

Министерство образования и науки Российской Федерации

Ярославский государственный университет

им. П. Г. Демидова

М. В. Лоханин

Архитектура современного компьютера

Учебное пособие

Рекомендовано

*Научно-методическим советом университета
для студентов, обучающихся по специальности
Электроника и микроэлектроника*

Ярославль 2011

УДК 004.38
ББК 3973.2_я73
Л81

*Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2011 года*

Рецензенты:

кандидат технических наук А. Б. Силантьев
Ярославский филиал Физико-технологического института РАН

Л81 **Лоханин, М. В.** Архитектура современного компьютера: учебное пособие / М. В. Лоханин; Яросл. гос. ун-т. им. П. Г. Демидова. – Ярославль: ЯрГУ, 2011. – 96 с.

ISBN 978-5-8397-0854-9

Учебное пособие подготовлено в соответствии с Государственным образовательным стандартом высшего профессионального образования и предназначено для студентов, обучающихся по специальности 210100.62 Электроника и нанoeлектроника (дисциплина “Архитектура классического компьютера”, блок Б3), очной формы обучения.

Рис. 37. Библиогр.: 14 назв.

УДК 004.38
ББК 3973.2_я73

ISBN 978-5-8397-0854-9

© Ярославский
государственный
университет
им. П. Г. Демидова, 2011

Оглавление

Введение	4
1. Дополнительные сведения по цифровой электронике	5
1.1. Элементы арифметико-логического устройства	5
1.2. Интерфейсные схемы	14
1.3. Элементы памяти	17
1.4. Кэширование памяти	22
2. Информационная магистраль	27
2.1. Общие принципы	27
2.2. ISA и её расширения	29
2.3. PCI и PCI Express	33
2.4. Шины расширения в других архитектурах	41
3. Процессор	44
3.1. Архитектура Фон–Неймана	44
3.2. Суперскалярный процессор	47
3.3. Процессоры IA-32 и Intel 64	50
3.4. Программная модель процессора	57
3.5. Режимы работы процессора	66
3.6. Адресация.	69
3.7. Вызовы подпрограмм, исключения и прерывания	75
3.8. Набор инструкций	78
3.9. Типы данных и инструкции FPU	79
3.10. SSE	83
4. Диск	87
4.1. Физические принципы	87
4.2. Кодирование информации на диске	88
4.3. Интерфейсы ATA и SATA	91
Список литературы	96

Введение

Информационные машины, или, как их чаще называют, компьютеры, занимают в современной жизни, науке, технике, бизнесе, развлечениях огромное место. Практически ни одно человеческое занятие не обходится без компьютеров. Знакомство с компьютером, как правило, обозначает умение использовать какое-то количество готовых программ, реже – умение написать свою программу, но сам компьютер интересует пользователя, а часто и программиста только с точки зрения стоимости. Такое положение дел абсолютно правильно, если речь не идет о технических специалистах и профессиональных программистах. В этом случае знание принципов функционирования и архитектуры компьютера необходимо потому, что такой специалист должен участвовать в разработке новой электронной техники, понимать, где и как используются микроэлектронные изделия, уметь написать программу, которая будет работать эффективно, понимать, что может компьютер и, наконец, понимать, чего компьютер не может.

Предполагается, что читатель знаком с логическими функциями, логическими вентилями, задержками, гонками и тактированием цифровых устройств, с функциями дешифраторов, мультиплексоров, триггеров, счетчиков, регистров, имеет минимальную практику написания и отладки программ на языках высокого уровня.

В первой главе рассматриваются несколько цифровых устройств, специфичных для вычислительной техники. В остальных главах реализация устройств на уровне вентилях не рассматривается – речь идет о принципах функционирования.

Безусловно, мы будем пытаться изложить то общее, что присуще любым компьютерным архитектурам, – начиная от сверхмощных вычислительных кластеров до простейших контроллеров, но основное внимание будет уделено архитектуре IBM PC, базирующейся на процессорах Intel и AMD разных поколений.

Глава 1.

Дополнительные сведения по цифровой электронике

1.1. Элементы арифметико-логического устройства

Сумматор. Сложение чисел – наверное, самая главная арифметическая операция, к которой сводятся многие сложные вычисления. Пусть мы вычисляем $z = x + y$, каждое из этих чисел имеет некоторое двоичное представление $x = x_{n-1} \dots x_1 x_0$, $y = y_{n-1} \dots y_1 y_0$ и $z = z_{n-1} \dots z_1 z_0$. Ясно, что результат вычислений, т. е. цифра, которую мы должны поставить в i -м разряде суммы (z_i) и перенос в следующий $i + 1$ разряд (C_i), зависит от цифр в соответствующих разрядах слагаемых (x_i, y_i) и от переноса из предыдущего $i - 1$ разряда (c_i). Следовательно, мы должны рассматривать цифру в сумме и перенос в следующий разряд как логические функции трех переменных $z_i = z_i(x_i, y_i, c_i)$ и $C_i = C_i(x_i, y_i, c_i)$. Таблица истинности для названных функций приведена в табл. 1.1.

с	х	у	z	С	с	х	у	z	С
0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1
0	1	1	0	1	1	1	1	1	1

Таблица 1.1. Таблица истинности для функций $z = z(x, y, c)$ и $C = C(x, y, c)$

Упражнение 1.1. *Просмотрите таблицу истинности 1.1 для функций $z = z(x, y, c)$ и $C = C(x, y, c)$ (индекс i опущен) и убедитесь, что в ней все “как учили в школе”.*

Замечание об обозначениях. Математический стиль обозначений логических функций громоздок и трудночитаем. Мы будем использовать обозначения, принятые в цифровой электронике. Инверсию будем обозначать \bar{x} , логическую сумму $x + y$, логическое произведение $x \cdot y$ или просто xy , исключающее или $x \oplus y$. На схемах инверсию символизирует кружок там, где вывод соединяется с прямоугольником, символизирующим логический вентиль, логическую сумму – символ 1, логическое произведение – & , исключающее или – \oplus .

Упражнение 1.2. По таблице истинности 1.1 запишите ДНФ для функций $z = z(x, y, c)$ и $C = C(x, y, c)$, упростите её и покажите что $z = x \oplus y \oplus c$ и $C = xy + c(x \oplus y)$.

На рисунке 1.1 показана схема одноразрядного сумматора, реализованная на элементах трех типов (И, ИЛИ, исключающее ИЛИ). В реальной электронике сумматор (как и всё остальное) состоит только из элементов И-НЕ (соответствующая схема громоздка и малопонятна). Справа показано обозначение одноразрядного сумматора как блока. На этом обозначении изменено направление распространения сигнала. Это нужно, чтобы в многоразрядных схемах старшие разряды оказывались слева от младших, как принято в нотации чисел.

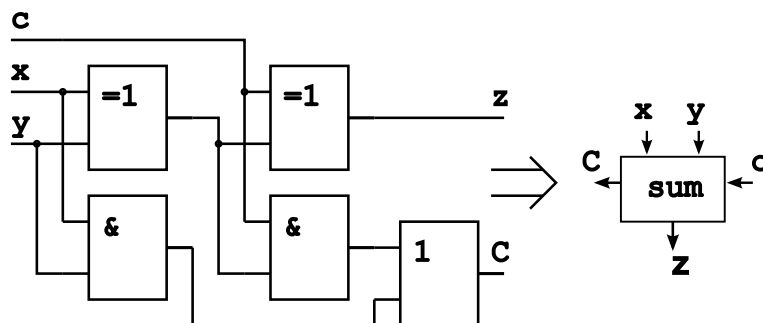


Рис. 1.1. Одноразрядный сумматор

Упражнение 1.3. Запишите промежуточные функции в схеме на рисунке 1.1 и убедитесь, что на выходах действительно $z = z(x, y, c)$ и $C = C(x, y, c)$.

Для сложения многоразрядных чисел такие сумматоры, очевидно, надо поставить так, чтобы перенос из предыдущего разряда приходил в следующий, т. е. $c_{i+1} = C_i$ для $i > 0$. Соответствующая схема для четырех разрядов показана на рисунке 1.2. На входе самого младшего разряда (least significant bit – LSB) необходимо установить ноль. Такой сумматор называется последовательным.

Куда “уходит” перенос из самого старшего разряда (most significant bit – MSB)? В нашем случае это C_3 . Перенос из MSB означает, что сумма не может быть представлена числом данной разрядности. Эта ситуация называется переполнением (overflow) и является ошибкой. В процессорах подобные исключительные ситуации после выполнения большинства инструкций запоминаются в специальном регистре флагов и приводят к генерации сигналов исключений.

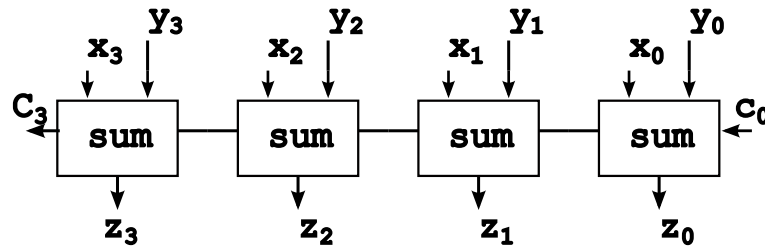


Рис. 1.2. Четырёхразрядный сумматор

Простая схема последовательного сумматора имеет серьезный недостаток: результат на выходах суммы будет верен, только когда во всех вентилях закончились переходные процессы. Если время задержки в одноразрядном сумматоре τ_s , то в n -разрядном последовательном сумматоре правильный ответ будет готов только спустя время $t = n\tau_s$. Таким образом, последовательный сумматор – медленный.

Можно заставить сумматор работать быстрее, но это потребует дополнительной аппаратуры. Задачу решают устройства ускоренного переноса (carry look ahead). В этом случае i -й одноразрядный сумматор вместо переноса вычисляет две логических функции, которые называют генерирование и прозрачность (g_i, p_i) . $g_i = 1$, только если разряд генерирует перенос, независимо от того, приходит перенос из младшего разряда или нет, $p_i = 1$, только если перенос из $i - 1$ разряда проходит в $i + 1$. Несложно понять, что $g_i = x_i y_i$, а $p_i = x_i + y_i$. Важно, что эти функции не требуют знания переноса из младшего разряда. На рисунке 1.3 показана реализация такого одноразрядного сумматора и его обозначение, используемое далее.

В каждом разряде перенос появляется, если он его генерирует ИЛИ пропускает из предыдущего разряда $\Rightarrow C_i = g_i + p_i C_{i-1} = g_i + p_i c_i$. Используя эту формулу, можно записать переносы из каждого разряда

$$\begin{aligned}
 C_0 &= g_0 + p_0 c_0 \\
 C_1 &= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
 C_2 &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
 C_3 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0.
 \end{aligned}
 \tag{1.1}$$

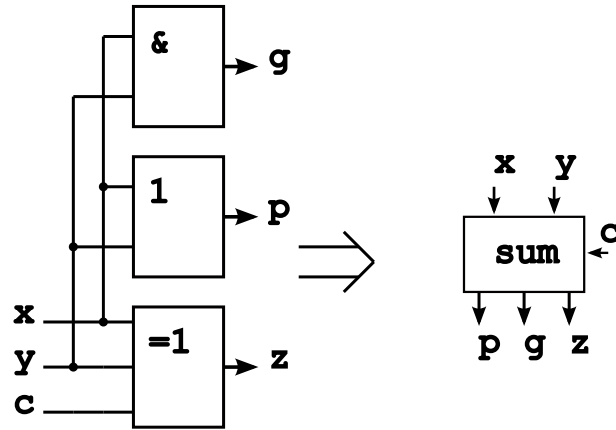


Рис. 1.3. Одноразрядный сумматор со схемами генерации и прозрачности

Упражнение 1.4. *Получите формулы (1.1).*

Ясно, что подобные формулы можно получить для любой разрядности. В формулах 1.1 важно, что для вычисления всех переносов требуется только знание переноса в младший разряд.

Упражнение 1.5. *Последнюю из формул (1.1) изобразите в виде схемы, полагая сигналы g_i, p_i и c_0 входными.*

Четырёхразрядный сумматор с ускоренным переносом изображен на рисунке 1.4. На этом рисунке c_g – перенос из предыдущей группы, $g_g = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$ – генерация переноса группой, $p_g = p_3p_2p_1p_0$ – прозрачность группы, $C_g = g_g + p_gc_g$ – перенос в следующую группу.

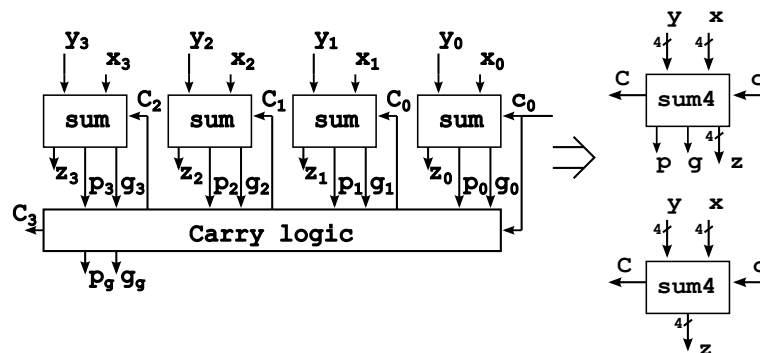


Рис. 1.4. Четырёхразрядный сумматор с ускоренным переносом

Формул стало существенно больше, но вычисления по этим формулам намного быстрее. Это происходит потому, что они выполняются параллельно. Задержка параллельного многоразрядного сумматора – это просто сумма задержек блока ускоренного переноса и одноразрядного

сумматора. При большой разрядности схема ускоренного переноса требует вентилях с очень большим числом входов, что становится невозможно технологически. Поэтому реальные сумматоры составляют из групп, используя сигналы g_g и p_g и повторяя логику ускоренного переноса на уровне групп. Так, из четырёх четырёхразрядных сумматоров можно получить быстрый 16-разрядный сумматор. К сожалению, вентилях будет много. Из всего сказанного ясно, как и какой ценой можно построить быстрые сумматоры на 32 и 64 разряда.

Ещё один важный вопрос – как быть с вычитанием? Весь фокус здесь заключается в том, что в сумматоре практически ничего менять не надо. Пусть мы вычисляем $z = x - y = x + (-y)$. Число $-y$ определяется свойством $y + (-y) = 0$. Обозначим $\bar{y} = \bar{y}_{n-1} \dots \bar{y}_1 \bar{y}_0$ число, полученное из y заменой $1 \leftrightarrow 0$ во всех разрядах. Тогда $y + \bar{y} = 11 \dots 11$. Если теперь прибавить к последнему равенству 1, то получится число из одних нулей (перенос из старшего разряда уйдет “в никуда”). Таким образом, $y + \bar{y} + 1 = 0$ (или $y + \bar{y} + 1 = 2^n$, т. к. 2^n эквивалентна нулю при n -разрядном представлении чисел), следовательно, $-y = \bar{y} + 1 \equiv \tilde{y}$. Число \bar{y} называют обратным кодом y (one’s complementary representation), а число \tilde{y} – дополнительным кодом (two’s complementary representation). Итак, вычитание – это сложение с числом в дополнительном коде $z = x - y = x + \tilde{y}$.

В таблице 1.2 приведены двоичные и дополнительные коды четырехразрядных чисел.

дес.	двоичн.	дополн.	дес.	двоичн.	дополн.
0	0000	0000	8	1000	1000
1	0001	1111	9	1001	0111
2	0010	1110	10	1010	0110
3	0011	1101	11	1011	0101
4	0100	1100	12	1100	0100
5	0101	1011	13	1101	0011
6	0110	1010	14	1110	0010
7	0111	1001	15	1111	0001

Таблица 1.2. Представление четырехразрядных беззнаковых чисел двоичным и дополнительным кодом.

Упражнение 1.6. Попробуйте, действуя как сумматор, вычитать числа, взяв коды вычитаемого из таблицы 1.2. Обратите внимание на случай, когда вычитаемое больше уменьшаемого. Что происходит в старшем разряде?

Разница между $x + y$ и $x - y$ заключается в том, что при сложении на входы y поступает прямой код числа, а на вход переноса в младший разряд ноль, при вычитании на входы y поступает число в обратном коде, а прибавление единицы можно реализовать, подав на вход переноса в младший разряд единицу вместо нуля, который там должен быть при сложении. Смену кода с прямого на инверсный можно выполнить при помощи вентилей исключающее ИЛИ. На схеме это выглядит, как показано на рисунке 1.5. Если на входе управления $\overline{add/sub}$ установлен ноль, то устройство выполняет сложение, если на этом входе единица, то выполняется вычитание.

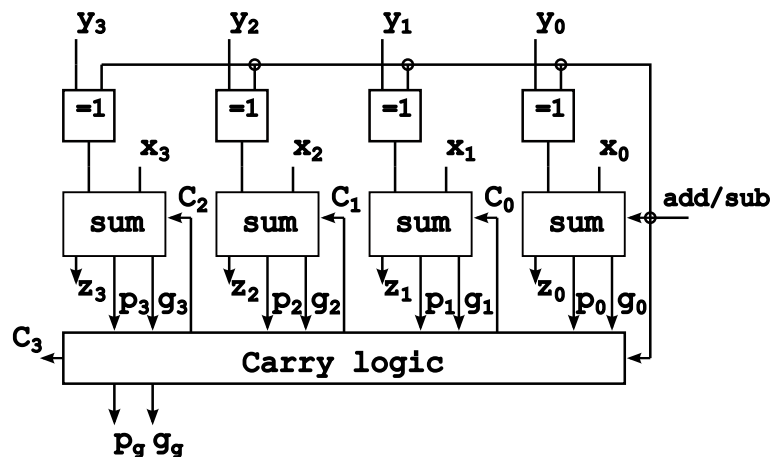


Рис. 1.5. Четырехразрядный сумматор/вычитатель с ускоренным переносом

До сих пор речь шла только о беззнаковых числах. Как строится сложение/вычитание для чисел со знаком? Для начала их надо представить в виде двоичных слов. Во-первых, один разряд надо отдать под знак – пусть это будет старший разряд, тогда коды положительных знаковых чисел совпадут с кодами беззнаковых. Коды отрицательных чисел можно строить по-разному. Очевидный вариант – старший разряд – это знак, а остальные разряды – модуль числа (sign – magnitude representation). Другой вариант – считать кодом отрицательного числа обратный код соответствующего положительного (one's complement representation). В обоих вариантах возникают два представления нуля и обычный сумматор не работает.

Упражнение 1.7. Нарисуйте колонку беззнаковых трехразрядных двоичных чисел, прочитайте их как представление знак – модуль и запишите десятичные эквиваленты. Прделайте то же для представления обратным кодом.

Наилучшим решением оказывается представление отрицательных чисел дополнительным кодом. В правой половине таблицы 1.3 выписа-

ны десятичные знаковые эквиваленты, вычисленные по правилу “смена знака – это дополнительный код”. В этой таблице число -8 представляет собой некоторое исключение (т. к. числа +8 нет), но все работает.

дополн. двоичн.	дес.	дополн. двоичн.	дес.
0000	+0	1000	-8
0001	+1	1001	-7
0010	+2	1010	-6
0011	+3	1011	-5
0100	+4	1100	-4
0101	+5	1101	-3
0110	+6	1110	-2
0111	+7	1111	-1

Таблица 1.3. Представление четырёхразрядных знаковых двоичных чисел

Вы можете самостоятельно убедиться, что сложение чисел со знаком происходит корректно при помощи обычного сумматора (рисунок 1.5). Вычитание числа со знаком также происходит корректно, если переключить режим сумматора/вычитателя, установив единицу на входе управления. При вычитании отрицательного числа сумматор будет вторично вычислять дополнительный код вычитаемого, которое уже в дополнительном коде. Это эквивалентно сложению с модулем вычитаемого, т. е. $x - (-y) = x + \tilde{y} = x + (2^n - \tilde{y}) = x + (2^n - (2^n - y)) = x + y$.

Таким образом, сумматор, показанный на рисунке 1.5, полностью решает задачу сложения и вычитания знаковых и беззнаковых целых чисел.

Переполнение в сумматоре/вычитателе при работе с знаковыми числами возможно, только если оба операнда имеют одинаковый знак. Признаком переполнения является смена этого знака в результате вычисления. Практически используется другой способ определения переполнения – несовпадение переносов в старший бит и из старшего бита. Вентиль, вычисляющий $V = C3 \oplus C2$ (см. рисунок 1.5), решает проблему определения переполнения.

Упражнение 1.8. Убедитесь, что если результат $-8 \leq z \leq +7$, т. е. представим четырёхразрядным кодом, то сумматор/вычитатель работает корректно. Используйте таблицу 1.3. Рассмотрите случаи переполнения.

Сложение и вычитание дробных и плавающих чисел существенно сложнее, и мы его рассматривать не будем.

Перемножитель. Располагая сумматором, можно выполнять умножение, сведя его к сложениям или сложениям и сдвигам, так когда-то и делали, но в современных компьютерах процессор без быстрого аппаратного перемножителя немислим. Умножение одnorазрядных целых чисел реализуется просто, поскольку таблица умножения одnorазрядных чисел совпадает с таблицей истинности логического произведения. Частичные произведения вычисляются мгновенно, все самое интересное начинается потом.

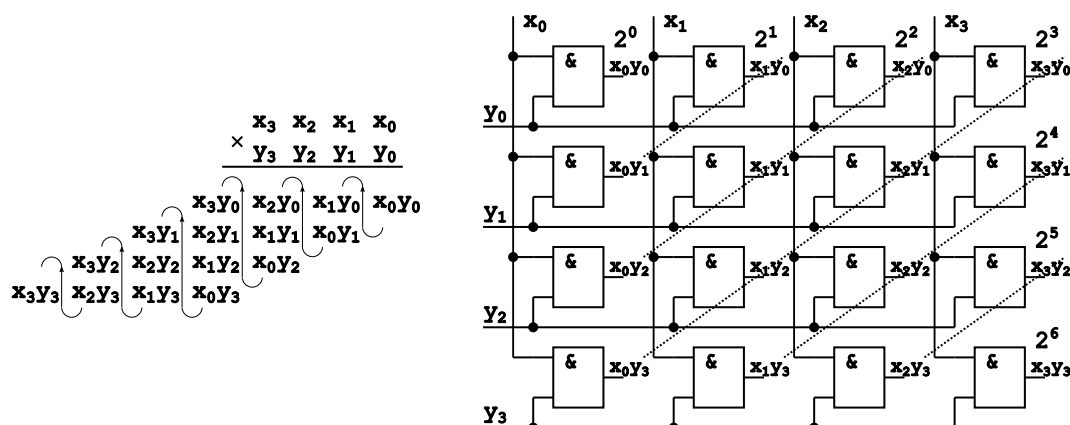


Рис. 1.6. Умножение в столбик и схема вычисления частичных произведений

На рисунке 1.6 показана схема вычисления частичных произведений. На рисунке 1.7 показана непростая схема, которая делает остальную работу. В том виде, как эта схема изображена, она выполняет более сложную функцию, чем произведение четырёхразрядных чисел x и y — $z_8 = x_4 \cdot y_4 + u_8 + w_4$. Входы u и w предназначены для наращивания разрядности; если это не требуется, то значительную часть сумматоров можно заменить элементами исключающее-ИЛИ.

Упражнение 1.9. Нарисуйте схему восьмиразрядного перемножителя, используя четырёхразрядные перемножители как составные блоки.

Перемножитель такого типа, при минимальных изменениях, поддерживает и дополнительную нотацию для знаковых чисел.

АЛУ. Как устроены вычислители побитных логических функций, устройства сдвига и т. п., понять несложно и Вы можете изобразить их самостоятельно. Мы рассмотрим вопрос о том, как можно представлять работу арифметико-логического устройства в целом и связь АЛУ с регистрами процессора. Безусловно, АЛУ является сложным устройством и приводимые схемы не претендуют на абсолютную точность, но идею из них понять можно.

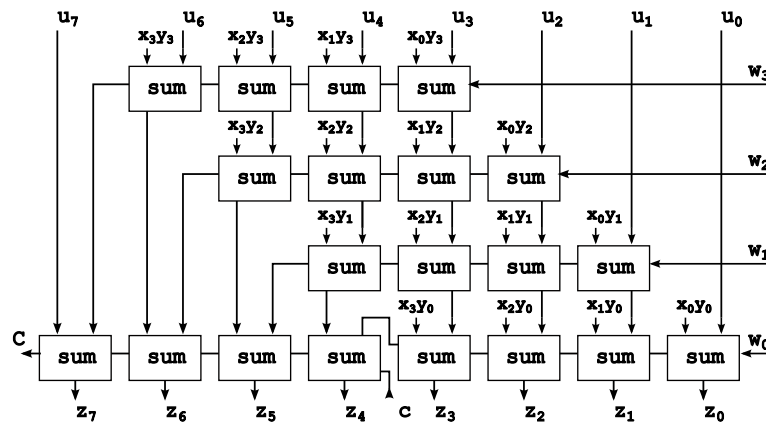


Рис. 1.7. Схема сборки частичных произведений (последовательные сумматоры показаны, чтобы не загромождать рисунок)

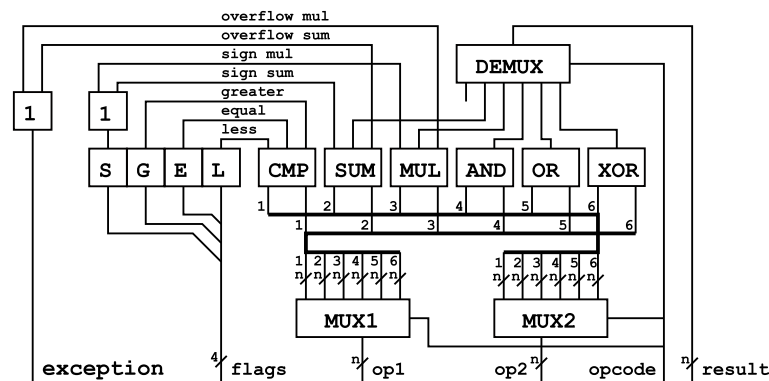


Рис. 1.8. Структура примитивного АЛУ

На рисунке 1.8 приведена схема арифметико-логического устройства, выполняющего арифметические сложение и вычитание со знаковыми и беззнаковыми целыми n -разрядными числами, логические операции с битами чисел и сравнение чисел. Код операции содержит адрес, управляющий мультиплексорами $MUX1$, $MUX2$ и демultipлексором $DEMUX$ и бит add/sub . Этот адрес определяет, в какое из вычислительных устройств будут переданы операнды и какую операцию будет выполнять сумматор. Один вход выходного демultipлексора свободен, т. к. результат сравнения передается через регистр флагов. Поскольку при выполнении операции могут образоваться отрицательные числа, в регистре флагов запоминается этот признак. Возможные результаты сравнения чисел (меньше, равны, больше) также запоминаются в регистре флагов. Переполнения при умножении и сложении отмечаются генерацией исключения.

