесь $a = W_{ск} / W$ – удельный вес скалярных операций.

Закон Амдала определяет принципиально важные для параллельных вычислений положения :

1. Ускорение вычислений зависит как от потенциального параллелизма задачи (величина $1 - a$), так и от параметров аппаратуры (числа процессоров n).
2. Предельное ускорение вычислений определяется свойствами задачи.

Пусть, например, $a = 0,2$ (что является реальным значением), тогда ускорение не может превосходить 5 при любом числе процессоров, т.е. максимальное ускорение определяется потенциальным параллелизмом задачи. Очевидной является чрезвычайно высокая чувствительность ускорения к изменению величины a .

Рассмотрим некоторые классификации параллельных вычислительных архитектур [11].

Классификация М. Флинна

Критерии классификации параллельных ЭВМ с крупноформатным параллелизмом могут быть разными: вид соединения процессоров, способ функционирования процессорного поля, область применения.

Одна из наиболее известных классификаций параллельных ЭВМ предложена Флинном и отражает форму реализуемого ЭВМ параллелизма. Основными понятиями классификации являются «поток команд» и «поток данных». Под потоком команд упрощенно понимают последовательность команд одной программы. Поток данных – это последовательность данных, обрабатываемых одной программой.

Согласно классификации Флинна имеется четыре больших класса ЭВМ:

1) *ОКОД (одиночный поток команд – одиночный поток данных)*, или SISD (Single Instruction – Single Data). Это последовательные ЭВМ, в которых выполняется единственная программа, т.е. имеется только один счетчик команд и одно арифметико-логическое устройство (АЛУ);

2) *ОКМД (одиночный поток команд – множественный поток данных)*, или SIMD (Single Instruction – Multiple Data). В таких ЭВМ выполняется единственная программа, но каждая ее команда обрабатывает много чисел. Это соответствует векторной форме параллелизма;

3) *МКОД (множественный поток команд – одиночный поток данных)*, или MISD (Multiple Instruction – Single Data). Подразумевается, что в данном классе несколько команд одновременно работает с одним элементом данных; однако эта позиция классификации Флинна на практике не нашла применения;

4) *МКМД (множественный поток команд – множественный поток данных)*, или MIMD (Multiple Instruction – Multiple Data). В таких ЭВМ одновременно и независимо друг от друга выполняется несколько программных ветвей, в определенные промежутки времени обменивающихся данными. Такие системы обычно называют многопроцессорными.

В класс МКМД входят машины двух типов: *с управлением от потока команд* (IF – Instruction Flow) и *управлением от потока данных* (DF – Data Flow).

Если в ЭВМ первого типа используется традиционное выполнение команд по ходу их расположения в программе, то применение ЭВМ второго типа предполагает активацию операторов по мере их текущей готовности.

Наиболее известными представителями векторных ЭВМ являются векторно-конвейерная ЭВМ CRAY и ЭВМ типа процессорной матрицы ILLIAC-4. Ограниченный вариант векторной архитектуры используется при разработке параллельных микропроцессоров. Впервые его применила корпорация Intel в микропроцессоре Pentium под названием *MMX (MultiMedia eXtension)* для вычислений с фиксированной точкой. Данный тип процессора, по классификации Флинна, относится к классу ОКМД (SIMD). Метод ОКМД позволяет по одной команде обрабатывать несколько пар операндов (векторная обработка). Сейчас многие производители микропроцессоров выпускают приборы с векторной архитектурой, причем в двух вариантах: как в виде сопроцессора на одном кристалле с универсальным процессором (например Intel в микропроцессоре Pentium), так и в виде специализированного микропроцессора для цифровой обработки сигналов.

Классификация Головкина [13]

В качестве основных признаков классификации, характеризующих организацию структуры и функционирования вычислительных систем с точки зрения, главным образом, параллельности работы, выберем следующие:

- 1) тип потока команд в центральной части вычислительной системы;
- 2) тип потока данных в центральной части вычислительной системы;
- 3) способ обработки данных в центральных устройствах обработки;
- 4) степень связанности компонентов вычислительной системы;
- 5) степень однородности основных компонентов вычислительной системы;
- 6) тип внутренних связей в вычислительной системе.

Эти шесть признаков определяют базовую схему классификации, содержащую семь уровней иерархии, в которой переход от i -го уровня к $i+1$ -му уровню определяется соответствующим i -м признаком (рис. 5.7).

В этой схеме используются следующие условные обозначения:

- а) ВС – вычислительные системы;
- б) ОК, МК – одиночный и множественный потоки команд соответственно;
- в) ОД, МД – одиночный и множественный потоки данных соответственно;
- г) С, Р – пословная и поразрядная обработка данных в центральных обрабатывающих устройствах соответственно;
- д) Нс, Вс – низкая и высокая степень связанности вычислительной системы соответственно;
- е) Ор, Нр – однородная и неоднородная вычислительная система соответственно;
- ж) Кн, Пм, Пр – системы со связями «канал – канал», через общую внешнюю память и непосредственно между процессорами соответственно;
- з) Ош, Мш, Пк – системы со связями через одну общую шину с разделением ее времени, со связями через множество шин при использовании многоходовых модулей оперативной памяти и с перекрестными связями при помощи матричного коммутатора соответственно.

Обозначения классов систем составляются из условных обозначений каждого узла схемы, начиная с узлов второго уровня, через которые нужно пройти по стрелкам от узла ВС до узла данного класса систем включительно, отделяя для удобства записи условные обозначения узлов первых трех уровней от условных обозначений узлов последних трех уровней наклонной чертой. Так,

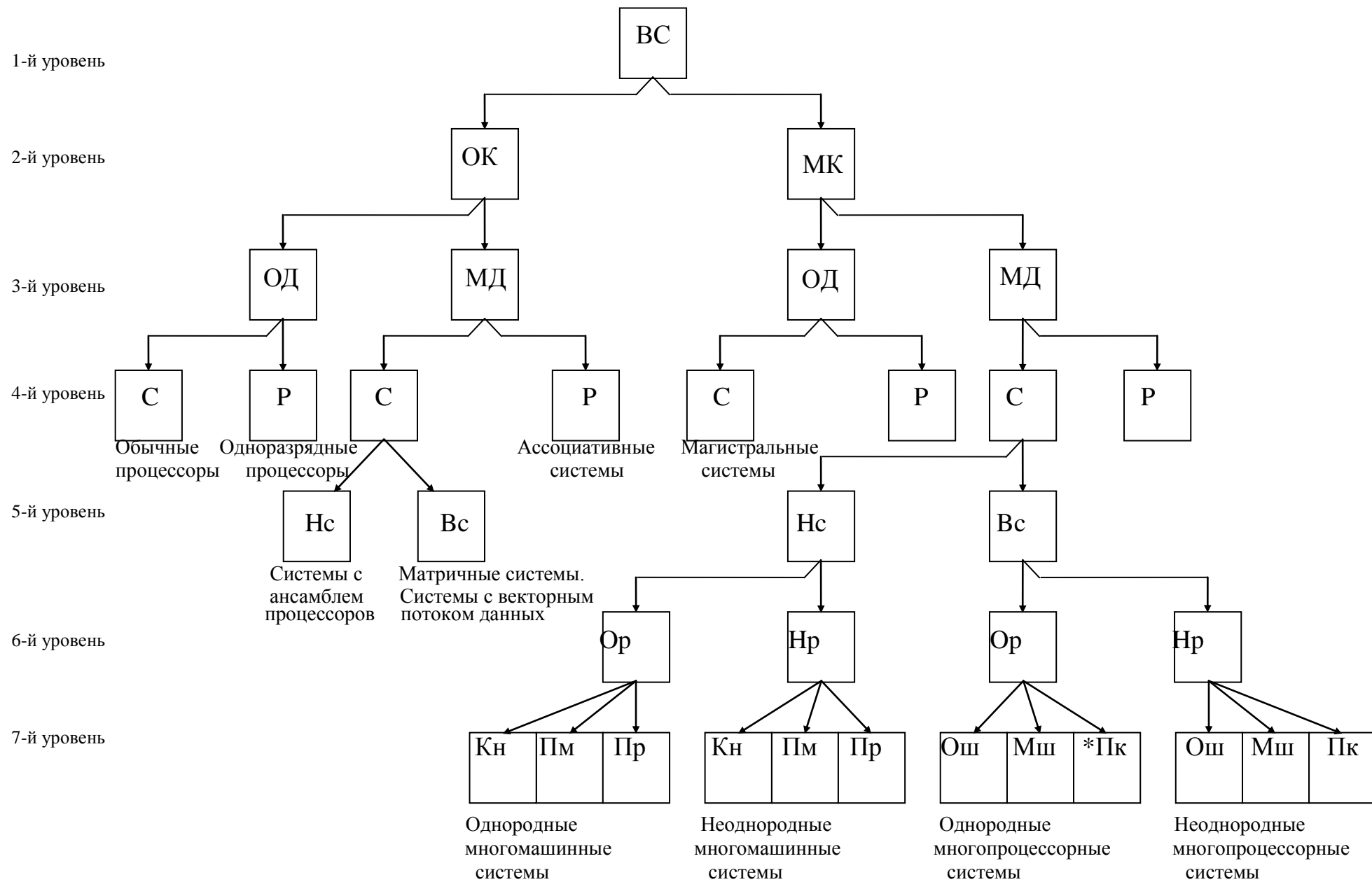


Рис. 5.7

например, класс вычислительных систем, помеченный на рис. 5.7 звездочкой (*), имеет обозначение МКМДС/ВсОрПк. Оно расшифровывается следующим образом: вычислительная система с множественным потоком команд и множественным потоком данных в центральной части, с пословной обработкой данных в центральных обрабатывающих устройствах, с высокой степенью связанности и однородной структурой, с перекрестными связями между процессорами и модулями памяти.

5.4.2. Многопроцессорные системы

К ним относятся: *массивно-параллельные системы (MPP), симметричные мультипроцессорные системы (SMP), системы с неоднородным доступом к памяти (NUMA), параллельные векторные системы (PVP), кластерные системы и комбинированные архитектуры.*

Основным параметром классификации параллельных компьютеров является наличие общей (как в SMP) или распределенной (как в MPP) памяти. Нечто среднее между SMP и MPP представляют собой NUMA-архитектуры, где память физически распределена, но логически общедоступна. Кластерные системы являются более дешевым вариантом MPP. При поддержке команд обработки векторных данных говорят о векторно-конвейерных процессорах, которые, в свою очередь, могут объединяться в PVP-системы с использованием общей или распределенной памяти. Все большую популярность приобретают идеи комбинирования различных архитектур в одной системе и построения неоднородных систем.

При организациях распределенных вычислений в глобальных сетях (Интернет) говорят о [мета-компьютерах](#), которые, строго говоря, не представляют из себя параллельных архитектур.

Рассмотрим кратко особенности многопроцессорных систем. В табл. 5.1–5.5 для каждого класса приводится следующая информация:

- краткое описание особенностей архитектуры;
- примеры конкретных компьютеров;
- перспективы масштабируемости;
- типичные особенности построения операционных систем;
- наиболее характерная модель программирования (хотя возможны и другие).

Массивно-параллельные системы (MPP)

Архитектура	<p>Система состоит из однородных <i>вычислительных узлов</i>, включающих:</p> <ul style="list-style-type: none"> - один или несколько центральных процессоров (обычно RISC); - локальную память (прямой доступ к памяти других узлов невозможен); - коммуникационный процессор или сетевой адаптер; - иногда – жесткие диски и/или другие устройства ввода-вывода. <p>К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)</p>
Примеры	IBM RS/6000 SP2 , Intel PARAGON/ASCI Red, SGI/CRAY T3E , Hitachi SR8000 , транспьютерные системы Parsytec
Масштабируемость	Общее число процессоров в реальных системах достигает нескольких тысяч (ASCI Red, Blue Mountain)
Операционная система	<p>Существуют два основных варианта:</p> <ol style="list-style-type: none"> 1. Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающих только работу расположенной в нем ветви параллельного приложения. Пример: Cray T3E. 2. На каждом узле работает полноценная UNIX-подобная ОС (вариант, близкий к кластерному подходу). Пример: IBM RS/6000 SP + ОС AIX, устанавливаемая отдельно на каждом узле
Модель программирования	Программирование в рамках модели передачи сообщений (MPI , PVM , BSPlib)

Симметричные мультимикропроцессорные системы (SMP)

Архитектура	Система состоит из нескольких однородных процессоров и массива общей памяти (обычно из нескольких независимых блоков). Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью. Процессоры подключены к памяти либо с помощью общей шины (базовые 2-, 4- процессорные SMP-серверы), либо с помощью crossbar-коммутатора (HP 9000). Аппаратно поддерживается когерентность кэшей
Примеры	HP 9000 V-class , N-class; SMP-сервера и рабочие станции на базе процессоров Intel (IBM, HP, Compaq, Dell, ALR, Unisys, DG, Fujitsu и др.)
Масштабируемость	Наличие общей памяти сильно упрощает взаимодействие процессоров между собой, однако накладывает сильные ограничения на их число – не более 32 в реальных системах. Для построения масштабируемых систем на базе SMP используются кластерные или NUMA -архитектуры
Операционная система	Вся система работает под управлением единой ОС (обычно UNIX-подобной, но для Intel-платформ поддерживается Windows NT). ОС автоматически (в процессе работы) распределяет процессы/нити по процессорам (scheduling), но иногда возможна и явная привязка
Модель программирования	Программирование в модели общей памяти. (POSIX threads, OpenMP). Для SMP-систем существуют сравнительно эффективные средства автоматического распараллеливания

Системы с неоднородным доступом к памяти (NUMA)

Архитектура	<p>Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти. Модули объединены с помощью высокоскоростного коммутатора. Поддерживается единое адресное пространство, аппаратно поддерживается доступ к удаленной памяти, т.е. к памяти других модулей. При этом доступ к локальной памяти в несколько раз быстрее, чем к удаленной.</p> <p>В случае, если аппаратно поддерживается когерентность кэш-памяти во всей системе (обычно это так), говорят об архитектуре <i>cc-NUMA</i> (cache-coherent NUMA)</p>
Примеры	<p>HP HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600</p>
Масштабируемость	<p>Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэш-памяти и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент максимальное число процессоров в NUMA-системах составляет 256 (Origin2000)</p>
Операционная система	<p>Обычно вся система работает под управлением единой ОС, как в SMP. Но возможны также варианты динамического «подразделения» системы, когда отдельные «разделы» системы работают под управлением разных ОС (например, Windows NT и UNIX в NUMA-Q 2000)</p>
Модель программирования	<p>Аналогично SMP</p>

Таблица 5.4

Параллельные векторные системы (PVP)

Архитектура	<p>Основным признаком PVP-систем является наличие специальных векторно-конвейерных процессоров, в которых предусмотрены команды однотипной обработки векторов независимых данных, эффективно выполняющиеся на конвейерных функциональных устройствах.</p> <p>Как правило, несколько таких процессоров (1 – 16) работают одновременно с общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP)</p>
Примеры	NEC SX-4/ SX-5 , линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/ T90 , CRAY SV1 , серия Fujitsu VPP
Модель программирования	Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением)

Таблица 5.5

Кластерные системы

Архитектура	<p>Набор рабочих станций (или даже ПК) общего назначения, используется в качестве дешевого варианта массивно-параллельного компьютера. Для связи узлов используется одна из стандартных сетевых технологий (Fast/Gigabit Ethernet, Myrinet и др.) на базе шинной архитектуры, или коммутатора.</p> <p>При объединении в кластер компьютеров разной мощности или разной архитектуры говорят о <i>гетерогенных</i> (неоднородных) кластерах.</p> <p>Узлы кластера могут одновременно использоваться в качестве пользовательских рабочих станций. В случае, когда это не нужно, узлы могут быть существенно облегчены и/или установлены в стойку</p>
Примеры	NT-кластер в NCSA, Beowulf -кластеры, «СКИФ»

Операционная система	Используются стандартные для рабочих станций ОС, чаще всего свободно распространяемые – Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и распределения нагрузки
Модель программирования	Программирование, как правило, в рамках модели передачи сообщений (чаще всего – MPI). Дешевизна подобных систем оборачивается большими накладными расходами на взаимодействие параллельных процессов между собой, что сильно сужает потенциальный класс решаемых задач

Примечание. Информация о кластерах семейства «СКИФ» находится на сайтах: <http://www.niievvm.by/super.htm> и <http://www.skif.bas-net.by/>.

