Содержание

1. ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС В ЭВМ

Допустим, вы имеете программу, написанную на языке высокого уровня.

Может ли эта программа сразу выполняться на ЭВМ?

Конечно, не может. Для этого необходимо преобразовать программу из так называемого исходного кода - собственно текста программы на языке высокого уровня (ЯВУ) - в исполняемый код. Этот процесс выполняется обычно в несколько этапов.

На первом этапе работает так называемый транслятор. Он может быть выполнен в двух видах: компилятор или интерпретатор. Интерпретатор обычно осуществляет перевод строк программы на ЯВУ последовательно в ходе выполнения программы. Поэтому появление сообщения об ошибке в программе может произойти в самом конце программы и вся работа ВМ пойдет насмарку.

Компилятор же осуществляет анализ всей программы, выделяет память для программы и требуемых переменных и оптимизирует код программы. Выявление ошибок в программе осуществляется еще до её выполнения. Поэтому компилятор обычно сложнее интерпретатора, но в работе предпочтительнее (работает быстрее на этапе выполнения программы и выявляет ошибки раньше, еще до выполнения программы).

Рис. 1.1. Получение исполняемого кода.

В результате работы компилятора образуется так называемый объектный модуль – код на языке ассемблера. Этот модуль обычно имеет расширение .obj и представляет собой приближенный к машинному языку код. В нем вместо обозначений в двоичном коде используются символьные имена переменных, символьные адреса и символьные обозначения команд. Строится таблица символов.

Затем следуют операции редактирования связей в объектном модуле – работает так называемый линкер, который связывает ваш объектный модуль с другими объектными модулями (используемыми в вашей программе функциями). В результате получается загрузочный модуль, наиболее близкий к выполняемому модулю.

Далее работает загрузчик «loader» который подключает к загрузочному модулю подпрограммы из статической системной библиотеки (имеющие расширение .lib). Эти подпрограммы загрузчик подгружает из библиотеки. Если статическая библиотека подгружается полностью, то программа сильно разбу-

хает в объеме. Поэтому применяют динамическую библиотеку, с подгружением подпрограмм только по мере их необходимости – только то, что будет практически использовать процессор в своей работе в ходе выполнения программы.

Иногда исполняемый код можно обработать и запомнить в формате .exe. С такими исполняемыми файлами следует обращаться очень осторожно, поскольку часто они содержат вредоносный код – вирусы, трояны и т.д., особенно, если эти программы взяты из непроверенных источников в интернете. Так, если вы скачиваете электронную книгу в формате pdf, то это безопасно, а если в формате .exe – опасно.

Часто процессы, выполняемые линкером и лоадером объединяют и обозначают процессом «линкование». Если программа имеет большой объем, то этот процесс может выполняться на ЭВМ довольно длительное время.

ЭВМ работает в двоичных кодах. Все, что относится к составлению программы и преобразованию ее в исполняемый код, относится к программированию. Поэтому мы при рассмотрении дисциплины «Архитектура ВМиС» будем рассматривать только работу в машинных кодах с использованием языка ассемблера для именования переменных и операций.

1.1. Команды ЭВМ

Команды ЭВМ могут быть самые разнообразные. Мы в первую очередь рассмотрим команды с двумя операндами типа сложение и умножение.

Что в общем случае входит в команду? Команда должна указывать:

1) Операцию, подлежащую выполнению;
2) Адреса операндов, над которыми производится операция (именно адреса, а не сами операнды!!!);
3) Адрес, по которому следует записать результат операции (адрес результата);
4) Адрес следующей выполняемой команды.

По сути, команда состоит из двух частей: кода операции и адресной части.

OP	Адресная часть

Рис. 1.2. Общий вид команды

1.2. Формат команды

Формат команды – это структура команды, то есть количество и назначение полей команды, а также разрядность каждого поля и команды в целом.

Для команды в процессоре отводится отдельный регистр – IR. По-

английски команда – это инструкция. По-русски этот регистр обозначается РК – регистр команды. Таким образом, в ЭВМ имеются два регистра для обращения к памяти – MAR – регистр адреса памяти, MDR – регистр данных памяти и теперь добавляем в ЭВМ еще регистр команд – IR.

Рассмотрим, как изменялись форматы команд при развитии ВМ.

1.3. История форматов команд

Вначале использовались команды, имеющие формат, показанный на рис.1.3. ЭВМ, использующие данный формат команд, назывались четырехадресными, в соответствии с количеством полей в адресной части формата команды.

OP	AD1	AD2	ADR	ADI
4 р-да	12 р-дов	12 р-дов	12 р-дов	12 р-дов

Рис.1.3. Формат 4-хадресной команды: OP- код операции, AD1, AD2 = адреса 1-го и 2-го операндов, ADR – адрес результата, ADI – адрес следующей команды.

Рассмотрим некоторую учебную ЭВМ, которая содержит 16 операций и память объемом 4К ячеек. Следовательно, для кодирования операции достаточно 4-х разрядов ($2^4 = 16$ команд), а для адресации любой ячейки данной памяти требуется 12 разрядов двоичного кода ($4К = 2^{12} = 4096$ ячеек памяти).

Таким образом, учитывая, что поле OP содержит 4 разряда, а все 4 адресных поля команды содержат по 12 разрядов, получаем 52 разряда - общую разрядность 4-хадресной команды учебной ЭВМ.

Если учесть, что для формирования одного разряда в регистре требуется один триггер на двух активных элементах, а в первых ЭВМ использовались электровакуумные лампы, потребляющие каждая 50 Вт, то для функционирования только одного регистра команд в 4-хадресной ЭВМ требовалось 5200 Вт (равносильно 5 электроутюгам). Отсюда пришли к выводу, что длинные регистры – это плохо. За счет чего можно уменьшить длину регистра?

Команды в памяти ВМ располагаются, как правило, последовательно друг за другом. Поэтому из состава команды убрали один адрес – адрес следующей команды – и заменили его специальным регистром – счетчиком команд. После выполнения очередной команды к нему автоматически добавляется единица, и он указывает адрес следующей команды. Этот регистр назвали PC – программный счетчик. Затем, при появлении микропроцессоров и появлении первых персональных компьютеров фирмы Intel, аббревиатуру PC стали использовать для обозначения персонального компьютера. Для регистра счетчика команд

стали использовать IP (точка инструкции). Соответственно, в русском языке для обозначения данного регистра используют аббревиатуры СК (счетчик команд), САК (счетчик адреса команды) или УК (указатель команды). Вы должны знать, что применительно к регистру адреса команды это одно и то же.

В результате удаления одного адресного поля команда стала трехадресной (рис.1.4) и стала содержать 40 разрядов.

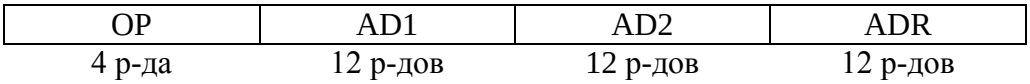


Рис.1.4. Формат 3-хадресной команды: OP- код операции,
AD1, AD2 = адреса 1-го и 2-го операндов, ADR – адрес результата

Следовательно, регистр стал потреблять меньше электроэнергии, получили экономию в количестве аппаратуры.

В дальнейшем в фирме IBM придумали переход на двухадресные команды. Выяснено, что в 70% случаев результат может заменить один из операндов, поскольку тот более не участвует в вычислениях. Естественно, при этом усложнился компилятор, поскольку в оставшихся 30% случаев необходимо сохранять значение второго операнда для последующих вычислений. Это действие возложили на компилятор, чтобы не изменять программы. В результате команда приобрела вид, показанный на рис.1.5. причем второе адресное поле содержит до операции адрес второго операнда, а после выполнения операции – адрес результата.

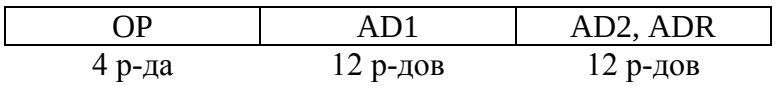


Рис.1.5. Формат 3-хадресной команды: OP- код операции,
AD1, AD2 = адреса 1-го и 2-го операндов, ADR – адрес результата

Итого в регистре команд осталось 28 разрядов. Но аппетит приходит во время еды. Задумались, нельзя ли обойтись только одним адресом операнда? Провели исследования, которые показали, что в большинстве случаев результат предыдущего вычисления используется как операнд следующего вычисления. Поэтому в архитектуру процессора добавили еще один регистр – аккумулятор, который использовался как адрес первого операнда и как адрес результата операции (рис.1.6).

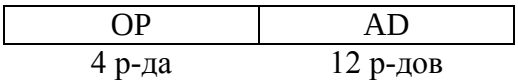


Рис.1.6. Формат 2-хадресной команды: OP- код операции, AD адрес операнда

При этом команда выполняется по алгоритму:

$$AC \leftarrow AC \ \Theta \ M[AD],$$

где AC – аккумулятор, Θ - любая операция ВМ, M[AD] – содержимое памяти по адресу AD.

Общее количество разрядов команды уменьшилось до 16.

Наконец, сделали следующий шаг. Придумали совсем безадресные команды. Для этого применили особую структуру памяти – стек. Стек – это такая структура памяти, которая растет вверх и используется сверху по принципу последним пришел – первым вышел. (LIFO – last input first output или FILO - first input last output) (рис.1.7).

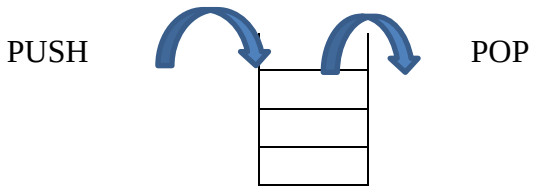


Рис.1.7. Стековая структура

Для обращения со стеком используются специальные команды: PUSH – положить в стек, и POP – взять из стека.

1.4. Примеры выполнения простой программы на ЭВМ

Рассмотрим примеры выполнения простой программы на ЭВМ с различными форматами команд. В качестве примера возьмем вычисление функции

Q=B*C + D*E.

Это выражение содержит две операции умножения и одну операцию сложения.

Предположим, что программа, выполняющая данную функцию, находится в памяти, начиная с адреса 11. Для 4-хадресной ЭВМ данная программа может быть представлена следующей последовательностью команд:

11 MUL B, C, T1, 12 / T1← B*C

12 MUL D, E, T2, 13 / T2← D*E

13 ADD T1, T2, Q, 14/ Q←T1+ = B*C + D*E

Здесь B, C, D, E, Q – адреса ячеек памяти, где хранятся соответствующие числа. T1 и T2 – адреса вспомогательных ячеек памяти.

Для удобства восприятия команд необходимо в тексте программы оставлять комментарии, которые пишутся в строках программы после символа / (слэш). Комментарии могут быть достаточно объемными.

Поскольку последний адрес в команде – адрес следующей выполняемой

команды, то в нашем случае, когда команды программы расположены в памяти ЭВМ последовательно, можно обойтись без этого адреса, заменив поле команды регистром счетчика команд, который будет наращивать свое значение на 1 после каждой выполненной команды.

Поэтому, исключив последнее поле в командах, для 3-хадресной ЭВМ имеем следующий текст программы:

11 MUL B, C, T1 / T1← B*C

12 MUL D, E, T2 / T2← D*E

13 ADD T1, T2, Q / Q←T1+ = B*C + D*E

Для двухадресной ЭВМ, которая имеет формат команды, приведенный на рис.1.5, имеем следующее:

MUL B, C / C ← B*C

MUL D, E / E ← D*E

ADD C, E / E←C + E

MOV E,Q / Q←E

К слову сказать, некоторые варианты ВМ размещают результат по адресу не второго, а первого операнда:

OP	AD1 ADR	AD2

Но это различие не принципиально.

Для одноадресной ЭВМ один из операндов перед исполнением команды должен находиться в аккумуляторе. Для этого вводится понятие загрузки аккумулятора. Вводится команда LOAD, LD или специально для загрузки аккумулятора LDA. Также вводится команда выгрузки из аккумулятора «запомнить» - STORE, ST или для аккумулятора STA. Тогда имеем:

LDA B / AC ← B

MUL C / AC← B*C

STA T1 / T1 ← AC

LDA D / AC ← D

MUL E / AC ← AC*D

ADD T1 / AC ← AC + T1 = B*C + D*E

STA Q / Q ← AC = B*C + D*E

Как видим, при уменьшении длины формата команды программа увеличивается в количестве слов, требуемых для выполнения той же функции. Это и понятно, ведь при уменьшении информативности команды (она становится короче) длина программы увеличивается.

Для стековой ЭВМ имеются одноадресные команды загрузки операндов в

стек и выгрузки результата, а также безадресные команды выполнения операций.

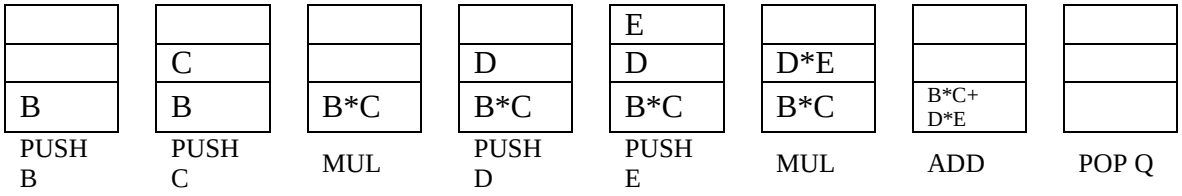


Рис.1.8. Пример выполнения вычислений на стековой ВМ

Сравним вычисления на ВМ с различной разрядностью.

Таблица 1.1

Адресность команды	Длина команды, разрядов	Кол-во строк программы	Длина программы, бит
4	52	3	156
3	40	3	120
2	28	4	112
1	16	7	112
стек	4 16	3 5	92

Анализ таблицы позволяет сказать, что при уменьшении количества полей в команде уменьшается общая длина программы, но увеличивается количество строк программы, поскольку информативность команды падает.

2. АРХИТЕКТУРА ВМ

Рассмотрение архитектуры ВМ можно проводить, по крайней мере, двумя путями. Либо рассмотреть конкретную ВМ и строить выводы для других ВМ. Либо рассматривать по порядку, как в учебнике. Будем рассматривать архитектуру ВМ на примере некоторой учебной ВМ, и потом можно будет легче усвоить, как построена любая другая ВМ.

2.1. Логическое проектирование учебной ВМ

Для построения структуры ВМ необходимо выполнить ряд этапов.

1) Выбор системы команд.

Для примера выберем одноадресную ВМ.

2) Разрядность ВМ.

Обычно разрядность выбирают кратной байту – восьми битам. 8 бит для ВМ уже слишком мало. Современные ВМ обычно 32-х или 64-х разрядные. Для упрощения рассмотрения выберем разрядность слова в ВМ 16.

3) Зададим объем памяти ВМ.

Наша ВМ будет универсальная, т.е. предназначенная для широкого круга

разнообразных задач. Первые ЭВМ имели память 4К слов. Поэтому для учебной ВМ мы тоже выберем такую память. Естественно, современные ЭВМ имеют память Гигабайт и более.

4) Определимся с разрядностью регистров.

Регистр адреса памяти MAR в соответствии с объемом памяти 4К слов (4К = 2^{12}) должен иметь 12 разрядов.

Регистр данных памяти MDR в соответствии с разрядностью слова ВМ должен иметь 16 разрядов.

Формат команды содержит два поля – поле кода операции и поле адреса операнда. Выберем количество команд в нашей учебной ВМ 16. Тогда поле кода операции будет содержать 4 разряда, а поле адреса – 12 разрядов. Всего длина команды соответствует 16 разрядам.

5) Счетчик команд должен иметь возможность адресации во всем объеме памяти ВМ, поэтому содержит 12 разрядов.

6) Регистр команды IR должен содержать саму команду – 16 разрядов.

7) Аккумулятор – соответствует разрядности слова в ВМ – 16 разрядов.

8) Таким образом, в составе ВМ будут иметься регистры: PC, MAR, MDR, IR, AC. Кроме того – регистры ввода и вывода информации InpR и OutR – они обычно длиной 1 байт. Также имеется АЛУ.

С составом ВМ определились. Как соединены составные части ВМ определим позже.

Таблица 2.1. Таблица команд ВМ.

№ и обозначение команды	Команда	Код команды		Адрес	Расшифровка команды
		2	16		
0 q_0	LDA	0000	0	AD	AC←M[AD]
1 q_1	STA	0001	1	AD	M[AD] ← AC
2 q_2	ADD	0010	2	AD	AC← AC+ M[AD]
3 q_3	SUB	0011	3	AD	AC← AC- M[AD]
4 q_4	AND	0100	4	AD	AC← AC∧M[AD]
5 q_5	OR	0101	5	AD	AC← AC∨M[AD]
6 q_6	XOR	0110	6	AD	AC← AC$\oplus$M[AD]
7 q_7	COM	0111	7	AD	AC← $M[AD]$ (обратный код)
8 q_8	NEG	1000	8	AD	AC← $M[AD] + 1$(доп. код)
9 q_9	INC	1001	9	AD	AC←M[AD]+1
10 q_{10}	DEC	1010	A	AD	AC←M[AD]-1
11 q_{11}	JMP	1011	B	AD	PC←AD (переход)
12 q_{12}	CLEA	1100	C	AD=FFF*	AC←0
13 q_{13}	INCA	1101	D	AD=FFF*	AC← AC+1
14 q_{14}	INP	1110	E	AD=FFF*	AC← INPR (ввод)
15 q_{15}	OUT	1111	F	AD=FFF*	OUTR ← AC (вывод)

Примечание *: AD=FFF означает заполнение единицами адресного поля.

Рассмотрим, как расшифровывается содержимое регистра команд.

Пусть IR = 0011 0001 0010 $1000_2 = 3128_{16}$

Первое поле длиной 4 разряда – это код команды. В нашем примере команда 3 – это операция вычитания SUB. Остальные 12 разрядов – это адрес ячейки памяти, где находится второй операнд. Первый операнд уже должен находиться в аккумуляторе.

2.2. Центральный процессор

Центральный процессор (ЦП) – главный элемент ВМ, который реализует выполнение команд.

В состав ЦП входят регистры:

PC – счетчик команд;

IR – регистр команды;

AC – аккумулятор;

MAR – регистр адреса памяти;

MDR – регистр данных памяти;

InpR, OutR – регистры ввода и вывода данных;

АЛУ – арифметико-логическое устройство;

M – память.

Все эти устройства входят в состав устройства управления (УУ) и операционного устройства (ОУ) ЦП. Рассмотрим их подробнее.

2.3. ОУ ЦП учебной ВМ

Представим восемь схем полных сумматоров SM с различными входными данными.

S1	S0			C_0			C_0
		A	“0”		A	“0”	
0	0	SM		0	SM		1
		A			A+1		
		A	B		A	B	
0	1	SM		0	SM		1
		A+B			A+B+1		
		A	B		A	B	
1	0	SM		0	SM		1
		A+*B*=A-B-1			A+*B* + 1=A-B		
		A	“1”		A	“1”	
1	1	SM		0	SM		1
		A-1			A		

Здесь “0” и “1” означают развёрнутый ноль и развернутую единицу, то есть ноль или единицу, содержащиеся в каждом разряде слова ВМ. S1 и S0 – это управляющие разряды, характеризующие операцию, которую выполняет

сумматор, C_0 – перенос в младший разряд сумматора.

На представленных схемах на первый вход сумматоров всегда подается одно и то же число А. На самом деле это не всегда так, что мы покажем впоследствии.

Для усвоения смысла преобразований в АЛУ рассмотрим работу двух небольших цифровых схем.

Цифровой вентиль

Пусть некоторая схема имеет два входа и один выход. Причем если на входе S имеется ноль, то на выходе тоже имеется ноль, если на входе S имеется 1, то выход схемы соответствует входу D.

S	D	Y
0	0	0
0	1	0
1	0	0
1	1	1

Фактически эта схема представляет собой известную схему «И». Она представляет собой цифровой вентиль, где один из входов (S) представляет собой разрешающий вход, а другой (D)- информационный.

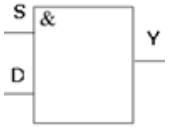


Рис. 2.1. Схема И.

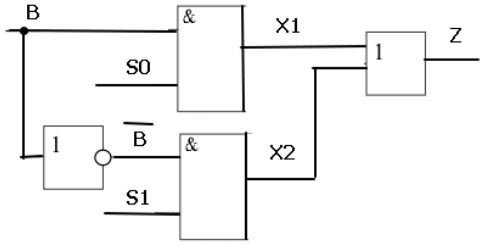


Рис. 2.2. Схема №2.

Таблицу состояний схемы сразу сокращаем, учитывая, что цифровые вентили пропускают на выход либо сигнал B либо его инверсию в зависимости от состояния управляющих входов S0 и S1.

S1	S0	X1	X2	Z
0	0	0	0	0
0	1	B	0	B
1	0	0	$\overline{B}$	$\overline{B}$
1	1	B	$\overline{B}$	1

При расположении такой схемы в каждом разряде мы имеем выполнение

логических функций поразрядно. АЛУ должно обеспечивать выполнение как арифметических, так и логических функций. Условимся, что при значении 1 управляющего сигнала М будут выполняться логические функции, а при значении 0 – арифметические. В качестве логических функций будем рассматривать функции ИЛИ, исключающее ИЛИ, И и инверсию.

Таким образом, АЛУ учебной ВМ выполняет следующие микрооперации в зависимости от значений управляющих входов:

M	S1	S0	C	Микрооперации	Наименование
0	0	0	0	A	Передача A
0	0	0	1	A+1	Инкремент A
0	0	1	0	A+B	Сумма
0	0	1	1	A+B+1	Сумма с инкрементом
0	1	0	0	A-B-1	Разность с декрементом
0	1	0	1	A-B	Разность
0	1	1	0	A-1	Декремент A
0	1	1	1	A	Передача A
1	0	0	X	A∨B	ИЛИ
1	0	1	X	A@B	Искл. ИЛИ
1	1	0	X	A∧B	И
1	1	1	X	$\overline{A}$	Не A

С составом ОУ ЦП мы определились ранее. Надо рассмотреть, каким образом все это соединено.

Прежде всего, вспомним сведения, известные из курса информатики.

Дешифратор

Это устройство, которое имеет единицу только на одном из выходов, соответствующему коду входных информационных сигналов.

Дешифратор на схемах обозначается следующим образом:

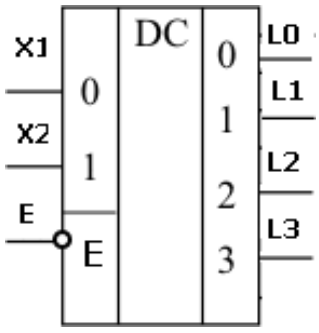


Таблица соответствия для двухвходового дешифратора имеет вид:

E	X2	X1	L0	L1	L2	L3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0

Вход E означает разрешение работы дешифратора. Этот вход обычно делают инверсным. Если разрешение есть, то E=0, и один из выходов в соответствии с кодом входных информационных сигналов дешифратора равен 1. Если E=1, то независимо от состояния входных сигналов все выходы дешифратора равны 0.