На блок-схемах АЛУ изображают в виде “штанов”. В процессоре АЛУ связано с оперативными регистрами ($Reg1$, $Reg2$, $Reg3$, $Reg4$), которые хранят данные, могут загружаться из внешней памяти, обмениваться

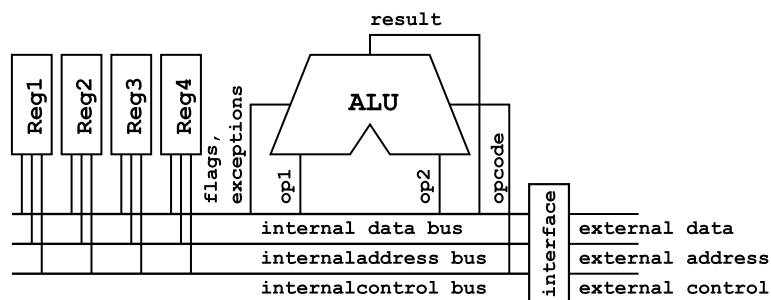


Рис. 1.9. Примитивный процессор (на схеме отсутствует устройство управления)

информацией друг с другом, служить источниками операндов и приемниками результата вычислений АЛУ и обмениваться информацией с внешней памятью. Эти связи показаны на рисунке 1.9.

Упражнение 1.10. Попробуйте написать список инструкций, которые способен выполнить такой процессор. Это, конечно, не *x86*, но список будет немалым.

Как получается, что одни и те же линии могут передавать данные в разных направлениях и не “мешать друг другу”, станет ясно в следующем разделе.

1.2. Интерфейсные схемы

Компьютер объединяет множество устройств, которые должны так или иначе обмениваться информацией друг с другом. Если мы будем соединять эти устройства по принципу “каждое с каждым”, количество связей и интерфейсов при увеличении числа устройств будет расти чрезвычайно быстро. Проблему решает магистрально-модульный принцип объединения устройств. Некоторое представление об этом принципе даёт рисунок 1.10.

Каждый из модулей, используемых в такой системе, имеет некоторую функциональную часть, которая осуществляет обработку информации, и интерфейсную часть, предназначенную исключительно для обмена информацией между модулями. Магистраль, объединяющая модули, – это набор линий, имеющих точно определённое назначение, и протокол обмена, т. е. последовательность действий, которые должно выполнить каждое устройство, участвующее в обмене.

Информационные магистрали – это основной элемент архитектуры цифровых устройств. В вычислительной технике любые устройства

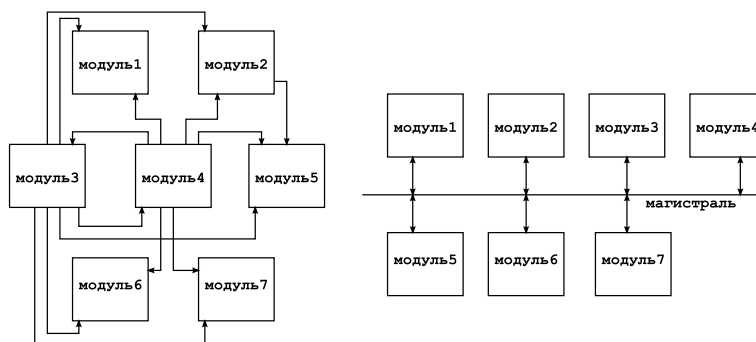


Рис. 1.10. Магистрально-модульный принцип

стандартизованы по интерфейсу, т. е. имеют интерфейсную часть, подчинённую стандарту какой-либо информационной магистрали. Особую роль в компьютерах играют магистрали расширения, т. е. магистрали вокруг которых объединяются разнородные устройства, образующие компьютер. Мы рассмотрим некоторые из них подробно в следующей главе.

Интерфейсная часть цифрового устройства содержит элементы, которые могут не выполнять логических функций, они предназначены для решения специальных задач:

- подключения и отключения устройства к линии магистрали и
- изменения направления распространения информации.

Рассмотрим какую-нибудь линию магистрали расширения (например, линию, по которой передаётся определённый бит данных), устройство, являющееся источником информации, должно быть генератором сигнала, устройство, в которое передаётся информация, – приемником. Любой генератор имеет конечное выходное сопротивление, а каждый приемник является нагрузкой. Если приемников будет много, то сопротивление нагрузки будет мало, и становится возможной ситуация, когда генератор не сможет установить высокий уровень. Другая типичная ситуация возникает, когда с линией соединяются несколько источников данных и каждый из них пытается установить низкий (0) или высокий (1) уровень напряжения. В этом случае начинается состязание выходных сопротивлений – у кого оно меньше, тот и будет “побеждать”. Следовательно, в интерфейсах устройств должна быть возможность отключения их от линий магистрали.

Эту задачу решают отключаемые вентили и вентили с отключаемым выходом. Можно представлять, что в первых имеется возможность отключить вентиль от линий питания. При этом токи не будут замыкаться и сопротивление входа и выхода станут очень высокими. Вентиль не нагружает источник сигнала и теряет способность генерировать сигнал. Устройства второго типа имеют ключ, отключающий только выход

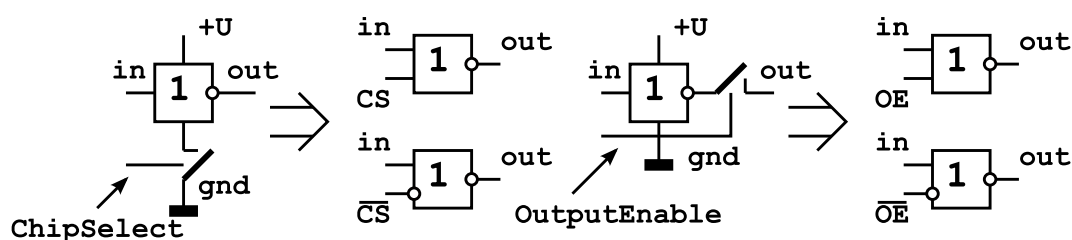
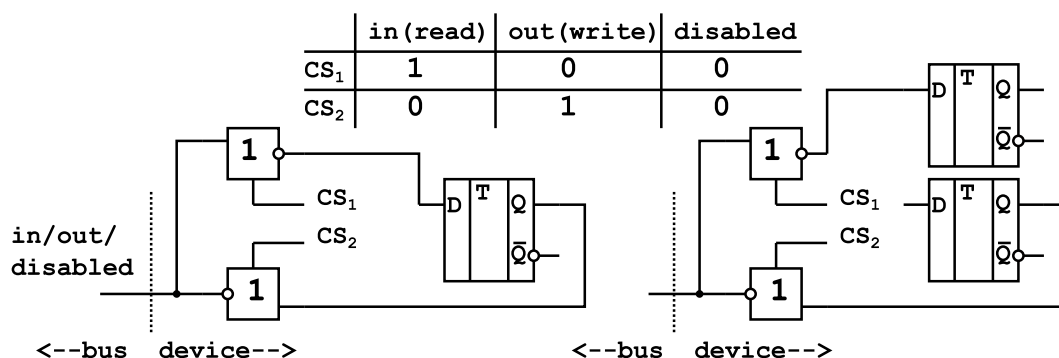


Рис. 1.11. Интерфейсные элементы

Рис. 1.12. Приемопередатчик (управляющие входы CS_1 и CS_2 для удобства показаны сверху и снизу, инверторы выбраны только для того, чтобы обозначить вход и выход управляемого вентиля)

устройства. Электрическая активность остальной части остаётся неизменной, благодаря этому устройства с отключаемым выходом быстрее становятся активными.

В обоих случаях мы получаем состояние, называемое высокоимпедансным или Z-состоянием. Условные изображения таких элементов приведены на рисунке 1.11. Показаны варианты отключения чипа и отключения выхода. Выключатели могут замыкаться по высокому или низкому уровню управляющего сигнала. Показаны обозначения элементов в обоих случаях. Управляющие сигналы CS (Chip Select – выбор чипа) и OE (Output Enable – разрешение выхода). Располагая интерфейсными элементами, мы можем решить и проблему изменения направления передачи сигнала. Схема, решающая эту задачу, показана на рисунке 1.12. У схемы имеются два варианта: в первом информация читается и пишется в один и тот же триггер, во втором – в разные. Эти триггеры могут принадлежать разным регистрам. В списках сигналов нам будут попадаться оба варианта. Несколько непривычный второй вариант обозначается терминами: “при чтении” – “on read”, “при записи” – “on write”. Например, в описании интерфейса устройства может быть такая фраза: при чтении – бит регистра состояния, при записи – бит регистра управления.

1.3. Элементы памяти

Особенности элементов памяти определяются простотой и однообразием выполняемых функций (запись, хранение, чтение) и огромными массивами однотипных элементов (байтов или битов).

Классификация. Элементы памяти весьма разнообразны и их классификация возможна по нескольким признакам. По способу доступа к элементу информации различают:

- RAM – random access memory-память с произвольным доступом,
- Sequential access memory – память с последовательным доступом (FIFO и LIFO).

Аббревиатура LIFO обозначает last in – first out (последним вошёл – первым вышел), иначе такой способ доступа к данным называют стек (stack – стопа). Наглядный пример – стопка тарелок: новую тарелку можно положить только сверху и взять можно только верхнюю. FIFO обозначает first in – first out (первым вошёл – первым вышел), иначе такой способ доступа называется очередь или буфер.

По возможности чтения и записи различают:

- RAM (random access memory) – память с произвольным доступом,
- ROM (read only memory) – память только для чтения.

Некоторую путаницу вызывает термин RAM, объединяющий два признака – доступ к произвольному элементу и возможность как чтения, так и записи. ROM – read only memory – память только для чтения – память, которая может читаться обычным способом, но запись в неё включает два действия – стирание (все или группа битов обнуляются) и последующую запись (в необходимые биты записывается единица). Первые ROM большого объёма требовали стирания информации ультрафиолетовым светом при повышенной температуре и последующей записи импульсами большой амплитуды. Будучи установленной в компьютер, такая память не модифицировалась и обычно использовалась для хранения базовой системы ввода/вывода - BIOS. EEPROM (electrically erasable programmable read-only memory) – электрически стираемая программируемая память, которую можно переписывать в системе.

По необходимости регенерации различают:

- SRAM (static random access memory) – статическая память с произвольным доступом,
- DRAM (dynamic random access memory) – динамическая память с произвольным доступом.

Статическая память не требует обслуживания в процессе хранения информации и работает, пока есть питание её элементов. Динамическая память теряет информацию за определённое время, и требуется её регенерация с определённой периодичностью. Элементы динамической памяти предельно просты и исключительно малы по размерам – это микроскопические конденсаторы, выполненные на кремниевой подложке. Стоимость таких элементов памяти мала, и это компенсирует все её недостатки.

По зависимости от источника энергии различают:

- Volatile memory – энергозависимая память,
- Nonvolatile memory – энергонезависимая память (часто поддерживается собственным источником).

Физические принципы. Элементом памяти может быть любое устройство, имеющее два устойчивых состояния, между которыми возможны переходы и которые можно идентифицировать тем или иным способом. Эти состояния ассоциируются с логическими «0» и «1». Устройства должны быть просты, надежны, малы и дешевы, поскольку таких устройств требуется очень много. В вычислительной технике в разное время использовалась память различных типов. Не претендуя на полноту списка, приведем некоторые типы запоминающих устройств.

1. Магнитная память. Эта память реализуется на магнитных элементах с так называемой прямоугольной петлей гистерезиса, т. е. элементах, у которых намагниченность в отсутствие внешнего поля может принимать только два возможных направления, которые соответствуют нулю и единице. Элементы МОЗУ чаще всего представляли собой ферритовые колечки с внешним диаметром вплоть до 0.5 мм. Главной особенностью такой памяти является чтение с разрушением состояния, т. е. считанная единица уничтожается и требуется цикл её регенерации. Предпринимались попытки изготовить МОЗУ с пленочными магнитными элементами, которые могли бы конкурировать с интегральной электроникой по объему, но их быстродействие оказалось хуже.

2. Триггеры. Самый примитивный триггер может быть реализован на 4 транзисторах, при этом получается хорошее быстродействие (как у логики), но цена таких элементов памяти оказывается высокой. Электронные триггеры являются элементами наиболее быстродействующей регистровой и кэш-памяти.

3. Устройства с хранением заряда. Имеются разновидности полевых транзисторов с изолированным затвором, под которым может долго сохраняться заряд. Этот заряд может уничтожаться ультрафиолетовым

светом или повышенным напряжением на затворе. Разновидностью таких приборов является flash память – энергонезависимая память с высокой скоростью записи и стирания. Такая память основана на полевом транзисторе с плавающим затвором.

4. Цилиндрические магнитные домены. Это микроскопические области в ферромагнитных пленках, имеющие намагниченность, ориентированную противоположно окружению. Такая область может перемещаться с большой скоростью вдоль пленки под действием стороннего магнитного поля и регистрироваться при прохождении под специальным считывающим проводником. Не потребляют электроэнергию при хранении, но сложны и дороги.

5. Сверхпроводящая память. Основана на свойствах так называемых джозефсоновских контактов, представляющих собой трёхслойную структуру из сверхпроводника, диэлектрика и ещё одного сверхпроводника. Ток через такой контакт зависит от того, каким образом изменялось напряжение на контакте и может при одном и том же напряжении иметь два значения – близкое к нулю и некоторое высокое значение.

6. Память на поверхности кремниевого чипа. В стадии разработки. Принцип предельно прост – раскладывание отдельных атомов в определённых позициях на поверхности атомарно гладкого кремния. Наличие или отсутствие атома считывается при прохождении над атомом игольчатого электрода как в туннельном микроскопе. Запись осуществляется тем же электродом при подаче на него импульса напряжения, при котором происходит сброс атома на поверхность. Энергонезависима.

7. Заряд конденсатора. Наиболее распространённый способ хранения информации. Несмотря на необходимость периодической регенерации, позволяет организовать чипы гигабайтной емкости и считывать информацию за приемлемое время (наносекунды).

8. Оптическая память. В варианте, используемом в современных CD и DVD, это просто создание на зеркальной поверхности точек – питов (pit – яма, ямка), от которых свет отражается диффузно.

Мы рассмотрим подробнее только два типа элементов памяти – триггер на полевых транзисторах и конденсатор. Элементы памяти таких типов очень широко используются в современных компьютерах в качестве быстродействующей регистровой и кэш-памяти, а также системной памяти. Статическая память на триггерах присутствует в современных компьютерах блоками от сотен байт до мегабайтов, чипы динамической памяти имеют емкость в несколько гигабайт. Принципиальная схема единичного бита таких типов приведена на рисунке 1.13. На правых рисунках показана электрическая схема, на левых – функциональный эквивалент.

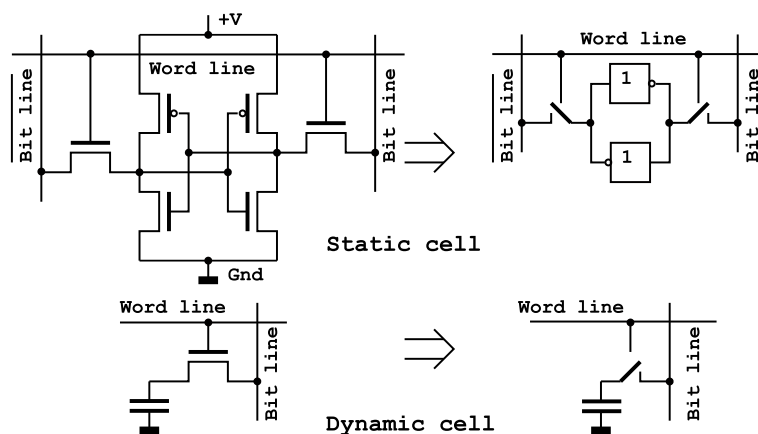


Рис. 1.13. Ячейки статической и динамической памяти (1 бит) с линиями доступа

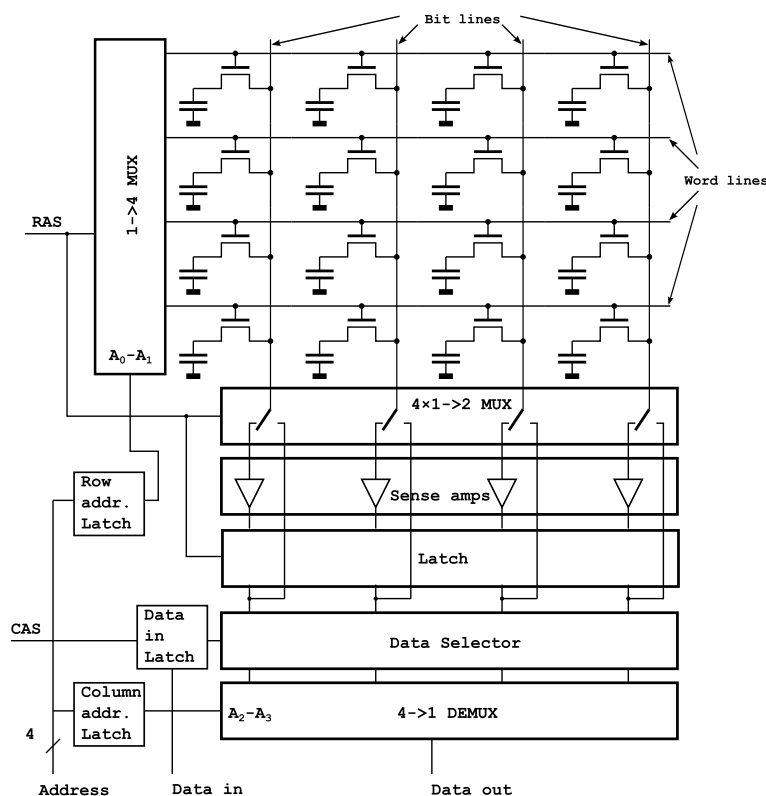


Рис. 1.14. Пример матрицы битов 4*4

В схеме статической памяти можно узнать установочный триггер, изображенная схема называется “6T memory cell” по числу транзисторов. Особенность этого триггера состоит в том, что он не имеет специальных входов установки и сброса. Запись информации, т. е. установка и сброс триггера, осуществляется “партизанским” способом. При присоединении выхода триггера к источнику с малым выходным сопротивлением этот источник “победит” триггер и установит свой уровень. Такой способ записи имеет недостаток – он медленнее, чем у настоящего установочного

триггера, но значительно быстрее, чем у элемента динамической памяти. Элементы статической памяти не изменяют своего состояния при чтениях и во время хранения информации. Полевые транзисторы, не входящие в триггер, являются ключами доступа. В принципе было бы достаточно одного ключа и одной линии битов, но два ключа позволяют измерять состояние триггера дифференциально, что значительно повышает помехоустойчивость и в конечном итоге быстродействие устройства. Как становится ясным из описания, такой элемент памяти не является цифровым устройством в точном смысле этого слова. В ещё большей степени это относится к элементам динамической памяти.

В динамической памяти битом является конденсатор. Если он несет заряд выше некоторого порога, то записана единица, если ниже, то записан ноль. Чтение информации связано с разрядом конденсатора. Таким образом чтение уничтожает информацию. Кроме того, при сколь угодно высоком качестве диэлектрика, саморазряд конденсатора неизбежен и динамическая память требует достаточно сложного обслуживания. Восстановление считанной информации называют регенерацией, и она происходит при каждом чтении. Все эти сложности окупаются малыми размерами элемента памяти. Какова бы ни была сложность микроэлектронного устройства, его стоимость пропорциональна площади, которую оно занимает на поверхности чипа, поэтому стоимость одного мегабайта динамической памяти существенно меньше стоимости статической памяти.

Организация памяти. Серьезной проблемой при организации массивов памяти большой емкости является большое число выводов устройства, если организовывать доступ к каждому биту индивидуально. Идеальный чип памяти должен иметь выводы данных, число которых равно ширине шины данных, и выводы адреса, число которых равно ширине шины адреса. Это реализуется при помощи матричной организации обращений к битам памяти.

Мы рассмотрим организацию динамической памяти, поскольку она является наиболее распространённой в системе и порождает проблемы, которые в значительной степени определяют архитектуру всего компьютера. На рисунке 1.14 показано соединение массива битов размером 4×4 с линиями слов и линиями битов. В реальных системах число линий слов и битов порядка нескольких тысяч ($2^{10} \div 2^{13}$). Младшая часть адреса ($A_0 - A_1$) запоминается регистром – защелкой адреса слова (row address latch). Двоичный код этого адреса выбирает линию слова, через которую пройдет импульс RAS (row access strobe). Это функция мультиплексора ($1 \rightarrow 4MUX$). Импульс RAS (row access strobe) открыва-

ет ключи доступа ко всем конденсаторам в выбранной линии. Выбранное слово затем считывается целиком, это медленный процесс. Заряды с конденсаторов, которые были заряжены (хранили 1), начинают проходить через вход усилителей (sense amps), импульсы тока усиливаются до логического стандарта и запоминаются в регистре – защелке (latch). Регистр-защелка – это быстродействующий регистр, у которого время срабатывания мало по сравнению с временем доступа к емкостям динамической памяти. Защелка теперь хранит большое слово. Задний фронт RAS переключает мультиплексоры ($4 \times 2 \rightarrow 1MUX$) и соединяет выходы защелки (latch) с линиями всех колонок, в результате конденсаторы, с которых были считаны единицы, заряжаются снова. Это происходит быстро, поскольку выходы защелки имеют малое выходное сопротивление. Таким способом осуществляется регенерация информации при чтении. Требуемый бит выбирается мультиплексором ($4 \rightarrow 1MUX$), который управляется старшей частью адреса ($A_2 - A_3$). Он попадает на внешнюю шину данных через приемопередатчик. Процедура записи в память похожа на то, что происходит при регенерации. Регенерация в строках, которые не считывались, осуществляется в циклах псевдоочтений, которые контроллер памяти генерирует автоматически. Как ясно из описания, если делать слова широкими, то после всех приготовлений можно быстро считывать массив, лежащий в защелке (latch). В течение этого времени можно начинать подготовку доступа к следующему слову. Таким образом, появляются широкие возможности оптимизации хранения и организации запросов. Далее, в разделе, посвященном кэшированию, мы увидим, как надо организовывать хранение информации в динамической памяти. Из описания обращения к памяти становится понятно, что время единичного обращения к памяти не является универсальной характеристикой её производительности. В случае считывания мы получаем некоторую задержку (latency time – латентное время) вначале и высокий темп передачи блока. Такая технология называется взрывным или пакетным режимом передачи данных (burst mode).

1.4. Кэширование памяти

Закон близости. Кэширование памяти. В фон-неймановском компьютере наиболее интенсивно происходит обмен информацией между центральным процессором и памятью. Этот обмен имеет два основных потока: первый – выборки команд, второй – загрузки и сохранения данных. В обоих потоках действует статистический «закон близости». Сформулировать этот закон можно приблизительно следующим обра-

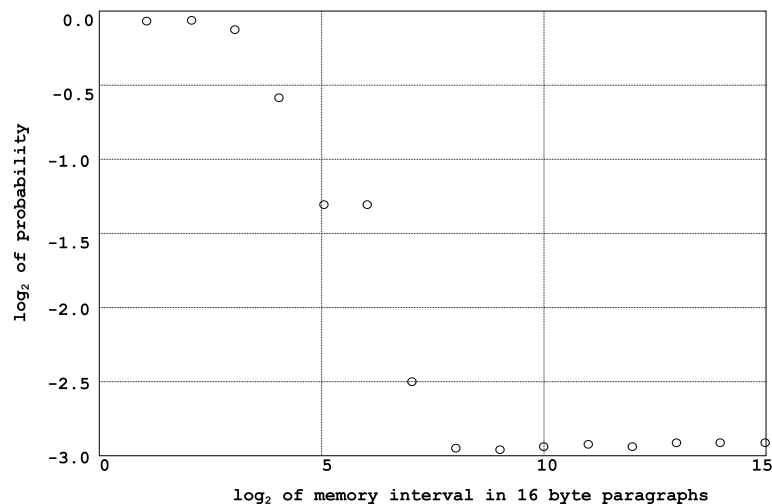


Рис. 1.15. Закон близости при обращениях к сегменту кода

зом: «Вероятность того, что следующее обращение к памяти будет сдвинуто относительно предыдущего, быстро убывает при увеличении сдвига». Закон можно проиллюстрировать графиком, показанным на рисунке 1.15.

