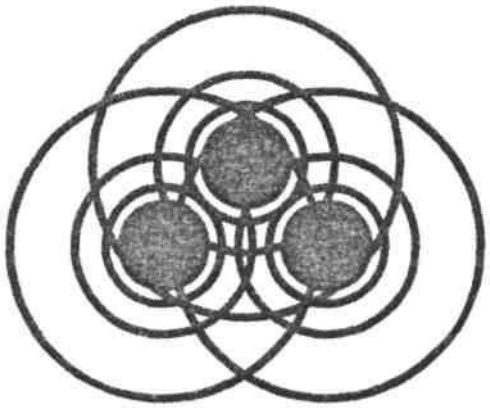


ЧАСТЬ II

Архитектура микропроцессорных систем

Интегральная технология (ИТ) за первые 20—30 лет своего развития достигла таких относительных темпов роста характеристик качества, которых не знала ни одна область человеческой деятельности (включая и такие бурно растущие, как авиация и космонавтика). Действительно, рассмотрим динамику изменений основных параметров ИТ за первые 20 лет ее развития (1960— 1980 гг.):

- степень интеграции N увеличилась на 5—6 порядков;
- площадь транзистора S уменьшилась на 3 порядка;
- рабочая частота f увеличилась на 1—3 порядка;
- факторы добротности:
  - $f \times N$ *увеличился на 5—7 порядков*;
  - $P \times t$ уменьшился на 4 порядка, где t— задержка на элементе, P — мощность, рассеиваемая элементом;
- надежность (при сопоставлении элементо-часов) увеличилась на 4—8 порядков;
- производительность технологии (в транзисторах) увеличилась на 4—6 порядков;
- цена на транзистор в составе ИС уменьшилась на 2—4 порядка.

Американцы подсчитали, что если бы авиапромышленность в те же годы имела аналогичные темпы роста соответствующих показателей качества (стоимость — скорость — расход топлива = стоимость — быстродействие — рассеиваемая мощность), то "Боинг 767" стоил бы $500, облетал земной шар за 20 мин и расходовал на этот полет 10 л горючего.

Успехи ИТ в области элементной базы позволяли "поглощать" кристаллом все более высокие уровни ЭВМ: сначала— логические элементы, потом — операционные элементы (регистры, счетчики, дешифраторы и т. д.), далее — операционные устройства. Степень функциональной сложности, достигнутой в ИС, определяется особенностью технологии, разрешающей способностью инструмента, а также структурными особенностями схемы: регулярностью, связностью.

Под *регулярностью схемы* здесь будем понимать степень повторяемости элементов и связей по одной или двум координатам (при размещении структуры на плоскости). *Связность* — число внешних выводов схемы.

Кроме того, следует иметь в виду, что выпуск ИС был экономически оправдан лишь для функционально универсальных схем, обеспечивающих их достаточно большой тираж.

С этой точки зрения интересно взглянуть на соотношение ИС логики и памяти в процессе эволюции ИС — СИС — БИС — СБИС. Первые ИС (степень интеграции $N \sim 10^1$) были исключительно логическими элементами. При достижении N примерно 10^2 стали появляться, наряду с операционными элементами, первые элементы памяти объемом в 16 — 64 — 128 битов.

По мере дальнейшего роста степени интеграции память стала быстро опережать "логику", т. к. по всем трем параметрам (регулярность, связность, тираж) имела перед логическими схемами преимущество. Действительно, структура накопителя ЗУ существенно регулярна (повторяемость элементов и связей по двум координатам), связность ее растет пропорционально логарифму объема (при увеличении объема памяти вдвое и сохранении без изменения способа доступа в БИС достаточно добавить лишь один вывод). Наконец, память "нужна всем" и "чем больше, тем лучше", особенно если "больше, но за ту же (почти) цену".

Что касается ИС логики, то на уровне $N \sim 10^3$ на кристалле можно уже размещать устройство ЦВМ (например, АЛУ, ЦУУ), но схемы логики (особенно управление) существенно нерегулярны, их связность (сильно зависящая от конкретной схемы) растет примерно пропорционально N, причем такие схемы, как правило, не являлись универсальными и не могли выпускаться большими тиражами (исключения в то время — БИС часов и калькуляторов).

Разработка первого микропроцессора (МП) — попытка создать универсальную логическую БИС, которая настраивается на выполнение конкретной функции после изготовления средствами программирования. На подобную БИС — МП первоначально предполагалось возложить лишь достаточно произвольные управляющие функции, однако позже МП стал использоваться как элементная база ЦВМ четвертого и последующих поколений. Появление МП вызвало необходимость разработки целого спектра универсальных логических БИС, обслуживающих МП: контроллеры прерываний и прямого доступа в память (ПДП), шинные формирователи, порты ввода/вывода и др.

Первый МП был разработан фирмой Intel и выпущен в 1971 г. на основе *p-МОП-технологии* (i4004). В 1972 и 1973 годах этой же фирмой были выпущены модели i4040, i8008. Эти микропроцессоры относились к т. н. *первому поколению,* обладали весьма ограниченными функциональными возможностями и очень быстро были вытеснены вторым поколением, которое было реализовано на основе n-МОП-технологии, что позволило, прежде всего, поднять тактовую частоту примерно на порядок относительно микропроцессоров первого поколения. Кроме того, прогресс интегральной технологии позволил повысить степень интеграции транзисторов на кристалле, а следовательно, увеличить сложность схемы.

Микропроцессоры *второго поколения,* самым распространенным из которого был выпущенный в 1974 г. i8080 (отечественный аналог— К580ВМ80), отличались достаточно развитой системой команд, наличием подсистем прерывания, прямого доступа в память, снабжался достаточным числом вспомогательных БИС, обеспечивающих управление памятью, параллельный и последовательный обмен с внешними устройствами, реализацию векторных прерываний, ПДП и др.

Многие идеи, заложенные в архитектуру систем на базе 8-разрядного микропроцессора i8080, неизменными используются и в современных мощных микропроцессорах.

Постоянное стремление к увеличению быстродействия ЭВМ привело разработчиков микропроцессоров "на поле" биполярной интегральной технологии, прежде всего — ТТЛ, где были выпущены микропроцессоры, отнесенные к *третьему поколению,* причем архитектура этих микропроцессоров существенно отличалась от их предшественников.

Известно, что для любого электронного прибора справедливо соотношение:

$$\Delta P \cdot \Delta t = \text{const},$$

где ΔP — энергия переключения, Δt — время переключения.

ТТЛ-транзисторы в составе ИС обладали (в то время) на порядок большим (по сравнению с n-МОП) быстродействием и соответственно на порядок большим потреблением мощности. Технологические трудности в то время не позволяли широко использовать активные способы отвода тепла от кристалла, поэтому единственный способ сохранения работоспособности кристалла в этих условиях — снижение степени интеграции.

Первый из выпущенных микропроцессоров третьего поколения— i3000 был двухразрядным! Очевидно, сохранение в этом случае традиционной архитектуры, характерной для микропроцессоров второго поколения, не привело бы к увеличению производительности системы, несмотря на то, что тактовая частота кристалла увеличивалась значительно (на порядок).

Решение этой проблемы повлекло значительные структурные изменения в микропроцессорах третьего поколения по сравнению со вторым:

- микропроцессоры выпускались в виде секций со средствами межразрядных связей, позволяющими объединять в одну систему произвольное число секций для достижения заданной разрядности. В состав секций включалось АЛУ, РОН и некоторые элементы устройства управления;
- устройство управления выносилось на отдельный кристалл (группу кристаллов), общий для всех процессорных секций;
- за счет резерва внешних выводов (малая разрядность) предусматривались

отдельные шины адреса, ввода и вывода данных, причем данные от разных источников вводились по различным шинам;

- кристаллы управления представляли собой управляющий автомат с программируемой логикой, что позволяет достаточно легко реализовать прак тически любую систему команд на фиксированной структуре операцион ного устройства.

Таким образом, разработчики систем на базе микропроцессоров третьего по коления получали две "дополнительные степени свободы" — возможност выбрать произвольную разрядность процессора (кратную разрядности секции) и самостоятельно реализовать практически произвольную систем команд, оптимизированную для решения задач конкретного класса.

Поскольку микропроцессор в такой архитектуре размещался на нескольких кристаллах БИС: арифметико-логические секции, схемы управления вместе с БИС памяти микрокоманд, вспомогательные БИС (например, схемы ускоренного распространения переноса для АЛС) и др., то подобные микропроцессоры стали называть *многокристальными,* в отличие от *однокристальных* микропроцессоров второго поколения.

Очевидно и то, что разработка систем на многокристальных микропроцессорах требовала значительно больших усилий, времени и квалификации разработчиков, по сравнению с разработкой системы на "готовых" микропроцессорах второго поколения с фиксированной структурой и системой команд.

В конце 70-х и начале 80-х годов прошлого века значительное число отечественных и зарубежных фирм разрабатывали и выпускали серии БИС многокристальных микропроцессоров, причем разрядность секций постепенно увеличивалась до 4, 8 и даже 16 битов.

К тому времени технология уже не являлась решающим фактором классификации МП, ибо появились разновидности технологий одного типа, обеспечивающие очень широкий спектр характеристик МП, широкое распространени получили комбинированные технологии (например, $И^2Л$ + ТТЛШ). Поэтом многокристальные МП выпускались как по биполярной, так и по МДП-технологиям.

Одной из наиболее удачных разработок этого направления можно считат комплект БИС серии Am2900 фирмы AMD и близкую ему по архитектуре отечественную серию К1804 [13].

Параллельно интенсивно развивалась архитектура однокристальных микропроцессоров, наиболее характерным представителем которой можно считать семейство *x*86 фирмы Intel. Развитие этого направления отличал безудержный рост производительности процессоров, обусловленный увеличением разрядности процессоров, тактовой частоты, реализацией параллелизма на всех уровнях работы процессора и применением других архитектурных решений, характерных ранее для "больших" ЭВМ.

Быстро возрастающие возможности микропроцессоров позволяли "захватывать" в область цифровой обработки информации все новые сферы человеческой деятельности (достаточно вспомнить появление и распространение персональных ЭВМ).

Однако в сфере применения микропроцессоров всегда существовали задачи, для решения которых не требовалась высокая производительность процессора (например, управление несложным инерционным технологическим оборудованием, бытовыми приборами). В этих случаях на первый план выступали такие параметры, как надежность, простота реализации (стоимость). Для решения таких задач использование мощных однокристальных микропроцессоров становилось существенно избыточным.

Возрастающие возможности технологии в этом случае использовались не для увеличения производительности процессора, а для размещения на кристалле, наряду с относительно простым процессором, тех устройств, которые в традиционной архитектуре располагались на плате рядом с микропроцессором в виде отдельных БИС (СИС): тактовый генератор, ПЗУ, ОЗУ, порты параллельного и последовательного обмена, контроллер прерываний, таймеры и др.

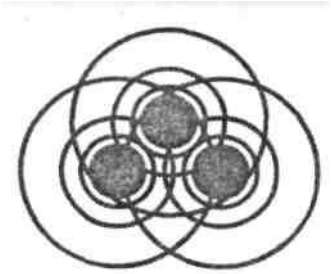
Таким образом, были получены полностью "самодостаточные" *однокристальные микроЭВМ* (ОМЭВМ). Это направление стало интенсивно развиваться, вначале на базе 8-

разрядной архитектуры. Наиболее популярными из них можно считать ОМЭВМ семейств MCS-51 фирмы Intel, MC68HCI1 фирмы Motorola, PIC 16 и PIC 18 фирмы Microchip.

По мере развития на кристаллах ОМЭВМ стали, помимо перечисленных выше устройств, размещать аналого-цифровые и цифроаналоговые преобразователи, блоки энергонезависимой памяти (EEPROM), сложные таймерные системы, схемы управления специализированными ВУ (например, семисегментной индикацией) и др.

Дальнейшее развитие технологии привело к появлению 16- и даже 32-разрядных однокристальных микроЭВМ (наиболее известные— от фирмы Motorola), включающих, наряду с мощным центральным процессором, специализированные процессоры — таймерный и ввода/вывода, работающие независимо от центрального, широкий набор блоков памяти и внешних устройств. Модульность архитектуры кристалла ОМЭВМ позволяет в рамках адного семейства варьировать в широких пределах набор параметров кристалла: состав и объем блоков памяти, набор внешних устройств и даже тип помещаемого на кристалл центрального процессора.

Таким образом, пользователю предоставляется возможность выбора в очень широких пределах архитектуры и параметров ОМЭВМ. При этом он получает "готовую" ЭВМ, не требующую схемотехнических и архитектурных доработок. В итоге современные ОМЭВМ практически полностью заняли ту нишу, в которой долгое время существовали многокристальные микропроцессоры.



ГЛАВА 6

Базовая архитектура микропроцессорной системы

Пожалуй, наиболее популярными в мире (и в нашей стране) являлись и являются однокристальные микропроцессоры семейства *x*86 фирмы Intel. Семейство берет свое начало от первого 8-разрядного микропроцессора i8080 (отечественный аналог— К580ВМ80) и включает 16- и 32-разрядные микропроцессоры i8086, i80286, i80386, i80486, Pentium,..., Pentium 4.

Схемотехнические решения систем на i8080 можно было бы считать базовыми, но его система команд значительно отличается от языка старших моделей микропроцессоров семейства. Поэтому "родоначальником" семейства принято считать первый 16-разрядный микропроцессор— i8086 (отечественный аналог— К1810ВМ86), на котором, кстати, были реализованы персональные ЭВМ IBM PC XT.

Анализ архитектуры микропроцессорных систем (МПС) целесообразно начинать с рассмотрения простейшей (базовой) модели, отражающей основные принципы организации процессора, его системы команд, функционирование основных подсистем. Большинство принципиальных решений, реализованных в МПС на базе младших моделей семейства, сохранились и в старших моделях.

Рассмотрим кратко организацию МПС на базе микропроцессора i8086. При этом выделим для рассмотрения следующие подсистемы:

- □ процессорный модуль;
- □ память;
- □ ввод/вывод;
- □ прерывания;
- □ прямой доступ в память со стороны ВУ.

6.1. Процессорный модуль

Процессорный модуль — основная часть любой МПС. Помимо собственно микропроцессора, он включает ряд вспомогательных схем, без которых МПС не может функционировать (тактовый генератор, интерфейсные схемы и др.).

6.1.1. Внутренняя структура микропроцессора

Структурная схема микропроцессора i8086 представлена на рис. 6.1. Микропроцессор включает в себя три основных устройства:

- УОД — устройство обработки данных;
- УСМ — устройство связи с магистралью;
- УУС — устройство управления и синхронизации.

УОД предназначено для выполнения команд и включает в себя 16-разрядное АЛУ, системные регистры и другие вспомогательные схемы; блок регистров (РОН, базовые и индексные) и блок микропрограммного управления.

УСМ обеспечивает формирование 20-разрядного физического адреса памяти и 16-разрядного адреса ВУ, выбор команд из памяти, обмен данными с ЗУ, ВУ, другими процессорами по магистрали. УСМ включает в себя сумматор адреса, блок регистров очереди команд и блок сегментных регистров.

УУС обеспечивает синхронизацию работы устройств МП, выработку управляющих сигналов и сигналов состояния для обмена с другими устройствами, анализ и соответствующую реакцию на сигналы других устройств МПС.

Микропроцессор i8086 может работать в одном из двух режимов — *минимальном* и *максимальном.* Минимальный режим предназначен для реализации однопроцессорной конфигурации МПС с организацией, подобной МПС на базе i8080, но с увеличенным адресным пространством, более высоким быстродействием и значительно расширенной системой команд. Максимальный режим предполагает наличие в системе нескольких микропроцессор, работающих на общую системную шину. МПС на базе i8086 с использованием максимального режима не получили широкого распространения. Более того, в последующих моделях своих микропроцессоров (80286, 80386, 80486) фирма Intel отказалась от поддержки мультипроцессорной архитектуры. Поэтому мы здесь не будем рассматривать особенности организации максимального режима.

На внешних выводах МП i8086 широко используется принцип *мультиплексирования сигналов* — передача разных сигналов по общим линиям с разделением во времени. Кроме того, одни и те же выводы могут использоваться для передачи разных сигналов в зависимости от режима (min — max).

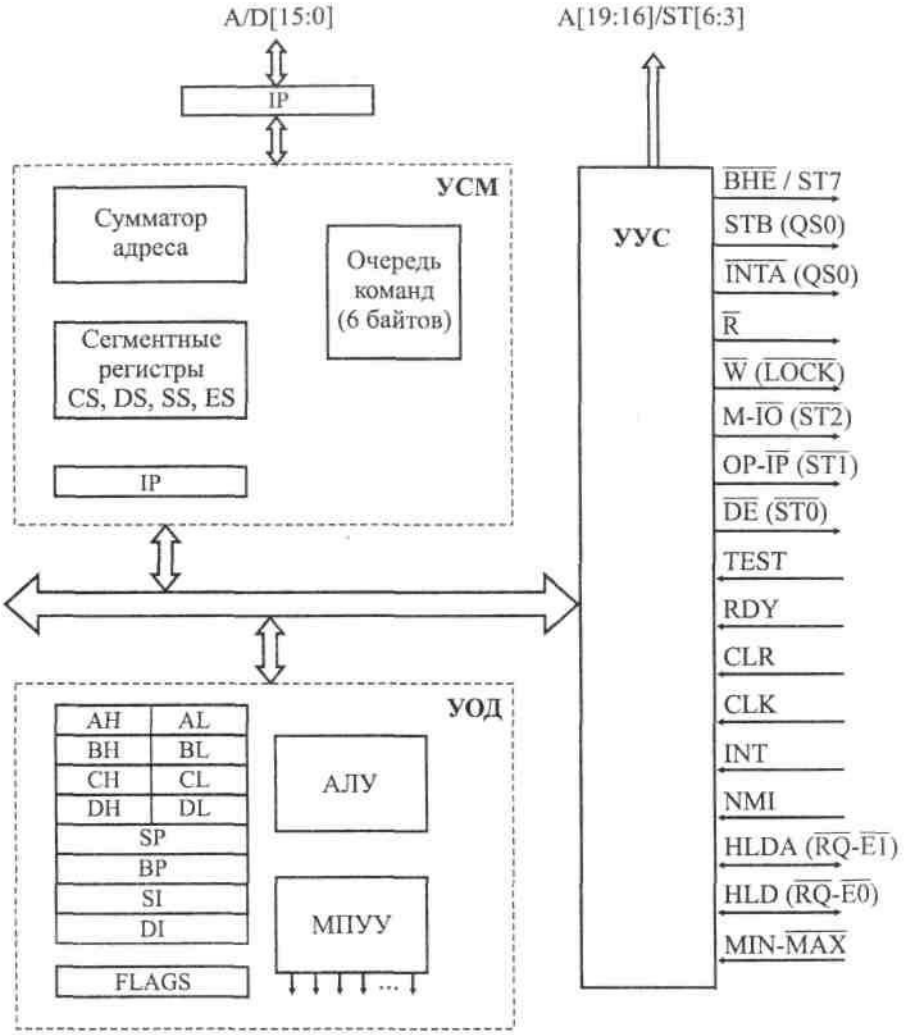


Рис. 6.1. Структура микропроцессора S8086

В табл. 6.1 приведено описание внешних выводов МП i8086. При описани выводов косой чертой (/) разделены сигналы, появляющиеся на выводе в разные моменты машинного цикла. В круглых скобках указаны сигналы, характерные только для максимального режима. Символ * после имени сигнала — знак его инверсии.

Таблица 6.1. Внешние выводы МП i8086

Внешний вывод	Описание
A/D[15:0]	Младшие 0—15 разряды адреса/данные
A[19:16]/ST[6:3]	Старшие 16—19 разряды адреса/сигналы состояния
BHE* /ST[7]	Разрешение передачи старшего байта данных/сигнал состояния
STB(QSO)	Строб адреса (состояние очереди команд)
R*	Чтение
W* /(LOCK*)	Запись (блокировка канала)
M-IO*(ST2*)	Память — внешнее устройство (состояние цикла)
OP-IP*(ST1*)	Выдача/прием (состояние цикла)
DE*(STO*)	Разрешение передачи данных (состояние цикла)
TEST*	Проверка
RDY	Готовность
CLR	Сброс
CLC	Тактовый сигнал
INT	Запрос внешнего прерывания
INTA*(QS1)	Подтверждение прерывания (состояние очереди команд)
NMI	Запрос немаскируемого прерывания
HLD(RQ*/EO)	Запрос ПДП (запрос/подтверждение доступа к магистрали)
NLDA(RQ* /EI)	Подтверждение ПДП (запрос/подтверждение доступа к магистрали)
MIN/MAX*	Потенциал задания режима (min = 1, max = 0)

6.1.2. Командный и машинный циклы микропроцессора

Микропроцессор i8086 работает в составе МПС, обмениваясь с памятью и ВУ словами длиной 2 байта, т. к. разрядность шины данных составляет 16 битов. В основе работы микропроцессора лежит *командный цикл* — действия по выбору из памяти и выполнению одной команды.

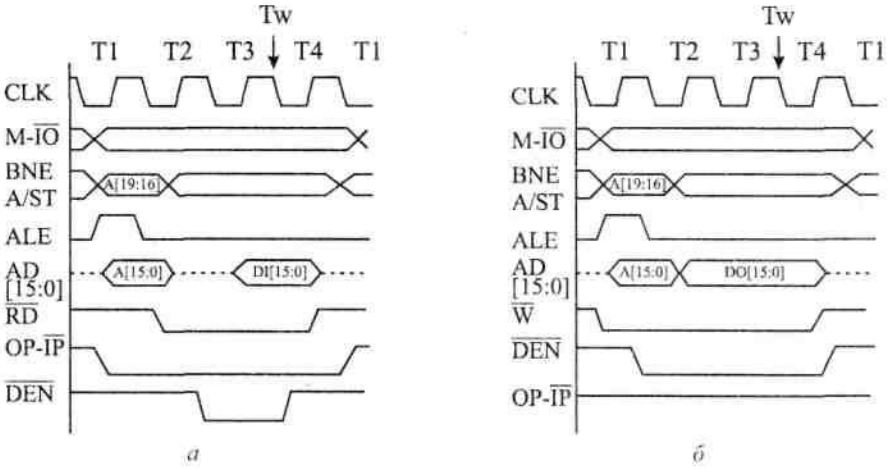
Любой командный цикл (КЦ) начинается с извлечения из памяти первого слова команды по адресу, хранящемуся в счетчике команд (PC). Команды i8086 могут иметь длину от 1 до 6 байтов, причем в первом слове содержится информация о длине команды. Таким образом, для извлечения из памяти одной команды может потребоваться одно или несколько обращений к ОЗУ. В зависимости от типа и формата команды, способов адресации и числа операндов командный цикл может включать в себя различное число обращений к памяти и ВУ, поскольку кроме чтения самой команды в КЦ может потребоваться чтение операндов и размещение результата.

Хотя обращения к ЗУ/ВУ располагаются в разных частях КЦ, выполняются они по единым правилам, соответствующим интерфейсу МПС, и реализованы на общем оборудовании управляющего автомата. Действия МПС по передаче в (из) МП одного слова команды (данных) называются *машинным циклом.* КЦ состоит из одного или нескольких машинных циклов (МЦ).

Машинный цикл включает выдачу процессором адреса памяти или внешнего устройства, по которому производится обращение, выдачу управляющих сигналов, характеризующих тип машинного цикла и направление передачи данных (M-IO, OP-IP), выдачу синхронизирующих (стробирующих) сигналов (STB, R, W) и собственно передачу данных. В i8086 реализована мультиплексированная шина адреса/данных. Это объясняется дефицитом внешних выводов кристалла и требует дополнительного такта для выдачи

адреса и дополнительного управляющего сигнала STB, идентифицирующего наличие адреса на общей шине A/D.

По большому счету разнообразие МЦ сводится к двум разновидностям — *чтению* (данные или команды принимаются в процессор) и *записи* (данные выдаются из процессора). Временные диаграммы соответствующих МЦ приведены на рис. 6.2.



а *б*

Рис. 6.2. Машинные циклы микропроцессора i8086:

а — цикл ЧТЕНИЕ; *б* — цикл ЗАПИСЬ

Цикл начинается с формирования в такте T1 сигнала M-IO, определяющего тип устройства — память или ВУ, с которым осуществляется обмен данными. Длительность сигнала M-IO равна длительности машинного цикла, и он используется для селекции адреса устройств. В T1 и в начале T2 МП выдает адреса A[19:16] и A[15:0] и сигнал BHE, который вместе с A0 определяет выбор передачи либо всего слова, либо одного из его байтов. По спаду строба ALE адрес фиксируется во внешних регистрах-защелках. В такте T2 происходит переключение шин: на выводы A[19:16]/ST[6:3] поступают сигналы состояния; а выводы A/D[15:0] используются для приема/передачи данных.