Комбинированные архитектуры

Возникли как попытки объединения архитектур с целью использовать их преимущества, сводя к минимуму их недостатки.

В частности, есть MPP-системы (например система IBM RS/6000 SP), узлы (гиперузлы) которых представляют собой симметричные мультипроцессоры (вычислительные системы SMP-архитектуры).

Другим вариантом комбинированной архитектуры является архитектура NUMA (например, системы Exemplar SPP1200 , Exemplar SPP2000 , Origin 2000 , Sequent NUMA-Q , RM600E). Она представляет собой попытку распространить на архитектуру MPP принципы архитектуры SMP с целью преодоления (точнее минимизации) ограничения на рост масштабируемости последней. Данный вариант комбинированной архитектуры отличается от предыдущего тем, что вся память (т.е. локальная память всех гиперузлов, являющихся SMP-мультипроцессорами) является общей, хотя и физически распределенной по гиперузлам. За счет введения дополнительных аппаратных (и программных) средств (каналов связи, кэшей) существенно (по сравнению с традиционной MPP-архитектурой) снижается (хотя и не исчезает) разница между временем доступа к локальной памяти и к памяти, расположенной на другом гиперузле. Это позволяет считать архитектуру NUMA расширением архитектуры SMP с существенно более высоким уровнем масштабирования.

Применительно к проблематике машин переработки знаний указанные вычислительные системы используются в качестве аппаратной основы при реализации, например, СУБД, экспертных систем. При этом приложения (например

СУБД) не только исполняются на параллельной технике, но и изначально ориентируются (разрабатываются) на распределенную параллельную реализацию. Отсутствие такой изначальной ориентации, очевидно, не является критичным при использовании вычислительных систем класса SMP с небольшим (не более десятка) количеством процессоров. Однако при использовании MPP и кластерных архитектур с большим количеством процессоров это существенно снижает эффективность их применения, что, очевидно, определяет необходимость учета распределенной параллельной обработки знаний, реализуемых на их основе.

5.4.3. Графодинамические параллельные асинхронные машины

Для современных систем, ориентированных на нечисловую обработку больших объемов сложноструктурированной информации, широкое использование мультимедиа-технологий, полиэкранных адаптивных интерфейсов и многоагентного взаимодействия в глобальных сетях, фон-неймановская архитектура не обеспечивает достаточной технологической гибкости, согласованности данных и реакции в реальном масштабе времени. Основными принципами организации современных архитектур вычислительных систем являются: открытость, параллельность, асинхронность, ассоциативность доступа (что обеспечивает поддержку асинхронизма), структурная перестраиваемость памяти.

Однако, несмотря на интенсивность проводимых исследований и разнообразие подходов к реализации аппаратной поддержки решения информационно-логических задач, большинство существующих вычислительных систем с нетрадиционной архитектурой весьма далеки от эффективного практического применения. Среди основных причин такого положения можно указать[14]:

1) отсутствие математически строгих моделей вычислений, лежащих в основе их архитектуры. Определенным прорывом в этом плане стала концепция транспьютера, однако некоторые свойства транспьютера и той модели, которую он поддерживает (примитивизм межкомпонентных коммуникаций, ориентация на статическое планирование загрузки и разделения функций и т.д.), смещают мультитранспьютерные системы в область специализированных ЭВМ;

2) отсутствие комплексного подхода и адекватных решаемому классу задач инструментально-технологических средств при разработке системного (и, как следствие, прикладного) программного обеспечения. Программное обеспечение часто либо разрабатывается в отрыве от аппаратуры (так, первые практически пригодные операционные системы для транспьютеров появились лишь спустя

4–5 лет после их создания!), либо, наоборот, жестко ориентировано на ее конкретную реализацию (многие Lisp-машины и Prolog-машины, машины баз данных). Инструментальные средства не поддерживают современные методологии разработки, что делает невозможным создание больших прикладных систем;

3) ориентация на узкий класс моделей представления/обработки информации и организации вычислительного процесса (Prolog, реляционные СУБД и т.д.). До настоящего времени в параллельных интеллектуальных системах коллективное поведение, использование знаний и ресурсов (вычислительных, коммуникационных и др.) не поддерживается на уровне базовых механизмов модели. Это приводит к неэффективности при разделении функций в динамически изменяемых коллективных структурах, при использовании различных по характеристикам коммуникаций, при реакции на отказы и т.п.

Целостный анализ совокупности вышеописанных тенденций показывает, что использование параллельных технологий коллективной обработки и развитых форм ассоциативного доступа к информации является необходимым условием решения проблем проектирования современных систем обработки информации.

Наиболее актуальным решение указанных проблем становится при проектировании современных интеллектуальных систем, в которых осуществляется обработка сложноструктурированных баз знаний и интеграция различных моделей решения интеллектуальных задач.

В основе одного из подходов к созданию компьютерных архитектур и компьютерных систем, ориентированных на обработку знаний, лежит графодинамическая парадигма обработки информации, описанная в [15]. Абстрактные машины обработки информации, построенные в соответствии с этой парадигмой, названы графодинамическими ассоциативными машинами, принципы организации которых сводятся к следующему [16]:

1) память является структурно перестраиваемой (графодинамической), т.е. информация кодируется в виде графовой структуры специального вида, а обработка информации сводится не только к изменению состояния (содержимого) элементов (элементарных областей) памяти, но и к изменению конфигурации связей между ними;

2) операции представляют собой демонические абстрактные процессы, инициируемые появлением в памяти структуры заданного вида (для каждой операции – своя инициирующая структура);

3) операции взаимодействуют между собой только через память. Следовательно, память абстрактной машины должна полностью отражать всё текущее состояние машины, т.е. отсутствует какое-либо непосредственное взаимодействие между операциями;

4) операции выполняются асинхронно и могут выполняться параллельно;

5) разрешение противоречий между конкурирующими процессами осуществляется с помощью блокировок элементов памяти. Каждая блокировка однозначно связана с процессом, который её поставил, и является для других процессов запретом на те или иные действия с заблокированными элементами.

Преимущество использования графодинамических параллельных асинхронных абстрактных машин в качестве инструмента для создания интеллектуальных систем нового поколения (систем, основанных на знаниях) обусловлено следующими обстоятельствами:

а) в них принципиально проще реализуется ассоциативный метод доступа к перерабатываемой информации;

б) существенно проще поддерживается открытый характер как самих машин, так и реализуемых на них формальных моделей. В частности, в графодинамических машинах существенно проще реализуются семиотические модели;

г) графодинамические параллельные асинхронные машины являются удобной основой для интеграции самых различных моделей обработки информации, ибо на базе графодинамических параллельных асинхронных абстрактных машин легко реализуются не только графодинамические, но и символьные формальные модели, не только параллельные, но и последовательные модели, не только асинхронные, но и синхронные модели, не только различные модели логического вывода, но и всевозможные модели реализации хранимых процедурных программ.

Остальные достоинства графодинамических параллельных асинхронных машин обусловлены достоинствами графовых информационных конструкций и графовых языков[15].

Приведенные обстоятельства позволяют сделать вывод, что графодинамические параллельные асинхронные абстрактные машины являются наиболее перспективной основой для реализации интеллектуальных систем нового поколения. При этом особое внимание следует уделить трем видам графодинамических параллельных асинхронных абстрактных машин, определяющим три уровня параллельных интеллектуальных систем:

– графодинамическим параллельным асинхронным абстрактным машинам логического вывода. Перерабатываемая информация (перерабатываемые знания) в таких машинах представляется в виде семантических конструкций (семантических сетей) на некотором графовом семантическом языке представления знаний. А операциями в этих машинах являются операции логического вывода, поддерживающие самые различные стратегии решения задач в базах знаний;

– графодинамическим параллельным асинхронным абстрактным машинам реализации хранимых процедурных программ, специально ориентированным на интерпретацию графодинамических параллельных асинхронных абстрактных машин логического вывода;

– графодинамическим распределенным асинхронным абстрактным машинам реализации хранимых процедурных программ, определяющим крупнозернистую архитектуру графодинамического параллельного асинхронного компьютера, ориентированного на поддержку всевозможных графодинамических параллельных асинхронных моделей обработки информации. Операции абстрактных машин данного уровня в отличие от абстрактных машин второго уровня имеют ограниченную область действия, т.е. имеют доступ не ко всей памяти, а только к какой-то ее области. Следовательно, вся память абстрактной машины разбивается на области, каждой из которых ставится в соответствие набор работающих над ней операций. При этом наборы операций, работающих над разными областями памяти, абсолютно аналогичны и отличаются только областью действия операторов. Таким образом, графодинамическая распределенная асинхронная абстрактная машина реализации хранимых процедурных программ представляет собой коллектив графодинамических абстрактных машин (которые будем называть модулями распределенной машины), взаимодействующих между собой путем обмена сообщениями. Особенности распределенных абстрактных информационных машин являются наличие: а) в каждом модуле входных и выходных буферов для приема и передачи сообщений; б) специальных операций пересылки сообщений из выходного буфера модуля-передатчика во входной буфер модуля-приемника (модуля-адресата); в) специальных операций обработки принятых сообщений.

5.4.4. Семейство суперкомпьютеров «СКИФ»

5.4.4.1. Общие сведения о семействе «СКИФ»

Семейство суперкомпьютеров «СКИФ» разрабатывалось по программе Союзного государства Беларуси и России «Разработка и освоение в серийном производстве семейства моделей высокопроизводительных вычислительных систем с параллельной архитектурой (суперкомпьютеров) и создание прикладных программно-аппаратных комплексов на их основе» (программа «СКИФ»).

Основная цель реализации программы – это реальное освоение высоких суперкомпьютерных технологий в России и Беларуси. Поэтому большое внимание уделяется подготовке и переподготовке кадров для работы с суперкомпьютерными технологиями, созданию пилотных прикладных комплексов и организации промышленного выпуска суперкомпьютеров.

Основополагающими архитектурными принципами создания суперкомпьютерных конфигураций «СКИФ» являются:

- базовая кластерная архитектура;
- иерархические кластерные конфигурации (метакластеры);
- универсальная двухуровневая архитектура.

Базовая кластерная архитектура. Концепция создания моделей семейства суперкомпьютеров «СКИФ» базировалась на масштабируемой кластерной архитектуре, реализуемой на *классических кластерах из вычислительных узлов на основе компонентов широкого применения* (стандартных микропроцессоров, модулей памяти, жестких дисков и материнских плат, в том числе с поддержкой симметричных мультипроцессорных систем (SMP)).

Кластерный архитектурный уровень – это тесно связанная сеть (кластер) вычислительных узлов, работающих под управлением ОС Linux – одного из клонов широко используемой многопользовательской универсальной операционной системы UNIX. Для организации параллельного выполнения прикладных задач на данном уровне используются:

- оригинальная система поддержки параллельных вычислений – *T-система*, реализующая автоматическое динамическое распараллеливание программ;

- классические системы поддержки параллельных вычислений, обеспечивающие эффективное распараллеливание прикладных задач различных классов (как правило, задач с явным параллелизмом): MPI, PVM, Norma, DVM и др. В семействе суперкомпьютеров «СКИФ» в качестве базовой классической системы поддержки параллельных вычислений выбран MPI, что не исключает использования других средств.

В части *T-системы* обеспечивается автоматическое динамическое распараллеливание программ, что освобождает программиста от большинства трудоемких аспектов разработки параллельных программ, свойственных различным системам ручного статического распараллеливания:

- обнаружение готовых к выполнению фрагментов задачи (процессов);
- их распределение по процессорам;
- их синхронизация по данным.

Все эти (и другие) операции выполняются в *T-системе* автоматически и в динамике (во время выполнения задачи). Тем самым при *более низких затратах на разработку параллельных программ обеспечивается более высокая их надежность*.

По сравнению с использованием распараллеливающих компиляторов, *T-система* обеспечивает более глубокий уровень параллелизма во время выполнения программы и более полное использование вычислительных ресурсов мультипроцессоров. Это связано с принципиальными алгоритмическими трудностями (алгоритмически неразрешимыми проблемами), не позволяющими во время компиляции (в статике) выполнить полный точный анализ и предсказать последующее поведение программы во время счета.

Кроме указанных выше принципиальных преимуществ *T-системы* перед известными сегодня методами организации параллельного счета, в реализации *T-*

системы имеется ряд технологических находок, не имеющих аналогов в мире, в частности:

- реализация понятия «неготовое значение» и поддержка корректного выполнения некоторых операций над неготовыми значениями. Тем самым поддерживается возможность выполнения счета в некотором процессе-потребителе в условиях, когда часть из обрабатываемых им значений еще не готова, т.е. не вычислена в соответствующем процессе-поставщике. Данное техническое решение обеспечивает обнаружение более глубокого параллелизма в программе;

- оригинальный алгоритм динамического автоматического распределения процессов по процессорам. Данный алгоритм учитывает особенности неоднородных распределенных вычислительных сетей. По сравнению с известными алгоритмами динамического автоматического распределения процессов по процессорам (например с диффузионным алгоритмом и его модификациями) алгоритм Т-системы имеет существенно более низкий трафик межпроцессорных передач. Тем самым Т-система обеспечивает снижение накладных расходов на организацию параллельного счета и предъявляет менее жесткие требования к пропускной способности аппаратуры объединения процессорных элементов в кластер. В качестве языка программирования Т-системы используется расширение языка СИ новыми Т-конструкциями.

Для организации взаимодействия вычислительных узлов суперкомпьютера в его составе используются различные сетевые (аппаратные и программные) средства, в совокупности образующие две системы передачи данных.