Мультиплексор

Из многих информационных входов I0…I3 мультиплексор подает на выход один, соответствующий коду, установленному на управляющих входах a1, a0. Структурная схема мультиплексора имеет вид:

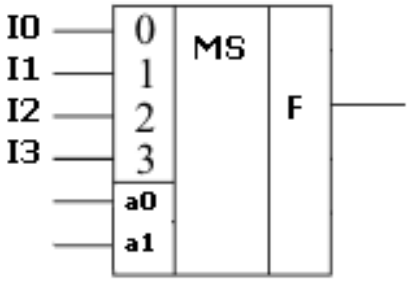
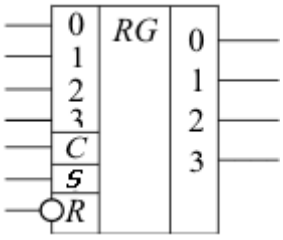


Таблица соответствия:

a1	a0	F
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Регистр

Регистры обычно строятся на D триггерах. Обозначение регистра на схемах может иметь вид:



На вход C подается синхросигнал. Регистр может срабатывать по переднему или по заднему фронту синхроимпульса. Вход S – управляющий. При S=0 регистр осуществляет хранение информации. При S=1 регистр переписывает свое состояние по синхросигналу в соответствии с состоянием входных информационных сигналов.

Синхросигнал обычно подается в виде импульса. При этом регистр изменяет свое состояние по переднему (возрастающему) или по заднему (спадаю-

щему) фронту импульса. В первом случае вход С обозначается с наклонным штрихом слева-снизу направо-вверх, во втором – со штрихом слева-сверху направо-вниз.

Рассмотрим схему центрального процессора (операционное устройство) в 2 этапа. Сначала рассмотрим упрощенную схему. Затем усложним её.

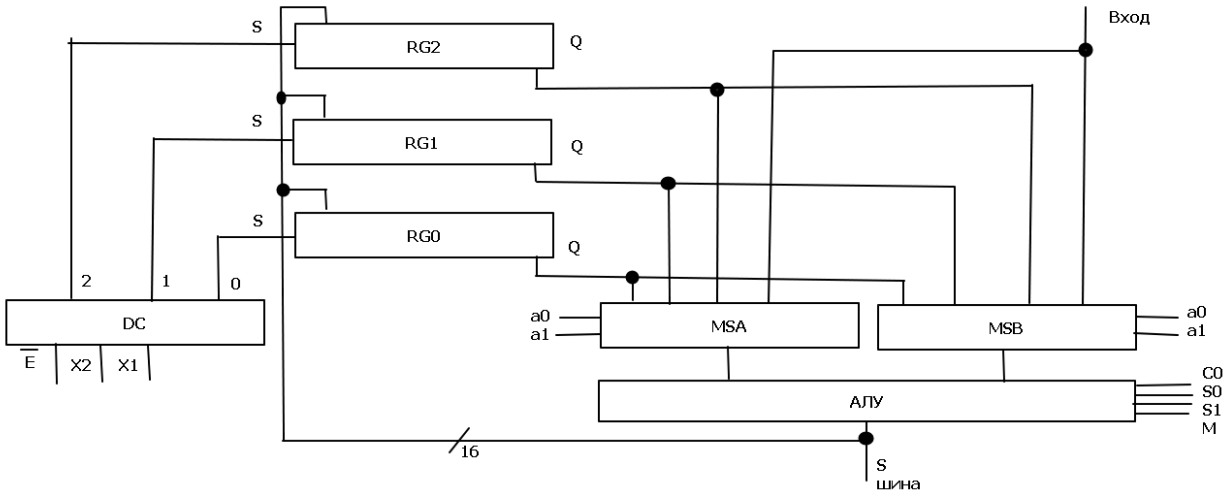


Рис. 2.3. Схема вспомогательная

Подачей управляющих сигналов на дешифратор, мультиплексоры и АЛУ ведает устройство управления ЦП, которое выдает так называемое управляющее слово процессора. Оно состоит из разрядов управления этими устройствами. – по 2 разряда на мультиплексоры MSA и MSB, 3 разряда на дешифратор DS и 4 разряда на АЛУ.

MSA		MSB		АЛУ				DC		
a0	a1	a0	a1	M	S1	S0	C0	E	X2	X1

Рассмотрим примеры.

1.Пусть надо в RG1 поместить сумму чисел из RG2 и RG0:

RG1 ← RG2 + RG0

10	00	0010	001

2. RG2 ←вход

11	XX	0000	010

3. RG0 ← RG2 ∧ вход

10	00	110X	000

4. S ← RG1

01	XX	1111	1XX

Рассмотрим более полный вариант схемы ЦП.

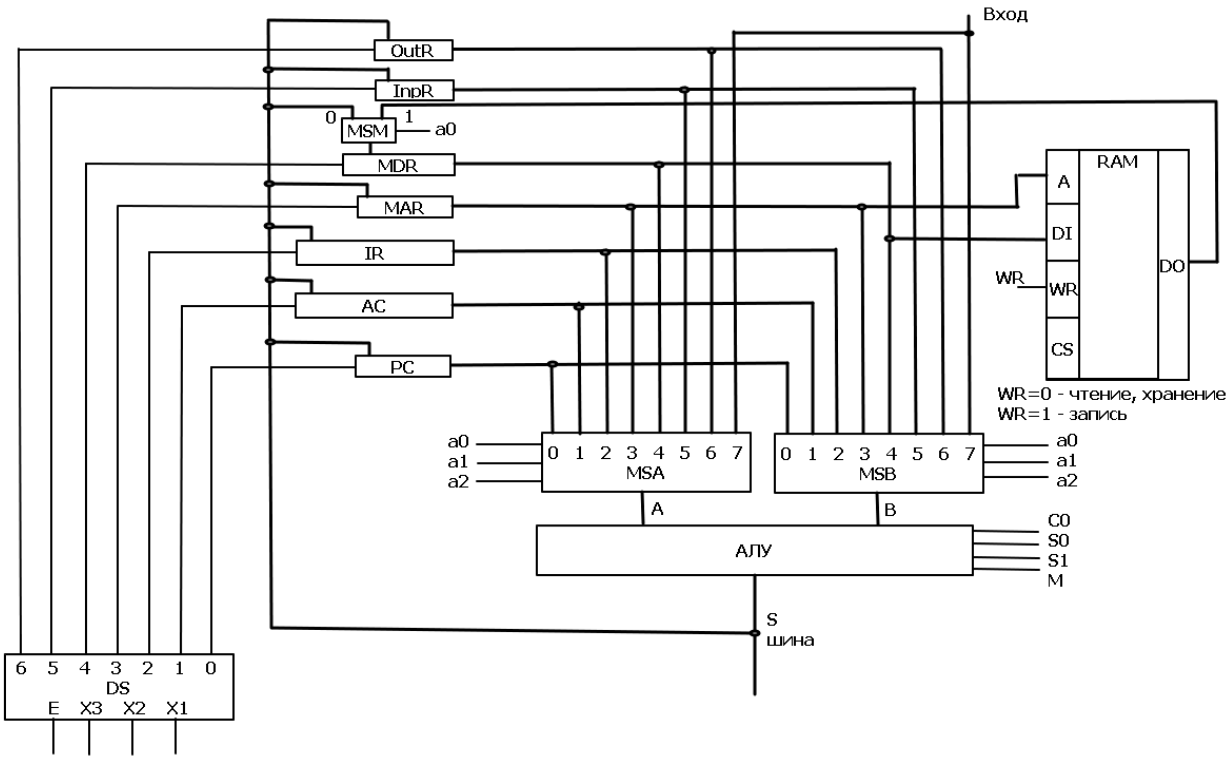


Рис. 2.4. Схема ОУ ЦП

Управляющее слово ВМ будет составлено из управляющих разрядов MSA, MSB, АЛУ, DS, MSM, WR.

MSA			MSB			АЛУ				DC				MSM	WR
a0	**a1**	**a2**	**a0**	**a1**	**a2**	**M**	**S1**	**S0**	**C0**	**E**	**X2**	**X1**	**X0**	**a0**	WR

Рассмотрим примеры управляющего слова ЦП для некоторых операций:

AC ←AC + MDR

MSA	MSB	АЛУ	DC	MSM	WR
001	100	0010	0001	x	0

MAR ← PC

MSA	MSB	АЛУ	DC	MSM	WR
000	xxx	0000	0011	x	0

MDR ← M[MAR]

MSA	MSB	АЛУ	DC	MSM	WR
xxx	xxx	xxxx	0100	1	0

Пустой такт NOP

MSA	MSB	АЛУ	DC	MSM	WR
xxx	xxx	xxxx	1xxx	x	0

Таким образом, мы спроектировали схему операционного устройства центрального процессора учебной ВМ.

2.4. Устройство управления ЦП

Задача устройства управления – обеспечить выдачу управляющего слова ЦП. Иначе он называется микровход, который управляет работой аппаратуры

ЦП. Обычно он записывается в постоянное запоминающее устройство (ПЗУ) центрального процессора (прошивается).

Что должно делать УУ? У нас есть исполняемый код программы. Дальше в ходе выполнения исполняемого кода на ВМ им занимается операционная система – программа, которая выдаёт команды для УУ ЦП.

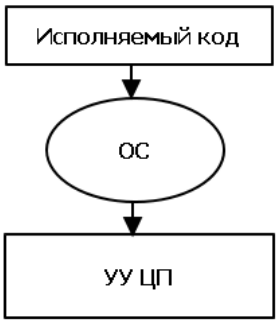


Рис. 2.5. Алгоритм управления в ВМ

Последовательность работы устройства управления задается циклами.

Для начала рассмотрим цикл выборки команды. Ведь прежде чем выполнить команду, надо выбрать её из памяти ВМ.

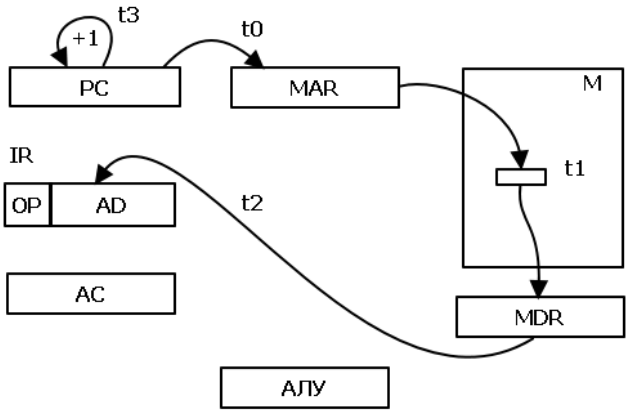


Рис. 2.6. Цикл выборки команды

Для выборки команды из памяти ВМ надо, во-первых, поместить адрес команды из регистра адреса команды в регистр адреса памяти, во-вторых, прочитать команду из ячейки памяти в регистр данных памяти, в-третьих. Переслать команду в регистр команды, в-четвертых, подготовить счетчик команд для выборки следующей команды – инкрементировать PC.

То есть, последовательно надо выполнить 4 действия, обозначенные на рис. 2.6 t0, t1, t2, t3. Следовательно, цикл выборки команды осуществляется за 4 такта.

Само выполнение команды происходит следующим образом:

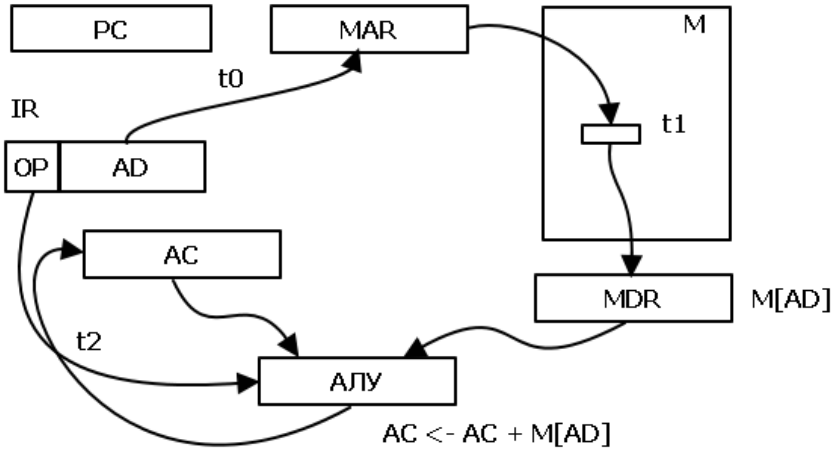


Рис. 2.7. Цикл выполнения команды

Рассмотрим цикл выполнения команды на примере операции сложения.

Нулевой такт – пересылка адреса операнда из регистра команды в регистр адреса памяти. Первый такт – выборка из памяти операнда и размещение его в регистре данных памяти. Второй такт – собственно выполнение сложения в АЛУ с размещением результата в аккумуляторе.

Здесь мы рассматриваем выполнение команды с прямой адресацией, когда в адресном поле команды находится адрес операнда.

Машину Фон Неймана стали совершенствовать, поскольку для выполнения некоторых команд более удобными являются другие методы адресации операндов. Всего способов адресации насчитывается около 20. Из них в каждой конкретной ВМ используется только часть, по выбору конструкторов ВМ и в соответствии с назначением ВМ (ориентированием ВМ на обработку определённых видов информации).

Допустим, что у нас в памяти содержится такая информация:

Адрес	Содержание
30	40
40	50
50	60

Рассмотрим выполнение команды LDA (загрузку аккумулятора) с помощью различных методов адресации.

Прямая адресация: LDA 30 /загрузить в аккумулятор слово из ячейки с адресом 30. В результате выполнения команды AC ← 40.

Непосредственная адресация: LDA # 30. /загрузить в аккумулятор слово «30». В результате выполнения команды AC ← 30. То есть в команде содержится сам операнд, и загрузка его из памяти не требуется.

Косвенная адресация: LDA @ 30. /загрузить в аккумулятор слово из ячей-

ки, адрес которой находится в ячейке 30. В результате выполнения команды АС ← 50. То есть в команде содержится не адрес операнда, а адрес адреса операнда. Соответственно, для подготовки операции требуется двойное обращение к памяти.

Существуют и другие способы адресации, которые здесь мы не будем рассматривать.

В общем случае для указания способа адресации в команде выделяется отдельное поле, содержащее необходимое количество разрядов. Для декодирования способа адресации при этом необходим ещё один цикл – цикл дешифрации команды.

Таким образом, мы получили наличие в УУ ЦП следующих циклов:

1) Цикл выборки команды – С0;
2) Цикл дешифрации команды – С1;
3) Цикл выполнения команды – С2;
4) Цикл обработки прерывания – С3.

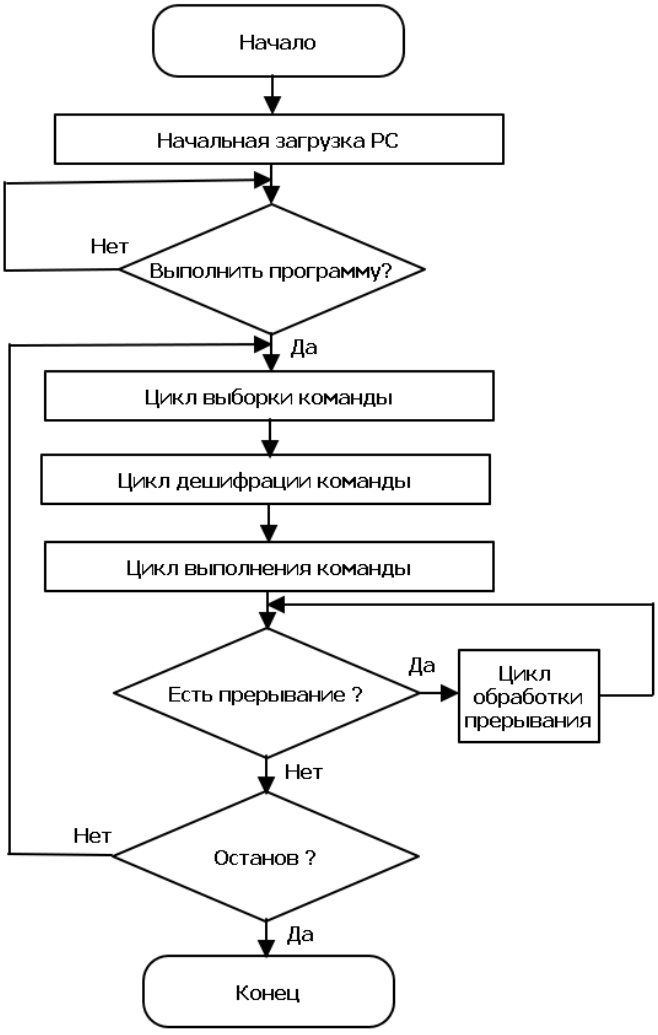


Рис. 2.8. Алгоритм работы устройства управления ЦП ВМ

Последний цикл необходим для обработки требований прерывания, вырабатываемых внешними устройствами ВМ (например, клавиатурой, устройством ввода/вывода и т.д.) или при появлении нештатных ситуаций при выполнении команд (деление на ноль, переполнение разрядов и др.). Как происходит цикл обработки прерывания мы рассмотрим позднее – при изучении операционных систем.

На основании изложенного алгоритм работы УУ ЦП имеет вид, представленный на рис. 2.8.

Как видите, ВМ реагирует на прерывание только после окончания выполнения очередной команды программы. В противном случае очень трудно запомнить текущее состояние ЦП для возврата в программу после обработки прерывания.

2.5. Структурная схема УУ ЦП

Структурная схема УУ ЦП изображена на рис. 2.9.

В основе УУ имеется ПЗУ; сигналы, вырабатываемые на его выходе, управляют работой ВМ. Генератор ВМ вырабатывает тактовую частоту. Сигнал SE представляет собой сигнал разрешения работы и останов. При SE=0 сигналы генератора ВМ не проходят через схему совпадения, и схема управления не работает – останов. При SE=1 сигналы генератора ВМ проходят через схему совпадения и поступают на вход двухразрядного счетчика, соединённого по выходу с дешифратором тактов. Логика работы последнего приведена на рис. 2.10.

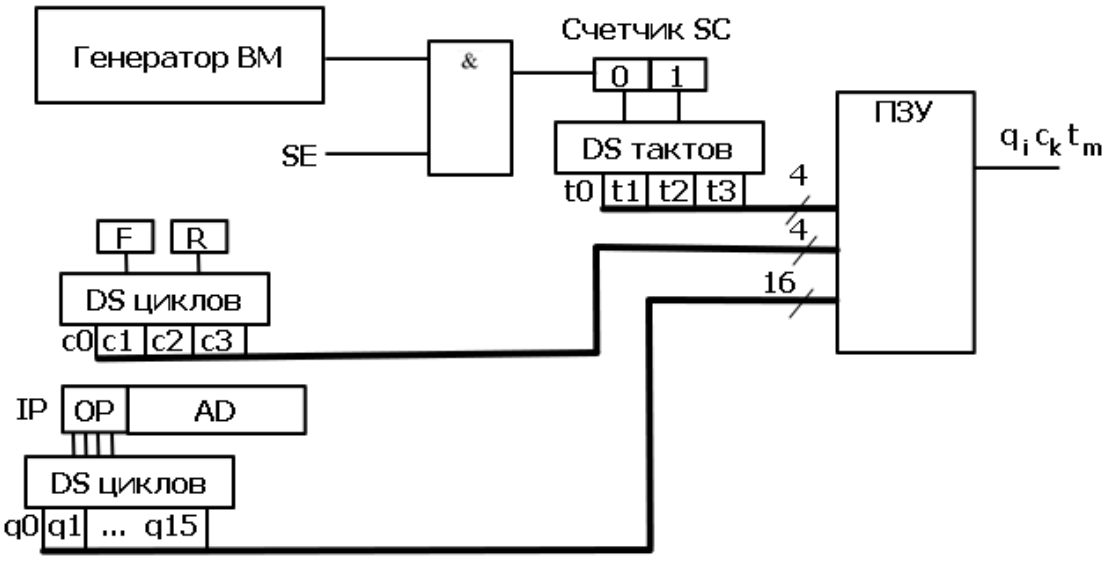


Рис. 2.9. Структурная схема УУ ЦП

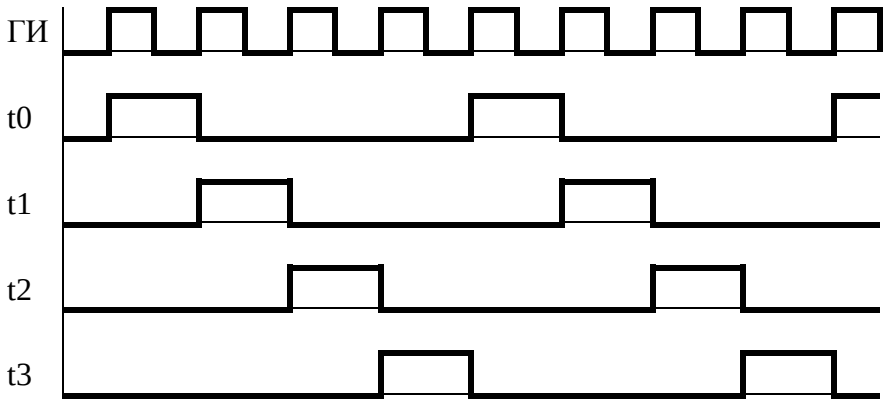


Рис. 2.10. Временная диаграмма работы DS тактов

В ВМ такты не пересекаются по времени.

DS циклов работает по выходам двух триггеров F и R. В зависимости от состояния этих триггеров формируется сигнал на соответствующем его выходе.

F	R	Цикл	Обозначение цикла
0	0	Выборки команды	C_0
0	1	Дешифрации команды	C_1
1	0	Выполнения команды	C_2
1	1	Прерывания	C_3

DS команды работает по состоянию разрядов регистра команд, соответствующих полю кода операции. В нашем случае код имеет 4 разряда, что соответствует 16 выходам дешифратора команд.

Выходы всех трех дешифраторов составляют адрес ПЗУ, в котором записана (прошита) информация соответствующая содержимому управляющего слова процессора данного такта данного цикла данной команды.

Такова в общем виде структура УУ ЦП.

Таблица команд учебной ВМ представлена в табл.2.1.

Обычно ВМ содержит команды различной длины – безадресные, одноадресные, двухадресные и др. В нашей учебной ВМ мы рассматриваем все команды единой длины равной 16 двоичным разрядам. Поэтому безадресные команды (с номерами от 12 до 15) содержат фиктивный адрес, равный FFF.

Рассмотрим для примера команду сложения ADD. Для неё в поле кода операции будет иметься значение 2 16-ричного кода. В адресном поле находится 12-разрядный адрес AD. Напишем микрокоманды циклов выборки и выполнения команды.

F=0, R=0, C_0=1

C_0t_0 : MAR← PC / подготовить выборку команды из памяти.

C_0t_1 : MDR← M[MAR] / прочесть команду из памяти по адресу MAR.

C_0t_2 : IR← MDR / поместить команду в регистр команды.

C_0t_3 : PC← PC +1, F← 1/ подготовить адрес след. команды, перейти к циклу выполнения команды.

F=1, R=0, C_2=1

C_2t_0 : MAR← IR[AD] / передать адрес операнда в регистр адреса памяти MAR.
C_2t_1 : MDR← M[MAR] / прочесть операнд из памяти по адресу MAR.
C_2t_2 : AC← AC+MDR / выполнить сложение.
C_2t_3 : F← 0/ перейти к циклу выборки команды.