Описанные выше машинные циклы являются *синхронными;* их длительность определяется только процессором. Однако такой обмен возможен лишь с устройствами, быстродействие которых не уступает процессорному. В противном случае микропроцессор должен реализовать *асинхронный* способ обмена, включающий анализ сигнала от устройства о готовности к обмену или о завершении процедуры обмена.

Роль такого сигнала в i8086 (и всех процессорах старших моделей семейства x86) играет вход RDY (от англ. *ready*— готовность), который всегда должен быть активным при синхронном обмене (с "быстрыми" устройствами). При обмене с "медленными" устройствами значение RDY должно оставаться неактивным (в разных процессорах активным для RDY может быть уровень логической 1 или логического 0) до тех пор, пока устройство, с которым связывается процессор, не завершит процедуру обмена, сообразуясь со своим быстродействием.

Время ожидания процессором готовности устройства может быть сколь угодно большим. Для этого в такте T3 процессор проверяет значение сигнала RDY, и если он неактивен, после такта T3 в машинный цикл вставляется произвольное количество тактов ожидания Tw, в каждом из которых анализируется значение RDY. При появлении активного значения RDY микропроцессор переходит к такту T4 и завершает МЦ. Таким образом, удается согласовывать работу микропроцессора с устройствами различного быстродействия.

6.1.3. Реализация процессорных модулей и состав линий системного интерфейса

Большинство микропроцессоров не могут работать в составе МПС без некоторых дополнительных схем, составляющих вместе с микропроцессором т. н. *процессорный модуль.* Прежде всего, на вход CLK микропроцессора необходимо подать прямоугольные импульсы тактовой частоты от специального внешнего тактового генератора.

Для микропроцессора i8086 частота тактовых импульсов может лежать в диапазоне 2—6 МГц.

На рис. 6.3 приведен один из вариантов упрощенной функциональной схемы процессорного модуля на базе i8086. На схеме не показаны некоторые элементы и связи (например, схема начального сброса и др.).

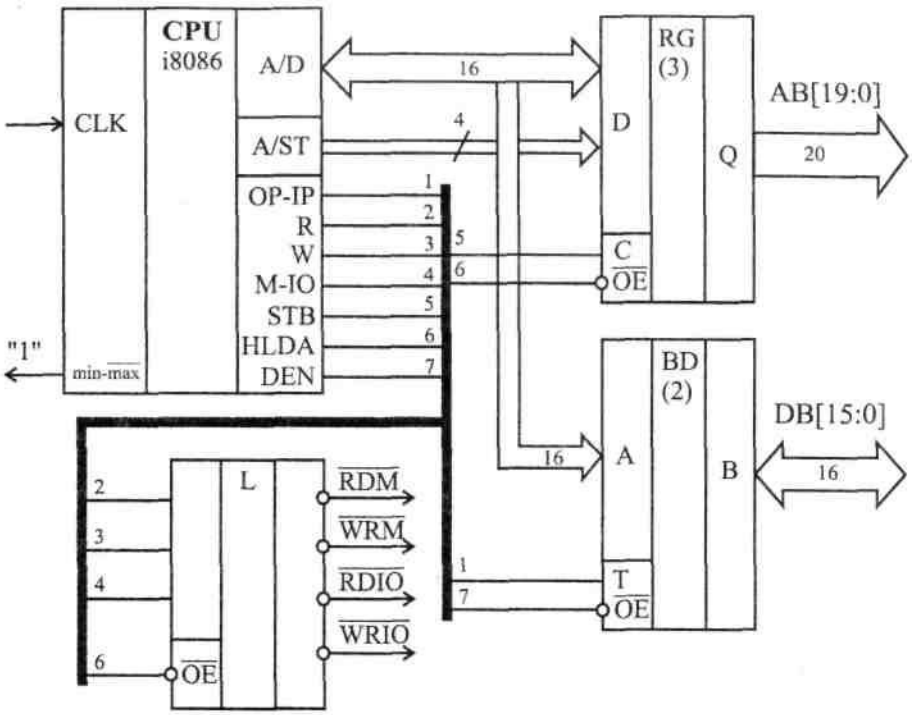


Рис. 6.3. Структура процессорного модуля на базе микропроцессора i8086

Микропроцессор i8086 реализован по n-МДП-технологии, и его выходные каскады не обеспечивают достаточной нагрузочной способности для линий системного интерфейса. Поэтому к выходным линиям микропроцессора обычно подключают буферные схемы BD, реализованные по технологии ТТЛ.

Кроме того, шины адреса и данных в i8086 мультиплексированы. Адрес удерживается на выводах микропроцессора только в течение одного такта машинного цикла, а использоваться должен весь МЦ. Поэтому адрес необходимо запомнить в специальных внешних регистрах-защелках RG (которые, кстати, играют и роль буферной схемы шины адреса).

Наконец, часто требуется преобразовать управляющие сигналы, выдаваемые микропроцессором, в стандартные сигналы системного интерфейса. Так, i8086 формирует выходные сигналы, идентифицирующие тип машинного цикла, и сигналы стробирования: M-IO, OP-IP, R, W. Системная шина использует сигналы записи и чтения памяти — RDM, WRM и записи и чтения внешнего устройства — RDIO, WRIO. Преобразования процессорных сигналов в шинные осуществляет простая логическая схема L.

6.2. Машина пользователя и система команд

Программная модель микропроцессора (рис. 6.4) включает в себя программно-

доступные объекты МПС, т. е. те объекты, состояние которых можно проанализировать и/или изменить при помощи команд микропроцессора. К таким объектам относятся внутренние регистры микропроцессора, ячейки памяти и порты ввода/вывода.

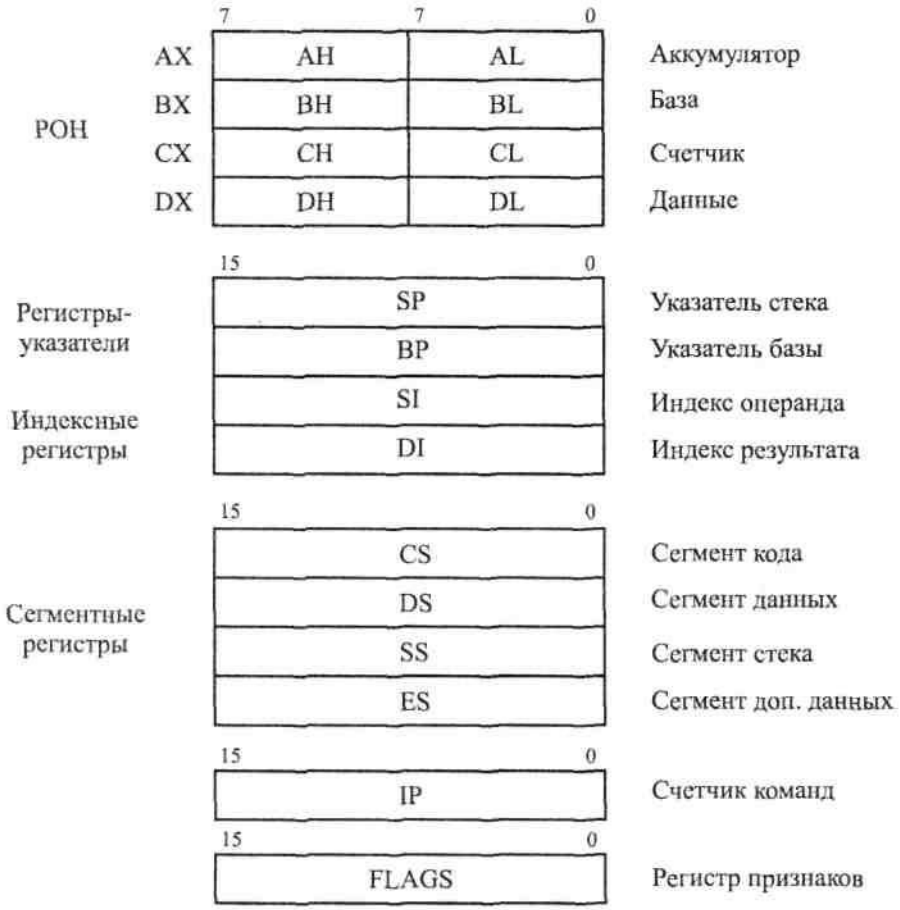


Рис. 6.4. Микропроцессор i8086 — машина пользователя

Рассмотрим *машину пользователя* i8086. Кроме показанных на рис. 6.4 регистров процессора, в машину пользователя i8086 включатся адресное пространство памяти объемом 1 Мбайт и два пространства портов ввода и вывода по 64 Кбайт каждое.

Помимо операций с 16-разрядными регистрами общего назначения (РОН) AX — DX, допускается обращение к каждому байту этих регистров: AL — DL, AH — DH. В процессорах семейства *x*86 система команд построена таким образом, что в некоторых командах РОН выполняют определенные по умолчанию функции счетчиков, индексных регистров, источников адреса и др.

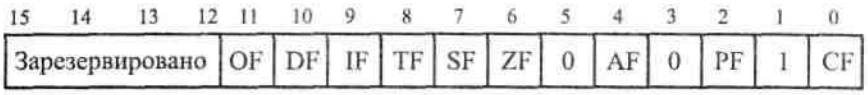


Рис. 6.5. Формат регистра признаков i8086

16-разрядные регистры BP, SI, DI используются для образования исполнительных адресов памяти, SP— указатель стека, IP— программный счетчик (СчК), Flags — регистр флагов, формат которого приведен на рис. 6.5, где:

- CF — перенос/заем из старшего разряда;
- PF — паритет (четность числа единиц в результате);
- AF — дополнительный перенос (из 3-го разряда);
- ZF — нулевой результат;
- SF — отрицательный результат (знак);

- OF — признак арифметического переполнения;
- DF — направление, определяет направление модификации адресов массивов в командах цепочек (увеличение или уменьшение адреса);
- IF — маскирует внешнее прерывание по входу INT (при IF = 1 прерывание разрешено);
- TF — управляет пошаговым режимом работы микропроцессора.

При TF = 1 после выполнения каждой команды автоматически формируется прерывание с вектором 1.

6.2.1. Распределение адресного пространства

Адресное пространство МП определяется в i8086 разрядностью шины адреса/данных и адреса и составляет 2^{20} байтов = 1 Мбайт. В этом адресном пространстве микропроцессору одновременно доступны лишь четыре сегмента, два из которых (DS и ES) предназначены для размещения данных, CS — сегмент кода (для размещения программы) и SS —сегмент стека.

Размеры сегментов определяются разрядностью логических адресов команд, данных и стека. Логические адреса команд и стека (верхушки) хранятся в 16-разрядных регистрах IP и SS соответственно, а логический адрес данных вычисляется в команде одним из многочисленных, предусмотренных системой команд, способов и также составляет 16 битов.

Таким образом, размер каждого сегмента в i8086 составляет 2^{16} байтов = 64 Кбайт. Положение сегмента в адресном пространстве (его начальный адрес) определяется содержимым одноименного сегментного регистра. Формирование физического адреса иллюстрируется на рис. 6.6, из которого видно, что граница сегмента в адресном пространстве может быть установлена не произвольно, а таким образом, чтобы начальный адрес сегмента был кратен 16.

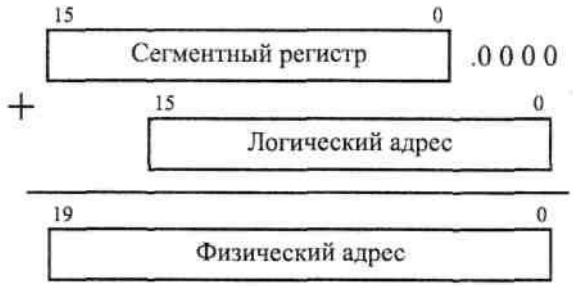


Рис. 6.6. Формирование физического адреса в i8086

По умолчанию сегментные регистры выбираются для образования физического адреса следующим образом: при считывании команды по адресу IP используется CS, при обращении к данным — DS или ES, при обращении к стеку — SS. С помощью специальных приставок к команде (префиксов) можно назначить для использования произвольный сегментный регистр (кроме пары CS:IP, которая не подлежит модификации). Границы сегментов могут быть выбраны таким образом, что сегменты будут изолированы друг от друга, пересекаться или даже полностью совпадать. Например, если загрузить CS = SS = DS = ES = 0, то все сегменты будут совпадать друг с другом и начинаться с нулевого адреса — вариант организации адресного пространства i8080.

6.2.2. Система команд i8086

Система команд i8086 и, вообще, всего семейства *x*86 подробно описана в многочисленных справочниках и руководствах, например [11, 12, 13], поэтому далее мы кратко остановимся только на особенностях системы команд i8086, не вдаваясь в излишние подробности.

i8086 отличается разнообразием форматов команд и способов адресации. Длина команды может составлять от 1 до 6 байтов, причем в первых двух байтах (иногда— в первом) определяется код операций, количество и длина операндов и способ их адресации. В остальных байтах команды могут размещаться непосредственный операнд, прямой адрес или смещение.

Большинство команд i8086 являются *двухадресными,* причем один адрес определяет регистр процессора, а другой — ячейку памяти или регистр.

Операнд в памяти может адресоваться *прямо* или *косвенно* посредством содержимого базовых (BP, BX) или индексных (SI, DI) регистров, а также их суммы. Предусмотрены многочисленные варианты относительной адресации, при которых логический адрес образуется как сумма двух или трех слагаемых— одного или двух регистров процессора и 8- или 16-разрядного смещения, размещаемого в команде.

Режимы адресации спроектированы с учетом эффективной реализации языков высокого уровня. Например, к простой переменной можно обратиться в режиме прямой адресации, а к элементу массива — в режиме косвенной адресации посредством BX, SI. Режим адресации через BP предназначен для доступа к данным из сегмента стека, что удобно при реализации рекурсивных процедур и компиляторов языков высокого уровня.

Система команд насчитывает 113 базовых команд, объединенных в следующие группы:

- □ команды передачи данных:
  - между регистрами и памятью (включая стек), обмен содержимым источника и приемника;
  - ввод, вывод, табличное преобразование;
  - загрузка исполнительного адреса в РОН, загрузка 4-байтового адресного объекта в регистры-указатели (начальный адрес сегмента и смещение в сегменте);
  - передача содержимого регистра F флагов в память, в стек и из стека;
- □ арифметические команды:
  - сложение, вычитание, умножение и деление двоичных чисел со знаком и без знака (произведение и делимое представляются числами двойной длины);
  - десятичная коррекция сложения и вычитания упакованных двоично-десятичных чисел;
  - десятичная коррекция сложения, вычитания, умножения и деления распакованных двоично-десятичных чисел;
- □ логические команды и сдвиги:
  - инверсия, конъюнкция, дизъюнкция, неравнозначность;
  - TEST — поразрядная конъюнкция операндов с установкой флагов, но без занесения результатов;
  - сдвиги на 1 или заданное число разрядов (константа сдвига располагается в CL);
- □ команды передачи управления: переходы, вызовы, возвраты имеют две разновидности — внутрисегментные ("близкие") и межсегментные ("дальние"). При близких передачах загружается только IP, при дальних — IP и CS. Передачи управления могут быть прямыми (целевой адрес — в команде) или косвенными (целевой адрес вычисляется с использованием стандартных режимов адресации). В 16 командах условных переходов проверяются отношения знаковых и беззнаковых чисел. Имеются 4 команды управления циклами, которые рассчитаны на размещение числа повторений цикла в регистре CX;
- □ команды обработки цепочек данных манипулируют последовательностями

байтов или слов в памяти. Время обработки цепочек этими командами гораздо меньше, чем соответствующей программной реализацией.

6.3. Функционирование основных подсистем МПС

Теперь можно рассмотреть функционирование основных подсистем базовой МПС с интерфейсом типа *"общая шина"*. Этот термин используется в двух смыслах: во-первых, как обозначение принципа организации связи процессора с другими устройствами в составе ЭВМ, во-вторых, как обозначение (в русском переводе) конкретного интерфейса Unibus мини-ЭВМ семейства PDP-11 фирмы DEC.

Unibus явился, пожалуй, первым интерфейсом, в котором были полностью реализованы принципы "общей шины":

- все линии интерфейса делятся на три группы: адрес, данные, управление;
- все устройства, в т. ч. процессор, подключаются к линиям интерфейса одинаковым образом;
- идентификация объектов на шине (ячеек памяти, регистров внешних устройств) осуществляется с помощью уникального для каждого объекта *адреса;*
- в каждый момент времени по шине могут взаимодействовать только два устройства, одно из которых является *активным,* а другое — *пассивным.* Активное устройство формирует адрес обмена, управляющие сигналы и может выдавать (в цикле *записи)* или принимать (в цикле *чтения)* данные, которые принимает или выдает пассивное устройство;
- обмен между устройствами может осуществляться в синхронном или асинхронном режиме. При *синхронном обмене* все временные характеристики обмена определяются только активным устройством, которое не анализирует ни готовность пассивного к обмену, ни факт завершения обмена. Синхронный обмен допустим лишь с быстродействующими пассивными устройствами (их быстродействие должно быть не ниже быстродействия активного устройства), которые всегда готовы к обмену (например, регистр двоичной индикации). При *асинхронном обмене* пассивное устройство формирует сигнал готовности к обмену и/или сигнал завершенияобмена, которые анализирует активное устройство.

Интерфейсы, реализующие принципы "общей шины", широко распространились в мини- и микроЭВМ, МПС различного назначения. Многие из них, правда, нарушали некоторые принципы "канонической общей шины", например, за счет появления отдельных адресных пространств регистров процессора и портов ввода и вывода. Однако основные принципы, изложенные выше, сохраняются в многочисленных разновидностях таких интерфейсов.

К достоинствам интерфейсов типа "общая шина" можно отнести его относительную простоту, гибкость системы и возможность ее модификации в широких пределах.

К недостаткам — невозможность распараллеливания процессов обмена (одновременно осуществляется связь только пары устройств). Кроме того, наличие на общей шине устройств с существенно различным быстродействием затрудняет достижение оптимальных характеристик системы.

В *разд. 6.1.3* мы подробно рассмотрели организацию процессорного модуля на базе процессора i8086. Состав внешних выводов процессорного модуля (см. рис. 6.3) позволяет подключить его к интерфейсу, реализованному по принципу общей шины:

- линии адреса AB[19:0];
- линии данных DB[15:0];
- линии управления RDM, WRM, RDIO,WRIO.

Другие линии управления, входящие в состав интерфейса, будут добавлены при

рассмотрении соответствующих подсистем.

6.3.1. Оперативная память

Объем адресного пространства МПС с интерфейсом "общая шина" определяется главным образом разрядностью шины адреса и, кроме того, номенклатурой управляющих сигналов интерфейса. Управляющие сигналы могут определять тип объекта, к которому производится обращение (ОЗУ, ВУ, стек, специализированные ПЗУ и др.). В случае, если МП не выдает сигналов, идентифицирующих тип пассивного устройства (или они не используются в МПС)— для селекции берутся только адресные линии. Число адресуемых объектов составляет в этом случае 2^k , где k — разрядность шины адреса. Будем называть такое адресное пространство *единым.* Иногда говорят, что ВУ в едином адресном пространстве "отображены на память", т. е. адреса ВУ занимают адреса ячеек памяти.

При использовании информации о типе устройства, к которому идет обращение, одни и те же адреса можно назначать для устройств разных типов, осуществляя селекцию с помощью управляющих сигналов.

Так, большинство МП выдают в той или иной форме информацию о типе обращения. В результате в большинстве интерфейсов присутствуют отдельные управляющие линии для обращения к памяти и вводу/выводу, реже — к стеку или специализированному ПЗУ. В результате суммарный объем адресного пространства МПС может превышать величину 2^k.

Например, системная шина МПС на базе микропроцессора i8086 включает 20-разрядную шину адреса и управляющие сигналы, идентифицирующие обращение к памяти (RDM, WRM) и вводу/выводу (RDIO, WRIO). Поэтому в системе доступны 1 Мбайт ячеек памяти (адреса 00000 — FFFFF) + 64 Кбайт адресов ввода + 64 Кбайт адресов вывода (0000 — FFFF). Последняя величина определяется тем, что в командах ввода/вывода процессоров семейства *x*86 адрес внешнего устройства имеет разрядность 16 битов.

Диспетчер памяти

При необходимости расширить объем памяти за пределы адресного пространства можно воспользоваться т. н. *диспетчером памяти.* В простейшем случае он представляет собой программно-доступный регистр, который должен располагаться в пространстве ввода/вывода. В него заносится номер активного в данный момент банка памяти, причем объем банка может равняться объему адресного пространства МП.

Очевидно, в каждый момент времени процессору доступен только один банк. При необходимости перехода в другой банк памяти МП должен предварительно выполнить программную процедуру (часто всего одну команду) перезагрузки содержимого регистра номера банка.

К развитию этой идеи можно отнести механизм сегментации памяти в 16- и 32-разрядных МП фирмы Intel.

6.3.2. Ввод/вывод

Подсистема ввода/вывода (ПВВ) обеспечивает связь МП с внешними устройствами, к которым будем относить:

- □ устройства ввода/вывода (УВВ): клавиатура, дисплей, принтер, датчики и исполнительные механизмы, АЦП, ЦАП, таймеры и т. п.;
- □ внешние запоминающие устройства (ВЗУ): накопители на магнитных дисках, "электронные диски", CD и др.

В рамках рассмотрения ПВВ будем полагать термины "УВВ" и "ВУ" синонимами, т. к. обращение к ним со стороны процессора осуществляется по одним законам.

ПВВ в общем случае должна обеспечивать выполнение следующих функций:

- □ согласование форматов данных, поскольку процессор всегда выдает/принимает данные в параллельной форме, а некоторые ВУ — в последовательной. С этой точки зрения различают устройства параллельного и последовательного

обмена. В рамках параллельного обмена не производится преобразование форматов передаваемых слов, в то время как при последовательном обмене осуществляется преобразование параллельного кода в последовательный и наоборот. Все варианты, при которых длина слова ВУ (больше 1 бита) не совпадает с длиной слова МП, сводятся к разновидностям параллельного обмена;

- организация режима обмена — формирование и прием управляющих сигналов, идентифицирующих наличие информации на различных шинах, ее тип, состояние ВУ (Готово, Занято, Авария), регламентирующих временные параметры обмена. По способу связи процессора и ВУ (активного и пассивного) различают синхронный и асинхронный обмены, различия между которыми мы обсудили в начале настоящей главы.

Параллельный обмен

Простейшая подсистема параллельного обмена должна обеспечить лишь дешифрацию адреса ВУ и электрическое подключение данных ВУ к системной шине данных DB по соответствующим управляющим сигналам. На рис. 6.7 показаны устройства параллельного ввода и вывода информации в составе МПС на базе буферных регистров К580ИР82.

Очевидно, при обращении процессора (он в подобных циклах играет роль активного устройства) к *устройству ввода,* адрес соответствующего регистра помещается процессором на шину адреса и формируется управляющий сигнал RDIO. Дешифратор адреса, включающий и линию RDIO, при совпадении адреса и управляющего сигнала активизирует выходные линии регистра и его содержимое поступает по шине данных в процессор.

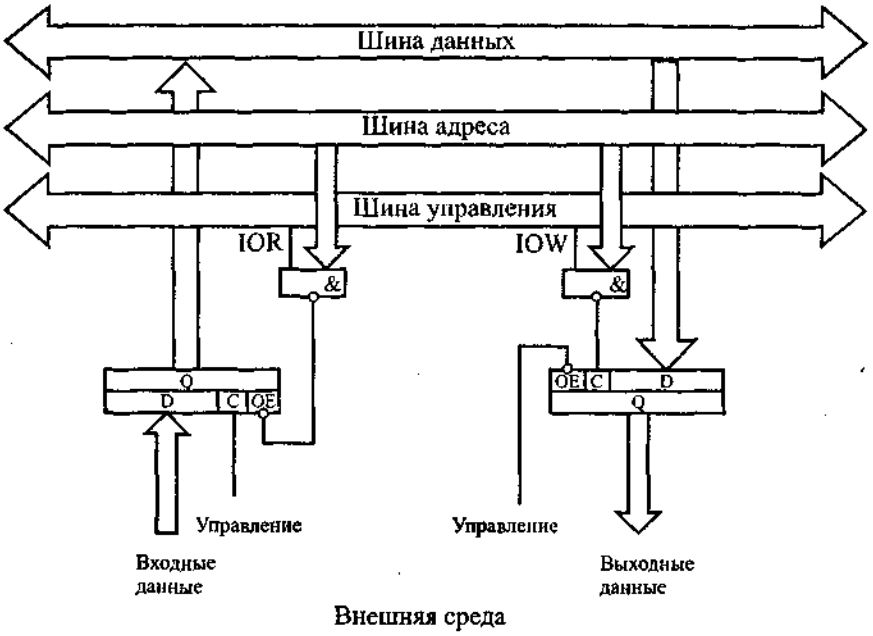


Рис. 6.7. Параллельный обмен па базе буферных регистров

Аналогично идет обращение к *устройству вывода.* Совпадение адреса устройства на шине адреса с активным уровнем сигнала WRIO обеспечивает "защелкивание" состояния шины данных в регистре вывода.

