1. Архитектура операционных систем
   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.)
   1. 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.

|  |
| --- |
| Стек |
| Куча |
| БУП |
| Исполняемый код |
| Данные |

Рис.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 РСВ) – 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 | С |
| 102 | F |
| … |  |  |

* 1. Место операционной системы в ВМ

Место ОС в ВМ поясняется Рис.2. Нижний уровень А – это уровень аппаратуры. Выше размещена ОС. Еще выше размещается прикладное ПО, т.е. те программы, которые выполняют задачи пользователя.

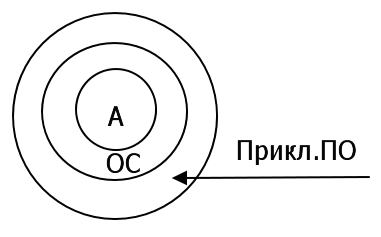


Рис.2. Место ОС в ВМ.

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

* 1. 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 С, и загружается при включении ВМ. Почему диски нумеруются не с буквы А? Раньше буквами А и В обозначались гибкие диски, на диске А размещалась ОС. Современные ОС по объему не вмещаются на гибкий магнитный диск.

* 1. Многослойная структура ОС.

Уровни ОС (слои) обозначаются в виде концентрических колец. (рис.3). На рис.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).

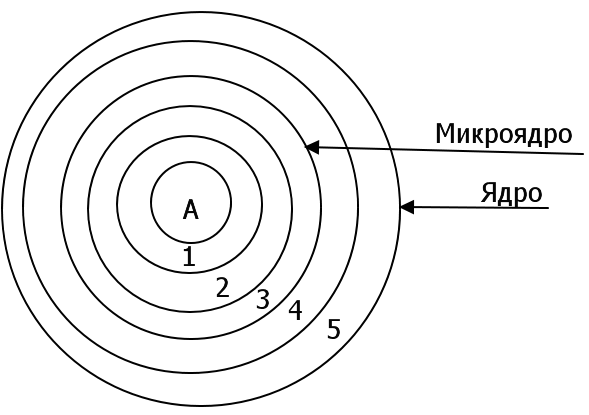


Рис.3. Многослойная структура ОС.

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.).

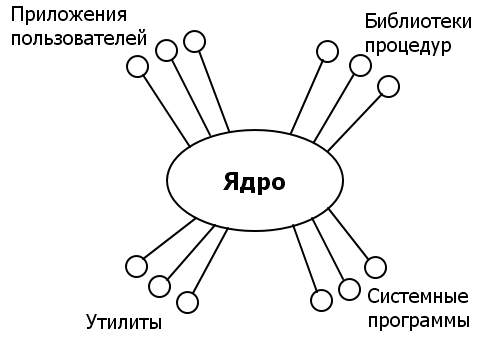


Рис.4. Окружение ОС.

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

* 1. 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.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.

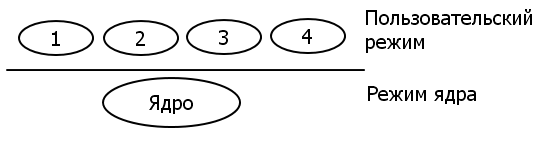
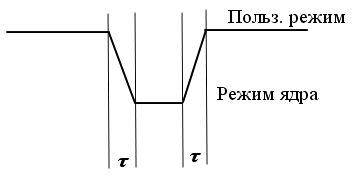
 

Рис.5. Выделение режимов пользовательского и защищенного.

Дальнейшим шагом в совершенствовании ОС стало введение микроядерной 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.

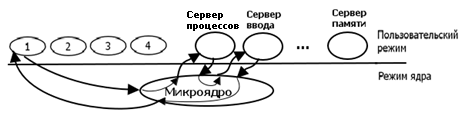


Рис.6. Работа микроядра.

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.

1. 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.

* 1. Страничное распределение.

На рис.7 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 = 220 байт. Физическая память имеет объем 16 Кбайт = 214 байт. Страница имеет объем 4 Кбайт = 212 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 22 = 4 страницы, а виртуальная – 28 = 256 страниц.

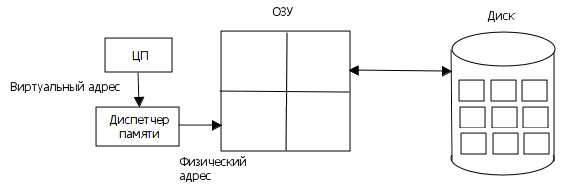


Рис.7. Взаимодействие виртуальной и физической памяти.

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=910 | 1 |  |  |  | 10=210 | 3 дор.2 сект. |
| … |  |  |  |  |  |  |
| 28-1= 25510 |  |  |  |  |  |  |

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;

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

Правила преобразования виртуального адреса в физический при страничном распределении памяти поясняется рис.8.

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.

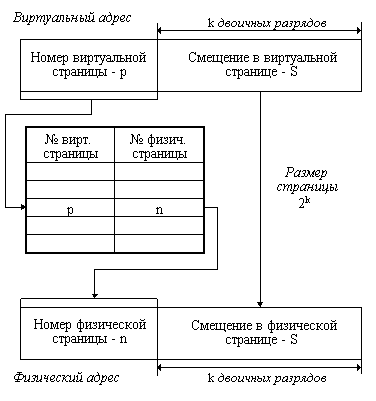


Рис.8. Преобразование виртуального адреса в физический при страничной организации памяти.