Так выполняются микрокоманды ВМ в циклах выборки и выполнения команды сложения. Аналогичным образом можно описать порядок выполнения любой команды из перечня команд ВМ.

ВМ фон Неймана первоначально была рассчитана, как и наша учебная ВМ, только для одного типа адресации – прямой адресации. Для других типов адресации, что задаётся обычно в специальном поле команды, требуется наличие цикла дешифрации команды и осуществление выборки операндов соответствующим образом. Мы цикл дешифрации команды не рассматриваем, поскольку считаем, что используется только прямая адресация операндов.

Следующая контрольная работа состоит в том, чтобы вы научились строить команды из микрокоманд подобно рассмотренному примеру. Объем работы в контрольной работе велик. Для оформления работы надо использовать по крайней мере один лист формата А4.

Рассмотрим типовую задачу и приведём пример её решения.

Задача. По адресу 5A8 записана команда сложения ADD с адресом F12. По этому адресу записан операнд 731F. В регистре АС находится операнд 8721. Определить информацию, которая будет иметься в регистрах ВМ PC, MAR, MDR, IR, AC после выполнения данной команды.

Решение.

1. Определяем исходное состояние регистров.
2. Выписываем микрокоманды циклов выборки команды и выполнения команды.
3. Рассчитываем управляющие слова ВМ для каждой микрокоманды.
4. Определяем значения регистров после выполнения каждой микрокоманды.
5. Результаты оформляем в виде табл.2.2.

Таблица 2.2.

Управл. сигналы, регистры	MSA	MSB	АЛУ	DS	MSM	WR	FR	PC	MAR	MDR	IR	AC
Исх. состояние регистров								5A8				8721
C_0t_0 : MAR← PC	000	xxx	0000	0011	x	0	00		5A8			
C_0t_1 : MDR← M[MAR]	xxx	xxx	xxxx	0100	1	0	00			2F12		
C_0t_2 : IR← MDR	100	xxx	0000	0010	x	0	00				2F12	
C_0t_3: PC←PC+1, F←1	000	xxx	0001	0000	x	0	10	5A9				
$q_2C_2t_0$: MAR← IR[AD]	010	xxx	0000	0011	x	0	10		F12			
$q_2C_2t_1$: MDR← M[MAR]	xxx	xxx	xxxx	0100	1	0	10			731F		
$q_2C_2t_2$: AC← AC+MDR	001	100	0010	0001	x	0	10					FA40
$q_2C_2t_3$: F← 0	xxx	xxx	xxxx	1xxx	x	0	00					

Таким образом, мы рассмотрели архитектуру устройства управления и операционного устройства учебной ВМ и принципы ее работы.

В результате мы получили следующую структурную схему ВМ:

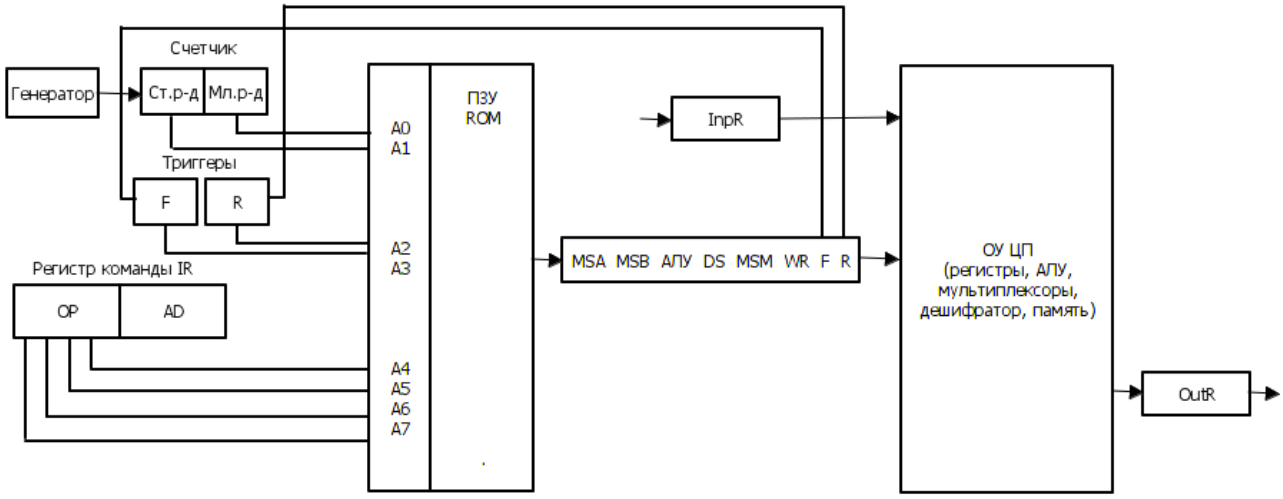


Рис. 2.11. Структурная схема ЦП.

Как видно из схемы, ПЗУ может работать и без дешифраторов на входе, то есть к адресным входам ПЗУ подключены непосредственно выходы счетчика тактов, выходы регистров F, R и выходы поля кода операции регистра команды IR. Благодаря этому адрес ПЗУ состоит из 8 разрядов, и ёмкость ПЗУ существенно уменьшается без потери функциональности.

На выходе ПЗУ имеется управляющее слово ЦП, состоящее из 18 разрядов. Это слово поступает на соответствующие управляющие входы устройств операционного устройства ВМ. На схеме отдельно выделены регистры ввода-

вывода, предназначенные для связи ВМ с внешним миром, а также показан регистр команды IR, который входит одновременно в состав УУ и ОУ ЦП ВМ.

3. АРХИТЕКТУРА ОПЕРАЦИОННЫХ СИСТЕМ

3.1. Терминология

В мире существует множество ВМ. Управляющие слова, непосредственно управляющие устройствами ВМ, называются микрокодом. Микрокод заносится в ПЗУ УУ ВМ. Этот процесс называется прошивкой. Этот термин исторически произошел от того, что ранее существовавшие ПЗУ основывались на ферромагнитных кольцах, характеризуемых хранением информации в зависимости от направления проводов, проходящих через кольца. На заводах по изготовлению ПЗУ были заняты в основном женщины, которые буквально с иглой в руках прошивали каждый бит (кольцо) проводами в заданном направлении. Отсюда и термин «прошивка». С тех пор изменилась элементная база ПЗУ, занесение информации в него производится другими способами, но термин программирования ПЗУ «прошивка» остался.

Следующий термин – опкод (операционный код). Это содержимое регистра команды, включающее код операции и адресную часть.

Таким образом, в архитектуре вычислительной машины выделяются несколько уровней:

1. Уровень цифровой логический – микросхемный, включающий микросхемы И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ, регистры, сумматоры, дешифраторы, мультиплексоры и т.д.
2. Уровень микроархитектуры – это построение ОУ ЦП, УУ ЦП, других крупных узлов процессора.
3. Уровень архитектуры набора команд.
4. Уровень операционной системы.
5. Уровень ассемблера – собственно машинного языка в символьных кодах, построения исполняемого кода с помощью соответствующих системных программ (транслятора, загрузчика, компоновщика и др.)

3.2. Операционная система

Первое, что делает ОС – это выделяет вашей исполняемой команде область в оперативной памяти ВМ. В этой области выделяются участки (сегменты), обладающие разными правами по доступу к ним и предназначенные для разных целей.

Рис.3.1. Разделы памяти процесса.

В первую очередь – это сегмент данных, поименованных в программе – так называемые глобальные переменные. Во-вторых, это сегмент исполняемого кода, который, как правило, не подлежит изменению в ходе выполнения программы. В-третьих, - это блок управления процессом (дескриптор или PCB) – основные сведения о процессе. В-четвертых, это стек, используемый для временного хранения данных при прерываниях программы. В-пятых, это так называемая куча, динамическая память процесса, где хранятся все временные переменные и другие данные, используемые в ходе вычислительного процесса.

Приведем пример. Пусть на нашей ВМ надо выполнить программу, вычисляющую формулу F=B+C.

Программа для вычисления в системе команд учебной ВМ выглядит в виде совокупности следующих операций:

LDA B
ADD C
STA F

После компиляции получаем исполнительный код, размещенный в памяти машины, например, с ячейки 010. Пусть данные размещены в памяти, начиная с ячейки 100. Тогда получаем следующее распределение памяти.

Адрес	Данные	Пояснение
010	0100	Исполняемый код
011	2101	
012	1102	
…		
100	B	Данные
101	C	
102	F	
…		

3.3. Место операционной системы в ВМ

Место ОС в ВМ поясняется рис.3.2. Нижний уровень А – это уровень аппаратуры. Выше размещена ОС. Еще выше размещается прикладное ПО, т.е. те программы, которые выполняют задачи пользователя.

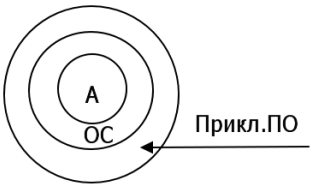


Рис.3.2. Место ОС в ВМ.

ОС имеет 2 лица: одно обращено к аппаратуре, другое – к нашей программе.

3.4. Задачи ОС

1-я задача ОС – скрыть от пользователя все детали аппаратуры и представить в ВМ в виде определенной рабочей среды – рабочего стола.

2 –я задача ОС (по отношению аппаратуре) – эффективное управление аппаратурой, всеми ресурсами в ВМ.

Какими ресурсами в ВМ мы располагаем? На ВМ работают, а правило, несколько программ одновременно. Каждая из них претендует на использование ресурсов ВМ. Первый ресурс – это время центрального процессора. Второй ресурс – оперативная память. Часто ее не хватает. Поэтому приходится организовывать взаимодействие с дисковой памятью. Третий ресурс – это система ввода-вывода. Ею надо эффективно управлять. Четвертый ресурс – файловая система, обеспечивающая хранение и пользование данными и программами на диске.

Диски в ВМ обозначаются буквами C, D, E … операционная система ка правило, размещается на диске C, и загружается при включении ВМ. Почему диски нумеруются не с буквы A? Раньше буквами A и B обозначались гибкие диски, на диске A размещалась ОС. Современные ОС по объему не вмещаются на гибкий магнитный диск.

3.5. Многослойная структура ОС.

Уровни ОС (слои) обозначаются в виде концентрических колец. (рис.3.3). На рис.3.3 обозначено:

1. Средства аппаратной поддержки.
2. Машинно-зависимые компоненты ОС.
3. Базовые механизмы ядра.
4. Менеджеры ресурсов (диспетчеры).
5. Интерфейс систем вызовов (API-функции).

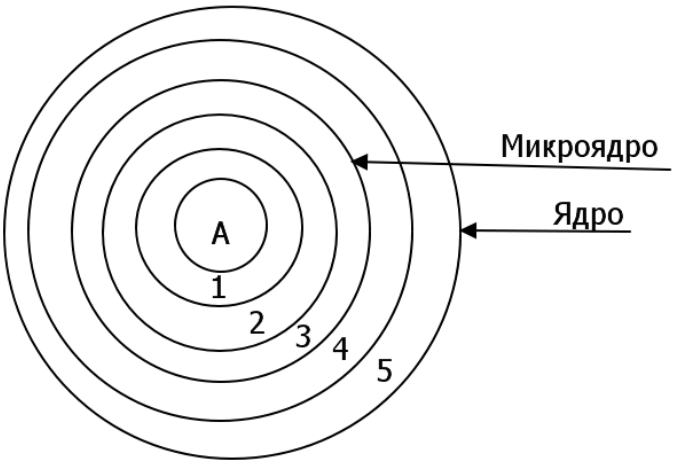


Рис.3.3. Многослойная структура ОС.

Задачи 1 и 2 слоев – скрыть детали аппаратуры, обеспечить идентичность работы программиста с аппаратурой разных изготовителей.

1, 2 и 3 уровни составляют микроядро ОС. Базовые механизмы ядра, а также уровни 1 и 2 не принимают никаких решений, а исполняют отдельные функции работы с аппаратурой.

На 4 уровне и выше уже принимаются решения. Каждый менеджер (памяти, устройств ввода- вывода и др.) ведет учет своих ресурсов, выделяет их при необходимости процессам, перераспределяет.

Самый верхний слой выполняет вызовы, запросы программ, так называемые системные вызовы. Раньше они назывались командами ОС.

Например, вызов ***read (fd, buffer, count)*** обозначает команду чтения из файла ***fd*** в область ***buffer*** количества слов ***count***. То есть читает буфер и образует кучу. Системные вызовы существенно влияют на работу программ.

В ядре выделяют микроядро – первые три слоя. Это связано с тем, что микроядро составляет неделимую часть ОС. Вместе с четвертым и пятым слоем оно образует ядро ОС.

С ядром взаимодействуют:

- - приложения пользователей;
- - библиотеки процедур;
- - утилиты - программы, реализующие отдельные задачи управления ОС;
- - системные программы (компилятор, загрузчик и др.).

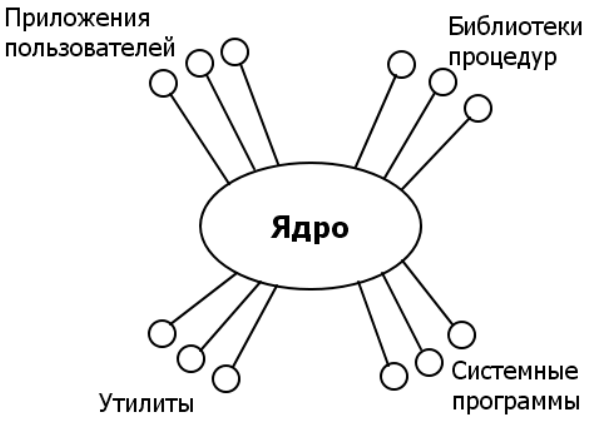


Рис.3.4. Окружение ОС.

Утилиты, например, обслуживают файловую систему, архивируют данные, генерируют случайные числа и т.д.

3.6. История ОС

Начало для создания ОС положила фирма Microsoft выпуском своей дисковой операционной системы MS-DOS. ОС была невелика по объёму и размещалась на гибком магнитном диске, вставляемом в дисковод А.

MS-DOS представляет собой монолитную ОС – ядро, ещё не распавшееся на слои. Ядро окружали согласно рис.3.4 программы пользователей (1), библиотеки подпрограмм (2), утилиты (3) и системные программы (4).

Работала ОС довольно плохо. Неприятности заключались в том, что при любом сбое программы пользователя или драйвера внешнего устройства ОС «рушилась». Приходилось снова загружать ОС и начинать работу вновь.

Стали искать пути выхода из этой ситуации. Для этого выделили пользовательский режим, куда поместили всё окружение (1 – 4), а для работы ядра ввели защищённый режим - привилегированный, так называемый режим ядра. В результате надежность работы ОС повысилась, но она стала работать медленнее. Ведь для вызова любой системной функции приходилось производить переключение из пользовательского режима в режим ядра и обратно, для чего требовалось затратить дополнительное время τ на каждое переключение.

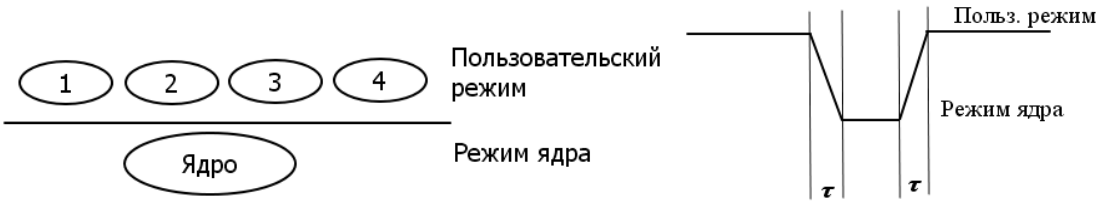


Рис.3.5. Выделение режимов пользовательского и защищенного.

Дальнейшим шагом в совершенствовании ОС стало введение микроядер-

ной архитектуры. Все драйвера вынесли в пользовательский режим. Те слои, которые раньше назывались менеджерами, стали называть серверами и тоже вынесли в пользовательский режим. Таким образом, в защищенном режиме осталось только микроядро.

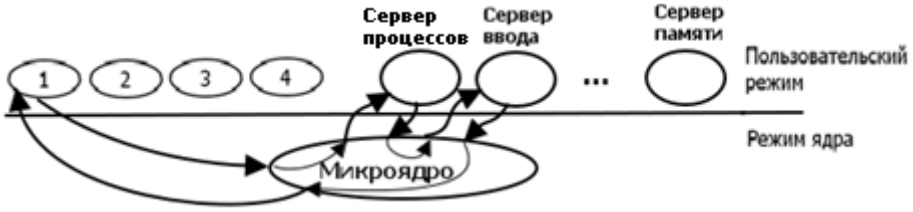


Рис.3.6. Работа микроядра.

Надежность ОС повысилась, но быстродействие еще более снизилось. Если пользовательская программа обращается к серверному процессу, который, в свою очередь, требует операции ввода вывода, то при каждом обращении к микроядру требуется переход от пользовательского к защищенному режиму и обратно, что требует дополнительных затрат времени. Таим образом при обращении к серверному процессу требуется затратить дополнительно 6τ времени.

В настоящее время микроядерный режим в чистом виде не используется из-за медленности его работы. Больше используют гибридный режим, в котором серверные процессы оставляют в микроядре, а сами драйвера выносят в пользовательский режим.

4. ПРОЦЕССЫ И ПОТОКИ

Процесс – это программа в ходе ее выполнения. Процессу выделяется память со своими разделами (рис.3.1) одним из разделов является блок управления процессом (БУП). Иначе его называют дескриптором процесса или объект-процессом.

4.1. Состояние процесса

Поскольку процессов в ВМ выполняется много, то каждый процесс большое время проводит в очередях. При этом он переходит из одного состояния в другое. Обобщенная схема включает в себя 5 состояний. Таких состояний может быть и больше. Например, в UNIX их 13.

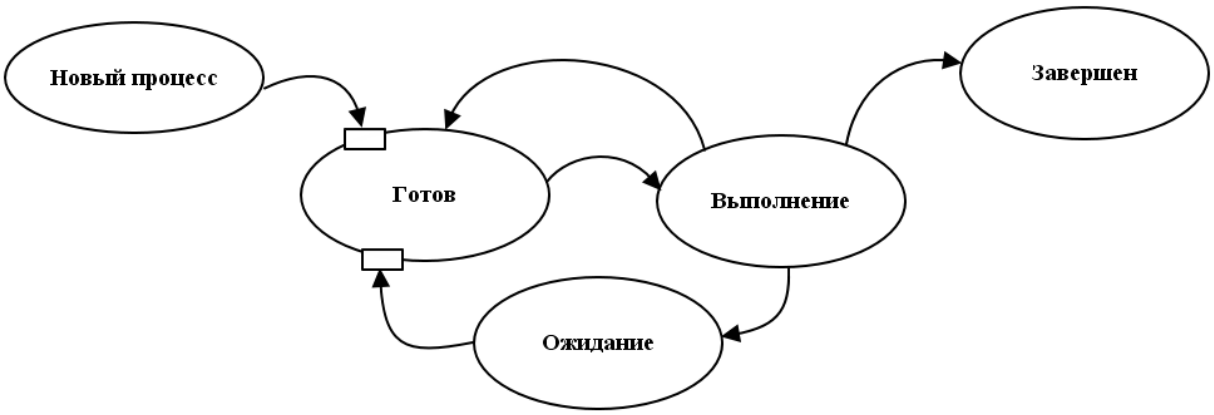


Рис.4.1. Схема состояний процесса.

Состояние «новый» - программа принята к исполнению, для нее выделена память с необходимыми разделами. Состояние «готов» - это очередь готовых процессов. Здесь работает диспетчер кратковременный планировщик. Его задача выбрать из списка готовых процессов 1 и перевести его в состояние выполнения. Ни один из процессов не может долго занимать центральный процессор. Поэтому каждому процессу выделяется квант времени, по окончанию которого процесс переводится в состояние готов – в очередь готовых процессов.

Еще один вариант – ожидание ввода- вывода. Здесь своя очередь процессов ожидающих некоторого события (реакции пользователя, освобождения устройства ввода – вывода и др.). после совершения события процесс снова переводится в состояние готов. Здесь работает долговременный диспетчер планировщик. Он отличается от кратковременного, управляющего выделением квантов времени, тем, что работает по своему алгоритму. Он может быть построен с учетом разных подходов (например, приоритетов).

Из чего состоит БУП?

- PID – идентификатор процесса;
- состояние процесса;
- указатель на родительский процесс, если процесс порожден другим процессом;
- счетчик команд;
- регистры ЦП, используемые процессом;
- ввод – вывод, указывает, откуда процесс загружен и куда выводит информацию;
- текущие параметры (насколько загружен ЦП, как используется куча ит.д.).

Рассмотрим, как производится переключение одного процесса на другой.

Пусть всего имеется два процесса, чередующиеся в фазе выполнения.

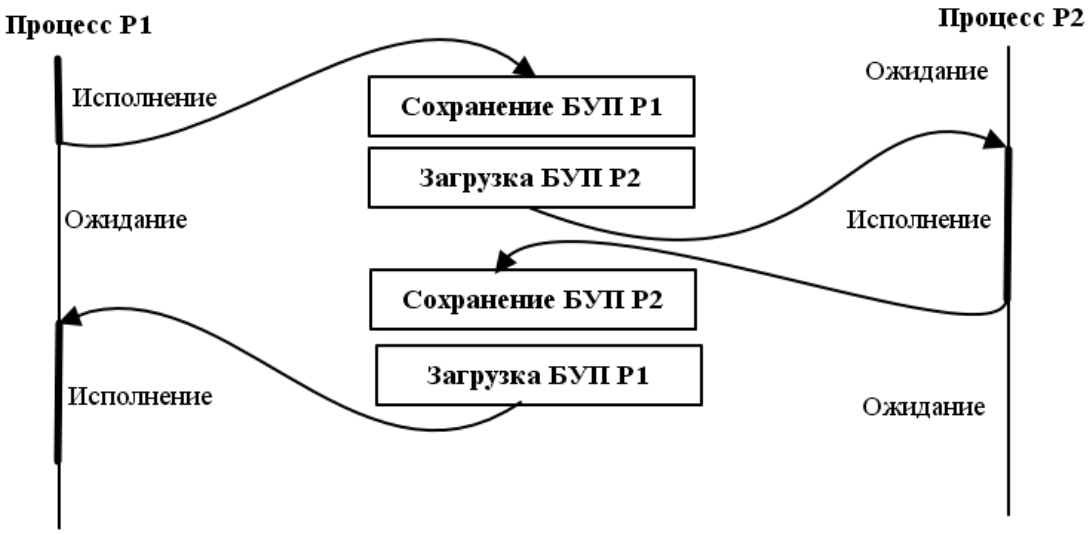


Рис.4.2. Переключение процессов.

В простейшем случае переключение процессов иллюстрируется рис. 4.2. На самом деле процедура переключения процессов сложнее. По существу, есть периоды времени, когда не выполняется ни один из процессов.

4.2. Потоки

При выполнении какого-либо процесса в нем могут присутствовать ветви, способные к параллельному выполнению. Например, при умножении матриц параллельно можно перемножать различные элементы матриц.