Характерно, что при таком способе обмена процессор не анализирует готовность ВУ к обмену, а длительность существования адреса, данных и управляющего сигнала целиком определяется тактовой системой процессора и принятым алгоритмом командного цикла. Напомним, что такой способ обмена называется *синхронным.* Синхронный обмен реализуется наиболее просто, но он возможен только с устройствами, всегда готовыми к

обмену, либо процессор должен перед выполнением команды ввода/вывода программными средствами убедиться в готовности ВУ к обмену (обычно в этом случае предварительно анализируется состояние флага готовности, формируемого ВУ). Кроме того, быстродействие ВУ, взаимодействующее с процессором в синхронном режиме, должно гарантировать прием/выдачу данных за фиксированное время, выделенное процессором на цикл обмена.

Во многих микропроцессорных комплектах выпускают специальные интерфейсные БИС, существенно расширяющие (по сравнению с использование регистров) возможности разработчиков при организации параллельного обмена в МПС. Такие БИС обычно имеют несколько каналов передачи информации, позволяют программировать направление передачи (ввод или вывод) по каждому каналу и выбирать способ обмена — синхронный или асинхронный.

Типичным примером такой БИС может служить программируемый контроллер параллельного обмена (далее "контроллер") 8255А (отечественный аналог — К580ВВ55).

Контроллер параллельного обмена К580ВВ55 [13] представляет собой трехканальный байтовый интерфейс и позволяет организовать обмен байтами с периферийным оборудованием в различных режимах. Он включает в себя три 8-разрядные канала ввода/вывода A, B и C, буфер шины данных, 8-разрядный регистр управления Y и блок управления.

Подключение контроллера к системной шине показано на рис. 6.8. Канал адресуются двумя линиями адреса A1, A0. В МПС контроллер размещают как правило, в пространстве адресов ввода/вывода. Поэтому в качестве стробов чтения и записи используются сигналы RDIO, WRIO, для селекции контроллера по CS дешифрируются старшие разряды адреса, а для выбора адресуемого объекта внутри контроллера — два младших.

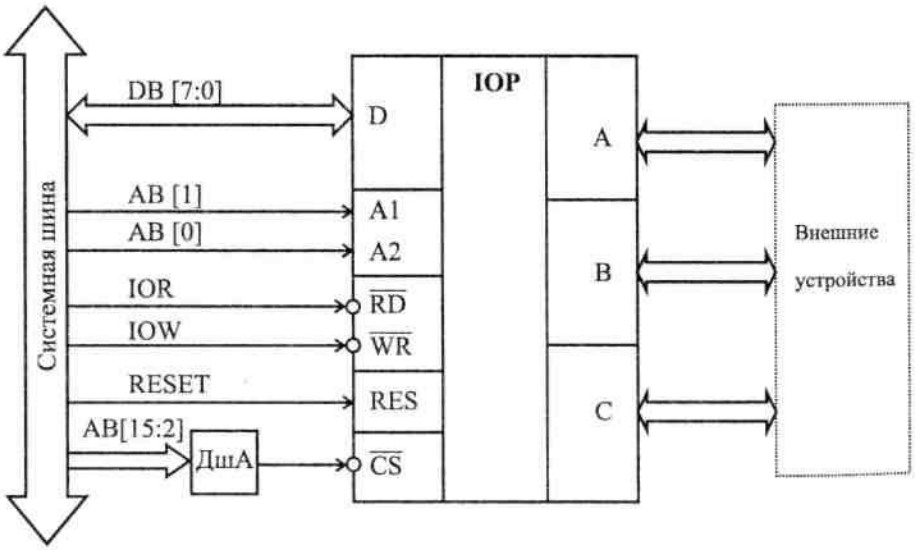


Рис. 6.8. Подключение контроллера 8255 к системной шине

Каналы контроллера программируются для работы в одном из трех режимов:

- режим "0" — синхронный однонаправленный ввод/вывод;
- режим "1" — асинхронный однонаправленный ввод/вывод;
- режим "2" — асинхронный двунаправленный ввод/вывод.

Режим работы контроллера устанавливается кодом управляющего слова, которое предварительно записывается в регистр управления Y.

В *режиме "0"* контроллер может работать как четыре порта ввода/вывода: A[7:0]. B[7:0], C[7:4], C[3:0], причем каждый порт может быть независимо запрограммирован на ввод или на вывод. При этом к порту, определенному как выходной, нельзя обращаться по чтению, а на входной порт нельзя выводить информацию.

В асинхронном однонаправленном *режиме "1"* могут работать только каналы A и

В, причем соответствующие линии канала С придаются каналам А и В для передачи управляющих сигналов. Как и в режиме "0", каналы А и В программируются на ввод или вывод (независимо).

В *режиме "2"* может работать только канал А, к которому в этом случае можно обращаться как по записи, так и по чтению (двунаправленный асинхронный обмен). При этом канал В может быть запрограммирован как на работу в режиме "1", так и в режиме "0".

Выбор режимов каналов и направления передачи данных в них осуществляется загрузкой во внутренний управляющий регистр Y соответствующего кода.

Линии канала С могут работать только в режиме "0", причем независимо можно запрограммировать направление передачи старшей и младшей тетрады канала С. Если для каналов А и/или В выбраны режимы "2" и/или "1", то соответствующие линии канала С перестают работать в режиме "0" и используются для передачи управляющих сигналов. Линии канала С, которые не используются при выбранной комбинации режимов каналов А и В, можно использовать как линии ввода или вывода канала С, работающего в "0"-режиме. Кроме того, всегда имеется возможность программного сброса/установки произвольного разряда канала С.

Режим "0" является синхронным и во многом напоминает рассмотренный выше механизм обмена с использованием регистров. Рассмотрим подробнее процесс асинхронного обмена в режиме "1".

Режим "1" обеспечивает однонаправленную асинхронную передачу информации между процессором и ВУ. При этом каналы А и В используются как регистры данных, а канал С — для приема и формирования управляющих сигналов, сопровождающих асинхронный обмен, причем каждый разряд канала С имеет строго определенное функциональное назначение [13].

Например, если канал запрограммирован на ввод в режиме "1", то процессор может вводить данные этого канала только "будучи уверенным" в их готовности. Об этой готовности ему должен сообщить контроллер путем установки специального признака — флага в определенном разряде регистра С и может быть, формированием запроса на прерывание с соответствующим вектором (о прерываниях подробнее *см. в разд. 6.3.3). С* другой стороны, внешнее устройство, подключенное к каналу, не должно выдавать новую порции информации, пока прежняя не будет прочитана процессором.

Для обеспечения синхронизации *ввода* в режиме "1" каналу придаются три линии канала С для передачи управляющих сигналов:

- STB (строб записи)— сигнал, формируемый ВУ для записи очередного байта данных в регистр канала:
- IBF (подтверждение приема)— сигнал, формируемый контроллером для ВУ в тот момент, когда процессор прочитал содержимое регистра каналг Пока сигнал IBF неактивен, ВУ запрещается вырабатывать новый строб записи;
- INT (запрос прерывания) — вырабатывается контроллером для процессор после того, как очередной байт данных запишется в регистр канала. Это же событие устанавливает флаг готовности канала в разряде регистра С.

Обмен начинается с подачи ВУ сигнала STB, по которому данные помещаются в регистр канала. Контроллер, во-первых, сбрасывает сигнал IBF, запрещая ВУ выработку нового строба, и, во-вторых, устанавливает флаг готовности и (может быть) формирует сигнал запроса на прерывание INT процессору. Процессор может достаточно долго не реагировать на сообщение готовности канала, занятый более приоритетными процедурами. Все это время установлены готовность и INT и сброшен IBF, новая порция информаци не может поступить в канал.

Когда процессор обратится по адресу канала и введет хранящуюся в регистре информацию, контроллер сбрасывает флаг готовности и запрос на прерывание INT и устанавливает сигнал IBF, разрешая ВУ записывать следующи байт в регистр канала. Однако ВУ может быть достаточно инерционным и довольно долго подготавливает следующую порцию информации, но пока ВУ не сформирует новый сигнал STB,

контроллер не выработает сигнал готовности и, следовательно, процессор не будет обращаться по адресу канала.

Подобный режим обмена позволяет исключить как потерю информации контроллере, так и повторный ввод в процессор прежней информации.

Аналогично реализуется и асинхронный режим вывода. Каналу, запрограгммированному на вывод в режиме "1", придаются три линии управления канала С:

- OBF (выходной буфер заполнен) — сигнал формируется контроллером для ВУ после того, как процессор записал в регистр канала новую порцию информации;
- АСК (подтверждение записи)— сигнал от ВУ контроллеру, подтверждающий прием очередного байта;
- INT (запрос прерывания) — запрос прерывания от контроллера процессору для выдачи процессором в канал следующего байта информации.

Процедуры ввода и вывода в режиме "2" осуществляются аналогично соответствующим процедурам в режиме "1".

Последовательный обмен

При организации последовательного обмена ключевыми могут считаться две проблемы:

- синхронизация битов передатчика и приемника;
- фиксация начала сеанса передачи.

Различают два способа передачи последовательного кода: *синхронный* и *асинхронный.*

При синхронном методе передатчик генерирует две последовательности — информационную TxD и синхроимпульсы CLK, которые передаются на приемник по разным линиям. Синхроимпульсы обеспечивают синхронизацию передаваемых битов, а начало передачи отмечается по-разному. При организации *внешней синхронизации* (рис. 6.9) сигнал начала передачи BD генерируется передатчиком и передается на приемник по специальной линии.

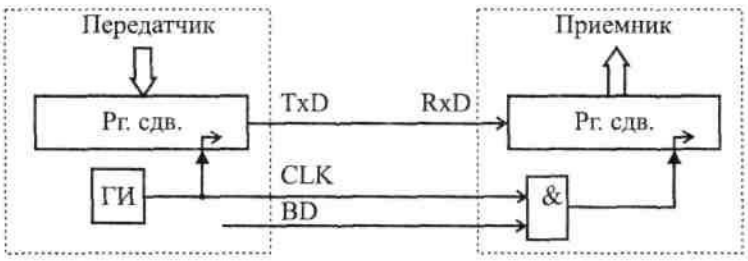


Рис. 6.9. Последовательный синхронный обмен

Системы с *внутренней синхронизацией* генерируют на линию данных специальные коды длиной 1—2 байта — символы синхронизации. Для каждого приемника предварительно определяются конкретные синхросимволы, таким образом можно осуществлять адресацию конкретного абонента из нескольких, работающих на одной линии. Каждый приемник постоянно принимает биты с линии RxD, формирует символы и сравнивает с собственными синхросимволами. При совпадении принятых символов с заданными для этого приемника синхросимволами последующие биты поступают в канал данных приемника. В случае реализации внутренней синхронизации между приемником и передатчиком "прокладывают" только две линии — данных и синхроимпульсов.

Наконец, при *асинхронном* способе обмена можно ограничиться одной линией — данных. Для надежной синхронизации обмена в асинхронном режиме:

- передатчик и приемник настраивают на работу с одинаковой частотой;

- передатчик формирует стартовый и стоповый биты, отмечающие начало и конец посылки;
- передача ведется короткими посылками (5—9 битов), а частоты передачи выбираются сравнительно низкими.

Принцип последовательного асинхронного обмена по единственной линии показан на рис. 6.10. Пока передачи нет, на линии передатчик удерживает высокий уровень (H). Передача начинается с выдачи в линию стартового бита низкого уровня (длительности всех битов τ одинаковы и определяются частотой передатчика $f_T = 1/\tau$).

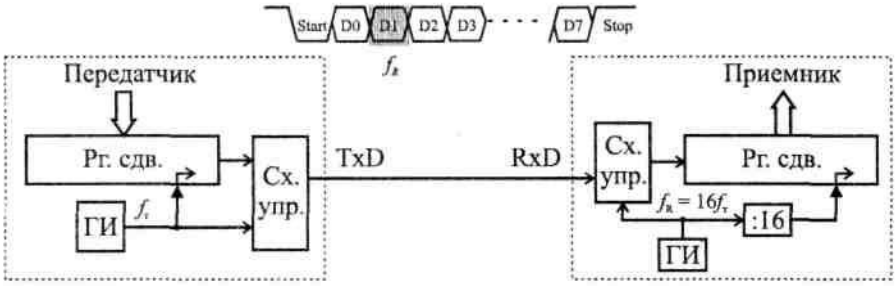


Рис. 6.10. Последовательный асинхронный обмен

Частота приемника f_R устанавливается равной $16 \times f_T$. Когда приемник обнаруживает на линии перепад H → L, он включает счетчик тактов до 16, причем еще дважды за период τ проверяет состояние линии. Если низкий уровень (L) подтверждается, приемник считает, что принял старт-бит, и включает счетчик принимаемых битов. Если во второй и третьей проверке на линии определяется H-уровень, то перепад считается помехой и старт-бит не фиксируется.

Каждый последующий (информационный) бит принимается таким образом, что за период τ трижды проверяется состояние линии (например, в 3, 8 и 11 тактах приемника) и значение принимаемого бита определяется по мажоритарному принципу. Принятый бит помещается слева в сдвиговый регистр приемника. После принятия последнего информационного бита (количество битов в посылке определяется протоколом обмена и составляет обычно от 5 до 9) обязательно должен последовать столовый бит H-уровня. Во время поступления стоп-бита содержимое сдвигового регистра приемника передается в память, а в регистр передатчика может загружаться новая порция информации для передачи. Отсутствие стопового бита воспринимается приемником как ошибка передачи посылки.

После стопового бита можно формировать стартовый бит новой посылки или "держать паузу" произвольной длительности, при которой на линии присутствует H-уровень.

Наличие стартового бита позволяет в начале каждой посылки синхронизировать фазы приемника и передатчика, компенсировав неизбежный уход фаз передатчика и приемника. Короткие посылки и относительно низкая частота передачи позволяют надеяться, что неизбежное рассогласование частот передатчика и приемника не приведет к ошибкам при передаче посылки.

6.3.3. Прерывания

Подсистема прерываний— совокупность аппаратных и программных средств, обеспечивающих реакцию программы на события, происходящие вне программы. Такие события возникают, как правило, случайно и асинхронно по отношению к программе и требуют прекращения (чаще временного) выполнения текущей программы и переход на выполнение другой программы (подпрограммы), соответствующей возникшему событию.

Различают внутренние и внешние (по отношению к процессору) события, требующие реакции подсистемы прерываний. К внутренним событиям относятся переполнение разрядной сетки при выполнении арифметических операций, попытка деления на 0, извлечение корня четной степени из отрицательного числа, появление

несуществующего кода команды, обращение программы в область памяти, для нее не предназначенную, сбой при выполнении передачи данных или операции в АЛУ и многое другое. Внутренние прерывания должны обеспечиваться развитой системой аппаратного контроля процессора, поэтому они не получили широкого распространения в простых 8- и 16-разрядных МП.

Внешние прерывания могут возникать во внешней по отношению к процессору среде и отмечать как аварийные ситуации (кончилась бумага на принтере, температура в реакторе превысила допустимый уровень, исполнительный орган робота дошел до предельного положения и т. п.), так и нормальные рабочие события, которые происходят в случайные моменты времени (нажата клавиша, исчерпан буфер принтера или ВЗУ и т. п.). Во всех этих случаях требуется прервать выполнение текущей программы и перейти на выполнение другой программы (подпрограммы), обслуживающей данное событие.

С точки зрения реализации внутренние и внешние прерывания функционируют одинаковым образом, хотя при работе подсистемы с внешними прерываниями возникают дополнительные проблемы идентификации источника прерывания. Поэтому ниже остановимся на рассмотрении внешних прерываний.

Анализ состояния внешней среды можно осуществлять путем программного сканирования — считывания через определенные промежутки времени слов состояния всех возможных источников прерываний, выделения признаков отслеживаемых событий и переход (при необходимости) на прерывающую подпрограмму (часто ее называют *обработчиком прерывания).*

Однако такой способ не обеспечивает для большинства применений приемлемого времени реакции системы на события, особенно при необходимости отслеживания большого числа событий. К тому же при коротком цикле сканирования большой процент процессорного времени тратится на проверку (чаще безрезультатную) состояния внешней среды.

Гораздо эффективней организовать взаимодействие с внешней средой таким образом, чтобы всякое изменение состояния среды, требующее реакции МПС, вызывало появление на специальном входе МП сигнала прерывания текущей программы. Организация прерываний должна быть обеспечена определенными аппаратными и программными средствами, которые мы и называем *подсистемой прерываний.*

Подсистема прерываний должна обеспечивать выполнение следующих функций:

- □ обнаружение изменения состояния внешней среды (запрос на прерывание);
- □ идентификация источника прерывания;
- □ разрешение конфликтной ситуации в случае одновременного возникновения нескольких запросов (приоритет запросов);
- □ определение возможности прерывания текущей программы (приоритет программ);
- □ фиксация состояния прерываемой (текущей) программы;
- □ переход к программе, соответствующей обслуживаемому прерыванию;
- □ возврат к прерванной программе после окончания работы прерывающей программы.

Рассмотрим варианты реализации в МПС перечисленных выше функций.

Обнаружение изменения состояния внешней среды

Фиксация изменения состояния внешней среды может осуществляться различными схемами: двоичными датчиками, компараторами, схемами формирования состояний и др. Будем полагать, что все эти схемы формируют в конечном итоге *логические сигналы* запроса на прерывание *z,* причем для определенности будем считать, что активное состояние этого сигнала передается *уровнем логической единицы* (Н-уровень).

Количество источников запросов в МПС может быть различно, в т. ч. и довольно велико. Дефицит внешних выводов МП в общем случае исключает возможность передачи каждого запроса от ВУ по "собственной" линии интерфейса. Обычно на одну линию запроса подключается несколько источников прерываний (по функции ИЛИ), а иногда и

все источники запросов — на единственный вход.

Управляющий автомат процессора должен периодически анализировать состояние линии (линий) запросов на прерывания. Каким образом выбирается период проверки? С одной стороны, этот период должен быть коротким, чтобы обеспечить быструю реакцию системы на события. С другой стороны, при переходе на обслуживание прерывания требуется сохранить текущее состояние процессора на момент прерывания, с тем, чтобы, завершив программу-обработчик, продолжить выполнение прерванной программы "с того же места", на котором произошло прерывание.

Напомним, что в основе работы процессора лежит *командный цикл (см. разд. 2.1),* состоящий, в свою очередь, из *машинных циклов,* каждый из которых длится несколько *тактов.* Осуществлять прерывание в произвольном такте невозможно, т. к. при этом пришлось бы сохранять в качестве контекста прерванной программы состояние всех элементов памяти процессора.

Прерывание по завершению текущего машинного цикла требует сохранение текущего состояния незавершенной команды. Например, при возникновении прерывания в том месте командного цикла, когда из памяти выбраны код команды и первый операнд, а для второго операнда только сформирован исполнительный адрес, следует сохранить: содержимое программного счетчика PC, код команды, первый операнд и адрес второго операнда. Сохранение этой информации требует как дополнительных аппаратных, так и временных затрат. Очевидно, при возврате к прерванной программе проще начать выполнение текущей команды заново. В этом случае сохранять при прерывании достаточно лишь значение PC. Поэтому в большинстве случаев процессоры анализируют состояние линий запросов *в конце каждого командного цикла.*

Идентификация источника прерывания

Различают два типа входов запросов на прерывания — *радиальные* и *векторные.* Получив запрос на прерывание, процессор должен идентифицировать его источник, т. е. в конечном счете определить начальный адрес обслуживающей это прерывание программы. Способ идентификации зависит от типа входа, на который поступил запрос.

Каждый радиальный вход связан с определенным адресом памяти, по которому размещается указатель на обслуживающую программу или сама программа. Если радиальный вход связан с несколькими источниками запросов, то необходимо осуществить программную идентификацию путем последовательного (в порядке убывания приоритетов) опроса всех связанных с этим входом источников прерывания. Этот способ не требует дополнительных аппаратных затрат и одновременно решает проблему приоритета запросов, однако время реакции системы на запрос может оказаться недопустимо большим, особенно при большом числе источников прерываний.

Гораздо чаще в современных МПС используется т. н. *векторная подсистема прерываний.* В такой системе микропроцессор, получив запрос на векторном входе 1NT, выдает на свою выходную линию сигнал подтверждения прерывания INTA, поступающий на все возможные источники прерывания. Источник, не выставивший запроса, никак не реагирует на сигнал INTA. Источник, выставивший запрос, получая сигнал INTA, выдает на системную шину данных "вектор прерывания" — свой номер или адрес обслуживающей программы или, чаще, адрес памяти, по которому расположен указатель на обслуживающую программу. Время реакции МПС на запрос векторного прерывания минимально (1—3 машинных цикла) и не зависит от числа источников.

Приоритет запросов

Для исключения конфликтов при одновременном возникновении нескольких запросов на векторном входе ответный сигнал INTA подается на источники запросов не параллельно, а последовательно — в порядке убывания приоритетов запросов. Источник, не выставлявший запроса, транслирует сигнал INTA со своего входа на выход, а источник, выставивший запрос, блокирует дальнейшее распространение сигнала INTA. Таким образом, только один источник, выставивший запрос, получит от процессора сигнал INTA и выдаст по нему свой вектор на шину данных.

Более гибко решается проблема организации приоритетов запросов при ис-

пользовании в МПС специальных *контроллеров прерываний.*

Конфликты на радиальном входе исключаются самим порядком программного опроса источников.

Приоритет программ

Прерывание в общем случае может возникать не только при решении "фоновой" задачи, но и в момент работы другой прерывающей программы, причем не всякую прерывающую программу допустимо прерывать любым запросом. В фоновой задаче также могут встречаться участки, при работе которых прерывания (все или некоторые) недопустимы. В общем случае в каждый момент времени работы процессора должно быть выделено подмножество запросов, которым разрешено прерывать текущую программу.

В МПС эта задача решается на нескольких уровнях. В процессоре обычно предусматривается программно-доступный флаг разрешения/запрещения прерывания, значение которого определяет возможность или невозможность всех прерываний. Для создания более гибкой системы приоритетов программ на каждом источнике прерываний может быть предусмотрен специальный программно-доступный триггер разрешения формирования запроса. В таком случае возможно формирование произвольного подмножества разрешенных в данный момент источников прерываний.

При использовании контроллера внешних прерываний, в нем обычно предусматривают специальный программно-доступный регистр, разряды которого *маскируют* соответствующие линии запросов на прерывание, запрещая контроллеру вырабатывать сигнал прерывания процессору, если запросы от ВУ поступают по замаскированным линиям. Однако замаскированные запросы сохраняются в контроллере и в дальнейшем, при изменении состояния регистра маски, могут быть переданы на обслуживание.

Обработка прерывания

К обработке прерывания отнесем фиксацию состояния прерываемой программы, переход к программе, соответствующей обслуживаемому прерыванию, и возврат к прерванной программе после окончания работы прерывающей программы.

Выше мы определили, что большинство процессоров может прервать выполнение текущей программы и переключиться на реализацию обработчика прерывания только после завершения очередной команды. При этом в качестве контекста прерванной программы необходимо сохранить текущее состояние счетчика команд PC, а в PC загрузить новое значение — адрес программы-обработчика прерывания. Очевидно, адрес возврата в прерванную программу (содержимое PC на момент прерывания) следует размещать в стеке, что позволит при необходимости осуществлять вложенные прерывания (когда в процессе обслуживания одного прерывания получен запрос на обслуживание другого).

Можно вспомнить, что подобный механизм реализован в системах команд многих процессоров для выполнения команд вызовов подпрограммы (CALL. JSR). В ЭТИХ командах адрес вызываемой подпрограммы содержится в коде команды.

В случае вызова обработчика прерывания его адрес необходимо связать либо со входом, на который поступил запрос (радиальные прерывания), либо с номером источника прерываний, сформировавшего запрос (векторные прерывания). В первом случае не требуется никаких внешних процедур для идентификации источника, сразу можно запускать связанный со входом обработчик. Понятно, здесь идет речь об отсутствии необходимости в аппаратных процедурах идентификации источника запроса. Если на радиальный вход "работают" несколько источников, то выбор осуществляется программными способами.

В случае векторных прерываний адрес перехода связывают с информацией, поступающей от источника запроса по шине данных в машинном цикле обслуживания прерывания — *вектором прерывания.*

Напомним, что любой командный цикл процессора начинается с чтения команды из памяти. В первом машинном цикле командного цикла процессор выдает на шину адреса

содержимое PC, формирует управляющий сигнал RDM и помещенное памятью на шину данных слово интерпретирует как команду (или ее начальную часть, если длина команды превышает длину машинного слова).