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.

* 1. 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.

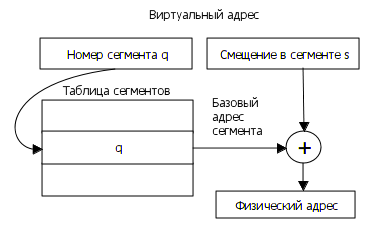


Рис.9. Преобразование виртуального адреса при сегментной организации памяти.

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.

* 1. 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. 9.

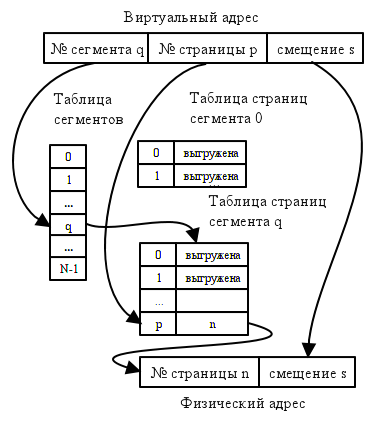


Рис.9. Вычисление адресов при сегментно-страничном распределении памяти.

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.

1. Система прерываний
   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.

* 1. 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.

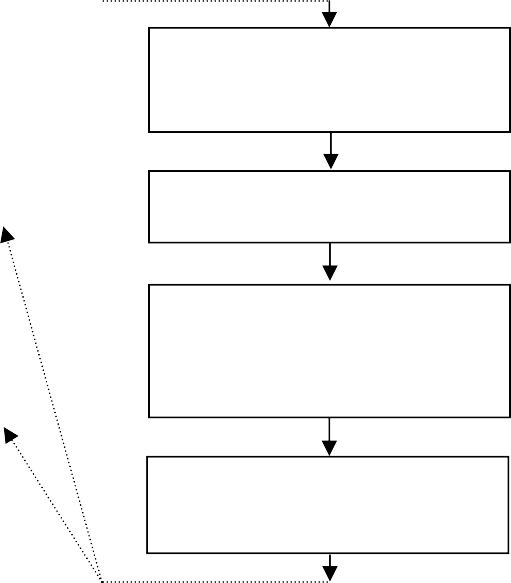
Поскольку для выполнения подпрограммы обработки прерывания используются различные регистры процессора (РОНы, счетчик команд, регистр флагов и т.д.), то информацию, содержащуюся в них в момент прерывания, необходимо сохранить для последующего возврата в прерванную программу.

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. 1).

* 1. 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**.



*Основная программа*

*Запрос прерывания*

*Сохранение вектора состояния программы*

*Обработка прерывания*

*Запрос   
прерывания*

*Восстановление вектора состояния программы*

*Возврат в основную программу*

Рис. 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.1.2).

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.

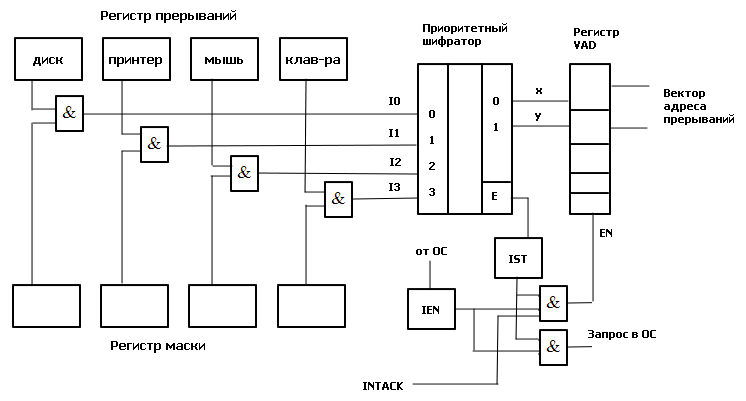


Рис.1.2. Схема обработки требования прерывания от внешнего устройства. IEN – Interrupt Enable – прерывание разрешено; IST – Interrupt Status – статус прерывания; INTACK – подтверждение прерывания; VAD – вектор адреса прерывания.

При поступлении хотя бы одного требования прерывания на входы приоритетного шифратора он устанавливает в единичное состояние свой выход Е 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.

* 1. Цикл прерывания

Управлением циклов в УУ ЦП занимаются триггеры F и R. Рассмотрим, как происходит управление при поступлении требований прерывания. В такте c2t3 цикла выполнения команды вырабатывается сигнал

Если c2t3:

Если c2t3:

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:

C3t0: M[SP] ← PC / запомнить адрес возврата в стеке.

C3t1: INTACK ← 1, VAD ← CD / подтвердить обработку прерывания, занести в VAD вектор прерывания.

C3t2: PC ← VAD / передать в счетчик команд адрес ПОП.

C3t3: IEN ← 0, C ← C0 / 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.

1. Процессы и потоки

Процесс – это программа в ходе ее выполнения. Процессу выделяется память со своими разделами (Рис.1) одним из разделов является блок управления процессом (БУП). Иначе его называют дескриптором процесса или объект- процессом.

|  |
| --- |
| Стек |
| Куча |
| БУП |
| Исполняемый код |
| Данные |

Рис.1

* 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.