Ранее вычислительный процесс рассматривался как последовательное выполнение команд, и повышение быстродействия достигалось, в основном, путем повышения тактовой частоты генератора ВМ. В 2004 году фирма Intel пришла к выводу, что предел повышения тактовой частоты скоро будет достигнут, и повышать быстродействие ВМ следует путем разработки многоядерных процессоров и применения распараллеливания операций.

В первых параллельных алгоритмах над задачей распараллеливания работали программисты. Естественно, это приводило к тому, что одни процессоры были перезагружены, а другие простаивали без работы. Выход нашли путем перекладывания задач распараллеливания операций на ОС, применяя, в том числе языки высокого уровня (ЯВУ), ориентированные на многопоточную обработку данных.

В настоящее время все ЯВУ делятся на однопоточные и многопоточные. К однопоточным относятся ранние ЯВУ (Паскаль, Фортран, Си, Си++ в первых своих редакциях).

Многопоточные ЯВУ стали разрабатываться ещё до появления многоядер-

ных процессоров. К таким языкам относится, прежде всего, Java. Он с самого начала был многопоточным. Так же многопоточными являются C#, Python, а также серия ЯВУ с расширением .Net: Visual C++ Net, Visual Basic Net. Эти языки уже при компиляции используют многопоточную модель вычислений.

Как ОС использует многопоточность?

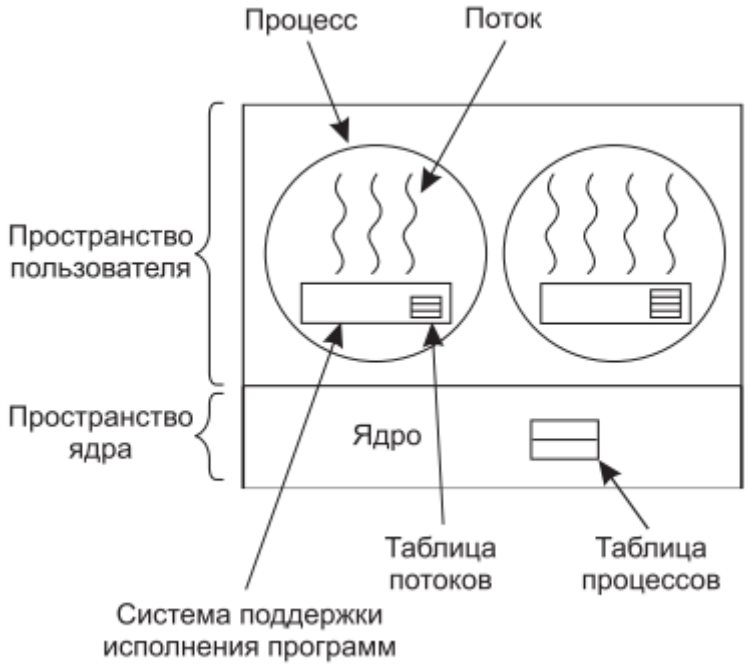


Рис.4.3. Управление потоками в процессах.

На рис. 4.3 приведена модель управления потоками со стороны процессов, выполняемых на ВМ. Показано, что ядро производит управление процессами 1 и 2, ничего не зная о потоках, созданных внутри этих процессов. Первый процесс создал три потока, второй – четыре потока. Все эти потоки выполняются лишь тогда, когда выполняется соответствующий процесс. Выполнение потоков происходит лишь на том процессоре, который выделен ОС соответствующему процессу. Внутри процессов имеются управляющие модули, которые создают таблицу потоков. Эти же модули и управляют выполнением потоков внутри процесса. Выполнение потоков и процессов осуществляется в пространстве пользователя, ядро выполняется в пространстве ядра в привилегированном режиме.

Надо отметить, что по-английски поток обозначается термином Thread, то есть «нить». Это обозначение в некотором смысле более приемлемо, поскольку оно более отличается от термина процесс и позволяет, в свою очередь, разделить поток на более мелкие составляющие – «волокна», что часто и делается ОС в настоящее время.

Чем хороши потоки по сравнению с процессами? Они используют то же адресное пространство, что и процессы, не требуют дополнительных накладных расходов при переходе от одного потока (нити) к другому потоку одного и того же процесса.

Положительным свойством управления потоками в пользовательском пространстве является то, что процессы и потоки выполняются быстро, не переходя в режим ядра. Кроме того, такой механизм можно реализовать в любой ОС.

Недостатком такого подхода является то, что все потоки одного процесса могут выполняться только на том ядре, которое ОС выделила данному процессу.

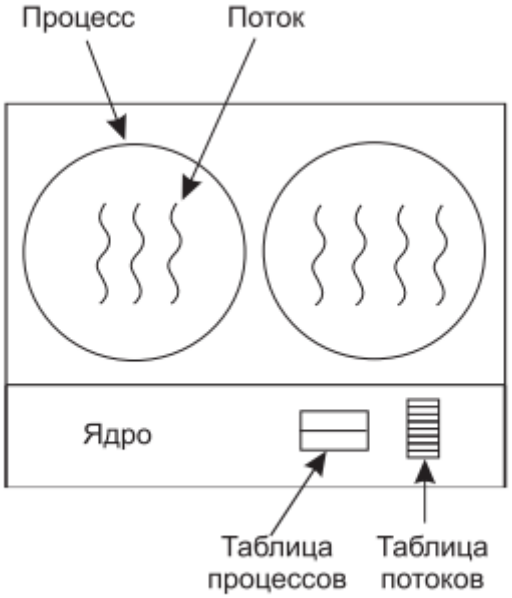


Рис.4.4. Управление потоками в ядре.

Более современный подход представлен на рис.4.4. Здесь управление потоками и процессами осуществляется в ядре ОС. То есть ядро знает о потоках каждого процесса и привлекается к их планированию и управлению ими. Такой подход позволяет смешивать управление потоками и процессами, в том числе выделять потокам одного процесса разные процессоры.

Достоинством такого подхода является то, что любой поток может выполняться на любом ядре процессора, что существенно повышает эффективность использования аппаратуры и приводит к повышению быстродействия. Недостатком такого подхода является то, что и процессы и потоки обращаются к ядру ОС, при этом увеличиваются накладные расходы на переключение пользовательского и защищенного режимов.

Вариантов осуществления управления потоками и процессами очень много. Есть очень много сложностей и неприятностей при практическом осуществ-

лении конкретных вариантов реализации управления.

Задачи, ориентированные для выполнения на многоядерных процессорах, хуже выполняются на процессорах с малым количеством ядер. Поэтому в общем случае число потоков и количество ядер процессора должно быть согласовано. Программы на более совершенные процессоры следует оптимизировать.

В настоящее время идет интенсивная работа по поиску возможных путей решения более сложных задач. В частности, рассматривается возможность для создания потоками подчиненных потоков следующего уровня, то есть разделять «нити» на отдельные «волокна».

5. УПРАВЛЕНИЕ ПАМЯТЬЮ

По характеру использования разделяют память на виртуальную, физическую, страничную и др. Дело в том, что обычно оперативной памяти не хватает для исполняемых на ВМ процессов.

В качестве первого выхода из этого положения предложили так называемые оверлеи. Программист сам разбивал программу на части – оверлеи, очередной оверлей загружался в ОП после выполнения предыдущего оверлея. Такой подход частично решал проблему с нехваткой оперативной памяти, но создавал трудности для программистов.

Вторым выходом из положения стало использование виртуальной памяти с возложением на операционную систему задач, связанных с загрузкой в ОП необходимых для работы процессов частей виртуальной памяти с диска. При этом решалась задача обеспечения для программистов прозрачности работы с памятью. Вся работа с загрузкой в ОП частей процесса с диска и выгрузкой обратно выполнялась диспетчером памяти ОС и становилась для программистов невидимой.

Виртуальная память – это совокупность программно-аппаратных средств, позволяющих пользователю писать и выполнять на ВМ программы, объём которых превосходит имеющуюся в ВМ оперативную (физическую) память. Для этого делают следующее:

- в исходном состоянии все данные и программный код размещают на диске;
- ОС при выполнении процесса перемещает данные и программный код между диском и ОП частями таким образом, чтобы пользователь этого не замечал;
- пользователь все время работает как бы в виртуальной памяти большого объема.

Как это осуществляется на практике?

Для этого существует три метода управления памятью: страничное распределение, сегментное распределение и сегментно-страничное.

5.1. Страничное распределение.

На рис.5.1 приведена схема взаимодействия виртуальной и физической памяти. В диске данные хранятся на дорожках секторами, размер которых равен 512 байт. Поэтому размер страницы выбирают кратным этому значению: 1К, 2К, 4К, 8К. В настоящее время преимущественно используют размер страницы 4К или 8К.

Пример.

Пусть внешняя память (диск) имеет объем 1Мбайт = 2^{20} байт. Физическая память имеет объем 16 Кбайт = 2^{14} байт. Страница имеет объем 4 Кбайт = 2^{12} байт. Соответственно, для адресации виртуальной памяти требуется 20 разрядов, физической памяти - 14 разрядов, смещения внутри страницы - 12 разрядов. Кроме того, отсюда следует, что физическая память содержит $2^2 = 4$ страницы, а виртуальная – $2^8 = 256$ страниц.

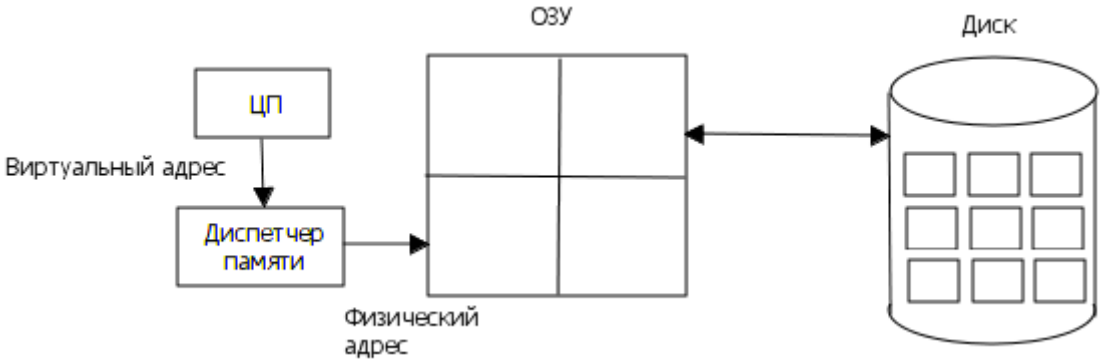


Рис.5.1. Взаимодействие виртуальной и физической памяти.

Виртуальный адрес состоит из двух частей: 8 старших разрядов – номер страницы, 12 младших разрядов – смещение в странице. Физический адрес тоже состоит из двух частей: 2 старших разряда – номер страницы, 12 младших разрядов – смещение в странице.

Пусть процессору потребовалось обратиться к байту на 9-й странице со смещением 2049. В физической памяти свободна страница номер 2, куда и будет переписана страница из виртуальной памяти. Надо определить виртуальный и физический адреса требуемого байта.

Для решения задачи надо построить шаблоны двух частей виртуального и физического адресов, преобразовать номера страниц виртуальной и физической памяти в двоичный код и записать их в свои места шаблонов, заполнив недостающие старшие разряды нулями. Затем преобразовать смещение в двоичный

код и записать его в соответствующую часть шаблонов адреса, тоже при необходимости заполнив недостающие старшие разряды нулями.

В результате получим:

- виртуальный адрес 00001001 100000000001.

- физический адрес 10 100000000001.

Для преобразования виртуального адреса в физический диспетчер памяти ОС строит страничную таблицу:

№ вирт. стр.	V	R	M	A	№ физ. стр.	Карта диска
0	0					
1	0					
…						
$1001=9_{10}$	1				$10=2_{10}$	3 дор.2 сект.
…						
$2^8-1=255_{10}$						

В таблице обозначено:

V – признак присутствия виртуальной страницы в ОП;

R – признак использования страницы. Обычно – несколько разрядов. При каждом обращении к странице значение инкрементируется. Чем больше R, тем активнее идет обращение к странице и тем важнее оставить ее в ОП при необходимости вытеснения страницы на диск;

M – признак модификации страницы. Равен 1, если страница модифицировалась, т.е. в нее производилась запись данных. При вытеснении такой страницы её необходимо сохранить на диск прежде записи на это место другой страницы;

A – признак прав доступа к странице. Обычно составляет 2 разряда.

Правила преобразования виртуального адреса в физический при страничном распределении памяти поясняется рис.5.2.

Обращение к таблице страниц может быть достигнуто быстрее при использовании буфера быстрого преобразования адреса TLB (Translation Lookaside Buffer). Если страничную таблицу хранить в памяти, то при каждом обращении к памяти надо обратиться прежде всего к таблице, т.е. требуется дополнительное обращение к памяти. Поэтому наиболее используемую часть страничной таблицы размещают в ассоциативной памяти, дорогой, но малой емкости. При этом поиск в таблице производится по определенным признакам и осуществляется до 10 раз быстрее, чем в обычной памяти.

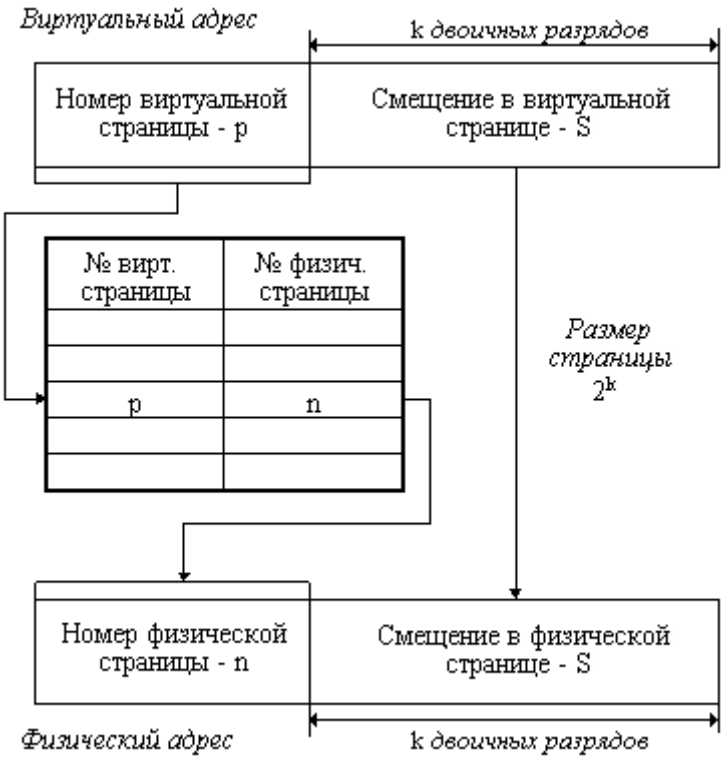


Рис.5.2. Преобразование виртуального адреса в физический при страничной организации памяти.

Кстати, точно так же ассоциативно производится поиск в памяти переводчиком с иностранных языков. Переводчик имеет в памяти готовые переводы наиболее встречаемых фраз и выдает готовый перевод, если запрос имеется в памяти. Только в случае отсутствия нужной информации в памяти переводчик пословно переводит фразу на другой язык.

Положительными качествами рассмотренного страничного распределения памяти являются его простота и быстродействие. Недостатками являются недостаточный учет прав доступа к информации, которая передается из диска в ОП и обратно одинаковыми страницами.

Для устранения этого недостатка применяют сегментное распределение памяти.

5.2. Сегментное распределение памяти

Программиста интересует, в первую очередь, назначение информации, представленной в памяти сегментами с различными правами доступа. Поэтому сегментное распределение предусматривает передачу информации между диском и ОП целыми сегментами с одинаковыми правами доступа. При этом возможны ситуации, когда один и тот же сегмент (например, подпрограмма) используется одновременно двумя и более процессами. Поэтому не обязательно

иметь много копий соответствующего кода в памяти каждого процесса, а достаточно иметь один такой сегмент в ОП и обращаться к нему из разных процессов по мере необходимости.

В случае сегментного распределения памяти для каждого процесса создается таблица сегментов, которая имеет сведения о размере сегмента, нахождении его в ОП и управляющую информацию, позволяющую определять, какой сегмент следует выгрузить из ОП при необходимости загрузки нового сегмента.

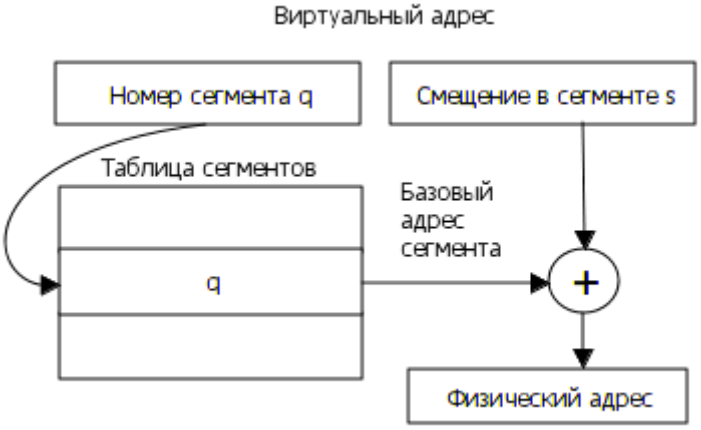


Рис.5.3. Преобразование виртуального адреса при сегментной организации памяти.

Фактически таблица сегментов аналогична таблице страниц при страничном распределении памяти. Отличие заключается в том, что сегменты имеют различный размер и находятся в виртуальной памяти не обязательно с начала страницы. Поэтому при вычислении физического адреса необходимо производить арифметические вычисления, учитывая базовый адрес сегмента в ОП и смещение внутри сегмента. Эти вычисления требуют больше времени по сравнению с вычислением физического адреса при страничном распределении, которое осуществляется посредством конкатенации номера страницы и смещения внутри страницы.

Кроме того, недостатком сегментного распределения является эффект фрагментации ОП ввиду неодинаковости размера сегментов.

Способом устранения этих недостатков является переход к сегментно-страничному распределению памяти.

5.3. Сегментно-страничное распределение памяти

Сегментно-страничное распределение памяти совмещает достоинства обоих вышерассмотренных способов распределения памяти. При этом виртуальный адрес состоит из трех составляющих: номера сегмента q, номера страницы

p и смещения внутри страницы s. Перемещение информации между диском и ОП осуществляется страницами одинакового размера. Для вычисления физического адреса используются две таблицы: таблица сегментов и таблица страниц сегмента.

Схема определения физического адреса показана на рис. 5.4.

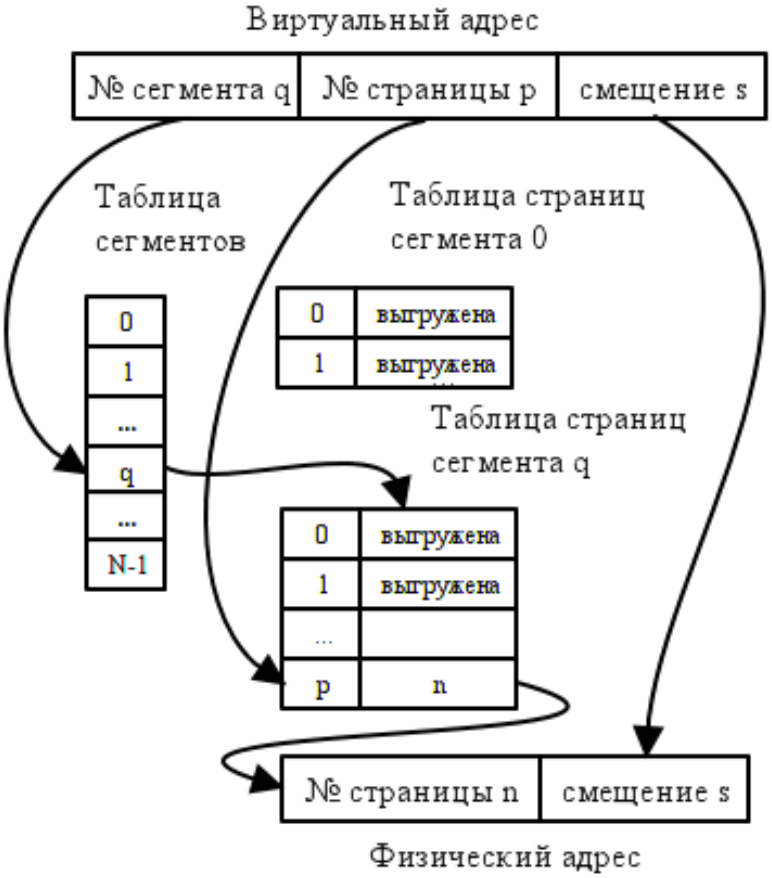


Рис.5.4. Вычисление адресов при сегментно-страничном распределении памяти.

Фрагментация памяти при таком распределении существенно уменьшается, поскольку передача информации осуществляется страницами одинакового размера, и неполной может быть только последняя страница сегмента. Кроме того, упрощается вычисление адресов, которое аналогично страничному распределению, но требует двойного обращения к памяти (используются две таблицы). По номеру сегмента выбирается таблица страниц сегмента, по номеру страницы в этой таблице выбирается номер страницы в физической памяти. Смещение внутри страницы виртуальной и физической памяти совпадают, поэтому используется операция конкатенации.

6. ФАЙЛОВАЯ СИСТЕМА

Операционная система должна обеспечить пользователям удобство работы с данными различного вида, хранящимися на диске. Для этого ОС использует вместо физического представления данных некоторую удобную для пользователей абстрактную логическую модель, представляемую в виде совокупности иерархически организованных каталогов и файлов. В результате пользователь видит данные в виде набора ярлыков или списков, выводимых на экран монитора утилитами типа Проводник (в ОС Windows).

6.1. Общие сведения о файлах и файловых системах

Файл – это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные, а также собственно хранимые в этой области данные и набор атрибутов, позволяющих ОС выполнять операции различного рода с этими данными.

Файловые системы должны обеспечить долговременное и надежное хранение информации в вычислительной системе, а также возможность совместного использования информации различными пользователями. Первое из этих свойств обеспечивается за счет хранения информации на запоминающих устройствах, не зависящих от питания, и за счет общей организации ОС, сбои в которой чаще всего не разрушают информацию, хранящуюся в файлах. Второе свойство обеспечивается понятными правилами символьного именования файлов, возможностью их группировки в иерархические структуры каталогов, наличием средств поиска файлов, создания, чтения, модификации и удаления файлов. Создатель файла или администратор имеет возможность задания прав доступа к файлам других пользователей.

Файловая система - это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие "файловая система" включает:

- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

6.2. Имена файлов

Файлы идентифицируются именами. Пользователи дают файлам символьные имена, при этом учитываются ограничения ОС как на используемые символы, так и на длину имени. Еще недавно эти границы были весьма узкими. Так в популярной файловой системе FAT длина имен ограничивается известной схемой 8.3 (8 символов - собственно имя, 3 символа - расширение имени), а в ОС UNIX SystemV имя не может содержать более 14 символов. Однако пользователю гораздо удобнее работать с длинными именами, поскольку они позволяют дать файлу действительно мнемоническое название, по которому даже через достаточно большой промежуток времени можно будет вспомнить, что содержит этот файл. Поэтому современные файловые системы, как правило, поддерживают длинные символьные имена файлов. К примеру, файловые системы NTFS и FAT32, используемые в ОС семейства Windows, устанавливают, что имя файла может содержать до 255 символов, не считая завершающего нулевого символа.