Системная сеть кластера (CC) или System Area Network (SAN) объединяет узлы кластерного уровня в кластер. Данная сеть поддерживает масштабируемость кластерного уровня суперкомпьютера, а также пересылку и когерентность данных во всех вычислительных узлах кластерного уровня суперкомпьютера. Системная сеть кластера строится на основе специализированных высокоскоростных линков

типа SCI, Myrinet, cLan, Infiniband и др., предназначенных для эффективной поддержки кластерных вычислений и соответствующей программной поддержки на уровне ОС Linux и систем организации параллельных вычислений (Т-система, MPI).

Вспомогательная сеть суперкомпьютера (ВС) с протоколом TCP/IP объединяет узлы кластерного уровня в обычную (TCP/IP) локальную сеть (*TCP/IP LAN*). Данная сеть может быть реализована на основе широко используемых сетевых технологий класса Fast Ethernet, Gigabit Ethernet, ATM и др. Данная сеть предназначена для управления системой, подключения внешней дисковой памяти и рабочих мест пользователей, интеграции суперкомпьютера в локальную сеть предприятия и/или в глобальные сети. Кроме того, данный уровень может быть использован и системой организации параллельных кластерных вычислений (Т-система, MPI) для вспомогательных целей (основные потоки информации, возникающие при организации параллельных кластерных вычислений, передаются через системную сеть кластера).

Примечание. В некоторых случаях аппаратура системной сети, например Myrinet, позволяет без ущерба для реализации кластерных вычислений поддерживать на этой же аппаратуре реализацию сети TCP/IP. В этих случаях аппаратные части обеих сетей (SAN и TCP/IP LAN) могут быть совмещены.

Кластерные конфигурации на базе только вспомогательной сети TCP/IP без использования дорогостоящих специализированных высокоскоростных линков класса SCI могут быть реализованы в рамках семейства «СКИФ» в виде самостоятельных изделий (*TCP/IP кластеры*). Программное обеспечение таких кластеров – ОС Linux, Т-система и соответствующая реализация MPI. Реализация сравнительно недорогих TCP/IP кластеров на базе «масштабирования вниз» архитектурных решений «СКИФ» (дополнительный или вторичный эффект) существенно расширяет область применения результатов реализации программы .

Иерархические кластерные конфигурации (метакластеры). Отдельные кластеры могут быть объединены в единую кластерную конфигурацию – кластер высшего уровня, или *метакластер (Metacluster)*. Метакластерный принцип позволяет создавать распределенные метакластерные конфигурации на базе локальных или глобальных сетей передачи данных. При этом, естественно, уменьшается степень связности подкластеров метакластерной конфигурации.

Системное программное обеспечение метакластера обеспечивает возможность реализации *гетерогенных систем*, включающих подкластеры различной архитектуры на различных программно-аппаратных платформах.

Одним из перспективных программных продуктов, с использованием которого возможна реализация метакластерных конфигураций (по крайней мере простой топологии типа point-to-point) на подкластерах с различными программно-аппаратными платформами, является IMPI (Interoperable Message Passing Interface). IMPI реализует стандартизованный протокол, обеспечивающий взаимодействие различных реализаций MPI. Это позволяет выполнять общую задачу на различной аппаратуре с использованием настраиваемых поставщиком (vendor-tuned) различных реализаций MPI на каждом узле кластерной конфигурации соответствующего уровня иерархии. Такая возможность удобна в случаях, когда объем вычислений задачи слишком велик для одной системы или когда разные части задачи оптимальнее выполняются на разных реализациях MPI.

IMPI определяет только протоколы, необходимые для взаимодействия различных реализаций MPI, а также может использовать собственные высокопроизводительные протоколы этих реализаций. Существуют свободно распространяемые (открытые) версии IMPI, например на базе LAM/MPI.

Преимущества и цели реализации иерархической (метакластерной) архитектуры.

Реализация архитектурных принципов иерархической организации суперкомпьютерных метакластерных конфигураций позволит решить важнейшие для создания моделей семейства суперкомпьютеров «СКИФ» задачи:

- *обеспечение реально достижимой и экономически эффективной масштабируемости архитектурных решений.* Это особенно важно для решения ключевой задачи программы: создание моделей суперкомпьютеров, позволяющих перекрыть широкий диапазон производительности и областей применения – от моделей суперкомпьютеров среднего класса (10-100 Гфлопс) до вычислительных систем с массовым параллелизмом сверхвысокой производительности (триллионы операций в секунду);

- *создание единого информационного пространства участников программы, а в перспективе – объединение научных сетей России и Беларуси, на базе распределенных сетевых суперкомпьютерных метакластерных конфигураций;*

- *обеспечение живучести суперкомпьютерных систем;*

- *объединение суперкомпьютерных конфигураций с разными архитектурными и программно-аппаратными платформами (гибридная метакластерная архитектура) в единую метакластерную суперкомпьютерную систему;*

- *создание глобальных сетевых конфигураций с гибридной метакластерной архитектурой терафлопового диапазона.* Такие метакластеры могут быть созданы путем объединения кластеров «СКИФ» с другими существующими в Республике Беларусь и Российской Федерации кластерными конфигурациями (например, в МГУ, Межведомственном суперкомпьютерном центре Минпромнауки Российской Федерации и РАН и др.).

Для оптимизации организации на суперкомпьютерах «СКИФ» параллельного счета задач как с крупноблочным (явным статическим или скрытым динамическим) параллелизмом, так и с конвейерным или мелкозернистым явным параллелизмом, с большими потоками информации, требующими обработки в реальном режиме времени, концепция предусматривает возможность реализации *универсальной двухуровневой архитектуры* суперкомпьютеров (рис. 5.8):

- 1-й уровень – базовый (кластерный) архитектурный уровень;
- 2-й уровень – потоковый архитектурный уровень, реализующий модель потоковых вычислений (data-flow).

Концепция предусматривает реализацию потокового архитектурного уровня как на базе однородной вычислительной среды (ОВС) с использованием оригинальных СБИС ОВС, разрабатываемых в рамках Программы, так и на базе других (альтернативных) структурных и технических решений (например, на базе нейроструктур, программируемых логических схем (FPGA типа XILINX) и др.). По сути вычислительные модули потокового уровня являются сопроцессорами вычислительных ресурсов кластерной конфигурации.



Рис. 5.8

Предпосылкой объединения двух программно-аппаратных решений (кластерного и потокового) для организации параллельной обработки в рамках одной вычислительной системы является то, что эти два подхода, как уже отмечалось, своими сильными сторонами компенсируют недостатки друг друга. Тем самым, в общем случае, каждая прикладная проблема может быть разбита на фрагменты:

- со сложной логикой вычисления, с крупноблочным (явным статическим или скрытым динамическим) параллелизмом, эффективно реализуемые на кластерном уровне с использованием Т-системы и других (классических) систем поддержки параллельных вычислений;
- с простой логикой вычисления, с конвейерным или мелкозернистым явным параллелизмом, с большими потоками информации, требующими обработки в реальном режиме времени, эффективно реализуемые на потоковом уровне.

Рассмотренные архитектурные принципы создания суперкомпьютеров семейства «СКИФ» позволяют эффективно реализовать любые виды параллелизма. Архитектура является открытой и масштабируемой, т.е. не накладывает жестких ограничений к программно-аппаратной платформе узлов кластера, топологии вычислительной сети, конфигурации и диапазону производительности суперкомпьютеров. Вычислительные системы, создаваемые на базе основополагающих концептуальных архитектурных принципов, могут оптимально решать как классические вычислительные задачи математической физики и линейной алгебры, так и специализированные задачи обработки сигналов, моделирования, задачи управления сложными системами в реальном времени и другие приложения.

5.4.4.2. Результаты выполнения программы «СКИФ»

В рамках программы «СКИФ» в течение 2000–2002 гг. разработано семейство базовых конструктивных модулей, предназначенных для построения моделей первого ряда семейства суперкомпьютеров кластерного типа.

Семейство базовых конструктивных модулей (БКМ) включает:

- базовые вычислительные узлы (узлы кластера) на платформе ПЭВМ промышленного типа габарита 3U, 2U и 1U (U=44,45 мм);
- стойки (монтажные шкафы) стандарта 19" высотой 15U, 18U, 24U, предназначенные для размещения вычислительных узлов, коммуникационных средств управляющей и системной сети, источников бесперебойного питания, средств системы охлаждения (вентиляции).

С применением приведенных выше базовых модулей была разработана конструкторская документация на ряд вариантов исполнений кластеров, изготовлены экспериментальные и опытные образцы, проведены приемочные испытания. Разработанные образцы кластеров в настоящее время используются для отработки системного программного обеспечения и прикладных программ.

Достигнутые в 2000–2002 гг. по программе «СКИФ» результаты сделали реальными поставки (начиная с 2003 г.) прикладных комплектов на базе кластерных конфигураций «СКИФ» в широком диапазоне производительности. Такие конфигурации являются высокопроизводительными, масштабируемыми, надежными и доступными Linux-серверами. Отметим, что разработанные по программе «СКИФ» суперсерверы по своим характеристикам сопоставимы с зарубежными системами аналогичного типа и конкурентоспособны по сравнению с ними.

Ключевой работой 2003 г. стало создание образца суперкомпьютерной конфигурации кластерного уровня с пиковой производительностью ~ 700 миллиардов команд в секунду («К-500»), который был предназначен для использования в качестве инструментальной среды для отработки основных принципов создания кластеров терафлопного диапазона (Т-кластеров).

Основные характеристики кластера «К-500»:

- применение современных вычислительных платформ с реализацией технологий Hyper-Threading на базе процессоров Intel Xeon;
- архитектура системной сети – SCI-3D (трехмерный тор);
- пиковая производительность вычислительного узла 10 – 12 Гфлопс, кластера – 700 Гфлопс;
- базовый типоразмер узлов кластера – 1U;
- количество вычислительных узлов – 64.

Для построения кластера «К-500» были разработаны и изготовлены новые модификации базовых конструктивных модулей высотой 22U, в которых установлены вычислительные узлы, коммутаторы управляющей сети типа Gigabit Ethernet, система охлаждения, источники бесперебойного питания и другие технические средства.

В кластере «К-500» реализована принципиально отличающаяся от разработанных в 2000 – 2002 гг. образцах кластеров система охлаждения как по конструктивному размещению, так и по организации управления системой охлаждения в зависимости от изменения температуры внутри конструктивных модулей.

Еще одной особенностью кластера «К-500» является система удаленного администрирования, поддержки и обслуживания, обеспечивающая удаленное и автоматическое выполнение операций:

- включение/отключение питания вычислительных узлов;
- аппаратный сброс узла;
- **взаимодействие с узлом в консольном режиме (управление режимом загрузки, работа с отладчиком ядра операционной системы, сбор сообщений из ядра системы и др.).**

Создание кластера «К-500» явилось качественно новым результатом в рамках программы «СКИФ», который позволил вплотную приблизиться к терафлопному диапазону производительности и подготовить задел к созданию в 2004 г. суперкомпьютера с массовым параллелизмом сверхвысокой

производительности (триллионы операций в секунду). В ноябре 2003 г. кластер «К-500» был включен в список 500 самых производительных компьютерных систем в мире.

В 2004 г. программа «СКИФ» была завершена разработкой суперкомпьютера терафлопного диапазона «СКИФ К-1000».

Основные характеристики кластера «К-1000»:

- пиковая производительность – 2,5 Тфлопс;
- LinPack – производительность – 2,0 Тфлопс;
- количество вычислительных узлов – 288;
- количество микропроцессоров – 576 (типа AMD Opteron 248 (2,2 ГГц));
- емкость оперативной памяти – 1152 Гбайт;
- емкость дисковой памяти – 23040 Гбайт;
- тип системной сети – Infiniband (IB) – 4х;
- тип управляющей (вспомогательной) сети – Gigabit Ethernet;
- сервисная сеть – СКИФ-ServNet v.2.0;
- конструктив узла (форм- фактор) - 1U;
- монтажные стойки – 8 х (2 по 22U);
- потребляемая мощность – до 120 кВА;
- мощность воздушного охлаждения кластера – 22400 м³/ч.

Для кластера «К-1000» была произведена доработка конструктивных модулей высотой 22U, разработанных для кластера «К-500», в части усиления системы воздушного охлаждения и обеспечения более удобного монтажа (общее количество кабелей системной и управляющей сетей в кластере «К-1000» – 900, суммарная длина связей ~ 2500 м).

Создание кластера «К-1000» позволило выйти на собственный путь развития высокопроизводительной вычислительной техники, уровень которой соответствует прогнозируемым требованиям со стороны широкой категории пользователей. Полученные результаты будут способствовать форсированному технологическому перевооружению ключевых отраслей промышленности стран–

участниц союзного государства, их реформированию с целью достижения мирового уровня качества продукции на базе новейших наукоемких информационных технологий и суперкомпьютерных конфигураций «СКИФ».

В ноябре 2004 г. кластер «К-1000» был включен в список ТОП-500 (98-е место).

За последние годы в Беларуси значительно расширился фронт работ по практическому использованию суперкомпьютерных систем, созданных по программе «СКИФ». В частности, проводились и проводятся работы:

- по региональному прогнозу погоды на 48 часов (в результате счета задачи на 32-процессорной установке «СКИФ» прогностические значения приземного давления, составляющих скорости ветра и осадков получены в 12 раз быстрее, чем при расчетах на обычных вычислительных средствах Гидрометеоцентра);

- по созданию в 2005 г. сквозной компьютерной технологии проектирования, испытаний и технологической подготовки турбокомпрессоров для наддува дизельных двигателей Минского моторного завода (компьютерная технология базируется на платформе «СКИФ» и программных системах LS-DYNA и Star-CD);

- по использованию суперкомпьютеров «СКИФ» для расчетов и моделирования остовов перспективных универсальных тракторов «Беларусь», которые принципиально не могут быть рассчитаны на традиционных средствах вычислительной техники. Получены положительные решения, ведется работа по совершенствованию методик;

- по расчету динамических характеристик почвообрабатывающих агрегатов с использованием программного обеспечения конечно-элементных расчетов, развернутого на кластерах «СКИФ»;

- по расчетам на «СКИФ» несущих конструкций карьерных самосвалов БелАЗ и шахтных крепей;

- по проектированию и моделированию в среде «СКИФ» карданных валов;

- по моделированию на «СКИФ» в среде LS-DYNA процессов лазерного спекания порошковых материалов для технологий быстрого прототипирования;
- по суперкомпьютерному моделированию столкновений транспортных средств с неподвижными препятствиями;
- по созданию медико-технологического комплекса на базе суперкомпьютера серии «СКИФ» для исследования микроциркулярного русла методом биомикроскопии.

В НАН Беларуси совместно с заинтересованными предприятиями республики завершены работы по созданию суперкомпьютерного центра коллективного пользования для проектирования и анализа объектов новой техники.

5.4.4.3. Особенности модуля потокового уровня

Модуль потокового уровня, или базовый вычислительный модуль однородной вычислительной среды (БВМ ОВС), предназначен для потоковой обработки информации с помощью матрицы процессорных элементов. Потоковая обработка информации построена на принципах параллельной конвейерной обработки.