Этот интуитивно понятный закон, дополненный простым соображением о том, что быстрая память дороже медленной, приводит к идее кэширования. Время обращения к динамической памяти (DRAM) составляет приблизительно 10нс, а время обращения к статической памяти (SRAM) – 1нс. Конкретные цифры меняются, но отношение времени обращения к статической памяти к времени обращения к динамической памяти остаётся примерно равным 1:10 в течение многих лет. Идея состоит в том, что необходимо разделять память на две части – быструю память небольшого объёма, в которой хранится фрагмент информации, необходимый для выполнения ближайших команд программы (как собственно команд, так и операндов, необходимых для их выполнения), и медленную, но большую основную память, из которой эти фрагменты будут время от времени копироваться. Первая часть памяти называется кэш (от французского слова *cache* – закладка, тайник). Мы рассмотрим основные понятия кэширования памяти, не касаясь схемотехнического уровня её реализации.

Уровни кэша. Если проводить идею кэширования до конца, то становится понятно, что оптимальной организацией памяти компьютера является иерархия кэшей нескольких уровней, различающихся соотношением скорость-объём-цена. Правильные архитектура аппаратуры и программирование могут дать огромный выигрыш в производительности системы в целом при использовании кэшей. Система будет работать

так, словно в её распоряжении имеется быстрая память большого объёма. Кэширование может также значительно улучшить работу основной памяти, поскольку, как мы видели, её производительность значительно возрастает при передаче информации блоками.

Уровни кэша начинаются с регистров процессора. Если угодно, их можно назвать кэшем нулевого уровня. Процессор обращается за инструкциями и операндами к кэшу первого уровня (Level 1 cache – L1). В архитектурах x86 кэш первого уровня делится на две части L1-code и L1-data. Он является самым быстрым и наиболее сложным по архитектуре. Промехи в L1 ведут в кэш второго уровня (Level 2 cache, L2). Это существенно больший однородный кэш с меньшим быстродействием и более простой архитектурой. В старых процессорах Intel кэш второго уровня был внешним и для обмена с этим кэшем например в процессорах семейства P5 существовала особая шина – backside bus, которая вместе с обычной шиной, предназначенной для связи с устройствами – frontside bus образовывала т. наз. двойную независимую шину – dual independent bus. Позднее кэш L2 стал частью самого процессора. В самых современных многоядерных процессорах имеется ещё и кэш третьего уровня (Level 3 cache, L3). Размеры кэшей L1 в современных процессорах Intel не изменяются и составляют 64KB (32KB – code, 32KB – data). Размеры кэша L2 достигают 8MB, а кэши L3 бывают и 30MB. Динамическую память можно рассматривать как кэш четвертого уровня, а диск – как кэш пятого уровня.

Когерентность, теги, поиск, попадания и промахи. Главной особенностью кэш-памяти является то, что она отображает фрагмент из основной памяти или кэша низшего уровня. Это приводит к необходимости поддержания их соответствия (consistency) или, как иногда говорят, когерентности (coherency). Проблема когерентности кэша является частью более общей проблемы – когерентности памяти, которую используют несколько пользователей или программ (shared memory). Кэш должен не только помнить какую-то конкретную информацию, но и адреса в основной памяти, откуда эта информация была загружена. Эту задачу решает специальная небольшая область памяти, которая называется памятью тегов (tag memory). Теги содержат адрес, с которого начинается единица хранения (строка – см. ниже) в основной памяти. Просмотр этой области, необходимый для решения вопроса о наличии в кэше требуемой информации, называют циклом поиска (snooper cycle). Этот поиск должен быть очень быстрым, иначе весь выигрыш от использования быстрой памяти будет сведен на нет долгим поиском. Поиск осуществляется компараторами адресов (запрошенного и хранящихся в памяти

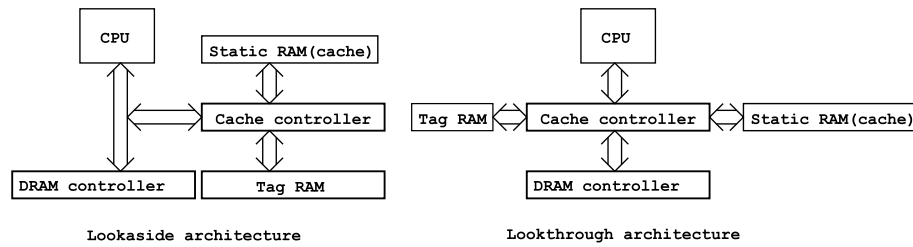


Рис. 1.16. Архитектуры чтения

тегов) – это основной элемент контроллера кэша. Результат поиска может быть положительным (информация имеется в кэше) или отрицательным (информации нет). Первый случай называют кэш попадание (cache hit), второй – кэш промах (cache miss).

Архитектуры чтения и политики записи. Поиск данных в памяти можно начинать двумя способами. Первый способ – пессимистический – начинать одновременно запрос к медленной основной памяти и в кэше (lookaside architecture). Второй способ – оптимистический – начинать поиск в кэше и, только если не повезет, в основной памяти (lookthrough architecture). Понятно, что первый способ явно лучший – если не найдем в кэше, то не потеряем время при поиске в динамической памяти. Но этот способ энергозатратный. Контроллер динамической памяти и особенно сама динамическая память при обращениях потребляет много энергии. Этот способ оказался неприемлемым в ноутбуках. На рисунке 1.16 показано расположение контроллера кэша в этих архитектурах.

Несколько сложнее вопрос о записях в кэш. Когерентность кэша требует одновременной замены данных в кэше и основной памяти (Write-Back), но это серьезная потеря времени и явная перестраховка. Поэтому используется и другой способ записей – запись только в кэш (Write-Through). При этом способе возникает несоответствие информации в кэше и основной памяти. Если данные модифицированы в кэше, но не модифицированы в основной памяти, их называют грязными или неправильными (dirty data, invalid data). Если данные модифицированы в основной памяти, но не модифицированы в кэше, их называют устаревшими (stall data – переводов много, приведен самый мягкий). Политику Write-Through можно реализовать, осуществляя отложенные записи в основную память, при освобождении магистрали.

Архитектуры кэша. Базовой единицей хранения информации в кэше является строка. Размер строки в разных архитектурах разный и варьирует от 2 до 512 байт. Обычно строку кэша выбирают так, чтобы её

размер был больше максимальной длины запроса процессора. Для архитектуры x86 самые большие типы данных (doublequadword и ХММ) имеют размер 16 байт. Наиболее рациональный выбор длины строки кэша – это максимальная длина передачи в пакетном цикле передачи информации (burst mode). Например, в процессорах семейства Р6 длина строки кэша составляет 32 байта, в Core i7 – 64 байта. Строка неделима при обменах с основной памятью, но после считывания можно обращаться к любой её части.

Основную память можно условно разделить на строки с длиной, совпадающей с длиной строки кэша. Кэши, в которых любая строка основной памяти может отображаться в любую строку кэша, называют полностью ассоциативными (Fully Associative cache). Цикл поиска, т. е. сравнения запрошенного адреса с адресами в тегах таких кэшей, возможен, только если кэш маленький.

Другая крайность в архитектуре кэша – условно разделить основную память на страницы (Memory Pages), размер которых совпадает с размером всего кэша, и разрешить отображение в строке кэша только строки с одной страницы. Такие кэши называют кэшами прямого отображения (Direct Mapped cache). В этом случае поиск значительно проще и может выполняться аппаратно контроллером кэша за минимальное время. Но такие кэши дают значительную долю промахов. И если, например, обращения к памяти прыгают между парой страниц (не слишком редкая ситуация – например, два массива, размещённых на разных страницах памяти), кэш постоянно будет грузиться новыми и новыми строками. Описанная ситуация называется пробуксовкой кэша (cache trashing).

Между этими крайностями лежат реально используемые кэши. Они называются наборно-ассоциативными (Set-Associative cache). В таких кэшах на строку в кэше может отображаться строка с двух или большего числа страниц. Такие кэши называются n -входовыми наборно-ассоциативными (n -Way Set-Associate cache). Относительно медленные восьмивходовые кэши используются в качестве L2, четырехвходовые – в качестве L1. Если объём кэша мал, то могут использоваться и полностью ассоциативные кэши.

Глава 2.

Информационная магистраль

2.1. Общие принципы

Устройства и интерфейсы. Все устройства, составляющие компьютер, должны взаимодействовать друг с другом, следовательно, независимо от функционального назначения устройства должен существовать некоторый универсальный способ обмена информацией, одинаковый для всех устройств, входящих в систему. Каждое устройство, помимо функциональной части, должно иметь одинаковую для всех устройств интерфейсную часть. Мы можем, исходя из самых общих соображений, сформулировать основные требования, которым должен удовлетворять способ обмена.

Адреса, данные, управление. Так или иначе, каждое устройство, участвующее в обмене, должно отличаться от остальных, иметь уникальное имя или адрес. На информационной магистрали для передачи адресов существует группа линий, предназначенных специально для этой цели, – адресная шина (address bus). Подавляющее количество адресов принадлежит контроллеру памяти, поскольку каждый байт информации в оперативной памяти должен иметь собственный адрес. Группа линий, предназначенная для передачи информации, называется шиной данных (data bus). Сигналы управления, определяющие тип передачи, синхронизирующие передачу, несущие сообщения об ошибках и т. д., образуют шину управления (control bus).

Задатчик и исполнитель. В каждый момент времени по шине могут взаимодействовать только два устройства: задатчик (master, initiator) и

исполнитель (slave, target). Задатчиком называют устройство, владеющее шиной адреса. С точки зрения электроники это обозначает, что адресные контакты задатчика находятся в активном состоянии и являются выходами. У второго участника обмена (slave) адресные контакты активны, но являются входами. Эта ситуация называется задатчик «владеет» шиной или «захватил» её. С точки зрения обмена задатчик инициирует обмен: выбирает партнера (исполнителя) и указывает тип передачи. Каждое чтение одновременно является записью для второго участника обмена. Точный смысл слова “запись” и “чтение” приобретают, только если известно, относительно какого устройства определяется направление передачи. Традиционно таким устройством является задатчик.

Прерывания. Если не будет существовать возможности прервать выполнение некоторой программы, то компьютер будет “жить богатой внутренней жизнью”, о которой никто ничего не узнает. Следует различать аппаратные прерывания и так называемые программные прерывания. Источником первых являются устройства – участники обмена (в том числе и процессор), вторые исходят от инструкций прерывания и представляют собой лишь способ выполнения программного кода. Выполнение прерывания происходит по следующему сценарию: устройство, требующее прерывания, устанавливает на магистрали сигнал требования прерывания (interrupt request – IRQ), по этому требованию процессор выполняет одну из загруженных в оперативную память программ – обработчиков прерывания (interrupt handlers). Выполнение программы-обработчика называют обслуживанием прерывания. Поскольку срочность выполнения различных прерываний различна, для них устанавливаются уровни приоритета. Поддержание очередности обслуживания прерываний с учетом приоритета осуществляется системным устройством – контроллером прерываний (interrupt controller – IC).

Прямой доступ к памяти. Передача информации от одного устройства ввода/вывода другому – явление экзотическое, но передача между памятью и устройством ввода/вывода – явление типовое. Конечно, можно его осуществить через процессор, но это значит, что в течение значительного времени процессор будет занят. Проблему решает прямой доступ к памяти (direct memory access – DMA). Умение управлять шиной адреса требует дополнительной аппаратуры, на которой можно сэкономить. Идея заключается в следующем: кроме процессора, который должен уметь управлять всей памятью на магистрали устанавливается ещё одно такое устройство – контроллер прямого доступа к памяти (DMA controller – DMAC), который “предлагает свои услуги” любому другому,

не умеющему управлять шиной адреса. На время передачи устройство и DMAC образуют “тандем”, в котором адресами управляет DMAC, а данными – устройство.

Мультиплексирование. Многие сигналы в принципе не могут появляться на линиях магистрали одновременно. В этом случае удобным приемом, позволяющим значительно упростить аппаратуру, является мультиплексирование. Суть мультиплексирования заключается в передаче разных по смыслу сигналов по одной линии в разное время. Этот прием используется на многих информационных магистралях. Крайнее свое выражение мультиплексирование получило в последовательных информационных магистралях, где назначение сигнала определяется его положением в стандартном пакете.

2.2. ISA и её расширения

История. Магистраль ISA (Industry Standard Architecture) впервые появилась в персональных компьютерах IBM PC/XT в 1981 году. Первоначально это была магистраль с 8-разрядной шиной данных и 20-разрядной шиной адреса. Структура и протокол обмена по этой магистрали не были запатентованы разработчиком и не являлись интеллектуальной собственностью корпорации IBM. Большой успех персональных компьютеров PC/XT в качестве офисных машин привел к появлению клона PC-подобных компьютеров и большого количества их производителей. Стандарт, не состоявшийся *de jure*, состоялся *de facto*. В дальнейшем он получил название ISA-8, чтобы отличать его от расширенного стандарта ISA-16, принятого в результате усилий специалистов производителей PC-совместимых компьютеров. В стандарте ISA-16 предусматривалась 16-разрядная шина данных и 24-разрядная шина адреса. В 1988 году консорциум компаний AST Research, Compaq, Epson, Hewlett-Packard, NEC, Olivetti, Tandy, Wyse, Zenith опубликовал 32-разрядный вариант этой магистрали, названный EISA (Extended Industry Standard Architecture). Этот интерфейс оказался дорогим и прижился только на серверах. В настольных компьютерах выиграл более простой вариант VESA local bus – VLB (VESA – Video Electronics Standards Association), который решил главную стоявшую в тот момент задачу – ускорил вывод изображений на монитор. Далее началось вытеснение всех вариантов ISA-магистрали из персональных компьютеров пришедшей на смену 64-разрядной магистралью PCI. Строго говоря, магистраль ISA никуда не делась и в современных компьютерах, где слоты ISA отсутствуют. В

диспетчере устройств на магистрали PCI вы можете найти устройство “Стандартный мост PCI-ISA”, за которым стоит эта магистраль и на ней имеется много устройств.

Сигналы и порядок обмена. Поскольку шина ISA предназначена для персональных компьютеров, предполагается, что на ней присутствует процессор, который является задатчиком по умолчанию. Второе устройство, которое может владеть шиной ISA, – это контроллер прямого доступа к памяти (DMA controller). Все линии шины ISA образуют несколько групп. В таблице 2.1 показаны линии данных и адреса. Прежде всего, необходимо выделить линии шины данных (data bus). Эти линии предназначены для передачи данных между устройствами, присутствующими на шине. Их назначение понятно из названий. При 16-разрядном обмене SD0-SD7 передают байт, имеющий меньший (четный) адрес и содержащий младшие биты 16-разрядного числа, а SD8-SD15 – передают старший (нечетный) байт, содержащий старшие биты 16-разрядного числа. Такой порядок называется “младшие в начале” (little-endian) и не является единственно возможным, например процессоры Motorola 6800 и 68000 требуют другого порядка, который называется big-endian. Существуют архитектуры, в которых порядок может переключаться (bi-endian). Проблема соотношения порядка битов и байтов (в более общем случае индивидуально адресуемых слов в памяти) начинает разрастаться в 32-разрядных и 64-разрядных архитектурах и влечет за собой несовместимости программных кодов и трудно определяемые ошибки.

Назначение	Названия ISA8	Названия ISA16
Данные	SD0-SD7	SD8-SD15
Адреса	SA0-SA19	LA17-LA23
Строб адреса	BALE	
Разрядность передачи		SBHE, M16, IO16

Таблица 2.1. Данные и адреса на магистрали ISA

Сразу обратим внимание на то, что адресные линии с 17-й по 19-ю присутствуют на шине в двух экземплярах. Кроме того, стандартные обозначения адресных линий для ISA8 и ISA16 различны. Аббревиатура SA обозначает – System Address, а LA – Latchable Address. Эти названия отображают различный способ выбора момента фиксации кодов на этих двух группах адресных линий.

Строб адреса (BALE – bus address latch enable) – это импульс, который генерируется задатчиком после того, как он установил биты SA

и одновременно с установкой битов LA. По заднему фронту BALE контроллер памяти и устройства, работающие как память, должны зафиксировать старшие адреса LA. Другого времени для их запоминания не будет, поскольку задатчик имеет право сразу после окончания BALE устанавливать старшие разряды адреса для следующего цикла. Таким образом делалась попытка организовать конвейер, т. е. начинать следующую передачу по магистрали до окончания предыдущей. Младшие разряды адреса (SA) могут восприниматься как быстрым контроллером памяти, так и медленными устройствами ввода/вывода, поэтому их необходимо удерживать до конца цикла. Обычно устройства фиксируют их по тому же заднему фронту BALE, но можно это сделать и позже.

Сигнал SBHE устанавливается задатчиком, если он требует 16-разрядной передачи. Исполнитель, которым может быть контроллер памяти или устройство, при опознании своего адреса устанавливают сигналы M16 или IO16, если они поддерживают 16-разрядный обмен. Если при этом ещё и адрес, к которому идет обращение, четный (SA0=0), то происходит 16-разрядный обмен.

Назначение остальных линий шины управления, которые приведены в таблице 2.2, будем рассматривать по группам. Первая группа определяет тип передачи. Источником или приемником данных могут быть только порт ввода/вывода (I/O port) или ячейка памяти (memory location).

Назначение	Названия ISA-8	Названия ISA-16
Тип операции.	SMEMR,IOR, SMEMW,IOW	MEMR, MEMW
Прерывания.	IRQ3-7	IRQ9-12, IRQ13-15
Запрос DMA.	DRQ1-3	DRQ0,5-7
Подтверждение	DAK1-3	DAK0,5-7
Разр. адреса	AEN	
Последняя передача	TC	
Тактовая частота	CLK,OSC	
Готовность устройства	CHRDY, NWS	

Таблица 2.2. Линии управления магистрали ISA

Поэтому в стандарте предусмотрены передачи 6 типов:

- чтение из памяти SMEMR,

- запись в память SMEMW,
- чтение из порта IOR,
- запись в порт IOW,
- чтение из нижней памяти MEMR,
- запись в нижнюю память MEMW.

SMEMR, SMEMW обозначает Standard Memory Read/Write. IOR и IOW обозначают I/O (port) Read/Write. Сигналы MEMR и MEMW присутствуют только на ISA16, и активный уровень на них обозначает передачу в память до 1МВ (достаточно прочитать только 20 линий адреса). Задние фронты этих импульсов указывают моменты фиксации данных и являются окончанием цикла передачи. Между обменами с контроллером памяти и устройствами имеется существенная разница. При обмене с памятью задатчик просто отсчитывает некоторое количество тактов (от 2 до 5), после чего предполагается, что данные готовы и происходит их запись или чтение. При обмене с устройством происходит вставка циклов ожидания устройства, которая прекращается после снятия сигнала устройством сигнала CHRDY (channel ready). Быстрое устройство может ускорить цикл обмена, установив сигнал NWS (no wait states).

Следующая группа сигналов ISA-магистрали управляет прерываниями. Сигналы требования прерываний (IRQ – interrupt request) исходят от устройств и обрабатываются контроллером прерываний, который преобразует их в один сигнал INT на шине процессора. Все дальнейшее определяется программным обеспечением. Отсутствующие на шине номера прерываний задействованы некоторыми выделенными устройствами: IRQ0 – таймер, IRQ1 – клавиатура, IRQ2 – каскадирование второго контроллера, IRQ8 – часы реального времени, IRQ13 – математический сопроцессор. Большинство остальных линий прерывания также заняты (порты COM и LPT, FDD и HDD). Стандартное распределение устройств IBM PC по номерам прерываний приведено в таблице 2.3. Рассмотренные аппаратные прерывания не следует путать с программными прерываниями, которые являются только способом обращения к системной процедуре (обработчику прерывания).

Механизм прямого доступа устройств к памяти – Direct Memory Access (DMA) – реализуется группой сигналов запроса DMA – DRQ (DMA request) и подтверждения DMA – DAK (DMA acknowledge). При прямом доступе к памяти устройство и контроллер DMA «объединяют усилия» для осуществления передачи. Устройство генерирует данные на шине данных, а контроллер DMA генерирует адреса и стробы данных. Прежде чем контроллер DMA заработает, он должен быть запрограммирован, т. е. в него необходимо передать начальный адрес в памяти,

Номер	Устройство	Номер	Устройство
IRQ0	System Timer	IRQ8	Real Time Clock
IRQ1	Keyboard	IRQ9	Redirect. to IRQ2
IRQ2	Cascade from slave PIC	IRQ10	Reserved
IRQ3	COM2	IRQ11	Reserved
IRQ4	COM1	IRQ12	Mouse Interface
IRQ5	LPT2	IRQ13	Coprocessor
IRQ6	Floppy Drive Controller	IRQ14	Hard Drive Controller
IRQ7	LPT1	IRQ15	Reserved

Таблица 2.3. Распределение прерываний между стандартными ISA-устройствами

размер блока в памяти и направление передачи. Эта передача происходит по шине данных в ответ на подтверждение данных (DAK). По каналу прямого доступа могут осуществляться и единичные передачи, но они имеют мало смысла. Сигнал TC (terminal count) устанавливается контроллером DMA, когда начинается предпоследняя передача в блоке данных.

Сигналы CLK и OSC – тактовые частоты. Второй из этих сигналов не используется. Когда-то он был нужен для синхронизации телевизора, который мог использоваться вместо монитора. CLK (clock) – основной сигнал синхронизации магистрали. По стандарту имеет частоту 8 МГц, но на многих компьютерах устанавливался 12 МГц, 16 МГц и даже 20 МГц.

2.3. PCI и PCI Express

История. PCI (Peripheral Component Interconnect – взаимосвязь периферийных компонентов) – стандарт, разработанный Intel в 1991 г. и переданный в 1992 г. созданной по инициативе Intel организации, которая называется PCI SIG – PSI Special Interest Group, в обязанности которой входит уточнение и подготовка новых версий стандарта, публикация официальных документов и экспертиза предложений по модификации стандарта.

Мотивом для разработки новой шины расширения для персональных компьютеров была ликвидация узких мест в магистрали ISA и её расширениях. На первых этапах разработки обсуждалась возможность присоединения аппаратуры прямо к магистрали процессора, но этот ва-

риант был отвергнут, поскольку такие системы потеряли бы гибкость, присущую открытой архитектуре. В первоначальной версии PCI имела тактовую частоту 33 МГц и теоретическую пиковую скорость передачи 133 МБайт/сек.

Подлинную популярность PCI приобрела в версии 2.2, в которой была зафиксирована технология Plug and Play и добавлена тактовая частота 66 МГц и пиковая производительность в 64-разрядном варианте достигла 533 МБайт/сек. Определённый вклад в продвижение PCI внесла корпорация Microsoft, которая обеспечила поддержку возможностей этой магистрали на программном уровне (Windows 95). После этих изменений PCI стала использоваться не только в PC, но и в компьютерах с процессорами Alpha, MIPS, PowerPC, SPARC. В настоящее время вытесняется последовательным вариантом этой магистрали – PCIE (PCI Express), в котором сохранена модель обмена PCI, но на физическом уровне используются более производительные последовательные каналы передачи.

Терминология. Для того чтобы понимать назначение сигналов магистрали PCI, необходимы несколько терминов для обозначения её свойств, отсутствовавших у магистрали ISA и её аналогов.

Мультиплексирование. Многие сигналы информационной магистрали являются взаимоисключающими в том смысле, что они не могут появляться на линиях магистрали одновременно. В этих случаях естественно использовать одни и те же физические линии для передачи разных сигналов. Значение сигналов на мультиплексируемых линиях PCI определяется фазой передачи. Различают фазу адреса и фазу данных.

Взрывная мода (burst mode) – режим блочного обмена информацией, когда адрес передаётся только в начале передачи блока, а в дальнейшем автоматически инкрементируется контроллером памяти или устройством. Передача начинается с фазы адреса, за которой следует несколько фаз данных. Их количество определяется командой.

Транзакция. На магистрали ISA обмен с устройствами и памятью имел различный протокол – для памяти просто отсчитывались такты, а для устройств магистраль ожидала подтверждения от устройства. По стандарту PCI эти обмены унифицированы и подтверждение (“квитирование”) обязательно для любых обменов. Это правило и отражено в термине “транзакция”. Мы будем использовать также термин “передача”.

Контроль четности. Добавление всего двух линий на магистрали позволяет сильно снизить вероятность ошибки в передачах. Устройство-передатчик информации, помимо слова данных, передаёт чётность этого слова ($p = \bigoplus_{i=0}^{i=n} b_i$). Устройство-приемник, получив слово, вычисляет

четность принятого слова и сравнивает её с четностью, переданной передатчиком. В случае несовпадения генерируется сигнал ошибки четности.

Если доступ к системной памяти идет через магистраль PCI, то необходимо знать, не хранятся ли требуемые данные в кэшах процессора. Цикл определения наличия данных в кэше называют *snooper cycle*.

Пространство конфигураций – блоки памяти, стоящие “на борту” PCI-устройства и несущие информацию об устройстве в стандартной форме.

Сигналы и порядок обмена. Для обработки данных, адресации, управления интерфейсом, арбитража и некоторых системных функций интерфейсу PCI требуется, как минимум, 47 выводов для целевого устройства и 49 выводов – для задатчика. В таблице 2.4 показаны функциональные группы выводов: слева указаны необходимые выводы, а справа – необязательные.

Функц. группа	Обяз. линии	Функц. группа	Необяз. линии
Адрес и данные	AD00-31, C/BE0-3, PAR	Адрес и данные	AD32-63, C/BE4-7, PAR64
Управление интерф.	Frame, Stop IRDY, TRDY DevSel, IDSel	Управление интерф.	Lock
Сообщения об ошибках	PErr, SErr	Прерывания	INTA, INTB, INTC, INTD
Арбитраж master only	Req, Gnt	Управление КЭШем	SBQ, SDONE
Системные линии	Clk, Rst	JTAG	TDI, TDO, TCK, TMS, TRST

Таблица 2.4. Названия сигналов магистрали PCI

Системные выводы. CLK (тактовая частота) обеспечивает синхронизацию всех транзакций на PCI и является входным для каждого PCI-устройства. Все другие сигналы PCI, за исключением перезагрузки и прерываний, привязаны к фронту CLK. PCI может тактироваться любой частотой от 0 Гц до 33 МГц. В версии 2.2 PCI допустима любая частота до 66 МГц.