Переход к длинным именам порождает проблему совместимости с ранее созданными приложениями, использующими короткие имена. Чтобы приложения могли обращаться к файлам в соответствии с принятыми ранее соглашениями, файловая система должна уметь предоставлять эквивалентные короткие имена (псевдонимы) файлам, имеющим длинные имена. Следовательно, важной задачей становится проблема генерации соответствующих коротких имен.

Длинные имена поддерживаются не только новыми файловыми системами, но и новыми версиями хорошо известных файловых систем. Например, в ОС Windows 95 используется файловая система VFAT, представляющая собой существенно измененный вариант FAT. Среди многих других усовершенствований одним из главных достоинств VFAT является поддержка длинных имен. Кроме проблемы генерации эквивалентных коротких имен, при реализации нового варианта FAT важной задачей была задача хранения длинных имен при условии, что принципиально метод хранения и структура данных на диске не должны были измениться.

Обычно разные файлы могут иметь одинаковые символьные имена. В этом случае файл однозначно идентифицируется так называемым составным именем, представляющем собой последовательность символьных имен каталогов. В некоторых файловых системах одному и тому же файлу не может быть дано несколько разных имен, а в других такое ограничение отсутствует. В последнем случае операционная система присваивает файлу дополнительно уни-

кальное имя, так, чтобы можно было установить взаимно-однозначное соответствие между файлом и его уникальным именем. Уникальное имя представляет собой числовой идентификатор и используется программами операционной системы. Примером такого уникального имени файла является номер индексного дескриптора в системе UNIX.

6.3. Типы файлов

Файлы бывают разных типов: обычные файлы, специальные файлы, файлы-каталоги.

Обычные файлы в свою очередь подразделяются на текстовые и двоичные. Текстовые файлы состоят из строк символов, представленных в ASCII-коде. Это могут быть документы, исходные тексты программ и т.п. Текстовые файлы можно прочитать на экране и распечатать на принтере. Двоичные файлы не используют ASCII-коды, они часто имеют сложную внутреннюю структуру, например, объектный код программы или архивный файл. Все операционные системы должны уметь распознавать хотя бы один тип файлов - их собственные исполняемые файлы.

Специальные файлы - это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла. Эти команды обрабатываются вначале программами файловой системы, а затем на некотором этапе выполнения запроса преобразуются ОС в команды управления соответствующим устройством. Специальные файлы, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.

Каталог - это, с одной стороны, группа файлов, объединенных пользователем исходя из некоторых соображений (например, файлы, содержащие программы игр, или файлы, составляющие один программный пакет), а с другой стороны - это файл, содержащий системную информацию о группе файлов, его составляющих. В каталоге содержится список файлов, входящих в него, и устанавливается соответствие между файлами и их характеристиками (атрибутами).

В разных файловых системах могут использоваться в качестве атрибутов разные характеристики, например:

- информация о разрешенном доступе,
- пароль для доступа к файлу,
- владелец файла,
- создатель файла,
- признак "только для чтения",

- признак "скрытый файл",
- признак "системный файл",
- признак "архивный файл",
- признак "двоичный/символьный",
- признак "временный" (удалить после завершения процесса),
- признак блокировки,
- длина записи,
- указатель на ключевое поле в записи,
- длина ключа,
- времена создания, последнего доступа и последнего изменения,
- текущий размер файла,
- максимальный размер файла.

Каталоги могут непосредственно содержать значения характеристик файлов, как это сделано в файловой системе MS-DOS, или ссылаться на таблицы, содержащие эти характеристики, как это реализовано в ОС UNIX (рис. 6.1). Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рис. 6.2).

Иерархия каталогов может быть деревом или сетью. Каталоги образуют дерево, если файлу разрешено входить только в один каталог, и сеть - если файл может входить сразу в несколько каталогов. В MS-DOS каталоги образуют древовидную структуру, а в UNIX'е - сетевую. Как и любой другой файл, каталог имеет символьное имя и однозначно идентифицируется составным именем, содержащим цепочку символьных имен всех каталогов, через которые проходит путь от корня до данного каталога.

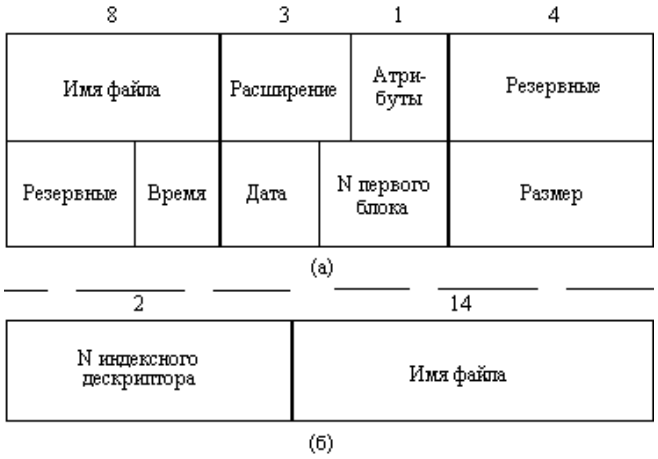


Рис. 6.1. Структура каталогов: а - структура записи каталога MS-DOS (32 байта); б - структура записи каталога ОС UNIX

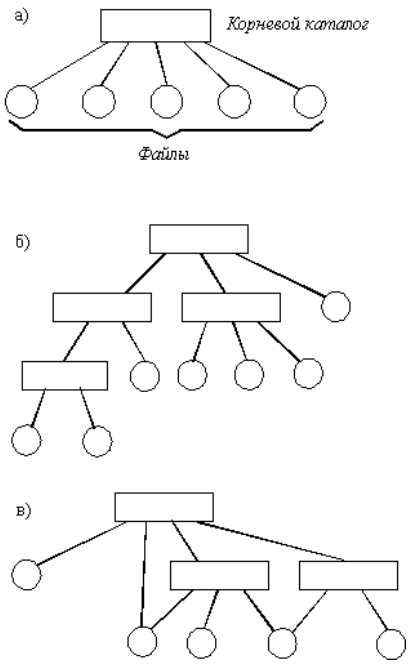


Рис. 6.2. Логическая организация файловой системы:
а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)

6.4. Организация файла

Логическая организация файла

Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей. Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством. Даже если физический обмен с устройством осуществляется большими единицами, операционная система обеспечивает программисту доступ к отдельной логической записи. На рис. 6.3 показаны несколько схем логической организации файла. Записи могут быть фиксированной длины или переменной длины. Записи могут быть расположены в файле последовательно (последовательная организация) или в более сложном порядке, с использованием так называемых индексных таблиц, позволяющих обеспечить быстрый доступ к отдельной логической записи (индексно-последовательная организация). Для идентификации записи может быть использовано специальное поле записи, называемое ключом. В файловых системах ОС UNIX и MS-DOS файл имеет простейшую логическую структуру - последовательность однобайтовых записей.

Рис. 6.3. Способы логической организации файлов

Физическая организация и адрес файла

Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске. Файл состоит из физических записей - блоков. Блок - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью. Непрерывное размещение - простейший вариант физической организации (рис. 6.4,а), при котором файлу предоставляется последовательность блоков диска, образующих единый сплошной участок дисковой памяти. Для задания адреса файла в этом случае достаточно указать только номер начального блока. Другое достоинство этого метода - простота. Но имеются и два существенных недостатка. Во-первых, во время создания файла заранее не известна его длина, а значит не известно, сколько памяти надо зарезервировать для этого файла, во-вторых, при таком порядке размещения неизбежно возникает фрагментация, и пространство на диске используется не эффективно, так как отдельные участки маленького размера (минимально 1 блок) могут остаться не используемыми.

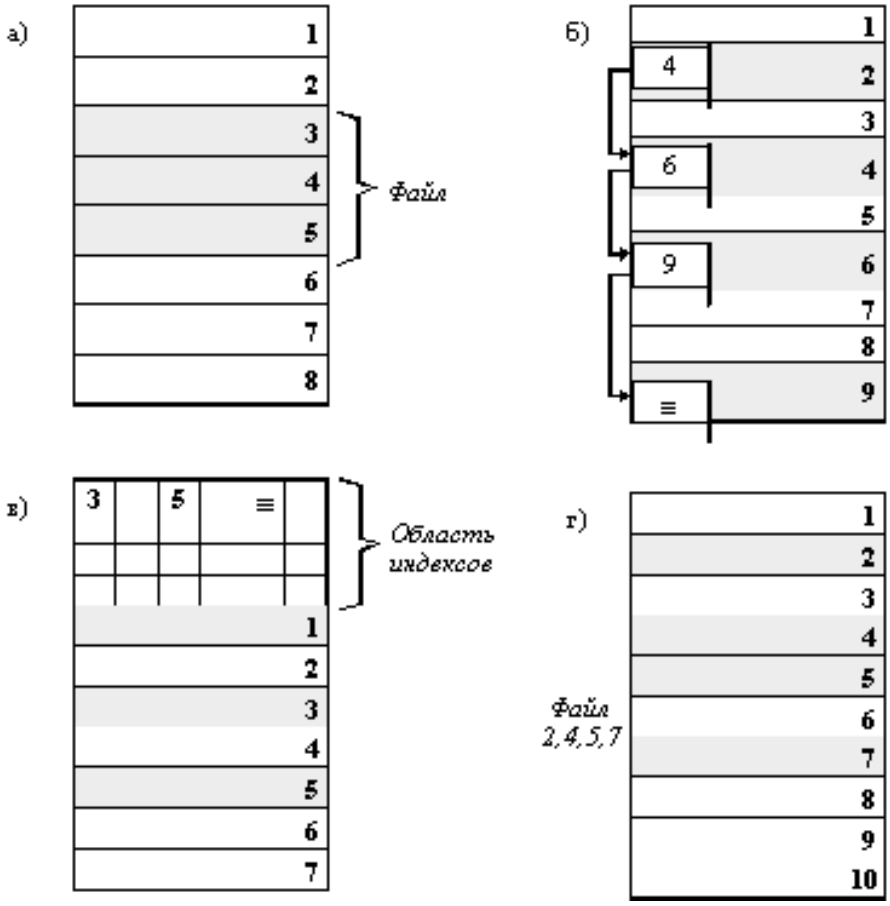


Рис. 6.4. Физическая организация файла
а - непрерывное размещение; б - связанный список блоков;
в - связанный список индексов; г - перечень номеров блоков

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти (рис. 6.4,б). При таком способе в начале каждого блока содержится указатель на следующий блок. В этом случае адрес файла также может быть задан одним числом - номером первого блока. В отличие от предыдущего способа, каждый блок может быть присоединен в цепочку какого-либо файла, следовательно, фрагментация отсутствует. Файл может изменяться во время своего существования, наращивая число блоков. Недостатком является сложность реализации доступа к произвольно заданному месту файла: для того, чтобы прочитать пятый по порядку блок файла, необходимо последовательно прочитать четыре первых блока, прослеживая цепочку номеров блоков. Кроме того, при этом способе количество данных файла, содержащихся в одном блоке, не равно степени двойки (одно слово израсходовано на номер следующего блока), а многие программы читают данные блоками, размер которых равен степени двойки.

Популярным способом, используемым, например, в файловой системе

FAT операционной системы MS-DOS, является использование связанного списка индексов. С каждым блоком связывается некоторый элемент - индекс. Индексы располагаются в отдельной области диска (в MS-DOS это таблица FAT). Если некоторый блок распределен некоторому файлу, то индекс этого блока содержит номер следующего блока данного файла. При такой физической организации сохраняются все достоинства предыдущего способа, но снимаются оба отмеченных недостатка: во-первых, для доступа к произвольному месту файла достаточно прочитать только блок индексов, отсчитать нужное количество блоков файла по цепочке и определить номер нужного блока, и, во-вторых, данные файла занимают блок целиком, а значит имеют объем, равный степени двойки.

В заключение рассмотрим задание физического расположения файла путем простого перечисления номеров блоков, занимаемых этим файлом. ОС UNIX использует вариант данного способа, позволяющий обеспечить фиксированную длину адреса, независимо от размера файла. Для хранения адреса файла выделено 13 полей. Если размер файла меньше или равен 10 блокам, то номера этих блоков непосредственно перечислены в первых десяти полях адреса. Если размер файла больше 10 блоков, то следующее 11-е поле содержит адрес блока, в котором могут быть расположены еще 128 номеров следующих блоков файла. Если файл больше, чем 10+128 блоков, то используется 12-е поле, в котором находится номер блока, содержащего 128 номеров блоков, которые содержат по 128 номеров блоков данного файла. И, наконец, если файл больше 10+128+128*128, то используется последнее 13-е поле для тройной косвенной адресации, что позволяет задать адрес файла, имеющего размер максимум 10+ 128 + 128*128 + 128*128*128.

Права доступа к файлу

Определить права доступа к файлу - значит определить для каждого пользователя набор операций, которые он может применить к данному файлу. В разных файловых системах может быть определен свой список дифференцируемых операций доступа. Этот список может включать следующие операции:

- создание файла,
- уничтожение файла,
- открытие файла,
- закрытие файла,
- чтение файла,

- запись в файл,
- дополнение файла,
- поиск в файле,
- получение атрибутов файла,
- установление новых значений атрибутов,
- переименование,
- выполнение файла,
- чтение каталога,

и другие операции с файлами и каталогами.

В самом общем случае права доступа могут быть описаны матрицей прав доступа, в которой столбцы соответствуют всем файлам системы, строки - всем пользователям, а на пересечении строк и столбцов указываются разрешенные операции (рис. 6.5).

Имена файлов

Имена пользователей	modern.txt	win.exe	class.dbf	unix.ppt
kira	читать	выполнять	–	выполнять
genya	читать	выполнять	–	выполнять читать
nataly	читать	–	–	выполнять читать
victor	читать писать	–	создать	–

Рис. 6.5. Матрица прав доступа

В некоторых системах пользователи могут быть разделены на отдельные категории. Для всех пользователей одной категории определяются единые права доступа. Например, в системе UNIX все пользователи подразделяются на три категории: владельца файла, членов его группы и всех остальных.

Различают два основных подхода к определению прав доступа:

- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными

правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

6.5. Кэширование диска

В некоторых файловых системах запросы к внешним устройствам, в которых адресация осуществляется блоками (диски, ленты), перехватываются промежуточным программным слоем-подсистемой буферизации. Подсистема буферизации представляет собой буферный пул, располагающийся в оперативной памяти, и комплекс программ, управляющих этим пулом. Каждый буфер пула имеет размер, равный одному блоку. При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса. Операция ввода-вывода считается выполненной, хотя физического обмена с устройством не происходило. Очевиден выигрыш во времени доступа к файлу. Если же нужный блок в буферном пуле отсутствует, то он считывается с устройства и одновременно с передачей запрашивающему процессу копируется в один из буферов подсистемы буферизации. При отсутствии свободного буфера на диск вытесняется наименее используемая информация. Таким образом, подсистема буферизации работает по принципу кэш-памяти.

6.6. Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рис. 6.6), в которой каждый уровень предоставляет некоторый интерфейс (набор функций) вышележащему уровню, а сам, в свою очередь, для выполнения своей работы использует интерфейс (обращается с набором запросов) нижележащего уровня.

Задачей символьного уровня является определение по символьному имени файла его уникального имени. В файловых системах, в которых каждый файл может иметь только одно символьное имя (например, MS-DOS), этот уровень отсутствует, так как символьное имя, присвоенное файлу пользователем, является одновременно уникальным и может быть использовано операционной системой. В других файловых системах, в которых один и тот же файл может иметь несколько символьных имен, на данном уровне просматривается цепочка каталогов для определения уникального имени файла. В файловой системе UNIX, например, уникальным именем является номер индексного дескриптора файла (i-node).

Рис. 6.6. Общая модель файловой системы

На следующем, базовом уровне по уникальному имени файла определяются его характеристики: права доступа, адрес, размер и другие. Как уже было сказано, характеристики файла могут входить в состав каталога или храниться в отдельных таблицах. При открытии файла его характеристики перемещаются с диска в оперативную память, чтобы уменьшить среднее время доступа к файлу. В некоторых файловых системах (например, HPFS) при открытии файла вместе с его характеристиками в оперативную память перемещаются несколько первых блоков файла, содержащих данные.

Следующим этапом реализации запроса к файлу является проверка прав доступа к нему. Для этого сравниваются полномочия пользователя или процесса, выдавших запрос, со списком разрешенных видов доступа к данному файлу. Если запрашиваемый вид доступа разрешен, то выполнение запроса продолжается, если нет, то выдается сообщение о нарушении прав доступа.

На логическом уровне определяются координаты запрашиваемой логической записи в файле, то есть требуется определить, на каком расстоянии (в байтах) от начала файла находится требуемая логическая запись. При этом абстрагируются от физического расположения файла, он представляется в виде не-

прерывной последовательности байт. Алгоритм работы данного уровня зависит от логической организации файла. Например, если файл организован как последовательность логических записей фиксированной длины l, то n-ая логическая запись имеет смещение l((n-1) байт. Для определения координат логической записи в файле с индексно-последовательной организацией выполняется чтение таблицы индексов (ключей), в которой непосредственно указывается адрес логической записи.

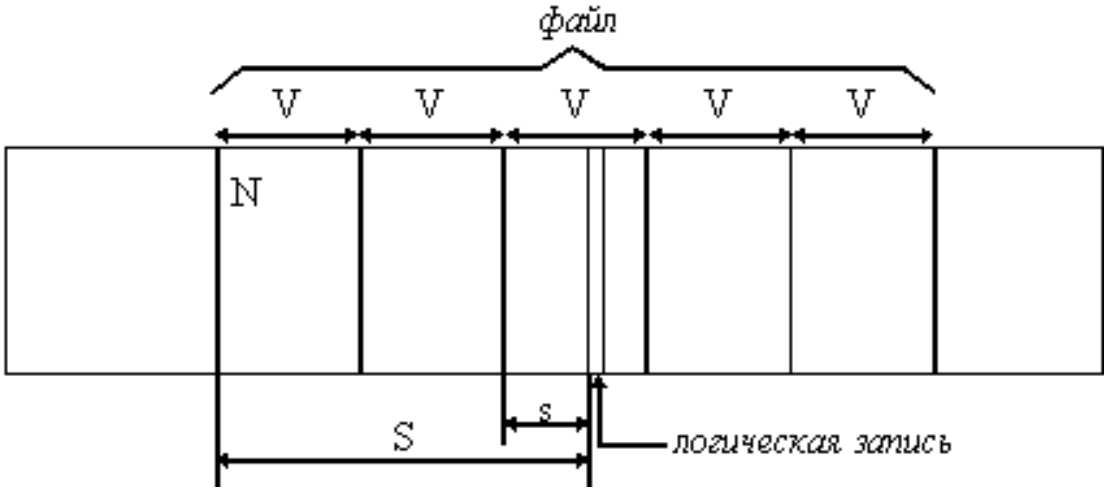


Рис. 6.7. Функции физического уровня файловой системы

Исходные данные:
V - размер блока
N - номер первого блока файла
S - смещение логической записи в файле
Требуется определить на физическом уровне:
n - номер блока, содержащего требуемую логическую запись
s - смещение логической записи в пределах блока
n = N + [S/V], где [S/V] - целая часть числа S/V
s = R [S/V] - дробная часть числа S/V

На физическом уровне файловая система определяет номер физического блока, который содержит требуемую логическую запись, и смещение логической записи в физическом блоке. Для решения этой задачи используются результаты работы логического уровня - смещение логической записи в файле, адрес файла на внешнем устройстве, а также сведения о физической организации файла, включая размер блока. Рис. 6.7 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физического уровня решает-

ся независимо от того, как был логически организован файл.

После определения номера физического блока, файловая система обращается к системе ввода-вывода для выполнения операции обмена с внешним устройством. В ответ на этот запрос в буфер файловой системы будет передан нужный блок, в котором на основании полученного при работе физического уровня смещения выбирается требуемая логическая запись.

6.7. Отображаемые в память файлы

По сравнению с доступом к памяти, традиционный доступ к файлам выглядит запутанным и неудобным. По этой причине некоторые ОС, начиная с MULTICS, обеспечивают отображение файлов в адресное пространство выполняемого процесса. Это выражается в появлении двух новых системных вызовов: MAP (отобразить) и UNMAP (отменить отображение). Первый вызов передает операционной системе в качестве параметров имя файла и виртуальный адрес, и операционная система отображает указанный файл в виртуальное адресное пространство по указанному адресу.

Предположим, например, что файл *f* имеет длину 64 К и отображается на область виртуального адресного пространства с начальным адресом 512 К. После этого любая машинная команда, которая читает содержимое байта по адресу 512 К, получает 0-ой байт этого файла и т.д. Очевидно, что запись по адресу 512 К + 1100 изменяет 1100-й байт файла. При завершении процесса на диске остается модифицированная версия файла, как если бы он был изменен комбинацией вызовов SEEK и WRITE.

В действительности при отображении файла внутренние системные таблицы изменяются так, чтобы данный файл служил хранилищем страниц виртуальной памяти на диске. Таким образом, чтение по адресу 512 К вызывает страничный отказ, в результате чего страница 0 переносится в физическую память. Аналогично, запись по адресу 512 К + 1100 вызывает страничный отказ, в результате которого страница, содержащая этот адрес, перемещается в память, после чего осуществляется запись в память по требуемому адресу. Если эта страница вытесняется из памяти алгоритмом замены страниц, то она записывается обратно в файл в соответствующее его место. При завершении процесса все отображенные и модифицированные страницы переписываются из памяти в файл.

Отображение файлов лучше всего работает в системе, которая поддерживает сегментацию. В такой системе каждый файл может быть отображен в свой собственный сегмент, так что k-ый байт в файле является k-ым байтом сегмен-

та. На рис. 6.8 а изображен процесс, который имеет два сегмента – кода и данных. Предположим, что этот процесс копирует файлы. Для этого он сначала отображает файл-источник, например, abc. Затем он создает пустой сегмент и отображает на него файл назначения, например, файл ddd.

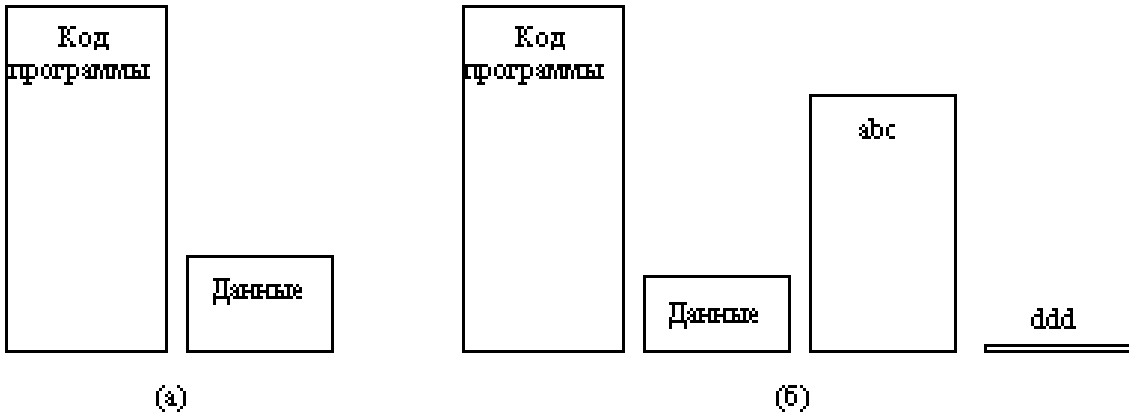


Рис. 6.8. (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) процесс после отображения существующего файла abc в один сегмент и создания нового сегмента для файла ddd

С этого момента процесс может копировать сегмент-источник в сегмент-приемник с помощью обычного программного цикла, использующего команды пересылки в памяти типа *mov*. Никакие вызовы READ или WRITE не нужны. После выполнения копирования процесс может выполнить вызов UNMAP для удаления файла из адресного пространства, а затем завершиться. Выходной файл ddd будет существовать на диске, как если бы он был создан обычным способом.