Матрица процессорных элементов является основным вычислительным устройством БВМ ОВС. Она состоит из нескольких плат с установленными на них БИС ОВС, соединенных шлейфами. Управление матрицей процессорных элементов осуществляется с помощью платы управления, устанавливаемой в PCI-слот вычислительного узла. Входные/выходные информационные потоки и сигналы управления поступают на матричное вычислительное устройство с платы управления. Перед обработкой данных на матричном вычислительном устройстве производится инициализация и загрузка программ в процессорные элементы. Программы, выполняемые на матрице, находятся в памяти платы управления. В памяти платы управления также находятся данные для обработки и результаты вычислений.

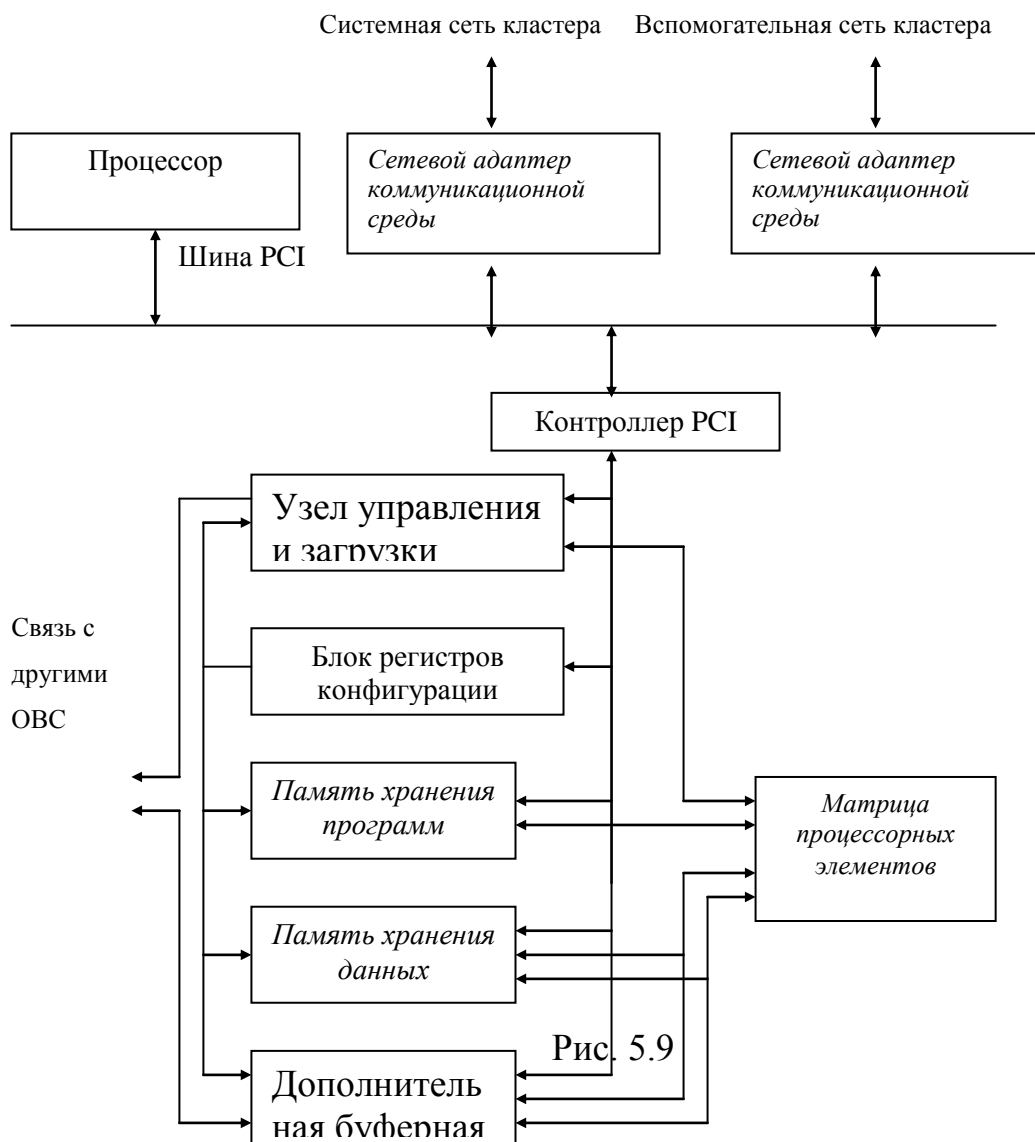
В настоящее время каждый БИС процессорных элементов, разработанных по программе «СКИФ», содержит 25 процессорных элементов. Все БИС, устанавливаемые на плате, имеют единую синхронизацию и организованы в

двухмерную структуру. Известно, что такие структуры отличаются линейной масштабируемостью. Линейная масштабируемость означает, что наращивание количества процессорных элементов влечет за собой линейный рост вычислительной производительности. Поэтому одним из способов увеличения производительности является увеличение количества процессорных элементов в БИС и увеличения частоты синхронизации их работы.

Другим способом повышения производительности БВМ ОВС является разработка плат управления, устанавливаемых в шины PCI X, PCI X-Express. Независимо от шины подключения основные блоки и узлы и связи между ними практически остаются неизменными за исключением контроллера шины.

На процессорном элементе БИС реализовано 49 команд. С помощью этих команд можно реализовать практически любой алгоритм. Поэтому БВМ ОВС можно настраивать на решение разных задач потоковой обработки.

Структурная схема БВМ ОВС приведена на рис. 5.9.



Плата управления ОВС и сетевые адаптеры подключаются через системную шину PCI. В состав платы управления входят узел управления и загрузки матрицы, память хранения данных, память хранения программ, блок регистров хранения конфигурации платы управления, контроллер шины PCI. В блок регистров хранения конфигурации платы управления управляющая программа записывает тип обрабатываемых матрицей данных, начальные адреса расположения данных в памяти хранения. Обработка данных на матрице процессорных элементов является одним из лучших решений для обработки сигналов в реальном масштабе времени, решения задач линейной алгебры и моделирования сложных процессов. Поэтому в зависимости от типов решаемых задач в вычислительную систему могут входить только БВМ ОВС или БВМ ОВС вместе с вычислительными узлами последовательной обработки на основе микропроцессоров Intel, AMD и т.д. Причем в зависимости от производительности процессоров серверной платформы БВМ ОВС и требований выполняемой задачи может устанавливаться одна или несколько плат управления или использоваться несколько БВМ ОВС. В этом случае для повышения производительности и эффективности выполнения программ целесообразно организовать специальную сеть передачи данных между платами управления отдельных ОВС. Это целесообразно делать, когда алгоритм потоковой обработки данных, реализованный в командах процессорных элементов матрицы, превышает ее размер. Для этого необходимо соединить шинами управления узлы управления и загрузки матрицы и шинами данных дополнительные памяти, введенные в состав платы управления, отдельных ОВС. Таким образом, вычислительную систему из одних БВМ ОВС можно расширять посредством соединения плат управления отдельных БВМ ОВС между собой через узел управления и загрузки и дополнительные буферные памяти. Дополнительная буферная память может быть выполнена как обычное ОЗУ или в виде набора регистров, являющегося FIFO-буфером. Аппаратная реализация таких связей между БВМ ОВС не представляется сложной, т.к. данные передаются в одном направлении из предыдущего блока ОВС в последующий для буферизации. Обработка принятых данных последующим блоком ОВС начнется после завершения обработки данных предыдущим. Управляющая программа должна для организации такой связи коммутировать информационную шину данных выходов матрицы не в память хранения данных своей платы управления, а в

дополнительную буферную память следующей платы управления (блока ОВС). Работа по этой специализированной сети включает передачу результата из памяти хранения данных платы управления в оперативную память БВМ ОВС, передачу данных между БВМ ОВС по системной сети, передачу данных из оперативной памяти последующего БВМ ОВС в память хранения данных платы управления.

Рассмотрим подробнее преимущества матрицы ОВС.

Как уже отмечалось, матрица ОВС теоретически является одним из самых эффективных методов решения специализированных задач. Она может обеспечить конкретной задаче быстроедействие, не достижимое для большинства многопроцессорных архитектур. Особенностью ОВС является то, что она является устройством конвейерного типа, ступенями которого являются процессорные элементы, используемые в вычислительном алгоритме. Все элементы матрицы имеют единую синхронизацию. Матрица имеет достаточно большое время для получения результата (многоступенчатый конвейер), но при этом сравнительно небольшое время между последовательными выдачами результатов работы. Это происходит из-за того, что значительное количество промежуточных значений обрабатывается на разных ступенях конвейера. Теоретически считывание входного потока данных из памяти может происходить с частотой работы памяти.

Процессорные элементы могут быть реализованы как на программируемых логических схемах (ПЛИС), так и на БИС заказной технологии с системой команд для программирования выполняемых функций. При этом для работы на высокой частоте синхронизации и выполнении операций за один такт необходимо ограничить разрядность выполняемых операций, выбрать полный набор команд и реализовать их аппаратным способом. Выполнение таких требований при разработке процессора позволит, с одной стороны, осуществить многоступенчатый конвейер при высокой частоте синхронизации, а с другой стороны, обеспечить размещение на одном не самом дорогом ПЛИСе (типа SPARTAN II) нескольких процессорных элементов или изготовить заказной СБИС с такими же возможностями по технологии, совместимой с производством

микросхем. Стоимость изготовления заказного БИС меньше стоимости ПЛИС такой же емкости. Преимуществом СБИС является также то, что при создании программы работы процессорного элемента СБИС используется графический макроассемблер, с помощью которого можно задавать конфигурацию процессорного элемента, выполняемую им команду и т.д. Для настройки ПЛИС используется язык электронного проектирования VERILOG или VHDL, использование которых для программирования функций ПЛИС гораздо сложнее. Малое количество связей между БИСами процессорных элементов, простая цоколевка позволяет изготавливать несложные, недорогие платы. Поэтому стоимость изготовления высокопроизводительной ОВС с многоступенчатым конвейером будет относительно невысокой.

Организация матричных микропроцессорных структур в виде двухмерной матрицы имеет свои преимущества. Как известно, основными способами параллельной обработки данных являются *параллелизм* и *конвейерность*.

Эффективность *параллелизма* заключается в том, что выполнение операций распределяется между несколькими однотипными устройствами. При этом выигрыш во времени выполнения вычисления относительно одного устройства будет во столько раз, сколько задействовано однотипных устройств. Например, если задействовано N однотипных устройств, то выигрыш составит N раз.

Конвейерную обработку можно рассмотреть как линейную структуру, в которой каждая ступень конвейера представляет собой модуль (для упрощения одноктактный), выполняющий свою функцию. Каждый модуль связан с предыдущим и последующим модулями конвейера. Время выполнения вычисления в таких конвейерных устройствах определяется величиной $m+n-1$, где m – количество одноктактных модулей,

n – количество операций.

Если предположить, что конвейер монополизирован выполнением одной операции, то время выполнения n операций равно $m \cdot n$. Поэтому можно

предположить, что при больших n (что верно при потоковой обработке данных, в которой n измеряется тысячами) ускорение конвейерных вычислений составляет m раз.

Двухмерная матрица процессорных элементов позволяет реализовать алгоритмы, которые, с одной стороны, запускают выполнение параллельных операций (по количеству элементов в строке), а с другой стороны, алгоритмы распределяют выполнение операций внутри столбца матрицы, обеспечивая конвейерную обработку между процессорными элементами столбца. Таким образом, если алгоритм использует два этих свойства, предоставляемых двухмерной матрицей, то производительность (скорость выполнения задач) по отношению к одному процессорному элементу увеличится в $(N \cdot m)$ раз. Это показывает, что двухмерная матрица отличается линейной масштабируемостью. Линейная масштабируемость означает, что наращивание количества процессорных элементов влечет за собой линейный рост вычислительной мощности. При этом наращивание выполняется единообразно, и локальность связи оказывается не только топологической, но и физической. Кроме того, двухмерная матрица отличается невысокими аппаратными и временными затратами на организацию межпроцессорных обменов по сравнению с кластерными структурами.

С другой стороны, проектирование на одном кристалле множества однотипных процессоров соответствует новым архитектурным направлениям построения СБИС, учитывающим особенности микроэлектроники. Расположение на одном кристалле множества связанных между собой процессоров позволяет уменьшить ограничения, накладываемые задержками распространения сигналов на количество вентилях, используемых для одного вычисления или одной операции.

Целесообразно стремиться к тому, чтобы устройства такого типа становились более компактными и высокочастотными, т.е. на одном кристалле

должно быть размещено 64–128 однобитных процессора, соединенных в виде двухмерной матрицы и работающих на частоте свыше 50 МГц.

На первом этапе разработки БИС процессорных элементов для ОВС, когда невелики степень интеграции (25 процессорных элементов в БИС) и частота работы процессорных элементов (8–20 МГц), целесообразно создавать устройства ОВС с большим количеством БИС процессорных элементов (более 300 элементов), чтобы в полной мере использовать преимущества конвейерной обработки.

5.4.4.4. Организация гибридных (двухуровневых) суперкомпьютерных систем семейства «СКИФ»

Организация межуровневой связи в гибридных ВВС

Для объединения БВМ ОВС с вычислительными узлами последовательной обработки используются те же сетевые интерфейсы и принципы, на которых реализованы кластеры с параллельной обработкой данных без БВМ ОВС, т.к. в основе построения БВМ ОВС лежит серверная платформа или ПЭВМ с микропроцессорами Intel, AMD, которая выполняет управляющие функции в БВМ ОВС. Поэтому после установки дополнительных сетевых адаптеров вычислительные узлы БВМ ОВС включаются во внутреннюю структуру кластера со всеми его коммуникационными средами. При этом назначение коммуникационных сред не меняется. Системная сеть служит для обмена сообщениями и загрузки данных, вспомогательная сеть – для загрузки программ и управления процессом вычисления, сервисная сеть – для включения/выключения электропитания и обеспечения сериальной консоли. Необходимость в специальной сети между БВМ ОВС при наличии нескольких БВМ ОВС в составе кластера будет оправдана только при использовании программ обработки потоков данных с количеством команд, превосходящим размерность матрицы БВМ ОВС.

К основным критериям выбора коммуникационной среды для объединения вычислительных узлов относятся:

- 1) пропускная способность коммуникационной среды;
- 2) пропускная способность одного адаптера и количество адаптеров на узел;

- 3) пропускная способность коммутатора и количество узлов, подключаемых к коммутатору;
- 4) задержка установления соединения (латентность);
- 5) степень масштабируемости;
- 6) практическое использование, практическая апробация или хотя бы реальное тестирование системы;
- 7) коммерческая доступность системы, ее реальная принадлежность конкретной компании и перспективы существования компании на определенное время;
- 8) наличие развитых программных средств поддержки работы адаптеров и средств его тестирования;
- 9) доступность программного обеспечения, возможности по его модификации и наличие поддержки со стороны производителя при минимальном проникновении в проект пользователя;
- 10) стоимость коммутаторов, адаптеров и программного обеспечения.