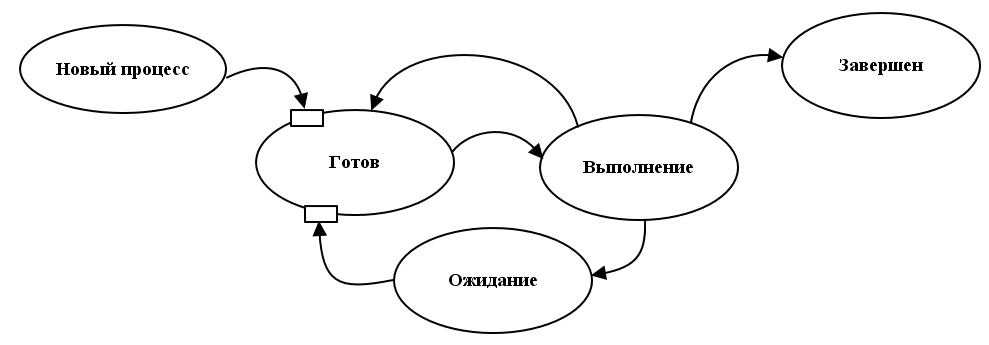


Рис.2. Схема состояний процесса.

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.

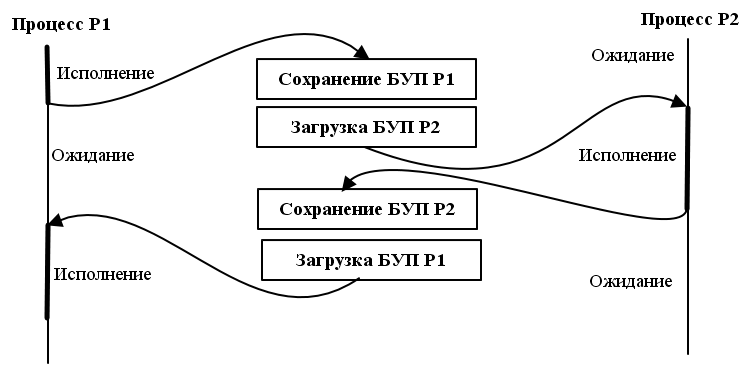


Рис.3. Переключение процессов.

В простейшем случае переключение процессов иллюстрируется рис. 3. На самом деле процедура переключения процессов сложнее. По существу, есть периоды времени, когда не выполняется ни один из процессов.

* 1. 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 многоядерных процессоров. К таким языкам относится, прежде всего, Java. Он с самого начала был многопоточным. Так же многопоточными являются С#, 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?

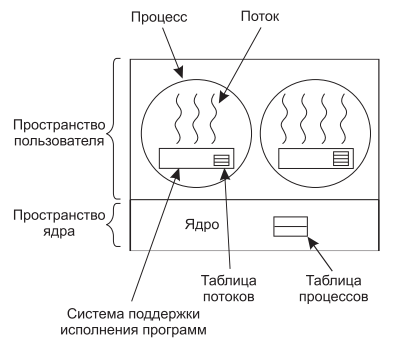


Рис.4. Управление потоками в процессах.

На рис. 4 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.

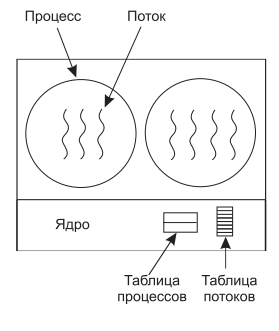


Рис.4.1. Управление потоками в ядре.

Более современный подход представлен на рис.4.1. 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 осуществлении 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».

1. 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).

* 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.
  1. 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 уникальное 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.

* 1. 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. 5.1). Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня (рис. 5.2).

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.

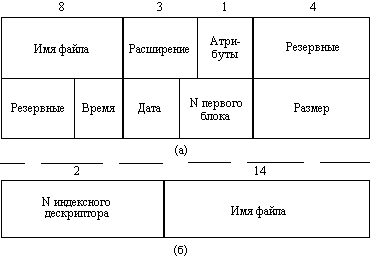


Рис. 5.1. Структура каталогов: а - структура записи каталога MS-DOS (32 байта); б - структура записи каталога ОС UNIX

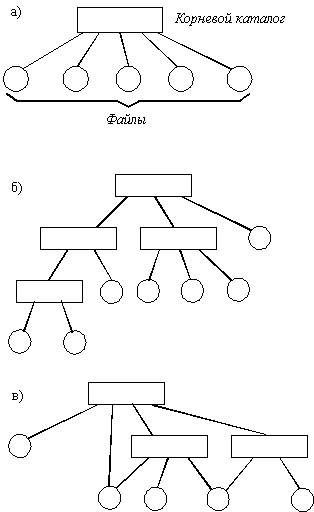


Рис. 5.2. Логическая организация файловой системы:   
а - одноуровневая; б - иерархическая (дерево); в - иерархическая (сеть)

* 1. 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.

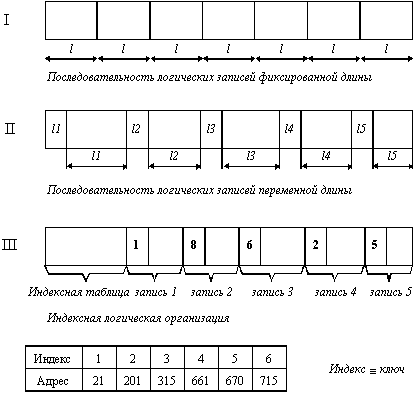


Рис. 6. Способы логической организации файлов

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. 6.1,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.