Хотя отображение файлов исключает потребность в выполнении ввода-вывода и тем самым облегчает программирование, этот способ порождает и некоторые новые проблемы. Во-первых, для системы сложно узнать точную длину выходного файла, в данном примере ddd. Проще указать наибольший номер записанной страницы, но нет способа узнать, сколько байт в этой странице было записано. Предположим, что программа использует только страницу номер 0, и после выполнения все байты все еще установлены в значение 0 (их начальное значение). Быть может, файл состоит из 10 нулей. А может быть, он состоит из 100 нулей. Как это определить? Операционная система не может это сообщить. Все, что она может сделать, так это создать файл, длина которого равна размеру страницы.

Вторая проблема проявляется (потенциально), если один процесс отображает файл, а другой процесс открывает его для обычного файлового доступа.

Если первый процесс изменяет страницу, то это изменение не будет отражено в файле на диске до тех пор, пока страница не будет вытеснена на диск. Поддержание согласованности данных файла для этих двух процессов требует от системы больших забот.

Третья проблема состоит в том, что файл может быть больше, чем сегмент, и даже больше, чем все виртуальное адресное пространство. Единственный способ ее решения состоит в реализации вызова MAP таким образом, чтобы он мог отображать не весь файл, а его часть. Хотя такая работа, очевидно, менее удобна, чем отображение целого файла.

6.8. Современные архитектуры файловых систем

Разработчики новых операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами. В новом понимании файловая система состоит из многих составляющих, в число которых входят и файловые системы в традиционном понимании.

Современная файловая система имеет многоуровневую структуру (рис. 6.9), на верхнем уровне которой располагается так называемый переключатель файловых систем (в Windows 95, например, такой переключатель называется устанавливаемым диспетчером файловой системы – installable filesystem manager, IFS). Он обеспечивает интерфейс между запросами приложения и конкретной файловой системой, к которой обращается это приложение. Переключатель файловых систем преобразует запросы в формат, воспринимаемый следующим уровнем – уровнем файловых систем.

Каждый компонент уровня файловых систем выполнен в виде драйвера соответствующей файловой системы и поддерживает определенную организацию файловой системы. Переключатель является единственным модулем, который может обращаться к драйверу файловой системы. Приложение не может обращаться к нему напрямую. Драйвер файловой системы может быть написан в виде реентерабельного кода, что позволяет сразу нескольким приложениям выполнять операции с файлами. Каждый драйвер файловой системы в процессе собственной инициализации регистрируется у переключателя, передавая ему таблицу точек входа, которые будут использоваться при последующих обращениях к файловой системе.

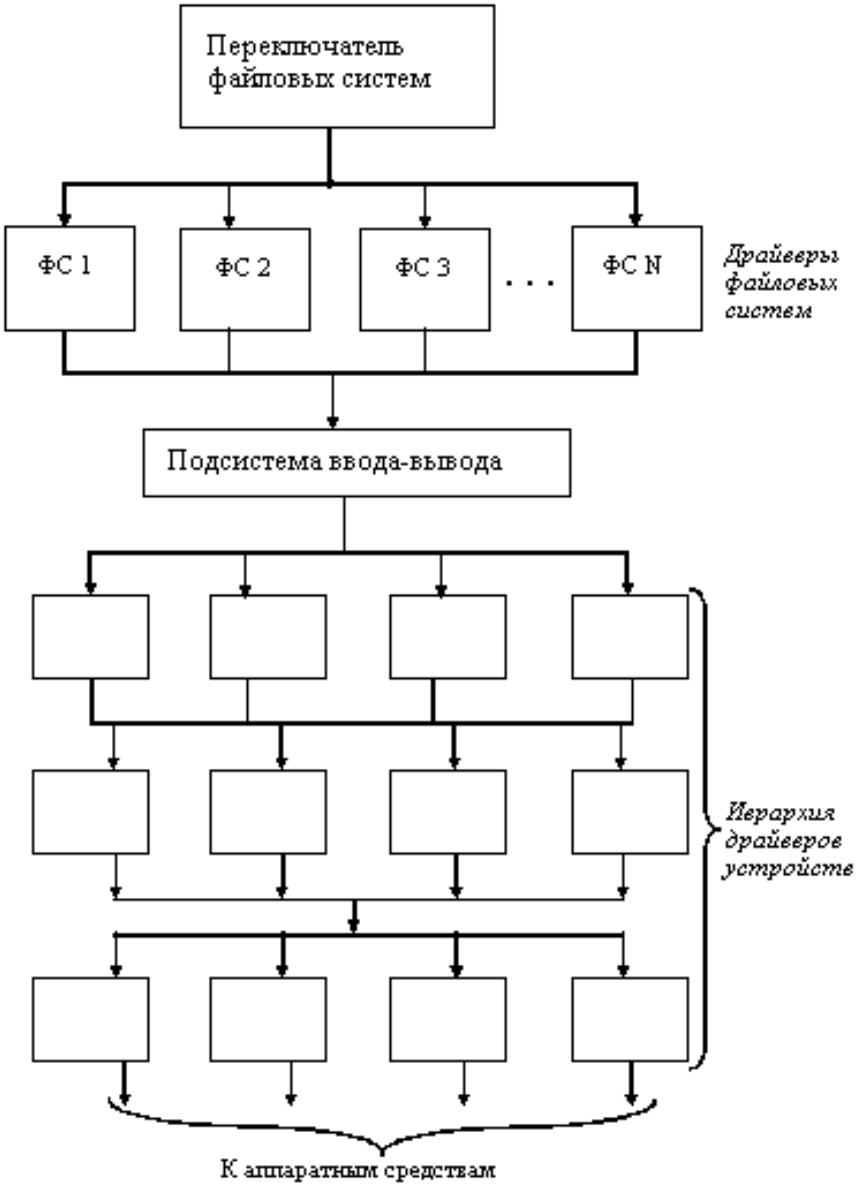


Рис. 6.9. Архитектура современной файловой системы

Для выполнения своих функций драйверы файловых систем обращаются к подсистеме ввода-вывода, образующей следующий слой файловой системы новой архитектуры. Подсистема ввода вывода - это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы. Обычно эти модули представляют собой драйверы портов, которые непосредственно занимаются работой с аппаратными средствами. Кроме этого подсистема ввода-вывода обеспечивает некоторый сервис драйверам файловой системы, что позволяет им осуществлять запросы к конкретным устройствам. Подсистема ввода-вывода должна постоянно присутствовать в памяти и организовывать совместную работу иерархии драйверов устройств. В эту иерархию могут входить драйверы устройств определенного типа (драйверы жестких дисков или накопителей на лентах), драйве-

ры, поддерживаемые поставщиками (такие драйверы перехватывают запросы к блочным устройствам и могут частично изменить поведение существующего драйвера этого устройства, например, зашифровать данные), драйверы портов, которые управляют конкретными адаптерами.

Большое число уровней архитектуры файловой системы обеспечивает авторам драйверов устройств большую гибкость - драйвер может получить управление на любом этапе выполнения запроса - от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера. Многоуровневый механизм работы файловой системы реализован посредством цепочек вызова.

В ходе инициализации драйвер устройства может добавить себя к цепочке вызова некоторого устройства, определив при этом уровень последующего обращения. Подсистема ввода-вывода помещает адрес целевой функции в цепочку вызова устройства, используя заданный уровень для того, чтобы должным образом упорядочить цепочку. По мере выполнения запроса, подсистема ввода-вывода последовательно вызывает все функции, ранее помещенные в цепочку вызова.

Внесенная в цепочку вызова процедура драйвера может решить передать запрос дальше - в измененном или в неизмененном виде - на следующий уровень, или, если это возможно, процедура может удовлетворить запрос, не передавая его дальше по цепочке.

7. СИСТЕМА ПРЕРЫВАНИЙ

7.1. Основные понятия. Типы прерываний

Прерывание - это сигнал, по которому процессор "узнает" о совершении асинхронного события. Другими словами, прерывание - это ответ вычислительной системы на наступление особого события, нарушающего последовательное выполнение команд текущей программы, то есть, это - изменение естественного порядка выполнения программы, которое связано с необходимостью реакции системы на работу внешних устройств, а также на ошибки и особые ситуации, возникшие при выполнении программы. "Ответ" системы на наступившее событие заключается в запуске программы обработки данного прерывания - **обработчика прерывания,** специальной программы, специфической для каждой возникшей ситуации, после выполнения которой, если это возможно, возобновляется работа прерванной программы.

Замечание. Таким образом, механизм обработки прерываний предоставляет возможность организации ветвления при реализации программных процессов.

В упрощенном представлении можно выделить три типа прерываний:

1) внутренние,
2) внешние,
3) внепроцессорные.

Внутренние прерывания. К прерываниям этого типа относят:

группу программных прерываний (деление на нуль, переполнение, неверная адресация и т. п.),

прерывания от схем контроля машины, сбоев системы питания и др.

Обработка прерываний этого типа состоит в выдаче сообщений о причине прерывания, прекращении выполнения текущей программы и перехода к реализации другой программы либо, если дальнейшее функционирование системы невозможно, только в выдаче диагностического сообщения, локализующего причину отказа.

В любом случае, при наступлении события, вызвавшего аварию, процессор не останавливается.

Внешние прерывания. Эту группу прерываний представляют прерывания от внешних устройств. Обработка событий, связанных с выполнением операций обмена данными между внешними устройствами и ОЗУ, в конечном счете, сводится к запуску **драйвера** - программы, реализующей обмен с устройством конкретного типа (драйвер клавиатуры, драйвер монитора и т. п.).

Внепроцессорные прерывания. Прерывания, обработка которых приводит к передаче управления общей шиной от процессора к контроллеру внешнего устройства с реализацией дальнейшего обмена между устройством и основной памятью по Общей шине напрямую без посредничества процессора, то есть без запуска какого-либо драйвера (см. п. 10.4.).

7.2. Общая организация прерываний

Механизм прерывания обеспечивается соответствующими аппаратно-программными средствами компьютера.

Задачей **аппаратных средств** обработки прерывания в процессоре ЭВМ является приостановка выполнения одной программы (иногда называемой основной) и передача управления подпрограмме обработки прерывания.

Поскольку для выполнения подпрограммы обработки прерывания используются различные регистры процессора (РОНы, счетчик команд, регистр фла-

гов и т.д.), то информацию, содержащуюся в них в момент прерывания, необходимо сохранить для последующего возврата в прерванную программу.

Обычно задача сохранения содержимого счетчика команд и регистра флагов, содержащего **вектор состояния** процессора возлагается на аппаратные средства обработки прерывания. Сохранение содержимого других регистров процессора, используемых в подпрограмме обработки прерывания, производится непосредственно в подпрограмме (рис. 7.1).

7.3. Организация системы прерываний с использованием векторов прерываний

Рассмотрим подробнее процесс обработки внешних прерываний.

Действия, выполняемые при этом процессором, как правило, те же, что и при обращении к обычной подпрограмме; различие в том, что при обращении к подпрограмме эти действия инициируются командой, а при обработке прерывания - управляющим сигналом от контроллера внешнего устройства, называемым **Запрос** (или **Требование**) **прерывания**.

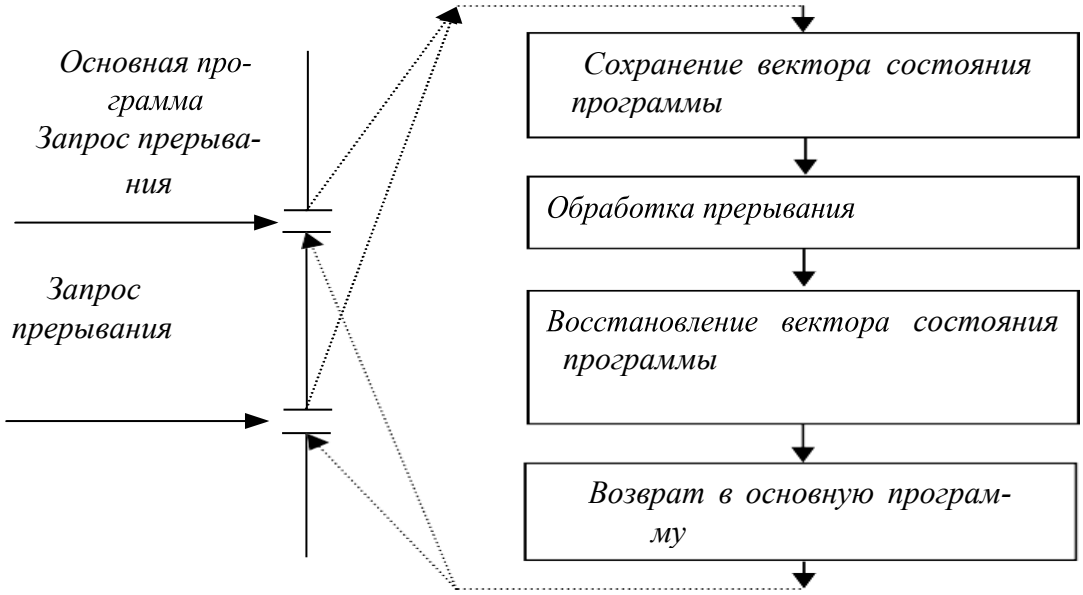


Рис.7.1. Структура подпрограммы обработки прерывания и ее связь с основной программой

Эта важная особенность обмена с прерыванием программы позволяет организовать обмен данными с внешними устройствами в произвольные моменты времени, не зависящие от программы, выполняемой в ЭВМ. Таким образом, появляется возможность обмена данными с внешними устройствами в реальном масштабе времени, определяемом внешней по отношению к ЭВМ средой (например, с датчиками, следящими за состоянием технологического процесса).

Прерывание программы по требованию внешнего устройства не должно оказывать на прерванную программу никакого влияния, кроме увеличения времени ее выполнения за счет приостановки на время выполнения подпрограммы обработки прерывания.

Формирование сигналов прерываний – запросов внешних устройств на обслуживание, происходит в их контроллерах – электронных схемах, обеспечивающих связь с внешним устройством. Еще есть драйвер – программа, обеспечивающая обслуживание прерывания. В серийных ЭВМ обычно используется одноуровневая система прерываний, то есть сигналы **Запрос прерывания** от всех внешних устройств поступают на один вход процессора. Поэтому возникает проблема идентификации внешнего устройства, запросившего обслуживание. Основным способом, решающим проблему идентификации в большинстве современных ЭВМ в настоящее время, является использование **векторов прерываний.**

Внешнее устройство, запросившее обслуживание, само идентифицирует себя с помощью адреса своего вектора прерывания - ячейки основной памяти, в которой хранится адрес начала программы обработки прерывания данного типа.

Векторы всех обработчиков прерываний собраны в единую **таблицу векторов прерываний,** располагающуюся в самых младших адресах оперативной памяти, имеющую объем 1 Кбайт и содержащую 4-х байтные элементы (векторы прерываний) для 256 обработчиков прерываний. Так как таблица всегда имеет нулевой начальный адрес и длину вектора в 4 байта, чтобы определить адрес вектора для прерывания типа i, достаточно просто умножить это значение на 4.

Вектор прерывания выдается контроллером не одновременно с запросом на прерывание, а только по разрешению процессора (рис.7.2).

Регистр прерываний составлен триггерами внешних устройств, устанавливаемыми в единичное состояние требованиями прерывания соответствующих контроллеров. Выходы триггеров поступают на входы приоритетного шифратора через элементы совпадения, вторые входы которых соединены с выходами регистра маски. Разряды этого регистра управляются ОС и устанавливаются в 0 при запрете прерывания соответствующего устройства и в 1, если прерывание разрешено.

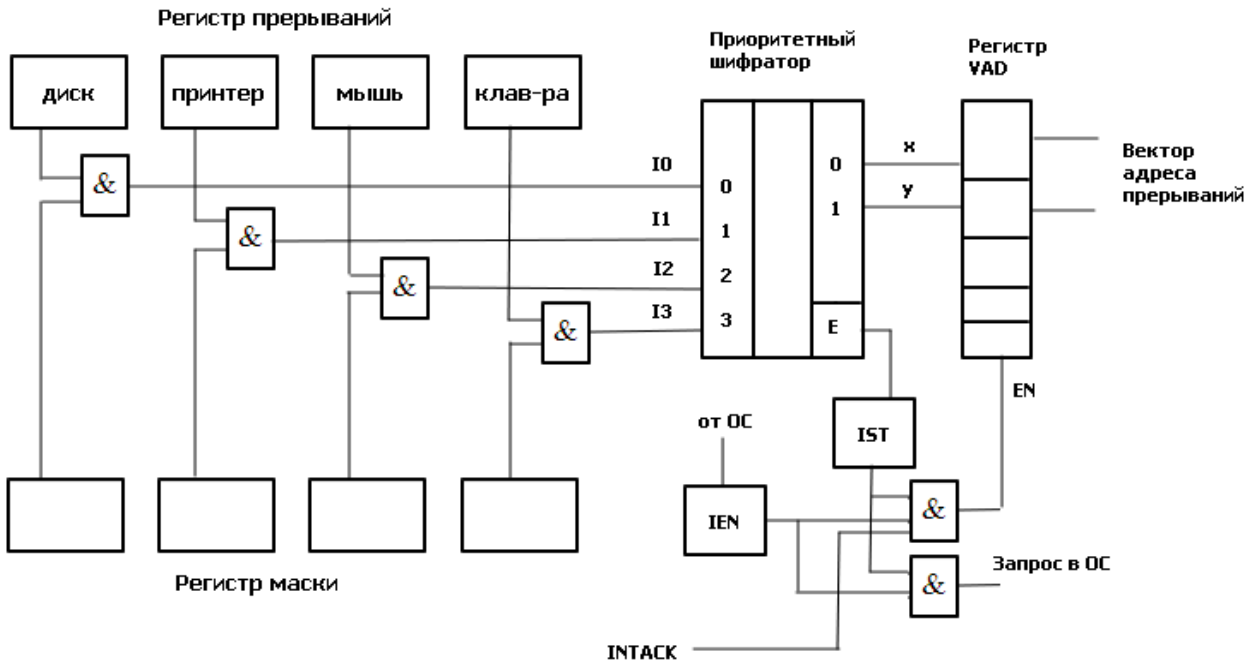


Рис.7.2. Схема обработки требования прерывания от внешнего устройства. IEN – Interrupt Enable – прерывание разрешено; IST – Interrupt Status – статус прерывания; INTACK – подтверждение прерывания; VAD – вектор адреса прерывания.

При поступлении хотя бы одного требования прерывания на входы приоритетного шифратора он устанавливает в единичное состояние свой выход E и устанавливает код приоритета прерывания на других своих выходах в соответствии с таблицей соответствия

I0	I1	I2	I3	Y	X	IST
1	X	X	X	0	0	1
0	1	X	X	0	1	1
0	0	1	X	1	0	1
0	0	0	1	1	1	1
0	0	0	0	0	0	0

В регистре прерываний устройства подключены в соответствии с их приоритетами. Чем выше приоритет, тем на меньший по значению вход приоритетного шифратора он поступает. Поэтому при одновременном поступлении требований прерывания от нескольких внешних устройств будет обрабатываться требование от устройства с высшим приоритетом. В начале оперативной памяти расположена таблица векторов прерываний. В каждой ячейке этой таблицы расположена команда безусловного перехода на начальный адрес соответствующей устройству программы обслуживания прерывания (ПОП).

Рассмотрим пример вычислительного процесса при обработке прерываний

от внешних устройств. Пусть программы обслуживания прерываний находятся по адресам 101-200 для диска (нулевой приоритет), 201-300 для принтера (1-й приоритет), 301-400 для мыши (2-й приоритет) и 401-500 для клавиатуры (3-й приоритет). Исполняемый код программы находится в области памяти 701-1500. Для сохранения адресов возврата предусмотрен стек.

Пусть при выполнении команды основной программы по адресу 809 пришло прерывание от клавиатуры. Затем, при выполнении команды по адресу 441 ПОП клавиатуры, пришло прерывание от принтера.

Надо изобразить ход вычислительного процесса в этих условиях.

7.4. Цикл прерывания

Управлением циклов в УУ ЦП занимаются триггеры F и R. Рассмотрим, как происходит управление при поступлении требований прерывания. В такте c_2t_3 цикла выполнения команды вырабатывается сигнал

Если $\overline{IEN * IST}$ c_2t_3: $F \leftarrow 0$

Если $(IEN * IST)$ c_2t_3: $R \leftarrow 1$

То есть, если имеется требование прерывания и разрешено прерывание от ОС, то выполняется установка в 1 триггера R, означающая переход к циклу прерывания.

В этом цикле выполняются следующие действия:

C_3t_0: M[SP] ← PC / запомнить адрес возврата в стеке.

C_3t_1: INTACK ← 1, VAD ← CD / подтвердить обработку прерывания, занести в VAD вектор прерывания.

C_3t_2: PC ← VAD / передать в счетчик команд адрес ПОП.

C_3t_3: IEN ← 0, C ← C_0 / Запретить прерывания. Перейти к выборке команды, адрес которой находится в PC.

Запрет прерывания необходим для того, чтобы обеспечить правильное выполнение прерывания, которое начал обслуживать процессор. В начале и в конце каждой ПОП имеются команды, которые нельзя прерывать.

В начале каждого драйвера имеются следующие стандартные команды:

1) Сбросить маски всех устройств, которые имеют приоритет ниже приоритета того устройства, прерывание которого обслуживается;
2) Сбросить регистры прерывания;
3) Запомнить содержание тех регистров процессора основной (прерванной) программы, которые используются в драйвере;
4) IEN ← 1 разрешить прерывания устройствам с более высоким приоритетом;

5) Перейти к выполнению основной части драйвера УВВ.

В конце драйвера:

1) IEN ← 0 запретить прерывания;
2) Восстановить содержание всех регистров процессора основной (прерванной) программы;
3) Сбросить требование прерывания того устройства, которое обслуживалось;
4) Восстановить регистр маски и шифратор приоритетный;
5) Восстановить адрес возврата в прерванную программу из стека PC ← M[SP];
6) Разрешить прерывания IEN ← 1.

Таким образом, мы рассмотрели мероприятия по обслуживанию прерываний.

8. КЛАССИФИКАЦИЯ И ТЕНДЕНЦИИ РАЗВИТИЯ АРХИТЕКТУР СОВРЕМЕННЫХ КОМПЬЮТЕРОВ

8.1. Классификации ЭВМ и ВС

Самой ранней и наиболее известной считается классификация архитектур вычислительных систем, предложенная в 1966 году **М. Флинном**. Классификация базируется на понятии ***потока,*** под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных М. Флинн выделяет четыре класса архитектур: SISD, MISD, SIMD и MIMD.