Рассмотрим кратко основные виды системной сети, примененные при создании кластерных систем семейства «СКИФ».

При разработке суперкомпьютеров «СКИФ» было применено три технологии: SCI, Myrinet и Infiniband.

Опыт построения «К-500» показал, что SCI годится для кластеров с количеством узлов не более 64. При дальнейшем наращивании количества узлов непропорционально возрастает трудоемкость коммутации, вероятность появления ошибок и падает производительность. К тому же отсутствие в планах фирмы «Dolphin» поддержки PCI-X, не говоря уже о PCI-Express, вызывает сомнение в перспективности этой технологии.

Myrinet – надежная и испытанная кластерная технология, вполне подходящая для проекта создания кластерных систем. Масштабируемость и удобство в монтаже для архитектур с количеством узлов до 128, пожалуй, лучшие из всех трех технологий, однако перспективы и масштабируемость за границу 128 узлов непонятны. Кроме того, закрытая архитектура такой технологии противоречит целям программы «СКИФ».

Infiniband. Проведенные измерения показали, что данная технология обладает (при маленьких сообщениях) задержками, сравнимыми с задержками Myrinet, при этом обеспечивает в 3 с лишним раза большую пропускную

способность. Измерения также показывают, что с увеличением размера сообщения задержка растет медленно и при размере сообщения 128 байтов она становится даже меньше, чем у SCI. Остальные преимущества:

- самая высокая пропускная способность из доступного интерконекта – в 2,5–3 раза лучше SCI и Myrinet;
- общепринятый стандарт, большое количество его поддерживающих производителей и компаний;
- перспективы интеграции с процессорами ведущих производителей (Intel, AMD);
- возможность интеграции хранилищ и управляющей сети;
- поддержка RDMA (Remote Direct Memory Access) для эффективной реализации новых функций MPI-2, а также для других перспективных средств коммуникаций между процессами (DSM и т.д.);
- поддержка протокола SDP – обеспечивает эффективную передачу сообщений с использованием стандартного механизма сокетов в Unix-подобных системах;
- существование аппаратных решений поддержки до 4000 узлов;
- появление решений, поддерживающих (по крайней мере на уровне коммутаторов) спецификацию 12x (30Gbit/s), полностью совместимую со спецификацией 4x.

Для реализации управляющей сети в суперкомпьютерах семейства «СКИФ» применялись коммуникационные сети типа Ethernet.

Программные средства сопряжения кластерного и потокового архитектурных уровней.

Средства взаимодействия двух уровней высокопроизводительной вычислительной системы (ВВС) обеспечивают возможность взаимодействия между кластерным и потоковым уровнями и реализуются в рамках сетей SAN или TCP/IP LAN. Следовательно, при реализации в модулях потокового уровня соответствующих сетевых интерфейсов эти модули, в принципе, могут выступать в качестве устройств системной сети (SAN) и/или вспомогательной сети суперкомпьютера (TCP/IP LAN). Программные средства сопряжения в части кластерного уровня должны включать в себя:

- набор драйверов устройств, обеспечивающих сопряжение кластерного и потокового уровней;
- базовую библиотеку стандартных примитивов обмена информацией и управления потоковым уровнем;
- библиотеку прикладных задач и подпрограмм, реализуемых с использованием потокового уровня;
- структуры данных и программные механизмы, обеспечивающие:
 - передачу Т-процесса, из которого осуществляется взаимодействие с модулем потокового уровня, в один из вычислительных узлов кластерного уровня, имеющих физический интерфейс с модулем потокового уровня;
 - осуществление удаленного вызова функций/прикладной задачи из вычислительного узла кластерного уровня, не имеющего интерфейса с модулем потокового уровня, с использованием механизмов, предназначенных для распределенной работы с файлами.

В части потокового уровня программные средства сопряжения должны включать в себя реализованный в виде специализированной библиотеки набор фрагментов программного кода, предназначенных для загрузки из кластерной компоненты в потоковую. Каждый из фрагментов непосредственно реализует на потоковом уровне ту или иную прикладную задачу или фрагмент вычислений, в частности:

- получает из кластерного уровня наборы входных данных;
- организует и осуществляет выполнение вычислений в модуле потокового уровня в соответствии с алгоритмом решения соответствующей прикладной задачи;
- передает из потокового в кластерный уровень наборы данных, содержащие результаты вычислений.

Описанный набор программных средств, структур данных и механизмов поддерживает возможности:

- передачи фрагмента решаемой задачи из Т-программы на вычисление в модуль потокового уровня;
- передачи фрагмента решаемой задачи из выполняемого в модуле потокового уровня кода на вычисление в кластерную компоненту.

Для выполнения фрагмента задания, предназначенного для решения на БВМ ОВС, в каждый БВМ ОВС записывается программа, состоящая из команд, выполняемых процессорными элементами матрицы для решения фрагмента задачи, а также информация о конфигурации платы управления. Это может происходить в разные моменты времени, если набор команд и данные передаются из разных БВМ последовательного типа. Как правило, на БВМ ОВС программы всех фрагментов задач записываются перед началом решения задачи, а данные направляются по мере готовности в память хранения данных платы управления. После получения данных управляющая программа БВМ посылает в плату управления команду загрузки программы в матрицу, а затем команду запуска вычислительного процесса.

Выполнение программы на вычислительном кластере смешанного типа начинается с загрузки и запуска на одном из вычислительных узлов последовательного типа ее начального процесса (первого фрагмента). Этот процесс в соответствии со структурой программы задачи может загружать на выполнение другие процессы (другие фрагменты этой задачи), которые в свою очередь тоже могут запускать новые вычислительные процессы. При этом один процесс может запускать несколько других процессов до момента своего окончания. До тех пор пока есть свободные вычислительные узлы, все процессы сразу же начинают выполняться на них. Если же свободных узлов (БВМ) нет, то процедура запуска фрагментов переходит в режим ожидания. Последовательные и параллельные фрагменты выполняются на вычислительных узлах соответствующего типа. Таким образом, в ходе выполнения программы решаемой задачи в системе функционирует максимальное количество процессов, а другая часть готовых к выполнению процессов ожидает освобождения БВМ данного типа.

Вычислительные узлы после выполнения очередного процесса осуществляют поиск следующего процесса своего типа путем рассылки по одной из коммуникационных сред управляющего запроса на процесс или данные. Функционирование кластера (его вычислительных узлов) продолжается до тех пор, пока все процессы (фрагменты задачи) не выполняются. В ходе выполнения фрагментов задачи вычислительные узлы обмениваются данными через коммуникационные среды.

После окончания выполнения фрагмента задачи БВМ ОВС через контроллер PCI платы управления выдается прерывание, оповещающая свой узел о готовности данных (результатов выполнения фрагмента задачи) для передачи другому узлу.

Конструктивное исполнение БВМ ОВС

Так как БВМ ОВС могут использоваться вместе с вычислительными узлами последовательного типа в кластерах, то конструктив БВМ ОВС должен быть совместим с наиболее эффективной технологией реализации вычислительных узлов последовательного типа. Этой технологией являются серверы типа «лезвие». Серверы такого типа представляют собой компактно упакованные в одном шасси одноплатные компьютеры, связанные высокоскоростной сетью и имеющие общий блок питания и систему вентиляции. При этом для упрощения конструкции размеры плат матрицы процессорных элементов должны совпадать с размерами системных плат, используемых в серверах типа «лезвие». Для уменьшения количества кабелей внешних соединений используется объединительная панель, в которую устанавливаются платы матриц и которая должна иметь разводку электропитания, шин связи между отдельными матрицами. Все это повысит производительность, приходящуюся на единицу объема стойки. Более того, соблюдение конструктивов серверов типа лезвие, позволит создать смешанные конфигурации вычислительных узлов на одном шасси. Это уменьшит эксплуатационные расходы на обслуживание кластера.

БВМ ОВС использует системные платы такого же типа, как и вычислительные узлы последовательного действия. Поэтому применение новых типов системных сетевых интерфейсов в кластерах смешанного типа не становится проблемой. БВМ ОВС не выставляет ограничений и дополнительных требований на организацию системной сети кластера. Использование таких новых сетевых интерфейсов, как Infiniband, позволяет увеличить скорость загрузки данными для обработки на БВМ ОВС и выдачу результатов обработки на вычислительный узел последовательного действия для запуска фрагмента задачи.

5.4.4.5. Перспективы дальнейшего развития семейства «СКИФ»

В завершение раздела рассмотрим основные принципы дальнейшего развития высокопроизводительных вычислительных систем (ВВС) семейства «СКИФ».

Предпосылки для развития ВВС семейства «СКИФ»

Востребованность работ по развитию в странах Союзного государства направления суперкомпьютерных технологий определяется их отставанием от ведущих мировых держав в развитии и применении важнейших наукоемких информационных технологий, нацеленных на решение сложных задач машиностроения, биотехнологии, геологоразведки, контроля окружающей среды, транспорта и связи, государственных, коммерческих, военных и других приложений. Программные же и аппаратные компоненты наукоемких информационных технологий, разработанные западными фирмами, из-за своей высокой цены недоступны подавляющему большинству белорусских промышленных предприятий. Это оборачивается невозможностью создания современной конкурентоспособной на внутреннем и внешнем рынках продукции как гражданского, так и оборонного назначения.

Применение суперкомпьютерных систем и наукоемких программных продуктов актуально также и в таких приложениях, как банки данных, информационно-аналитические системы, ситуационные центры управления, системы управления в реальном масштабе времени, боевые информационно-управляющие системы и др.

Для эффективного решения задач на базе наукоемких технологий для различных областей применения необходимо использование принципиально новых суперкомпьютерных технологий обработки данных в широком диапазоне производительности – от сотен миллиардов операций в секунду до вычислительных систем с массовым параллелизмом сверхвысокой производительности (триллионы операций в секунду). Потребность в высокопроизводительных вычислительных ресурсах не может быть удовлетворена путем применения существующих или ожидаемых в ближайшие годы на мировом рынке персональных компьютеров или серверов на платформе Intel. Закупка же импортных суперкомпьютерных конфигураций связана с систематическими политическими ограничениями со стороны потенциальных экспортеров и с утечкой значительных финансовых средств за рубеж.

Проблема усугубляется также и крайней сложностью освоения современных наукоемких программных систем мирового уровня. Необходима инфраструктура, предоставляющая комплексные услуги по внедрению основных компонентов наукоемких информационных технологий, в том числе предоставление доступа к суперкомпьютерным вычислительным ресурсам, консультации по выбору и применению технологий, сопровождение компьютерных средств и программного обеспечения, обучение и др. Решению этих проблем может способствовать создание в Республике Беларусь и Российской Федерации суперкомпьютерных центров для развития и внедрения наукоемких информационных технологий. Это обеспечит предоставление услуг для решения наукоемких задач, возникающих в промышленности и в других областях народного хозяйства, требующих компьютерных и информационных ресурсов, владение которыми недоступно или экономически нецелесообразно для отдельных организаций.

В рамках программы «СКИФ» создан существенный научно-технический задел, позволяющий создавать суперкомпьютерные конфигурации в широком диапазоне производительности, вплоть до триллиона операций в секунду. Для решения задач на базе наукоемких технологий для различных областей применения в рамках программы «СКИФ» были созданы кластеры «СКИФ К-500» и «СКИФ К-1000». Включение этих кластеров в список пятисот наиболее мощных вычислительных установок в мире означает достижение важного прямого политического эффекта – Республика Беларусь и Россия наравне с США, Японией и еще несколькими странами стали обладателями критической суперкомпьютерной технологии, повысив престиж Союзного государства как разработчика этой технологии.

Анализ хода реализации программы «СКИФ» с учетом реальных объемов финансирования работ позволяет сделать вывод, что благодаря ей обеспечены научно-технические и производственные предпосылки для широкого практического внедрения суперкомпьютерных технологий, включая создание суперкомпьютерных центров. Реализация этих предпосылок позволит уменьшить отставание стран-участниц Союзного государства от ведущих мировых держав в развитии и применении новейших наукоемких технологий в таких областях, как машиностроение, биотехнология, геологоразведка, контроль окружающей среды, транспорт и связь, в государственных, коммерческих, военных и других

приложениях. Применение суперкомпьютерных систем и наукоемких программных продуктов актуально также и в таких приложениях, как банки данных, информационно-аналитические системы, ситуационные центры управления в реальном масштабе времени и др.

Интегральный экономический и политический эффект от комплексной реализации программы обеспечивается тем, что ее результаты будут способствовать форсированному технологическому перевооружению ключевых отраслей промышленности стран-участниц Союзного государства, их реформированию с целью достижения мирового уровня качества продукции на базе новейших наукоемких информационных технологий и суперкомпьютерных конфигураций «СКИФ».

Предпосылкой получения экономического эффекта от реализации программы является комплексный подход, базирующийся на единой идеологии создания семейства недорогих моделей суперкомпьютеров с параллельной архитектурой, предназначенных для оптимального решения широкого класса задач с большими объемами вычислений.

Результаты комплексной реализации программы «СКИФ» являются существенным научно-техническим и организационным заделом для дальнейшего развития суперкомпьютерного направления, в том числе для формирования новых программ Союзного государства по развитию суперкомпьютерного направления «СКИФ».

Стратегия развития суперкомпьютерного направления «СКИФ»

С учетом созданного в рамках программы «СКИФ» научно-технического задела главной целью развития ВВС семейства «СКИФ» может быть освоение и адаптация передовых зарубежных и отечественных наукоемких технологий на отечественных суперкомпьютерных конфигурациях семейства «СКИФ», внедрение этих технологий в основные области гражданской и военной промышленности и социально-экономическую сферу Союзного государства, оптимизация отечественных суперкомпьютерных конфигураций семейства «СКИФ» с учетом требований современных наукоемких технологий и специфики их приложений.

Создаваемые суперкомпьютеры семейства «СКИФ» являются базой для реализации сформированной Министерством промышленности, науки и технологий Российской Федерации и Национальной академии наук Беларуси научно-технической программы Союзного государства «Развитие и внедрение в государствах – участниках Союзного государства наукоемких компьютерных технологий на базе мультимикропроцессорных вычислительных систем» (шифр «Триада»).

Наукоемкие компьютерные технологии представляют собой сложные программные комплексы, реализующие алгоритмы решения задач со значительным использованием математического аппарата и знаний по естественным наукам. Примерами наиболее практически значимых наукоемких технологий являются базирующиеся на аппарате математической физики и вычислительной математики программные комплексы для решения задач инженерного проектирования: расчет прочности и моделирования разрушения конструкций; расчет теплофизических процессов; решение задач электромагнетизма и газогидродинамики. Другой пример – программные наукоемкие комплексы, базирующиеся на интенсивном использовании аппарата дискретной математики, комбинаторных вычислений, теории сложных систем, математической лингвистики и алгебры, для решения задач управления сложными объектами и социально-экономическими системами; построения, тестирования и оценки систем управления в реальном времени.