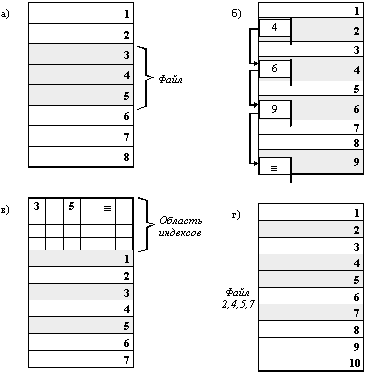


Рис. 6.1. Физическая организация файла   
а - непрерывное размещение; б - связанный список блоков;   
в - связанный список индексов; г - перечень номеров блоков

Следующий способ физической организации - размещение в виде связанного списка блоков дисковой памяти (рис. 6.1,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. 7).

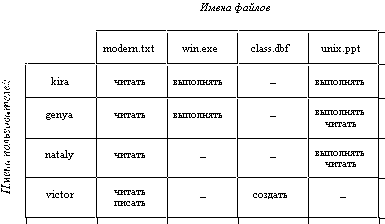


Рис. 7. Матрица прав доступа

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.
  1. 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.

* 1. Общая модель файловой системы

Функционирование любой файловой системы можно представить многоуровневой моделью (рис. 8), 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).

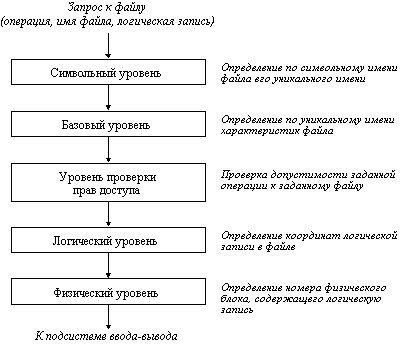


Рис. 8. Общая модель файловой системы

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 непрерывной 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.

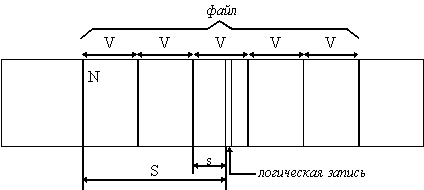


Рис. 9. Функции физического уровня файловой системы

*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. 9 иллюстрирует работу физического уровня для простейшей физической организации файла в виде непрерывной последовательности блоков. Подчеркнем, что задача физического уровня решается 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.

* 1. 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 байтв файле является k-ым байтом сегмента. На рис. 10 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.

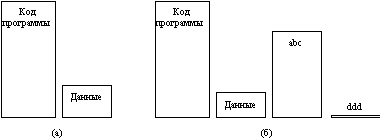


Рис. 10. (а) Сегменты процесса перед отображением файлов в адресное пространство; (б) процесс после отображения существующего файла abc в один сегмент и создания нового сегмента для файла ddd

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.

* 1. 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. 11), 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.

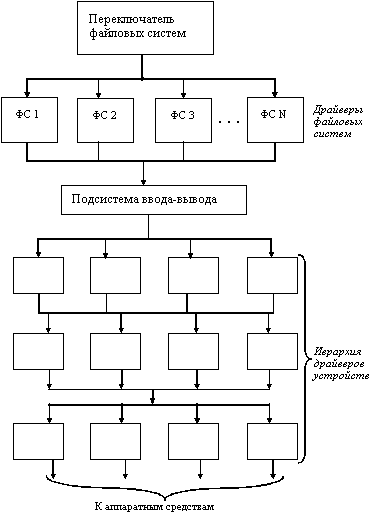


Рис. 11. Архитектура современной файловой системы

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), драйверы, 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.

**31\_03 по 04\_04 Практические занятия Архитектура ВМиС ,** **ИКБО-01-18; ИКБО-02-18; ИКБО-03-18; ИКБО-13-18; ИКБО-12-18; ИКБО-16-18; ИНБО-05-18; ИНБО-06-18; ИНБО-02-18; ИНБО-03-18; ИНБО-01-18; ИНБО-04-18**

**Контрольная работа №1**

**Управляющее слово ЦП**

Цель контрольной работы

Целью работы является проверка усвоения материалов по архитектуре операционного устройства центрального процессора ВМ и взаимодействия входящих в нее элементов.

Управляющее слово ЦП составляется из разрядов управления устройств, входящих в ОУ ЦП.

Для выполнения контрольной работы необходимо уяснить порядок функционирования составляющих операционное устройство элементов вычислительной техники: мультиплексоров, дешифратора, арифметико-логического устройства (АЛУ), регистров и оперативного запоминающего устройства.

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:

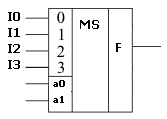


Рис. 1. Схема мультиплексора.

Таблица соответствия

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

Дешифратор

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

Дешифратор на схемах обозначается следующим образом:

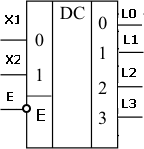


Рис. 2. Схема дешифратора.

Таблица соответствия для двухвходового дешифратора имеет вид:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Е | Х2 | Х1 | 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 Е означает разрешение работы дешифратора. Этот вход обычно делают инверсным. Если разрешение есть, то =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.