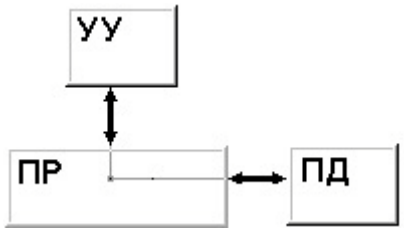


Рис. 8.1. Структура архитектуры класса SISD

SISD (single instruction stream/single data stream) – одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины, или иначе, машины фон-неймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно и каждая команда инициирует одну операцию с одним потоком данных. Для увеличения скорости обработки команд и скорости выполнения арифметических операций может при-

меняться конвейерная обработка. Поэтому в этот класс попадают ВМ со скалярными и с конвейерными функциональными устройствами.

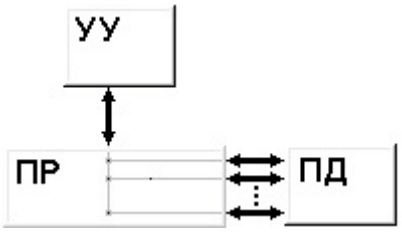


Рис. 8.2. Структура архитектуры класса SIMD.

SIMD (single instruction stream/multiple data stream) – одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными – элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производиться либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине CRAY-1.

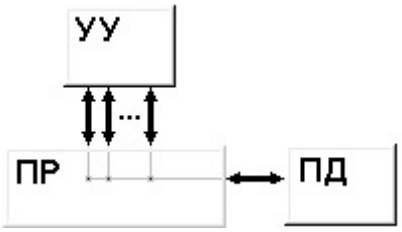


Рис. 8.3. Структура архитектуры класса MISD.

MISD (multiple instruction stream/single data stream) – множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни М.Флинн, ни другие специалисты в области архитектуры компьютеров до некоторого времени не могли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей относят конвейерные машины к данному классу. К данному классу, по-видимому, можно отнести появившиеся многоядерные компьютеры.

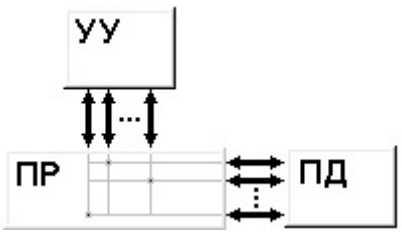


Рис. 8.4. Структура архитектуры класса MIMD.

MIMD (multiple instruction stream/multiple data stream) – множественный

поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Можно отметить два недостатка в классификации М. Флинна. Во-первых, некоторые заслуживающие внимания архитектуры, например, dataflow и векторно-конвейерные машины, четко не вписываются в данную классификацию. Во-вторых, класс MIMD чрезвычайно заполнен. Поэтому необходимо средство, более избирательно систематизирующее архитектуры, которые по М.Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Роджер Хокни разработал свой подход к классификации для более детальной систематизации компьютеров, попадающих в класс MIMD по **систематике М. Флинна**. Пытаясь систематизировать архитектуры внутри этого класса, Р. Хокни получил иерархическую структуру, представленную на рис. 8.5. Основная идея классификации состоит в следующем. Множественный поток команд может быть обработан двумя способами: либо одним конвейерным устройством обработки, работающем в режиме разделения времени для отдельных потоков, либо каждый поток обрабатывается своим собственным устройством. Первая возможность используется в MIMD-компьютерах, получивших название конвейерных.

Рис. 8.5. Иерархическая структура архитектуры класса MIMD

Архитектуры, использующие вторую возможность, в свою очередь, де-

лятся на два класса. В первый класс попадают MIMD-компьютеры, в которых возможна прямая связь каждого процессора с каждым, реализуемая с помощью переключателя. Во втором классе находятся MIMD-компьютеры, в которых прямая связь каждого процессора возможна только с ближайшими соседями по сети, а взаимодействие удаленных процессоров поддерживается специальной системой маршрутизации.

Среди MIMD-машин с переключателем Р. Хокни выделяет те, в которых вся память распределена среди процессов как их локальная память, например, PASM, PRINGLE, IBM SP2 без SMP-узлов. В этом случае общение самих процессоров реализуется с помощью сложного переключателя, составляющего значительную часть компьютера. Такие машины носят название **MIMD-машин с распределенной памятью**. Если память – разделяемый ресурс, доступный всем процессорам через переключатель, то MIMD-машины являются системами с общей памятью (BBN Butterfly, Cray C90). В соответствии с типом переключателей можно проводить классификацию и далее: простой переключатель, многокаскадный переключатель, общая шина и т. п. Многие современные вычислительные системы имеют как общую разделяемую память, так и распределенную локальную. Такие системы принято называть гибридными MIMD с переключателем.

Особенности организации и функционирования архитектур с общей, распределенной и смешанной памятью

Классифицируя современные компьютеры, которые практически все относятся к классу MIMD, будем основываться на анализе используемых в системах способах организации оперативной памяти. На рис. 8.6 приведена классификация систем MIMD, основанная на разных способах организации памяти.

Данный поход позволяет различать два важных типа многопроцессорных систем – **мультипроцессоры** (multiprocessors или системы с общей разделяемой памятью) и **мультикомпьютеры** (multicomputers или системы с распределенной памятью).

Для **мультипроцессоров** учитывается способ построения общей памяти. Возможный подход – использование единой (централизованной) общей памяти. Такой подход обеспечивает однородный доступ к памяти (uniform memory access или UMA) и служит основой для построения векторных суперкомпьютеров (parallel vector processor, PVP) и симметричных мультипроцессоров (symmetric multiprocessor или SMP). Среди примеров первой группы суперкомпьютер Cray T90, ко второй группе относятся IBM eServer p690 и др.

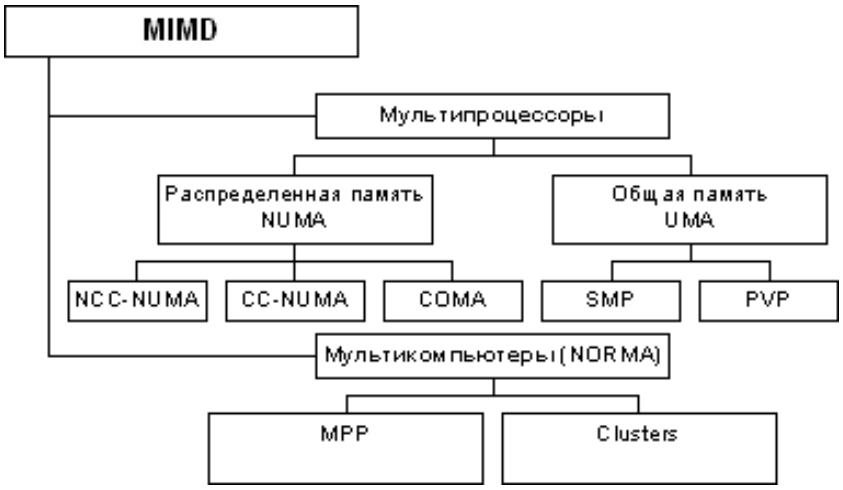


Рис. 8.6. Классификация систем MIMD

Общий доступ к данным может быть обеспечен и при физически распределенной памяти (при этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти). Такой подход именуется как неоднородный доступ к памяти (non-uniform memory access или NUMA).

Среди систем с таким типом памяти выделяют:

- Системы, в которых для представления данных используется только локальная кэш память имеющихся процессоров (cache-only memory architecture или COMA); примерами таких систем являются, например, KSR-1 и DDM;
- Системы, в которых обеспечивается однозначность (когерентность) локальных кэш памяти разных процессоров (cache-coherent NUMA или CC-NUMA); среди систем данного типа SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;
- Системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэша (non-cache coherent NUMA или NCC-NUMA); к данному типу относится, например, система Cray T3E.

Мультикомпьютеры уже не обеспечивают общий доступ ко всей имеющейся в системах памяти (no-remote memory access или NORMA). Данный подход используется при построении двух важных типов многопроцессорных вычислительных систем – массивно-параллельных систем (massively parallel processor или MPP) и кластеров (clusters). Среди представителей первого типа систем

– IBM RS/6000 SP2, Intel PARAGON/ASCI Red, транспьютерные системы Parsytec и др.; примерами кластеров являются, например, системы AC3

Velocity, NCSA/NT Supercluster и др.

Следует отметить чрезвычайно быстрое развитие кластерного типа многопроцессорных вычислительных систем.

Далее приводятся базовые характеристики основных классов современных компьютеров.

Массивно-параллельные системы (MPP)

На рис.8.7 представлена типовая архитектура вычислительных систем с распределенной памятью.

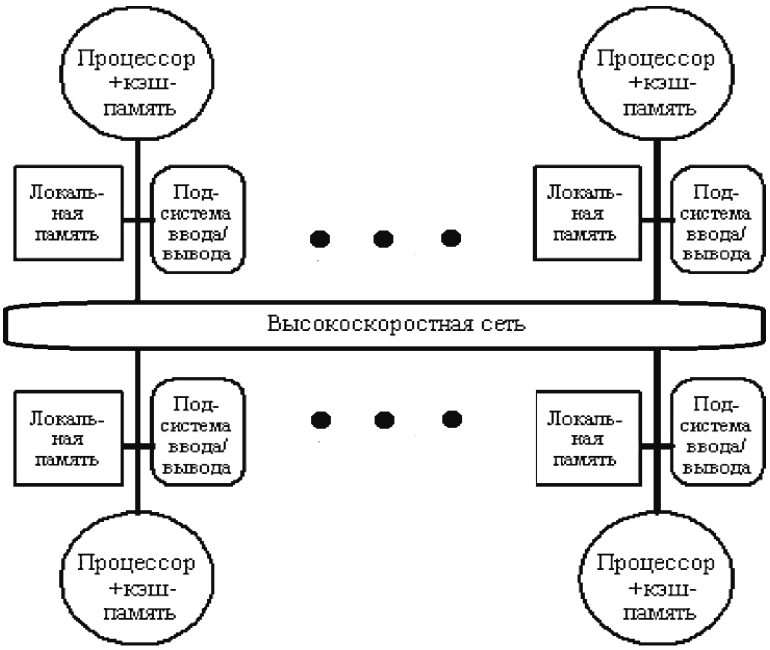


Рис. 8.7 Типовая архитектура машины с распределенной памятью

Архитектура: Система состоит из однородных вычислительных узлов, включающих:

- один или несколько центральных процессоров (обычно RISC),
- локальную память (прямой доступ к памяти других узлов невозможен),
- коммуникационный процессор или сетевой адаптер,
- иногда – жесткие диски (как в SP) и/или другие устройства Вв/Выв.

К системе могут быть добавлены специальные узлы ввода-вывода и управляющие узлы. Узлы связаны через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.)

Примеры: IBM RS/6000 SP2, Intel PARAGON/ASCI Red, SGI/CRAY T3E, Hitachi SR8000, транспьютерные системы Parsytec и др.

Масштабируемость: Общее число процессоров в реальных системах до-

стигает нескольких тысяч (ASCI Red, Blue Mountain).

Операционная система: Существуют два основных варианта реализации:

1. Полноценная ОС работает только на управляющей машине (front-end), на каждом узле работает сильно урезанный вариант ОС, обеспечивающий 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 + операционная система AIX, которая устанавливается отдельно на каждом узле.

Модель программирования: Программирование в рамках модели передачи сообщений (MPI, PVM, BSPlib)

Распределенность памяти означает, что каждый процессор имеет непосредственный доступ только к своей локальной памяти, а доступ к данным, расположенным в памяти других процессоров, выполняется другими способами.

Чтобы переслать информацию от процессора к процессору, необходим механизм передачи сообщений по сети, связывающей вычислительные узлы. Для абстрагирования от подробностей функционирования коммуникационной аппаратуры и программирования на высоком уровне, используются библиотеки передачи сообщений. Несмотря на существенные различия средств межпроцессорного взаимодействия в разных системах по скоростным параметрам и по способу аппаратной реализации, библиотеки обмена сообщениями выполняют приблизительно одни и те же функции.

Выбор топологии машины часто определяет способ решения прикладной задачи. Надо заметить, что оптимизация алгоритмов для параллельных архитектур существенно отличается от той же работы для последовательных систем. Если переход с одного скалярного процессора на другой практически никогда не требует пересмотра алгоритма, то алгоритм, идеально приспособленный для одной параллельной архитектуры, на другой машине (с тем же числом процессоров того же типа) может работать неприемлемо медленно. Для оценки производительности распределенной системы, кроме топологии связей, необходимо знать скорость выполнения арифметических операций, время инициализации канала связи и время передачи единицы объема информации. Если топология системы не тривиальна, то в состав операционной системы или пакета передачи сообщений приходится включать процедуры маршрутизации сообщений, работающие на каждом узле и обеспечивающие пересылку транзитных со-

общений. Они также вызывают задержку при передаче информации между узлами, не имеющими прямого канала связи.

Таким образом, применение дешевых процессоров позволяет сделать относительно недорогой суперкомпьютер. Широкому распространению подобных архитектур препятствует в основном отсутствие эффективных параллельных программ, полностью использующих их возможности.

Симметричные мультипроцессорные системы (SMP)

На рис. 8.8 приведена типовая архитектура мультипроцессорных вычислительных систем с общей памятью.

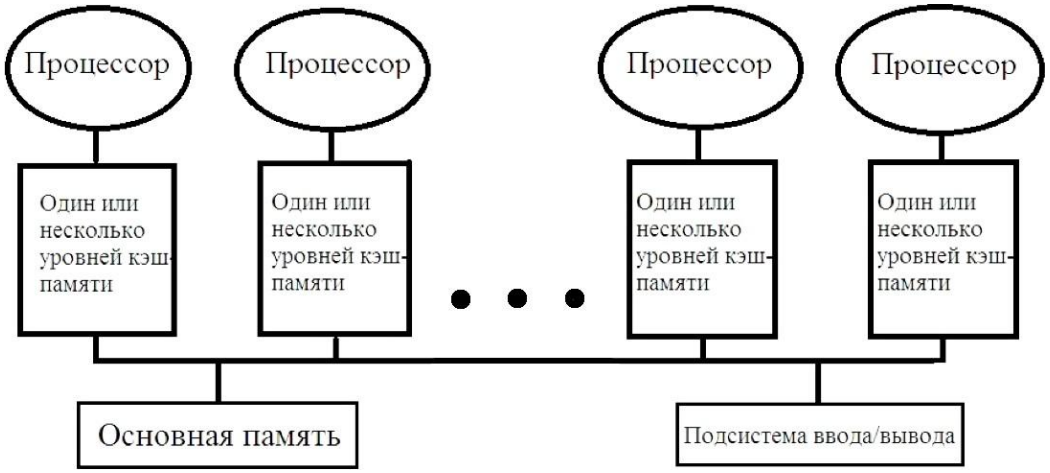


Рис. 8.8. Типовая архитектура мультипроцессорной системы с общей памятью

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-сервера), 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.

SMP - это один компьютер с несколькими равноправными процессорами, но с одной памятью, подсистемой ввода/вывода и одной ОС. Каждый процессор имеет доступ ко всей памяти, может выполнять любую операцию ввода/вывода, прерывать другие процессоры и т.д., но это представление справедливо только на уровне программного обеспечения. На самом же деле в SMP имеется несколько устройств памяти.

Каждый процессор имеет, по крайней мере, одну собственную кэш-память, что необходимо для достижения хорошей производительности, поскольку основная память работает слишком медленно по сравнению со скоростью процессоров (и это соотношение все больше ухудшается), а кэш работает со скоростью процессора, но дорог, и поэтому устройства кэш-памяти обладают относительно небольшой емкостью. Из-за этого в кэш помещается лишь оперативная информация, остальное же хранится в основной памяти. тсюда возникает проблема когерентности кэшей – получение процессором значения, находящегося в кэш-память другого процессора. Это решается при помощи отправки широковещательного запроса всем устройствам кэш-памяти, основной памяти и даже подсистеме ввода/вывода, если она работает с основной памятью напрямую, с целью получения актуальной информации.

Имеется еще одно следствие, связанное с параллелизмом. Неявно производимая аппаратурой SMP пересылка данных между кэшами является наиболее быстрым и самым дешевым средством коммуникации в любой параллельной архитектуре общего назначения. Поэтому при наличии большого числа коротких транзакций (свойственных, например, банковским приложениям), когда приходится часто синхронизовать доступ к общим данным, архитектура SMP является наилучшим выбором; любая другая архитектура работает хуже.

Тем не менее, архитектуры с разделяемой общей памятью не считаются перспективными. Основная причина довольно проста. Рост производительности в параллельных системах обеспечивается наращиванием числа процессоров, что приводит к тому, что узким местом становится доступ к памяти. Увеличение локальной кэш-памяти не способно полностью решить проблему: задача поддержания согласованного состояния нескольких банков кэш-памяти столь же трудна. Как правило, на основе общей памяти не создают систем с числом процессоров более 32, при необходимости объединяя их в кластерные

или NUMA- архитектуры.

Системы с неоднородным доступом к памяти (NUMA)

На рис. 8.9 изображена типовая архитектура мультипроцессорных вычислительных систем с неоднородным доступом к памяти.

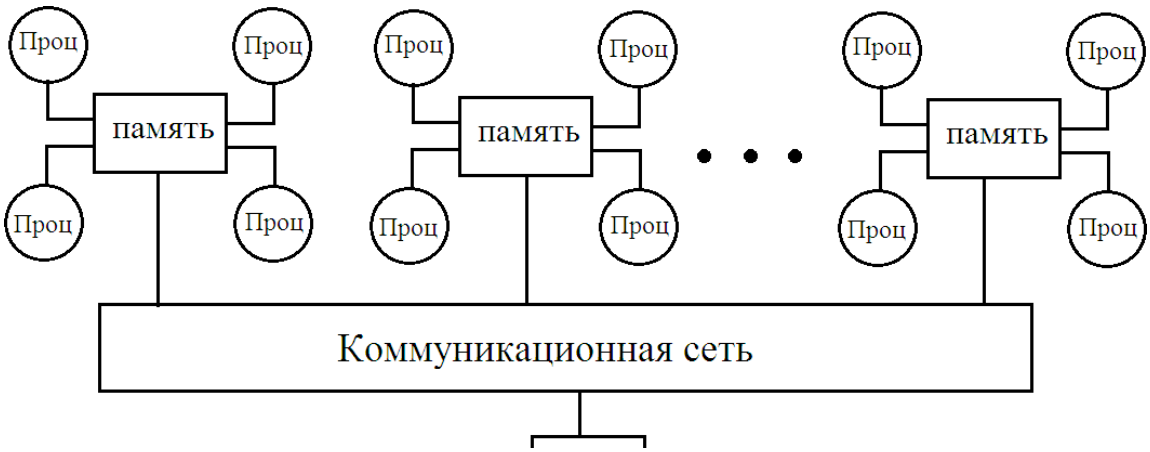


Рис. 8.9. Типовая архитектура мультипроцессорной системы с неоднородным доступом к памяти

Архитектура: 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. Доступ к локальной памяти узла осуществляется в несколько раз быстрее, чем к удаленной.

В случае, если аппаратно поддерживается когерентность кэшей во всей системе (обычно это так), говорят об архитектуре CC-NUMA (cache-coherent NUMA). **Примеры:** HP 9000 V-class в SCA-конфигурациях, SGI Origin2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000, SNI RM600 и др.

Масштабируемость: Масштабируемость NUMA-систем ограничивается объемом адресного пространства, возможностями аппаратуры поддержки когерентности кэшей и возможностями операционной системы по управлению большим числом процессоров. На настоящий момент, максимальное число процессоров в NUMA-системах составляет 256 (Origin2000).

Операционная система: 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).

Модель программирования: Аналогично SMP.

По сути своей NUMA представляет собой большую SMP-систему, разбитую на набор более мелких и простых SMP. Аппаратура позволяет работать со всеми отдельными устройствами основной памяти составных частей системы (называемых обычно узлами) как с единой гигантской памятью. Этот подход порождает ряд следствий. Во-первых, в системе имеется одно адресное пространство, распространяемое на все узлы. Реальный (не виртуальный) адрес 0 для каждого процессора в любом узле соответствует адресу 0 в частной памяти узла 0; реальный адрес 1 для всей машины – это адрес 1 в узле 0 и т.д., пока не будет использована вся память узла 0. Затем происходит переход к памяти узла 1, затем узла 2 и т.д. Для реализации этого единого адресного пространства каждый узел NUMA включает специальную аппаратуру (Dir), которая решает проблему когерентности кэшей, обеспечивая получение актуальной информации от других узлов.

Понятно, что этот процесс длится несколько дольше, чем если бы требуемое значение находилось в частной памяти того же узла. Отсюда и происходит словосочетание "неоднородный доступ к памяти". В отличие от SMP, время выборки значения зависит от адреса и от того, от какого процессора исходит запрос (если, конечно, требуемое значение не содержится в кэше).

Поэтому ключевым вопросом является степень "неоднородности" NUMA. Например, если для взятия значения из другого узла требуется только на 10% большее время, то это никого не задевает. В этом случае все будут относиться к системе как к SMP, и разработанные для SMP программы будут выполняться достаточно хорошо.

Однако в текущем поколении NUMA-систем для соединения узлов используется сеть. Это позволяет включать в систему большее число узлов, до 64 узлов с общим числом процессоров 128 в некоторых системах. В результате, современные NUMA-системы не выдерживают правила 10% – лучшие образцы замедление 200–300% и даже более. При такой разнице в скорости доступа к памяти для обеспечения должной эффективности следует позаботиться о правильном расположении требуемых данных. Чтобы этого добиться, можно соответствующим образом модифицировать операционную систему (и это сделали поставщики систем в архитектуре NUMA). Например, такая операционная система при запросе из программы блока памяти выделяет память в узле, в котором выполняется эта программа, так что когда процессор ищет соответствующие данные, то находит их в своем собственном узле. Аналогичным образом

должны быть изменены подсистемы (включая СУБД), осуществляющие собственное планирование и распределение памяти (что и сделали Oracle и Informix). Как утверждает компания Silicon Graphics, такие изменения позволяют эффективно выполнять в системах с архитектурой NUMA приложения, разработанные для SMP, без потребности изменения кода.

Параллельные векторные системы (PVP)

Архитектура: 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.

Как правило, несколько таких процессоров (1–16) работают одновременно над общей памятью (аналогично SMP) в рамках многопроцессорных конфигураций. Несколько таких узлов могут быть объединены с помощью коммутатора (аналогично MPP).

Примеры: NEC SX-4/SX-5, линия векторно-конвейерных компьютеров CRAY: от CRAY-1, CRAY J90/T90, CRAY SV1, серия Fujitsu VPP и др.

Модель программирования: Эффективное программирование подразумевает векторизацию циклов (для достижения разумной производительности одного процессора) и их распараллеливание (для одновременной загрузки нескольких процессоров одним приложением).

Кластерные системы

Архитектура: 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.

При объединении в кластер компьютеров разной мощности или разной архитектуры, говорят о гетерогенных (неоднородных) кластерах.