RST (сброс) используется для приведения регистров и секвенсоров (от слова sequencer – генератор последовательностей) PCI-устройств к стандартному состоянию.

Адресные выводы и выводы данных. AD[31::00] Адрес и данные мультиплексированы на одних и тех же выводах PCI. Транзакция шины состоит из фазы адреса, сопровождаемой одним или большим количеством фаз данных. PCI поддерживает как чтение, так и запись блоками (Burst mode). В течение фазы адреса в AD00-31 содержится физический адрес (32 бита). При вводе-выводе это адрес байта, при обращениях к памяти и пространству конфигураций – адрес двойного слова (DWORD). Когда идут фазы данных, AD[07::00] содержит младший значащий байт (lsb), а в AD[31::24] содержится старший значащий байт (msb). Передаваемые данные «устойчивы» и правильны, когда активен сигнал IRDY, а читаемые данные «устойчивы» и правильны, когда активен TRDY. Передача происходит, когда активны оба сигнала IRDY и TRDY.

C/BE[3::0] Bus Command/Byte Enable («команды шины / разрешение байта») мультиплексированы на одних выводах PCI. Во время фазы адреса C/BE[3::0] определяет команду шины. Команды шины приведены в таблице 2.5.

0000	Interrupt Acknowledge
0001	Special Cycle
0010	I/O Read
0011	I/O Write
0100	Зарезервировано
0101	Зарезервировано
0110	Memory Read
0111	Memory Write
1000	Зарезервировано
1001	Зарезервировано
1010	Configuration Read
1011	Configuration Write
1100	Memory Read Multiple
1101	Dual Address Cycle
1110	Memory read Line
1111	Memory Write and Invalidate

Таблица 2.5. Команды магистрали PCI

В фазе данных C/BE[3::0] используется в качестве Byte Enable. Byte Enable действителен для всей фазы данных и определяет, какие байты передаваемого двойного слова значимы. C/BE0 соответствует LSB, а C/BE 3 соответствует MSB.

PAR Parity – это контроль четности по линиям AD00-31 и C/BE0-3. Сигнал PAR верен в течение одного такта после фазы адреса. Для фаз данных PAR верен в течение такта после того, как будет активен IRDY при записи или TRDY – при чтении. Сигналом PAR управляет задатчик шины в фазе адреса и в фазе данных при записи; целевое устройство управляет сигналом PAR в фазе данных при чтении.

Интерфейсные управляющие выводы. FRAME Cycle Frame (кадр) управляется текущим задатчиком для указания начала и продолжительности передачи. FRAME становится активным, в начале транзакции. Пока FRAME активен, идет передача данных. Когда сигнал FRAME становится неактивным, транзакция переходит в заключительную фазу данных.

IRDY (Initiator Ready) и TRDY (Target Ready) указывают готовность инициатора и целевого устройства и используются совместно. Если активны оба сигнала, то первый же фронт сигнала CLK осуществляет транзакцию. Циклы ожидания вставляются до тех пор, пока не станут активны оба сигнала.

STOP показывает, что текущее целевое устройство посылает задатчику запрос на останов текущей транзакции.

LOCK показывает элементарную операцию, которой для завершения требуется множество транзакций.

IDSEL (Initialization Device Select) – выбор устройства инициализации, используется для выбора кристалла при чтении и записи конфигурации.

DEVSEL (Device Select) – выбор устройства. Показывает, что устройство дешифровало свой адрес как целевое устройство.

Арбитражные выводы (только для мастеров шины). REQ (Request) – запрос. Показывает арбитру, что данному агенту требуется использование шины.

GNT (Grant) – разрешение показывает агенту, что ему разрешен доступ к шине.

Выводы для сообщения об ошибках. PERR (Parity Error) – ошибка контроля четности. Предназначен только для сообщений об

ошибках контроля четности во всех транзакциях PCI, за исключением специального цикла (Special Cycle). Вывод PERR управляется агентом, получающим данные в течение двух тактов, после того, как обнаружена ошибка контроля данных по четности.

SERR (System Error) – системная ошибка. Предназначен для выдачи сообщений об ошибках контроля четности адреса, данных по команде Special Cycle или любых других катастрофических системных ошибках.

Выводы прерывания (необязательно). Сигналы INTx асинхронны по отношению к CLK. PCI предусматривает одну линию прерываний для устройства с одной функцией и до четырёх линий прерывания – для многофункциональных устройств. Для однофункционального устройства может использоваться только линия INTA, в то время как три других линий прерывания не имеют никакого значения.

INTA – INTD (Interrupt A – Interrupt D) используется для запроса прерывания. Любая функция на многофункциональном устройстве может быть соединена с любой линией INTx.

Выводы поддержки кэша (необязательно). Выводы поддержки кэша присутствовали только в первых версиях магистрали PCI. Позднее, когда кэши второго уровня стали частью процессора, нужда в этих линиях отпала. SBO Если активен Snoor Backoff, то имеется кэш-попадание. SDONE Snoor Done указывает состояние проверки кэша. Когда сигнал неактивен, то это показывает, что результат все ещё ожидается. Когда сигнал активен, то это показывает, что проверка завершена.

Выводы расширения шины до 64-бит (необязательно). AD[63::32]. Address и Data (адрес и данные) мультиплексированы на одних и тех же выводах и обеспечивают 32 дополнительных разряда. В течение фазы адреса передаются старшие 32 бита 64-разрядного адреса; в противном случае, эти биты резервные, но при этом они устойчивы и не определены. В течение фазы данных, когда активны REQ64 и ACK64, передаются дополнительные 32 бита данных.

C/BE [7::4]. Bus Command/Byte Enables в течение фазы адреса передаёт фактическую команду шины по линиям C/BE[7::4]; в противном случае, эти биты зарезервированы и не определены. В течение фазы данных по линиям C/BE[7::4] передаётся байт, который показывает, какой байт содержит значимые данные, при условии, что активны оба сигнала REQ64 и ACK64. Сигнал C/BE[4] применяется к байту 4, а C/BE[7] применяется к байту 7.

REQ64. Когда Request 64-bit Transfer управляется текущим инициатором шины, то он показывает, что тот желает передать данные, используя для пересылки 64 бита.

ACK64. Когда Acknowledge 64-bit Transfer управляется устройством, которое успешно дешифрировало данный адрес в качестве агента текущего доступа, то он показывает, что агент желает передать данные, используя при этом 64 бита. ACK имеет такие же временные параметры, как и DEVSEL. PAR64

PAR64. Parity Upper DWORD – это бит контроля четности для линий AD[63::32] и C/BE[7::4].

Порт JTAG периферийного сканирования (boundary scan). Стандарт IEEE 1149.1, Порт для тестирования и архитектура периферийного сканирования («Test Access Port and Boundary Scan Architecture»), включен в качестве необязательного интерфейса для PCI устройств. Включение в состав устройства порта для тестирования (TAP – Test Access Port) позволяет использовать периферийное сканирование для проверки PCI-устройств и материнской платы, на которой они установлены. TAP состоит из пяти выводов, которые используются для организации последовательного интерфейса с контроллером TAP внутри PCI-устройства.

- TCK in Test Clock используется для синхронизации ввода собранной информации и данных в устройство и их вывода во время работы с TAP.
- TDI Test Data Input используется для последовательного ввода в устройство тестирующих данных и команд при работе с TAP.
- TDO Test Output используется для последовательного вывода тестирующих данных и команд из устройства при работе с TAP.
- TMS Test Mode Select используется для управления в устройстве состоянием контроллера TAP.
- TRST Test Reset обеспечивает асинхронную инициализацию контроллера TAP. Этот сигнал по стандарту IEEE 1149.1 необязателен.

Основные команды шины PCI.

1. Подтверждение прерывания (0000). Контроллер прерываний автоматически опознает и реагирует на команду INTA (interrupt acknowledge). На фазе данных он передает (выставляет) вектор прерывания на линии адреса/данных.

2. Специальный цикл (0001). Значение информации на шине адреса/данных при передаче команды специальный цикл указано в таблице.

AD15-AD0	
0x0000	Processor Shutdown
0x0001	Processor Halt
0x0002	x86 Specific Code
0x0003 to 0xFFFF	Reserved

3. I/O Read (0010) и I/O Write (0011). Операция ввода-вывода с PCI-устройством. На линиях адреса/данных одновременно с этой командой передается адрес порта PCI-устройства. Линии AD0 и AD1 обязательно должны декодироваться устройством (нельзя использовать линии C/BE). Порт PCI-устройства может быть 8- или 16-разрядным. Пространство конфигураций также может быть доступным через обмен с устройствами. Это можно сделать через порты ввода/вывода 0x0CF8 (Address) и 0x0CFC (Data). Запись в порт адреса должна быть первой.

4. Memory Read (0110) и Memory Write (0111). Чтение или запись в системную память. Линии адреса/данных содержат двойное слово адреса. Линии AD0 и AD1 могут не декодироваться, требуемый байт указан на линиях C/BE.

5. Configuration Read (1010) и Configuration Write (1011). Чтение или запись в конфигурационное пространство PCI устройства, которое имеет стандартную длину в 256 байт (точнее 64 двойных слова). В зависимости от того, что содержат AD0-1, остальные линии адреса/данных имеют разное значение. Если в AD0-1 стоят два нуля, то предполагается, что PCI устройство уже выбрано сигналом IDSEL и тогда AD2-7 содержат индекс двойного слова, AD8-10 – номер адресуемой функции (одного из восьми устройств, которые могут присутствовать на карте PCI). Если в AD0-1 стоят 01, то адрес интерпретируется сложнее: AD2-7 – слово, AD8-10 – функция, AD11-15 – устройство, AD16-23 – шина. Таким образом, при втором способе обращения к пространству конфигураций можно обращаться к одному из 32 устройств на каждой из 256 шин PCI, которые могут присутствовать в системе. Стандартное расположение информации в пространстве конфигурации показано в таблице. Обмен производится двойными словами.

Много информации о PCI устройствах и их изготовителях можно найти в Internet ¹

Информация из пространства конфигураций отображается диспетчером устройств, правда, эта техническая информация сильно разбавлена сведениями о программном обеспечении, обслуживающем устройство (драйверы).

¹См., например: www.pcidatabase.com; <http://pci-ids.ucw.cz/>

Addr.	32<Bit>24	23<Bit>16	15<Bit>8	7<Bit>0
00	Device ID		Vendor ID	
04	Status		Command	
08	BaseClass Code	SubClass Code	Prog. Interface	Revision ID
0C	BIST	Header	Latency timer	CacheLine Size
10-24	Base Address Register 0-5			
28	Cardbus CIS pointer			
2C	Subsystem ID		Subsystem Vendor ID	
30	Expansion ROM Base Address			
34	Reserved			
38	Reserved			
3C	MaxLat	MnGNT	INT-pin	INT-line
40-FF	Available for PCI unit			

Таблица 2.6. Стандартные поля в конфигурации PCI устройства

6. Multiple Memory Read (1100). Эта команда является расширением обычного чтения памяти, она используется для чтения больших блоков информации без генерации адресов и кэширования. На шине адреса/данных содержится адрес начала блока и пока держится FRAME идет передача двойных слов, расположенных одно за другим в памяти.

7. Dual Address Cycle (1101). Эта команда позволяет генерировать 64-разрядный адрес при наличии только 32 линий в шине адреса/данных. Собственно тип передачи указывается во втором цикле адреса. Недоступен для систем на процессорах Intel.

8. Memory-Read Line (1110). Этот тип передачи используется для чтения памяти строками кэша (типично 32 байта).

9. Memory Write and Invalidate (1111). По этой команде в память пишется строка кэша, а в самом кэше соответствующая строка объявляется неверной (грязной) в том смысле, что она не соответствует аналогичной строке в основной памяти.

2.4. Шины расширения в других архитектурах

Далеко не все старые компьютеры основывались на магистрально-модульной архитектуре или имели единую шину расширения. Одной из первых шин расширения, ставшей промышленным стандартом, бы-

ла магистраль Unibus, которая легла в основу легендарных PDP-11 и ранних VAX компьютеров. Она использовалась в изделиях Digital Equipment Corporation (DEC). Магистраль была разработана около 1969 года Gordon Bell и студентом Harold McFarland. Unibus имела 18-разрядную шину адреса, 16-разрядную шину данных, линии прерываний, прямого доступа к памяти, контроля четности, сообщений о низком уровне питающих напряжений, нескольких линий синхронизации и другие линии общим числом 56 (без питаний). Память устройств отображалась в единое пространство адресов.

Позднее DEC использовала магистраль VAXBI (VAX Bus Interconnect). Эта магистраль была полностью 32-разрядной и мультиплексированной. На магистрали поддерживался арбитраж, и любое устройство могло быть задатчиком.

Самые мощные суперкомпьютеры (1976-1990) CRAY-1 и CRAY-2 не имели единой информационной магистрали, подчиненной какому-либо стандарту.

Первый персональный компьютер Altair 8800, построенный в 1974 году Ed Roberts (по легенде – в гараже), использовал для объединения устройств шину S-100, которая по существу продолжала выводы процессора Intel 8080. Магистраль имела две однонаправленных 8-разрядных шины данных и одну двунаправленную 16-разрядную шину адреса. Успех Altair 8800 привел к тому, что S-100 использовалась в клоне Altair-подобных машин и была стандартизована в 1983 г. (IEEE-696). Успех был так велик, что разработчики процессоров делали процессоры совместимыми по выводам с i8080, чтобы имелась возможность устанавливать их в S-100 машины (наиболее известный пример Zilog Z80). Эта волна пошла на убыль только начиная с 1981 г. после появления IBM PC на магистрали ISA8.

IBM, потерпев некоторую неудачу с ISA8, сделала ставку на новую магистраль MicroChannel – MC, которая была лицензирована, и IBM имела на неё полные авторские права. Но техническая политика, выбранная фирмой, оказалась более чем странной. На основе этой магистрали начиная с 1987 года выпускались компьютеры IBM PS/2. Конечно, MC была более совершенна даже по сравнению с ISA16, но все погубили закрытые стандарты. На магистрали MC сразу была реализована технология Plug and Play, шина была 32-разрядной, теоретическая пропускная способность 66 МБ/сек (реально 40 МБ/сек), полный арбитраж – любое устройство могло стать задатчиком и существовала возможность передач от одного устройства к другому. IBM активно противодействовала распространению этой магистрали. За право производить устройства для MC требовалось платить нереально большие деньги, тех-

нические решения не раскрывались в литературе. В результате уже упомянутая группа компаний (её называли “бандой девяти”) разработала и опубликовала стандарт EISA, который и похоронил MicroChannel (правда, и сам прожил недолго). В 1996 году IBM прекратила игру на рынке персональных компьютеров.

Магистраль NuBus была разработана в 1970-х годах в МИТ. Использовалась в компьютере Western Digital (NuMashine), а впоследствии Texas Instruments (NuMashine, Explorer), позднее использовалась в Macintosh II и других моделях. NuBus – 32-разрядная мультиплексированная магистраль с простым протоколом обмена, рассчитанная на четыре устройства расширения.

Глава 3.

Процессор

3.1. Архитектура Фон–Неймана

История. В основе архитектуры Фон–Неймана лежит идея хранимой программы. Программа ничем не отличается от данных и считывается из памяти, как и данные, которые она обрабатывает. Хотя идею хранимой программы можно усмотреть ещё в понятии универсальной машины Тьюринга (UTM), которая описана в фундаментальной статье “On computable numbers with an application to the entscheidungsproblem”, опубликованной Аланом Тьюрингом в *Proceedings of London Mathematical Society* в мае 1936 года, обычно родоначальником всех современных архитектур называют Джона фон Неймана. Его работа “First Draft of a Report on the EDVAC” (Набросок отчета по теме EDVAC) официально не публиковалась, поскольку касалась секретных проектов ENIAC и EDVAC, но рукопись была скопирована по инициативе офицера, отвечавшего за режим секретности (!!), и разослана в конце июня 1945 года по почте некоторым специалистам, участвовавшим в проекте (в том числе и за пределами США)¹. Машины, чем-то похожие на EDVAC, который был закончен 1952 году, уже существовали в Великобритании (Colossus Mk1 1943 г. и Colossus Mk2 1944 г., Baby 1948 Манчестер – первая в мире построенная машина с хранимой программой), в Германии (V-1 - Z-1 1938 г., Z-2 1939 г., Z-3 1941 г., Z-4 1948 г., созданные практически в одиночку инженером Конрадом Цузе), в СССР (МЭСМ С. А. Лебедева, 1950 год). Но именно в американской машине был впервые предложен важный принцип, сделавший EDVAC прототипом универсальных вычислительных машин, – принцип хранимой программы.

¹Подробности этой истории можно прочитать в Википедии: URL:<http://en.wikipedia.org/wiki/First-Draft-of-a-Report-on-the-EDVAC>.

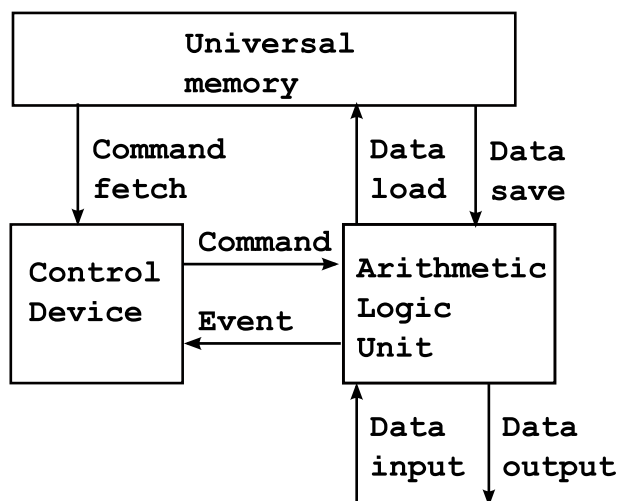


Рис. 3.1. Устройства и связи в компьютере фон Неймана

Архитектура фон Неймана. Устройства, показанные на рис. 3.1, реализуют идею архитектуры фон Неймана. Компьютер состоит из

- устройства управления – control device;
- арифметико-логического устройства (АЛУ) – arithmetic-logic unit (ALU);
- системной памяти – system memory;
- устройства ввода/вывода – input/output device (I/O device).

Порядок работы следующий. В двух областях памяти хранятся:

- программа – двоичные коды, которые интерпретируются как инструкции,
- и данные – коды, которые интерпретируются как двоичные числа или символы.

В дальнейшем эти области памяти получили названия сегмента кода (code segment) и сегмента данных (data segment). Устройство управления, которое содержит указатель следующей инструкции (next instruction pointer или просто IP), выполняет однотипные действия.

- Выбирает (fetch) инструкцию в соответствии с адресом в указателе инструкций и увеличивает адрес в указателе инструкций так, чтобы он указывал на следующую инструкцию в программе.
- В соответствии с полученной инструкцией генерирует машинный код, передаваемый в АЛУ (decode). Этот код определяет операцию, выполняемую АЛУ, и адреса данных, которые требуются для её выполнения. Адреса, с которых загружаются данные и сохраняется результат являются частью инструкции.

- АЛУ загружает (load) данные из сегмента данных в соответствии с указанными в инструкции адресами.
- АЛУ выполняет (execute) инструкцию.
- АЛУ сохраняет (save) результат в память.

Заметим, что последовательность `fetch – decode – load – execute – save` скорее всего и следует считать той единственной программой, которую выполняет любой компьютер.

Из приведённой последовательности действий (трехадресная машина) могут быть исключены операции загрузки данных (операндов) и сохранения результатов, но тогда в наборе инструкций должны появиться отдельные инструкции загрузки и сохранения и должно быть определено место, куда помещается результат по умолчанию (двухадресная машина).

Безусловно, эти последовательности, способ их исполнения и устройства, составляющие компьютер в разных архитектурах, претерпевают некоторые изменения, но от этого не изменяются принципы построения компьютера.

- 1) Принцип двоичного кодирования. Для представления данных и команд используется двоичная система счисления.
- 2) Принцип однородности памяти. Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления — чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.
- 3) Принцип адресуемости памяти. Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка; память внутренняя.
- 4) Принцип последовательного программного управления. Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой, в последовательности, определяемой программой.
- 5) Принцип жесткости архитектуры. Неизменяемость в процессе работы топологии, архитектуры, списка команд.

Эти принципы были опубликованы в работе Burks A.W., Goldstine H.H., Neumann J. Preliminary Discussion of the logical Design of an Electronic Computing Instrument в 1946 году и называются принципами

фон Неймана. С очень малыми отклонениями, касающимися однородности памяти (регистры процессора и иерархия кэшей в этот принцип не укладываются), архитектура всех современных компьютеров подчинена этим принципам.

3.2. Суперскалярный процессор

Конвейеризация (Pipelining). Выполнение каждой инструкции программного кода всегда требует нескольких шагов. Инструкцию необходимо:

- выбрать из памяти (fetch),
- декодировать (decode),
- вычислить адреса источников (source addr. calc.),
- загрузить операнды (load),
- собственно выполнить (execute),
- вычислить адрес приемника (destination addr. calc.),
- записать результат (save).

Эту последовательность действий можно развернуть во времени, полагая для простоты, что каждое действие занимает один такт (таблица 3.1).

clock num.	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
fetch	i1							i2		
decode		i1							i2	
source addr.			i1							i2
load				i1						
execute					i1					
dest. addr.						i1				
save							i1			

Таблица 3.1. Последовательность действий трехадресной машины фон Неймана

При обычном порядке выполнения инструкции на каждый этап уходит некоторое время, а устройства, реализующие каждый шаг в этой последовательности, большую часть времени стоят. Основной идеей конвейера является такая организация выполнения потока инструкций, при

которой устройства не простаивают, а выполняют свою часть обработки следующей инструкции, не дожидаясь выполнения предыдущей. Сказанное можно проиллюстрировать следующей схемой (таблица 3.2).

clock num.	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
prefetch	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
fetch		i1	i2	i3	i4	i5	i6	i7	i8	i9
decode 1			i1	i2	i3	i4	i5	i6	i7	i8
decode 2				i1	i2	i3	i4	i5	i6	i7
execute					i1	i2	i3	i4	i5	i6
writeback						i1	i2	i3	i4	i5

Таблица 3.2. Целочисленный конвейер Pentium MMX

В этой схеме предполагается, что каждый этап выполнения инструкции выполняется за один такт процессора, и инструкции требуют одинакового числа этапов выполнения. Преимущества очевидны, за один такт выполняются четыре инструкции. Но на пути реализации этой схемы возникает много препятствий.

Зависимость по данным. Рассмотрим фрагмент программного кода

- 1) mov r3 r7
- 2) ld r8 (r3)
- 3) add r3 r3
- 4) ld r9 (r3)
- 5) sub r9 r8.

Понятно, что инструкция 2 не может быть выполнена, пока не будет выполнена инструкция 1 (не загружен адрес), инструкцию 3 нельзя выполнить вместе с инструкцией 2, т. к. модифицируется r3, который нужен инструкции 2, и т. д. Эта ситуация называется зависимостью по данным. Сама по себе она неизбежна, но для функционирования конвейера конечной глубины и не требуется её устранять полностью, нужно только сделать независимыми несколько ближайших инструкций для поддержания конвейера. Все зависимости по данным, формально являющиеся препятствием для работы конвейера, распадаются на 4 типа:

- 1) RAW (read after write) – чтение после записи;
- 2) RAR (read after read) – чтение после чтения;
- 3) WAW (write after write) – запись после записи;
- 4) WAR (write after read) – запись после чтения.

RAR – фактически не является зависимостью. WAW и WAR происходят от бедности – следующая инструкция не обязательно должна писать туда же, куда писала, или читать то, что читала предыдущая инструкция.

Переименование регистров. Эти зависимости устраняются простым (естественно только на словах) приемом, который называется переименование регистров. Для этой цели можно, например, различать физические регистры и логические, и в ситуациях WAW и WAR делать соответствующую замену в инструкции. Естественным образом необходимо иметь механизм, восстанавливающий правильный порядок. Необходимым элементом этого механизма является таблица псевдонимов – *alias table*. Существует несколько вариантов реализации этой идеи. Единственной принципиальной зависимостью является RAW. Идея решения этой проблемы состоит в том, чтобы, обходя ситуации WAW и WAR, переупорядочить инструкции так, чтобы отодвинуть выполнение второй из инструкций, находящихся в RAW-зависимости на «безопасное расстояние», т. е. на момент времени, когда читаемый операнд будет готов. Чтобы иметь возможность переупорядочивать инструкции, процессору необходимо «видеть» хотя бы несколько ближайших инструкций. Эта область программного кода называется «окном просмотра», физически окно просмотра реализуется в буфере переупорядочения (ReOrder Buffer - ROB). Процессоры, содержащие избыток регистров и самостоятельно переупорядочивающие поток инструкций, называют суперскалярными. Эта идеология лежит в основе IA-32.

VLIW-процессоры. Задачу переупорядочения инструкций можно возложить как на процессор, так и на компилятор. В последнем случае потенциально окном просмотра может быть весь код. Процессоры, ориентированные на этот подход, называют VLIW-процессоры (VLIW – very long instructive word). В этом случае в машинном коде содержатся инструкции для каждого из параллельно работающих устройств, а в процессоре нет никаких устройств для переупорядочения. VLIW-процессорами являются Itanium 2 (Intel), Trimedia (Philips), Crusoe (Transmeta), многоядерные процессоры TILE64, TILE-Gx (Tilera), российские Эльбрус 2000, Эльбрус S (МЦСТ).