Регистр

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

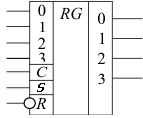


Рис. 3. Схема регистра.

На вход С 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.

Арифметико-логическое устройство

АЛУ выполнено на основе комбинационных схем, выходные состояния которых полностью определяются входными информационными и управляющими сигналами.

Таблица микроопераций АЛУ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| M | S1 | S0 | C | Микрооперации | Наименование |
| 0 | 0 | 0 | 0 | A | Передача А |
| 0 | 0 | 0 | 1 | A+1 | Инкремент А |
| 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 А |
| 0 | 1 | 1 | 1 | A | Передача А |
| 1 | 0 | 0 | Х | A۷B | ИЛИ |
| 1 | 0 | 1 | Х | AB | Искл. ИЛИ |
| 1 | 1 | 0 | Х | A۸B | И |
| 1 | 1 | 1 | Х |  | Не А |

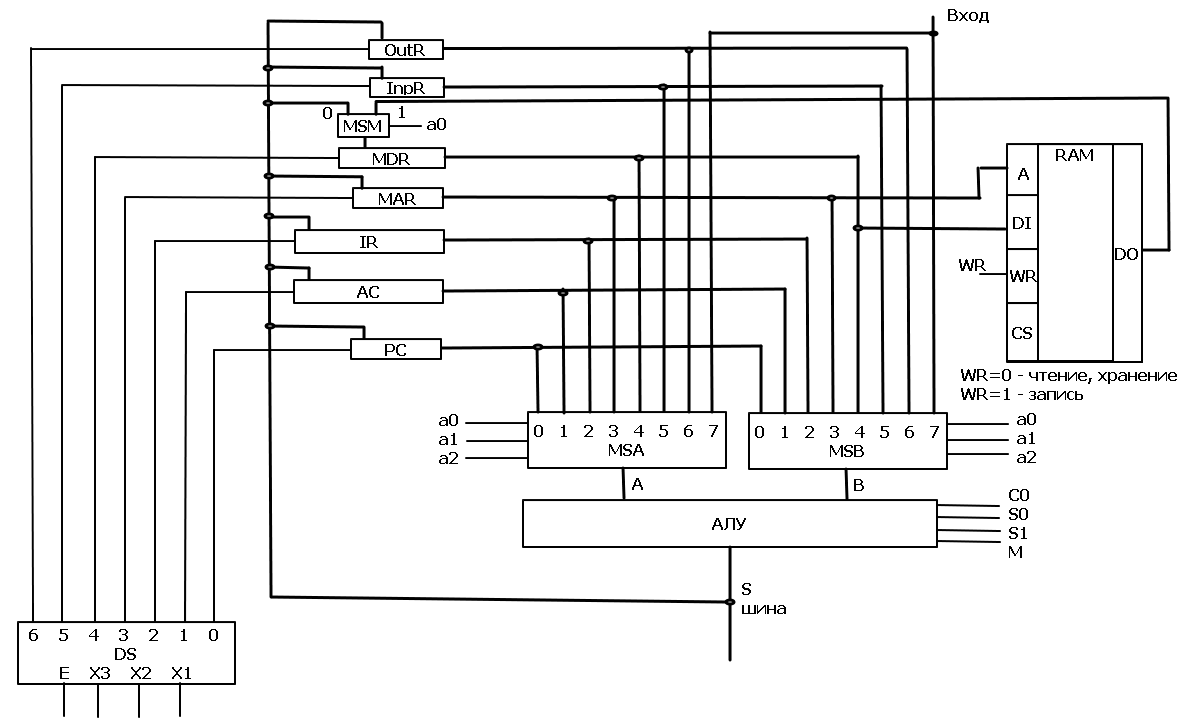
Операционное устройство

Операционное устройство (ОУ) осуществляет операции (арифметические и логические) над входными информационными сигналами. Характер операций, входные и выходные сигналы (регистры, вход, выход и оперативное запоминающее устройство) определяются управляющими сигналами устройств, входящих в состав ОУ.

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MSA | | | | MSB | | | | | АЛУ | | | | | DC | | | | | MSM | | | | WR | |
| **a0** | **a1** | **а2** |  | | **a0** | **a1** | **а2** |  | **M** | **S1** | **S0** | **C0** |  | **E** | **X3** | **X2** | **X1** |  | | **a0** |  | **WR** | |  |

Задача устройства управления – обеспечить выдачу управляющего слова ЦП. Иначе он называется микрокод, который управляет работой аппаратуры ЦП. Обычно он записывается в постоянное запоминающее устройство (ПЗУ) центрального процессора (прошивается).

  
Рис. 4. Схема операционного устройства.

Задача контрольной работы состоит в том, чтобы расписать управляющее слово ЦП для различных микроопераций.