Если в конце очередного командного цикла процессор обнаруживает (незамаскированный) запрос на векторном входе, он начинает следующий командный цикл с небольшими изменениями: содержимое PC по-прежнему выдается на шину адреса (чтобы не нарушать общности цикла), но вместо сигнала RDM формирует сигнал INTA. Источник запроса (чаще — контроллер прерываний) в ответ на сигнал INTA формирует на шину данных код команды вызова подпрограммы, в адресной части которой размещается адрес обработчика соответствующего прерывания.

Такой простой способ реализации векторных прерываний, с использованием уже существующего механизма вызова подпрограмм, был реализован, например, в микропроцессоре i8080 с контроллером прерываний i8259. Однако этот механизм, как, впрочем, и все остальное, допускает дальнейшее совершенствование.

Прежде всего, желание иметь возможность располагать подпрограммы в произвольной области памяти приводит к необходимости размещать в поле адреса команды вызова полноразрядный адрес (16— 20— 32 бита). В этом случае длина команды превышает длину машинного слова и ее ввод требует нескольких машинных циклов (например, в i8080 — трех), что увеличивает время реакции системы на запрос прерывания.

Для преодоления этого недостатка в систему команд процессора включают дополнительно "укороченные" команды вызова длиной в одно машинное слово. Эти команды в процессорах 8080 и *x*86 имеют мнемокод INT. В микропроцессоре i8080 имеется 8 таких команд длиной в 1 байт, адресующих подпрограммы по фиксированным адресам памяти: 0000h, 0008h, 0010h, ..., 0038h.

В процессорах *x*86 имеется 256 вариантов двухбайтовых команд INT 00h, ..., INT FFh, байт поля адреса которых (называемый *вектором)* после умножения на 4 указывает на четырехбайтовую структуру, определяющую произвольный адрес в адресном пространстве памяти.

Напомним, что доступ в память процессоров *x*86 (в *реальном режиме)* осуществляется только в рамках сегментов размером в 64 Кбайт. Положение начала сегмента в адресном пространстве памяти определяется содержимым 16-разрядного сегментного регистра, а положение адресуемого байта внутри сегмента— 16-разрядным смещением. Среди команд передачи управления различают *короткие* и *длинные переходы* (вызовы). При коротком вызове подпрограмма должна располагаться в текущем сегменте кода, и ее вызов сопровождается только изменением счетчика команд (в *x*86 он обозначается. как IP). При длинном вызове новое значение загружается как в IP, так и в сегментный регистр кода CS. Таким образом, для осуществления длинного вызова (перехода) в адресном поле команды необходимо разместить 4 байта.

Механизм векторных прерываний в процессорах *x*86 в реальном режиме реализован следующим образом. В начальных адресах 00000h, ..., 003FFh пространства памяти размещается таблица векторов прерываний объемом 1 Кбайт, включающая 256 строк таблицы — четырехбайтовых структур CS:IP, которые определяют адреса соответствующих обработчиков прерываний. В цикле обработки векторного прерывания (запрос по входу INT), процессор получает от источника байт — номер строки таблицы векторов прерываний, из которой и загружаются новые значения CS и IP. Старые значения CS:IP (адрес возврата) размещаются в стеке.

Запросу по радиальному входу NMI соответствует вектор 2, поэтому появление активного значения не вызывает машинного цикла обслуживания прерывания, а сразу вызывается обработчик по адресу из ячеек памяти 00008h, ..., 0000Bh. Кстати, любой обработчик прерывания (независимо от значения маскирующих флагов) можно вызвать программно с помощью команды INT *nn*, где *nn* — номер строки таблицы векторов прерываний.

Таким образом, команда INT отличается от команды CALL, во-первых, способом адресации вызываемой подпрограммы (прямой адрес — в команде CALL, косвенный— в

INT), во-вторых, при реализации INT в стек, помимо CS и IP, помещается содержимое регистра признаков процессора — FLAGS. Соответственно, завершаться подпрограмма, вызываемая командой INT, должна командой IRET ("возврат из прерывания"). Действие IRET отличается от действия RET извлечением из стека дополнительного слова в регистр FLAGS.

6.3.4. Прямой доступ в память

В процессе работы МПС с интерфейсом типа "общая шина" часто возникает необходимость передачи достаточно больших массивов данных между памятью и ВУ (например, копирование сектора диска, загрузка видеопамяти и т. п.). При наличии в системе единственного активного устройства — процессора возможен единственный путь решения этой задачи — программно-управляемый обмен "Память → Процессор → ВУ" (или "ВУ → Процессор → Память").

Рассмотрим вариант программно-управляемого обмена между памятью и внешним устройством в МПС на базе МП i8080 [13]. Пусть необходимо передать массив данных длиной *L*, начиная с адреса ADR на ВУ с адресом AIO. Положим, что начальный адрес массива загружен в регистровую пару HL, а длина массива— в регистр С. Тогда фрагмент программы обмена может иметь вид, представленный в табл. 6.2.

Таблица 6.2. Фрагмент программы обмена

Мнемокод	Комментарий	Количество МЦ
LM: MOV A,M	Чтение байта в Акк	2
OUT AIO	Выдача байта на ВУ	3
INX H	Модификация адреса	1
DCRC	Модификация счетчика	1
JNZ LM	Переход, если массив не исчерпан	3
	Всего машинных циклов:	10

Таким образом, для того чтобы в рамках процедуры копирования массива данных переслать из памяти в ВУ один байт данных, потребуется десять машинных циклов. Процессоры с более совершенной системой команд (например, *x*86) могут использовать для этой цели меньшее число МЦ, но все равно их будет более одного.

Управляя обменом, микропроцессор "ведет" два счетчика — адресов массива и количества переданных байтов и формирует на магистраль сигналы управления. Если снабдить ВУ аппаратными счетчиками и схемой формирования управляющих сигналов (т. н. "канал прямого доступа в память" — ПДП), то передачу одного байта (слова) можно осуществить за один МЦ без участия процессора. Необходимо лишь на время передачи данных под управлением канала ПДП блокировать работу процессора, отключив его от системной шины. Для этого служит вход захвата шины HLD. Если подать на него активный уровень, то МП по окончании текущего МЦ, безусловно, перейдет в режим ожидания, переведя все свои выходные линии, кроме HLDA, в высокоимпе-дансное состояние, а выход HLDA — в состояние логической 1. Выходной сигнал HLDA используется для отключения процессорного модуля от системной шины — перевода шинных формирователей, включенных между локальной и системной шиной, в высокоимпедансное состояние.

Если в МПС используется несколько ВУ, снабженных каналом ПДП, то целесообразно использовать специальный контроллер ПДП, который обеспечивает программирование каналов ПДП, подключение их к системной шине и дисциплину обслуживания.

ГЛАВА 7

Эволюция архитектур микропроцессоров и микроЭВМ

В *главе 6* была рассмотрена архитектура 16-разрядного микропроцессора i8086 и систем на его основе. Эту архитектуру мы (условно) будем считать базовой. Уже в ней по сравнению с первыми 8-разрядными системами (на базе i8080) реализован ряд новых архитектурных решений:

- расширена система команд (по набору операций и способам адресации);
- архитектура микропроцессора ориентирована на мультипроцессорную работу. Разработана группа вспомогательных БИС (контроллеров и специализированных процессоров) для организации мультимикропроцессорных систем различной конфигурации;
- начато движение в сторону совмещения во времени выполнения различных операций. Микропроцессор включает два параллельно работающих устройства: обработки данных и связи с магистралью, что позволяет совместить во времени процессы обработки информации и передачи ее по магистрали;
- введена новая (по сравнению с i8080) организация памяти, которая далее использовалась во всех старших моделях семейства Intel — *сегментация памяти.*

Можно сказать, что основная цель совершенствования микропроцессоров — это увеличение их производительности. Достигается эта цель различными путями: повышением тактовой частоты работы кристалла, совершенствованием операционных устройств (например, применение параллельного умножителя), организацией параллельной во времени работы нескольких устройств, совершенствованием системы команд (с ориентацией под конкретный класс задач), эффективной организацией иерархии памяти, опережающим выполнением ряда процедур командных циклов, организацией мультизадачных и мультипроцессорных систем и другими способами.

На примере микропроцессоров Intel линии 8080 → 8086 → 80286 → 80386 → 80486 → Pentium → ... → Pentium 4 (это семейство принято обозначать как *x*86) можно проследить реализацию многих из перечисленных выше путей. Рассмотрим некоторые из них, не придерживаясь хронологической последовательности нововведений. Более подробные сведения о рассматриваемых в этой главе вопросах можно найти в [3, 11, 12, 14].

7.1. Защищенный режим и организация памяти

Первый шаг для увеличения производительности систем на базе процессоров *x*86 был сделан в направлении мультипроцессорной конфигурации. В 8086 предусмотрены два режима работы — *минимальный* и *максимальный (см. разд. 6.1.1),* причем последний ориентирован на организацию мультипроцессорных систем. Часть выводов микропроцессора в максимальном режиме вместо сигналов управления шиной передает коды внутренних состояний управляющего автомата; кроме того, в составе серии выпускались специализированные модули, которые обеспечивали доступ микропроцессора к системной шине — арбитраж шины.

Однако широкого распространения подобная архитектура не получила, поскольку при отсутствии на кристалле микропроцессора достаточно "вместительной" внутренней памяти процессоры постоянно ожидают в очереди на доступ к шине. Характерно, что в последующих моделях семейства — 80286, 80386, 80486 поддерживалась только однопроцессорная конфигурация, и лишь в Pentium вновь вернулись к возможности

организации многопроцессорных систем.

7.1.1. Сегментная организация памяти

Как вы, очевидно, помните, в микропроцессоре 8086 в рамках адресного пространства объемом 1 Мбайт одновременно было доступно четыре сегмента по 64 Кбайт каждый.

В следующих моделях микропроцессоров семейства *x*86[1] в рамках т. н. *защищенного режима* (protect mode, P-режим) организовано *линейное адресное пространство* объемом 2^{32} байтов, в котором допускается создание практически любого числа сегментов.

Если в 8086 единственным атрибутом сегмента был его начальный адрес, то в P-режиме старших моделей семейства *x*86 для описания многочисленных атрибутов предусмотрена специальная структура — дескриптор.

Дескриптор— это 8-байтовый блок, содержащий атрибуты области линейных адресов — *сегмента.* Дескриптор включает в себя информацию о положении сегмента в линейном адресном пространстве, размере сегмента, типе информации, хранящемся в сегменте и правах доступа к ней, а также другие атрибуты сегмента. Формат дескриптора представлен на рис. 7.1.

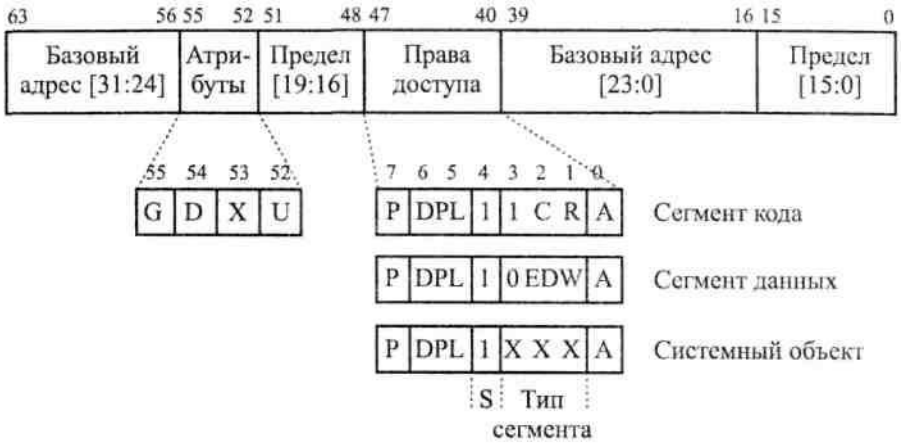


Рис. 7.1. Формат дескриптора

Назначение полей дескриптора:

- *базовый адрес*[2] [31:0] определяет место сегмента (начальный адрес) внутри 4-гигабайтного адресного пространства;
- *предел* [19:0] определяет размер сегмента с учетом бита гранулярности (см. далее).

Поле *атрибутов* включает следующие признаки:

- G — бит гранулярности. При значении G = 0 размер сегмента задается в байтах, а при G = 1 — в страницах по 4 Кбайт. В первом случае максимальный размер сегмента может достигать 1 Мбайт, во втором — 4 Гбайт;
- D — бит размера по умолчанию (от англ. *defaults size)* обеспечивает совместимость с процессором 80286. При D = 0 находящиеся в сегменте операнды считаются имеющими размер 16 битов, иначе — 32 бита:

---

[1] В рамках этой главы будем понимать под обозначением *x*86 микропроцессоры i80386 и старше.

[2] Из рис. 7.1 видно, что дескриптор содержит ряд полей, имеющих название и размер. Например, 32-разрядный адрес "базовый адрес [31:0]"; но в формате данного дескриптора он расположен в разрядах [63:56] и [39:16]. То же относится и к полю "предел". Это связано с желанием фирмы Intel сохранить преемственность форматом младших моделей.

- X — зарезервирован Intel и не должен использоваться программистом (содержит 0);
- U — бит пользователя (от англ. *user)* предназначен для использования системным программистом. Процессор игнорирует этот бит.

Байт *права доступа* (AR) имеет несколько отличающуюся структуру для дескрипторов сегментов разных типов, но некоторые поля этого байта являются общими для всех дескрипторов:

- P — бит присутствия (от англ. *present)* сегмента, если P = 0, то дескриптор не может использоваться, т. к. сегмент отсутствует в ОЗУ. При обращении к сегменту, дескриптор которого имеет P = 0, формируется соответствующее прерывание;
- DRL — уровень привилегий дескриптора (от англ. *descriptor privilege level*) определяет уровень привилегий, ассоциируемый с той областью памяти, которую описывает дескриптор;
- S — определяет роль дескриптора в системе: при S = 0 — системный дескриптор, служит для обращения к таблицам LDT или шлюзам для входа в другие задачи, включая программы обслуживания прерываний. При S = 1 дескриптор обеспечивает обращение к сегментам программ или данных, включая стек;
- A — бит обращения, устанавливается, когда проходит обращение к сегменту. Операционная система может следить за частотой обращения к сегменту путем периодического анализа и очистки A.

Трехбитное поле *тип сегмента* определяет целевое использование сегмента, задавая допустимые в сегменте операции. Значение этого поля для системных дескрипторов (S = 0) безразлично. Для несистемных сегментов биты поля *тип сегмента* имеют следующие значения:

- бит 3 различает сегменты кода (1) и данных (0);
- для сегмента кода бит 2 (Conforming) отмечает при C = 1 т. н. "подчиненные сегменты" (см. далее), а бит 1 (Read) при R = 1 допускает чтение кода как данных с помощью префикса замены сегмента;
- для сегмента данных бит 2 (Expand Down) определяет т. н. "расширение вниз" — для сегментов стека ED = 1, а для сегментов собственно данных ED = 0;
- бит 1 (Write) показывает возможность записи в сегмент при W = 1.

Дескрипторы хранятся в памяти и группируются в дескрипторные таблицы:

- GDT — глобальная дескрипторная таблица;
- IDT — дескрипторная таблица прерываний;
- LDT — локальная дескрипторная таблица.

Причем, если GDT и IDT — общесистемные, присутствуют в системе в единственном экземпляре и являются общими для всех задач, то LDT может создаваться для каждой задачи.

Максимальный размер дескрипторной таблицы может составлять

2^{13} =8192 дескриптора ($2^{13}\times 8$ = 65 536 байтов).

Дескрипторная таблица локализуется в памяти с помощью соответствующего регистра. 48-битовые регистры GDTR и IDTR содержат 32-битовое поле базового адреса таблицы и 16-битный предел (размер) таблицы с байтовой гранулярностью.

Для локализации LDT используется 16-разрядный регистр LDTR, содержащий только селектор сегмента, в котором размещена таблица. Таблицы LDT хранятся как сегменты, а дескрипторы этих сегментов размещаются в GDT. Селектор регистра LDTR выбирает из GDT нужный дескриптор, и атрибуты LDT становятся доступны процессору. С LDTR, как и с сегментными регистрами, ассоциируется соответствующий "теневой

регистр", в который помещается выбранный из GDT дескриптор LDT текущей задачи. При переключении задачи достаточно заменить 16-разрядное содержимое LDTR, а процессор автоматически загрузит теневой регистр.

Доступ к памяти в любом режиме *x*86 возможен лишь в область, определенную как сегмент. Количество доступных в данный момент сегментов определяется числом сегментных регистров (CS, SS, DS, ES, FS, GS). Однако в защищенном режиме содержимое сегментного регистра не является базой сегмента, а рассматривается как селектор сегмента и имеет формат, приведенный на рис. 7.2.

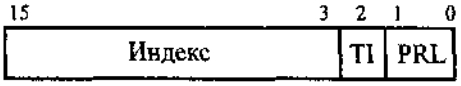


Рис. 7.2. Формат селектора

Индекс определяет смещение внутри дескрипторной таблицы, которая соответственно разрядности индекса может содержать 2^{13} 8-байтовых дескрипторов. Бит TI определяет *тип дескрипторной таблицы:* 0 — глобальная, 1 — локальная. Поле RPL определяет *запрашиваемый уровень привилегий.*

Итак, селектор адресует дескриптор сегмента в одной из дескрипторных таблиц. Всякий раз, когда производится перезагрузка сегментного регистра (замена селектора), адресуемый им дескриптор извлекается из соответствующей дескрипторной таблицы и помещается в "теневой регистр" дескриптора. Все последующие обращения к этому сегменту не требуют чтения из дескрипторной таблицы.

Логический адрес в защищенном режиме, как и в реальном, описывается парой RS:EA, где RS— содержимое выбранного сегментного регистра, EA — эффективный адрес, генерируемый программой (смещение в сегменте).

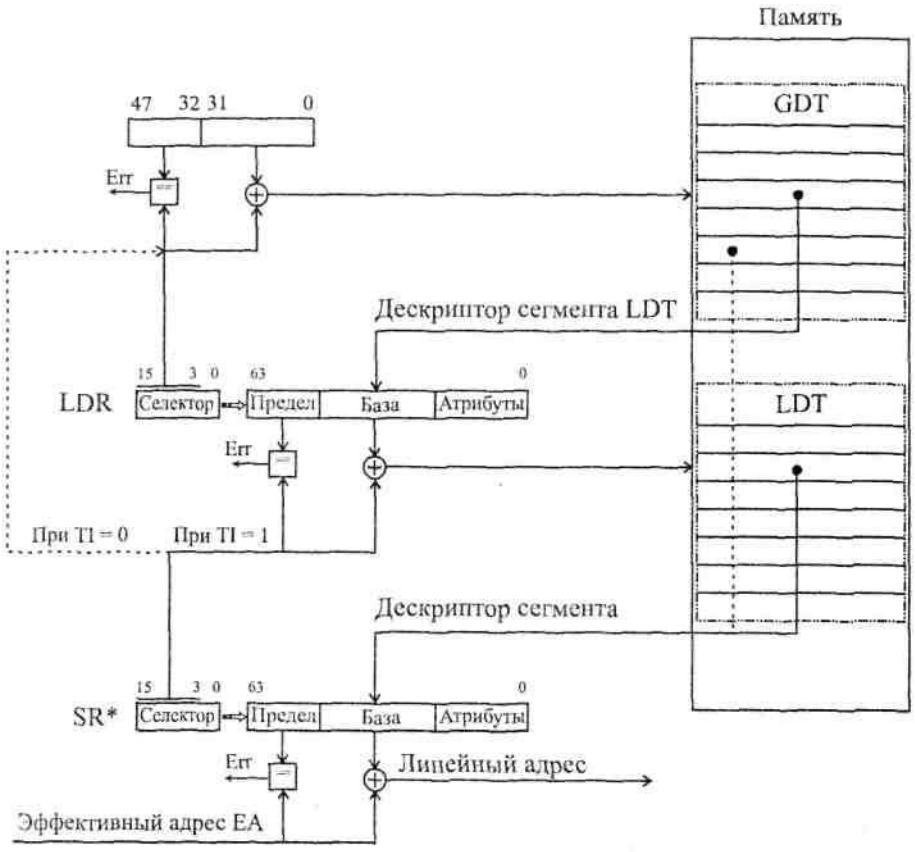


Рис. 7.3. Преобразование логического адреса в линейный

Процесс загрузки дескрипторных регистров и преобразования эффективного (логического) адреса в линейный протекает следующим образом (рис. 7.3):

1. При переходе в защищенный режим в памяти создается глобальная дескрипторная таблица, базовый адрес которой размещается в регистре GDTR.
2. Несколько сегментов определяется в памяти, и их дескрипторы помещаются в GDT.
3. При запуске очередной задачи можно определить дополнительно несколько сегментов и для хранения их дескрипторов создать локальную дескрипторную таблицу, как системный сегмент, дескриптор которого хранится в GDT, а его положение в GDT определяется селектором в регистре LDTR В теневой регистр LDTR автоматически помещается дескриптор сегмента LDT.
4. При загрузке в любой сегментный регистр нового содержимого в соответствующий теневой регистр автоматически помещается новый дескриптор из GDTR или LDTR.
5. При генерации программой очередного адреса EA из соответствующего теневого сегментного регистра выбирается базовый адрес сегмента и складывается со значением EA. Полученная сумма представляет собой линейный адрес.

В приведенной выше процедуре не отражены особые случаи, которые могут возникать при различных нарушениях (ошибках) в процессе формирования линейного адреса.

Механизм сегментации можно искусственно подавить, назначив все базовые адреса сегментов равными нулю и определив длину всех сегментов в 4 Гбайт. Таким образом, в адресном пространстве определится единственный сегмент размером 2^{32} байтов.

Сегмент в защищенном режиме — область памяти, снабженная рядом атрибутов: типом, размером, положением в памяти, уровнем привилегий и др. Сегмент может начинаться и кончаться, где угодно, и его размер — произвольный. Другой элемент памяти — страница— имеет строго фиксированный размер (4 Кбайт) и положение в линейном адресном пространстве: страница всегда выровнена по границе 4-килобайтовых фрагментов, т.е. 12 младших разрядов адреса страницы — всегда нули.

7.1.2. Страничная организация памяти

Наряду с сегментной организацией в микропроцессорах *x*86 возможна дополнительно *страничная организация памяти.* Механизм страничной организации памяти может включаться (выключаться) программно путем установки (сброса) флага PG регистра CR0.

Все линейное адресное пространство делится на разделы, число которых может достигать 1024. Каждый раздел, в свою очередь, может содержать до 1024 страниц (рис. 7.4), размер которых фиксирован— 4 Кбайт, причем начальные адреса страниц жестко фиксированы в физическом адресном пространстве: границы страниц совпадают с границами 4-килобайтовых блоков.

32-разрядный логический адрес, полученный на предыдущем этапе преобразования адреса, рассматривается состоящим из трех полей:

- [31:22] — номер раздела (TABLE);
- [21:12] — номер страницы в разделе (PAGE);
- [11:0] — номер слова на странице (смещение).

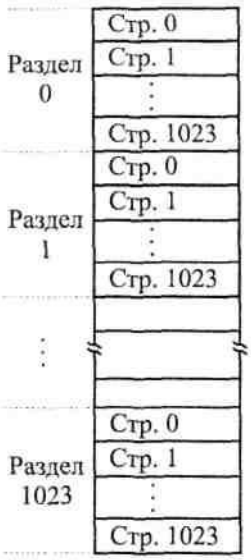


Рис. 7.4. Разделы в линейном адресном пространстве

Начальные адреса страниц данного раздела (вместе с атрибутами страницы) хранятся в памяти в страничной таблице, размер которой 1024 стр. × 4 байта = 4096 байтов.

Поскольку в задаче может быть несколько разделов и, следовательно, столько же страничных таблиц, то начальные адреса всех страничных таблиц одного сегмента хранятся в специальной таблице — *каталоге раздела*.

Линейный 32-разрядный адрес является исходной информацией для формирования 32-разрядного физического адреса (рис. 7.5) с помощью каталога раздела и страничной таблицы (СТ). Старшие 10 разрядов линейного адреса определяют номер строки каталога разделов, который локализуется содержимым системного регистра CR3.