Ветвления. Ветвления (условные переходы) также представляют собой препятствие для конвейерного выполнения кода. В этом случае от результата вычислений зависит, какой из фрагментов кода будет выпол-

няться, и, естественно, необходимо дождаться, когда этот результат «поспел» – опять простой (stall). Обычно поступают следующим образом, при первом прохождении ветвления предполагается, что:

- безусловный переход выполняется,
- условный переход вперед не выполняется,
- условный переход назад выполняется.

Далее решения принимаются на основании статистики результатов прохождения точки ветвления. Эту идею реализует так называемый буфер предсказания ветвлений (ВТВ – branch target buffer), который помнит вхождения в ветвления и принятые при этом решения. Каждое следующее решение принимается на основании статистики по предыдущим решениям.

Переупорядочение. Переупорядочение реализуется при помощи устройства, которое имеет несколько названий: станция резервации (reservation station), буфер переупорядочения (reorder buffer – ROB), пул инструкций (instruction pool), есть и другие названия. Инструкции в виде записей определенной структуры загружаются в ROB, и по мере прохождения этапов конвейерной обработки “обрастают” отметками. После готовности к исполнению они поступают на одно из исполнительных устройств (АЛУ) и после выполнения отмечаются как выполненные. Таким образом, порядок исполнения определяется не программным кодом, а готовностью инструкции к исполнению. Такой стиль исполнения называется внеупорядоченным (out of order). Не следует путать с неупорядоченным исполнением. Переупорядочение – это главный инструмент, при помощи которого поддерживается конвейер.

3.3. Процессоры IA-32 и Intel 64

Мы будем обсуждать узлы IA-32 процессора и связи между ними только с точки зрения реализации суперскалярной архитектуры, отвлекаясь от таких подробностей, как наличие и параметры того или иного устройства у конкретной модели. Поэтому обсуждаемые схемы соответствуют всем новейшим Intel-процессорам, а возможно, и ни одному из них. Чтобы внести хоть какую-то определённости, скажем, что всё, о чем говорится ниже, относится к процессорам старше Pentium Pro. Этот процессор в ряду изделий фирмы Intel был «гадким утенком» (страшно дорогой, работающий хуже первых Pentium на 16-разрядных приложениях), но именно он открывает семейство P6 и в своей структуре со-

держит практически все элементы PII, PIII и PIV. Элементы, присущие самым старшим моделям (PIV и далее), будут отмечаться особо.

Наиболее подробно мы опишем архитектуру семейства P6, включавшего в себя Pentium Pro, Pentium II, Pentium II Xeon, Celeron, Pentium III и Pentium III Xeon. К сожалению, позднее корпорация Intel прекратила публикацию столь подробных документов. В самом грубом приближении структуру процессора можно представить следующим образом (см. рис. 3.2).

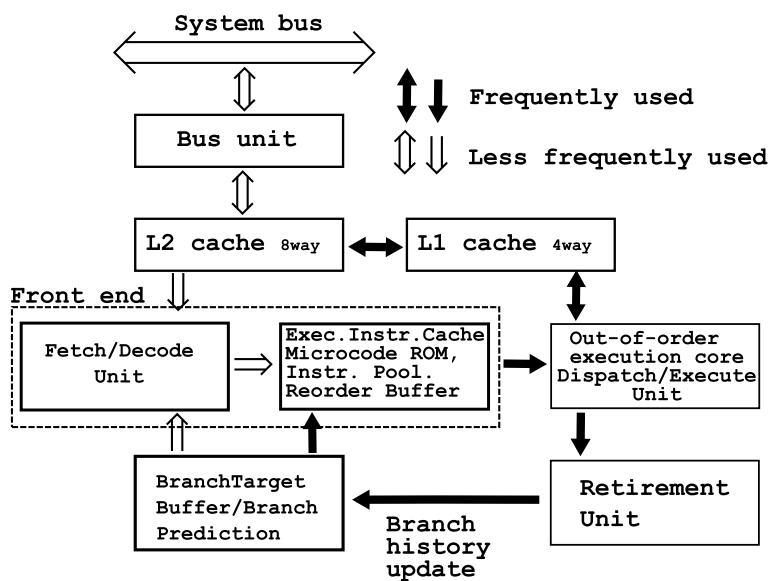


Рис. 3.2. Структура процессоров семейства P6

Эта схема не отражает полностью устройства, входящие в процессор, но позволяет наглядно представить потоки инструкций и данных.

Поток инструкций выбирается из кэша инструкций, декодируется и направляется в пул инструкций. Из пула инструкции могут выбираться в том порядке, который обеспечивает параллельное функционирование исполнительных устройств. Выполнение инструкций так или иначе сводится к чтению данных из кэша и записи результатов выполненных операций в кэш. Сообразно с этой логической схемой мы можем выделить в составе процессора следующие устройства:

- подсистему памяти (Memory subsystem);
- устройство выборки/декодирования (Fetch/decode unit);
- пул инструкций (Instruction pool);
- исполнительно-распределительное устройство (Dispatch/execute unit);
- завершающее устройство (Retire unit).

Подсистема памяти (Memory Subsystem). Подсистема памяти содержит:

- системную шину (system bus),
- кэш второго уровня (L2),
- шинный интерфейс (bus interface unit),
- кэш инструкций (L1 instructions),
- кэш данных (L1 data),
- устройство интерфейса памяти (memory interface unit),
- буфер переупорядочения обращений к памяти (memory reorder buffer).

Системная шина фактически находится за пределами собственно процессора и является для процессора ближайшим «внешним миром». Системная шина процессора (host bus) – это информационная магистраль, которая связывает процессор с другими процессорами, которые могут присутствовать в системе, мостом, ведущим к устройствам ввода/вывода и кэшам второго уровня. Частота обращений к кэшу второго уровня в процессорах старших моделей значительно возросла по сравнению с частотой обращений к устройствам ввода/вывода, и поэтому в отличие от старых Pentium и ещё более ранних процессоров (семейство P5), у которых интерфейс процессора был единым, в первых процессорах семейства P6 использовалась так называемая двойная независимая шина (Dual Independent Bus). Со всеми устройствами ввода/вывода (включая основную память) связь осуществляется через фасадную шину (Front Side Bus), а связь с кэшем второго уровня – через внутреннюю шину (Local Bus). Процессоры этих времен (Pentium II) изготавливались на платах вместе с кэшем L2, который был внешним и северным мостом. Такая организация позволяла значительно разгрузить внешнюю шину. Позднее в процессорах Pentium III кэш второго уровня перекочевал на чип процессора, а северный мост вернулся на материнскую плату, процессоры снова приобрели привычный формфактор.

Структура системной шины процессоров Intel напоминает структуру шин расширения (ISA, PCI), но имеет значительно больше линий и существенно более сложный протокол. Она, как и шина PCI, ориентирована на транзакции (запрос – подтверждение) и поддерживает пакетную (без генерации каждого адреса) передачу данных. Передача данных на системной шине конвейеризована. Каждая транзакция состоит из нескольких фаз: арбитраж, запрос, ошибка, завершение, ответ, данные. В этой ситуации, как мы видели ранее (конвейер процессоров P5 и P6), возможна такая организация обмена, когда на каждом такте шины

на её линиях присутствуют различные фазы соседних транзакций, что значительно повышает производительность шины.

Все сигналы процессора организуются шинным интерфейсным устройством (Bus Interface Unit). Устройство интерфейса памяти одновременно является одним из исполнительных устройств и предназначено для генерации адресов (запросы к памяти). Буфер переупорядочения обращений к памяти предназначен для организации выполнения запросов к памяти в переупорядоченной последовательности. Чтения могут происходить не в той последовательности, которая указана в исходном коде, и готовность данных к записи наступает вне порядка, установленного кодом. Буфер поддерживает правильный порядок записей (независимо от момента готовности).

Устройство выборки/декодирования (Fetch/decode unit). В состав устройства выборки/декодирования входят:

- устройство выборки инструкций (Instruction fetch unit),
- буфер направлений ветвления (branch target buffer),
- декодер инструкций (instruction decoder),
- генератор последовательностей микрокода (microcode sequencer),
- таблица псевдонимов регистров (register alias table).

Для поддержания работающего конвейера необходимо, чтобы всегда имелся запас «готовых к употреблению» инструкций, из которых могут выбираться те, которые могут быть исполнены. Поддержание пула инструкций является задачей, которую решает устройство выборки/декодирования. В этом устройстве происходит превращение потока кодов из кодового сегмента в последовательность микроопераций (uops). Микрооперации представляют собой примитивные двухадресные операции (op destination source). Совокупность этих микроопераций является истинным набором инструкций, выполняемых процессором, и называется RISC-ядром процессоров x86. В командах стандартного набора инструкций имеется большое количество неявных действий процессора, связанных, например, с вычислением адресов, сложными строковыми инструкциями, вычислением функций и т. д. Иногда количество микроопераций, необходимое для реализации одной стандартной инструкции, достигает нескольких десятков. Все стандартные инструкции процессора распадаются на две группы:

- простые (1 – 4 микрооперации),
- сложные (более 4 микроопераций).

В зависимости от принадлежности к одной из этих групп декодирование производится различными способами:

- простые инструкции обрабатываются декодером (decoder) по определенным правилам;
- сложные инструкции заменяются готовыми последовательностями микроопераций, хранящимися в ПЗУ (или ПЛМ) генератора последовательностей микрокода (microcode sequencer).

Выборка инструкций происходит следующим образом. Устройство выборки за один такт читает строку кэша инструкций (32 байта), просматривает адреса, фигурирующие в ней, метит начало и конец инструкций в первых 16 байтах этой строки и передаёт эту группу инструкций в декодер. Устройство выборки также вычисляет значение указателя инструкции (Next IP), основываясь на информации буфера направлений ветвления, состояния прерывания при исполнении и одного из целочисленных АЛУ в случае промаха предсказания ветвления. Наиболее важным элементом этого процесса является предсказание ветвлений. Устройство предварительной выборки агрессивно выдирает из исходного кода неветвящуюся последовательность инструкций, запоминая те места, где может произойти ветвление в буфере направлений ветвления. Буфер хранит 512 записей, состоящих из адресов ветвлений и четырёх битов предыстории прохождения данного ветвления за четыре последних прохода. На основании этой информации принимается решение о том, будет или не будет ветвление на данном шаге. Базовые правила, по которым принимается решение при первом вхождении в точку ветвления, следующие:

- безусловный переход всегда выполняется;
- ветвление назад выполняется;
- ветвление вперед не выполняется.

Устройство декодирования содержит три параллельных декодера — два для простых инструкций и один для сложных инструкций. Последний просто выдаёт адрес сложной инструкции в ПЗУ генератора последовательностей микрокода (microcode sequencer). За один такт устройство декодирования может выдать до шести микроопераций (по одной с каждого декодера простых инструкций и четыре с декодера сложных инструкций). Для устранения зависимостей по данным декодированные инструкции проходят переименование регистров, которое запоминается в таблице псевдонимов. Для фактического исполнения команд процессоры семейства R6 могут предоставить 40 регистров. В результате всех действий устройства выборки/декодирования возникает поток микроопераций в порядке, фиксированном исходным кодом, но неветвящийся и без зависимостей по данным.

Пул инструкций (Instruction pool). Пул инструкций хранится и обрабатывается в буфере переупорядочения инструкций (ROB – ReOrder Buffer). В новых документах это устройство получило два новых названия Execution Instruction Cache или Microcode ROM (кэш исполняемых инструкций или память микрокодов). Переупорядочение происходит следующим образом. Пул инструкций (как бы он ни назывался) сканируется (просматривается) станцией резервации из исполнительно-распределительного устройства. Оно (исполнительно-распределительное устройство) выбирает те из инструкций, которые могут быть выполнены полностью или спекулятивно, если не хватает одного операнда, и отправляется (dispatch) в одно из исполнительных устройств. После выполнения инструкция метится как годная в отставку (retire). С другой стороны, пул инструкций сканируется устройством завершения (retirement unit), которое разыскивает выполненные инструкции, восстанавливает логические имена регистров и метит выполненные инструкции как негодные, тем самым удаляя их из пула.

Исполнительно-распределительное устройство (Dispatch/execute unit). В устройство входят:

- станция резервации (reservation station);
- целочисленные устройства (integer units);
- устройства для чисел с плавающей точкой (floating-point units);
- устройства генерации адресов (address generation units);
- в старших моделях эти устройства дополнены устройством SSE (Streaming SIMD Extension).

Функции планирования и распределения инструкций по исполнительным устройствам выполняет станция резервации, она непрерывно сканирует пул инструкций, выбирая те из них, которые готовы к выполнению, и направляя их на исполнение в соответствующее исполнительное устройство. Два целочисленных АЛУ могут выполнять две микрооперации одновременно, одно из этих устройств может выполнять ветвящиеся микрооперации. Последнее может обнаруживать неправильные предсказания ветвлений и сообщать об ошибке предсказания в буфер направления ветвлений для перезагрузки конвейера. Обнаружение ошибок происходит следующим образом. Каждая ветвящаяся инструкция «метится» двумя адресами перехода – предсказанным и отброшенным. При выполнении этой инструкции целочисленное АЛУ определяет, какое из направлений выбрано. Если выбрано предсказанное направление, то все спекулятивно выполненные инструкции метятся как годные, и выполнение продолжается. Если выбранное направление неправильно, то все

инструкции после ветвления метятся как негодные и тем самым удаляются из пула. Далее в буфер направления ветвления сообщается правильный адрес ветвления и начинается выборка инструкций с нового адреса. Устройство генерации адресов выполняет микрооперации загрузки и сохранения. Выполнение загрузки требует только адреса в памяти, поэтому оно кодируется одной микрооперацией, при сохранении требуются адрес и данные, поэтому оно кодируется двумя микрооперациями. Каждое из АЛУ плавающих чисел выполняет все операции с плавающими числами. При переключении процессора в режим выполнения команд MMX эти АЛУ переконфигурируются и начинают выполнять набор инструкций MMX. Функции SSE – устройства будут рассматриваться отдельно.

Устройство завершения (Retirement Unit). Устройство завершения восстанавливает логические имена регистров и удаляет инструкции из пула. Подобно станции резервации устройство завершения сканирует пул инструкций, отыскивая те из них, которые выполнены и не содержат зависимостей по данным. Оно завершает выполнение инструкций и удаляет их в порядке, определяемом исходным кодом, учитывая прерывания, исключения, точки останова и ошибки предсказания ветвлений.

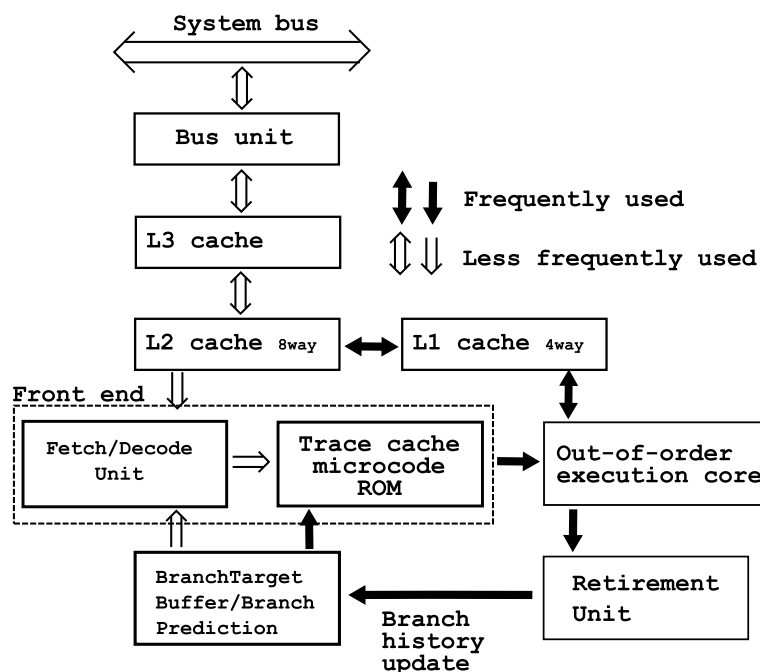


Рис. 3.3. Структура процессоров с архитектурой “Intel NetBurst ©Microarchitecture”

В дальнейшем начиная с 2000 года появляются процессоры Pentium IV и Xeon с архитектурой “Intel NetBurst ©Microarchitecture.” В 2004 и

2006 годах появляются Pentium M и Core Duo с архитектурой Pentium M и наконец в 2008 году серия процессоров Intel Atom, на которых заканчивается IA-32. 64-разрядные процессоры появляются с 2004 года. Это Xeon (NetBurst), Pentium IV Extreme Edition (NetBurst, Hyper-Threading Technology) и далее продолжается серия 64-разрядных процессоров с архитектурой “Intel NetBurst ©Microarchitecture.”

Структура процессоров с такой архитектурой изображается в документах Intel следующим образом (рисунок 3.3). При сравнении со структурой процессоров P6 прежде всего видно появление кэша L3, а несчастный квадратик, который имел прежде четыре названия, получил пятое – Trace cache microcode ROM. Но в описаниях приводятся очень убедительные преимущества перед P6. Помимо работы целочисленных АЛУ на удвоенной частоте, глубокой конвейеризации, такого количества АЛУ разных типов, которое способно выполнить до 126 инструкций за такт, сделать до 48 загрузок и до 24 сохранений за такт, значительно сокращенных потерь времени при ошибках в предсказании ветвлений отмечается, что Trace cache microcode ROM хранит декодированные фрагменты кода.

Рассмотрение дальнейшего развития архитектуры процессоров выходит за пределы настоящего курса, поскольку оно пошло по пути создания многопроцессорных систем на одном чипе. Сначала это делалось путем создания логических процессоров из набора исполнительный аппаратуры (АЛУ разных типов), а далее совершенствование технологии позволило размещать на одном чипе несколько физически отдельных процессоров, которые объединены только большим кэшем L3.

3.4. Программная модель процессора

Несмотря на то что процессор в конечном итоге предназначен для выполнения программы, далеко не все устройства, входящие в него, могут управляться командами этой программы. Программисту доступны только некоторые из них. Поэтому для многих целей достаточно знания только видимых устройств, т. е. таких, на работу которых можно влиять непосредственно при помощи программного кода. Но для наиболее полного использования возможностей процессора необходимо знать о существовании и функционировании некоторых «теневых» устройств, которые никак не упоминаются в наборе инструкций процессора и командах ассемблера. Ниже описываются устройства процессора, доступные программисту. В основном это регистры. Мы рассмотрим группы регистров, связанные с ними типы данных и группы инструкций.

Универсальные регистры. Регистры общего назначения. Все регистры общего назначения могут использоваться по усмотрению программиста. Список инструкций процессора допускает участие каждого из них в любых операциях над целочисленными типами данных, но у некоторых из этих регистров имеются определенные функции, которые они выполняют по умолчанию (без ведома программиста). Эти функции относятся главным образом к формированию исполнительного адреса (части физического адреса). Регистры А, В, С, D абсолютно универсальны и не выполняют никаких функций, кроме явно определяемых программным кодом. Эти регистры могут почти произвольно использоваться программистом для вычислений. Четыре регистра А, В, С, D могут использоваться для 8-разрядных (AH, AL, BH, BL, CH, CL, DH, DL), 16-разрядных (AX, BX, CX, DX), 32-разрядных (EAX, EBX, ECX, EDX) и 64-разрядных (RAX, RBX, RCX, RDX) переменных.

Регистры ESI и EDI (Source Index, Destination Index) также могут использоваться целиком и частями. Эти регистры используются в строковых инструкциях (сравнение и сканирование строк байтов, слов и двойных слов) для указания положения элементов строки. Их содержимое автоматически инкрементируется или декрементируется. В 64-разрядных архитектурах расширяются до RSI и RDI.

Регистры EBP и ESP (или их младшие части BP и SP) предназначены для указания положения в стеке (Stack Frame Base Pointer, Stack Pointer). Если в программе имеются более одного вложения (обращения к подпрограмме из подпрограммы), то стек оказывается разбит на кадры (Frames), соответствующие этим вложениям. SP указывает на вершину кадра в стеке (адрес SS:(E)SP). Вызванная процедура, прежде чем заталкивать (PUSH) в стек переменные, предназначенные для передачи вызывающей программе (передача переменных через стек), копирует ESP в EBP, а при выполнении каждой команды PUSH EBP автоматически инкрементируется, указывая, куда поместить следующую переменную. При возврате из процедуры ESP указывает начало кадра в стеке, где хранятся результаты работы процедуры, а EBP указывает на конец этого кадра. При выполнении каждой команды POP содержимое EBP автоматически декрементируется. В 64-разрядных архитектурах расширяются до RSP и RBP.

На рисунке 3.4 показаны названные регистры и их расширения, доступные в 64-разрядных архитектурах. Реально всех этих регистров в процессоре P6 нет, они существуют только логически, функции каждого из них автоматически передаются одному из 40 (а в Pentium 4 128) регистров общего назначения, соответствие устанавливается при помощи таблицы псевдонимов.

63	31	15	7	0	16 bit	32 bit	64 bit
		AH	AL		AX	EAX	RAX
		BH	BL		BX	EBX	RBX
		CH	CL		CX	ECX	RCX
		DH	DL		DX	EDX	RDY
		EBP				EBP	RBP
		ESI				ESI	RSI
		EDI				EDI	RDI
		ESP				ESP	RSP
		R8				R8D	R8
		R9				R9D	R9
		R10				R10D	R10
		R11				R11D	R11
		R12				R12D	R12
		R13				R13D	R13
		R14				R14D	R14
		R15				R15D	R15

Рис. 3.4. Целочисленные регистры 32- и 64-разрядных процессоров Intel (штриховой линией выделены регистры 32-разрядных процессоров)

Указатель инструкции. Указатель инструкции EIP при выполнении каждой команды хранит адрес следующей инструкции в кодовом сегменте (адрес CS:EIP). В 32-разрядных процессорах его размер 32 разряда. В 64-разрядных процессорах этот регистр называется RIP и имеет размер 64 бита.

Регистр флагов. Регистр флагов не является регистром в обычном понимании этого слова, не существует ни одной команды, которая загружала бы этот регистр. EFLAGS – это совокупность отдельных битов, которые можно разделить на:

- биты состояния,
- биты управления,
- системные биты.

Биты состояния – указывают состояние процессора после выполнения очередной инструкции. Эти биты не могут модифицироваться, исключение составляет CF, который изменяется инструкциями STC и CLC (set carry, clear carry), через этот флаг иногда возвращается значение булевой функции. Из названий, приведённых на рисунке 3.5, непонятно назначение только флага AF (adjust flag). Этот флаг устанавливается при операциях с BCD при заёме единицы из старшего разряда (при вычитании). Бит управления только один – DF (direction flag). При сброшенном бите направления строковые операции выполняются в направлении возрастающих адресов, а при установленном – «задом наперед».

Системные биты перечислим по порядку:

- IF (Interrupt enable) управляет откликом на маскируемые прерывания. Если бит сброшен, то прерывания игнорируются.
- TF (Trap flag) устанавливается для пошагового исполнения инструкций.
- IOPL (I/O privilege level - 2 бита) – уровень привилегий для операций ввода/вывода (инструкции IN и OUT). Уровень привилегий для текущего кода должен быть выше или равен указанному в этом поле. Управляется инструкциями POPF и IRET в программе с наивысшим уровнем привилегий.
- NT (Nested task) управляет прерванными и вызванными задачами. Устанавливается, если текущая задача выполняется после прерывания предыдущей.
- RF (Resume flag) Управляет реакцией процессора на исключения отладки.
- VM (Virtual-8086 mode). Устанавливается для включения режима виртуальной 8086 машины.
- AC (Alignment check). Устанавливается для включения режима проверки выравнивания данных.
- VIF (Virtual interrupt)и VIP (Virtual interrupt pending). Используются для управления прерываниями в многозадачных системах.
- ID (Identification). Запрещает или разрешает инструкцию CPUID.

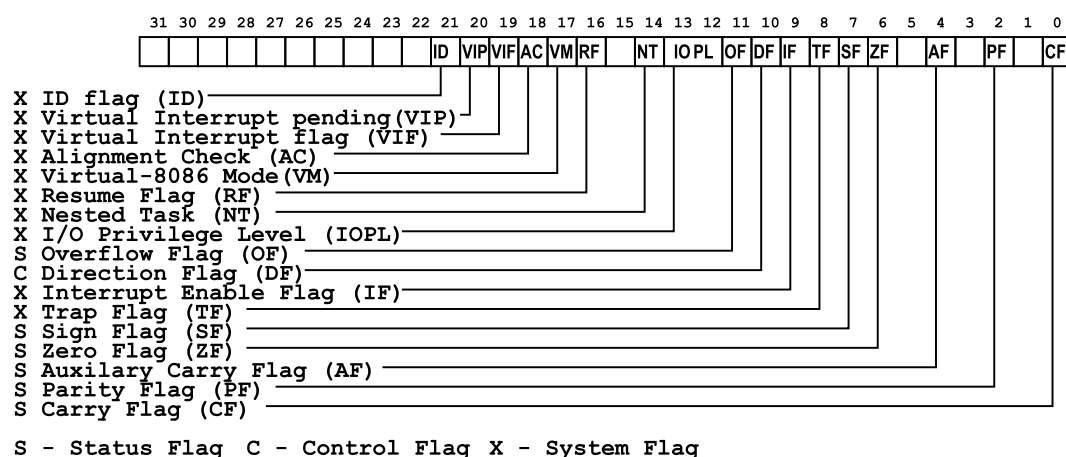


Рис. 3.5. Регистр флагов

В 64-разрядных процессорах EFLAGS становится 64-разрядным RFLAGS, но дополнительные биты ничего не обозначают.