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.

Примеры: NT-кластер в NCSA, Beowulf-кластеры, кластеры МГУ и СПбГУ, кластер МЭИ (см.рис. 8.10 и 8.11) и др.

Операционная система: Используются стандартные для рабочих станций ОС, чаще всего, свободно распространяемые – Linux/FreeBSD, вместе со специальными средствами поддержки параллельного программирования и рас-

пределения нагрузки.

Модель программирования: Программирование, как правило, в рамках модели передачи сообщений (чаще всего - MPI).

8.2. Организация схем коммутации

Важнейшим аспектом создания высокопроизводительных архитектур является построение средств коммутации.

Структура линий коммутации между процессорами вычислительной системы (топология сети передачи данных) определяется, как правило, с учетом возможностей технической реализации. Немаловажную роль при выборе структуры сети играет и анализ интенсивности информационных потоков при параллельном решении наиболее распространенных вычислительных задач. Рассмотрим основные схемы коммутации в МВС разной системы организации памяти.

Организация схем коммутации в МВС с общей памятью

На рис. 8.10 представлена типовая архитектура МВС с общей памятью.

Замечание. В реальных вычислительных системах (ВС) с общей памятью количество процессоров n не превосходит 32. Число модулей памяти m, осуществляющих хранение данных, не превосходит n, то есть $m \le n \le 32$.

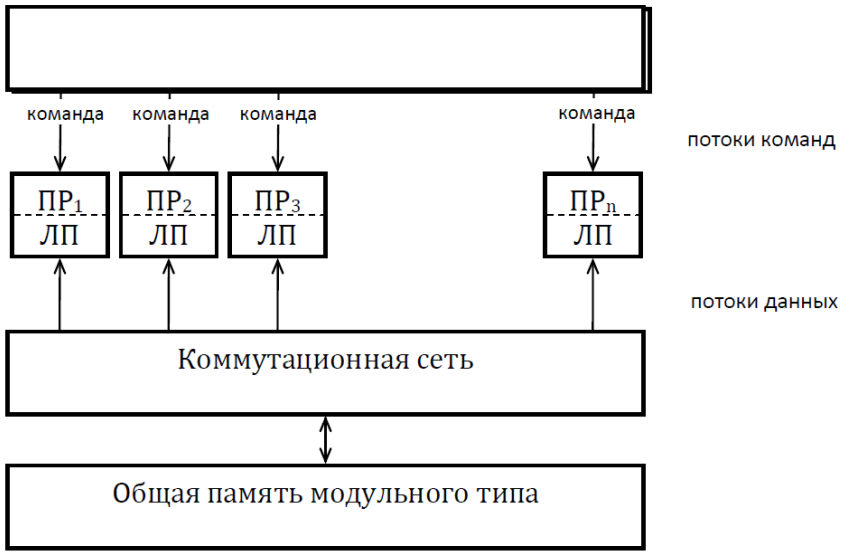


Рис. 8.10. Типовая архитектура МВС с общей памятью

Возможно несколько вариантов построения ВС с общей памятью:

1. ВС с единственным модулем памяти ($m \ge 1$), или многовходовой памятью;
2. ВС с несколькими модулями памяти ($1 \le m \le n$), или несколькими многовходовыми памятями;
3. ВС с количеством модулей памяти равным количеству процессоров ($m \le n$) и

коммутацией в виде коммутационной сети.

На рис. 8.11, 8.12 и 8.13 представлены варианты реализации.

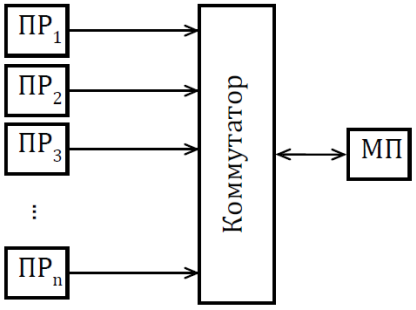


Рис. 8.11. Единый модуль памяти m = 1

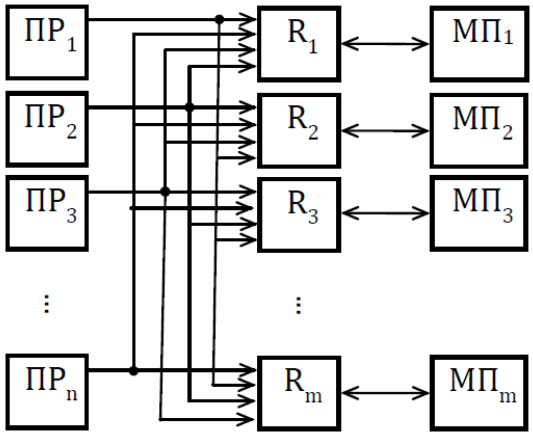


Рис. 8.12. Несколько модулей памяти 1 < m < n

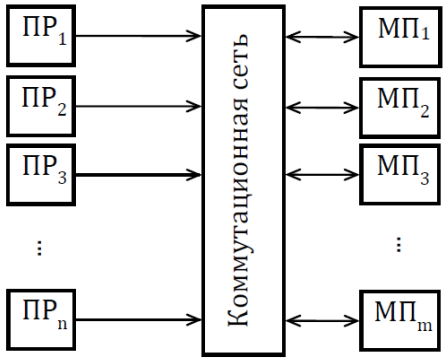


Рис. 8.13. Число модулей памяти равно числу процессоров m ≤ n

Организация средств коммутации в архитектуре “Butterfly”

Для архитектуры “Butterfly” - n = m = 256 . Коммутационный узел (КУ), как изображено на рис. 8.14, имеет 4 входа и 4 выхода, причем каждый вход соединён с каждым выходом.

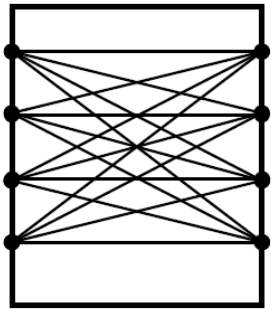


Рис. 8.14. Коммутационный узел «4 х 4»

Коммутационный блок (КБ) представляет собой аналогичную КУ структуру, «атомарными» в которой являются КУ (рис. 8.15).

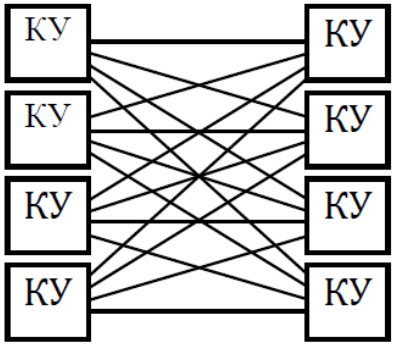


Рис. 8.15. Коммутационный блок «16 х 16»

Коммутационная сеть по типу Butterfly изображена на рис. 8.16.

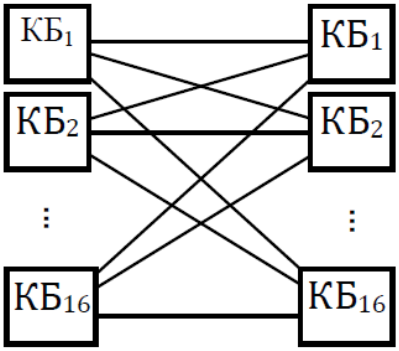


Рис. 8.16. Коммутационная сеть по типу Butterfly «256 х 256»

В МВС с общей памятью в качестве средства передачи информации часто используют шины. Каждый модуль памяти имеет свою шину, к которой подключаются все процессорные узлы. Схематически это показано на рис. 8.17.

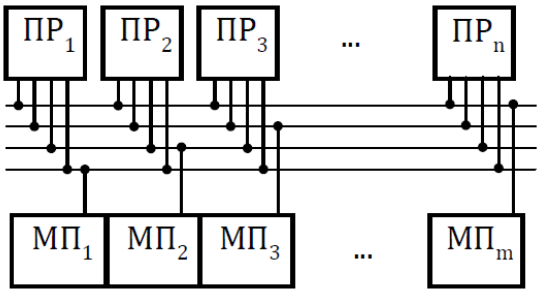


Рис. 8.17. МВС с общей памятью

Организация схем коммутации в МВС с распределенной памятью

На рис. 8.18 представлена типовая архитектура МВС с распределенной памятью.

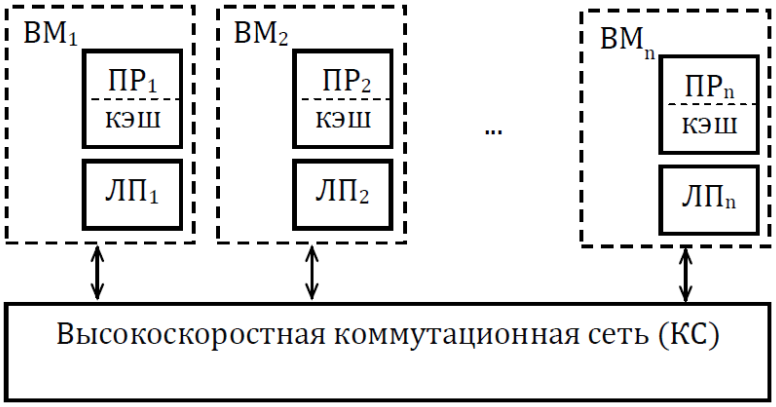


Рис. 8.18. Типовая архитектура схем коммутации в МВС с распределенной памятью

При обсуждении этого типа архитектур будем использовать следующие обозначения: число шин – m, число процессоров – n. Также будем учитывать ограничение m n/2. Средства коммутации для этого типа архитектуры представляют важнейший структурный элемент ВС.

Наиболее распространенные решения при построении КС для МВС с распределенной памятью можно разбить на три группы.

1. ВС со связями через общую шину (см. рис. 8.19).

Такая схема является, с одной стороны, дешевой и просто реализуемой, а также соответствует структуре передачи данных при решении многих вычислительных задач. Легко наращивается число подключаемых вычислительных модулей (ВМ). С другой стороны, для обеспечения эффективной работы шины требуется тщательное планирование использования шины (арбитр шины), работающей в режиме разделения времени, от которой зависит производительность всей системы.

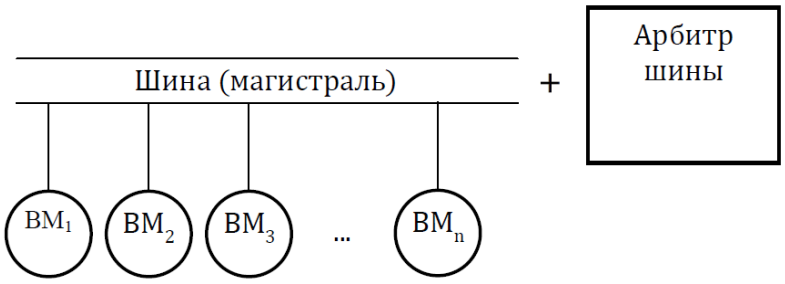


Рис. 8.19. Архитектура схем коммутации в МВС с распределенной памятью со связями через общую шину

2. Более дорогой вариант – архитектура со связями через несколько шин – представлена на рис. 8.20. При таком способе связи поддерживается режим прямого доступа к памяти, причем передача производится обычно блоками из фиксированного набора слов.

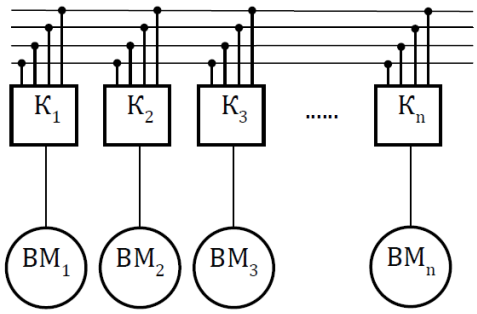


Рис. 8.20. Архитектура схем коммутации в МВС с распределенной памятью со связями через несколько шин

Достоинством архитектур МВС, использующих при коммутации несколько шин, является бо'льшая производительность и надежность, чем у аналогов с одной шиной. Однако для организации эффективных вычислений необходимы n коммутаторов шин, а это высокие аппаратные затраты.

3. На рис. 8.21 представлена архитектура со связями через многоступенчатый переключатель.

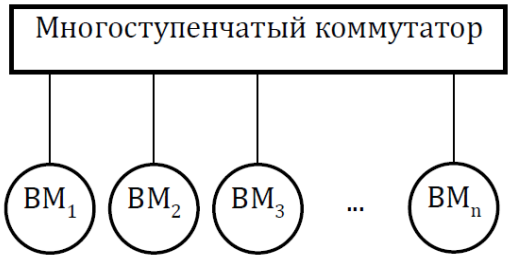


Рис. 8.21. МВС с распределенной памятью со связями через многоступенчатый переключатель

Существует большое разнообразие в организации коммутации такого вида.

При соединении первого и последнего процессоров линейки через коммутатор получается топология, называемая **кольцом** (рис. 8.22).

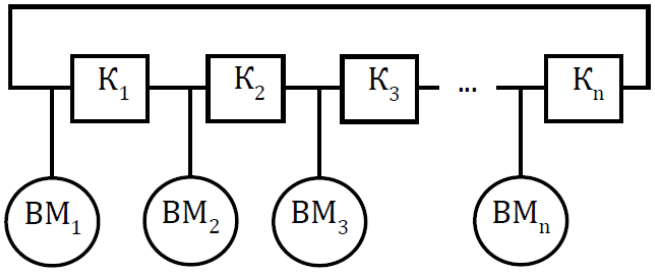


Рис. 8.22. Многоступенчатый коммутатор кольцевого типа

Система, в которой каждый ВМ связан с каждым другим ВМ прямой линией связи, построена на основе КС с **полным набором связей** (рис. 8.23). Такая топология обеспечивает высокую надежность и минимальные затраты при передаче данных, однако является сложно реализуемой при большом количестве процессов. Каждый узел – это ВМ со своим многовходовым коммутатором.

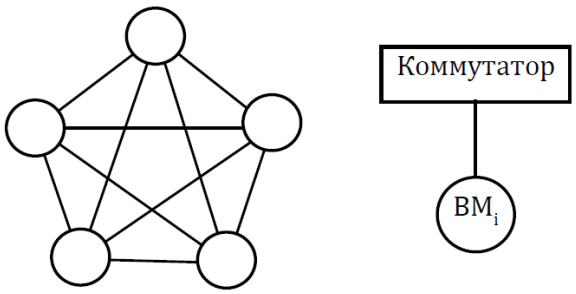


Рис. 8.23. Схема КС с полным набором связей

Система, в которой все ВМ с помощью своих коммутаторов имеют линии связи с некоторым центральным коммутатором или центральным коммутационным узлом (ЦКУ) или управляющим процессором, называется ВС с топологией типа «**звезда».**

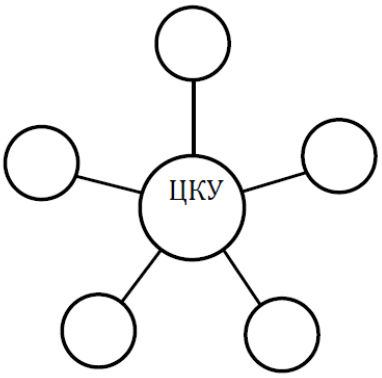


Рис. 8.24. Коммутатор по типу «звезды»

ВС, в которой топология линий связи образует прямоугольную сетку (обычно двух- или трехмерную), называется системой с **топологией решетки.** Подобная топология может быть достаточно просто реализована и, кроме того, эффективно использована при параллельном выполнении многих численных алгоритмов (например, при реализации методов анализа математических моделей, описываемых дифференциальными уравнениями в частных производных). На рис. 8.25 представлена топология связей по типу «двумерной решетки». Каждый ВМ связан с 4-мя соседями и через них с любыми другими ВМ.

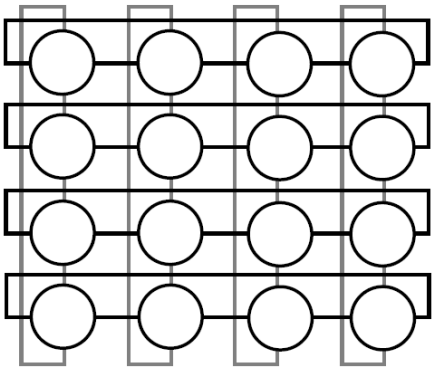


Рис. 8.25. Структура ВС по типу «решетки»

Топология ВС по типу **«гиперкуб»** – частный случай решетки, когда по каждой размерности сетки имеется только два процессора. Примеры гиперкуб-коммутаторов представлены на рис. 8.26. Гиперкуб содержит 2^N процессоров при размерности гиперкуба N.

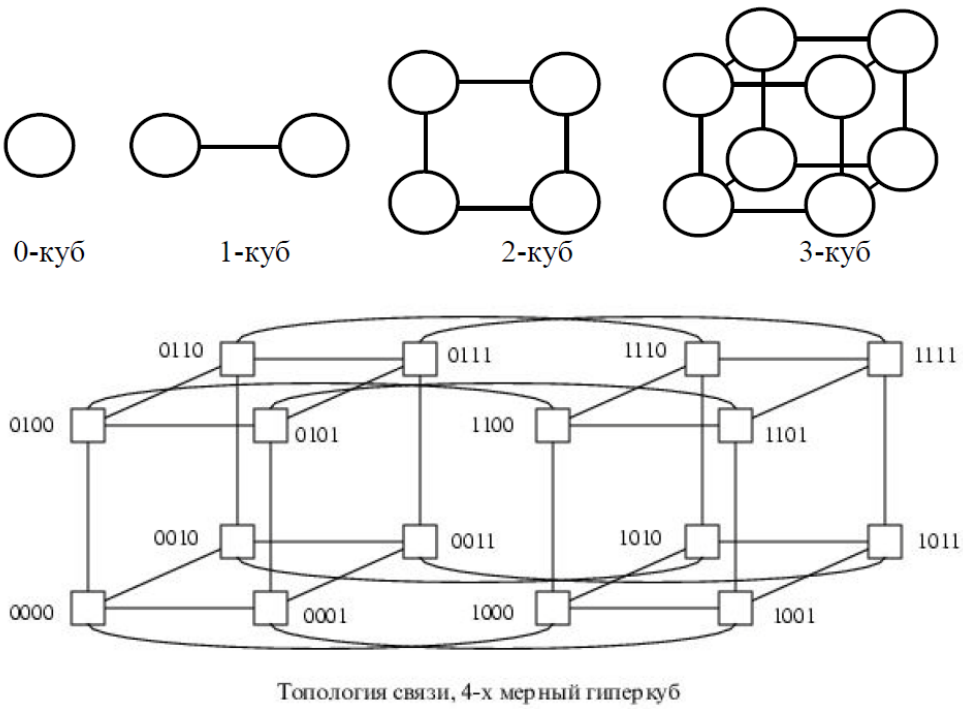


Рис. 8.26. Коммутатор по типу «гиперкуба»

Такой вариант организации сети передачи данных достаточно широко распространен на практике, достаточно прост в построении, и характеризуется следующими свойствами:

1. Два процессора имеют соединение, если двоичные представления их номеров имеют только одну различающуюся позицию.
2. В N-мерном гиперкубе каждый процессор связан ровно с N соседями.
3. Кратчайший путь между двумя любыми процессорами имеет длину, совпадающую с количеством различающихся битовых значений в номерах процессов. Отсюда следует, что максимальная длина пути в N-мерном гиперкубе равна N.
4. N-мерный гиперкуб может быть разделен на два (N–1)-мерных гиперкуба (всего возможно N таких разбиений).

Очевидные преимущества такой топологии:

1. ВМ, располагаясь в вершине N куба, не отстоит более чем на N-ребер ни от какого другого ВМ, что значительно облегчает создание эффективных коммуникаций в системе (например, для N=12 допустимое число ВМ 2^{12} =4096!);
2. Т.к. структура соединений в N-кубе хорошо согласуется с двоичной логикой, то достаточно легко реализуется алгоритм маршрутизации для передачи сообщений между узлами;
3. Между любой парой ВМ существует несколько альтернативных путей коммуникаций, что позволяет в целом снизить задержки при передачах.

Архитектура систем со смешанной организацией памяти

Для решения практических задач часто используют вычислительные системы со смешанной организацией памяти. На рис. 8.27 схематично представлен пример такой архитектуры.

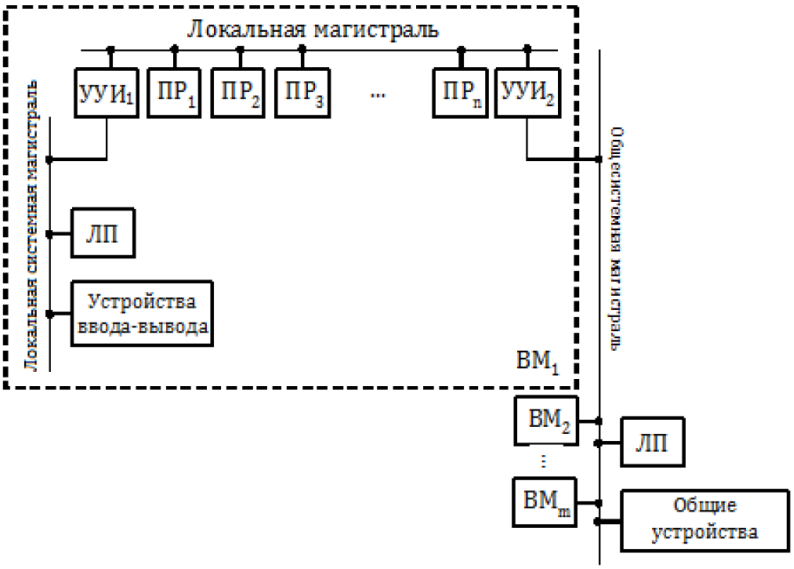


Рис. 8.27. Типовая архитектура систем со смешанной организацией памяти
Здесь: УУИ1, УУИ2 – устройства управления интерфейсами; ПР – процессор; ЛП – локальная память; ВМ – вычислительный модуль.

ЛИТЕРАТУРА

1. Современные операционные системы : Пер. с англ. / Э. Таненбаум. — СПб.: Питер, 2006. — 1038 с.

2. Сетевые операционные системы : Учебник для вузов / В. Г. Олифер, Н. А. Олифер. — СПб.: Питер, 2009. — 668 с.

3. Основы современных операционных систем : Учеб. пособие для вузов / В. О. Сафонов. — М.: ИНТУИТ, 2011. — 583 с.

4. Архитектура ЭВМ и систем : учебное пособие для бакалавров / О. П. Новожилов. — М.: Юрайт, 2015. — 528 с.

5. Архитектура ЭВМ и операционные среды : Учебник для вузов / В. Г. Баула, А. Н. Томилин, Д. Ю. Волканов. — М.: Академия, 2012. — 336 с.

6. Организация ЭВМ и систем : Учебник для вузов / С.А.Орлов, Б.Я. Цилькер — СПб.: Питер, 2015. — 668 с.

Сведения об авторах

Мусихин Александр Григорьевич, кандидат технических наук, доцент кафедры вычислительной техники института информационных технологий Российского технологического университета – МИРЭА.

Смирнов Николай Алексеевич, кандидат технических наук, доцент, профессор кафедры вычислительной техники института информационных технологий Российского технологического университета – МИРЭА.