Поскольку каталог разделов имеет размер 1 Кбайт × 4 байта, он занимает точно одну страницу (CR3[11:0] = 0) и содержит 4-байтовые поля, формат которых показан на рис. 7.6. Помимо базового адреса страничной таблицы, это поле хранит атрибуты страницы. Извлеченный из каталога базовый адрес страничной таблицы складывается (конкатенируется) с разрядами [21:12] линейного адреса для получения адреса строки страничной таблицы, из которой, в свою очередь, извлекается базовый адрес страницы. Конкатенацией базового адреса страницы с разрядами [11:0] линейного адреса получается *физический адрес.*

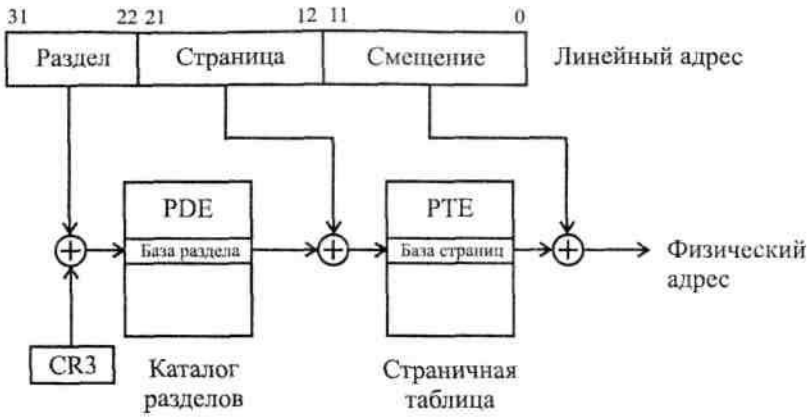


Рис.7.5. Преобразование линейного адреса в физический

Такая двухуровневая организация страничной таблицы позволяет значительно экономить память для хранения страничных таблиц. Действительно, если рассматривать разряды [31:20] линейного адреса как номер строки страничной таблицы, то ее (таблицы)

размер должен составлять $2^{20} \times 4$ байтов, т. е. 4 Мбайт. Абсолютное большинство задач никогда не использует такого количества страниц, однако, во избежание возникновения особого случая (внутреннего прерывания) необходимо поддерживать всю такую таблицу целиком.

При двухуровневой организации страничного преобразования (см. рис. 7.5) в памяти достаточно хранить каталог разделов и страничные таблицы только реально существующих разделов. Максимальное число разделов может достигать 1024, однако во многих случаях достаточно бывает двух-трех разделов, а то и единственного.

Каждая четырехбайтовая строка каталога разделов и страничной таблицы содержит, помимо 20-разрядного базового адреса, атрибуты страницы, определяющие ее назначение, положение в физической памяти, а также информацию, позволяющую аппаратно поддерживать некоторые алгоритмы замещения страниц при страничных сбоях. Формат строки этих таблиц представлен на рис. 7.6.

Атрибуты страницы (СТ) :

- P — бит присутствия, при P = 0 страница отсутствует в оперативной памяти, попытка обращения к ней вызывает прерывание 14 — "страничный сбой";
- R/W— чтение/запись, если работает программа с уровнем привилегий 3 (низший), то при R/W = 0 разрешается только чтение, но не запись на страницу;
- U/S— пользователь/супервизор, при U/S = 0 блокируется запрос с уровнем привилегий 3; при запросе с уровнями привилегий 0, 1, 2 значения битов R/W, U/S игнорируются;
- A — бит доступа, устанавливается процессором при любом обращении к странице;
- D — признак записи на страницу.

Биты A и D используются операционной системой (ОС) для поддержки виртуальной памяти, проверку и сброс этих битов осуществляет ОС. Кроме того, биты 9—11 могут использоваться ОС для своих целей, например, для хранения времени последнего обращения на страницу.

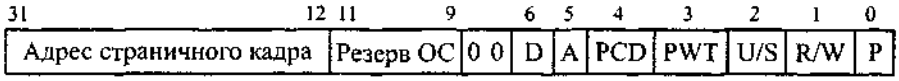


Рис. 7.6. Формат строки каталога разделов и страничных таблиц

В *x*86 предусмотрена *ассоциативная память страничных таблиц,* которая называется *буфером ассоциативной трансляции* — TLB.

TLB представляет собой 32 ячейки АЗУ 1-го рода, поле признаков которого (теги) включают старшие 20 разрядов линейного адреса. Информационное поле ячейки включает 20 старших битов физического адреса страницы и ряд ее атрибутов. Биты D, U/S, R/W имеют тот же смысл, что в слове СТ, а бит достоверности V сбрасывается при записи в CR3 нового слова (смена каталога). После преобразования очередного линейного адреса в физический бит V в этой ячейке устанавливается.

Наличие TLB позволяет при кэш-попадании избежать обращения к ОЗУ при преобразовании линейного адреса. При кэш-промахе микропроцессор выполняет процедуру формирования физического адреса по каталогу раздела и СТ. Полученный из СТ 20-разрядный базовый адрес вместе с 20-разрядным тегом заносятся в свободную ячейку TLB или занимают ячейку, в которой хранится адрес, введенный в TLB ранее других.

Так как TLB хранит адреса 32 страниц по 4 Кбайт, то непосредственно доступными становятся физические адреса 128 Кбайт памяти.

7.1.3. Защита памяти

В *x*86 предусмотрены два вида защиты памяти: на уровне сегментов и на уровне страниц.

Защита памяти на уровне сегментов

В *x*86 определено понятие *привилегии для сегмента* и установлены 4 уровня привилегий PL (рис. 7.7), которые задаются номерами от 0 (наиболее защищенный) до 3 (низший).

В ядро входит часть ОС, обеспечивающая инициализацию работы, управление доступом к памяти, защиту и ряд других жизненно важных функций, нарушение которых полностью выводит из строя МПС. Основная часть ОС должна иметь уровень 1. К уровню 2 обычно относят ряд служебных программ ОС, например, драйверы внешних устройств, системы управления базами данных, специализированные подсистемы программирования и др.

Рис. 7.7. Кольца защиты сегментов

Выше отмечалось, что основой организации памяти *x*86 является сегмент. С каждым сегментом (данных, кода или стека) ассоциируется уровень привилегий DPL и все, что находится внутри этого сегмента, имеет данный уровень привилегий. DPL располагается в байте доступа дескриптора сегмента, поэтому его называют *уровнем привилегий дескриптора* (Descriptor Privilege Level), однако правильнее считать его уровнем привилегий сегмента.

Уровень привилегий выполняющегося кода называется *текущим уровнем привилегий* CPL (Current Privilege Level или Code Privilege Level) и он задается полем RPL селектора в сегментном регистре CS. Значение CPL можно считать уровнем привилегий процессора в текущий момент времени, т. к. при передаче управления сегменту кода с другим уровнем привилегий процессор будет работать на новом уровне привилегий.

Каждый селектор выбирает точно один дескриптор и, соответственно, один сегмент, но конкретный сегмент могут идентифицировать несколько селекторов ("альтернативное именование"). Младшие два бита селектора содержат поле *запрашиваемого уровня привилегий* RPL (Requested Privilege Level). Это поле не влияет на выбор дескриптора, но учитывается при контроле привилегий.

Таким образом, текущее состояние системы защиты характеризуется следующими признаками:

- CPL— уровень привилегий выполняемого кода, размещается в поле RPL сегментного регистра кода CS;
- DPL — уровни привилегий для каждого из восьми открытых сегментов, располагаются в байте доступа дескрипторов, помещенных в "теневые регистры";
- RPL — определяют уровни привилегий источника селектора, размещаются в полях RPL сегментных регистров.

Процессор постоянно контролирует, обладает ли текущая программа достаточным уровнем привилегий, чтобы:

- выполнять некоторые команды;
- обращаться к данным других программ;
- передавать управление внешнему (по отношению к программе) коду командами передачи управления типа FAR.

Рис. 7.8. Правила доступа к сегментам

В системе команд существуют специальные *привилегированные команды,* которые могут выполняться процессором, работающим только на уровне привилегий 0. При попытке выполнить их на другом уровне привилегий генерируется прерывание 13 — нарушение общей защиты.

К привилегированным относятся команды:

- останов процессора;
- сброс флага переключенной задачи;
- загрузка регистров дескрипторных таблиц;
- загрузка регистра задачи;
- загрузка слова состояния машины;
- модификация флага прерываний IF*;
- команды ввода/вывода*.

Последние две группы команд (отмеченные *) не обязательно выполняются на нулевом уровне, достаточно, чтобы уровень привилегий программы был выше уровня привилегий ввода/вывода, определяемого полем IOPL в регистре EFLAGS.

Защита доступа к данным

Данные из сегмента могут выбираться только программой, имеющей такой же или более высокий, чем сегмент, уровень привилегий. Программам не разрешается обращение к данным, которые имеют более высокий уровень привилегий, чем выполняемая программа. Программы могут использовать данные на своем и более низких уровнях привилегий. Ограничения на возможность доступа к данным иллюстрирует рис. 7.8.

Контроль реализуется двумя способами. Во-первых, проверка привилегий осуществляется при загрузке селектора в один из сегментных регистров данных— DS, ES, FS или GS. Если значение DPL того сегмента, который выбирает селектор, численно меньше CPL, процессор не загружает селектор и формирует *нарушение общей защиты.* Во-вторых, после успешной загрузки селектора при использовании его для фактического обращения к памяти процессор контролирует, разрешена ли для этого сегмента запрашиваемая операция (чтение или запись). Кроме того, при обращении контролируется значение запрашиваемого уровня привилегий RPL, причем обращение разрешается, если DPL > max(RPL, CPL), иначе формируется прерывание 13.

Обращение к сегменту стека возможно, если RPL = DPL = CPL, причем сегмент стека должен иметь разрешение на запись — бит W в байте доступа должен быть установлен.

Защита сегментов кода

Межсегментная передача управления происходит по командам JMP, CALL, RET, INT, IRET. Для передачи управления существуют жесткие ограничения: передавать управление в

общем случае можно только в пределах своего уровня привилегий, т. е. DPL целевого дескриптора должен быть точно равен CPL (см. рис. 7.8).

Однако часто бывает необходимо обойти установленные ограничения (например, фрагменты операционной системы могут использоваться программами пользователя). В *x*86 предусмотрены два механизма передачи управления между уровнями привилегий: *подчиненные сегменты* и *шлюзы вызова.*

Если сегмент кода определен как подчиненный (установлен в 1 бит подчиненности C в байте доступа дескриптора), то для него вводятся другие правила защиты. С подчиненными сегментами не ассоциируется конкретный уровень привилегий, он устанавливается равным уровню привилегий вызывающей программы. Поэтому код подчиненных сегментов не должен содержать привилегированных команд.

Когда управление передается подчиненному сегменту, биты поля RPL регистра CS не изменяются на значение поля DPL дескриптора нового сегмента кода, а сохраняют прежнее значение. Только в этой единственной ситуации биты поля RPL регистра CS не соответствуют битам поля DPL дескриптора текущего выполняемого сегмента кода.

При использовании подчиненных сегментов сохраняется одно ограничение — значение DPL дескриптора подчиненного сегмента всегда должно быть меньше или равно текущему значению CPL. Другими словами, передача управления подчиненному сегменту разрешается только во внутренние, более защищенные сегменты. Если бы это ограничение нарушалось, то возврат в вызывающую программу был бы вызовом неподчиненного более защищенного сегмента, что никогда не разрешено.

Наличие подчиненных сегментов кода обеспечивает некоторую свободу передачи управления между уровнями привилегий. Для реализации фактического изменения уровня привилегий привлекаются особые системные объекты, называемые *шлюзами вызова* (рис. 7.9).

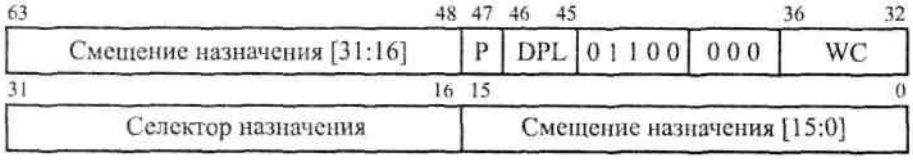


Рис. 7.9. Шлюз вызова

Хотя шлюзы вызова и располагаются в дескрипторной таблице, они, по существу, дескрипторами не являются, т. к. не определяют никакого сегмента. Поэтому в дескрипторе шлюза отсутствует база и граница сегмента, а содержится лишь селектор вызываемого сегмента программы и относительный адрес шлюза — задается фактически адрес *селектор: смещение* точки входа той процедуры (назначения), которой шлюз передает управление. Байт доступа имеет тот же смысл, что и в обычных дескрипторах, а пятибитовое поле WC указывает количество параметров, переносимых из стека текущей программы в стек новой программы.

При этом, если вызываемая программа имеет более высокий уровень привилегий, чем текущая, то для нее по команде CALL создается новый стек, позиция которого определяется из сегмента состояния задачи TSS. В этот стек последовательно записываются: старые значения SS и ESP, параметры, переносимые из старого стека, старые CS и EIP. По команде RET происходит возврат к старому стеку.

Дескриптор шлюза вызова действует как своеобразный интерфейс между сегментами кода на разных уровнях привилегий. Шлюзы вызова идентифицируют *разрешенные точки входа* в более привилегированные программы, которым может быть передано управление.

Селектор, определяющий шлюз вызова, можно загружать только в сегментный регистр CS для передачи управления сегменту кода на другом уровне привилегий.

Защита памяти на уровне страниц

В отличие от 80386, процессоры 80486 и Pentium имеют дополнительные поля в

элементе страничной таблицы [3]. Формат строки СТ i80486 представлен на рис. 7.10, сравните его с форматом на рис. 7.6.

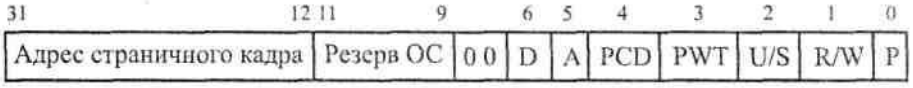


Рис. 7.10. Формат строки СТ процессора 80486

Кроме используемых в i80386 полей:

- P — бит присутствия;
- R/W — чтение/запись;
- U/S — пользователь/супервизор;
- A — бит доступа;
- D — признак записи на страницу;
- резерв ОС

введены биты управления кэшированием:

- PCD — запрет кэширования страницы;
- PWT — сквозная запись.

На уровне страниц в 80486 предусмотрены две разновидности контроля:

- ограничение адресуемой области;
- контроль типа.

Ограничение адресуемой области. Для страниц и сегментов привилегии интерпретируются по-разному: для сегментов— 4 уровня, для страниц — только 2, определяемые битом U/S. При U/S = 0 страница имеет уровень супервизора, иначе— уровень пользователя. На уровне супервизора работают обычно операционные системы, драйверы ВУ, а также располагаются защищенные данные (например, страничные таблицы). Уровни привилегий сегментов отображаются на уровень привилегий страниц: если значение CPL равно 0, 1 или 2. то процессор работает на уровне супервизора, при CPL = 3 — на уровне пользователя. На уровне супервизора доступны все страницы, а на уровне пользователя — только страницы уровня пользователя.

Контроль типа. Механизм защиты распознает только два типа страниц: с доступом только по считыванию (R/W = 0) и с доступом по считыванию/записи (R/W = 1), причем в 80386 ограничение по записи действительно только для уровня пользователя. Программа уровня супервизора игнорирует значение бита R/W и может записывать на любые страницы. В отличие от 80386, процессор 80486 разрешает защитить от записи страницы уровня пользователя в режиме супервизора: установка в регистре CR0 бита WR = 1 обеспечивает чувствительность режима супервизора к защищенным от записи страницам режима пользователя.

Для любой страницы атрибуты защиты ее элемента каталога разделов могут отличаться от атрибутов защиты ее элемента страничной таблицы. Процессор контролирует атрибуты защиты в таблицах обоих уровней и принимает решение таким образом, что всякое разночтение в уровне привилегий раздела и страницы всегда разрешается в сторону большей защиты (предоставления меньших прав пользователю).

7.2. Мультизадачность

Под *мулътизадачностью* понимают способность процессора выполнять несколько задач "одновременно". Конечно, процессор традиционной архитектуры не может выполнять строго одновременно более одного потока команд, однако он может некоторое время выполнять один поток команд, потом быстро переключиться на выполнение другого потока команд, потом третьего, потом — снова первого и т. д. Такая организация вычислительных процессов при высоком быстродействии процессора создает иллюзию одновременности (параллельности) выполнения нескольких задач.

Для реализации мультизадачности необходимо:

- □ располагать быстродействующим процессором;
- □ процессор должен аппаратно поддерживать механизм быстрого переключения задач;
- □ процессор должен аппаратно поддерживать механизм защиты памяти;
- □ использовать специальную мультипрограммную операционную систему.

Под *задачей* в мультизадачной системе понимается программа, которая выполняется или ожидает выполнения, пока выполняется другая задача, причем в определение задачи обычно включают ресурсы, требуемые для ее решения (объем памяти, процессорное время, дисковое пространство и др.).

Рассмотрим, как реализуется механизм переключения задач в процессорах *x*86.

7.2.1. Сегмент состояния задачи

Переключение задач в мультизадачной системе предполагает сохранение состояния приостанавливаемой задачи на момент ее останова. Информация о задаче, сохраняемая для последующего восстановления прерванного процесса, называется ее *контекстом.* В системе выделяется область оперативной памяти, доступная только ОС, в которой хранятся контексты задач. Для минимизации времени переключения контекста следует сохранять и восстанавливать минимальную информацию о каждой задаче.

В какой-то степени процесс переключения задачи напоминает вызов процедуры. Отличие состоит в том, что при вызове процедуры информация о точке возврата (автоматически) и содержимое некоторых РОН (программно) помещается в стек, что определяет свойство реентерабельности процедур (возможность вызова самой себя). Задачи не являются реентерабельными, т. к. контексты сохраняются не в стеке, а в фиксированной (для каждой задачи) области памяти в специальной структуре данных, называемой *сегментом состояния задачи* (Task State Segment, TSS), причем каждой задаче соответствует один TSS.

Сегмент TSS определяется дескриптором, который может находиться только в GDT. Формат дескриптора TSS похож на дескриптор сегмента кода и содержит обычные для дескриптора сегмента поля: базового адреса, предела, DPL, биты гранулярности (G = 0) и присутствия P, бит S = 0 — признака системного сегмента. В поле типа бит занятости B показывает, занята задача или нет. Занятая задача выполняется сейчас или ожидает выполнения. Процессор использует бит занятости для обнаружения попытки вызова задачи, выполнение которой прервано. Поле предела должно содержать значение, не меньшее 67h, что на один байт меньше минимального размера TSS. Формат 32-разрядного TSS представлен на рис. 7.11.

Процедура, которая обращается к дескриптору TSS, может вызвать переключения задачи. В большинстве случаев поле DPL дескрипторов сегментов TSS должно содержать 00, поэтому переключение задач могут проводить только привилегированные программы (на нулевом уровне).

Сегмент TSS не является ни сегментом кода, ни сегментом данных. Доступ к нему имеет только процессор, но не задача, даже на нулевом уровне! Если предполагается программно использовать сегмент TSS, то следует применить *альтернативное именование.*

Обращение к дескриптору TSS не предоставляет возможность процедуре считать или модифицировать сегмент TSS. Загрузка селектора дескриптораTSS в сегментный регистр вызывает особый случай. Доступ к сегменту TSS возможен только с помощью альтернативного именования, когда сегмент данных отображен на ту же область памяти.

Сегмент состояния задачи TSS (рис. 7.11) включает в себя содержимое всех пользовательских регистров процессора, причем 8 регистров общего назначения хранятся в сегменте в том же порядке, в каком они помещаются в стек командой PUSHAD. Кроме того, в TSS сохраняются значения трех указателей стека SS*i* : ESP*i* для трех уровней привилегий— $i \in \{0,1, 2\}$. Сохранение в TSS регистров CS и EIP позволяет осуществлять рестарт задачи, при этом гарантируется правильное действие команд условных переходов, т. к. в TSS сохраняется и EFLAGS. Сохранение в TSS содержимого регистров CR3 и LDTR

позволяет для каждой задачи образовывать свой каталог разделов и локальную дескрипторнуютаблицу.

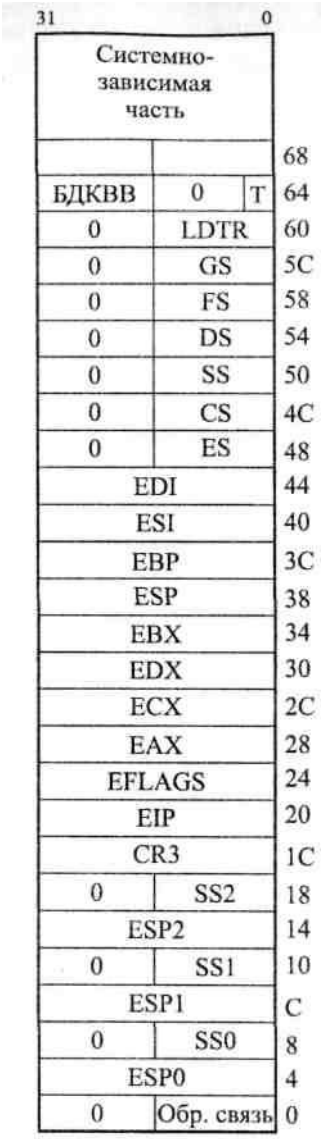


Рис. 7.11. Сегмент TSS

В сегменте TSS имеется также несколько дополнительных полей. Поле *обратной связи* содержит селектор TSS той задачи, которая выполнялась перед данной; с его помощью можно организовать цепь вложенных задач. Поле *базы двоичной карты разрешения ввода/вывода* (БДКВВ) содержит 16-битовое смещение в данном сегменте TSS, с которого начинается сама двоичная карта ввода/вывода. Эта карта позволяет определить произвольное подмножество адресов в пространстве ввода/вывода, по которым данной задаче разрешено обращаться независимо от уровня привилегий. Если в этом поле — 00h, то карта отсутствует. *Бит ловушки T* применяется для отладки: когда в TSS T= 1, при переключении на данную задачу генерируется особый случай отладки (прерывание 1).

При переключении задач между ними не передается никакой информации, т. е. они максимально изолированы друг от друга. Этим исключается искажения задач и обеспечивается возможность прекращения и запуска любой задачи в любой момент времени и в любом порядке.

С целью экономии времени на процедуру переключения задач все поля TSS разделяются на "статические" и "динамические". К статическим относятся поля указателей стека трех уровней и содержимое регистра LDTR — они остаются неизменными в течение всего времени существования задачи. Содержимое статических полей TSS определяется ОС при создании задачи. Статические поля процессор только считывает при переключении

задачи. Поля регистров и поле обратной связи модифицируются при каждом переключении задачи.

До перехода в мультипрограммный режим необходимо определить дескрипторы TSS, разместить сами сегменты TSS в адресном пространстве и правильно инициировать их. Напомним, что селекторы TSS нельзя загружать в сегментные регистры, поэтому для работы с TSS следует пользоваться альтернативным именованием, т. е. псевдонимами этих сегментов. При загрузке начальных значений полей TSS в CS : EIP указывают точку старта программы (задачи), а в регистр SS — селектор сегмента стека с правильным уровнем привилегий. Если предполагается работа задачи на разных уровнях привилегий, следует инициализировать поля SS*i*: ESP*i*, а если задача рассчитана на использование локальной дескрипторной таблицы и страничного преобразования, в сегменте TSS потребуется инициировать поля LDTR и CR3.

В сегменте TSS отсутствуют поля для регистров CR0 и CR2, следовательно, их значение не изменяется при переключениях задач. Поэтому страничное преобразование и условия работы с устройством "плавающей арифметики" (определяются полями CR0 и CR2) являются глобальными для всех задач. Для каждой задачи может быть свой каталог разделов, но страничное преобразование может быть разрешено или запрещено только для всей системы. Переключение задач не затрагивает регистры GDTR и IDTR, а также регистры отладки и проверки.

Минимальный размер сегмента TSS должен быть 104 байта (68h). Однако пользователь может увеличить размер сегмента TSS для размещения дополнительной информации, например, состояние устройства регистров сопроцессора "плавающей арифметики" FPU, списка открытых файлов, двоичной карты ввода/вывода и др. Однако когда процессор привлекает TSS для переключения задач, он игнорирует все данные сверх аппаратно поддерживаемых 104 байтами, и эту дополнительную информацию из TSS считывают программно.

7.2.2. Переключение задачи

Переключение задачи в *x*86 могут вызвать следующие четыре события:

- □ старая задача выполняет команду FAR CALL или FAR JMP, И селектор выбирает *шлюз задачи;*
- □ старая задача выполняет команду FAR CALL или FAR JMP, И селектор выбирает *дескриптор TSS;*
- □ старая задача выполняет команду IRET для возврата в предыдущую задачу; эта команда приводит к переключению задачи, если в регистре EFLAGS *бит вложенной задачи* NT = 1;
- □ возникло аппаратное или программное прерывание и соответствующий элемент дескрипторной таблицы прерываний IDT содержит *шлюз задачи.*

Под термином "старая задача" ("выходящая задача") будем понимать ту задачу, выполнение которой прекращается; под термином "новая задача" ("входящая задача") будем понимать ту задачу, которую начинает выполнять процессор.