Сегментные регистры. Сегментные регистры участвуют в формировании физического адреса инструкции программного кода (CS), элемента данных (DS, ES, FS и GS) или элемента стека (SS). В режиме прямой адресации они хранят адрес головы сегмента, а в защищенном режиме указывают на индекс дескриптора, тип дескрипторной таблицы и права доступа. В 64-разрядном режиме механизм дескрипторов сегмента практически отменен (принята плоская модель памяти). Кодовый сегмент, сегмент стека и два сегмента данных (те, на которые указывают DS и ES) являются прямыми указателями головы сегмента. Размеры сегментных регистров одинаковы в 32- и 64-разрядных процессорах – 16 бит.

Регистры управления памятью. Регистры системных таблиц указывают на расположение в памяти глобальной таблицы дескрипторов (GDT) – GDTR и таблицы дескрипторов прерываний (IDT) – IDTR. Для процессоров IA-32 их размер 48(16+32) бит, в процессорах Intel64 80(64+16) бит.

Регистры системных сегментов. Кроме сегментов стека, кода и данных, в многозадачных ОС могут использоваться ещё два сегмента – TSS (Task State Segment) и LDT (Local Descriptor Table). Первый служит для сохранения весьма объёмного контекста задачи, необходимого для переключения задач, а второй содержит дополнительную дескрипторную таблицу, которую любое приложение может организовать «для себя». На поверхности лежит только 16-разрядный селектор этих сегментов – дескрипторы грузятся в теневые регистры TR (Task Register) и LDTR (Local Descriptor Table Register). Размеры этих регистров: 80 (IA-32) или 112 (Intel64) бит.

Регистры FPU/MMX. Это видимая часть устройства обработки плавающих чисел (floating point unit – FPU). Основа FPU – регистровый файл из восьми регистров размером по 10 байт (80 бит) каждый. В этих регистрах могут находиться операнды для плавающей арифметики в формате extended (tenbyte). В каком бы формате (single, double, extended) ни хранились эти операнды в памяти, при загрузке они разворачиваются до десятибайтового формата. Команды MMX – MultiMedia eXtensions не имеют отдельной аппаратуры для их выполнения. Упакованные целочисленные переменные MMX загружаются в регистры FPU.

Обозначения битов в регистрах состояния, управления и тегов, входящих в состав FPU, показаны на рисунках 3.7, 3.8 и 3.9. Назначение практически всех битов понятно из названий.

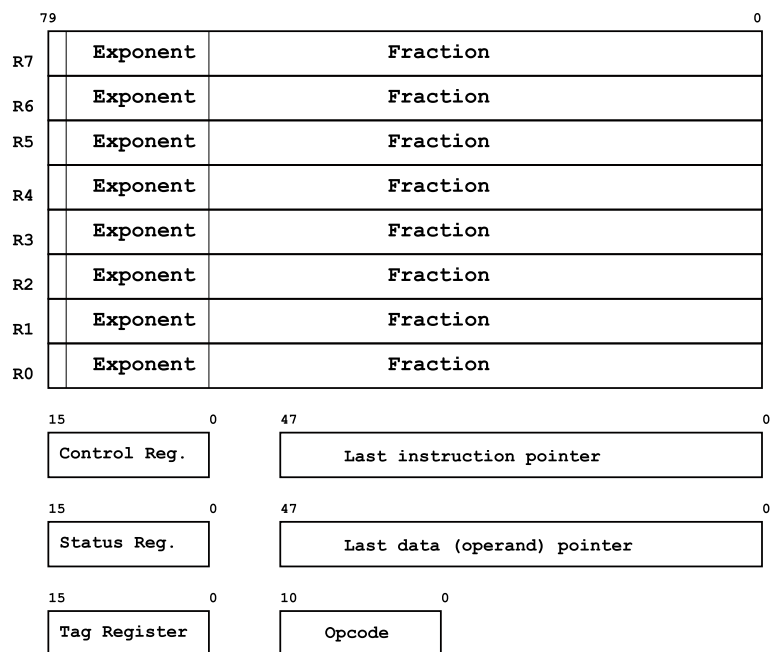


Рис. 3.6. Регистры FPU

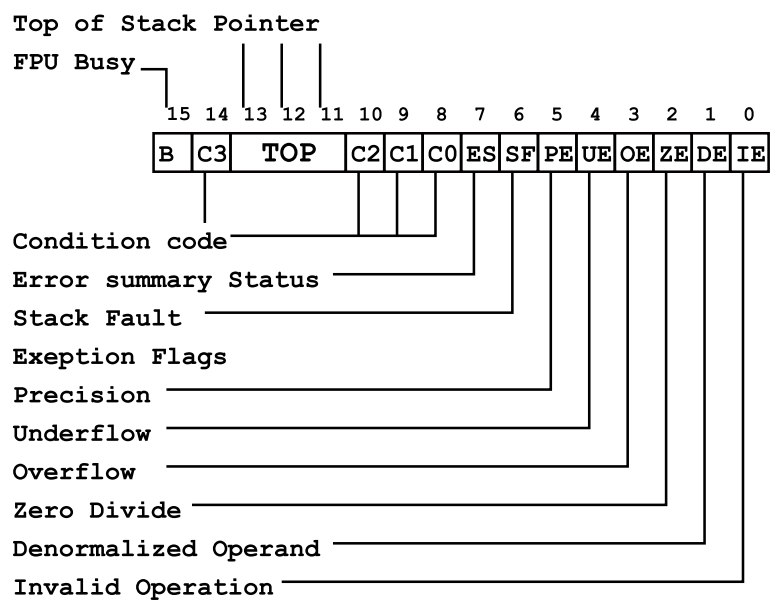


Рис. 3.7. Биты регистра состояния FPU

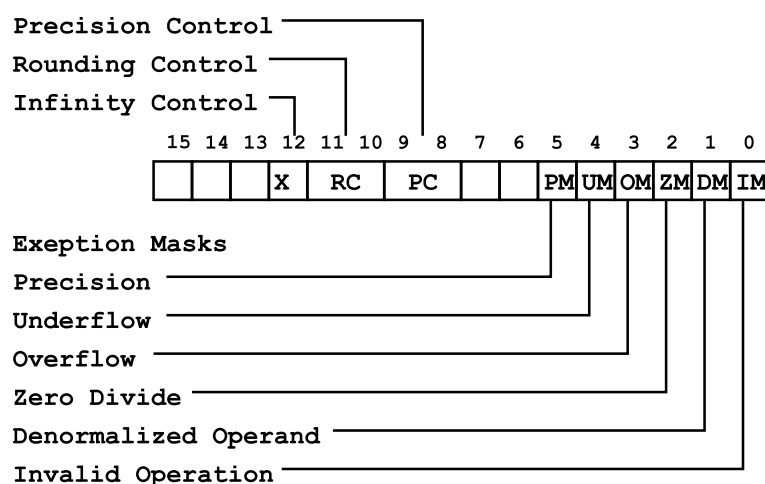


Рис. 3.8. Биты регистра управления FPU

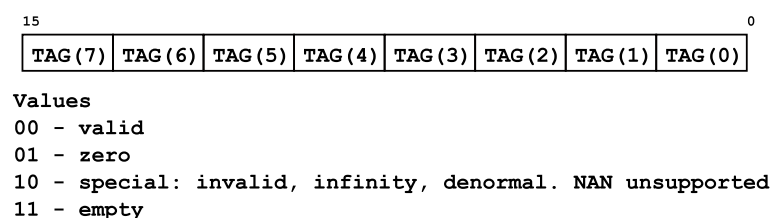


Рис. 3.9. Биты регистра тегов FPU

Регистры SSE. Для выполнения набора инструкций SSE, SSE2, SSE3, SSE4.1 и SSE4.2 имеется специальный векторный процессор, видимой частью которого являются 8 128-разрядных регистров XMM. Данные в этих регистрах могут быть следующих типов:

- 4× Packed Single,
- 2 ×Packed Double,
- 16×Packed byte,
- 8×Packed word,
- 4×Packed doubleword,
- 2×Packed quadword,
- 1×Packed doublequadword.

Инструкция SSE выполняется параллельно со всеми числами в упаковке (отсюда и название технологии – Single Instruction Multiple Data -SIMD). В 64-разрядных процессорах число регистров XMM удваивается.

Регистры AVX. В некоторых 64-разрядных процессорах (Intel Xeon processor E7 series, Intel Xeon processors X3600, X5600, Intel Core i7 980X processor) имеются ещё 8 регистров AVX длиной 256 бит предназначенных для вычислений по технологии AESNI. Данные, которые могут обрабатываться в этих регистрах: 8 single FP, 4 double FP или два 128-разрядных числа. Главной частью этого расширения набора инструкций

является кодирование и декодирование сообщений по стандарту AES (Advanced Encryption Standard)

Регистры управления. Регистры управления содержат большое количество информации, необходимой для поддержания защищенного режима и других особенностей работы процессора.

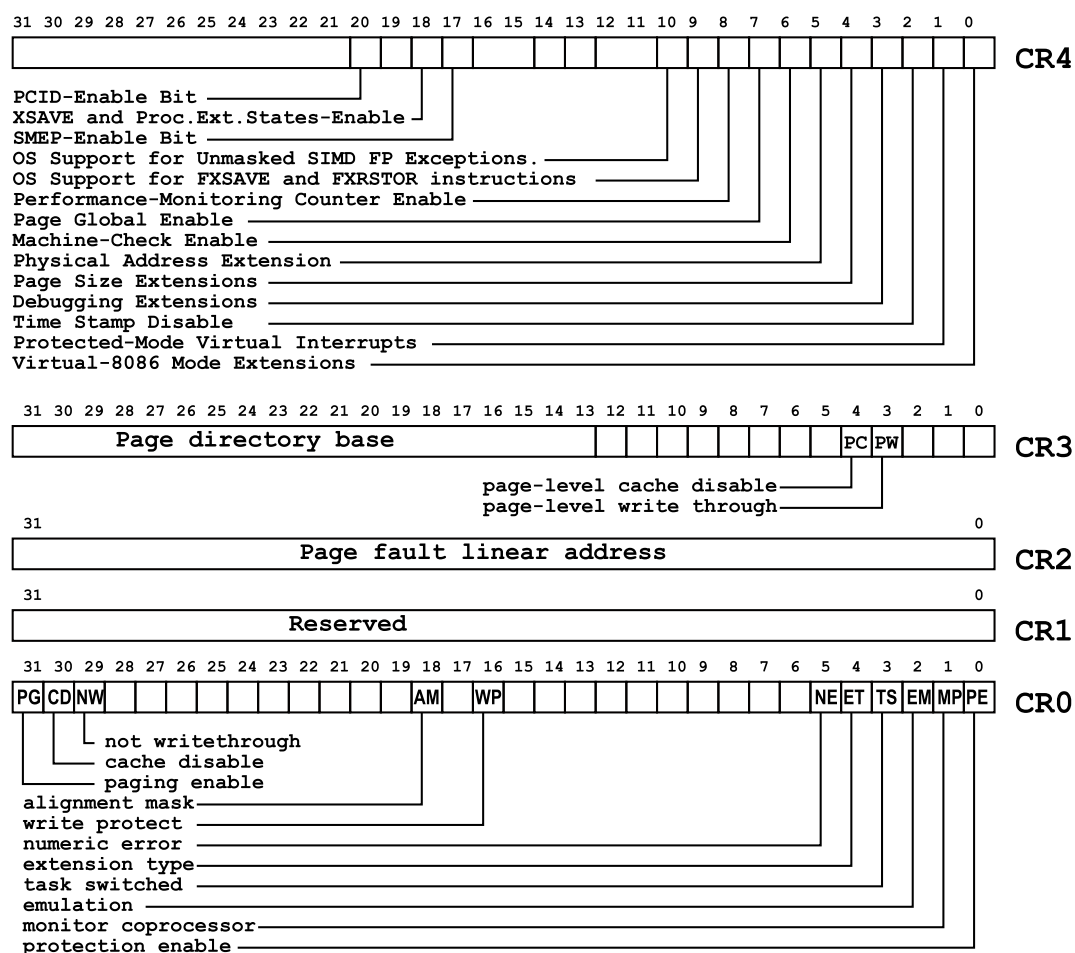


Рис. 3.10. Регистры управления IA-32

На рисунке 3.10 приведен подробный список назначения полей регистров управления. Назначение многих полей ясно из названий. Исчерпывающую информацию о назначении всех полей и способах их использования можно найти в Intel® 64 and IA-32 Architectures Software Developers Manual. Vol. 3A.

Регистры отладки. Регистры отладки необходимы для обеспечения работы отладчика (debugger) в режиме выполнения программы по фрагментам. Точки останова – это места в коде, где необходимо остановиться и проконтролировать состояние переменных, используемых в програм-

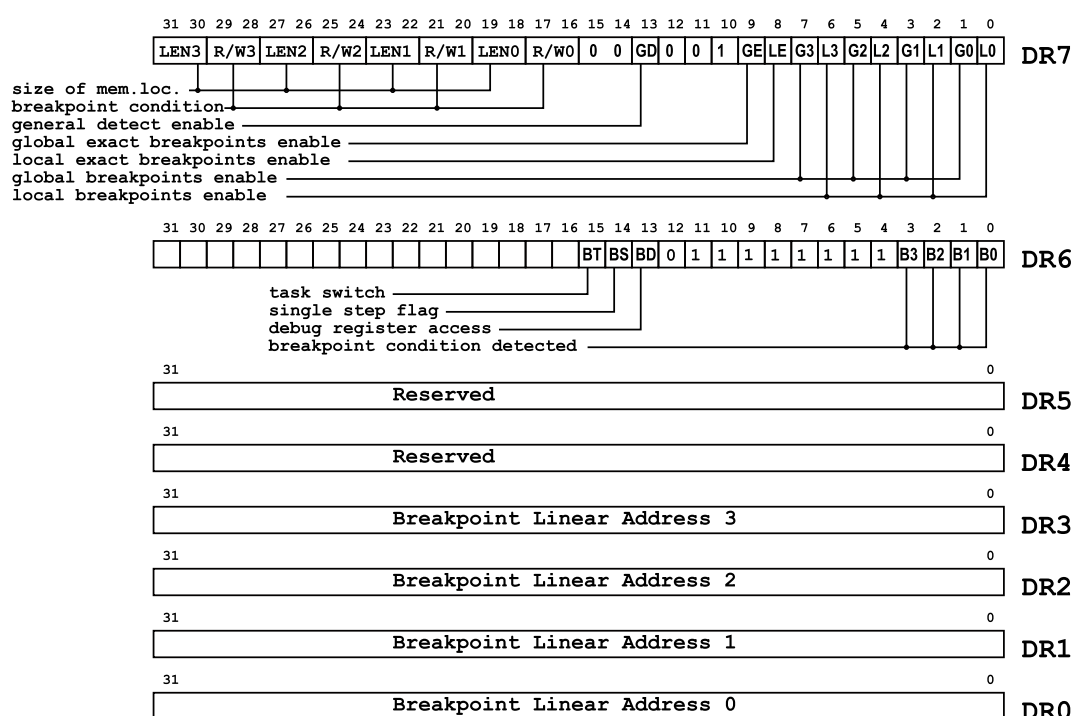


Рис. 3.11. Регистры отладки в IA-32

ме. Назначение регистров, полей и битов в этой группе показано на рисунке 3.11.

Счётчики системных событий. Несколько счётчиков могут быть настроены на подсчёт системных событий, таких как кэш-попадания, кэш-промахи, операции ввода/вывода и т. д. Контролируя содержимое этих счётчиков при помощи команды RDPMC, можно получить информацию об узких местах (bottleneck) в программе и добиться максимальной производительности.

Счетчик мониторинга производительности. Счётчик мониторинга производительности непрерывно после включения процессора считает его такты. Огромный размер этого счётчика (64 разряда) позволяет даже при тактовой частоте в несколько гигагерц считать такты в течение нескольких лет до переполнения. Инструкция RDTSC (read time stamp counter) позволяет считывать содержимое счётчика в пару 32-разрядных регистров EAX и EBX. Произведя такое считывание в начале некоторого фрагмента программы и в его конце, можно вычислить время его выполнения. Правильно откалиброванный при помощи часов реального времени, счётчик позволяет формировать и измерять временные интервалы с высокой точностью, не производя достаточно длительных обращений к часам реального времени.

3.5. Режимы работы процессора

Архитектура IA-32 поддерживает три основных режима работы процессора: защищённый режим виртуальных адресов, режим реальных адресов и режим управления системой.

Защищённый режим виртуальных адресов (Protected virtual address mode). Этот режим позволяет получить наивысшую производительность и возможности и рекомендуется для использования всеми новыми приложениями и операционными системами. Главной отличительной чертой защищённого режима является аппаратно поддерживаемый механизм управления памятью, позволяющий предоставить каждой задаче собственное изолированное и защищённое адресное пространство, размер которого может превосходить даже физически допустимый объём оперативной памяти. Режим устанавливает иерархию привилегий для задач, которая не позволяет приложениям разрушить операционную систему или другие приложения. В частности, он позволяет предоставить приложению виртуальную машину. Этот вид защищённого режима называется V86.

Режим реальных адресов (Real address mode). В этом режиме процессоры семейств P5 и P6 работают подобно старинному процессору 8086, но намного быстрее и имеют некоторые дополнительные возможности (например, перехода в защищённый режим). Адресное пространство, предоставляемое задаче в этом режиме, представляет собой сегменты, не превосходящие по объёму 64К. Единственным и очень сомнительным преимуществом режима реальных адресов является свободный доступ к любой области памяти. Этот режим чаще всего используется как переходной, необходимый для подготовки запуска процессора в защищённом режиме.

Режим управления системой (System management mode – SMM). Этот режим позволяет управлять энергопотреблением системы, изменять тактовую частоту, переводить устройства и процессор в спящий режим. Такой режим особенно важен в компьютерах с автономным питанием (laptop, palmtop). Переход в режим управления системой возможен при подаче активного сигнала на один из выводов процессора или при возникновении аппаратного прерывания специального типа. Процессор при таком переходе полностью запоминает состояние текущей задачи и при обратном выходе может продолжать её выполнение.

!h

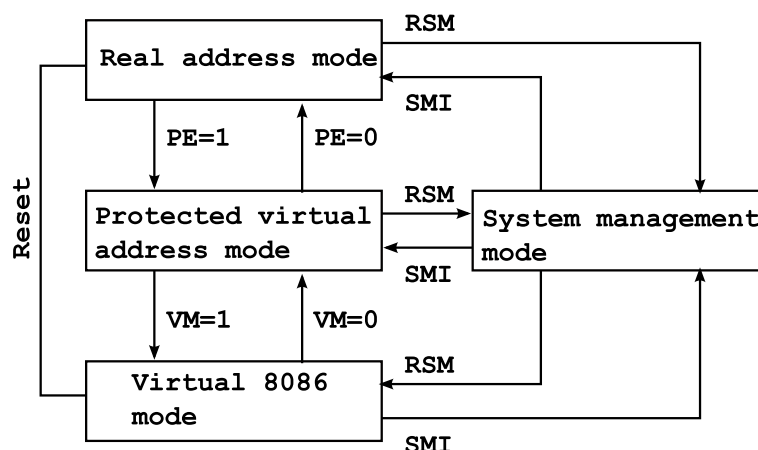


Рис. 3.12. Режимы работы IA-32 процессора (PE и VM – флаги в EFLAGS, SMI – прерывание управления системой, RSM – инструкция управления системой)

Режим системного управления (SMM) может использовать только обработчик системного прерывания (SMI), который является частью базовой системы ввода/вывода (BIOS). Возможности и условия переходов между этими режимами иллюстрируются рисунком 3.12. Как видно из схемы, даже переход в SMM возможен только по системному прерыванию. Возврат в предыдущий режим происходит по команде RSM (Return System Management).

Приложения (т. е. нормальные программы, которые можно написать, откомпилировать, собрать и запустить) используют только три из названных четырёх режимов работы процессора – Real Mode, Protected Mode и V86. Для функционирования процессора в Protected Mode и V86 необходимы некоторые приготовления (формирование GDT и IDT), потому что при включении процессор оказывается в Real Mode, а переход в Protected Mode осуществляется программно (например, при загрузке ОС) путём установки флага PE в EFLAGS. Обратный переход в Real Mode возможен только по RESET.

Нереальный режим (Unreal mode). Нереальный режим (Unreal mode), или режим линейных адресов (Linear address mode), не является легальным режимом работы процессора и не описывается в документах Intel. Каждый из перечисленных выше легальных режимов – это всего лишь сочетание установок флагов, управляющих возможностями процессора, и фирма Intel только рекомендует определённые из них в качестве режимов работы процессора. Эти режимы были тщательно протестированы на множестве программ, и Intel несет определённую ответ-

ственность за последствия. Безусловно, можно попробовать много других возможностей установки режима работы процессора. Большинство из этих режимов бесполезны, но есть и полезные.

К таким режимам относится режим линейных адресов. Суть этого режима состоит в том, что программист получает возможность обращения к большим (сопоставимым с полным объёмом памяти) сегментам данных, оставаясь в режиме реальных адресов. Такая задача возникает, например, в программировании обмена информацией с приборами в реальном времени.

Режим был предложен в 1989 году Томасом Роденом. Дело в том, что ограничения на размер сегмента (64Kb) устанавливаются в режиме реальных адресов при помощи того же дескриптора сегмента, который грузится в теневой регистр и в котором указан размер сегмента равный 64Kb. Идея перехода в режим линейных адресов состоит во временном выходе в защищённый режим, связывании с одним из дополнительных сегментов данных (обычно GS) дескриптора с размером во всю физическую память и возврате в режим реальных адресов.

Этот прием был использован фирмой Watcom для создания расширителя DOS, который называется DOS4GW (DOS 4 Gigabyte Watcom), и использовался в играх для DOS. Расширителей режима реальных адресов такого типа в настоящее время довольно много, наиболее известны GO32, DOS32.

Режимы работы Intel-64 процессора. В 64-разрядных процессорах ко всем легальным режимам работы 32-разрядного процессора **добавляется** режим IA-32E, который распадается на два подрежима:

- Режим совместимости. В режиме совместимости любое 16- или 32-разрядное приложение защищенного режима IA-32 будет выполняться без перекомпиляции. Поддерживается сегментация, страничная переадресация (включая доступ к памяти до 16 Гб при использовании больших страниц памяти), все уровни привилегий остаются в силе. Вся исполнительная среда только расширяется. Поддерживаются все типы данных 32-разрядного процессора. Исключение составляют приложения, использующие System Management и v86.
- 64-разрядный режим. В этом режиме приложениям доступно 64-разрядное адресное пространство (2^{64} байт). В этом режиме число регистров общего назначения и SIMD-регистров (MMX и SSE) увеличивается с 8 до 16, в инструкциях, которые используют этот

ресурс, необходим префикс (REX) в опкодах инструкций. Деятельность FPU не изменяется.

3.6. Адресация.

Режим реальных адресов. Основным отличием работы процессора в защищённом режиме от работы в режиме реальных адресов является способ формирования физического адреса. В режиме реальных адресов физический адрес образуется из содержимого сегментного регистра и исполнительного адреса (или смещения), находящегося в одном из 16-разрядных регистров общего назначения. При этом для образования 20-разрядного числа (8086) селектор сегмента (содержимое сегментного регистра) сдвигается на 4 разряда вперед и складывается с исполнительным адресом. Поскольку сдвиг на 4 разряда эквивалентен умножению на 16, физический адрес можно вычислить по формуле

$$PA = 16 \cdot Seg + EA.$$

Сказанное можно проиллюстрировать следующей диаграммой (рис. 3.13).

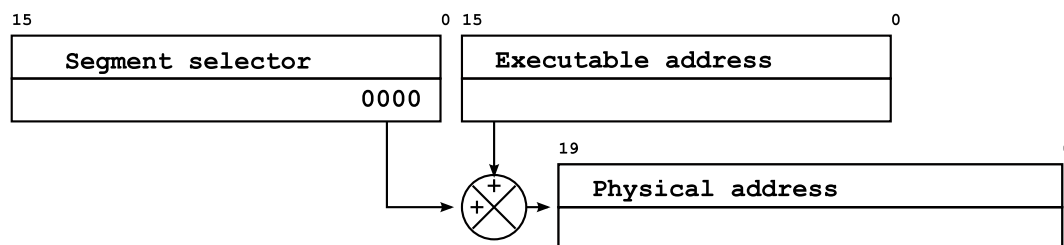


Рис. 3.13. Формирование физического адреса в режиме реальных адресов

Такой способ формирования физического адреса приводит к фиксированному размеру сегмента (64К). Сегменты оказываются «свернутыми в кольцо», поскольку никакого переноса из смещения не происходит. В 32-разрядных машинах попытка «вылезти» за пределы сегмента размером 64К пресекается генерацией исключения GP. Такая ситуация возможна при обращении к 16- или 32-байтным данным у границы сегмента. Более того, все адресное пространство процессора 8086 оказывается закольцованным, поскольку адресных линий только 20 (A0-A19). 80286 и 32-разрядные машины (80386+) для программной совместимости должны повторять эту ситуацию, хотя фактически шина адреса у них шире (24 - 286, 32 - 386+).

Для отработки ситуации переноса в разряд A20 на системных платах под процессор Intel 286 стали устанавливать специальное устройство – Gate A20, которое не пропускает на шину адреса перенос в A20 из EA. В 32-разрядных процессорах для этой цели служит специальный вход A20M, по которому перенос в A20 запрещается. Единичка в 20-м разряде шины адреса может появиться, только если она устанавливается в сегментном регистре.