На базе результатов программы «СКИФ» могут быть сформированы и другие (наряду с программой «Триада») направления – защита информации, специальные условия эксплуатации и т.п. Однако *все эти направления перспективны лишь при развитии стратегического направления – оптимизации отечественных суперкомпьютерных конфигураций семейства «СКИФ» с учетом специфики требований современных наукоемких технологий и других суперкомпьютерных приложений. Развитие этого стратегического направления предполагается в рамках новой программы Союзного государства (базовой для развития суперкомпьютерного направления «СКИФ») с условным шифром «СКИФ-2».*

К основным возможным направлениям работ в рамках программы «СКИФ-2» относятся:

- гетерогенные кластерные и метакластерные системы;

- отказоустойчивость, живучесть, надежность (программные и конструктивно-технологические методы, горячий резерв и т.д.);
- перспективные методы комплексирования (например Hyper-Transport, Infiniband etc);
- гибридные узлы и архитектуры (оригинальные СБИС, нейроструктуры, FPGA);
- Grid-конфигурации на базе суперкомпьютеров «СКИФ»;
- объединение внутренней шины и интерконнекта;
- эффективные механизмы (например альтернатива MPI) обмена данными (новый транспортный уровень);
- направление Lisa – самоконфигурирование, самодиагностика и т.п. (концепция IBM);
- системы разделения ресурсов в кластерных и метакластерных конфигурациях и GRID системах;
- защита данных (в том числе с учетом ПЭМИ);
- перспективные конструктивные решения («лезвие бритвы» и т.п.);
- мобильные (возимые) системы на базе Grid-технологий;
- условия эксплуатации;
- объединенная система управления и мониторинга кластерными структурами (развитие сервисной сети);
- развитие направления T-система (T-Grid);
- апгрейдируемость (способность к модернизации).

Тема 6. АБСТРАКТНЫЕ ИНФОРМАЦИОННЫЕ МАШИНЫ

6.1 Понятие абстрактной информационной машины

Абстрактной машиной в рамках данного курса будем называть математическую формализацию, которая моделирует правила выполнения программы (или, иначе, алгоритмы) для реальной вычислительной машины (компьютера).

Отметим так же то обстоятельство, что *абстрактные машины* позволяют адекватно моделировать различные подходы и стратегии вычислений.

Уточнив понятия об *абстрактных машинах*, перейдем к исследованию особенностей различных классов рассматриваемых формализаций, проиллюстрировав выделенные классы необходимыми примерами из математической теории и практики программирования.

Прежде всего, следует отметить то обстоятельство, что практически все без исключения абстрактные модели вычислительных машин оперируют понятием *состояния*, изменения которого и отражают ход выполнения программы.

При этом следует, во-первых, выделить ранние, "наивные" формализации на *состояниях*, которые не были практически поддержаны ни языками программирования, ни собственно компьютерами. К ранним *абстрактным машинам* можно отнести известные из истории математики машины *Тьюринга* и *Поста*. Первая *абстрактная машина* характеризовалась бесконечной лентой для хранения "инструкций", а также считывающей и записывающей головкой, передвигающейся по ней; вторая машина действовала подобно первой.

Несмотря на объективные трудности практической реализации, ранние *абстрактные машины*, безусловно, были весьма значимыми для развития компьютерных наук, т.к. предвосхитили появление и обозначили ряд этапов развития реальных компьютеров и языков программирования для них.

Машина Тьюринга – математическая абстракция, представляющая вычислительную машину общего вида. Была предложена Аланом Тьюрингом в 1936 году для формализации понятия алгоритма.

Машина Тьюринга является расширением модели конечного автомата и, согласно тезису Чёрча — Тьюринга, способна имитировать (при наличии соответствующей программы) любую машину, действие которой заключается в переходе от одного дискретного состояния к другому.

В состав Машины Тьюринга входит бесконечная в обе стороны *лента*, разделённая на ячейки, и *управляющее устройство* с конечным числом состояний.

Управляющее устройство может перемещаться влево и вправо по ленте, читать и записывать в ячейки символы некоторого конечного алфавита. Выделяется особый *пустой* символ, заполняющий все клетки ленты, кроме тех из них (конечного числа), на которых записаны входные данные.

В управляющем устройстве содержится *таблица переходов*, которая представляет алгоритм, *реализуемый* данной Машиной Тьюринга. Каждое правило из таблицы предписывает машине, в зависимости от текущего состояния и наблюдаемого в текущей клетке символа, записать в эту клетку новый символ, перейти в новое состояние и переместиться на одну клетку влево или вправо. Некоторые состояния Машины Тьюринга могут быть помечены как *терминальные*, и переход в любое из них означает конец работы, остановку алгоритма.

Машина Тьюринга называется *детерминированной*, если каждой комбинации состояния и ленточного символа в таблице соответствует не более одного правила, и *недетерминированной* в противном случае.

Машина Тьюринга называется *вероятностной*, если она детерминированная и из любого состояния и значений на ленте, машина может совершить один из нескольких (можно считать, без ограничения общности – двух) возможных переходов, а выбор осуществляется вероятностным образом.

Вероятностная Машина Тьюринга похожа на недетерминированную машину Тьюринга, только вместо недетерминированного перехода машина выбирает один из вариантов с некоторой вероятностью.

Тезис Чёрча фундаментальное утверждение для многих областей науки, таких, как теория вычислимости, информатика, теоретическая кибернетика и др. Это утверждение было высказано Алонзо Чёрчем и Аланом Тьюрингом в середине 1930-х годов.

В самой общей форме оно гласит, что *любая интуитивно вычислимая функция является частично вычислимой, или, эквивалентно, может быть вычислена с помощью некоторой машины Тьюринга.*

Тезис Чёрча-Тьюринга невозможно строго доказать или опровергнуть, поскольку он устанавливает «равенство» между строго формализованным понятием частично вычислимой функции и неформальным понятием «интуитивно вычислимой функции».

Физический тезис Чёрча-Тьюринга гласит: *Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга.*

Машина Поста - абстрактная вычислительная машина, предложенная Эмилем Л. Постом (Emil L. Post), которая отличается от машины Тьюринга большей простотой. Обе машины эквивалентны и были созданы для уточнения понятия алгоритм. Прежде всего надо сказать, что машина Поста не есть реально существующее, сделанное кем-то устройство. Машина Поста, как и ее близкий родственник машина Тьюринга, представляет собой мысленную конструкцию, хотя устройство, позволяющее моделировать работу машины Поста в случае небольших программ и небольших объемов вычислений, было изготовлено в 1970 году в Симферопольском государственном университете. Однако для нас не будет существенным факт,

что машины Поста на самом деле нет. Напротив, мы будем предполагать ее как бы "существующей".

Машина Поста состоит из ленты и каретки (называемой также считывающей и записывающей головкой). Лента бесконечна и разделена на секции одинакового размера. Порядок, в котором расположены секции ленты, подобен порядку, в котором расположены все целые числа. Поэтому естественно ввести на ленте "целочисленную систему координат", занумеровав секции числами ..., -3, -2, -1, 0, 1, 2, 3,...

В каждой секции ленты может быть либо ничего не записано (такая секция называется пустой), либо стоять метка "V" (тогда секция называется отмеченной).

Информация о том, какие секции пустые, а какие отмеченные, образует состояние ленты. Иными словами, состояние ленты - это распределение меток по ее секциям. На точном математическом языке состояние ленты - это функция, которая каждому числу (номеру секции) ставит в соответствие либо метку, либо "пусто". Состояние ленты, в процессе работы, может меняться.

Каретка может передвигаться вдоль ленты влево и вправо. Когда она неподвижна, она стоит против ровно одной секции ленты; говорят, что каретка обозревает эту секцию, или держит ее в поле зрения. Информация о том, какие секции пустые, а какие отмеченные и где стоит каретка, образуют состояние машины Поста.

Таким образом, состояние машины Поста складывается из состояния ленты и указания номера той секции, которую обозревает каретка.

За единицу времени, которая называется шагом, каретка может сдвинуться на одну секцию влево или вправо. Кроме того, каретка может поставить (напечатать) или уничтожить (стереть) метку в той секции, против которой она стоит, а также распознать, стоит или нет метка в обозреваемой ею секции.

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Эта работа происходит по инструкции определенного вида, называемого программой. Для машины Поста возможно составление различных программ из набора команд, но эти команды строго определены. Командой машины Поста называется выражение, имеющее один из следующих шести видов:

- 1) команда движения вправо: $m . => n$
- 2) команда движения влево: $m . <= n$
- 3) команда печати метки: $m . p \ n$
- 4) команда стирания метки: $m . s \ n$
- 5) команда передачи: $m . ? \ n_1, n_2$
- 6) команда остановки: $m . stop$

Число m , стоящее в начале команды, называется номером команды, а число n , стоящее после команды (а у команды передачи управления n_1 и n_2), называется отсылкой. У команды остановки отсылка отсутствует.

Программой машины Поста называется конечный непустой (т.е. содержащий хотя бы одну команду) список команд машины Поста, обладающий следующими двумя свойствами.

1. На первом месте в этом списке стоит команда с номером 1, на втором месте (если оно есть) - команда с номером 2 и т.д.; вообще на k -м месте стоит команда с номером k .

2. Отсылка любой из команд списка совпадает с номером некоторой (другой или той же самой) команды списка (более точно: для каждой отсылки каждой команды списка найдется в списке такая команда, номер которой равен рассматриваемой отсылке).

Чтобы машина Поста работала, надо задать некоторую программу и некоторое состояние машины, т.е. как-то расставить метки по секциям ленты (в частности, можно все секции оставить пустыми) и поставить каретку против одной из секций.

Работа машины на основании заданной программы происходит следующим образом. Машина приводится в начальное состояние и приступает к выполнению первой команды программы. Каждая команда выполняется за один шаг, а переход от выполнения одной команды к выполнению другой происходит по следующему правилу: пусть на k -м шаге выполнялась команда с номером m , тогда,

1) если эта команда имеет единственную отсылку n , то на $(k+1)$ -м шаге выполняется команда с номером n ;

2) если эта команда имеет две отсылки n_1 и n_2 , то на $(k+1)$ -м шаге выполняется одна из двух команд - с номером n_1 или с номером n_2 ;

3) если же выполняющаяся на k -м шаге команда вовсе не имеет отсылки, то на $(k+1)$ -м шаге и на всех последующих шагах не выполняется никакая команда - машина останавливается.

Выполнение команды движения вправо состоит в том, что каретка сдвигается на одну секцию вправо.

Выполнение команды движения влево состоит в том, что каретка сдвигается на одну секцию влево.

Выполнение команды печати метки состоит в том, что каретка ставит метку на обозреваемой секции. Выполнение этой команды возможно лишь в том случае, если обозреваемая перед началом выполнения команды секция пуста. Если же на обозреваемой секции уже стоит метка, то команда считается невыполнимой.

Выполнение команды стирания метки состоит в том, что каретка уничтожает (стирает) метку в обозреваемой секции. Выполнение этой команды возможно лишь в том случае, если обозреваемая секция отмечена (в ней стоит метка). Если же на обозреваемой секции нет метки, команда считается невыполнимой.

Выполнение команды передачи управления с отсылками n_1 и n_2 никак не изменяет состояние машины: ни одна из меток не уничтожается и не ставится, и каретка также остается неподвижной (машина делает "шаг на месте"). Однако если секция, обозреваемая перед началом выполнения

команды, была отмечена, то следующей должна выполняться команда с номером n1. Если же эта секция была пустая, то следующая должна выполняться команда с номером n2 (роль команды передачи управления сводится, следовательно, к тому, что каретка во время выполнения этой команды как бы "распознает", стоит ли перед ней метка).

Выполнение команды остановки тоже никак не меняет состояния машины и состоит в том, что машина останавливается.

Если теперь, задав программу и какое-либо начальное состояние, пустить машину в ход, то осуществится один из следующих трех вариантов.

1. В ходе выполнения программы машина дойдет до выполнения невыполнимой команды (печать метки в непустой секции или стирание метки уже в пустом месте). Выполнение программы тогда прекращается, машина останавливается - происходит безрезультативная остановка (аварийная остановка).

2. В ходе выполнения программы машина дойдет до выполнения команды остановки. Программа в этом случае считается выполненной, машина останавливается - происходит результативная остановка (нормальное завершение).

3. В ходе выполнения программы машина не дойдет до выполнения ни одной из команд, указанных в первых двух вариантах. Выполнение программы при этом никогда не прекращается, машина никогда не останавливается - процесс работы машины происходит бесконечно (зацикливание). При зацикливании и аварийном завершении не получаем никаких результатов. При нормальном завершении получаем результаты, которые могут быть правильными или неправильными. Правильные результаты возможны при правильно составленном алгоритме и соответствующих входных данных.

В качестве примера составим программу для следующей задачи. Необходимо сложить два натуральных числа, находящихся на произвольном расстоянии. Начальное положение головки - где-то над первым числом.

- | | | |
|-----|--------|--|
| 1. | <= 2 | Поиск начала первого числа: сдвигаемся влево |
| 2. | ? 3,1 | до тех пор, пока не встретим не отмеченную ячейку. |
| 3. | => 4 | Сдвинуться вправо на одну метку и |
| 4. | s 5 | и удалить ее. |
| 5. | => 6 | Поиск конца первого числа: сдвиг вправо |
| 6. | ? 7,5 | пока головка не встанет на неотмеченную ячейку и |
| 7. | p 8 | ставим ее |
| 8. | => 9 | Проверить заполнен ли промежуток между числами |
| 9. | ? 1,10 | если промежуток не заполнился, то переход на начало. |
| 10. | stop | Конец программы |

Очевидно, что использование в Машине Поста нескольких лент и (или) головок позволило бы избежать многочисленных перемещений головки по ленте и перезаписи чисел, существенно упростив программирование, однако эти ограничения на устройство Машины Поста никак не ограничивают возможности вычислений в принципе.

6.2 Программная реализация абстрактных информационных машин.

В 1946 году группа учёных во главе с Джоном фон Нейманом (Герман Голдстайн, Артур Беркс) опубликовали статью «Предварительное рассмотрение логической конструкции Электронно-вычислительного устройства». В статье обосновывалось использование двоичной системы для представления данных в ЭВМ (преимущественно для технической реализации, простота выполнения арифметических и логических операций. До этого машины хранили данные в десятичном виде), выдвигалась идея использования программами общей памяти. Имя фон Неймана было достаточно широко известно в науке того времени, что отодвинуло на второй план его соавторов, и данные идеи получили название «Принципы фон Неймана».

Принцип программного управления.

- Программа состоит из набора команд, которые выполняются процессором друг за другом в определенной последовательности. Он обеспечивает автоматизацию процессов вычислений на ЭВМ.

Принцип однородности памяти.

- Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления – чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными. Иногда этот принцип называют «принцип хранимой команды». И это отсутствие принципиальной разницы между программой и данными дало возможность ЭВМ самой формировать для себя программу в соответствии с результатом вычислений.

Принцип адресуемости памяти.

Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка. Это позволяет обращаться к произвольной ячейке (адресу) без просмотра предыдущих.

Компьютеры, построенные на этих принципах, относят к типу фон-нейманских. На сегодняшний день это подавляющее большинство компьютеров, в том числе и IBM PS – совместимые.