Таким образом, селекторами в командах переходов и вызовов могут быть как селекторы TSS (прямое переключение задачи), так и селекторы шлюзов задачи (косвенное переключение задачи). В последнем случае дескриптор шлюза задачи обязательно содержит селектор TSS.

Формат дескриптора шлюза задачи приведен на рис. 7.12.

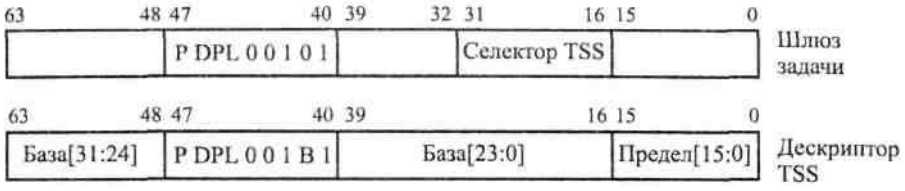


Рис. 7.12. Форматы шлюза задачи и дескриптора TSS

Старая задача должна быть достаточно привилегированна для доступа к шлюзу задачи или к сегменту TSS. Правила привилегий обычные:

- max(CPL, RPL) > DPL шлюза задачи:
- max(CPL, RPL) > DPL сегмента TSS.

Процедура возврата из прерываний IRET всегда возвращает управление прерванной программе. Если флаг NT сброшен в 0, производится обычный возврат, а если он установлен в 1 — происходит переключение задачи. При этом процессор сохраняет свое состояние в сегменте TSS старой задачи, загружает в регистр TR содержимое поля обратной связи — селектор новой задачи ("задачи-предка", т. к. осуществляется возврат) и восстанавливает из сегмента TSS контекст новой задачи. Благодаря наличию в каждом сегменте TSS поля обратной связи можно поддерживать многократные вложения задач. Характерно, что команда возврата из подпрограммы RET не чувствительна к значению флага NT и не может осуществить переключение задачи.

После модификации TR и загрузки нового контекста из сегмента TSS процессор отмечает этот сегмент как занятый (устанавливает бит 41 занятости Busy в его дескрипторе). Занятый TSS может относиться либо к выполняющейся, либо ко вложенной задаче. Переключение на задачу, отмеченную как занятая, не производится! В частности, это исключает возможность реализации реентерабельных задач. Исключение представляет только команда IRET, которая возвращает управление задаче-предку (очевидно, будучи вложенной, она отмечена как запятая).

При переключении задачи процессор устанавливает также флаг переключения задачи TS в регистре CR0. Сброс этого флага может осуществиться только привилегированной командой CLTS. TS применяется для правильного использования некоторых системных ресурсов, в частности — устройства плавающей арифметики (Float Point Unit, FPU). Если при каждом переключении задачи сохранять состояние FPU, то на это уйдет много времени, причем новая задача может вообще не использовать ресурсы FPU, и тогда такое сохранение окажется напрасным. В процессоре 80486 команды FPU анализируют состояние флага TS и если TS = 1, формируется *особый случай 7* и вызывается системная процедура сохранения состояния FPU. После этого флаг TS сбрасывается.

При переключении задачи процессор не фиксирует факт использования новой задачей FPU. Очевидно, это забота обработчика прерывания 7 — сам обработчик или ОС может поддерживать в TSS флаг использования сопроцессора. Кроме того, обработчик прерывания 7 может запоминать селектор TSS последней программы, использующей FPU.

Итак, процесс переключения задачи можно представить следующим образом.

Имеется TR с теневым регистром дескриптора TSS, определяющий TSS старой задачи. Если селектор в командах FAR JMP, FAR CALL, IRET (NT = 1), INT указывает прямо (дескриптор TSS) или косвенно (шлюз задачи) в GDT на системный объект переключения задачи, то производится переключение задачи:

- процессор сохраняет контекст старой задачи в сегменте TSS старой задачи;
- процессор загружает в TR селектор сегмента новой задачи;
- процессор загружает в сегмент TSS новой задачи селектор TSS старой задачи (в поле обратной связи);
- получив доступ к сегменту TSS новой задачи, процессор загружает контекст новой задачи в регистры (в том числе CS : EIP — точка старта);
- процессор устанавливает флаги NT (в регистре EFLAGS) и TS (в CR0 для анализа командами FPU), устанавливает бит занятости задачи в дескрипторе TSS новой задачи.

7.3. Прерывания и особые случаи

Прерывания текущей программы могут возникать по следующим трем причинам:

- внешний сигнал по входам FNTR или NMI;

- аномальная ситуация, сложившаяся при выполнении конкретной команды и зафиксированная аппаратурой контроля;
- находящаяся в программе команда прерывания INT *n*.

Первая из указанных выше причин относится к аппаратным прерываниям, а две другие — к программным.

Программные прерывания, вызываемые причинами 2 и 3, называют обычно *особыми случаями* (иногда используют термин *исключения).* Особые случаи возникают, например, при нарушении защиты по привилегиям, превышении предела сегмента, делении на ноль и т. д.

Все особые случаи классифицируются как нарушения, ловушки или аварии.

Нарушение (fault) — этот особый случай процессор может обнаружить до возникновения фактической ошибки (например, нарушение правил привилегий или отсутствие сегмента в оперативной памяти). Очевидно, после обработки нарушения можно продолжить программу, осуществив рестарт виновной команды.

Ловушка (trap) — обнаруживается после окончания выполнения виновной команды. После ее обработки процессор возобновляет действия с той команды, которая следует за "захваченной" (например, прерывание при переполнении или команда INT *n*). Большинство отладочных контрольных точек также интерпретируются как ловушки.

Авария (abort)— приводит к потере контекста программы, ее продолжение невозможно. Причину аварии установить нельзя, поэтому осуществить рестарт программы не удается, ее необходимо прекратить. К авариям ("выходам из процесса") относятся аппаратные ошибки, а также несовместимые или недопустимые значения в системных таблицах.

Общая реакция процессора на прерывания или особые случаи состоит в сохранении минимального контекста прерываемой программы (в стеке — адрес возврата и, может быть, некоторую дополнительную информацию), идентификации источника прерывания или особого случая и передаче управления соответствующему обработчику (программе, подпрограмме, задаче). Однако имеются принципиальные различия по формированию сохраняемого контекста.

При возникновении нарушений в стек обработчика особого случая в качестве адреса возврата включается CS : EIP команды, вызвавшей нарушение.

При распознавании ловушки (к ним относятся и большинство внешних прерываний) процессор включает в стек адрес возврата, относящийся к следующей за ловушкой команде.

Наконец, при авариях содержательный адрес возврата отсутствует, поэтому рестарт задачи при авариях невозможен.

Принцип реализации прерываний (внешних) и особых случаев (внутренних) в микропроцессорах фирмы Intel сохранился практически неизменным с МП 8086.

В 8086 сигналы запросов на обработку прерываний формировались либо аппаратурой контроля процессора (внутренние прерывания), либо поступали из внешней среды на входы процессора INTR или NMI. При обнаружении (разрешенного) запроса в стек помещались текущие значения FLAGS, CS и IP.

В процессе идентификации источника запроса ему ставился в соответствие восьмиразрядный двоичный код *n* — вектор прерывания, причем за каждым внутренним прерыванием и за внешним NMI жестко закреплялся свой вектор, а для запросов, поступивших по входу INTR, реализовывалась процедура ввода вектора с внешней шины. Далее, определенный вектор прерывания рассматривался как номер строки таблицы, располагающейся с нулевого физического адреса памяти. Четырехбайтовыми элементами этой таблицы были адреса CS : IP точек входа в подпрограммы — обработчики прерываний. Таким образом, в системе поддерживалось до 256 различных обработчиков прерываний, причем векторы 0—31 резервировались за внутренними прерываниями (и NMI), а остальные — для внешних прерываний.

При идентификации источника запроса INTR выполняются следующие действия

(при условии, что флаг IF = 1, иначе запрос INTR игнорируется):

1. Генерируется два цикла шины для ввода вектора внешнего прерывания.
2. Помещается в стек содержимое регистра FLAGS.
3. Помещается в стек содержимое регистра CS.
4. Помещается в стек содержимое регистра IP.
5. Сбрасывается в 0 флаг разрешения внешних прерываний IF, запрещая восприятие новых запросов по входу INTR до явной установки флага IF в 1 командой STI.
6. По значению вектора *n* обращаются к *n*-му элементу таблицы векторов прерываний и из нее загружаются новые значения регистров CS : IP.
7. Начинается выполнения обработчика прерывания с точки входа, определяемой CS : IP.

Сохраненное в стеке старое содержимое регистров CS : IP образует адрес возврата. Когда обработчик прерываний заканчивает свои действия, он должен выполнить команду возврата IRET, которая, извлекая из стека содержимое FLAGS, CS, IP, возвращает управление прерванной программе.

Механизм реализации внешних и внутренних прерываний МП 80486 и Pentium в R-режиме аналогичен описанному выше, однако в P-режиме он значительно усовершенствован:

- □ таблица векторов прерываний трансформирована в дескрипторную таблицу прерываний IDT (рис. 7.13);
- □ более сложен процесс перехода к обработчику прерывания или особого случая;
- □ обработчику передается дополнительная информация о причине возникновения особого случая.

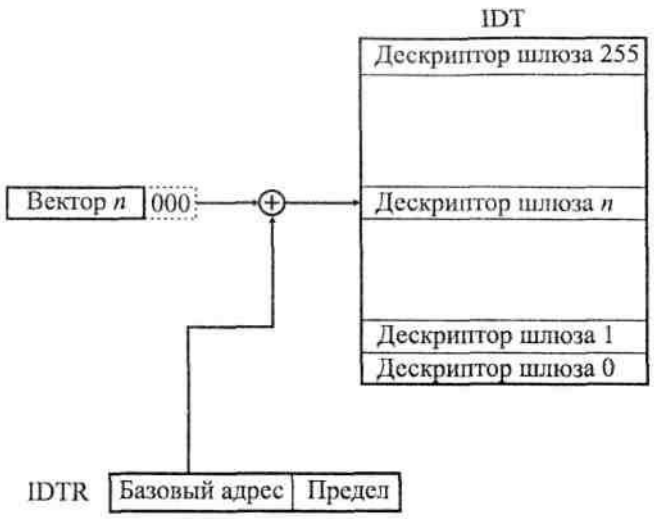


Рис. 7.13. Дескрипторная таблица прерываний

Механизм передачи управления обработчику особого случая (прерывания) соответствует обычному способу передачи управления через шлюз вызова. При этом процессор аппаратно включает в стек значения EFLAGS, CS : EIP (адрес возврата) прерываемой программы и, кроме того, в некоторых случаях— код ошибки и текущие значения SS, ESP (последние — при смене привилегий).

Точное значение адреса возврата зависит от того, является ли особый случай нарушением, ловушкой или аварией. Первые 32 вектора зарезервированы за особыми случаями P-режима (табл. 7.1).

Таблица 7.1. Особые случаи

Вектор	Причина	Тип
0	Ошибка деления	Нарушение
1	Отладка	Нарушение/ловушка
2	Немаскируемое прерывание NMI	Ловушка
3	Контрольная точка	Ловушка
4	Переполнение	Ловушка
5	Нарушение границы массива	Нарушение

Таблица 7.1 *(окончание)*

Вектор	Причина	Тип
6	Недействительный код операции	Нарушение
7	Устройство недоступно	Нарушение
8	Двойное нарушение*	Авария
9	Не используется	
10	Недействительный TSS*	Нарушение
11	Неприсутствие сегмента*	Нарушение
12	Нарушение стека*	Нарушение
13	Нарушение общей защиты*	Нарушение/ловушка
14	Страничное нарушение*	Нарушение
15	Зарезервирован	
16	Ошибка операции с плавающей точкой	Нарушение
17	Контроль выравнивания*	Нарушение
18—31	Зарезервированы	

Примечание. * — включает в стек код ошибки.

7.3.1. Дескрипторная таблица прерываний

Дескрипторная таблица прерываний IDT является прямой заменой таблицы векторов прерываний процессора 8086. Она должна определять 256 обработчиков прерываний и особых случаев, поэтому ее максимальный размер составляет 256 × 8 = 2048 байтов. Таблица IDT может находиться в любой области памяти, процессор локализует ее с помощью 48-битного регистра IDTR, который содержит базовый адрес и предел таблицы. Таблицу не рекомендуется объявлять короче максимального размера, т. к. любое обращение за пределы таблицы вызывает нарушение общей защиты, а вектор внешнего прерывания, вообще говоря, может иметь любое значение в диапазоне 00—FFh.

В таблице IDT разрешается применять только три вида дескрипторов: *шлюз ловушки, шлюз прерывания* и *шлюз задачи* (но не дескриптор TSS). Шлюзы прерывания и ловушки имеют большое сходство со шлюзом вызова (см. рис. 7.9). Единственное отличие состоит в отсутствии в этих шлюзах 5-битового поля счетчика WC, которое в шлюзе вызова определяет число параметров, передаваемых в вызываемую подпрограмму через стек. Соответствующее поле в шлюзах прерывания и ловушки зарезервировано.

Напомним, что шлюз вызова (а также ловушки и прерывания) содержит селектор сегмента кода и смещение внутри него, которые однозначно определяют точку передачи управления. Шлюз задачи содержит лишь селектор сегмента состояния задачи TSS (разумеется, каждый дескриптор содержит и байт доступа).

После локализации сегмента кода обработчика особого случая и включения информации в стек, выполнение начинается с той команды, которая определяется

смещением в шлюзе ловушки. Обработчик действует до тех пор, пока не достигнет команды IRET. По этой команде процессор извлекает из стека адрес возврата и содержимое регистра флажков (а также 48-битный указатель внешнего стека, если при обработке особого случая происходила смена уровней привилегий).

Если особый случай вызывается через шлюз прерывания, процессор сбрасывает флаг разрешения прерывания IF после включения в стек адреса возврата и содержимое регистра флажков, но до выполнения первой команды обработчика. При переходе через шлюз ловушки никакие флаги не изменяются.

Обработка особого случая через шлюз задачи аналогична действию команды FAR CALL, приводящей к переключению задачи. Однако здесь невозможен прямой переход через дескриптор TSS, а требуется промежуточный шлюз задачи. С дескриптором шлюза задачи ассоциируется уровень привилегий, который не должен быть выше уровня привилегий прерываемой задачи. По существу, прерываемая задача как бы выполняет команду FAR CALL вызова другой задачи через шлюз задачи, и здесь действуют стандартные правила защиты по привилегиям.

Обработка особого случая через шлюз задачи, т. е. в другой задаче, имеет определенные преимущества:

- автоматически сохраняется весь контекст прерванной задачи;
- обработчик особого случая не может исказить прерванную задачу, т. к. он полностью изолирован от нее;
- обработчик прерывания может работать на любом уровне привилегий и в заведомо правильной среде; он может иметь свое локальное адресное пространство благодаря наличию отдельной локальной дескрипторной таблицы (при необходимости).

К недостаткам применения шлюза задачи для вызова обработчика можно отнести:

- замедленную реакцию процессора на особый случай;
- в шлюзе задачи невозможно определить начальную точку выполнения задачи;
- сложность получения информации о прерванной задаче.

Переключение задачи, инициируемое особым случаем, производит вложение задачи обработчика в прерванную задачу. Старая задача остается занятой, а новая — обработчик особого случая — отмечается как занятая, причем в ней будет установлен флажок NT и загружено поле обратной связи. Состояние флага IF в новой задаче не меняется и определяется тем значением бита регистра EFLAGS, которое хранилось в TSS. Поэтому, если обработчик особого случая через шлюз задачи предназначен для обработки внешних прерываний, следует в TSS установить IF = 0, "аппаратно" запретив внешние прерывания на время работы обработчика, или пока он явно не установит IF = 1.

7.3.2. Учет уровня привилегий

Все особые случаи должны обрабатываться через шлюзы. В дескрипторах шлюзов всех трех типов, содержащихся в IDT, имеется поле уровня привилегий дескриптора DPL, определяющее минимальный уровень привилегий, необходимый для использования шлюза. Для обработчиков прерываний рекомендуется устанавливать DPL = 3, чтобы обработка особого случая не зависела от уровня привилегий текущей задачи.

Шлюзы ловушек или прерываний должны передавать управление сегменту кода с более высоким или равным уровнем привилегий. Обработчику не разрешается работать на уровне привилегий, который ниже уровня прерываемой задачи (если, конечно, он не является отдельной задачей). Из-за непредсказуемости возникновения прерываний и особых случаев требуется гарантировать невозможность нарушения правил защиты по привилегиям при обработке особого случая. Этого можно достичь двумя способами:

- определить все обработчики особых случаев, не вызывающие переключения задачи, в сегментах кода с уровнем привилегий 0; такие обработчики будут действовать всегда, независимо от значения CPL программы;

- определить все обработчики особых случаев, не вызывающие переключения задачи, в подчиненные сегменты кода.

7.3.3. Код ошибки

В некоторых особых случаях процессор включает в стек 4 байта кода ошибки (error code), причем действительными являются только 2 младших байта, остальные включаются лишь для выравнивания стека. Когда процессор обнаруживает

- недействительный сегмент TSS:
- нарушение неприсутствия;
- нарушение стека;
- нарушение общей защиты,

он включает в стек обработчика особого случая информацию, идентифицирующую "виновный" дескриптор.

Формат кода ошибки напоминает селектор, т. к. большинство особых случаев связано с ошибками дескрипторов и содержит следующие поля:

- биты [15:3] — индекс (номер строки дескрипторной таблицы);
- бит [2] — TI, как и в других селекторах, определяет принадлежность дескриптора к локальной (TI = 1) или глобальной (TI = 0) дескрипторной таблице;
- бит [1] — I. Если 1 = 1, то индекс в старших битах [15:3] кода ошибки относится к дескрипторной таблице прерываний IDT;
- бит [0] — EXT = 1 означает, что особый случай был вызван аппаратным прерыванием (внешним) или возник, когда процессор обрабатывал другой особый случай.

Если процессор не может сформировать содержательный код ошибки, он включает в стек код, равный 0.

Помимо кода ошибки в некоторых особых случаях дополнительная диагностическая информация находится в других регистрах процессора. Например, при страничном нарушении в регистре CR2 содержится линейный адрес, преобразование которого привело к ошибке. Обработчик этого особого случая может обратиться к соответствующим элементам PDE и PTE. Для особого случая отладки полезная информация содержится в регистре состоянш отладки DR6.

7.3.4. Описание особых случаев

Далее приводится краткое описание действий процессора *x*86 при возникновении особых случаев [3].

Ошибка деления (0) — автоматически формируется, когда в команде DIV итн IDIV делитель равен нулю или частное слишком велико для получателя (AL/AX/EAX).

Отладка (1)— формируется в следующих случаях (может быть нарушением или ловушкой):

- нарушение контрольной точки по адресу команд;
- ловушка контрольной точки по адресу данных;
- нарушение общей защиты;
- ловушка покомаидной работы (флаг TF =1);
- ловушка контрольной точки по переключению задачи (в сегменте TSS бит T= 1).

Немаскируемое прерывание NMI (2) — единственное внешнее радиальное прерывание.

Контрольная точка (3) — формируется при выполнении команды INT3 (код

операции — CCh). Передача управления обработчику особого случая является частью команды INT3, адрес возврата в стеке относится к началу следующей команды. Тот же обработчик вызывается при выполнении внешнего прерывания с вектором 03 или двухбайтовой команды INT 03.

Переполнение (4)— возникает при выполнении команды INTO при условии установки в 1 флага переполнения OF. Как и для INT3 передача управления обработчику особого случая является частью команды INTO, адрес возврата в стеке относится к началу следующей команды. Обычно команда INTO применяется в компиляторах для выявления переполнения в арифметике знаковых чисел. Тот же обработчик вызывается при выполнении внешнего прерывания с вектором 04 или команды INT 04.

Нарушение границы массива (5) — возникает при выполнении команды BOUND, если контрольная проверка дает отрицательный результат, т. е. проверяемый (первый) операнд не попадает в диапазон значений, определенных вторым (нижняя граница) и третьим (верхняя граница) операндами команды.

Недействительный код операции (6)— генерируется, когда операционное устройство процессора обнаруживает неверный код операции, несоответствие типа операндов коду операции, попытку выполнения привилегированных команд в R-режиме, неверные байты mod r/m или sib, использование префикса блокировки LOCK с командами, которые нельзя блокировать. Характерно, что существует несколько одно- и двухбайтовых кодов, зарезервированных фирмой Intel для развития системы команд, и хотя им в 80486 не соответствуют никакие команды, зарезервированные коды не вызывают особого случая.

Устройство[1] *недоступно* (7) — возникает в двух ситуациях:

- процессор выполняет команду ESC и бит EM (эмуляция сопроцессора) в регистре CR0 установлен в 1;
- процессор выполняет команду WAIT или ESC и бит TS (переключение задачи) в регистре CR0 установлен в 1.

В первом случае программист намерен выполнить операции плавающей арифметики программно.

Второй случай может возникнуть после переключения задачи. Бит TS аппаратно устанавливается в 1 при переключении задачи, а первая же встретившаяся в новой задаче команда сопроцессора вызывает особый случай 7, ибо контекст устройства с плавающей точкой старой задачи не сохранен. Обработчик особого случая 7 сохраняет старое состояние устройства с плавающей точкой в сегменте TSS старой задачи и загружает новое состояние устройства из сегмента TSS новой задачи. (В случае работы цепочки вложенных задач обработчик должен программно отследить ту старую задачу, которая последней использовала FPU.) После этого привилегированной командой CLTS сбрасывается флажок TS и осуществляется возврат на команду устройства с плавающей точкой. Так как теперь флаг TS сброшен, команда сопроцессора будет выполнена.

Двойное нарушение (8) — обычно, когда процессор обнаруживает особый случай при, попытке вызвать обработчик предыдущего особого случая, два особых случая обрабатываются последовательно. Если процессор не может обрабатывать их последовательно, он сигнализирует о двойном нарушении.

Для определения того, когда о двух нарушениях следует сообщать как о двойном нарушении, процессор подразделяет все особые случаи на три класса:

- легкие особые случаи — векторы 1, 2, 3, 4, 5, 6, 7,16;
- тяжелые особые случаи — векторы 0, 10, 11, 12, 13;
- страничные нарушения — вектор 14.

Когда возникают два легких особых случая или один легкий и один тяжелый, эти два события допускают последовательную обработку. При появлении двух тяжелых событий их обработать нельзя, поэтому процессор формирует особый случай двойного

[1] Под устройством здесь понимается аналог математического сопроцессора — устройство с плавающей точкой FPU.

нарушения. Аналогичное состояние наступает, если после страничного нарушения возникает тяжелый особый случай или второе страничное нарушение, хотя если, наоборот, после тяжелого особого случая возникает страничное нарушение, эти события могут быть обработаны.

Если при попытке вызвать обработчик двойного нарушения возникает любое другое нарушение, процессор переходит в режим отключения. Этот режим аналогичен состоянию процессора после выполнения команды HLT. До восприятия сигналов NMI или RESET никакие команды не выполняются, причем если отключение возникло при выполнении обработчика немаскируемого прерывания, запустить процессор может только сигнал RESET.

Недействительный сегмент TSS (10)— возникает при попытке переключения на задачу с неверным сегментом TSS. Поскольку сегмент TSS может определять LDT, сегменты кода, стека, данных, к особому случаю 10 относятся ситуации с нарушением границ этих сегментов, нарушением прав доступа, запретом записи в сегмент стека и др. Нарушения могут возникать как в контексте старой, так и новой задачи, поэтому обработчик особого случая 10 должен сам быть задачей и вызываться через шлюз задачи (десятая строка IDT должна содержать шлюз задачи).

Неприсутствие сегмента (11)— формируется, когда бит присутствия сегмента в дескрипторе P = 0. Это нарушение допускает рестарт, если обработчик особого случая 11 реализует механизм виртуальной памяти на уровне сегментов.

Нарушение стека (12) — возникает в двух ситуациях:

- в результате нарушения предела любой операции, которая обращается к регистру SS (POP, PUSH, ENTER, неявное использование стека при обращении к памяти, например, MOV AX, [BP+6]);
- при попытке загрузить в регистр SS дескриптор, который отмечен неприсутствующим.