Замкнутость сегментов в реальном режиме обеспечивала некоторую защиту их друг от друга, но сегментные регистры любое приложение могло перезагружать «абсолютно безнаказанно» – никакие средства этого процесса не контролировали. Контроль всевозможных «наездов» (например, стека на данные или код) являлся головной болью программиста.

Вернёмся к вычислению эффективного адреса. Здесь также имеются различия между реальным и защищённым режимом. Для реального режима все способы адресации могут быть записаны при помощи одной формулы

$$EA = Base + Index + Displacement$$

или более подробно, с указанием места, где может находиться каждый из компонентов адреса

$$EA = \begin{pmatrix} BX \\ BP \end{pmatrix} + \begin{pmatrix} SI \\ DI \end{pmatrix} + \begin{pmatrix} 0 \\ i8 \\ i16 \end{pmatrix}.$$

Защищённый режим. В защищённом режиме действуют два механизма адресации:

- сегментация и
- страничная организация.

В соответствии с этим мы можем определить три адреса:

- логический адрес – тот, который генерирует программа (EA),
- линейный адрес – тот, который получается после определения сегмента (LA) и
- физический адрес – тот, который появляется на шине адреса после страничной переадресации (PA).

Заметим, что если сегментация является обязательным атрибутом защищённого режима, то страничная переадресация может использоваться, а может и не использоваться (если в регистре CR0 сброшен бит PG). В случае, когда страничная организация отключена, линейный адрес становится физическим и поступает на системную шину.

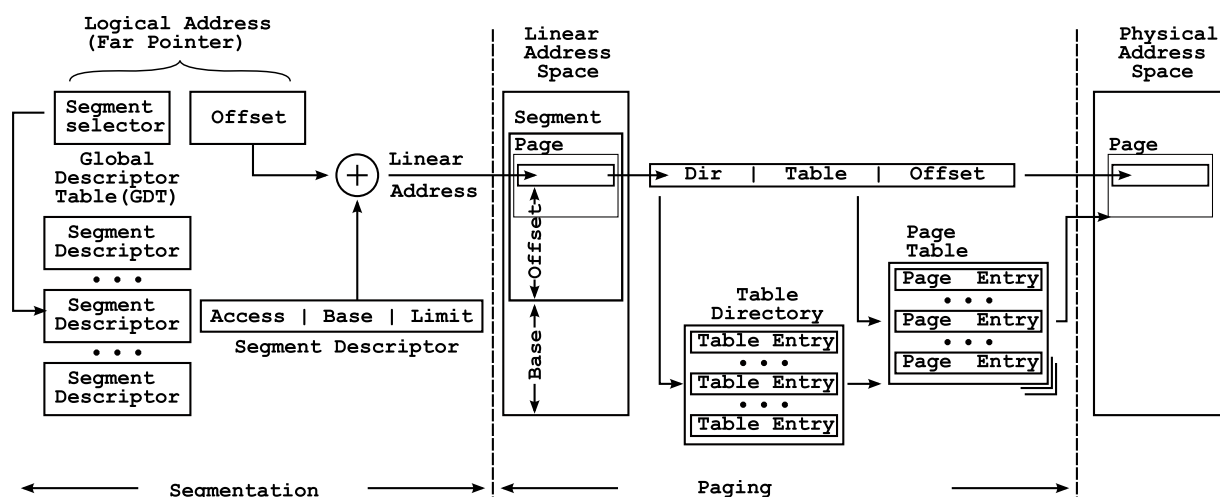


Рис. 3.14. Сегментация и страничная организация памяти.

Сегментация. Сегмент в защищённом режиме является более сложным образованием, нежели в реальном режиме. Средства защищённого режима позволяют:

- использовать сегменты практически любого размера (до 4 Гбайт);
- ограничивать права доступа к сегменту. Эта возможность позволяет реализовать защиту данных и программ;
- организовывать переключение задач.

В защищённом режиме селектор сегмента не является адресом головы сегмента, а участвует в его выборе косвенно. Селектор только указывает на более подробный источник информации о сегменте – его дескриптор (descriptor – описатель). Количество дескрипторов, которые могут использоваться программами, достаточно велико (2^{13}), а размер – 96 бит, поэтому дескрипторы не хранятся в самом процессоре. Для поддержки защищённого режима необходимы таблицы дескрипторов, которые хранятся в оперативной памяти. Мы можем проиллюстрировать формирование адреса следующей диаграммой (рисунок 3.14).

Для функционирования процессора в защищённом режиме перед переключением в него в памяти должны быть сформированы таблицы дескрипторов:

- 1) глобальная таблица дескрипторов (GDT),
- 2) таблица дескрипторов прерываний (IDT),
- 3) необязательная локальная таблица дескрипторов (LDT).

Селектор сегмента в защищенном режиме содержит три поля:

- 1) 13-разрядное поле индекса (Index), определяющее положение требуемого дескриптора в таблице;
- 2) 1-разрядное поле (TI – Table Index), определяющее тип дескрипторной таблицы, к которой обращается данный селектор (TI=0 if GDT, TI=1 if LDT);
- 3) 2-разрядное поле (RPL – Requested Privilege Level), определяющее уровень привилегий, который запрашивает программа, загружающая селектор.

В архитектуре IA32 определены четыре уровня привилегий для задач и данных. Уровень привилегий задачи (Task Privilege Level) определяет доступные для этой задачи сегменты и инструкции. Уровень привилегий определяется полем уровня привилегий (PL) в селекторе кодового сегмента данной задачи (загруженного в данный момент). Максимальные привилегии имеет задача с $PL = 0$, минимальные с $PL = 3$. По идее $PL = 0$ должен соответствовать ядру операционной системы, $PL = 1$ системным сервисам, $PL = 2$ системным утилитам и, наконец, $PL = 3$ – пользовательским программам. С любой задачей связаны некоторый сегмент кода, стека и сегменты данных, все они имеют одинаковый уровень привилегий. Поэтому уровень привилегий фактически является атрибутом сегмента. Уровень привилегий, зафиксированный в дескрипторе сегмента, называют DPL (Descriptor Privilege Level). Если некоторая задача в текущий момент времени загружена и выполняется, то её уровень привилегий называют текущим (CPL – Current Privilege Level). Если выполняется команда, загружающая сегментный регистр некоторым селектором, то уровень привилегий в этом селекторе называют запрашиваемым (RPL – Requested Privilege Level). С каждым сегментным регистром (SS, CS, DS, ES, FS, GS, TS и LDTS) аппаратно связан недоступный для программиста регистр, предназначенный для загрузки дескриптора соответствующего сегмента. При поступлении команды загрузки селектора в сегментный регистр (явной или неявной) автоматически происходит следующее:

- 1) определяется дескрипторная таблица (глобальная или локальная). Если указанной таблицы нет в памяти, генерируется исключение GP;
- 2) разыскивается указанный дескриптор (по индексу). Если дескриптора с таким индексом нет, генерируется исключение GP;
- 3) сравниваются запрошенный уровень привилегий (RPL) и текущий уровень привилегий (CPL – Current Privilege Level) уровня привилегий в селекторе сегмента кода) с уровнем привилегий дескриптора (DPL);

- 4) если сегмент доступен для данного кода, происходит загрузка дескриптора запрошенного сегмента в регистр дескриптора. Если сегмент оказывается недоступным, генерируется исключение GP.

Дескриптор сегмента содержит несколько полей, важнейшими из которых являются база сегмента (segment base) и размер сегмента (segment limit), числа в этих полях указывают расположение сегмента в пространстве линейных адресов. При формировании линейного адреса используется поле базы сегмента и исполнительный адрес. Исполнительный адрес в 32-разрядных процессорах формируется по следующей формуле

$$EA = Base + Index \cdot Scale + Displacement$$

или более подробно

$$EA = \begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{pmatrix} + \begin{pmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} + \begin{pmatrix} None \\ i8 \\ i16 \\ i32 \end{pmatrix}.$$

В отличие от 16-разрядной адресации реального режима адрес формируется по более сложной схеме, содержащей новый элемент Scale, и в формировании адреса могут участвовать все регистры общего назначения. Единственное ограничение – указатель стека (ESP) не может быть индексом. В зависимости от сочетания используемых полей исполнительного адреса различают несколько методов адресации.

Страничная организация. Страничная организация (Paging) памяти служит, главным образом, для разбиения адресного пространства на удобные для формирования физических адресов области равного размера. Границы страниц могут быть никак не связаны с границами сегментов, но такое разбиение вряд ли разумно.

Страничная организация служит основным средством для организации виртуальной памяти. Если страница отсутствует в памяти, то процессор генерирует исключение PF – отказ страницы, и обработчик этого исключения может отсутствующую страницу подгрузить (swar), например, с диска.

Со страничной организацией связано использование дополнительных линий шины адреса процессора (A32-A35), их использование возможно

только при включенном блоке страничной переадресации, при этом процессор может адресовать 64 ГБ оперативной памяти.

В 32-разрядных процессорах разных поколений возможно использование страниц различных размеров. Общими для всех IA32-процессоров являются страницы размером 4 КБ. Начиная с Pentium, появилась возможность использовать страницы размером 4 МБ. В семействе P6 используются страницы размером 4 КБ и 2 МБ. В Pentium III возможны страницы всех трех размеров.

Страничная переадресация основана на интерпретации линейного адреса в том же духе, как это происходит при сегментации. Для страниц размером 4Кбайт страничная переадресация может быть представлена, как показано на рисунке 3.14. Линейный адрес при страничной переадресации разбивается на три части. Старшие разряды (22-31) указывают на элемент директории страниц (PDE – Page Directory Entry – своего рода дескриптор в таблице), средние (12-21) разряды линейного адреса указывают на элемент таблицы страниц (PTE – Page Table Entry), т. е. на страницу и, наконец, младшие разряды определяют элемент памяти внутри страницы. Положение директории страниц определяется управляющим регистром CR3, в котором имеется поле PDBR – Page Directory Base. Весь управляющий регистр CR2 хранит линейный адрес последнего отказа страницы. В регистре CR4 имеются биты, управляющие расширениями размера страницы.

Директория страниц содержит 1024 элемента PDE, каждый из которых указывает на таблицу страниц. Каждая таблица страниц содержит 1024 элемента PTE, каждый из которых указывает на страницу памяти. Элементы PDE и PTE почти одинаковы по своей структуре и, кроме базового адреса таблицы или страницы, содержат биты управляющие доступом к таблице или странице:

- G — Granularity
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type
- DPL — Descriptor privilege level
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)

Бит P – presence определяет наличие страницы или таблицы в памяти. Бит G – global используется в Р6 для определения глобальности страницы. Биты PWT и PCD определяют политику кэширования и запреты кэширования для страницы или таблицы страниц. При страничной переадресации также имеются уровни привилегий, но их только два – пользователь и супервизор. Страница или таблица страниц может быть также доступной или недоступной для чтения. Этими свойствами управляют биты U/S и R/W.

3.7. Вызовы подпрограмм, исключения и прерывания

Все команды, которые перезагружают сегмент кода, как записанные в кодовом сегменте, так и получаемые процессором аппаратно через выходы прерываний, используют стек, и последовательность действий процессора во всех этих случаях во многом похожа. Стек – это непрерывная область в памяти, которая содержится в сегменте стека, идентифицируется селектором, загруженным в регистр SS, и дескриптором в теновом регистре. Главное назначение стека – это запоминание адресов возврата из подпрограммы и передача параметров между вызывающей программой и подпрограммами.

Новая информация записывается в стек командой PUSH и читается из стека командой POP. В каждый момент времени доступна только вершина стека, т. е. информация, записанная в последний раз (структура LIFO). Регистр ESP содержит указатель на текущую вершину стека (её адрес), его содержимое автоматически уменьшается при выполнении команды PUSH и увеличивается при выполнении команды POP. Таким образом, стек растёт вниз от адреса, указанного в селекторе или в поле Base в дескрипторе сегмента, в сторону меньших адресов и сокращается вверх, в сторону больших адресов. Стеков может быть много, любая программа может использовать несколько стековых сегментов, но в каждый момент времени доступен только один – текущий стек – тот, который указан в регистре SS.

Данные в стеке должны выравниваться, т. е. иметь одинаковую ширину: либо 16 бит, либо 32 бита. Регистр ESP должен изменяться либо на 2, либо на 4 в командах PUSH и POP, этим управляет флаг D в дескрипторе стекового сегмента, а также префикс размера адреса. Обычно стек разбит на кадры (frames). Каждый такой кадр содержит информацию, необходимую для организации связи между программами, и (возможно)

параметры, которые необходимо передать из одной программы в другую. Указатель на базу кадра содержится в регистре ЕВР.

Прерывания и исключения – это события, указывающие на то, что в процессоре, системе или в текущей программе или задаче имеются условия, требующие внимания процессора (выполнение каких то действий). Обычно они приводят к принудительной передаче управления специальной процедуре, называемой обработчиком прерывания или исключения. Такое действие называется обслуживанием или обработкой прерывания. Прерывания и исключения обычно возникают в случайные моменты времени. По источнику сигнала прерывания их делят на

- аппаратные и
- программные.

Первые генерируются аппаратурой процессора или внешней по отношению к процессору аппаратурой (устройства ввода/вывода, управление системой). Вторые возникают при исполнении определенных инструкций процессора. Вектора исключений и прерываний – это идентификационные номера, указывающие на дескриптор в таблице дескрипторов прерываний (interrupt descriptor table – IDT). Эта таблица может содержать до 256 дескрипторов, указывающих на точки входа (gates) обработчиков прерываний и исключений. Первые 32 дескриптора (от 0 до 31) зарезервированы стандартными для IA-32 типами прерываний и исключений. Их обозначения и описания приведены в таблице 3.3. Остальные предоставляются пользователю и операционной системе.

Внешние по отношению к процессору прерывания поступают в процессор через вывод INTR (P6-) или APIC (0:1) (PIV, Xeon +). Такие прерывания называются маскируемыми. Реакция на маскируемые прерывания определяется флагом IF. Если он установлен, то эти прерывания игнорируются. Внутренние прерывания делятся на

- исключения ошибок программного кода,
- исключения, генерируемые инструкциями процессора,
- исключения контроля процессора.

Исключения ошибок программного кода по способу исполнения обработчика делятся на

- отказы (faults),
- ловушки (traps),
- аборты (aborts).

Отказы – это прерывания, которые могут быть исправлены, и после исправления программа может исполняться далее без потери непрерывности. При сообщении об отказе процессор восстанавливает состояние до

№	sign	comment	source
0	DE	Divide Error	DIV and IDIV instructions.
1	DB	Debug	Any code or data reference.
2	NMI	Interrupt	Non-maskable external interrupt.
3	BP	Breakpoint	INT 3 instruction.
4	OF	Overflow	INTO instruction.
5	BR	Bound range exceeded	BOUND instruction.
6	UD	Invalid Opcode (UnDefined Opcode)	UD2 instruction or reserved opcode.
7	NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.
9	MF	CoProcessor Segment Overrun(reserved)	Floating-point instruction.
10	TS	Invalid TSS	Task switch or TSS access.
11	NP	Segment Not Present	Loading segment registers or accessing system segments.
12	SS	Debug	Any code or data reference. Stack Segment Fault Stack operations and SS register loads.
13	GP	General Protection	Any memory reference and protection check.
14	PF	Page Fault	Any memory reference.
15		Reserved	
16	MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	AC	Alignment Check	Any data reference in memory.
18	MC	Machine Check Error	Codes and source are model dependent.
19	XF	SIMD Floating-Point Exception	SIMD Floating-Point Instruction
20- 31		Reserved	
32- 255		Maskable Interrupts	External interrupt from INTR pin or INT n instruction.

Таблица 3.3. Стандартные прерывания

отказа и перезапускает программу. Вторичный отказ интерпретируется как аборт.

Ловушки предназначены для отладки, они возникают при установленном флаге TF после выполнения каждой инструкции и возвращают управление следующей после ловушки инструкции. При взаимодействии с программой отладчиком это позволяет проходить отлаживаемую программу по шагам.

Аборты – это исключения, при которых невозможно определить место в программном коде, где такое исключение произошло, и, следовательно, невозможно предпринять что-либо для продолжения работы программы. Обработчики таких исключений обычно запоминают состояние процессора, прекращают работу программы и передают управление операционной системе или, вызвавшей данную программу, программе.

В наборе инструкций процессора имеются инструкции, позволяющие обратиться к обработчику прерывания как к подпрограмме, – это инструкции INTO, INT 3, BOUND и INT n, в которой операнд n может иметь значение от 0 до 255. Реально эти прерывания не являются прерываниями в истинном смысле слова.

Ошибки контроля процессора (machine-check errors) появились только в процессорах семейства Р6. Эти ошибки детектируются новой аппаратурой контроля процессора, и при этом процессор передает управление на вектор 18 и предоставляет код ошибки, который сохраняется в стековом сегменте.

3.8. Набор инструкций

Инструкции общего назначения. Инструкции общего назначения выполняют основные перемещения данных, арифметические, логические и строковые операции, управляют потоком операций и обычно используются для написания прикладных и системных программ. Эти инструкции используют операнды, содержащиеся в памяти, регистрах общего назначения, а также адресную информацию из регистров общего назначения и сегментных регистров. Список этих инструкций очень большой и является предметом изучения в курсе программирования на языке Assembler.

Инструкции FPU. Инструкции FPU управляют работой математического сопроцессора и главным образом осуществляют вычисления с плавающими числами. Они имеют дело со специальными регистрами математического сопроцессора (FPU – floating point unit). Это устрой-

ство является основой для вычислительных программ и будет подробно рассмотрено в следующем разделе.

Инструкции MMX и SSE. Инструкции MMX и SSE управляют работой двух разных устройств. MMX (MultiMedia Extensions) – это технология распараллеливания вычислений на уровне данных. Проще говоря, одна инструкция MMX выполняет операцию сразу с несколькими парами операндов. В процессорах Intel нет специальной аппаратуры для выполнения этих инструкций. Они выполняются FPU. Практически MMX – это режим работы FPU. Главный тип данных MMX – упаковки целых чисел.

SSE (Streaming SIMD Extensions) – ещё одна технология параллельных вычислений на уровне данных. Для выполнения этой группы инструкций имеется специальное устройство, выполняющее операции с упаковками плавающих чисел. Эта группа команд накапливалась в линейке процессоров начиная с Pentium 4. Базовый список инструкций SSE постоянно расширялся (SSE2, SSE3, SSE4.1, SSE4.2). Технология SSE, по-видимому, вытеснит FPU в вычислительных приложениях. Обе технологии будут рассмотрены подробно в следующих разделах.

Системные инструкции. Группа инструкций, предназначенных главным образом для операционных систем. В прикладных программах практически не используются, хотя некоторые инструкции могут быть полезны для приложений реального времени.

3.9. Типы данных и инструкции FPU

Правила вычислений с плавающими числами оговорены стандартами IEEE 754 и 854. Любые процессоры должны придерживаться этих стандартов. В процессорах IA32 они реализованы полностью. По стандарту плавающие числа записываются в форме, которая показана на рисунке 3.15.

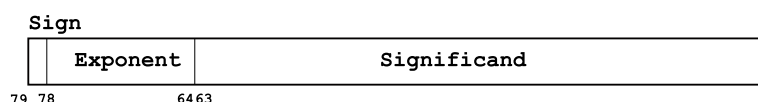


Рис. 3.15. Плавающее число

Здесь Sign – знак числа, Significand – значащая часть или мантисса, Exponent – порядок. Размеры этих полей в процессорах IA-32 и Intel64

одинаковы для всех плавающих чисел, обрабатываемых FPU. Для плавающих чисел следует различать форматы хранения и формат обработки чисел. Для обработки числа всегда разворачиваются в десятибайтовый формат, показанный на рисунке 3.15, а для хранения используются форматы Single, Double и Extended, которые имеют размер 4, 8 и 10 байтов и изображены на рисунке 3.16. Плавающее число интерпретируется следующим образом:

$$Number = Sign \times Significand \times 2^{Exponent}$$

Мантисса – число в интервале (0.5,1), представленное разложением по отрицательным степеням двойки. Нетрудно видеть, что такое разложение обязательно начинается с 1. Если это условие выполняется и порядок находится в допустимых пределах, то число называется нормализованным (NORMAL), если мантисса начинается с нуля, то число называется денормализованным (DENORMAL).

С денормализованными числами ещё можно продолжать вычисления, но точность при этом теряется (не соответствует формату), и имеется риск получить неверный результат. Денормализованные числа заполняют отрезок на действительной оси между истинным нулем и минимальным нормализованным числом. Возникновение денормализованного числа отмечается в теге регистра.

Порядок не содержит знакового бита явно, но содержит степень двойки в смещённой форме (Biassed exponent). Максимально допустимое значение этого поля 1111...110, а минимальное 0000...000. Так сделано, чтобы количество значений, которые принимает экспонента, было нечётным, содержало ноль и было симметричным относительно него. Записи, в которых поле Exponent имеет значение 1111...11 при нулевой мантиссе, являются бесконечностями со знаком (+INF > -INF).

Записи, в которых поля Exponent и Significand нулевые, называются нулями (ZERO). В зависимости от поля Sign различают +ZERO и -ZERO, но в операциях сравнения +ZERO = -ZERO.

Если порядок равен 11...11, а мантисса ненулевая, то в регистре FPU не число (Not a Number - NAN). Если мантисса начинается с 0, то попытка выполнить с ними арифметические операции вызывает исключение, поэтому их называют Signaling NAN – SNAN. При разных знаковых полях возникают +SNAN и -SNAN.

Если мантисса начинается с единицы, то исключение не возникает, такие записи называются тихими не-числами (Quiet NAN), они тоже могут иметь знак +QNaN и -QNaN.

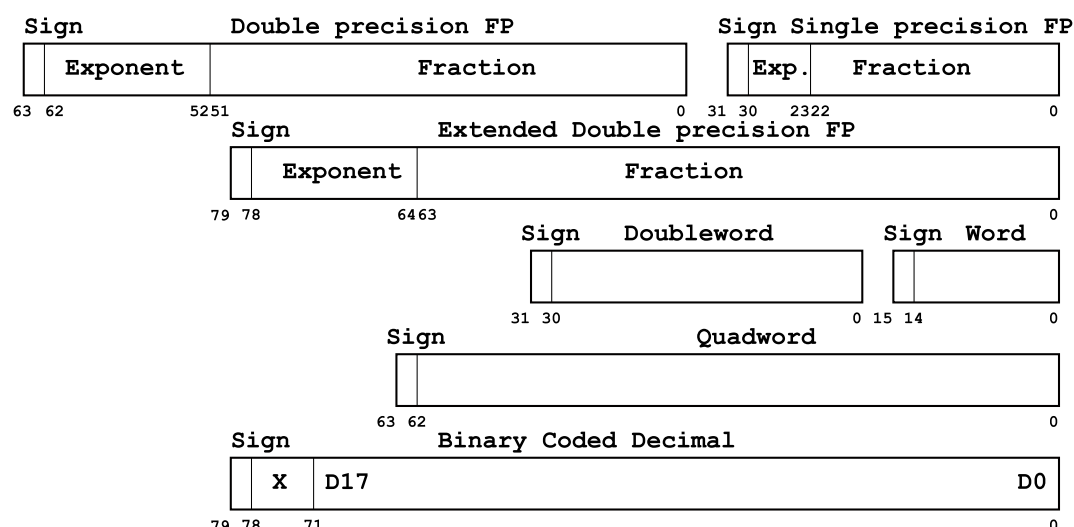


Рис. 3.16. Типы данных, обрабатываемые FPU

Многие вычислительные программы продолжают вычисления при возникновении $+\text{INF}$, $-\text{INF}$, $+\text{QNaN}$ и $-\text{QNaN}$. Это возможно если замаскировать некоторые исключения в регистре управления FPU.

FPU может выполнять операции и с целыми числами, они интерпретируются как знаковые и могут быть 16-, 32- и 64-разрядными.

Последним типом, который обрабатывает FPU, является 18-значные целые десятичные числа (BCD), но их можно только грузить и сохранять. Все типы, обрабатываемые FPU, показаны на рисунке 3.16.

При выполнении операций в математическом сопроцессоре могут возникать исключительные ситуации, которые могут маскироваться при установке соответствующего флага в регистре управления FPU. Эти исключительные ситуации следующие:

- I (Invalid Operation) – недействительные операции: чтение пустого регистра, ZERO/ZERO, INF/INF, операции с NaN, корень из отрицательного числа и т. д.;
- D (Denormalized Operand) – денормализованный операнд;
- Z (Zero Divide) – деление на нуль;
- O (Overflow) – переполнение;
- U (Underflow) – денормализация результата;
- P (Precision loss) – потеря точности.

Если исключение не замаскировано (сброшен флаг), то возникает немаскируемое исключение (ES) и аппаратный сигнал ошибки на выходе FPU. Некоторые инструкции сопроцессора имеют два варианта – ожидающий исключение и не ожидающий исключение. Ожидающие инструкции анализируют состояние флага ES. Имеется и отдельная инструкция ожидания исключения – WAIT/FWAIT.

Команды сопроцессора образуют пять групп:

- команды передачи;
- команды сравнения;
- арифметические команды;
- вычисление трансцендентных функций;
- команды управления.