Архитектура фон Неймана – широко известный принцип совместного хранения программы и данных в памяти компьютера. Вычислительные системы такого рода часто обозначают термином абстрактная машина фон Неймана или машина фон Неймана (рис 6.1), однако, соответствие этих понятий не всегда однозначно. В общем случае, когда говорят об архитектуре фон Неймана, подразумевают физическое отделение процессорного модуля от устройств хранения программ и данных.

Наличие заданного набора исполняемых команд и программ было характерной чертой первых компьютерных систем. Сегодня подобный

дизайн применяют с целью упрощения конструкции вычислительного устройства. Так, настольные калькуляторы, в принципе, являются устройствами с фиксированным набором выполняемых программ. Их можно использовать для математических расчётов, но невозможно применить для обработки текста и компьютерных игр, для просмотра графических изображений или видео. Изменение встроенной программы для такого рода устройств требует практически полной их переделки, и в большинстве случаев невозможно. Впрочем, перепрограммирование ранних компьютерных систем всё-таки выполнялось, однако требовало огромного объёма ручной работы по подготовке новой документации, перекоммутации и перестройки блоков и устройств и т.п.

Всё изменила идея хранения компьютерных программ в общей памяти. Ко времени её появления использование архитектур, основанных на наборах исполняемых инструкций, и представление вычислительного процесса как процесса выполнения инструкций, записанных в программе, чрезвычайно увеличило гибкость вычислительных систем в плане обработки данных. Один и тот же подход к рассмотрению данных и инструкций сделал лёгкой задачу изменения самих программ.

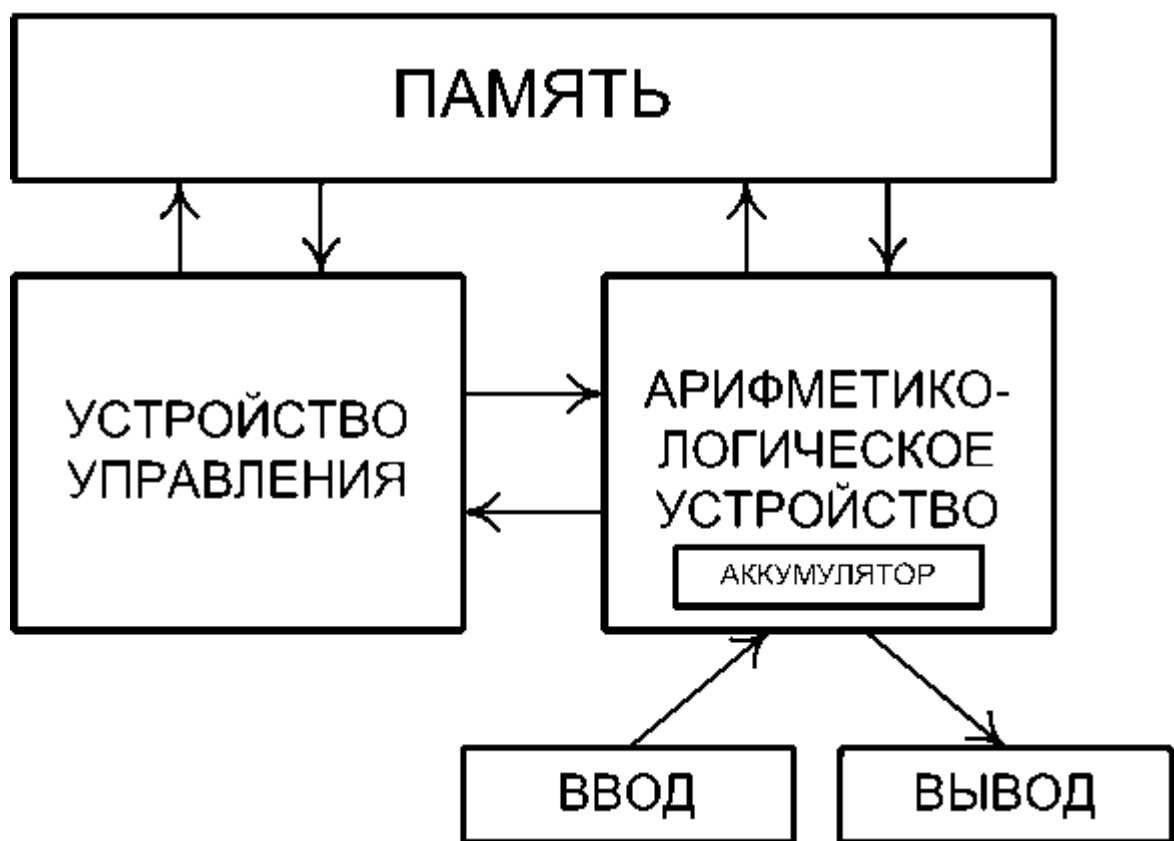


Рис. 6.1 – Схема абстрактной машины фон-Неймана

6.3 Языки функционального программирования

Программы на традиционных языках программирования, таких как Си, Паскаль, Java и т.п. состоят из последовательности модификаций значений некоторого набора переменных, который называется состоянием. Если не

рассматривать операции ввода-вывода, а также не учитывать того факта, что программа может работать непрерывно (т.е. без остановок, как в случае серверных программ), можно сделать следующую абстракцию. До начала выполнения программы состояние имеет некоторое начальное значение σ_0 , в котором представлены входные значения программы. После завершения программы состояние имеет новое значение σ , включающее в себя то, что можно рассматривать как «результат» работы программы. Во время исполнения каждая команда изменяет состояние; следовательно, состояние проходит через некоторую конечную последовательность значений:

$$\sigma = \sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \dots \rightarrow \sigma_n = \sigma$$

Состояние модифицируется с помощью команд присваивания, записываемых в виде $v=E$ или $v:=E$, где v — переменная, а E — некоторое выражение. Эти команды следуют одна за другой; операторы, такие как `if` и `while`, позволяют изменить порядок выполнения этих команд в зависимости от текущего значения состояния. Такой стиль программирования называют императивным или процедурным.

Функциональное программирование представляет парадигму, в корне отличную от представленной выше модели. Функциональная программа представляет собой некоторое выражение (в математическом смысле); выполнение программы означает вычисление значения этого выражения.

С учетом приведенных выше обозначений, считая что результат работы Употребление термина «вычисление» не означает, что программа может оперировать только с числами; результатом вычисления могут оказаться строки, списки и вообще, любые допустимые в языке структуры данных.

императивной программы полностью и однозначно определен ее входом, можно сказать, что финальное состояние (или любое промежуточное) представляет собой некоторую функцию (в математическом смысле) от начального состояния, т.е. $\sigma = f(\sigma_0)$.

В функционально программировании используется именно эта точка зрения: программа представляет собой выражение, соответствующее функции f . Функциональные языки программирования поддерживают построение таких выражений, предоставляя широкий выбор соответствующих языковых конструкций.

При сравнении функционального и императивного подхода к программированию можно заметить следующие свойства функциональных программ:

- Функциональные программы не используют переменные в том смысле, в котором это слово употребляется в императивном программировании. В частности в функциональных программах не используется оператор присваивания.
- Как следствие из предыдущего пункта, в функциональных программах нет циклов.
- Выполнение последовательности команд в функциональной программе бессмысленно, поскольку одна команда не может повлиять на выполнение следующей.

- Функциональные программы используют функции гораздо более замысловатыми способами. Функции можно передавать в другие функции в качестве аргументов и возвращать в качестве результата, и даже в общем случае проводить вычисления, результатом которого будет функция.

- Вместо циклов функциональные программы широко используют рекурсивные функции. На первый взгляд функциональный подход к программированию может показаться странным, непривычным и мало полезным, однако необходимо принять во внимание следующие соображения.

Прежде всего, императивный стиль в программировании не является жестко заданной необходимостью. Многие характеристики императивных языков программирования являются результатом абстрагирования от низкоуровневых деталей реализации компьютера, от машинных кодов к языкам ассемблера, а затем к языкам типа Фортрана и т.д. Однако нет причин полагать, что такие языки отражают наиболее естественный для человека способ сообщить машине о своих намерениях. Возможно, более правилен подход, при котором языки программирования рождаются как абстрактные системы для записи алгоритмов, а затем происходит их перевод на императивный язык компьютера.

Далее, функциональный подход имеет ряд преимуществ перед императивным. Прежде всего, функциональные программы более непосредственно соответствуют математическим объектам, и следовательно, позволяют проводить строгие рассуждения. Установить значение императивной программы, т.е. той функции, вычисление которой она реализует, может оказаться довольно трудно. Напротив, значение функциональной программы может быть выведено практически непосредственно.

Например, рассмотрим следующую программу на языке Haskell:

```
factorial n = if n == 0 then 1 else n * factorial (n - 1)
```

Практически сразу видно, что эта программа соответствует следующей частичной функции:

$$f(n) = (n! \mid n \geq 0 \perp n < 0)$$

(Здесь символ \perp означает неопределенность функции, поскольку при отрицательных значениях аргумента программа не завершается.) Однако для программы на языке Си это соответствие не очевидно:

```
int f (int n)
{
  int x = 1;
  while (n > 0)
  {
    x = x * n;
    n = n - 1;
  }
  return x;
}
```

Следует также сделать замечание относительно употребления термина «функция» в таких языках как Си, Java и т.п. В математическом смысле «функции» языка Си не являются функциями, поскольку:

- Их значение может зависеть не только от аргументов;
- Результатом их выполнения могут быть разнообразные побочные эффекты (например, изменение значений глобальных переменных)
- Два вызова одной и той же функции с одними и теми же аргументами могут привести к различным результатам.

Вместе с тем функции в функциональных программах действительно являются функциями в том смысле, в котором это понимается в математике. Соответственно, те замечания, которые были сделаны выше, к ним не применимы. Из этого следует, что вычисление любого выражения не может иметь никаких побочных эффектов, и значит, порядок вычисления его подвыражений не оказывает влияния на результат. Таким образом, функциональные программы легко поддаются распараллеливанию, поскольку отдельные компоненты выражений могут вычисляться одновременно.

Основы лямбда-исчисления. Подобно тому, как теория машин Тьюринга является основой императивных языков программирования, лямбда-исчисление служит базисом и математическим «фундаментом», на котором основаны все функциональные языки программирования.

Лямбда-исчисление было изобретено в начале 30-х годов логиком А. Черчем, который надеялся использовать его в качестве формализма для обоснования математики. Вскоре были обнаружены проблемы, делающие невозможным его использование в этом качестве (сейчас есть основания полагать, что это не совсем верно) и лямбда-исчисление осталось как один из способов формализации понятия алгоритма.

В настоящее время лямбда-исчисление является основной из таких формализаций, применяемой в исследованиях связанных с языками программирования. Связано это, вероятно, со следующими факторами:

- Это единственная формализация, которая, хотя и с некоторыми неудобствами, действительно может быть непосредственно использована для написания программ.
- Лямбда-исчисление дает простую и естественную модель для таких важных понятий, как рекурсия и вложенные среды.
- Большинство конструкций традиционных языков программирования может быть более или менее непосредственно отображено в конструкции лямбда-исчисления.
- Функциональные языки являются в основном удобной формой синтаксической записи для конструкций различных вариантов лямбдаисчисления. Некоторые современные языки (Haskell, Clean) имеют 100% соответствие своей семантики с семантикой подразумеваемых конструкций лямбда-исчисления.

В математике, когда необходимо говорить о какой-либо функции, принято давать этой функции некоторое имя и впоследствии использовать его, как, например, в следующем утверждении:

Пусть $f : \mathbb{R} \rightarrow \mathbb{R}$ определяется следующим выражением:

$f(x) = \begin{cases} 0, & \text{если } x = 0, \\ x^2 \sin(x), & \text{если } x \neq 0 \end{cases}$

Тогда $f_0(x)$ не интегрируема на интервале $[0, 1]$.

Многие языки программирования также допускают определение функций только с присваиванием им некоторых имен. Например, в языке Си функция всегда должна иметь имя. Это кажется естественным, однако поскольку в функциональном программировании функции используются повсеместно, такой подход может привести к серьезным затруднениям.

Представьте себе, что мы должны всегда оперировать с арифметическими выражениями в подобном стиле:

Пусть $x = 2$ и $y = 4$. Тогда $xx = y$.

Лямбда-нотация позволяет определять функции с той же легкостью, что и другие математические объекты. Лямбда-выражением будем называть конструкцию вида $\lambda x.E$

где E — некоторое выражение, возможно, использующее переменную x . Пример. $\lambda x.x^2$ представляет собой функцию, возводящую свой аргумент в квадрат.

Использование лямбда-нотации позволяет четко разделить случаи, когда под выражением вида $f(x)$ мы понимаем саму функцию f и ее значение в точке x . Кроме того, лямбда-нотация позволяет формализовать практически все виды математической нотации. Если начать с констант и переменных и строить выражения только с помощью лямбда-выражений и применений функции к аргументам, то можно представить очень сложные математические выражения.

Применение функции f к аргументу x мы будем обозначать как $f\ x$, т.е., в отличие от того, как это принято в математике, не будем использовать скобки

По причинам, которые станут ясны позднее, будем считать, что применение функции к аргументу ассоциативно влево, т.е. $f\ x\ y$ означает $(f(x))(y)$. В качестве сокращения для выражений вида $\lambda x.\lambda y.E$ будем использовать запись $\lambda x\ y.E$ (аналогично для большего числа аргументов). Также будем считать, что «область действия» лямбда-выражения простирается вправо насколько возможно, т.е., например, $\lambda x.x\ y$ означает $\lambda x.(x\ y)$, а не $(\lambda x.x)y$.

На первый взгляд кажется, что нам необходимо ввести специальное обозначение для функций нескольких аргументов. Однако существует операция каррирования, позволяющая записать такие функции в обычной лямбда-нотации. Идея заключается в том, чтобы использовать выражения вида $\lambda x\ y.x + y$. Такое выражение можно рассматривать как функцию $\mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$, т.е. если его применить к одному аргументу, результатом будет функция, которая затем принимает другой аргумент.

Таким образом:

$$(\lambda x\ y.x + y)\ 1\ 2 = (\lambda y.1 + y)\ 2 = 1 + 2.$$

Переменные в лямбда-выражениях могут быть свободными и связанными. В выражении вида $x^2 + x$ переменная x является свободной; его значение зависит от значения переменной x и в общем случае ее нельзя переименовать. Однако в таких выражениях как

$\sum_{i=1}^n i$ или $\int_0^x \sin(y) dy$ переменные i и y являются связанными; если вместо i везде использовать обозначение j , значение выражения не изменится.

Следует понимать, что в каком-либо подвыражении переменная может быть свободной (как в выражении под интегралом), однако во всем выражении она связана какой-либо операцией связывания переменной, такой как операция суммирования. Та часть выражения, которая находится «внутри» операции связывания, называется областью видимости переменной.

В лямбда исчислении выражения $\lambda x.E[x]$ и $\lambda y.E[y]$ считаются эквивалентными (это называется α -эквивалентностью, и процесс преобразования между такими парами называют α -преобразованием). Разумеется, необходимо наложить условие, что y не является свободной переменной в $E[x]$.