Нарушение общей защиты (13) — все нарушения защиты, которые не служат причиной конкретного особого случая, вызывают особый случай общей защиты:

- превышение предела сегмента (кроме стека);
- передача управления сегменту, который не является выполняемым;
- запись в защищенный от записи сегмент;
- считывание из выполняемого сегмента;
- загрузка в SS селектора сегмента, защищенного от записи;
- загрузка в регистры SS, DS, ES, FS, GS селектора системного сегмента;
- загрузка в регистры SS, DS, ES, FS, GS селектора выполняемого сегмента;
- обращение к памяти через DS, ES, FS, GS, когда в них пустой селектор;
- переключение на занятую задачу;
- нарушение правила привилегий

  и др.

Страничное нарушение (14)— возникает, когда разрешено страничное преобразование и имеет место одна из следующих ситуаций:

- в элементе каталога разделов или таблицы страниц, используемом для преобразования линейного адреса в физический, сброшен бит присутствия;
- процедура не имеет достаточного уровня привилегий для доступа к адресуемой странице.

Ошибка операции с плавающей точкой (16)— сигнализирует об ошибке, возникшей в команде устройства с плавающей точкой.

Контроль выравнивания (17)— возникает при нарушении выравнивания операндов. Операнды считаются выровненными, если адрес двухбайтового слова является четным (младший разряд равен 0), адрес четырехбайтового двойного слова кратен 4, а адрес восьмибайтовой структуры данных кратен 8. Для разрешения контроля выравнивания должны выполняться три условия:

- бит AM в регистре CR3 установлен;
- флаг AC установлен;
- выполняется программа на уровне привилегий 3.

7.4. Средства отладки

Традиционно средства отладки микропроцессоров ограничивались наличием:

- короткой команды программного прерывания, которую можно было устанавливать вместо первого байта любой команды;
- аппаратной реализации пошагового (покомандного) режима, который инициировался установкой специального бита T в регистре флагов.

По мере усложнения МПС возможности внешних аппаратных средств по наблюдению операций, происходящих внутри процессора, уменьшаются. Поэтому в схемах мощных процессоров стали предусматривать разнообразные средства отладки.

Основу средств отладки в процессорах *x*86 старших моделей составляют специализированные регистры отладки — программируемые регистры задания контрольных точек, регистры управления и состояния отладки. Они заменяют собой средства аппаратных внутрисхемных эмуляторов. Процессоры *x*86 старших моделей обеспечивают не только покомандную работу, но и регистрацию переключения на конкретную задачу, установку контрольных точек по адресам команд и фиксацию модификации значений переменных в памяти.

Регистры отладки поддерживают контрольные точки по командам и данным. В общем, под *контрольной точкой* понимается адрес, при использовании которого программой возникает особый случай отладки. Установка контрольной точки по команде обеспечивает регистрацию команды по любому линейному адресу. Задание контрольной точки по данным позволяет узнать, когда производится обращение к конкретной переменной.

Отладочные средства *x*86 включают в себя:

- однобайтовую команду контрольной точки INT3, которую можно вставлять в программу по любому адресу; при выполнении этой команды генерируется особый случай отладки;
- флаг пошагового режима TF в регистре EFLAGS, позволяющий выполнить программу по командам;
- четыре регистра отладки DR0—DR3, которые определяют четыре независимые контрольные точки по командам или данным; регистр управления отладкой DR7 и регистр состояния отладки DR6;
- флаг ловушки T в TSS, который вызывает особый случай отладки при переключении на задачу с установленным в 1 битом T;
- флаг возобновления RF в регистре EFLAGS, с помощью которого подавляются многократные особые случаи в одной и той же команде.

Все эти средства действуют как *ловушки,* следя за возникновением условий, представляющих интерес для программиста. Когда возникает такое условие, формируется особый случай отладки (с вектором 1); только команда INT3 генерирует прерывание с вектором 3. Зарезервированный вектор отладки 1 упрощает процедуру вызова отладчика.

Итак, отладчик по вектору 1 вызывается в следующих случаях:

- при TF = 1 после каждой команды;
- при TTSS = 1 в момент переключения на задачу;
- ловушка контрольной точки по данным;
- нарушение контрольной точки по команде.

Команда INT3 предоставляет альтернативный способ задания контрольной точки и особенно удобна, если контрольные точки размещаются в исходном коде или требуется установить более четырех контрольных точек.

Программные контрольные точки задаются путем замены обычных команд на команды INT3 — при этом требуется осуществлять запись в сегмент кода (создавать альтернативный сегмент данных), а после отладки — восстанавливать исходный код. Использование аппаратных контрольных точек исключает необходимость модификации кода (можно отлаживать программу, размещенную в ПЗУ), а так же допускает контроль обращения к данным.

Рассмотренные средства позволяют вызывать отладчик как процедуру в контексте текущей задачи или как отдельную задачу при выполнении одного из следующих условий:

- выполнение команды контрольной точки INT3;
- выполнение любой команды (при TF = 1);
- выполнение команды по указанному адресу;
- считывание или запись байта, слова или двойного слова по указанному адресу;
- запись байта, слова или двойного слова по указанному адресу;
- переключение на конкретную задачу;
- попытка изменить содержимое регистра отладки.

7.4.1. Регистры отладки

Регистры отладки, форматы которых приведены на рис. 7.14, включают в себя:

- четыре регистра DR0—DR3, предназначенные для хранения линейных адресов четырех контрольных точек, каждая из которых независимо может быть определена как контрольная точка по команде или по данным;
- регистр DR7 управления отладкой, включающий поля, которые определяют свойства контрольных точек и некоторые параметры процесса отладки;
- регистр DR6 состояния отладки, предназначенный для идентификации причины прерывания отладки;
- наконец, зарезервированные регистры DR5, DR4.

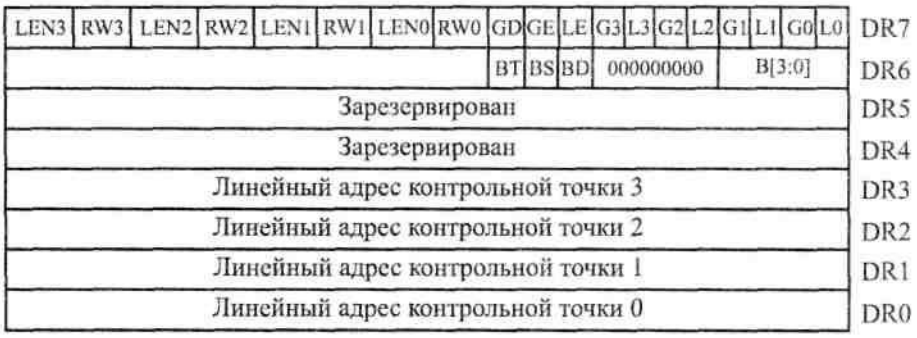


Рис. 7.14. Форматы регистров отладки

Рассмотрим форматы регистров состояния и управления.

Выше отмечалось, что все события отладки, кроме INT3, вызывают прерывание с вектором 1. Следовательно, при возникновении особого случая отладки встает вопрос о причине прерывания. Именно *для идентификации причин прерывания 1* предусмотрен регистр состояния отладки DR6.

Младшие четыре бита B0—B3 относятся к четырем контрольным точкам и единичное состояние B*i* означает достижение контрольной точки, линейный адрес которой находится в регистре DR*i*.

Флаг BS (Step) устанавливается в 1, когда процессор начинает отрабатывать особый случай, вызванный ловушкой покомандной работы, т. е. при BS = 1 причиной особого случая является состояние TF = 1 регистра EFLAGS. Этот случай имеет высший приоритет среди всех случаев отладки, когда BS = 1, могут быть установлены и другие биты состояния отладки.

Флаг BT (Task) устанавливается в 1, когда особый случай отладки вызван переключением на задачу, в TSS которой установлен бит ловушки T =1.

Регистры отладки доступны (по записи или чтению) только в реальном режиме или в защищенном режиме по привилегированной (т. е. разрешенной к выполнению только на нулевом уровне) команде MOV:

```
mov eax, dr6
mov dr1, eax
```

В регистре DR7 предусмотрен флаг GD, который, будучи установленным, обеспечивает "сверхзащиту" всех обращений к регистрам отладки, вызывая при любой попытке обращения к этим регистрам *прерывание 1*. Для идентификации этой ситуации в DR6 предусмотрен бит BD. Он устанавливается в 1, если следующая команда будет считывать или записывать в один из восьми регистров отладки. Характерно, что при вызове процедуры обработчика с вектором 1 бит GD автоматически сбрасывается, что обеспечивает обработчику возможность доступа к регистрам отладки.

Процессор никогда не сбрасывает биты регистра состояния отладки DR6, поэтому обработчик особого случая должен сбрасывать их программно, иначе причины особых случаев отладки будут накапливаться.

Возможно, при возникновении особого случая отладки в состоянии 1 будут находиться несколько битов DR6 (что возможно при разрешении аппаратных контрольных точек) или, наоборот, в DR6 не окажется единичных битов. Последнее возможно, если возникло внешнее прерывание с вектором 1 или выполняется команда INT1.

Регистр управления отладкой DR7 содержит для каждой из четырех контрольных точек следующие поля, определяющие ее характеристики: L*i*, G*i*, RW*i* и LEN*i*, а также два однобитовых поля LE и GE, определяющие свойства, общие для всех контрольных точек. Назначение этих полей приведены в табл. 7.2. Кроме того, в регистре DR7 бит 13— GD— предназначен для включения режима защиты регистров отладки от любого обращения со стороны программ пользователя.

Таблица 7.2. Назначение полей регистра DR7

Поле	Назначение
L*i*	Локальное разрешение контрольной точки $i \in \{0, 1, 2, 3\}$. Когда бит L*i* находится в 1, разрешена аппаратная контрольная точка, линейный адрес которой находится в DR1, но только в текущей задаче. При переключении задачи сбрасываются все биты L*i*

Таблица 7.2 *(окончание)*

Поле	Назначение
G*i*	Глобальное разрешение контрольной точки *i*. Функционирует аналогично биту L*i*, но на него не действует переключение задач. Сбросить бит G*i* можно только программно
LE	Локальная точность
GE	Глобальная точность
RW*i*	Определяет вид контрольной точки *i*: • 00 — контрольная точка по команде; • 01 — контрольная точка по данным с обращением по записи; • 10 — не определена и не используется; • 11 — контрольная точка по данным с обращением по считыванию или записи (но не выбор команды)
LEN*i*	Для контрольной точки *i* по данным определяет длину данных: • 00 — однобайтовое поле (и все команды); • 01 — двухбайтовое поле (слово); • 10 — не определена и не используется; • 11 — четырехбайтовое поле (двойное слово)

Контрольная точка может быть локальной (в пределах одной задачи) или глобальной — в зависимости от значений битов L*i* и G*i*.

Допускается одновременное значение L*i* = G*i* = 1 или L*i* = G*i* = 0. В первом случае это эквивалентно G*i* = 1, а во втором — контрольная точка запрещена и соответствие линейного адреса из регистра DR*i* адресу команды не вызывает особого случая (но B*i* в регистре DR6 устанавливается в 1).

Аппаратные контрольные точки по командам устанавливаются путем загрузки в один из регистров DR*i* линейного адреса требуемой команды, установки в 00 соответствующих полей RW*i* и LEN*i*, установки в 1 бита L*i* и/или G*i*. После этого процессор начинает контролировать устройство предвыборки команд. Когда фиксируется равенство адреса команды и содержимого одного из "разрешенных" регистров DR*i*, хранящих контрольную точку по командам, формируется особый случай отладки, причем в DR6 устанавливается в 1 бит B*i*.

Адреса команд в регистрах DR0—DR3 должны быть 32-разрядными линейными, а не логическими *(селектор : смещение)* или физическими. Линейный адрес не зависит от страничного преобразования, поэтому контрольная точка действует даже тогда, когда целевая команда участвует в свопинге и отображается на различные адреса физической памяти.

Если контрольная точка установлена как локальная, она сбрасывается при переключении задачи, причем значение DR7 не сохраняется в TSS, поэтому при восстановлении задачи контрольные точки не возобновляются. При необходимости следует предусмотреть программное возобновление контрольных точек. Для этого можно в расширении сегмента TSS той задачи, в которой определены локальные контрольные точки, записать значения DR0—DR3 и DR7, а так же установить в TSS бит T = 1. При переходе к такой задаче вызывается обработчик особого случая, который и восстановит значения регистров DR из сегмента TSS.

Аппаратные контрольные точки по командам являются *нарушениями,* т. е. процессор включает в стек адрес команды, вызвавшей нарушение, и обработчик особого случая возвращает управление на ту же команду. Поскольку аппаратные контрольные точки по команде проверяются до выполнения самой команды, процессор должен вновь сформировать особый случай. Необходимо обойти данную команду, для этого используется флаг возобновления RF в регистре EFLAGS. Процессор автоматически устанавливает RF= 1 при возникновении любого нарушения, включая и аппаратные контрольные точки по командам. Аппаратные прерывания по входам INTR и NMI, а так же программные ловушки и аварии не воздействуют на флаг RF. Когда RF = 1, процессор игнорирует особый случай аппаратной контрольной точки по командам, а после первой же команды, которая выполнена без особых случаев, процессор сбрасывает RF.

Аппаратные контрольные точки по данным устанавливаются с помощью тех же регистров отладки DR. Процессор формирует особый случай отладки как *ловушку,* когда происходит обращение к данным по установленным в DR адресам. Разрешается совместное применение контрольных точек по командам и по данным, причем возможно произвольное назначение типа контрольной точки (поле RW*i* в регистре DR7).

Процессор контролирует выравнивание данных, если их длина, указанная в поле LEN*i*, равна слову или двойному слову. В этом случае при сравнении текущего адреса сегмента данных с адресом контрольной точки игнорируются один или два младших бита регистра DR*i*.

Конвейерная архитектура старших моделей *x*86 обеспечивает одновременную обработку нескольких команд. Случается, что контрольная точка по данным фиксируется только после выполнения нескольких следующих команд. В регистре управления отладкой DR6 предусмотрены биты, задающие локальную LE и глобальную GE "точность" определения контрольной точки по данным. Будучи установленными в 1, эти биты замедляют внутренние операции процессора таким образом, что сообщают об обращении по контролируемому адресу данных точно в тот момент, когда происходит обращение к памяти.

Биты LE, GE действуют только на контрольные точки по данным, являются общими

для всех таких точек, причем LE автоматически сбрасывается при переключении задачи, а GE может быть сброшен лишь программно.

Регистрация нескольких особых случаев

Если команда, на которую настроена контрольная точка, вызывает данные по контролируемому адресу, процессор правильно сформирует два особых случая отладки. Первый— по команде — является нарушением, и обработчик этого особого случая возвращает управление той же команде. При выполнении команды второй особый случай не возникает (см. выше назначение бита RF), зато фиксируется ловушка по данным. Причина текущего особого случая фиксируется в DR6.

7.5. Увеличение быстродействия процессора

Одним из самых распространенных способов определения производительности процессора является оценка времени T решения некоторой (тестовой) задачи. Очевидно,

$$T = \frac{N \cdot S}{f}, \qquad (7.1)$$

где N — количество выполненных при решении задачи машинных команд; S — среднее количество тактов, приходящихся на выполнение одной команды; f — тактовая частота процессора.

Если длительности различных команд (в тактах) существенно отличаются друг от друга, то более точно можно оценить значение T по выражению

$$T = \frac{1}{f} \cdot \sum_{i=1}^{N} S_i, \qquad (7.2)$$

где S_i — число тактов i-й команды.

Используются и более точные (и, соответственно, более сложные) методы оценки производительности [7, 11, 12], однако и из выражений (7.1). (7.2) видны пути увеличения производительности процессора:

- увеличение тактовой частоты (решения лежат в области технологии СБИС);
- сокращение длины программы (совершенствование технологии программирования, разработка оптимизирующих компиляторов);
- сокращение числа тактов, приходящихся на выполнение одной команды.

Последнее возможно за счет усложнения схемы процессора, при этом значительное усложнение может привести к сокращению числа тактов команды при увеличении "глубины схемы", что повлечет за собой увеличение длительности такта, так что выигрыш может обернуться проигрышем.

Магистральным путем увеличения производительности ЭВМ можно считать *параллелизм* на различных уровнях.

Существуют две основные формы параллелизма [11]:

- параллелизм на уровне процессов;
- параллелизм на уровне команд.

В первом случае над одной задачей могут одновременно работать несколько процессоров или других устройств ЭВМ.

Во втором случае параллелизм реализуется в пределах отдельных команд.

Обычно стремятся совмещать во времени процедуры обращения к памяти и обработки информации, параллельно выполнять арифметические операции сразу над несколькими (или даже всеми) разрядами операндов, одновременно выполнять несколько последовательных команд программы (разумеется, на разных стадиях) и т. п.

Уже в младшей модели семейства *x*86 — микропроцессоре 8086 предусматривалась одновременная работа двух основных устройств — *обработки данных* и *связи с*

магистралью. Подобный механизм (с модификациями) сохранился и в старших моделях семейства.

Особенно эффективным способом организации параллельных операций в компьютерной системе является *конвейерная обработка команд.*

Далее мы кратко рассмотрим некоторые из перечисленных методов увеличения производительности процессора. Более подробную информацию по этим вопросам можно найти, например, в [11, 12].

7.5.1. Конвейеры

При отсутствии конвейера процессор выполняет программу, по очереди выбирая из памяти и активизируя ее команды.

Процесс обработки команды может быть разбит, например, на следующие шаги (стадии):

- F — выборка (от англ. *fetch*) — чтение команды из памяти;
- D — декодирование (от англ. *decode*) — декодирование команды;
- A — формирование адресов (от англ. *address generate*) и выборка операндов;
- E — выполнение (от англ. *execute)*— выполнение заданной в команде операции;
- W — запись (от англ. *write)* — сохранение результата по целевому адресу.

Приведенное разбиение не является единственно возможным — в некоторых случаях рассматривают четырехстадийный командный цикл, иногда (например, для процессоров, реализующих команды над числами с плавающей запятой) — восьмистадийный и др.

Для реализации каждой из стадий командного цикла в процессоре предусмотрено соответствующее оборудование (регистры, дешифраторы, сумматоры, управляющие автоматы или их фрагменты), причем операционные элементы разных стадий обычно слабо пересекаются между собой. Поэтому когда очередная команда завершает действия на одной стадии, например F, и переходит на следующую — D, то оборудование стадии F "простаивает" и может быть использовано для чтения следующей команды. Таким образом, очередная команда может начинать выполнение, не дожидаясь окончания командного цикла предыдущей команды.

При рассмотрении пятистадийного командного цикла одновременно на разных стадиях может выполняться до пяти команд. Организация *пятистадийного конвейера* потребует дублирования некоторых операционных элементов на разных стадиях (например, регистра команд PC) и усложнение схемы управления, однако игра стоит свеч.

Очевидно, очередная команда может перейти с одной стадии командного цикла на другую при выполнении двух условий:

- действия команды на текущей стадии завершены;
- предыдущая команда освободила оборудование следующей стадии.

При условии, что каждая стадия выполняется в любой команде одинаковое количество тактов (например, один), конвейер работает идеально, и одновременно всегда выполняются пять команд (для пятистадийного конвейера).

Для большинства процессоров т. н. CISC[1]-архитектуры (к ним относятся, в частности, процессоры семейства *x*86) такая идеальная ситуация складывается далеко не всегда.

Действительно, команда, извлекаемая из памяти на стадии F, может иметь разную длину и, следовательно, извлекаться из памяти за разное число машинных циклов.

В зависимости от заданного в команде способа адресации операндов время выполнения стадии A может быть существенно различным (сравните непосредственную

[1] Complex Instruction Set Computer — компьютер е полным набором команд.

адресацию и косвенно-автоинкрементную). Расположение адресуемых операндов (и размещение результата) в памяти разного уровня так же существенно влияет на время реализации стадии A (и стадии W).

Наконец, на стадии E время выполнения операции зависит не только от типа операции (короткие— сложение, конъюнкция,..., длинные— умножение, деление), но даже иногда и от значений операндов.

Учитывая отмеченные выше обстоятельства, можно сказать, что конвейеры процессоров с классической CISC-архитектурой редко работают "на полную мощность", находясь значительную часть времени в ожидании завершения "длинных" операций.

Желание увеличить производительность конвейеров привело к появлению процессоров т. н. RISC[1]-архитектуры, из систем команд которых были исключены все факторы, тормозящие реализацию командного цикла — длинные команды, сложные способы адресации, размещение операндов в ОЗУ. К особенностям RISC-архитектуры можно отнести:

- форматы всех команд имеют одинаковую длину, в крайнем случае, разнообразие длин форматов ограничивается двумя вариантами;
- все операции выполняются за одинаковый промежуток времени (обычно 1 или 2 такта);
- операнды всех арифметических и логических операций располагаются только в регистрах, к оперативной памяти обращаются только команды загрузки и сохранения;
- сверхоперативная память представлена большим числом регистров (32—256).

Реализация этих особенностей, с одной стороны, позволяет приблизить работу конвейера к идеальной, с другой стороны — существенно ограничивает возможности системы команд процессора. Действительно, из системы операций исключаются "длинные" операции — умножение, деление, операции над числами с плавающей запятой и др. Исключаются сложные (и эффективные) способы адресации, например, автоиндексные. Это приводит к значительному увеличению длины программ, увеличению времени на выборку команд из памяти, при этом среднее число команд, выполняемых в единицу времени, в RISC-процессорах значительно больше, чем в CISC-процессорах.

По мере совершенствования интегральной технологии появилась возможность ценой значительных аппаратных затрат (теперь разработчики уже могли их себе позволить) резко сократить время выполнения "длинных" операций. например, за счет реализации матричного умножителя, табличной арифметики и др. В этом случае длинные операции можно включать в систему команд RISC-процессоров, не нарушая принципов их организации, но увеличивая эффективность системы команд.

В настоящее время понятия "RISC-" и "CISC-архитектура" скорее являются обозначением некоторых принципов проектирования, но не характеристиками конкретных процессоров. Современные процессоры, как правило, реализованы по "гибридному" принципу: содержат ядро RISC, которое выполняет простые и самые распространенные команды за один такт на стадию конвейера, а сложные команды интерпретируются как некая последовательность простых (на уровне микрокоманд). Пользователь же в этом случае имеет дело с системой команд привычной CISC-архитектуры, а внутренние вопросы реализации командного цикла его, как правило, не интересуют.

При реализации конвейеров возникает еще одна проблема, связанная с его оптимальной загрузкой — команды условной передачи управления (в среднем каждая 5—6-я команда программы). Действительно, когда такая команда передается со стадии F на стадию D, на стадию F надо ставить следующую команду, но какую? Условие перехода будет проверено лишь на стадии E, тогда же определится адрес следующей команды. Здесь возможны два пути решения:

- приостановить загрузку конвейера до завершения командой перехода стадии E;
- загрузить конвейер "наугад" командой по одному из двух возможных адресов, а

[1] Reduccd Instruction Set Computer — компьютер с сокращенным набором команд.

на стадии E проверить правильность выбора и, если он оказался неверным — очистить весь конвейер и начать загрузку заново по правильному адресу.

Второй путь представляется предпочтительным, т. к. конвейер не останавливается, и (в среднем) в половине случаев мы избежим потери времени на перезагрузку. Результаты работы конвейера будут еще лучше, если мы научимся правильно предсказывать адрес перехода, чтобы вероятность угадывания адреса приближалась к 1.

В современных процессорах часто предусматривают специальные аппаратные блоки предсказания переходов. В разных процессорах реализуются различные алгоритмы предсказания, основанные на анализе результатов выполнения предыдущих команд переходов [12].

7.5.2. Динамический параллелизм

Один конвейер хорошо, а два лучше? Почему бы, если позволяют ресурсы интегральной технологии, не построить два конвейера и ставить на них одновременно пару команд программы. В процессоре Pentium именно так и сделали — предусмотрели два 5-уровневых конвейера, которые могли работать одновременно и выполнять две целочисленные команды за машинный такт.

Однако возможность одновременной постановки на два разных конвейера пары последовательных команд программы ограничивается рядом условий. Очевидно, что нельзя ставить на разные конвейеры две последовательные команды, если вторая использует в качестве операнда результат работы первой или, во всяком случае, необходимо гарантировать, что к началу стадии выборки операндов второй команды первая (на другом конвейере) уже завершит стадию размещения результата. Существуют и другие ограничения, которые определяют т. н. условия "спаривания" последовательных команд (pairing), позволяющие размещать их одновременно на разных конвейерах.

В процессоре Pentium два конвейера не являются равноправными. Один из них (U) может принять любую команду, а другой (V) — только удовлетворяющую условиям "спаривания" (довольно сложным) с командой, поставленной на U. Если эти условия не соблюдаются, следующая команда так же помещается на U-конвейер, а V-конвейер пропускает такт. Некоторые команды могут появляться только на U-конвейере.

Разумеется, производительность двухконвейерного процессора, при прочих равных условиях, превышает производительность одноконвейерного, но далеко не в два раза. Эффективность во многом определяется, насколько часто будут встречаться в программе пары последовательных команд, допускающих "спаривание".