Задание

|  |  |
| --- | --- |
| **№ №варианта** | **Записать управляющее слово** |
| **1** | S🡨 INPR  IR |
| **2** | INPR🡨 IR  MAR |
| **3** | IR🡨 MAR  OUTR |
| **4** | MAR🡨 OUTR – AC |
| **5** | OUTR🡨AC – MDR – 1 |
| **6** | AC🡨 MDR + PC + 1 |
| **7** | MDR🡨 PC + INPR |
| **8** | OUTR🡨 |
| **9** | S🡨 INPR + 1 |
| **10** | INPR🡨 IR + MAR |
| **11** | IR🡨 MAR + OUTR + 1 |
| **12** | MAR🡨 OUTR – AC – 1 |
| **13** | OUTR🡨 AC – MDR |
| **14** | AC🡨 |
| **15** | MDR🡨 |
| **16** | PC🡨 |
| **17** | OUTR🡨 IR – 1 |
| **18** | IR🡨 |
| **19** | PC🡨 |
| **20** | S🡨 |
| **21** | INPR🡨 MAR + OUTR |
| **22** | MDR🡨 OUTR + IR + 1 |
| **23** | AC🡨 IR – PC – 1 |
| **24** | MAR🡨 INPR – AC |
| **25** | PC🡨 |
| **26** | IR🡨 OUTR + PC |
| **27** | OUTR🡨 MDR + INPR + 1 |
| **28** | S🡨 INPR – AC – 1 |
| **29** | MDR🡨 AC – MAR |
| **30** | INPR🡨 |
| **31** | AC🡨 |
| **32** | MAR🡨 |
| **33** | S🡨 OUTR |
| **34** | S🡨 AC+OUTR |
| **35** | S🡨 IR+INPR+1 |
| **36** | S🡨OUTR – MDR |
| **37** | S🡨INPR – MAR – 1 |
| **38** | S🡨IR+1 |
| **39** | S🡨OUTR – 1 |
| **40** | S🡨 AC  PC |

Выводы

Сделать выводы по проделанной работе.

**31\_03\_2020\_4ПАРА\_АрхВМиС\_Лекция\_СМИРНОВ\_Н\_А\_ИНБО-01,-02, -03,04,- 05, 06,18**

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

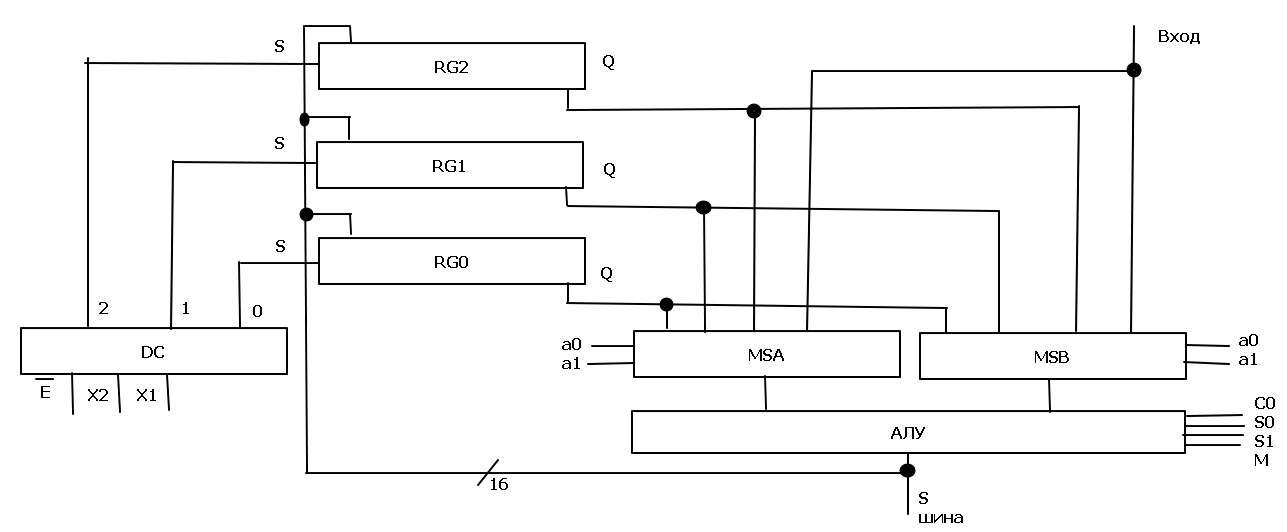


Рис. 1 Схема вспомогательная

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 DС 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 | ХХ | 0000 | 010 |

3. RG0 ← RG2 ۸ вход

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 00 | 110Х | 000 |

4. S ← RG1

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | ХХ | 1111 | 1ХХ |

Рассмотрим более полный вариант схемы ЦП.