6.4 Абстрактные машины логического программирования и соответствующие им логические компьютеры.

Лисп-машина – универсальная вычислительная машина, архитектура которой оптимизирована для эффективного выполнения программ на языке Лисп.

Эквивалентна (абстрактной) машине Тьюринга (и обычному персональному компьютеру) по критерию полиномиальной сводимости.

Несмотря на то, что Лисп-машины никогда не были широко распространены (около 7000 во всём мире на 1988), многие распространённые ныне идеи и программные технологии были впервые разработаны с помощью Лисп-машин, например на тех, что использовались в исследовательском центре XEROX Parc:

- Сборка мусора;
- Лазерная печать;
- Многооконные системы;
- Растровая графика высокого разрешения;
- Рендеринг;
- Множество сетевых инноваций.

Лисп-машины предоставляли широкие возможности по проведению экспериментальных разработок в области компьютерных наук. На базе разработок Лисп-машин было создано новое поколение инженерных рабочих станций.

Но на определенном этапе экспоненциального роста вычислительной мощности (закон Мура) аппаратная поддержка лямбда-исчисления утратила экономический смысл для компьютеров широкого потребления, и производители Лисп-машин ушли с рынка.

С другой стороны вполне возможно среди моря заказных СуперЭВМ есть и Лисп-машины, например для корпоративного стратегического планирования или исследований в области искусственного интеллекта.

Основной механизм Лиспа – инкапсулированная в список определяющая голова списка и подключённый к ней хвост списка, который рекурсивно также может быть списком. Лисп-машина способна воспринимать каждый поступающий на неё список на самом абстрактном уровне, например как мета-Лисп-машину, модифицирующую воспринимающую машину. В такой динамичной, высокоабстрактной среде можно реализовать как строго-научные системы, так и неисчислимое множество программистских трюков и генераторов всевозможных машин.

Любая программа на Лиспе состоит из последовательности *выражений* (форм). Результат работы программы состоит в вычислении этих выражений. Все выражения записываются в виде *списков* — одной из основных структур Лиспа, поэтому они могут легко быть созданы посредством самого языка. Это позволяет создавать программы, изменяющие другие программы или макросы, позволяющие существенно расширить возможности языка.

Внешне исходный код программы на Лиспе отличается обилием круглых скобок; редактирование программ значительно упрощается использованием текстового редактора, поддерживающего автоматическое выравнивание кода, подсветку соответствующих пар скобок и такие специальные команды, как «перейти через список вправо» и т. д.

Под словами «Лисп-редактор» скрывается технология Emacs текстовых процессоров / лексических анализаторов и много других фундаментальных технологических идей, вошедших в операционную среду Линукс (Особенно Debian). Содержательна также история EMACS: она связана с одним из лидеров свободного программного обеспечения Столлманом.

Список является последовательностью элементов любого рода, в том числе других списков. Например, (1 3/7 'foo #'+) состоит из целого числа, рациональной дроби, *символа* foo и указателя на функцию сложения. Выражения представляются списками в префиксной записи: первый элемент должен быть *формой*, то есть *функцией*, *оператором*, *макросом* или *специальным оператором*; прочие элементы суть аргументы, передаваемые форме для обработки. Функция list возвращает список состоящий из её аргументов: например, (list 1 3/7 'foo #'+) возвращает список, упомянутый ранее. Если некоторые элементы являются выражениями, то сначала вычисляется их значение: (list 1 2 (list 1 2)) возвращает (1 2 (1 2)). Арифметические операторы действуют так же, например (+ 4 (* 2 3)) выдаёт 10.

Специальные операторы позволяют управлять последовательностью вычислений. С их помощью реализуются ветвления и циклы. Оператор if позволяет вычислить одно из двух выражений в зависимости от выполнения условия, которое тоже является выражением. Если его результат не nil, то вычисляется первый аргумент, иначе вычисляется второй. Например, (if nil (list 1 2 "foo") (list 3 4 "bar")) всегда возвращает (3 4 "bar").

Символьная природа языка (то есть отсутствие в символьном пространстве традиционной метрической геометрии расстояний, последовательностей и т. д.) позволяет легко и продуктивно распараллеливать Лисп-процессы. Что нашло использование в сверхмощных телекоммуникационных, сетевых Лисп-системах.

Пример Куайн (программы, выводящей свой исходный код) на Лиспе:

```
((lambda (x) (list x (list 'quote x))) '(lambda (x) (list x (list 'quote x))))
```

Данная программа должна работать на большинстве диалектов Лиспа, в том числе и на Scheme.

Итеративная версия функции определения N-го числа Фибоначчи с использованием макроса Loop:

```
(defun fibonacci (n)
  (loop repeat n
    for a = 1 then b
    and b = 1 then (+ a b)
    finally (return a)))
```

Пролог. Разработка языка Prolog началась в 1970 г. Аланом Кулмероз и Филиппом Русселом. Они хотели создать язык, который мог бы делать логические заключения на основе заданного текста. Название Prolog является сокращением от "PROgramming in LOGic". Этот язык был разработан в Марселе в 1972 г. Принцип резолюции Ковальского, сотрудника Эдинбургского университета, казался подходящей моделью, на основе которой можно было разработать механизм логических выводов. С ограничением резолюции на дизъюнкты Хорна унификация привела к эффективной системе, где неустранимый недетерминизм обрабатывался с помощью процесса отката, который мог быть легко реализован. Алгоритм резолюции позволял создать выполняемую последовательность, необходимую для реализации спецификаций, подобных приведенному выше отношению flight. Первая реализация языка Prolog с использованием компилятора Вирта ALGOL-W была закончена в 1972 г., а основы современного языка были заложены в 1973 г. Использование языка Prolog постепенно распространялось среди тех, кто занимался логическим программированием, в основном благодаря личным контактам, а не через коммерциализацию продукта. В настоящее время существует несколько различных, но довольно похожих между собой версий. Хотя стандарта языка Prolog не существует, однако версия, разработанная в Эдинбургском университете, стала наиболее широко используемым вариантом. Недостаток разработок эффективных приложений Prolog сдерживал его распространение вплоть до 1980 г.

Программа на языке Prolog состоит из набора фактов, определенных отношений между объектами данных (фактами) и набором правил (образцами отношений между объектами базы данных). Эти факты и правила вводятся в базу данных через операцию consult. Для работы программы пользователь должен ввести запрос - набор термов, которые все должны быть истинны. Факты и правила из базы данных используются для определения

того, какие подстановки для переменных в запросе (называемые унификацией) согласуются с информацией в базе данных. Язык Prolog, как интерпретатор, приглашает пользователя вводить информацию. Пользователь набирает запрос или имя функции. Выводится значение (истина - yes, или ложь - no) этого запроса, а также возможные значения переменных запроса, присвоение которых делает запрос истинным (то есть унифицирует запрос). Если ввести символ ";", тогда отображается следующий набор значений переменных, унифицирующий запрос, и так до тех пор, пока не исчерпается весь набор возможных подстановок, после чего Prolog печатает no и ждет следующего запроса. Возврат каретки воспринимается как прекращение поиска дополнительных решений.

Хотя выполнение программы на языке Prolog основывается на спецификации предикатов, оно напоминает выполнение программ на аппликативных языках LISP или ML. Разработка правил языка Prolog требует того же рекурсивного мышления, что и разработка программ на этих аппликативных языках. Язык Prolog имеет простые синтаксис и семантику. Поскольку он ищет отношения между некоторыми рядами объектов, основными структурами данных являются переменная и список. Правило ведет себя подобно процедуре, за исключением того, что концепция унификации более сложна, чем относительно простой процесс подстановки параметров в выражения.

Smalltalk (произносится [смол'ток]) – объектно-ориентированный язык программирования с динамической типизацией, разработанный в Хероу PARC Аланом Кэйем, Дэном Ингаллсом, Тедом Кэглера, Адель Голдберг, и другими в 1970-х годах. Язык был представлен как Smalltalk-80 и с тех пор широко используется. Smalltalk продолжает активно развиваться и собирает вокруг себя преданное сообщество пользователей.

Smalltalk оказал большое влияние на развитие многих других языков, таких как: Objective-C, Actor, Java, Groovy и Ruby. Многие идеи 1980-х и 1990-х по написанию программ появились в сообществе Smalltalk. К ним можно отнести рефакторинг, шаблоны проектирования (применительно к ПО), карты Класс-Обязанности-Взаимодействие и экстремальное программирование в целом. Основатель концепции Wiki, Вард Каннингем, также входит в сообщество Smalltalk.

Основными идеями Smalltalk'а являются:

- Всё – объекты. Строки, целые числа, логические значения, определения классов, блоки кода, стеки, память – всё представляется в виде объектов. Выполнение программы состоит из посылок сообщений между объектами. Любое сообщение может быть послано любому объекту; объект-получатель определяет, является ли это сообщение правильным, и что надо сделать, чтобы его обработать.

- Всё доступно для изменения. Если вы хотите изменить интегрированную среду разработки, вы можете сделать это в работающей системе, без остановки, перекомпиляции и перезапуска. Если вам необходима в языке новая управляющая структура, вы можете добавить её. В

некоторых реализациях вы можете также изменить синтаксис языка или способ работы сборщика мусора.

- Динамическая типизация — это означает, что вы не указываете типы переменных в программе, что делает язык гораздо лаконичней. (Как объяснено выше, является ли операция правильной, определяет объект-получатель, а не компилятор).

- Model-view-controller (MVC) шаблон структуры пользовательского интерфейса. (В последнее время используют и другие концепции реализации пользовательского интерфейса – например, Morphic в Squeak и Pollock в VisualWorks).

- Dynamic translation: современные коммерческие виртуальные машины компилируют байткоды в машинные коды для быстрого выполнения.

Smalltalk также использует другие современные идеи:

- Сборка мусора встроена в язык и незаметна разработчику.
- Программы Smalltalk'а обычно компилируются в байткоды и выполняются виртуальной машиной (VM), что позволяет выполнять их на любом оборудовании, для которого существует VM.

Одной из неожиданных особенностей Smalltalk'а является то, что традиционные конструкции: if-then-else, for, while, и т. д. не являются частью языка. Все они реализованы с помощью объектов. Например, решение принимается с помощью послышки сообщения ifTrue: логическому объекту, и передаёт управление фрагменту текста если логическое значение истинно. Есть всего три конструкции:

- послышка сообщения объекту;
- присваивание объекта переменной;
- возвращение объекта из метода;

и несколько синтаксических конструкций для определения объектов-литералов и временных переменных.

Чтобы лучше понять, как работает механизм обмена сообщениями, можно представить каждый объект как веб-сервер, отвечающий на запросы. При этом, на запросы можно просто выдавать заранее предопределённый ответ, аналог этому – выдача веб-страницы, расположенной по определённому пути; можно перенаправить запрос-сообщение другому объекту, аналог – прокси-сервер; изменить запрос по определённым правилам, аналог – техника url rewriting. Если для реакции на сообщение нет предопределённого метода, то вызывается метод #doesNotUnderstand:, так же, как веб-сервер открывает страницу с сообщением об ошибке, если задан несуществующий путь к веб-странице.

Следующий пример, показывающий нахождение гласных в строке, иллюстрирует стиль Smalltalk'а. Символ | определяет переменные, : определяет параметры, а символы [и] можно, для начала, воспринимать, как аналог фигурных скобок { и } в C-подобных языках:

```
| aString vowels |  
aString := 'This is a string'.
```

```
vowels := aString select: [:aCharacter | aCharacter isVowel].
```

В последней строке посылается сообщение `select:` с аргументом (блоком кода). При этом вызывается метод `select:` из класса `Collection` (одного из предков класса `String`). Текст этого метода показан ниже:

```
select: aBlock  
| newCollection |  
newCollection := 1/>self species new.  
1/>self do: [:each |  
    (aBlock value: each)  
    ifTrue: [newCollection add: each]].  
^newCollection
```

Он осуществляет перебор своих элементов (это метод `do:`), выполняя переданный ему блок `aBlock` для каждой буквы; когда блок выполняется (в примере – `aCharacter isVowel`), он создаёт логическое значение, которому затем посылается `ifTrue:`. Если это значение `true`, то буква добавляется в возвращаемую строку. Из-за того что `select:` определён в абстрактном классе `Collection`, мы также можем использовать его так:

```
| rectangles aPoint |  
rectangles := OrderedCollection  
    with: (Rectangle left: 0 right: 10 top: 100 bottom: 200)  
    with: (Rectangle left: 10 right: 10 top: 110 bottom: 210).  
aPoint := Point x: 20 y: 20.  
collisions := rectangles select: [:aRect | aRect containsPoint: aPoint].
```

ЛИТЕРАТУРА

1. Аладьев В.З., Хунт Ю.А., Шишаков М.Л. Основы информатики. – М.: Филинь, 1999.
2. Заморин А.П. Этапы эволюции ЭВМ. – М.: Знание, 1987.
3. Каган Б.М. Электронные вычислительные машины и системы. – М.: Энергия, 1999.
4. Корнеев В.В. Параллельные вычислительные системы. – М.: Нолидж, 1999.
5. Лысиков Б.Г. Арифметические и логические основы цифровых автоматов: Учебник для вузов. – Мн.: Выш. шк., 1980.
6. Майерс Г. Архитектура современных ЭВМ: В 2 кн./ Пер. с англ.; Под ред. В.К. Потоцкого. – М.: Мир, 1985.
7. Майоров С.А., Новиков Г.И. Структура ЭВМ. – Л.: Машиностроение, 1979.
8. Перспективы развития вычислительной техники: Справ. пособие в 11 кн. Кн. 2. /Под ред. Ю.М. Смирнова. – М.: Высш. шк., 1989.
9. Перспективы развития вычислительной техники: Справ. пособие в 11 кн. Кн. 3. /Под ред. Ю.М. Смирнова.– М.: Высш. шк., 1989.
10. Сименс Дж. ЭВМ пятого поколения. Компьютеры 90-х годов. – М.: Финансы и статистика, 1986.
11. Шпаковский Г.И. Параллельные микропроцессоры для цифровой обработки сигналов и медиаданных. – Мн.: БГУ, 2000.
12. Фути К., Судзуки Н. Языки программирования и схемотехника СБИС: Пер. с япон. – М.: Мир, 1988.
13. Головкин Б.А. Параллельные вычислительные системы. – М.: Наука, 1980.
14. Голенков В.В. и др. Программирование в ассоциативных машинах. – Мн.: БГУИР, 2001.
15. Голенков В.В. и др. Представление и обработка знаний в графодинамических ассоциативных машинах. – Мн.: БГУИР, 2001.
16. Голенков В.В. и др. Семантическая модель сложноструктурированных баз данных и баз знаний: Учеб. пособие. – Мн.: БГУИР, 2004.
17. Хамахер К., Вранешич Э., Заки С. Организация ЭВМ. – СПб.: Питер, 2004.
18. Таненбаум Э. Архитектура компьютера. – СПб.: Питер, 2002.