Очевидно, движение в направлении увеличения в процессоре числа конвейеров, работающих по описанным выше принципам, бесперспективно. Условия "страивания" (если можно так сказать) и т. д. будут настолько сложны, что редко будут выполняться.

Следующим шагом на пути увеличения производительности было решение, которое принято называть *динамическим параллелизмом,* а процессоры, реализующие этот принцип, называют *суперскалярными.* Рассмотрим фрагмент программы, написанный на некотором условном языке:

```
MOV R1, R4
ADD R1, @R0
MOV R5, R6
SUB R5, R7
CLR R6
. . .
```

Вторая команда этого фрагмента использует в качестве операнда содержимое ячейки памяти (косвенно-регистровая адресация) и, следовательно, попав на конвейер, будет "тормозить" его[1] на стадии A. В то же время следующие три команды этого фрагмента никак не связаны с результатами работы второй команды, выполняют действия только над регистрами и могли бы выполняться еще до завершения предыдущей команды, однако в этом случае пришлось бы изменить порядок выполнения команд, определенный

[1] Здесь мы не обсуждаем возможности кэш-памяти

программой, чего конвейер не предусматривает.

Суперскалярная архитектура процессора предполагает, что команды (на некотором ограниченном участке программы) могут выполняться не только в порядке их размещения в программе, но и по мере возможности их выполнения независимо от порядка следования. Возможность выполнения определяется, во-первых, отсутствием зависимостей от ранее расположенных, но еще не завершенных команд, во-вторых, наличием свободных ресурсов процессора, необходимых для выполнения команды.

Одним из первых микропроцессоров, реализующих механизм динамического параллелизма, был процессор Pentium Pro (Pentium II) фирмы Intel (рис. 7.15).

На кристалле процессора размещаются два блока кэш-памяти первого уровня, в одном из которых (кэш-С) размещается программа, а в другом (кэш-D) — данные.

Рис. 7.15. Структура процессора Pentium Pro

Устройство выборки/декодирования выбирает очередную команду из кэш-С (в порядке их размещения в программе), при необходимости заменяет сложные команды на последовательность микрокоманд, снабжает каждую команду полем признаков (тегом) и помещает в специальным образом организованную память — *пул команд.*

Устройство диспетчирования постоянно анализирует, с одной стороны, содержимое пула команд и выявляет команды, готовые к выполнению на какой-нибудь стадии, с другой стороны — свободные в данный момент операционные устройства. При совпадении "желания" (команда завершила предыдущую стадию и готова к выполнению следующей) и "возможностей" (свободны соответствующие ресурсы) устройство диспетчирования отправляет команду на выполнение независимо от порядка поступления команд в пул. После завершения обработки на очередной стадии команда возвращается в пул с соответствующей пометкой в поле тега.

Устройство отката размещает результаты выполнения команд по адресам назначения. Оно просматривает содержимое пула команд, отыскивает команды, завершившие работу, и извлекает их из пула с размещением результата в строгом соответствии с порядком расположения команд в программе.

Небольшое количество регистров в архитектуре процессоров Intel приводит к интенсивному использованию каждого из них и, как следствие, к возникновению множества мнимых зависимостей между командами, использующими один и тот же регистр. Поэтому, чтобы исключить задержку в выполнении команд из-за мнимых зависимостей, устройство диспетчирования/выполнения работает с дублями регистров,

находящимися в пуле команд (одному регистру может соответствовать несколько дублей).

Реальный набор регистров контролируется устройством отката, и результаты выполнения команд отражаются на состоянии вычислительной системы только после того, как выполненная команда удаляется из пула команд в соответствии с истинным порядком команд в программе.

Таким образом, принятая в Pentium Pro технология динамического выполнения может быть описана как оптимальное выполнение программы, основанное на предсказании будущих переходов, анализе графа потоков данных с целью выбора наилучшего порядка исполнения команд и на опережающем выполнении команд в выбранном оптимальном порядке. Однако следует иметь в виду, что процессор оптимизирует выполнение только ограниченного участка программы, который в текущий момент располагается в пуле.

Суперскалярная архитектура предполагает наличие на кристалле процессора нескольких параллельно работающих операционных устройств (в т. ч. и нескольких одинаковых). Так, например, RISC-процессор PowerPC содержит шесть параллельно работающих исполнительных устройств: блок предсказания ветвлений, два устройства для выполнения простых целочисленных операций (сложение, вычитание, сравнение, сдвиги, логические операции), одно устройство для выполнения сложных целочисленных операций (умножение, деление), устройство обработки чисел с плавающей запятой и блок обращения к внешней памяти. При этом обеспечивается одновременное выполнение четырех команд.

Все операции обработки данных выполняются с регистровой адресацией. При этом для хранения целочисленных операндов используется блок, включающий тридцать два 32-разрядных регистра, а для хранения операндов с плавающей запятой — блок из тридцати двух 64-разрядных регистров.

Выборка данных из памяти производится только командами пересылки, которые выполняются блоком обращения к памяти и осуществляют загрузку данных в регистры или запись их содержимого в память.

При параллельной работе исполнительных устройств возможно их одновременное обращение к одним регистрам. Чтобы избежать ошибок, возникающих при этом в случае записи нового содержимого до того, как другим устройством будет считано предыдущее, введены буферные регистры— 12 для целочисленных регистров и 8 — для регистров с плавающей запятой. Эти регистры служат для промежуточного хранения операндов, дублируя основные регистры блоков, используемые при выполнении текущих операций. После завершения операций производится перезапись полученных результатов в основные регистры (обратная запись).

7.5.3. VLIW-архитектура

Для реализации динамического параллелизма в процессорах с традиционной системой команд и способами компиляции программного кода требуются весьма сложные схемы организации пула, планировщики, схемы "отката" и др. Процессоры такой архитектуры имеют несколько операционных блоков различного, а иногда и одинакового назначения, которые могут работать параллельно, например, 1—2 блока вычисления адресов, 2—3 блока АЛУ для чисел с фиксированной запятой, блок обработки чисел с плавающей запятой, блок размещения результата, блок предсказания переходов и др.

Поскольку процессор может планировать и формировать последовательность выполнения команд на ограниченном (размером пула) участке программы, то для эффективной загрузки операционных блоков требуются не только сложные и эффективные процедуры планирования, но и некоторое "везение" — хорошо, если в пул загружены команды, для выполнения которых нужны различные операционные блоки, а если нет?

Один из путей дальнейшего повышения эффективности подобных систем лежит в области разработки *специальных компиляторов,* которые упаковывают несколько простых команд в "очень длинное командное слово" (VLIW — аббревиатура от Very Long Instruction Word) таким образом, чтобы в одной "очень длинной команде" можно было

использовать все существующие в процессоре операционные блоки. В этом случае командное слово соответствует набору функциональных устройств.

VLIW-архитектуру можно рассматривать как статическую суперскалярную, поскольку распараллеливание кода производится на этапе компиляции, а не динамически во время исполнения, т. е. в машинном коде VLIW присутствует явный параллелизм.

Одним из примеров воплощения идей VLIW может служить предложенная Intel в содружестве с HP концепция 64-разрядной архитектуры микропроцессора IA-64 (Intel 64-bit Architecture, 64-разрядная архитектура Intel). Для ее обозначения использована аббревиатура EPIC (Explicitly Parallel Instruction Computing, вычисления с явным параллелизмом команд).

Процессор, разработанный на базе этой концепции, отличающийся следующими особенностями:

- □ большое количество регистров: 128 64-разрядных регистров общего назначения (целочисленных), плюс 128 80-разрядных регистров арифметики плавающей запятой, плюс 64 1-разрядных предикатных регистра;
- □ масштабируемость архитектуры до большого количества функциональных устройств. Это свойство представители фирм Intel и HP называют "наследственно масштабируемым набором команд" (inherently scaleable instruction set);
- □ явный параллелизм в машинном коде: поиск зависимостей между командами производит не процессор, а компилятор;
- □ предикация (predication): команды из разных ветвей условного ветвления снабжаются предикатными полями (полями условий) и запускаются на выполнение параллельно;
- □ загрузка по предположению (speculative loading): данные из медленной основной памяти загружаются заранее.

Формат команды 1A-64 включает код операции, три 7-разрядных поля операндов — 1 приемник и 2 источника (операндами могут быть только регистры), особые поля для вещественной и целой арифметики, 6-разрядное предикатное поле.

Команды IA-64 упаковываются (группируются) компилятором в "связку" длиною в 128 разрядов. Связка содержит 3 команды и шаблон, в котором указаны зависимости между командами в связке (можно ли с командой k_1 запустить параллельно k_2, или же k_2 должна выполниться только после k_1), а также между другими связками (можно ли с командой k_3 из связки c_1 запустить параллельно команду k_4 из связки c_2).

Одна такая связка, состоящая из трех команд, соответствует набору из трех функциональных устройств процессора. Процессоры IA-64 могут содержать разное количество таких блоков, оставаясь при этом совместимыми по коду. Ведь благодаря тому, что в шаблоне указана зависимость и между связками, процессору с N одинаковыми блоками из трех функциональных устройств будет соответствовать командное слово из $N \times 3$ команд (N связок). Таким образом, обеспечивается *масштабируемость* IA-64.

Предикация — способ обработки условных ветвлений. Суть этого способа — компилятор указывает, что обе ветви выполняются на процессоре параллельно. Если в исходной программе встречается условное ветвление, то команды из разных ветвей помечаются различными предикатными регистрами (команды имеют для этого предикатные поля), далее они выполняются совместно, но их результаты не записываются, пока значения предикатных регистров не определены. Когда, наконец, вычисляется значение условия ветвления, предикатный регистр, соответствующий "правильной" ветви, устанавливается в 1, а другой— в 0. Перед записью результатов процессор будет проверять предикатное поле и записывать результаты только тех команд, предикатное поле которых содержит предикатный регистр, установленный в 1.

Загрузка по предположению — это механизм, который предназначен снизить простои процессора, связанные с ожиданием выполнения команд загрузки из относительно медленной основной памяти. Компилятор перемещает команды загрузки данных из памяти

так, чтобы они выполнились как можно раньше. Следовательно, когда данные из памяти понадобятся какой-либо команде, процессор не будет простаивать. Перемещенные таким образом команды называются командами загрузки по предположению и помечаются особым образом. Непосредственно перед командой, использующей загружаемые по предположению данные, компилятор вставляет команду проверки предположения.

Выводы

Основная особенность EPIC — распараллеливанием потока команд занимается компилятор, а не процессор.

Достоинства данного подхода:

- □ упрощается архитектура процессора; вместо распараллеливающей логики на EPIC-процессоре можно разместить больше регистров, функциональных устройств;
- □ процессор не тратит время на анализ потока команд на предмет возможности их параллельного выполнения— эту работу уже выполнил компилятор;
- □ возможности процессора по анализу программы во время выполнения ограничены сравнительно небольшим участком программы, тогда как компилятор способен произвести анализ по всей программе;
- □ если некоторая программа должна запускаться многократно, выгоднее распараллелить ее один раз (при компиляции), а не каждый раз, когда она исполняется на процессоре.

Недостатки:

- □ компилятор производит статический анализ программы, раз и навсегда планируя вычисления. Однако даже при небольшом изменении начальных данных путь выполнения программы может сколь угодно сильно измениться;
- □ серьезно увеличивается сложность компиляторов. Значит, увеличится число ошибок в них, время компиляции;
- □ еще более увеличивается сложность отладки, т. к. отлаживать придется оптимизированный параллельный код;
- □ производительность EPIC будет всецело зависеть от качества компилятора;
- □ проблематичным пока видится преемственность программного обеспечения при переходе на новые поколения микропроцессоров (скомпилированный код очень сильно "привязан" к конкретной архитектуре процессора).

Тем не менее, представители Intel и HP называют EPIC концепцией следующего поколения и противопоставляют ее CISC и RISC. По мнению Intel, традиционные архитектуры имеют фундаментальные свойства, ограничивающие производительность.

Производители RISC-процессоров не разделяют подобного пессимизма. Кстати, в 1980-х годах, когда возникла концепция RISC, прозвучало много заявлений, что концепция CISC устарела, имеет фундаментальные свойства, ограничивающие производительность. Но процессоры, причисляемые к CISC (например, семейство *x*86), широко используются до сих пор, их производительность растет.

В действительности же, все эти аббревиатуры — CISC, RISC, VLIW, EPIC — обозначают только *идеализированные концепции.* Реальные современные микропроцессоры трудно подвести исключительно под какой-либо из перечисленных выше классов. Просто в наиболее совершенных современных процессорах заложено большое число удачных идей, использующих многие рассмотренные здесь концепции.

7.6. Однокристальные микроЭВМ

При рассмотрении процессов эволюции современных процессоров и ЭВМ, прежде всего, обращают внимание на увеличение производительности системы (быстродействие процессора). Некоторые из путей повышения быстродействия были обсуждены в

предыдущих разделах, другие, реализующие параллелизм на уровне процессов (мультипроцессоры, векторные, массивно-параллельные, компьютерные сети [11, 12], нейроматричные процессоры, и др.), выходят за рамки настоящей книги.

Однако всегда существовали и существуют задачи, для решения которых вовсе не требуется высокое быстродействие процессора и мощная система команд. На первый план здесь выступают другие характеристики: стоимость, надежность, малые габариты, способность работать в экстремальных климатических условиях, при значительных перепадах питающего напряжения и на фоне высокого уровня электромагнитных помех, с автономным питанием.

Для получения таких характеристик растущие возможности интегральной технологии можно использовать не для увеличения разрядности и вычислительной мощности процессора, а размещая на кристалле, наряду с простым (на первых порах — восьмиразрядным) процессором, все другие устройства, входящие в состав ЭВМ: регистры, различные типы памяти (на первых порах — небольшого объема), тактовый генератор, порты параллельного и последовательного обмена, различные внешние устройства (таймеры, АЦП и др.).

При этом получается полностью "самодостаточный" кристалл БИС (СБИС) *однокристальной микроЭВМ* (некоторые авторы используют термин *микроконтроллер,* учитывая, что основная сфера применения подобных изделий — управляющие системы, работающие в реальном времени).

В настоящее время многие фирмы (Motorola, Intel, Microchip, Zilog и др.) выпускают широкую номенклатуру подобных однокристальных микроЭВМ (ОМЭВМ), отличающихся разрядностью, системой команд, типами и объемом памяти, составом и характеристиками внешних устройств. Большинство ОМЭВМ выпускается с 8-разрядным процессором, но на рынке присутствуют и 16- и даже 32-разрядные ОМЭВМ.

Базовые принципы архитектуры ОМЭВМ можно проиллюстрировать на примере 8-разрядных ОМЭВМ. Далее кратко отметим некоторые особенности этих контроллеров. Подробнее архитектура ряда ОМЭВМ и примеры их применения описаны в [4].

Контроллеры разных фирм, несмотря на кажущиеся различия, имеют много общих черт, определяющих тенденции развития современных ОМЭВМ малой и средней производительности. Попробуем отметить некоторые из них.

- Все без исключения контроллеры имеют встроенные тактовые генераторы; для запуска большинства из них используют одну из четырех возможных внешних цепей: источник внешних тактовых импульсов, кварцевый резонатор, LC-цепь, RC-цепь. Последние две можно использовать лишь в системах, где точностью временных привязок можно пренебречь. Большинство контроллеров имеют в своем составе динамические элементы памяти, что ограничивает допустимую тактовую частоту не только сверху, но и снизу (обычно— до 1 МГц). ОМЭВМ Motorola используют на кристалле только статические элементы памяти, что позволяет работать на произвольно низких системных тактовых частотах. Снижение тактовой частоты контроллера целесообразно при управлении инерционными объектами, если необходимо отслеживать достаточно длительные временные задержки (от долей секунды до десятков секунд).
- Процессоры большинства ОМЭВМ реализуют классическую ("интелловскую") систему команд, включающую одно- и двухадресные команды с операциями над ячейками памяти и регистрами, с использованием разнообразных способов адресации (прямая, регистровая, косвенно-регистровая, индексная, непосредственная). Предусмотрен широкий выбор команд передачи управления, в т. ч. вызовы подпрограмм. Во многих контроллерах реализовано умножение и деление. Процессоры семейств MCS-51 и MC68HC11 имеют развитую систему операций с битами.
- Память большинства ОМЭВМ организована по *гарвардской архитектуре*, предполагающей различные адресные пространства для памяти программ и памяти данных (исключение составляют лишь контроллеры фирмы Motorola, традиционно поддерживающие единое адресное пространство). Такое решение

снижает риск потери управления при выполнении программы, но ограничивает возможности по распределению ресурсов памяти системы.

- На кристалле могут располагаться различные типы памяти: масочное или однократно программируемое ПЗУ, ППЗУ со стиранием ультрафиолетовым излучением, электрически стираемое ППЗУ (флэш-память), ОЗУ (регистры).
- Многие параллельные порты контроллеров допускают двунаправленный обмен, часто возможно независимое программирование линий порта на ввод или вывод. Допускается выбор типа выхода — обычный ТТЛ-вывод или вывод с открытым коллектором (стоком). Большинство линий портов имеют одну или несколько альтернативных функций, выбор которых осуществляется программно.
- Важным элементом ОМЭВМ являются системы контроля времени, представленные различными счетчиками с управляемыми коэффициентами пересчета и возможностью выбора источника счетных импульсов: тактовый генератор — в режиме таймера и внешний вывод — в режиме счетчика внешних событий. Более мощные контроллеры имеют в своем составе таймерные системы, включающие несколько модулей сравнения, автозахвата, ШИМ (широтно-импульсная модуляция) и др.
- Последовательные каналы включаются обычно в старшие модели семейств. Предусматриваются либо универсальные синхронно-асинхронные приемопередатчики, программируемые на работу в определенном режиме, либо отдельные блоки SCI (UART) — асинхронный приемопередатчик и SPI— синхронный периферийный интерфейс, работающие независимо друг от друга.
- Средства работы с аналоговыми сигналами включают в себя компараторы, многоканальные 8-разрядные аналого-цифровые и реже — цифроаналоговые преобразователи.
- Подсистема прерываний включает несколько внешних радиальных входов и большое число внутренних прерываний, которые генерируются в системе контроля времени, АЦП, последовательных и параллельных каналах.

В современных микроконтроллерах предусматривается широкий набор специальных средств, повышающих надежность и эффективность функционирования систем управления. Прежде всего, это т. н. *сторожевой таймер* WDT (Watch-Dog Timer), который предотвращает аварийное зацикливание программы. В некоторых контроллерах предусматриваются специальные схемы, следящие за правильной работой тактового генератора. В системах с автономным питанием большое значение имеют средства энергосбережения. Большинство контроллеров программно можно переводить в специальные *режимы пониженного энергопотребления* (в состоянии ожидания) с остановкой основных подсистем (в т. ч. иногда и тактового генератора), но с сохранением контекста задачи. Выход из таких режимов возможен по разрешенному прерыванию или по сбросу (часто системный сброс реализуется как одно из прерываний).

16- и 32-разрядные ОМЭВМ построены обычно по модульному принципу. На внутрикристальный системный интерфейс могут подключаться процессоры различной вычислительной мощности, различные модули памяти, контроллеры параллельного и последовательного обмена, модули АЦП и ЦАП, таймерные сопроцессоры и сопроцессоры ввода/вывода. В зависимости от требований решаемой задачи пользователь может выбрать подходящую конфигурацию кристалла ОМЭВМ. В качестве примера коротко рассмотрим семейство 32-разрядных микроконтроллеров фирмы Motorola.

Отличительной особенностью ОМЭВМ фирмы Motorola является модульная технология построения многофункциональных устройств на одном кристалле. Определен стандарт внутрикристальной межмодульной шины IMB и множество наборов системных модулей, из которых собирается ОМЭВМ:

- *процессорные ядра* (CPU), включающие 16- и 32-разрядные микропроцессоры различной вычислительной мощности, но относящиеся к одному семейству;
- *системные интеграционные модули* (SIM), контролирующие внешнюю шину,

запуск, инициализацию и конфигурацию микроконтроллера. Они включают в себя тактовый генератор, блок системной конфигурации и защиты, блок тестирования и интерфейс с внешней шиной. Модули отличаются друг от друга, главным образом, разрядностью шин адреса и данных;

- *модули памяти*, отличающиеся типом и объемом запоминающих устройств: ОЗУ, ПЗУ(включая однократно программируемое пользователем), ЭСППЗУ;
- *модули последовательных портов* включают различные варианты синхронных и асинхронных программируемых контроллеров последовательного обмена. Предусмотрены модули, содержащие несколько различных интерфейсов;
- таймерные системы представлены различными вариантами таймерных сопроцессоров (TPU). TPU способен независимо от процессора выполнять как простые, так и сложные таймерные функции, его можно считать отдельным специализированным микропроцессором, который осуществляет две основные операции— проверку на совпадение (от англ. *match* — сравнение) и сохранение значения счетчика-таймера в момент изменения состояния какого-либо входа (от англ. capture — захват) над одним операндом — временем. Выполнение любой из них называется событием. Обслуживание событий сопроцессором замещает обработку прерываний центральным процессором. С помощью двух основных операций TPU может реализовать значительный набор функций: счет внешних событий, захват по внешнему входу, сравнение временных интервалов, широтно-импульсную модуляцию, измерение периода входного сигнала, программируемую генерацию импульсов и многие другие, программируемые пользователем. TPU, естественно, имеет собственную систему команд, его программы хранятся на общей памяти системы или в специализированной памяти программ TPU;
- *системы аналогового ввода* реализованы на различных вариантах АЦП (ADC), отличающихся разрядностью получаемого кода и способом (а следовательно, и временем) аналого-цифрового преобразования.

На рис. 7.16 приведена структура 32-разрядного микроконтроллера MC68332.

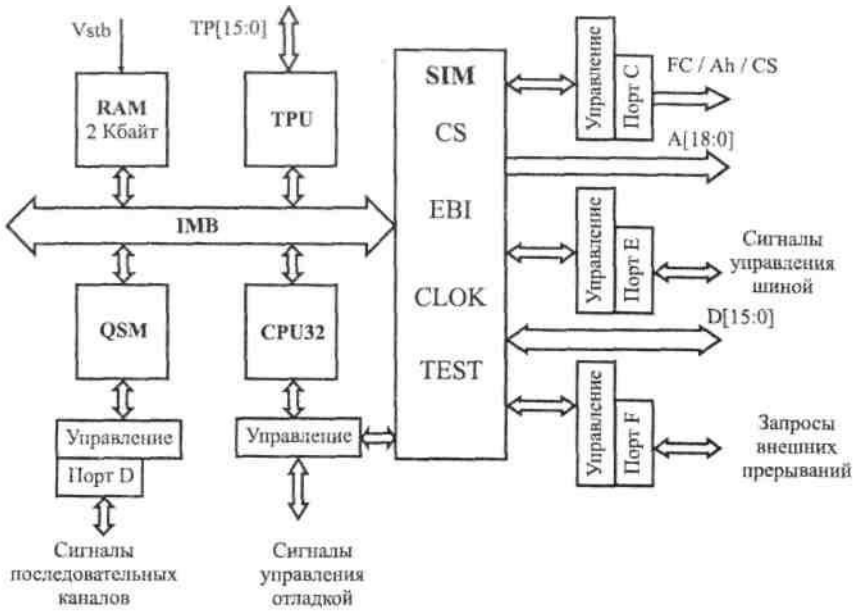


Рис. 7.16. Структура MC68332

Кристалл микроконтроллера содержит следующие модули:

- CPU32 — 32-разрядный центральный процессор семейства MC68000;
- RAM — 2 Кбайт ОЗУ с независимым питанием для размещения программ и данных (напомним, что большинство изделий Motorola поддерживают архитектуру фон Неймана с единым адресным пространством программ и

данных);

- QSM — подсистемы последовательного ввода/вывода, включающие универсальный асинхронный приемопередатчик (UART) и дуплексный синхронный последовательный интерфейс (SPI);
- TPU — шестнадцатиканальный таймерный сопроцессор;
- SIM — системный интеграционный модуль.

Внешняя память может при необходимости подключаться к контроллеру к шинам *адреса* A[18:0], *данных* D[l 5:0],*управления шиной.*

Линии порта C могут использоваться для выдачи функционального кода, идентифицирующего состояние процессора и адресное пространство текущего цикла шины (FC[2:0]), старших разрядов адреса (Ah) в режиме расширенного адресного пространства и сигналов выбора кристалла (CS), разрешающих работу периферийных устройств по запрограммированным адресам.

Порт F принимает запросы внешних прерываний, линии порта D могут использоваться для передачи сигналов последовательных интерфейсов.

Группа линий TP[15:0]— входы/выходы каналов таймерного сопроцессора. На вход Vstb может подаваться независимое питание для сохранения информации в ОЗУ при отключении основного питания.