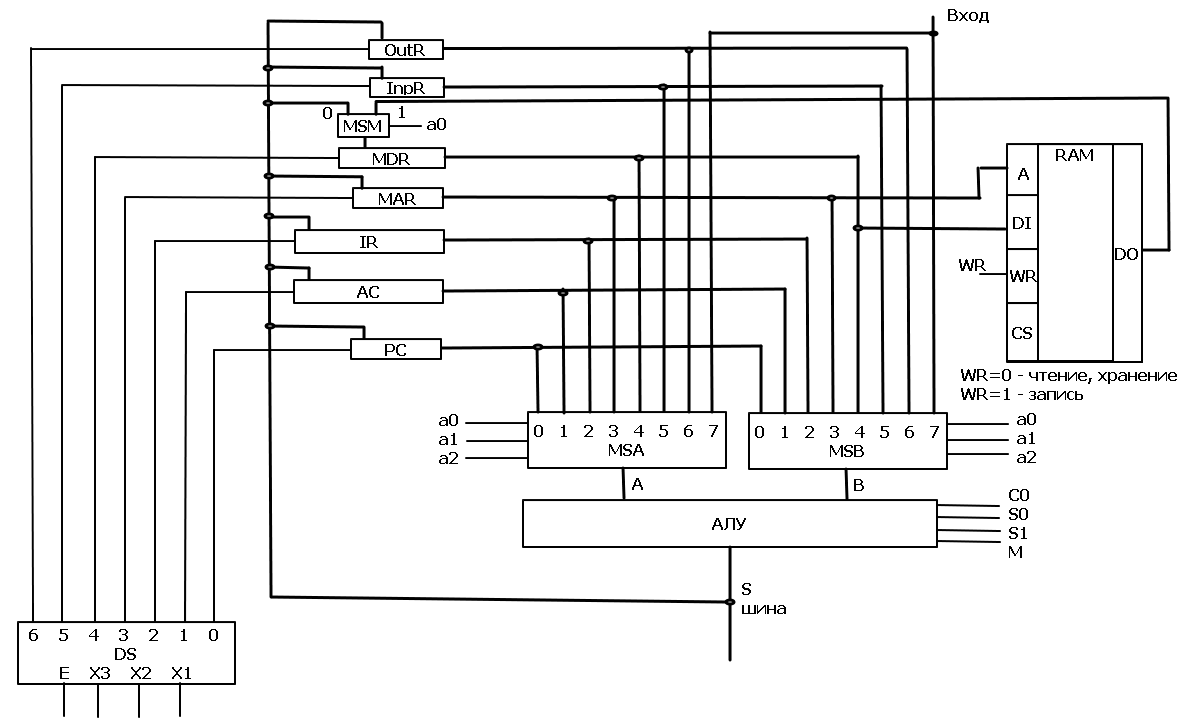


Рис. 2. Схема ОУ ЦП

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MSA | | | | MSB | | | | АЛУ | | | | | DC | | | | | MSM | | | | WR | |
| **a0** | **a1** | **а2** |  | **a0** | **a1** | **а2** |  | **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.

* 1. Устройство управления ЦП

Задача устройства управления – обеспечить выдачу управляющего слова ЦП. Иначе он называется микрокод, 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.

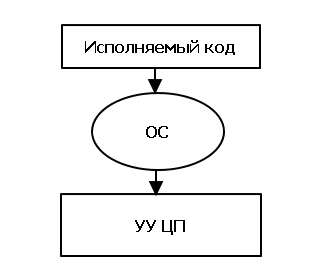


Рис. 3 Алгоритм управления в ВМ

Последовательность работы устройства управления задается циклами.

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

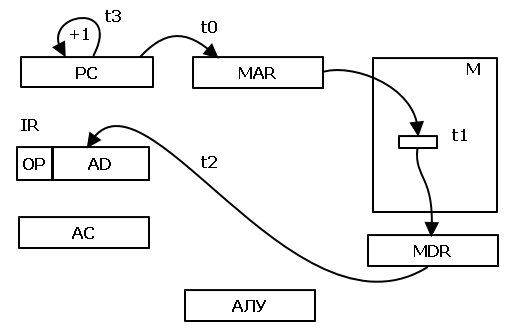


Рис. 4 Цикл выборки команды

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. 4 t0, t1, t2, t3. Следовательно, цикл выборки команды осуществляется за 4 такта.

Само выполнение команды происходит следующим образом:

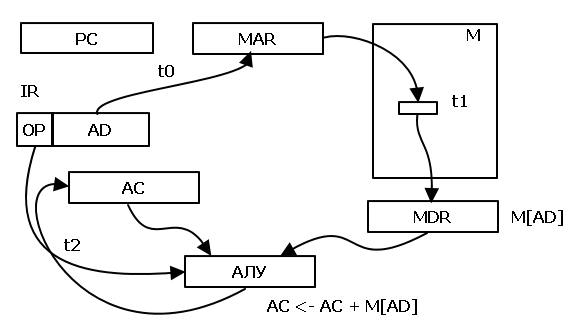


Рис. 5 Цикл выполнения команды

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 АC ← 40.

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

Косвенная адресация: LDA @ 30. /загрузить в аккумулятор слово из ячейки, адрес которой находится в ячейке 30. В результате выполнения команды АC ← 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.