Технология MMX и 3D Now!. Аббревиатура MMX обозначает MultiMedia eXtensions – расширения мультимедиа (графика, звук, коммуникации). Эти расширения вносят в процессор общего назначения элементы сигнального процессора, который предназначен для обработки потока информации в реальном масштабе времени. Особенность работы приложений мультимедиа – выполнение однотипных операций с максимальной производительностью. Одним из способов достижения максимальной производительности является выполнение инструкции сразу с группами операндов (Single Instruction Multiple Data – SIMD). Действие типичной инструкции MMX можно изобразить схемой, показанной на рисунке 3.17

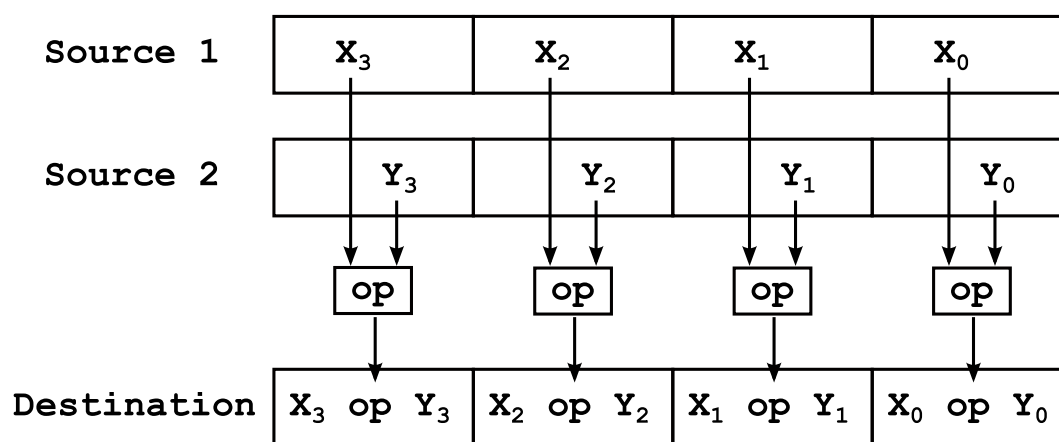


Рис. 3.17. Действие типовой SIMD инструкции.

В процессорах IA32 команды расширения MMX выполняются той же аппаратурой, что и команды плавающей арифметики. Иначе говоря, MMX – это режим работы FPU.

Команды MMX оперируют со специальными типами данных, которые называются упакованными. В 64-разрядных регистрах MMX могут храниться и обрабатываться

- восемь байтов (Packed byte),
- четыре двухбайтовых слова (Packed word),

- два четырехбайтовых слова (Packed doubleword),
- одно восьмибайтовое слово (Quadword).

Особенностью выполнения MMX-инструкций является насыщающаяся арифметика. Это обозначает, что, если результатом выполнения арифметической инструкции является число, не представимое в рамках беззнакового целого типа, результатом остаётся максимальное (111..11) или минимальное (000..00) представимое число. Такая арифметика, безусловно, порождает ошибки, но эти ошибки не так сильно влияют на результат в мультимедийных приложениях. Например, при воспроизведении звука насыщающаяся арифметика может породить срезанные верхушки в сигнале, но обычная арифметика порождает разрывы, что значительно хуже.

3.10. SSE

Аббревиатура SSE обозначает Streaming SIMD Extensions – расширение потоковых SIMD инструкций. Аббревиатура SIMD, в свою очередь, представляет собой термин из теории параллельной обработки данных – Single Instruction Multiple Data (одна инструкция – множество данных). Мы рассмотрим эти инструкции подробнее.

Инструкции SSE. Инструкции SSE распадаются на четыре группы:

- SIMD-инструкции, работающие с плавающими числами одинарной точности в XMM регистрах,
- инструкции управления состоянием,
- 64-битные SIMD-инструкции, работающие с целыми числами в MMX регистрах,
- инструкции управления кэшированием, предвыборки и упорядочения.

Первая группа инструкций сама распадается на несколько групп:

- инструкции перемещения данных,
- арифметика с упакованными данными,
- инструкции сравнения,
- логические инструкции,
- инструкции перемешивания и распаковки,
- инструкции преобразования.

Эта группа инструкций образует базовый блок (практически всё необходимое) для данных одинарной точности, обрабатываемых в регистрах XMM. Структура типовой инструкции этого блока похожа на типовую инструкцию MMX, разница лишь в том, что теперь 128-разрядный (16-байтовый) регистр содержит четыре плавающих числа одинарной точности (по 4 байта каждое). Такие числа в терминологии Intel называются *packed data* (мы будем переводить этот термин – упакованные данные), а по терминологии AMD – *vector data*, что, наверное, точнее. Кроме упакованных данных, инструкции SSE позволяют работать со скалярными данными в регистрах XMM. В случае скалярных инструкций операции производятся с одним четырёхбайтовым плавающим числом.

Инструкции управления состоянием позволяют сохранять и восстанавливать состояние регистра управления и состояния – MXCSR.

64-битные SIMD-инструкции, работающие с целыми числами в MMX регистрах. Эти инструкции расширяют блок инструкций MMX. Они позволяют вычислять среднее по упаковке, находить наибольшее и наименьшее число в упаковке, перемешивать слова в упаковке, вставлять и извлекать слово, вычислять сумму абсолютных разностей.

Инструкции управления кэшированием, инструкция предвыборки и упорядочения сохранения. Эти инструкции позволяют сбрасывать числа из упаковки в основную память мимо кэшей и загружать данные в кэш выбранного уровня (*prefetch n*).

Инструкции SSE2. Эти инструкции состоят из четырех групп:

- инструкции работы с упакованными числами двойной точности в регистрах XMM,
- инструкции преобразования чисел двойной точности регистрах XMM,
- 128-битные целочисленные инструкции,
- управление кэшированием и упорядочением инструкций.

Инструкции работы с упакованными числами двойной точности в регистрах XMM добавляют к базовому блоку SSE инструкций новый тип – упакованные плавающие числа двойной точности (два числа в упаковке). Как и в базовом блоке, эта группа инструкций распадается на те же подгруппы:

- инструкции перемещения данных,
- арифметика с упакованными данными,
- инструкции сравнения,

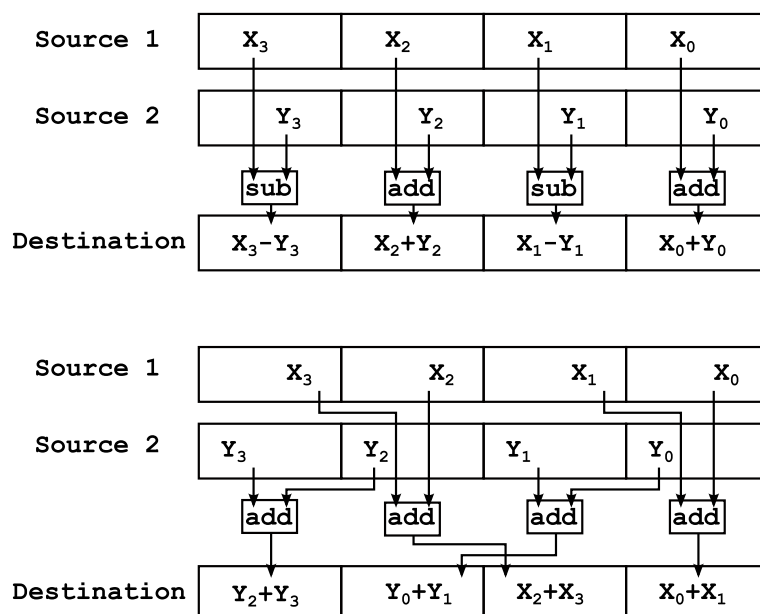


Рис. 3.18. Примеры асимметричных и горизонтальных инструкций SSE.

- логические инструкции,
- инструкции перемешивания и распаковки,
- инструкции преобразования.

Инструкции преобразования чисел двойной точности в регистрах XMM – группа инструкций, позволяющая преобразовывать данные для FPU в данные для SSE, и наоборот. Преобразования касаются как векторных, так и скалярных типов SSE.

128-битные целочисленные инструкции выполняют преобразования 32-разрядных целых (CPU и MMX) в плавающие XMM-данные одинарной точности, и наоборот, выполняют арифметические действия с беззнаковыми 32-разрядными целыми, перемешивают 16-разрядные целые, выполняют сдвиги 64-разрядных целых, распаковывают 32-разрядные целые.

Инструкции SSE3. Набор SSE3 примечателен появлением так называемых горизонтальных и асимметричных инструкций, примеры выполнения которых показаны на рисунке 3.18.

Инструкции SSE4.1. и SSE4.2. Инструкции SSE4.1. улучшили группу операций с упакованными целыми числами (dword), для плавающих упакованных типов добавили вычисление скалярных произведений, которые могут широко использоваться для реализации алгоритмов линейной алгебры и обработки изображений. Две инструкции реализуют вычисление CRC32 и вычисление бита четности второго операнда.

В одной инструкции реализованы стандартные алгоритмы контроля целостности данных.

Инструкции SSE4.2. Эта группа инструкций ввела обработку строк с использованием ресурсов векторного процессора.

Глава 4.

Диск

4.1. Физические принципы

Техника записи и считывания цифровой информации на магнитный носитель была разработана ещё в те времена, когда использовались магнитные ленты. Информация кодируется участками поверхности с чередующимися направлениями намагниченности. В традиционных дисках используется горизонтальная намагниченность (вдоль поверхности диска), в настоящее время используются также магнитные слои с вертикальным направлением намагниченности, что позволяет существенно увеличить плотность записи. Рисунок 4.1 даёт представление об устройстве головок для чтения/записи и ориентации намагниченности.

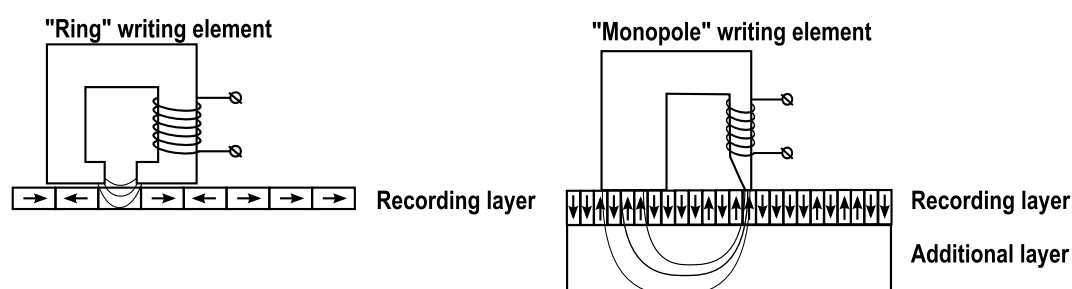


Рис. 4.1. Традиционный и вертикальный способы записи цифровой информации на магнитном носителе

Запись информации на поверхности CD, DVD и BlueRay осуществляется при помощи питов (pit – яма, ямка). Пит создаётся коротким импульсом мощного (100 – 300 мВт) инфракрасного лазера. Этот импульс воздействует на краситель или тонкий слой металла, изменяя его отражение (пит рассеивает свет, окружение отражает). При чтении свет сфокусированного маломощного (0.3 – 1 мВт) лазера модулируется проходящими через фокус питами. Представление о размере и форме

питов, а также об используемой оптике дает рисунок 4.2. Размер фокального пятна и, следовательно, плотность записи определяются длиной волны излучения и апертурой (углом сходимости пучка света). Размер пятна определяется формулой Аббе – $d = 4\lambda/\pi \sin\alpha$. При переходе $CD \Rightarrow DVD \Rightarrow BlueRay$ длина волны убывает ($780\text{нм} \Rightarrow 650\text{нм} \Rightarrow 405\text{нм}$), а апертурный угол растёт ($\sin\alpha = 0.45 \Rightarrow 0.6 \Rightarrow 0.85$), что и позволяет увеличить емкость диска от 0.7 ГБ до 25 ГБ. Малое фокальное пятно сложнее удерживать на отражающей поверхности вращающегося диска, что ужесточает требования к точности систем автофокусировки и удержания пятна на дорожке питов. На рисунке 4.2 показано устройство считывающей головки CD/DVD. Мощный лазер, предназначенный для стирания и записи информации, не показан.

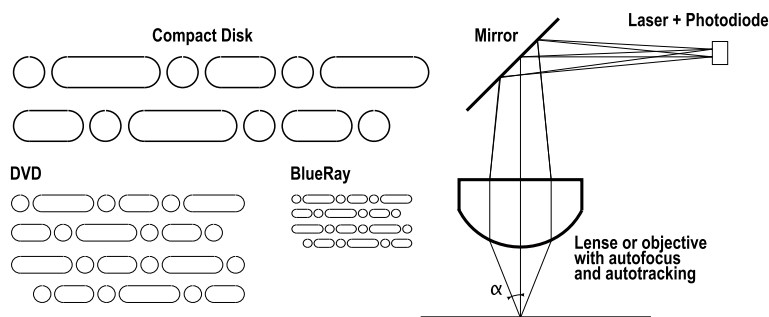


Рис. 4.2. Относительные размеры питов для CD, DVD и Blue Ray (Справа оптическая схема чтения)

На физическом уровне основным ресурсом магнитного носителя является число переключений магнитного потока на единицу длины дорожки (FCPI – flux change per inch). У оптических носителей основной показатель – размер пита.

4.2. Кодирование информации на диске

Предположим, что сигнал с магнитной головки или фотоприемника имеет следующую форму (см. рисунок 4.3).

Какой из двух кодов, показанных снизу, записан во фрагменте? Ответ появляется, только если указан шаг по времени (синхронизация). Этот шаг будет зависеть от скорости вращения диска, которая не может поддерживаться настолько точно, чтобы его можно было установить раз и навсегда. Но его можно указать в начале каждого сектора, передавая последовательность чередующихся нулей и единиц. Соответствующая частота запоминается генератором на приемном конце (Phase loop lock – PLL), и далее декодирование производится с учетом заданного временного дискрета. Единицы и нули кодируются не уровнями сигнала, а

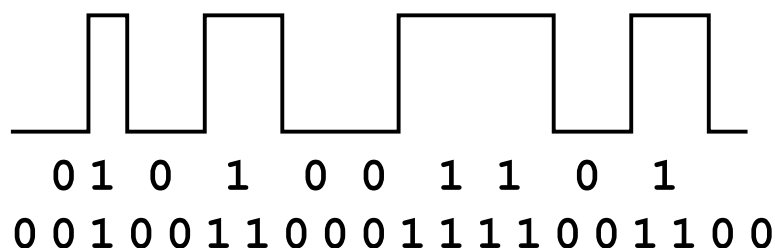


Рис. 4.3. Пример чередования намагниченности.

переходами. Простейший способ следующий: считать, что считан ноль, если за время такта не произошёл переход, и считать, что считана единица, если был переход (безразлично какой). При таком способе кодирования одинаково плохая ситуация возникает при слишком длинных и слишком коротких последовательностях одинаковых символов (нулей или единиц). Такая последовательность называется *run length* (пробег). Коды выбираются так, чтобы они были ограничены. Общее название таких кодов Run Length Limited – RLL. В каждом конкретном случае название кода имеет вид (m,n) RLL (m – минимальная длина run length, n – максимальная длина run length). Понятно, что такой код позволяет кодировать от m до n битов на один переход. Стандарты интерфейсов магнитных накопителей следуют за методами кодирования информации. Поэтому и названия интерфейсов магнитных накопителей практически повторяют названия методов кодирования информации.

- 1) MFM. Метод MFM (Modified Frequency Modulation) является разновидностью частотной модуляции, широко используемой в радиовещании и связи. Отличие заключается в том, что модифицированная модуляция позволяет обеспечить двукратное повышение плотности записи данных. За один переход (смену направления) намагниченности можно записать от одного до трех бит данных. Сигналы с головки передаются по кабелю данных в аналоговой форме; данные отделяются от сигналов синхронизации с помощью аналогового устройства – сепаратора, устанавливаемого на плате контроллера.
- 2) RLL. Run Length Limited. Другой способ модуляции (2,7 RLL или просто RLL), предложенный компанией IBM в 1986 году, использует перекодирование исходной информации с введением избыточности. Избыточность позволяет замечать и исправлять ошибки. Метод RLL преобразует данные в шестнадцатитривиновые слова, позволяющие записывать за один переход состояния намагниченности диска от 2 до 7 бит (эти цифры и включены в название метода).

- 3) ESDI По мере роста скорости работы компьютеров интерфейс RLL перестал удовлетворять всем требованиям, и в 1985 году был разработан новый стандарт ESDI, который, по сути, являлся простым расширением возможностей своего предшественника.
- 4) SCSI Первоначальный вариант интерфейса SCSI (Small Computer System Interface) был предложен в конце 1970-х годов Shugart Associates (старое название Seagate) взамен разработанной компанией IBM системной шины IPI (интеллектуальный периферийный интерфейс). После неудачи в конкурентной борьбе с фирмой IBM этот интерфейс был предложен комитету ANSI X3T9.2 как интерфейс нижнего уровня под названием SCSI. В 1984 году этот комитет закончил разработку спецификации SCSI-1, и в 1986 году она была опубликована в окончательном виде. Этот интерфейс обеспечивал подключение широкого класса периферийных устройств, таких как винчестеры, принтеры, сканеры, стримеры, приводы CD-ROM и др. Интерфейс SCSI получился достаточно дорогим, но зато позволял подключать к магистрали до восьми устройств. В значительно модифицированном виде (Fast-SCSI и Wide-SCSI) SCSI используется до сих пор в серверах, для которых емкость накопителей критична (а цена – нет).
- 5) IDE/ATA Спецификация IDE/ATA была предложена в качестве недорогой альтернативы интерфейсам ESDI и SCSI для персональных компьютеров семейств IBM PC XT/AT. В результате сотрудничества компании Western Digital с Compaq Computer Corporation был разработан интерфейс IDE (Integrated Drive Electronics), называемый также ATA (AT attachment). Первые промышленные устройства на базе IDE/ATA были выпущены в 1986 году. Интерфейс был стандартизован (ANSI X3T9.2/90143) в 1990 году, как ATA (AT Attachment). Именно ATA-диски стали наиболее распространенными в настольных компьютерах. В настоящее время этот интерфейс называют PATA (parallel ATA), чтобы отличать его от SATA (serial ATA)
- 6) SATA. С дисками такого типа ситуация не устоялась. В настоящее время SATA – это сериализатор (на диске) и десериализатор (на материнской плате). (Сериализатор – преобразователь параллельного кода в последовательный, десериализатор – наоборот.) Но со временем последовательный порт SATA станет частью чипсета на материнской плате и контроллер на диске будет иметь порт SATA.

- 7) USB. USB дисков не бывает. Там внутри стоит устройство – ATA-USB bridge, которое просто перекодирует команды ATA интерфейса и данные в последовательный код магистрали USB. Это устройство оказалось дешевым, и его проще «приделать» к готовому ATA-диску.
- 8) RAID (Redundund Array of Inexpensive Drives – избыточный массив дешевых дисков) – это ATA или SCSI устройства.

4.3. Интерфейсы ATA и SATA

Краткий обзор интерфейсов магнитного накопителя показывает, что смелый программист, которому зачем-то понадобилось обойти драйверы Windows или Linux, функции Int 21 MSDOS, всю файловую систему или даже функции Int 13 BIOS упрется в интерфейс ATA. На этом уровне программист увидит регистры контроллера жесткого диска (пространство ввода/вывода).

Аппаратура современной материнской платы (чипсет) позволяет обращаться к нескольким магистралям ATA (до четырех). На каждой магистрали могут находиться два ATA устройства. Каждая магистраль имеет группу байтовых портов (десять байтов), адреса которых начинаются с базового адреса. В таблице 4.1 показаны адреса четырех магистралей ATA.

№магистрали	Рег. команды	Рег. управления	№прерывания
1	1F0-1F7	3F6-3F7	14
2	170-177	376-377	15
3	1E8-1EF	3EE-3EF	11
4	168-16F	36E-36F	10

Таблица 4.1. Адреса регистров контроллера ATA

Магистрали 1 и 2 есть всегда, магистрали 3 и 4 некоторые контроллеры (чипсет попроще) не предоставляют.

В таблице 4.2 указано назначение регистров команды и управления для канала 1 (для остальных каналов их нетрудно получить при помощи сдвига).

DR – регистр данных доступен только при PIO, в DMA его не прочитывать. Регистр на самом деле 16-разрядный старшие разряды читаются в ER. Регистр ошибок ER читают если, после команды возник бит ошибки в регистре состояния или после команды «диагностика». Биты регистра ER имеют следующее значение:

Адрес	При чтении	При записи
1F0	Регистр данных (DR)	Регистр данных (DR)
1F1	Регистр ошибок (ER)	Рег. свойств (FR)
1F2	Счетчик секторов (SC)	
1F3	Сектор / LBA 0-7	
1F4	Цилиндр (CL)/LBA 8-15	
1F5	Цилиндр (CH)/LBA 16-23	
1F6	Устр. и головка / Устр. и LBA 24-27	
1F7	Регистр состояния (SR)	Рег. команд (CR)
3F6	Альтерн. регистр состояния (AC)	Рег. управл. (DC)
3F7	Не используется	Не используется

Таблица 4.2. Назначение регистров контроллера АТА

- Bit 0 – не найден адресный маркер сектора
- Bit 1 – не найдена нулевая дорожка при рекалибровке
- Bit 2 – аварийное прекращение выполнения команды
- Bit 3 – получен запрос на смену носителя информации
- Bit 4 – сектор (CHS) не найден
- Bit 5 – произведена смена носителя информации
- Bit 6 – некорректируемая ошибка данных
- Bit 7 – зарезервировано.

Отдельные коды читаются целиком (без битовой раскладки):

- 01 – нет ошибок (0 – исправно, 1 – исправно или отсутствует)
- 00, 02-7F – ошибка (0 – неисправно, 1 – исправно или отсутствует)
- 81 – 0 – исправно, 1 – неисправно
- 80, 82-FF – оба устройства неисправны.

В регистр свойств (FR) пишутся дополнительные коды при подаче некоторых служебных команд. По размерам регистров сектора, цилиндра и устройства и головки понятно, что, по мнению АТА интерфейса, у диска не бывает больше

- CHS – 256 сект. \times 65536 цил. \times 128 гол. \times 512 байт = 136.9 ГБ
- LBA – $2^{31} \times 512$ байт = 1024 ГБ.

Современные терабайтные диски используют сектора большего размера (4КБ – 2010 год).

Все команды передаются в регистр команд. Всего может быть 256 команд, которые делятся на

- обязательные (mandatory),
- дополнительные (optional),
- команды изготовителя (vendor specific).

Обязательных команд всего 18. Дополнительные команды и команды изготовителя – основа диагностического и ремонтного программного обеспечения. Ниже приводится список обязательных команд интерфейса ATA (таблица 4.3).

№	Команда	Код
1	Execute device diagnostic	90
2	Identify device	EC
3	Initialize device parameters	91
4	Read DMA (with retry)	C8
5	Read DMA (without retry)	C9
6	Read multiple	C4
7	Read sector(s) (with retry)	20
8	Read sector(s) (without retry)	21
9	Read verify sector(s) (with retry)	40
10	Read verify sector(s) (without retry)	41
11	Seek	70
12	Set features	EF
13	Set multiple mode	C6
14	Write DMA (with retry)	CA
15	Write DMA (without retry)	CB
16	Write multiple	C5
17	Write sector(s) (with retry)	30
18	Write sector(s) (without retry)	31

Таблица 4.3. Обязательные команды интерфейса ATA

Список литературы

1. Потемкин, И. С. Функциональные узлы цифровой автоматики. М.: Энергоатомиздат, 1988.
2. Mano, M. M., Kime C. Logic and Computer Design Fundamentals. 4th edition. - Prentice Hall Press Upper Saddle River. - NJ, USA, 2007.
3. Гук М. Аппаратные средства IBM PC. Энциклопедия, 2-е изд. – СПб.: Питер, 2003.
4. URL:<http://www.intel.com/content/www/us/en/processors>. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1, 2A, 2B, 2C, 3A, 3B, and 3C.
5. URL:<http://www.intel.com/content/www/us/en/processors>. Intel® 64 and IA-32 Architectures Optimization Reference Manual
6. URL:<http://www.intel.com/content/www/us/en/processors>. Intel® SSE4 Programming Reference
7. URL:http://support.amd.com/us/Processor_TechDocs/APM_v1_-24592.pdf. AMD64 Architecture Programmer's Manual: Application Programming.
8. URL:http://support.amd.com/us/Processor_TechDocs/APM_v1_-24593.pdf. AMD64 Architecture Programmer's Manual: System Programming.
9. URL:http://support.amd.com/us/Processor_TechDocs/APM_v1_-24594.pdf. AMD64 Architecture Programmer's Manual: General-Purpose and System Instructions.
10. URL:http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html. D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. March, 1991 issue of Computing Surveys.

-
11. URL:http://www.techfest.com/hardware/bus/isa_sokos.htm The ISA And Pc/104 Bus by Mark Sokos.
 12. URL:http://www.techfest.com/hardware/bus/eisa_sokos.htm The EISA Bus by Mark Sokos.
 13. URL:<http://people.na.infn.it/garufi/didattica/CorsoAcq/PCI.Local.Bus.Specification.Revision.3.0.pdf>. PCI Local Bus Specification Revision 3.0 August 12, 2002.
 14. URL:<http://rutracker.org/forum/viewtopic.php?t=3684825> PCI Express® Base Specification. Revision 3.0 Version 0.9 August 10, 2010.

Учебное издание

Лоханин Михаил Владимирович

**АРХИТЕКТУРА СОВРЕМЕННОГО
КОМПЬЮТЕРА**

Учебное пособие

Редактор, корректор М. В. Никулина
Компьютерный набор и верстка М. В. Лоханин

Подписано в печать 15.12.2011. Формат 60×84/16.
Бумага тип. Усл. печ. л. 6,04. Уч.-изд. л. 5,0
Тираж 50 экз. Заказ

Оригинал-макет подготовлен
в редакционно-издательском отделе
Ярославского государственного университета
им. П. Г. Демидова.

Отпечатано на ризографе.
Ярославский государственный университет
им. П. Г. Демидова
150000, Ярославль, ул. Советская, 14.