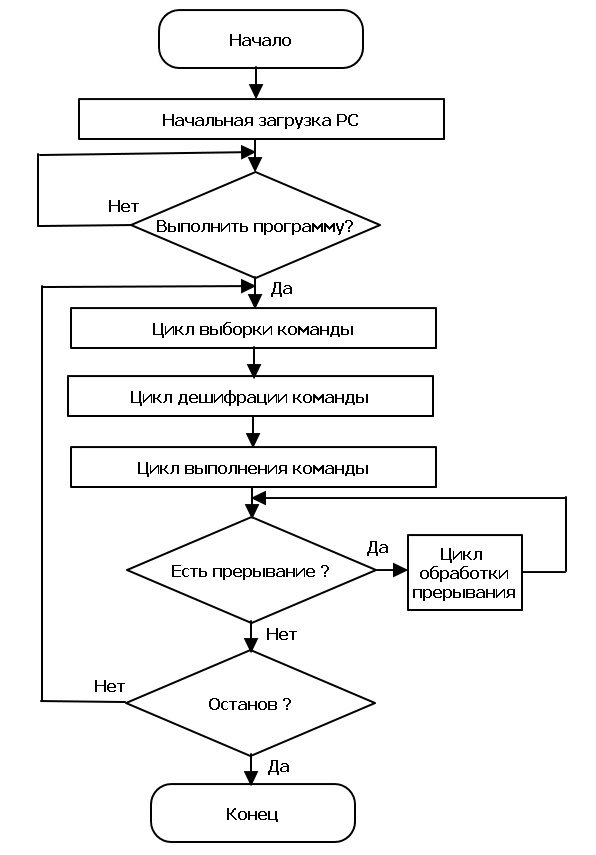


Рис.6. Алгоритм работы устройства управления ЦП ВМ

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. 6.

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.

* 1. Структурная схема УУ ЦП

Структурная схема УУ ЦП изображена на рис. 7.

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. 7.

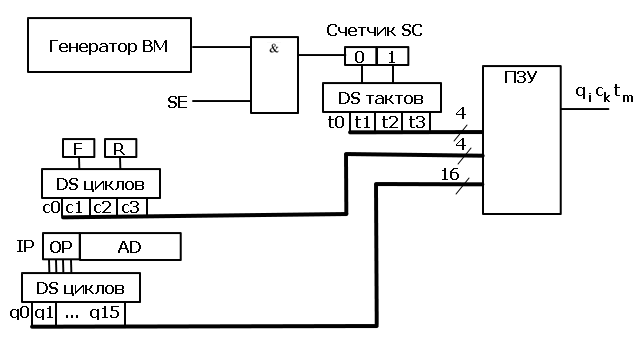


Рис. 7. Структурная схема УУ ЦП

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ГИ |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| t0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| t1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| t2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| t3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Рис. 8. Временная диаграмма работы DС тактов

В ВМ такты не пересекаются по времени.

DС 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 | С0 |
| 0 | 1 | Дешифрации команды | С1 |
| 1 | 0 | Выполнения команды | С2 |
| 1 | 1 | Прерывания | С3 |

DС 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 таблице.

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, С0=1

С0t0 : MAR← PC / подготовить выборку команды из памяти.  
С0t1 : MDR← M[MAR] / прочесть команду из памяти по адресу MAR.  
С0t2 : IR← MDR / поместить команду в регистр команды.  
С0t3 : PC← PC +1, F← 1/ подготовить адрес след. команды, перейти к циклу выполнения команды.

F=1, R=0, С2=1

С2t0 : MAR← IR[AD] / передать адрес операнда в регистр адреса памяти MAR.  
С2t1 : MDR← M[MAR] / прочесть операнд из памяти по адресу MAR.  
С2t2 : AC← AC+MDR / выполнить сложение.  
С2t3 : 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 5А8 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 | DС | MSM | WR | FR | PC | MAR | MDR | IR | AC |
| Исх. состояние регистров | 5А8 |  |  |  | 8721 |
| С0t0 :  MAR← PC | 000 | xxx | 0000 | 0011 | x | 0 | 00 |  | 5А8 |  |  |  |
| С0t1 : MDR← M[MAR] | xxx | xxx | xxxx | 0100 | 1 | 0 | 00 |  |  | 2F12 |  |  |
| С0t2 :  IR← MDR | 100 | xxx | 0000 | 0010 | x | 0 | 00 |  |  |  | 2F12 |  |
| С0t3: PC←PC+1, F←1 | 000 | xxx | 0001 | 0000 | x | 0 | 10 | 5А9 |  |  |  |  |
| q2С2t0 :  MAR← IR[AD] | 010 | xxx | 0000 | 0011 | x | 0 | 10 |  | F12 |  |  |  |
| q2С2t1 : MDR← M[MAR] | xxx | xxx | xxxx | 0100 | 1 | 0 | 10 |  |  | 731F |  |  |
| q2С2t2 : AC← AC+MDR | 001 | 100 | 0010 | 0001 | x | 0 | 10 |  |  |  |  | FA40 |
| q2С2t3 : 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:

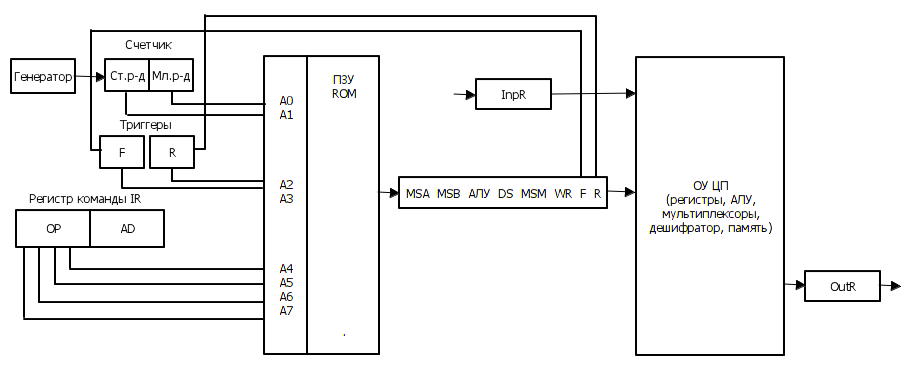


Рис. 9. Структурная схема ЦП.

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.